Universidade de Aveiro Departamento de Engenharia Mecânica
2013

Miguel Armando Riem de Oliveira

Sistemas Automáticos de Informação e Segurança para Apoio na Condução de Veículos

Automatic Information and Safety Systems for Driving Assistance

Universidade de Aveiro Departamento de Engenharia Mecânica
2013

Miguel Armando Riem de Oliveira

Sistemas Automáticos de Informação e Segurança para Apoio na Condução de Veículos

Automatic Information and Safety Systems for Driving Assistance

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Mecânica, realizada sob orientação científica de Vitor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro, e de Angel Domingo Sappa, Investigador Senior do Centro de Visão por Computador da Universidade Autónoma de Barcelona.

O júri / The jury

Presidente / President	**Prof. Doutora Maria Celeste da Silva do Carmo** Professora Catedrática da Universidade de Aveiro
Vogais / Committee	**Prof. Doutor Miguel Ángel Sotelo** Professor Catedrático da Universidade de Alcalá - Espanha
	Prof. Doutor Francisco António Cardoso Vaz Professor Catedrático Aposentado da Universidade de Aveiro
	Prof. Doutor Angel Domingo Sappa Investigador Senior do Centro de Visão por Computador - Espanha (Coorientador)
	Prof. Doutor António Paulo Gomes Mendes Moreira Professor Associado da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor Vitor Manuel Ferreira dos Santos Professor Associado da Universidade de Aveiro (Orientador)

Agradecimentos /
Acknowledgements

Quero deixar um agradecimento ao Professor Vitor Santos, pelo seu apoio constante ao longo destes anos. A inteligência, o entusiasmo e a dedicação que coloca no seu trabalho são exemplos contagiantes. O Professor Angel Sappa acolheu-me em Barcelona em 2010. Desde então mantivemos contacto semanal, tendo a sua ajuda sido muito valiosa. Das nossas conversas destaco a sua inteligência, paciência e ponderação. A ele quero também deixar um agradecimento especial. Não posso deixar de mencionar também o Professor Paulo Dias, que sempre se mostrou disponível para conversar e discutir comigo os mais variados detalhes do meu trabalho. A ele um muito obrigado. Ao longo destes anos, trabalhei em estreita colaboração com muitas pessoas no Laboratório de Automoção e Robótica da Universidade de Aveiro. Não posso deixar de mencionar especialmente o Jorge Almeida e o David Gameiro, mas também o Procópio Stein, Ricardo Pascoal, Remi Sabino, Miguel Almeida e ainda todas as pessoas que me acompanharam no projeto *Atlas* e no Departamento de Engenharia Mecânica da Universidade de Aveiro.
Não é possível conduzir a bom termo um desafio desta magnitude sem o apoio constante e incondicional de uma família sempre presente: Laura, Vítor, João e Celeste.
À Ema, sempre comigo.
Finalmente, quero deixar um agradecimento à Fundação para a Ciência e Tecnologia, Ministério da Educação e Ciência, que financiou este trabalho com a bolsa de referência SFRH/BD/43203/2008.

Palavras-chave Condução Autónoma, Sistemas Avançados de Apoio à Condução, Robótica, Correção de Cor, Reconstrução 3D, Veículos Autónomos.

Resumo

O objeto principal desta tese é o estudo de algoritmos de processamento e representação automáticos de dados, em particular de informação obtida por sensores montados a bordo de veículos (2D e 3D), com aplicação em contexto de sistemas de apoio à condução. O trabalho foca alguns dos problemas que, quer os sistemas de condução automática (AD), quer os sistemas avançados de apoio à condução (ADAS), enfrentam hoje em dia. O documento é composto por duas partes. A primeira descreve o projeto, construção e desenvolvimento de três protótipos robóticos, incluindo pormenores associados aos sensores montados a bordo dos robôs, algoritmos e arquitecturas de software. Estes robôs foram utilizados como plataformas de ensaios para testar e validar as técnicas propostas. Para além disso, participaram em várias competições de condução autónoma tendo obtido muito bons resultados. A segunda parte deste documento apresenta vários algoritmos empregues na geração de representações intermédias de dados sensoriais. Estes podem ser utilizados para melhorar técnicas já existentes de reconhecimento de padrões, deteção ou navegação, e por este meio contribuir para futuras aplicações no âmbito dos AD ou ADAS. Dado que os veículos autónomos contêm uma grande quantidade de sensores de diferentes naturezas, representações intermédias são particularmente adequadas, pois podem lidar com problemas relacionados com as diversas naturezas dos dados (2D, 3D, fotométrica, etc.), com o carácter assíncrono dos dados (multiplos sensores a enviar dados a diferentes frequências), ou com o alinhamento dos dados (problemas de calibração, diferentes sensores a disponibilizar diferentes medições para um mesmo objeto). Neste âmbito, são propostas novas técnicas para a computação de uma representação multi-câmara multi-modal de transformação de perspectiva inversa, para a execução de correcção de côr entre imagens de forma a obter mosaicos de qualidade, ou para a geração de uma representação de cena baseada em primitivas poligonais, capaz de lidar com grandes quantidades de dados 3D e 2D, tendo inclusivamente a capacidade de refinar a representação à medida que novos dados sensoriais são recebidos.

Keywords Autonomous Driving, Advanced Driver Assistance Systems, Robotics, Color Correction, 3D Reconstruction, Autonomous Vehicles.

Abstract

The main object of this thesis is the study of algorithms for automatic information processing and representation, in particular information provided by onboard sensors (2D and 3D), to be used in the context of driving assistance. The work focuses on some of the problems facing todays Autonomous Driving (AD) systems and Advanced Drivers Assistance Systems (ADAS). The document is composed of two parts. The first part describes the design, construction and development of three robotic prototypes, including remarks about onboard sensors, algorithms and software architectures. These robots were used as test beds for testing and validating the developed techniques; additionally, they have participated in several autonomous driving competitions with very good results. The second part of this document presents several algorithms for generating intermediate representations of the raw sensor data. They can be used to enhance existing pattern recognition, detection or navigation techniques, and may thus benefit future AD or ADAS applications. Since vehicles often contain a large amount of sensors of different natures, intermediate representations are particularly advantageous; they can be used for tackling problems related with the diverse nature of the data (2D, 3D, photometric, etc.), with the asynchrony of the data (multiple sensors streaming data at different frequencies), or with the alignment of the data (calibration issues, different sensors providing different measurements of the same object). Within this scope, novel techniques are proposed for computing a multi-camera multi-modal inverse perspective mapping representation, executing color correction between images for obtaining quality mosaics, or to produce a scene representation based on polygonal primitives that can cope with very large amounts of 3D and 2D data, including the ability of refining the representation as new information is continuously received.

Contents

Acronyms **xxxv**

1 Introduction **1**

1.1 Motivation . . . 2

1.2 Objectives . . . 6

1.3 Thesis Structure . . . 7

2 State of the Art **9**

2.1 Milestones Achieved in Autonomous Driving . . . 9

2.2 Research Groups in Autonomous Driving . . . 11

2.3 Autonomous Driving Competitions . . . 20

2.4 The Near Future of Autonomous Driving . . . 25

2.5 Conclusions . . . 28

3 Robot Prototypes **29**

3.1 Introduction . . . 29

3.2 *Atlas2000* . . . 29

3.2.1 Mechanical Components . . . 31

3.2.2 Electronic Components . . . 32

3.2.3 Sensors . . . 32

3.3 *AtlasMV* . . . 34

3.3.1 Mechanical Components . . . 35

3.3.2 Electronic Components . . . 38

3.3.3 Sensors . . . 40

3.4 *AtlasCar* . . . 41

3.4.1 Mechanical and Electronic Components . . . 43

3.4.2 Sensors . . . 44

3.5 Algorithms . . . 48

3.5.1 Sensor Fusion . . . 48

3.5.2 Road Detection 52
3.5.3 Road Maintenance Area 56
3.5.4 Path Planning 59
3.6 Results 74
3.7 Conclusions 75

4 Software Architectures **77**
4.1 Introduction 77
4.2 Related Work 78
4.2.1 Lightweight Communications and Marshalling 78
4.2.2 Carnegie Mellon Robot Navigation Toolkit 79
4.2.3 Robot Operating System 79
4.2.4 Others 86
4.3 Laboratory of Automation and Robotics Toolkit 87
4.3.1 Information Exchange 88
4.3.2 Data Logging and Playback 92
4.4 The LARtk on the *AtlasMV* 93
4.5 The ROS on the *AtlasCar* 97
4.5.1 Coordinate Frames 97
4.5.2 Vehicle Interface and Control 100
4.5.3 Cameras 102
4.5.4 Stereo Camera 103
4.5.5 Pan and Tilt Unit 103
4.5.6 Planar Laser Range Finders 106
4.5.7 3D Laser Range Finder 108
4.5.8 Multi Target Tracking 110
4.6 Conclusions 113

5 Inverse Perspective Mapping **115**
5.1 Introduction 115
5.2 Related Work 116
5.3 Classical Inverse Perspective Mapping 121
5.4 Road and Vehicle Coordinate frames 123
5.5 Proposed Approach 124
5.5.1 Half Space of Projection 124
5.5.2 Desired Area of Perception 126
5.5.3 Image boundaries 127
5.5.4 Laser Generated Polygon 128

5.5.5 Image Projection Polygon 129
5.6 Results 130
5.6.1 Test Platforms 131
5.6.2 Computational Performance 132
5.6.3 IPM Accuracy 134
5.6.4 IPM Accuracy using Laser Range Finder 136
5.6.5 Tests in Real Environments 137
5.7 Conclusions 138

6 Photometric Calibration **141**
6.1 Introduction 141
6.2 Related Work 142
6.3 Mean Shift Based Local Color Correction 146
6.3.1 Image Segmentation 147
6.3.2 Local Color Transfer Functions 148
6.3.3 Color Influence Maps 150
6.3.4 Weighted Color Correction 151
6.3.5 Results 151
6.4 3D Gaussian Mixture Models 156
6.4.1 Probabilistic Segmentation 156
6.4.2 Color Transfer Functions 159
6.4.3 Color Correction 160
6.4.4 Results 161
6.5 Probabilistic Color Correction using sets of Truncated Gaussians 168
6.5.1 Mean Shift Color Segmentation 168
6.5.2 Local Joint Image Histogram 169
6.5.3 Modeling the Mutual Information 170
6.5.4 Color Palette Mapping Functions 174
6.5.5 Expanding the Color Palette Mapping Functions 175
6.5.6 A Case Study 176
6.5.7 Comparison with other Algorithms 179
6.5.8 Analysis of the effect of Mean shift to the overall Color Correction 188
6.6 Conclusions 189

7 Data Sets and Preprocessing **191**
7.1 Introduction 191
7.2 Related Work 191
7.3 Data Sets 192

7.4 Preprocessing of 3D point clouds 200
7.4.1 Local and Spatial point filters 200
7.4.2 Surface Normals Estimation 204
7.4.3 Statistical Outlier Removal 207
7.5 Conclusions 210

8 Geometric Scene Reconstruction **213**
8.1 Introduction 213
8.2 Related Work 214
8.3 Proposed Approach 215
8.3.1 Representation 215
8.3.2 Support Plane 217
8.3.3 From RANSAC Inliers to Primitive Support Points 221
8.3.4 Plane Model Estimation 223
8.3.5 Bounding Polygon 224
8.3.6 Basic Geometric Properties 226
8.3.7 A Cascade Processing Configuration 230
8.4 Results 231
8.4.1 Qualitative Analysis 231
8.4.2 Data Compression 232
8.4.3 Computation Time 234
8.4.4 Comparison to Other Approaches 236
8.5 Conclusions 250

9 Geometric Scene Refinement **251**
9.1 Introduction 251
9.2 Related Work 253
9.3 Proposed Approach 258
9.3.1 Incremental Bounding Polygon Computation 264
9.3.2 Incremental Support Plane Computation 266
9.3.3 A Cascade Processing Configuration for the Expansion 267
9.4 Results 268
9.4.1 Qualitative Analysis 269
9.4.2 Computation Time 269
9.4.3 Comparison of Approaches With and Without Expansion 272
9.4.4 Evolution of Primitives 276
9.5 Conclusions 281

10 Photometric Scene Reconstruction **283**
10.1 Introduction 283
10.2 Related Work 284
10.3 Proposed Approach 288
10.3.1 Data Dependent Triangulation 288
10.3.2 Image Bounding Polygon 298
10.3.3 Projection Polygon 301
10.3.4 Mapping Images to Geometric Polygonal Primitives 307
10.3.5 Handling the Convexity of Delaunay Meshes 311
10.4 Results 315
10.4.1 Two Cameras Overlapping Projections 315
10.4.2 Two Cameras Non Overlapping Projections 317
10.4.3 Single Camera Multiple Primitives 318
10.4.4 Qualitative Analysis 320
10.5 Conclusions 321

11 Photometric Scene Refinement **323**
11.1 Introduction 323
11.2 Related Work 325
11.3 Proposed Approach 331
11.3.1 Mapping Multiple Projections 335
11.3.2 Sequential Updating of Complex Meshes 344
11.3.3 Projection Quality Functions 349
11.4 Results 354
11.4.1 Texture Improvement Over Multiple Projections 354
11.4.2 Evolution of the Global Primitive Mesh 358
11.4.3 Full Scene Geometric and Photometric Reconstruction 365
11.5 Conclusions 370

12 Conclusions **373**
12.1 Publications 376
12.2 Future Work 378

A **407**

B **411**

C **417**

D **421**

List of Tables

1.1 Source [World Health Organization 2004]. Change in rank order of DALYs for the ten leading causes of the global burden of disease. 3

2.1 Results of the Defense Advanced Research Projects Agency (DARPA) Grand Challenge, 2005 edition [Wikipedia 2004]. The Table only shows the teams that completed the challenge. 21

2.2 Results of the DARPA Urban Challenge, 2007 [Wikipedia 2004]. The Table only shows the teams that completed the challenge. 21

3.1 Path planning parameters values used on the AtlasMV. 70

3.2 Robots positioned in the first three places, Autonomous Driving Competitions (ADCs) since 2001. Robots signaled in blue were developed by the Laboratory of Automation and Robotics, University of Aveiro. 74

3.3 Results of the participations of the *Atlas2000* in the ADCs. 75

3.4 Results of the participations of the *AtlasMV* in the ADCs. 75

6.1 A list of the algorithms used for comparing the proposed approach. 162

6.2 Average and standard deviation of CS and SS scores for the two data sets. Average processing time per image is also provided. The proposed approach (# 10) obtains the highest average CS score. 163

6.3 The output of the comparative methods and the proposed approach (alg. #10) for three images in the data set. The image pairs are shown on the top of the table. Below each image the CS and SS scores are displayed. For image *Synthetic* #34, algorithm #9 achieves the highest CS score. In the case of *Synthetic* #23 and *Real* #6, the proposed approach outperforms all other methods. 164

6.4 Average and standard deviation of CS and SS scores for the set of selected images. It also shows the average processing time per image. The methods are sorted by average CS score. Note that the fastest algorithm [Brown & Lowe 2007] gets the worst CS score. The proposed algorithm obtains the highest average CS score, the second best on average SS score and is the fastest processing of the top three in CS scores. 165

6.5 The output of the comparative methods and the proposed approach (**GMM**) for three of the images in the data set. The image pairs are show on the top of the table. Bellow each image the algorithm reference, CS and SS scores are displayed. In the first two images the proposed approach obtains the best CS score, while in the third image it scores close to the best. 167

6.6 Algorithms tested in this evaluation. The algorightm presented in this paper, will be referred to as #11. 180

6.7 Corrected overlapping areas of image pair numbers 6, 35 and 37 of the synthetic data-set. The **PSNR** (metric 1, *Mt 1*) and **S-CIELAB** (metric 2, *Mt 2*) scores are displayed. Best scores are highlighted in blue. 181

6.8 Corrected overlapping areas of image pair numbers 1, 12 and 21 of the real data-set. The **PSNR** (metric 1, *Mt 1*) and **S-CIELAB** (metric 2, *Mt 2*) scores are displayed. Best scores are highlighted in blue. 182

6.9 Mean and standard deviations of the **PSNR** scores for each algorithm in both data sets. Best results are highlighted in blue. 185

6.10 Mean and standard deviations of the **S-CIELAB** scores for each algorithm in both data sets. Best results are highlighted in blue. 185

6.11 Algorithms sorted by their average score. #10 and #11 were not evaluated (*NE*) in [Xu & Mulligan 2010], since they are posterior. 186

6.12 Evaluation of the performance of the Algorithms according to the **PSNR** score. Rows represent: the % of images where the algorithm achieved the best (1^{st}), second best (2^{nd}), third best (3^{rd}), or worse ($> 3^{rd}$) score. The last row shows the % of times an algorithm failed to color correct the image. Best results are highlighted in blue. . . . 187

6.13 Evaluation of the performance of the Algorithms according to the **S-CIELAB** score. Rows represent: the % of images where the algorithm achieved the best (1^{st}), second best (2^{nd}), third best (3^{rd}), or worse ($> 3^{rd}$) score. The last row shows the % of times an algorithm failed to color correct the image. Best results are highlighted in blue. . . 187

7.1 3D point cloud data sets available. Use soft copy for access to the hyper links. . . . 192

7.2 Several information on each of the logs. 193

7.3 Event channels on the Massachusetts Institute of Technology (MIT) logs and whether they are converted to Robot Operating System (ROS) bag files or not. 194

7.4 Comparison of the size of the 3D point clouds provided by the *AtlasCar* and the *Talos* vehicles. 195

7.5 Information on each of the locations in this sequence. Columns description: pt, number of points; size, memory size in mega bytes; t, mission time in seconds; d, traveled distance in meters. 196

7.6 Information on each of the locations in this sequence. Columns description: pt, number of points after voxel grid filtering; size, memory size in mega bytes; ratio, compression ratio obtained by the voxel grid filter; t, mission time in seconds; d, traveled distance in meters. 199
7.7 Size of the point clouds of sequence 1 after applying the preprocessing algorithm. For comparison with raw values see Table 7.5. 205
7.8 Size of the point clouds of sequence 2 after applying the preprocessing algorithm. For comparison with raw values see Table 7.6. 205

8.1 Comparison of the computation time of several approaches for surface reconstruction on the MIT data sets. 240
8.2 Comparison of the accuracy of the several approaches using Ball Pivoting Algorithm (BPA) results as ground truth and Hausdorff distance as metric. 243
8.3 Comparison of the Hausdorff distance accuracy of the Geometric Polygonal Primitives (GPP) 2 approach using: the standard approach, convex hull and ground plane included (also in Table 8.2); the convex hull with no ground plane included; the concave hull with ground plane; and the concave hull without ground plane. 249

9.1 Comparison of the computation time of several approaches for the reconstruction of the whole scenarios of the MIT data sets, when an approach of reconstructing each point cloud and then merging reconstructed meshes is used. 255

List of Figures

1.1 Source [European Road Safety Observatory 2011]. Fatalities, accidents and injured per year in the European Union (EU) countries. 4

1.2 Source [European Road Safety Observatory 2011]. Share of fatalities as a function of: (*a*) type of road; (*b*) weather conditions; (*c*) type of vehicle. Data for EU countries, 2009. 5

2.1 Four prototypes from Carnegie Mellon University: (*a*) Navlab I; (*b*) Sandstorm; (*c*) Boss; and (*d*) Highlander. 12

2.2 Two prototypes from the VaMoRs project: (*a*) VaMoRs; (*b*) VaMP. 14

2.3 Four prototypes from the VisLab group: (*a*) ARGO; (*b*) TerraMax; (*a*) BRAiVE; and (*d*) one of the four vehicles used on the VisLab Intercontinental Autonomous Challenge (VIAC) challenge. 16

2.4 Three prototypes from the Stanford University: (*a*) Stanley; (*b*) Junior; and (*c*) Google Driverless Car. 18

2.5 A satellite view of the Darpa Urban Challenge competition area, the George Air Force Base in Victorville, California. Pictures of some locations in the scenario are also displayed. The image on the left shows the race start and finish site. 22

2.6 Schematics of the ADC: (*a*) the full scenario; (*b*) traffic lights; (*c*) road maintenance and obstacle; (*d*) cross walk and parking zone; (*e*) traffic signs. 23

2.7 Pictures of the ADC: (*a*) the scenario at Portuguese Robotics Open (PRO) 2006; (*b*) traffic lights; (*c*) tunnel; (*d*) obstacles and traffic signs; (*e*) road maintenance area. . . 24

3.1 The *Atlas2000* autonomous robot. The robot participated in the PRO of 2005 (*a*), 2006 (*b*), 2007 (*c*), 2008 (*d*), 2009 (*e*), 2010 (*f*), 2011 (*g*). 30

3.2 An overview of *Atlas2000*: (*a*) The basic structural platform; (*b*) traction system; (*c*) steering system. 31

3.3 The braking mechanism of the *Atlas2000*. 32

3.4 The electronic components of the *Atlas2000*. 33

3.5 The sensors onboard the *Atlas2000*: (*a*) 2005; (*b*) 2011. 33

3.6 The *AtlasMV* autonomous robot. The robot participated in the PRO of 2008 (*a*), 2009 (*b*), 2010 (*c*) and 2011 (*d*). 34
3.7 An overview of the custom made parts of the *AtlasMV* robot. 35
3.8 An overview of *AtlasMV*: (*a*) bottom layer. (*b*) upper layer. 36
3.9 The mechanical components of the *AtlasMV*: (*a*) steering mechanism; (*b*) traction mechanism: (*c*) both systems mounted on the platform. 36
3.10 The compressed air braking system of the *AtlasMV*: (*a*) pistons and supporting frame; (*b*) structure already mounted; (*c*) the compressed air reservoir; (*d*) the compressed air circuit (blue tubes). 37
3.11 The electronic boards of the *AtlasMV*: (*a*) main board; (*b*) main board with other boards mounted; (*c*) distance sensors board; (*d*); power distribution board; (*e*) motor control board; (*f*) lights control board. 39
3.12 The sensors onboard the *AtlasMV*: (*a*) motor controlled camera support; (*b*) laser range finder; (*c*) early version, 2008; (*d*) recent version, 2011. 40
3.13 The *AtlasCar* autonomous vehicle. 42
3.14 (*a*) the engine compartment before the installation of the secondary alternator; (*b*) the engine compartment with the new alternator installed, visible in the bottom left part of the image; (*c*) a CAD drawing of the alternators in their new configuration. Source [Santos *et al.* 2010]. 43
3.15 (*a*) The power stabilization equipment onboard the *AtlasCar*: from left to right, the inverter, the UPS, the buffer battery; (*b*) The final installation of the *AtlasCar* trunk. Inside the white boxes are the AC-DC power converters and the PLC. 44
3.16 A schematic of the power supply lines of the *AtlasCar*. Source [Santos *et al.* 2010]. . 45
3.17 The *AtlasCar* full scale robotic platform. It is equipped with an active perception unit (*A*), a stereo rig (*B*), four LRF (*C, D, H*), a thermal vision camera (*F*), GPS (*G*) and an inertial measurement unit (*E*). 45
3.18 Coverage of the lasers mounted onboard the *AtlasCar*. The scan plane of each laser is depicted in different colors: (blue) right bumper; (green) left bumper; (magenta) center top roof; (red) rotating laser mounted on the roof. 46
3.19 Coverage of the 3D laser scanner mounted on the roof. Several possible positions are displayed. 46
3.20 Images from the Point Grey Research Bumblebee XB3 Stereo Camera (XB3) camera. These three images are used for two stereo pairs. 47
3.21 Images from the peripheral and foveated cameras 47
3.22 Non metric fusion of images: (*a*) manual calibration application for configuring the non metric fusion; (*b*) a mosaic obtained by fusion of two images. Source [Oliveira & Santos 2007] 49

3.23 Metric fusion of images using Inverse Perspective Mapping (IPM): (*a*) image from the left camera; (*b*) image from the right camera; (*c*) bird's eye view of the road. . . 50
3.24 The Denavit Hartenberg kinematics chain for one of the cameras of the *AtlasMV*: (*a*) the cameras support structure; (*b*) a detail of the camera's mounting; (*c*) a diagram of the kinematics chain. 51
3.25 Images produced by IPM using different positions of the Pan and Tilt Unit (PTU): (*a*) pan angle 0 degrees, tilt angle -30 degrees; (*b*) pan angle 0 degrees, tilt angle -45 degrees; (*c*) pan angle 30 degrees, tilt angle -30 degrees. 51
3.26 Overlaying laser range measurements onto the IPM image: (*a*) image from left camera; (*b*) image from right camera; (*c*) bird's eye view with laser range measurements (red dots). 52
3.27 Flood fill based road detection: (*a*) original image; (*b*) binarized image; (*c*) flood fill from inside the road; (*d*) erase upper region of the image; (*e*) flood fill from outside the road; (*f*) final mask of the region contained by the road. 53
3.28 Road detection using a flood fill based algorithm: (*a*) intersection; (*b*) right turn; (*c*) approaching the cross. 54
3.29 (*a*) the result of a top hat operation on the IPM image; (*b*) the brightness profile generated after the top hat image. 55
3.30 A diagram of the search routine that in ran over each candidate group. 55
3.31 Testing a line candidate: (*a*) computing several statistical descriptors; (*b*) testing a candidate (green accepted as lane marker, red discarded). 56
3.32 Lane marker detection on the ADC: (*a*) a left turn; (*b*) a right turn. 56
3.33 Examples of the lane marker detection in images from real roads. 57
3.34 Road maintenance area detection: (*a*) polar representation of the orange segmented mask (in yellow) and convex hull (blue); same information reprojected to Cartesian space. 58
3.35 A sequence of navigation inside the road maintenance area, from (*a*) to (*b*) to (*c*) to (*d*). In the images, the obstacle free region is shown in blue and the orange detected color is marked in yellow. 59
3.36 (*a*) A radial search over the left side dilated cone mask; (*b*) the detection of the bases of the cones on the left (blue) and on the right (red). 59
3.37 The non-holonomic model of the vehicle is used to define the paths the robot will execute as a function of the angle imposed on the steering wheels. 61
3.38 An example of a set of paths defined using equally spaced wheel angle and arc length between nodes. 63

3.39 The several navigation markers for a common road scenario. The road is described by the left, central and right points, $\mathtt{L}^k$, $\mathtt{C}^k$ and $\mathtt{R}^k$, represented by the triangles in the figure. The attractor points ($\mathbf{A}^k$, represented as circles in the figure) are generated using the central lane marker points combined with the road width (Λ) and the desired lane positioning behavior directive (Γ). Laser obstacles generate repelling points ($\mathtt{U}^k$) and are represented by circles with a cross. 64

3.40 An example of the calculation of Ω^1 for nodes ${}^2\Pi^{i-1}$ and ${}^2\Pi^i$. For each node, the minimum distance to all attractor points is selected (represented with a solid line). Then the average of these minima is calculated. In this figure the Ω^1 criteria will have the following score arrangement: $\hat{\Omega}^1({}^j\zeta) > \hat{\Omega}^1({}^{\cdots}\zeta) > \hat{\Omega}^1({}^2\zeta) > \hat{\Omega}^1({}^1\zeta)$. . . . 65

3.41 Calculation of Ω^2 scores. For each node, the minimum angular difference to the closest attractor points is selected. Then the average of these minima is calculated. In this figure, because higher index trajectories suit better the attractor points heading, the following occurs: $\hat{\Omega}^2({}^j\zeta) > \hat{\Omega}^2({}^{\cdots}\zeta) > \hat{\Omega}^2({}^2\zeta) > \hat{\Omega}^2({}^1\zeta)$. 67

3.42 Calculation of Ω^3 scores. For each node, the minimum distance to the closest repelling points is selected. Then the average of these minima is calculated. In this figure the $\hat{\Omega}^3({}^j\zeta)$ score will be the highest of all trajectories, since path ${}^j\zeta$ is the one with best laser obstacle clearance. 68

3.43 Polyline representation of the robot using the robot width as an estimate for defining several line segments. They are then tested for intersection with the lines defined by consecutive laser obstacles repulsor points $\mathtt{U}$. In this case, path ${}^2\zeta$ intersects with the laser obstacle (marked by stars), which leads to a score $\hat{\Omega}^4({}^j\zeta) = 0$. 69

3.44 Some examples of path evaluation. The paths are represented by the green arcs. The highest scoring path is highlighted in yellow. 70

3.45 Some examples of path evaluation in obstacle populated scenarios. The paths are represented by the green arcs. The highest scoring path is highlighted in orange, and the laser obstacles are represented by the thick magenta lines. 71

3.46 Typical predefined paths used in the ADC: (*a*) turn scenarios; (*b*) lane change scenarios; (*c*) emergency maneuvers; (*d*) same as in (*b*) but paths have twice as many nodes. 72

3.47 Some examples of path evaluation, attractor point is represented by the blue circle, obstacle by the red lines. 73

4.1 An example of three nodes exchanging messages. *a*) the message exchange setup, with nodes signalled as ellipses and topics as squares; (*b*) a log of the message traffic for each topic. Horizontal scale in seconds. 81

4.2 A small two joint robot for case studying the *tf* ROS package. (*a*) a scheme of the robot with the annotated links; (*b*) the connection tree of the links. 84

4.3 (*a*) A diagram of the nodes running; (*b*) the frequency of messages published by the nodes. . . . 84
4.4 Movement of the joints: (*a*) Time 0 seconds, Joint 1 at 0 degrees, Joint 2 at 0 degrees; (*b*) Time 2 seconds, Joint 1 at 25 degrees, Joint 2 at -25 degrees; (*c*) Time 4 seconds, Joint 1 at 50 degrees, Joint 2 at -50 degrees; (*d*) Time 6 seconds, Joint 1 at 75 degrees, Joint 2 at -75 degrees; . . . 85
4.5 A 3D model of the *AtlasCar* and its onboard sensors visualised using 3D visualization environment for robots using ROS (RVIZ) and the *AtlasCar* Unified Robot Description Format (URDF). An interactive marker (red circle) enables the user to set the desired laser position. . . . 87
4.6 A simple publish subscribe setup. . . . 88
4.7 An example of a time sequence (vertical axis) of a publish/subscribe message exchange. The publishing module (on the left) has a faster cycle time (represented by the brackets) than the subscriber module (on the right). Because of this, the subscriber module receives two messages while it is still processing the first one. . . . 89
4.8 A query respond with heartbeat setup. . . . 90
4.9 A query response time flow. Heartbeat messages (dotted line) are published whenever the server module (on the left) generates new information. The query module (on the right) queries the information (dashed line) only when it actually requires it. A query message is followed by a response message with the actual data. . . . 91
4.10 A query response time flow using shared memory. The query module (on the right) queries for the shared memory id (dashed arrow). The server module (on the left) responds with the information (dotted arrow). The query process then attaches itself to the shared memory and unmarshalls the data whenever a heartbeat is received. . . 92
4.11 The Camera acquisition module. . . . 93
4.12 The laser acquisition module. . . . 94
4.13 The PTU control module. . . . 94
4.14 The base module. . . . 94
4.15 The remote operation module. . . . 95
4.16 The sound player module. . . . 95
4.17 The inverse perspective mapping module. . . . 96
4.18 The output of the inverse perspective mapping module. . . . 96
4.19 A diagram of the several coordinate frames defined on the *AtlasCar*.(*a*) isometric view; (*b*) top view; The XYZ axis are represented in red green and blue colors respectively. . . . 98
4.20 A diagram of the tree structure of all the coordinate frames define on-board the *AtlasCar*. . . . 99

4.21 The Computation graph of the Vehicle Electronics Interface, Vehicle Manual Control, Coordinate Frames Transforms and Device Mapper layers. These are basic functionalities of the *AtlasCar*. In this diagram, nodes are represented as ellipses and messages as rectangles. 101

4.22 Camera layer computation graph. 102

4.23 Stereo camera layer 104

4.24 Stereo processing on the *AtlasCar*. (*a*) the image from the left camera, common to both stereo pairs; (*b*) the disparity image of the short stereo pair; (*c*) the disparity image of the wide stereo pair. 105

4.25 Point clouds obtained from stereo. (*a*) fake colors showing the stereo pair provenience of each point (*red* short, *green* wide pair) ; (*b*) and (*c*) the same scene with true textures. 105

4.26 The computation graph of the PTU control layer. 106

4.27 Direct control of the PTU. 106

4.28 PTU foveation control. In this sequence, the target marker (green ball) is moved from right to left (*top* to *bottom* sequence in the figure); (*a*) a 3D visualisation of the PTU and the marker; (*b*) the image from the peripheral camera; (*c*) the image from the foveated camera. 107

4.29 Planar laser range finders layer. 107

4.30 The computation graph of the 3D laser range finder layer. 108

4.31 The laser scans obtained with the 3D laser on different positions. 108

4.32 3D reconstruction using the 3D laser on-board the *AtlasCar*. (*a*) reconstructed scene obtained by the accumulation of several scans show in Fig. 4.31; (*b*) the same scene with fake colors. 109

4.33 Multi target tracking. The four on-board laser scanners provide laser scans on different planes: (*yellow*) right bumper laser; (*blue*) left bumper laser; (*red*) center roof laser; (*magenta*) rotating laser; The tracking is performed on a plane, in this case defined on the XY plane of the vehicle center bumper frame, represented by the grey grid. 110

4.34 The computation graph of the multi target tracking layer. 111

4.35 An example of the planar point cloud fusion. (*a*) a scene showing all the laser scans and the tracking plane; (*b*) the same scene with the points included in the fused point cloud signaled as green spheres. 111

4.36 A tracking sequence (*top* to *bottom*). (*a*) A 3D representation of the scene; (*b*) the image taken from the scene. 112

5.1 (a) A typical road scene with a camera mounted on the host vehicle facing the road. The camera reference system is labelled $X_cY_cZ_c$ and the road reference system is labelled $X_rY_rZ_r$. (b) An example of an image captured by the camera. This image is used as input to IPM. (c) The output image of IPM. Since the road is viewed from above no perspective distortion is present. 118

5.2 Some examples of problems with the current IPM method. The input image *(a)* is projected with a small error in pitch estimation, which results in an IPM image where lines are not parallel *(c)*. In *(b)*, the presence of other vehicles also disrupts the resulting IPM image *(d)*. 119

5.3 A typical road scene. The host vehicle has a camera mounted on the roof. Note that the figure shows the reference systems of both the vehicle and the road, since they may not coincide. 124

5.4 An example of a pixel that cannot be projected (*green*) since its optical ray intersects the road plane on the back of the image plane. Inversely, the pixel in *red* is projectable. 125

5.5 The half space of projection computed after the image plane. 126

5.6 The desired area of perception, polygon (ψ_{dap}) in *green*. All vertices of this polygon should contained by the half space of projection, according to (5.19). 127

5.7 The projection of image boundary pixels onto the road plane results in the image boundaries polygon (ψ_{Γ}). 128

5.8 A typical urban road scenario with several obstacles near the host vehicle. 129

5.9 A road scenario with several obstacles: isometric view (*a*) and top view (*b*). The projection polygon ($\psi_{projection}$) is shown in red. It is obtained by the interception of the desired area of perception (ψ_{dap}) in green, the image boundaries polygon (ψ_{Γ}) in blue and the laser generated polygon (ψ_{laser}) in yellow. 130

5.10 Two of the test platforms used for testing the proposed approach: dual camera PTU unit (*a*); the ATLASMV small scale robot (*c*), equipped with a Laser Range Finder (LRF) and a multi-camera perception unit (*b*). 131

5.11 The entire projection obtained using *Camera 0* of the *ATLASMV* test platform *(a)*. The correctly projected pixels *(b)*. Pixels that where incorrectly projected *(c)*. Bellow each image, an enlarged region of the pixels in shown. 132

5.12 Some key frames of the test sequence. *First row*: images taken from *Camera 0*, the blue area is the area of projected pixels, the red is the area outside the desired area of perception and the green area is the area outside the half plane of projection; *Second row*: a map of the projection. Projected/unprojected pixels from *Camera 0* in green/red. Projected/unprojected pixels from *Camera 1* in magenta/blue; *Third row*: the IPM resulting image after mapping both cameras. In columns, different snapshots of the test sequence: 0 (*a*), 2 (*b*), 5.5 (*c*) and 12 (*d*) seconds. 133

5.13 (*a*) Time taken to perform the IPM projection for both cameras. The classic IPM time and the time of the proposed approach are shown. (*b*) Percentage of time saved and number of projected pixels. Proposed approach compared to the classic approach. Key frames of Fig. 5.12 signaled as the vertical black lines. 134

5.14 The resulting IPM projection when errors in yaw (*a*), (*b*) and in pitch (*d*), (*e*) are introduced in the calculation. The reference projection, where no errors were introduced, is shown in (*c*). 135

5.15 IPM accuracy (η_{IPM}) scores for errors in camera pose. Results are presented for errors in yaw and pitch angles. 135

5.16 Obstacle positioned at: (*a*) 0.3 meters in front; (*b*) 0.5 meters to the left; (*c*) 0.75 meters to the right; and (*d*) 1.5 meters in front. 136

5.17 IPM accuracy (η_{IPM}) for the classic IPM (*dotted lines*) and the proposed approach (*dashed lines*) 136

5.18 Comparison of the classical IPM (*middle column*) with the proposed multi-modal IPM (*right column*). The input image (*left column*) shows the image projection polygon highlighted in yellow. Four examples are shown, one in each row. 138

5.19 Using the proposed IPM approach in real scenarios. (*a*) Images of the three cameras onboard the *AtlasCar*; (*b*) The distribution of mapping for each camera ; (*c*) IPM using just green camera; (*d*) IPM using all cameras 139

6.1 Images involved in a color correction procedure: (*a*) source image (**S**); (*b*) overlapping area of the source ($\mathbf{S}_p$); (*c*) overlapping area of the target ($\mathbf{T}_p$); (*d*) overlapping area of the corrected image ($\hat{\mathbf{T}}_p$); (*e*) the target image (**T**); (*f*) mosaic of source and target; (*g*) mosaic composed of the source and corrected images. 143

6.2 An example of a source (**S**) and target (**T**) image pair: (*a*) source image; (*b*) target image; (*c*) the Mean shift color segmentation of the **T** image. 148

6.3 Distribution of the parameters of eq (6.3) for the regions segmented in the target source image pair displayed in Fig. 6.2. 150

6.4 Color Influence Maps for the four regions highlighted in Fig. 6.2 (*c*). 151

6.5 Color correction of the target image: (*a*) original image; (*b*) global approach from [Reinhard *et al.* 2001]; (*c*) local approach from [Tai *et al.* 2005]; (*d*) Mean shift based approach. 152

6.6 Mosaics obtained after color correction: (*a*) with original target image; (*b*) corrected using global approach from [Reinhard *et al.* 2001]; (*c*) corrected using local approach from [Tai *et al.* 2005]; (*d*) corrected using Mean shift based approach. 153

6.7 Color correction examples of Westminster Abbey, London (*first column*), Golden Gate bridge, San Francisco (*second column*), Astronomical clock, Prague (*third column*), and Big Ben, London (*fourth column*). The mosaic with the original source image (the ground truth) (*first row*), global color correction (*second row*), local approach (*third row*) and Mean shift based color correction (*fourth row*). 154

6.8 Color correction examples of Sagrada Familia, Barcelona (*first column*), Ponte Vechio, Florence (*second column*), and Times Square, New York (*third column*). The mosaic with the original source image (the ground truth) (*first row*), global color correction (*second row*), local approach (*third row*) and Mean shift based color correction (*fourth row*). 155

6.9 A mosaic of two images of a city landscape scene *(c)*. The colors of the target image *(a)* must be corrected using information from the source image *(b)* to avoid the artifacts present in the mosaic. 157

6.10 Single channel color segmentation of the Source *(a)* and Target *(e)* images. Single channel histograms, Gaussian components (*dashed*) and total Gaussian mass (*solid black*) of the source *(b)* and target *(f)* images. 3D color segmentation of the source *(c)* and target *(g)* images. Color distribution of all pixels (*green dots*) and 3D Gaussian components (*red ellipsoids*) of the source *(d)* and target *(h)* images. 158

6.11 The mapping of the red color channel for the target / source image pair of Fig. 6.9. The global color transfer function is obtained using [Reinhard *et al.* 2001]. Several local color transfer functions are obtained using [Tai *et al.* 2005], which combined result in a non linear weighted color transfer function. 159

6.12 The *AtlasCar* robotic platform with several onboard cameras (*top*). Onboard images taken with two cameras from a typical road scene. Source image (*bottom-left-top*), Target image(*bottom-left-bottom*); the mosaic of both which shows a clear difference in colors (*bottom-right*); 165

6.13 Comparison of standard $\mathcal{N}$ calculation (*dashed blue*) with truncated $\mathcal{N}$ maximum likelihood estimation (*dashed red*). A ground truth Gaussian model (*solid green*) was used to generate the observed data sets for color values (*signal histogram*). 172

6.14 Examples of the modeling procedure for two images: case 1, *top* and case 2, *bottom*. The joint image histograms with the estimated color pallete mapping function in blue (*a*) and (*d*): the modeled Gaussians (*b*) and (*e*); and a 3D view of the modeled Gaussians (*c*) and (*f*). 173

6.15 The images used in this case study: the mosaic of both source and target images is shown without color correction *(*a), the source image is on the *left* and the target image is on the *right* of the mosaic; the areas of the source and target images that overlap, respectively S_p *(b)* and T_p *(c)*; and the result of color segmentation of T_p using Mean shift *(d)*. 176

6.16 Graphs from region 1 (*first row*), 5 (*second row*) and 7 (*third row*): the regions, *(a)*, *(d)* and *(g)*; the joint image histograms with the computed local color palette mapping function in blue *(b)*, *(e)* and *(h)*; and the fitted Gaussians *(c)*, *(f)* and *(i)*. 177

6.17 emph(a) Joint image histograms of the color segmented regions (shown in Fig. 6.15 (*d*)). *(b)* Color palette mapping functions (CPMF). Data is assigned to the corresponding region color (see Fig. 6.15 *(d)*). 179

6.18 Results of this case study: the target image (a); the Mean shift color segmentation of the target image (b); the corrected image (c); the mosaic obtained after performing color correction with the proposed approach (d). 180

6.19 Mosaics obtained using the image pairs from the data set of [Xu & Mulligan 2010]. Synthesized data set: (a) image 17; (b) image 32 ; and (c) image 40. Real data set: (d) image 8; (e) image 21; and (f) image 23. (top) Original mosaics; ($bottom$) Results from the proposed approach. 183

6.20 Analysis of the effect of the color segmentation output to the effectiveness of the color correction performed by the proposed approach. (a) **PSNR** scores; (a) **S-CIELAB** scores. 189

7.1 A snapshot of the MIT viewer application showing data from a log. 195

7.2 Path travelled by the robot during sequence 1. Key locations are annotated both in the map and the zoomed in image. 196

7.3 Location *C* of the sequence. Isometric (a) and bird's eye (c) view of the 3D data; front 6mm (b), front (e), rear (h), left (f) and right (g) camera images; (e) satellite view of the location. 197

7.4 Path travelled by the robot during sequence 2. Key locations are annotated both in the map and the zoomed in image. 198

7.5 Location *E* of the sequence. Isometric (a) and bird's eye (c) view of the 3D data; front 6mm (b), front (e), rear (h), left (f) and right (g) camera images; (e) satellite view of the location. 199

7.6 The *Talos* MIT autonomous vehicle. (a) a picture of the car; (b) the 3D model. . . . 200

7.7 The local sensor filter. Blue points are all the acquired points, green points are the points kept by the filter. (a) a top view of the scene; (b) a detail of the points captured that belong to the vehicle; (c) a detail of a wall close to the ρ_{max} threshold. 202

7.8 The input point cloud (green points) is reduced using the from the spatial resolution filter. The 3D grid is shows b grey lines, while the kept points are signaled in red. *(a)* top view; *(b)* isometric view. 203

7.9 Results from 3D point cloud preprocessing algorithm. In blue the raw 3D data, in green the point cloud after the local sensor filter, in red the point cloud after the spatial resolution filter. 204

7.10 Size of the accumulated point clouds using raw (red) and preprocessed data (blue) as a function of the mission time. (*a*) sequence 1; (*a*) sequence 2. 205

7.11 Normal estimation results using K nearest neighbours: (*a*) $K = 10$; (*b*) $K = 30$; results using radius search: (*c*) $R = 0.1$; (*d*) $R = 0.5$ meters; 208

7.12 Results from the estimated normals consistency filter. Neighbor points are shown in blue, neighbor points orientations as green arrows and mean orientation vector in red. A query point in a tree canopy (*a*) and (*b*); query point on the road (*c*) and (*d*); (*e*) points excluded by the filter (shown in red). 209

8.1 An example of a geometric polygonal primitive used to describe a wall panel, in sequence 1, location *C* (see Fig. A.3). (*a*) image from the front camera; (*b*) image from the front 6mm camera; (*c*) 3D view of the wall panel range points (in gray), the detected polygonal geometric primitive (in blue) and local primitive coordinate system. 218

8.2 Plane detection examples using the freely oriented plane approach in sequence 1, location *C*. Five best Random Sample Consensus (RANSAC) candidates are shown. (*a*) inliers of each plane candidate with different colors; (*b*) a detail of (*a*); (*c*) the bounding polygon of each of the five plane candidates; (*d*) the plane candidates with filled colors. 220

8.3 Plane detection examples using the oriented plane approach in sequence 1, location *C*. The user defined direction was $\mathbf{q}_u$ was defined as the vertical direction. The five best RANSAC candidates are shown. (*a*) inliers of each plane candidate with different colors; (*b*) a detail of (*a*) (the same as in Fig 8.3 (*b*)); (*c*) the bounding polygon of each of the five plane candidates; (*d*) the plane candidates with filled colors. 221

8.4 Using clustering of point clouds in order to separate inliers RANSAC. (*a*) detection without clustering; (*b*) detection with clustering. 223

8.5 Principal Component Analysis (PCA) for plane model coefficients refinement. (*a*) a set of 3D points; (*b*) the fitted plane and the input points (red or green depending on whether they are above or below the plane); (*c*) and (*d*) another example with the polygon shown in Fig. 8.3 (*b*). 224

8.6 The computation of the bounding polygons: (*a*) and (*b*) the RANSAC inliers shown in blue, the projection of these to the support plane shown in green; (*c*) and (*d*) the convex and concave hull polygons in blue and red, respectively. 227

8.7 Location *H*, sequence 2 (see Fig. B.8, page 415). Examples of geometric polygonal primitives detection using different values of area ($\mathbf{A}_t$) and solidity ($\mathbf{S}_t$) minimum thresholds: (*a*) $\mathbf{A}_t = 0.5$, $\mathbf{S}_t = 0.1$; (*b*) $\mathbf{A}_t = 5$, $\mathbf{S}_t = 0.1$; (*c*) $\mathbf{A}_t = 10$, $\mathbf{S}_t = 10$. . . . 228

8.8 Location *C*, sequence 1, influence of the minimum area and solidity thresholds on: (*a*) the number of polygons; (*b*) the number of explained points. 229

8.9 Detection of geometric polygonal primitives in the data sets of sequence 1: (*a*) location *C*; (*b*) location *D*. 232

8.10 Detection of geometric polygonal primitives in the data sets of sequence 2: (*a*) location *F*; (*b*) location *G*;(*c*) location *H*;(*d*) location *I*. 232

8.11 Analysis of the geometric polygonal primitives data compression potential for all the detected primitives in sequence 1: (*a*) the number of vertices versus the number of support points; (*b*) the number of vertices versus the area represented by the polygon. 233

8.12 An analysis of the cascade processing: (*a*) a graph showing the number of points left to process for a given input point cloud, as a function of the index of the detected primitive; (*b*) the ratio of decrease of the remaining points from the input point cloud; (*c*) the number of primitive support points; (*d*) the time it takes to perform the detection of each of the geometric polygonal primitives as a function of the primitives index. . 235

8.13 Qualitative comparison of several surface reconstruction methodologies in location *E* of MIT sequence 1: (*a*) and (*b*) BPA; (*c*) and (*d*) Greedy triangulation (GT); (*e*) and (*f*) Poisson Surface Reconstruction (POIS); (*g*) GPP parameters set 1; (*h*) GPP parameters set 2; 238

8.14 Comparison between the number of primitives (*a*) and total area (*b*) of the meshes generated by all surface reconstruction approaches. All locations of sequence 1. . . . 242

8.15 Qualitative analysis of the one sided Hausdorff distance for the tested algorithms in location *C* sequence 1: (*a*) GT; (*b*) POIS; (*c*) GPP parameters set 1; (*d*) GPP parameters set 2; A Red-Green-Blue color map is used to code the distance. Red represents zero distance and blue maximum distance. 244

8.16 An example of the Hausdorff distance calculation for location *E*, sequence 1: (*a*) the ground truth, i.e., results from BPA; (*b*) the representation generated by GPP 2; (*c*) The points that were sampled in the ground truth mesh; (*d*) The points that were sampled in the GPP 2 mesh; A Red-Green-Blue colormap is used to code the distance. Red represents zero distance and blue maximum distance. 246

8.17 Comparison between the ground truth mesh (in grey) and the polygons generated by GPP (in yellow): (*a*) and (*b*) two different examples of wall surfaces; (*c*) and (*d*) images of the surfaces corresponding to (*a*) and (*b*), respectively. 247

8.18 Representations obtained when using alternatives for the GPP 2 method for location *E*, sequence 1: (*a*) the standard GPP 2, with ground plane and convex hull; (*b*) discarded ground plane, convex hull; (*c*) with ground plane, concave hull; (*d*) discarded ground plane, concave hull. 248

8.19 Results from the Hausdorff distance obtained when using alternatives for the GPP 2 method for location *E*, sequence 1: (*a*) the standard GPP 2, with ground plane and convex hull; (*b*) discarded ground plane, convex hull; (*c*) with ground plane, concave hull; (*d*) discarded ground plane, concave hull. A Red-Green-Blue color map is used to code the distance. Red represents zero distance and blue maximum distance. . . . 249

9.1 Raw 3D points for all locations of scenario 1: (*a*) accumulated point cloud; (*b*) same as in (*a*) but data points from locations are color coded: location *A* (yellow), *B* (red), *C* (green), *D* (blue) and *E* (cyan). 252

9.2 BPA surface reconstruction over the accumulated point cloud of all the locations in sequence 1. 255

9.3 Accumulation of scene reconstructions of the location *A* (yellow), *B* (red), *C* (green), *D* (blue) and *E* (cyan), for the generation of the reconstructed scenario. Reconstructed scenario (*left*) and scenario with color coded scene primitives origin (*right*); (*a*) BPA;(*b*) GT;(*c*) POIS;(*d*) GPP 2, shown without the ground plane for an easier visualization. 256

9.4 Details of images shown in Fig. 9.3: (*a*) and (*c*) BPA and GPP 2 reconstruction of an object, respectively; (*b*) and (*d*) the same for another object. Color code: location *A* (yellow), *B* (red), *C* (green), *D* (blue) and *E* (cyan). 257

9.5 Comparison of alternatives to scenario reconstruction: (*a*) reconstruction after accumulating all the point clouds; (*b*) reconstructing each point cloud and then merging local meshes. Color code: location *C* (green), *D* (blue) and *E* (cyan). 258

9.6 Orthogonal part of the geometric polygonal primitives expansion operation: (*a*) points are tested for their orthogonal distance to the support plane; (*b*) included points are projected to the support plane. 260

9.7 Expansion of geometric polygonal primitives. The polygonal primitive (polygon in black) that is to be expanded. In the next received range measurements, a point cloud containing candidates for expansion is defined. These have inliers (points that belong to the same object of the polygon, circles in Figure) and outliers (points that do not belong to the same object of the polygon, squares in Figure). 261

9.8 Expansion of geometric polygonal primitives. Solid lines in color represent the convex hull at the start of a given iteration, dashed lines the expanded polygon. Crosses over points mean they where added to the polygon in a given expansion: (*a*) iteration 0 (blue); (*b*) iteration 1 (green); (*c*) iteration 2 (red); (*d*) iteration 3 (cyan); (*e*) iteration 4 (magenta); 262

9.9 The effect of the longitudinal offset value, four examples: (*a*) the case shown in Fig. 9.8; (*b*) the offset value is too large and the expansion propagates to the outliers; (*c*) a non iterative example where the offset value is tunned not to include any outliers; (*d*) a non iterative example where the offset value is tunned to include all inliers. 263

9.10 GPP 1 reconstruction for sequence 1: (*a*) The reconstructed scenario after the input point cloud of location *A* is received; (*b*) after location *B*; (*c*) after location *C*; (*d*) after location *D*; (*e*) after location *E*. 270

9.11 Comparison between the number of polygons generated by the algorithm using expansion and not using expansion through sequence 1. 271

9.12 Comparison of the processing time used in the expansion, detection and other parts of the algorithm during a pass through sequence 1: (*a*) algorithm with the expansion mechanism; (*b*) algorithm without the expansion. 271

9.13 Analysis of the expansion mechanism: (*a*) number of points to use as input to the detection; (*b*) number of detection searches. 273

9.14 Number of accumulated points during sequence 1 according to their ground, vertical or non explained attributes: (*a*) algorithm with the expansion mechanism; (*b*) algorithm without the expansion. 275

9.15 Analysis of the ground polygons data. Comparison between the algorithm using expansion and not using expansion through sequence 1. (*a*) number of ground points explained; (*b*) area of the ground polygons. 275

9.16 Analysis of the vertical polygons data. Comparison between the algorithm using expansion and not using expansion through sequence 1. (*a*) number of vertical points explained; (*b*) area of the vertical polygons; 276

9.17 Analysis of the evolution of each of the polygons through sequence 1. Only pair index polygons are shown to simplify the graphs: (*a*) number of support points per polygon; (*b*) number of vertices in the convex hull; (*c*) area of the polygons. 277

9.18 Analysis of the evolution of each of the polygons through sequence 1, in particular the number of vertices in the convex hull. Only pair index polygons are shown to simplify the graphs. 278

9.19 Analysis of the updating of the support planes of primitives after an expansion. Sequence 1: (*a*) polygonal primitive 2; (*b*) polygonal primitive 6; 279

9.20 Evolution of polygonal primitive 0 (the ground plane) through sequence 1: (*a*) location *A*; (*b*) location *B*; (*c*) location *C*; (*d*) location *D*; (*e*) location *E*; 280

9.21 Evolution of polygonal primitives through sequence 1 (*top*) and sequence 2 (*bottom*): (*a*) location *C*, sequence 1; (*b*) location *D*, sequence 1; (*c*) location *G*, sequence 2; (*d*) location *H*, sequence 2. 280

10.1 Projecting a chessboard image onto several projection planes: (*a*) the image to be projected; (*b*) the projection planes and a representation of the camera's position as the red-green-blue reference system. 286

10.2 Projection using complete mapping versus texture mapping for the three projection planes displayed in Fig. 10.1 (*b*): (*a*) complete mapping, plane 1; (*b*) texture mapping, plane 1; (*c*) complete mapping, plane 2; (*d*) texture mapping, plane 2; (*e*) complete mapping, plane 3; (*f*) complete mapping, plane 3. 287

10.3 Projection of the chessboard image using Data Dependent Triangulation (DDT) mapping: (*a*) DDT triangulation ; (*b*) DDT mapping, plane 1; (*c*) DDT mapping, plane 2; (*d*) DDT mapping, plane 3. 291

10.4 Comparison of the texture mapping (red diamonds) and DDT mapping (blue squares). Background projection is obtained by performing the complete mapping of pixels, so it may be regarded as ground truth. Only the chessboard corners are shown. The three projection planes displayed in Fig. 10.1 (*b*): (*a*) plane 1; (*b*) plane 2; (*c*) plane 3; (*d*) the average projection error as a function of the difference between the image optical axis vector and the plane's normal vector. 292

10.5 Data dependent triangulation: (*a*) naive triangulation; (*b*) line segments detected using Hough transform; (*c*) Delaunay triangulation using line segments vertices as input; (*d*) constrained Delaunay triangulation using vertices and line segments as input. 296

10.6 Projection using complete mapping (*left column*), texture mapping (*middle column*) and data dependent triangulation mapping (*right column*). The three projection planes displayed in Fig. 10.1 (*b*) are shown: planes 1, first row, plane 2, second row, and plane 3, third row. 297

10.7 Images from onboard cameras sometimes view portions of the host vehicles: (*a*) *AtlasCar* stereo camera; (*b*) *Talos* forward center camera. 299

10.8 Image bounding polygons for the camera of the *Talos* viewing location *C*, sequence 1: (*a*) forward camera; (*b*) 6 millimeter lens forward camera; (*c*) rear camera; (*d*) forward left camera; (*e*) forward right camera; (*f*) the coordinate frames of the all the cameras. 301

10.9 Projections of the image bounding polygons shown in Fig. 10.8 to the world coordinate frame, considering the ground plane primitive (shown in blue). Data from location *C*, sequence 1: (*a*) the scene viewed from a birds eye view; (*b*) the projections of the image bounding polygons. 304
10.10 The algorithm for computing the projection polygon in image space: (*a*) location *D*, sequence 1, a single primitive detected (blue); (*b*) the image bounding polygon (dark color); (*c*) the projection polygon (dark color); (*d*) the image with the image bounding polygon (red) and the projection polygon (blue) 306
10.11 Demonstration scene for mapping images to geometric polygonal primitives: (*a*) a view of the image of the scene; (*b*) the 3D scene with the location of the camera and the support plane of the primitive that corresponds to the wall of the house, where the windows and the door are. 308
10.12 The projection polygon: (*a*) in the image space; (*b*) in 3D space. 308
10.13 Setting up the constrained Delaunay triangulation: (*a*) line segments detected using Hough lines transform; (*b*) vertices and constraints used as input to the constrained Delaunay triangulation. 309
10.14 Triangular mesh: (*a*) in image space; (*b*) projected to 3D space. 310
10.15 Triangular mesh in 3D space: (*a*) triangles and texture; (*b*) texture only. 311
10.16 A detailed view of the triangulated mesh: (*a*) the index of each vertex; (*b*) the projection polygon overlayed onto the mesh. 312
10.17 The inside or outside state of each triangle (or face) in the mesh. 314
10.18 Texture mapping only triangles inside the projection polygon: (*a*) triangles and texture; (*b*) texture only. Figure 10.15 shows results when all triangles are mapped. . . . 314
10.19 Texture mapping a primitive with images from different cameras: (*a*) the scene; (*b*) image from the forward center camera; (*c*) image from the forward center 6 millimeter camera; (*d*) right side detail of the triangulated mesh, forward center camera; (*e*) the same for forward 6 millimeter camera; (*f*) primitive with texture using forward center camera; (*g*) the same for 6 millimeter camera. 316
10.20 A situation where two cameras map different portions of the same polygon: (*a*) the scene with the vehicle position; (*b*) image from the forward left camera; (*c*) image from the forward center camera. 318
10.21 A case where a single image texture maps several primitives: (*a*) the image scene; (*b*) the support planes of each primitive; (*c*) the projection polygons in image space; (*d*) the projection polygons in 3D space; (*e*) the detected Hough line segments; (*f*) the input vertices and constraints to the Delaunay triangulation. 319
10.22 Triangulated meshes and textured primitives: (*a*) meshes in image space; (*b*) meshes in 3D space; (*c*) and (*d*) texture mapped primitives. 320

10.23 Two different views of a textured complex scene representation, sequence 1. 321

11.1 Local image meshes from example shown in Fig. 11.8: (a), (b), (c), (d), local triangulated meshes from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively. 326

11.2 Textures obtained from local image meshes of the shown in Fig. 11.8: (a), (b), (c), (d), textures from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively. 327

11.3 Textures obtained from local image meshes of the shown in Fig. 11.8: (a), (b), (c), (d), textures from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively; ($e$) the overlay of all textures using an alpha channel fusion strategy. 328

11.4 Adding all vertices in the local image meshes to produce a global mesh. Projections are colored with a black to orange colormap: (a), (c), (e) and (g) all the vertices used as input to the global mesh with one, two, three and four projections, respectively; (b), (d), (f) and (h) textures generated for each of the meshes computed from the set of vertices from (a), (c), (e) and (g), respectively. (i) a detail of (h). 329

11.5 Conditional adding of all the vertices in the local image meshes to produce a global mesh. Projections are colored with a black to orange colormap: (a), (b) and (c), all the vertices used as input to the global mesh with two, three and four projections, respectively; (d) the texture mapped from the final stage of the global mesh. 330

11.6 Mapping using triangle quality functions. 331

11.7 An example of overlapping projection polygons in multiple projections of the same mission time: (a) a 3D scene with the primitive $k = 0$ (in blue) and the image bounding polygons (black to orange colormap); (b) the same scene with the projection polygons represented; (c) a detailed view of (b); (d), (e), (f), (g), (h), images from projections $\mathbf{C}^{\{k=0,l=0,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=1,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=2,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=3,t=t_p\}}$ and $\mathbf{C}^{\{k=0,l=4,t=t_p\}}$, respectively. 332

11.8 An example of overlapping projection polygons in multiple projections at different mission times t_0, t_1 and t_2: (a) a 3D scene with the primitive $k = 4$ (in blue) and projection polygons (black to orange colormap); (b) the vehicles position at times t_0, t_1 and t_2, respectively from right to left in the figure: (c), (d), (e), (f), images from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively. 334

11.9 The triangles overlap test: (a) intersection returns points, overlap true; (b) intersection returns line segments, overlap true; (c) intersection returns polygons, overlap true; (d) intersection returns empty, overlap false; (e) intersection returns points, overlap false; (f) intersection returns line segments, overlap false; 339

11.10 The insertion operation: (*a*) candidate triangle and initial mesh; (*b*) insertion of candidate triangle's vertices; (*c*) insertion of the candidate triangle's vertices and constraints; (*d*) preparation of the mesh followed by the insertion of the candidate triangle's vertices and constraints. 342
11.11 The insertion operation, example 1: (*a*) candidate triangles and initial mesh $\mathtt{M}$; (*b*) first insertion, mesh $\mathtt{M}^{*}$; (*c*) second insertion, mesh $\mathtt{M}^{**}$; (*d*) third insertion, mesh $\mathtt{M}^{***}$. 345
11.12 The projection parent status of each triangle (same example as in Fig. 11.11): (*a*) candidate triangles and initial mesh $\mathtt{M}$; (*b*) first insertion, mesh $\mathtt{M}^{*}$; (*c*) second insertion, mesh $\mathtt{M}^{**}$; (*d*) third insertion, mesh $\mathtt{M}^{***}$. 347
11.13 Insertion over orphan triangles (same example as in Fig. 11.11): (*a*) candidate triangles and initial mesh $\mathtt{M}^{***}$; (*b*) the same as in (*a*) but with the color of each triangle associated to the corresponding parent projection. 348
11.14 Times Square: example of a photometrically non static environment. 350
11.15 Example of images from two different projections but from the same camera: (*a*) image with good quality; (*b*) image with bad quality. 351
11.16 Quality functions for multiple projections at different mission times t_1 and t_2: (*a*) a 3D scene with the primitive $k = 4$ (in blue) and the vehicle's position at times t_1 and t_2, respectively from right to left in the figure: (*c*), (*d*), (*e*), (*f*), images from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively. 352
11.17 Computing a representation for a vertical wall: (*a*) a 3D view; (*b*) camera front center projection at time t_1, black color; (*c*) camera front center projection at time t_2, red color; (*d*) camera front center projection at time t_3, orange color; (*e*) camera front left projection at time t_3, yellow color; 355
11.18 The evolution of the global primitive's mesh. (*a*) time $t = t_1$; (*b*) time $t = t_2$; (*c*) time $t = t_3$, insertion of front center camera; (*d*) time $t = t_3$, insertion of front left camera. 356
11.19 The evolution of the global primitive's texture. (*a*) time $t = t_1$; (*b*) time $t = t_2$;; (*c*) time $t = t_3$, insertion of front center camera; (*d*) time $t = t_3$, insertion of front left camera. 357
11.20 Global primitive meshes for three different cases of mapping of a single camera to the ground plane primitive: (*a*) front center camera; (*b*) rear center camera; (*c*) front left camera; Colors denote the projection instant: t_1 black, t_2 red and t_3 yellow. . . . 359
11.21 Images and local triangulated meshes for all projections shown in Fig. 11.20: Left column, front center camera (Fig. 11.20 (*a*)); Middle column, rear center camera (Fig. 11.20 (*b*)); Right column, front left camera (Fig. 11.20 (*c*)); First row: time t_1; Second row: time t_2; Third row: time t_3; 360

11.22 Number of triangles from each projection over the total number of triangles in the global mesh as a function of mission time, for the examples shown in Fig. 11.20: (*a*) front center camera; (*b*) rear center camera; (*c*) front left camera; 361
11.23 Distribution of triangles according to projection. Rear center camera only: (*a*) a 3D view; (*b*) projection at time t_1; (*c*) projection at time t_2; (*d*) projection at time t_3 . . . 362
11.24 Images and local triangulated meshes for all projections shown in Fig. 11.23: First row front center teleobjective camera; Second row front center camera; Third row, front right camera; Fourth row, front left camera; Fifth row, rear center camera; Left column: time t_1, (Fig. 11.20 (*a*)); Middle column: time t_2, (Fig. 11.20 (*b*)); Right column: time t_3, (Fig. 11.20 (*c*)); 363
11.25 Number of triangles from each projection over the total number of triangles in the global mesh as a function of mission time, for the examples shown in Fig. 11.23. . . 364
11.26 Full reconstruction of the sequence 1, MIT data set: (*a*) a top view of the entire scenario; (*b*) virtual camera 1, red in (*a*); (*c*) virtual camera 2, green in (*a*); (*d*) virtual camera 3, blue in (*a*); (*e*) virtual camera 4, yellow in (*a*); 366
11.27 Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_1 = 30$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively. 367
11.28 Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_2 = 60$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively. 368
11.29 Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_3 = 90$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively. 368
11.30 Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_4 = 120$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively. 369
11.31 Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_5 = 150$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively. 369
11.32 Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_6 = 180$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively. 370

A.1 Location *A* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 407
A.2 Location *B* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 408
A.3 Location *C* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 408

A.4 Location *D* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 409

A.5 Location *E* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 409

B.1 Location *A* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 411

B.2 Location *B* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 412

B.3 Location *C* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 412

B.4 Location *D* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 413

B.5 Location *E* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 413

B.6 Location *F* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 414

B.7 Location *G* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 414

B.8 Location *H* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 415

B.9 Location *I* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location. 415

C.1 Qualitative comparison of several surface reconstruction methodologies in location *A* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 417

C.2 Qualitative comparison of several surface reconstruction methodologies in location *B* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 418
C.3 Qualitative comparison of several surface reconstruction methodologies in location *C* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 418
C.4 Qualitative comparison of several surface reconstruction methodologies in location *D* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 419
C.5 Qualitative comparison of several surface reconstruction methodologies in location *E* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 419

D.1 Qualitative comparison of several surface reconstruction methodologies in location *A* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 421
D.2 Qualitative comparison of several surface reconstruction methodologies in location *B* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 422
D.3 Qualitative comparison of several surface reconstruction methodologies in location *C* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 422
D.4 Qualitative comparison of several surface reconstruction methodologies in location *D* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 423
D.5 Qualitative comparison of several surface reconstruction methodologies in location *E* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 423
D.6 Qualitative comparison of several surface reconstruction methodologies in location *F* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 424
D.7 Qualitative comparison of several surface reconstruction methodologies in location *G* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 424
D.8 Qualitative comparison of several surface reconstruction methodologies in location *H* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 425
D.9 Qualitative comparison of several surface reconstruction methodologies in location *I* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2; 425

List of Algorithms

8.1 Detection of a support plane using RANSAC 219
8.2 Clustering of point clouds 222
8.3 Cascade configuration for the detection of geometric polygonal primitives 230
9.1 Cascade configuration for the expansion of existing geometric polygonal primitives and detection of new ones. 268
10.1 Compute which triangles are inside the projection polygon 313

Acronyms

3DGMM	3D Gaussian Mixture Models
AD	Autonomous Driving
ADAS	Advanced Drivers Assistance Systems
ADC	Autonomous Driving Competition
BPA	Ball Pivoting Algorithm
CARMEN	Carnegie Mellon Robot Navigation Toolkit
CIM	Color Influence Maps
DARPA	Defense Advanced Research Projects Agency
DDT	Data Dependent Triangulation
DOF	Degree of Freedom
ELROB	European Land Robot Trial
ERC	European Research Council
EU	European Union
GMM	Gaussian Mixture Models
GPP	Geometric Polygonal Primitives
GT	Greedy triangulation
IMU	Inertial Measurement Unit
IPC	Inter Process Communications
IPM	Inverse Perspective Mapping

LARtk	Laboratory of Automation and Robotics Toolkit
LCM	Lightweight Communications and Marshalling
LRF	Laser Range Finder
MIT	Massachusetts Institute of Technology
OPENCV	Open Source Computer Vision Library
PCA	Principal Component Analysis
PCL	Point Cloud Library
PLC	Programmable Logic Controller
POIS	Poisson Surface Reconstruction
PRO	Portuguese Robotics Open
PROMETHEUS	Programme for European Traffic of Highest Efficiency and Unprecedented Safety
PTU	Pan and Tilt Unit
RANSAC	Random Sample Consensus
ROS	Robot Operating System
RVIZ	3D visualization environment for robots using ROS
SLERP	Spherical Linear Interpolation
URDF	Unified Robot Description Format
VIAC	VisLab Intercontinental Autonomous Challenge
XB3	Point Grey Research Bumblebee XB3 Stereo Camera

Chapter 1

Introduction

Mobility is a cornerstone of nowadays society. Many people spend a considerable part of their day travelling from place to place. It is expected that they do so with comfort and, most importantly, with safety. However, more than 3000 deaths occur each day due to road accidents, worldwide. The principal causes for road accidents are well documented. Nonetheless, the solution to the problem is quite complex. It involves actions on different levels, from reshaping traffic laws, to making roads and cars safer, or to change the awareness of people to the responsability of driving.

Let us consider the problem of how to improve the safety of vehicles. Most people have been in a situation where a passanger suddenly warns the driver about a potential crash and, by doing so, an accident is avoided. In fact, driver distraction is one of the most common causes for accidents. Sometimes there are no additional passengers or they are not paying attention to the road. Let us suppose that it is the vehicle itself that is capable of perceiving the road, and that it can aknowledge potentially dangerous situations. Equipped with these capabilities, cars may cease to be *blind* driven machines and turn into ever vigilant automatic copilots. Faced with a dangerous situation, the vehicle may issue a warning to the driver or, in severe cases, even take control of the brake or steering systems. Automatic scene awareness should make a significant impact on the overall safety of vehicles. To *see* the environment around them, vehicles must be equipped with onboard sensors. The information they provide is then processed in order to grasp an understanding of the scene.

In this thesis, the focus is the study of algorithms that process data from onboard sensors and generate information about the road scene. The potential applications are both Autonomous Driving (AD) as well as Advanced Drivers Assistance Systems (ADAS). In AD applications, it is expected that the automatic systems take full control of the vehicle, i.e., the sytems must drive the vehicle without any human intervention, whereas in ADAS applications, those systems should monitor the actions of the drivers and intervene only when necessary. Although ADAS applications should most likely be the first to be implemented in vehicles, AD research is very interesting as a setting for testing the robustness of the systems. Anyhow, AD and ADAS algorithms share the same design and implementation details, as well as the same ultimate goal which is to make the vehicle aware of the road.

From the research made in both AD and ADAS, there one significant conclusion: no sensor is sufficient by itself. Because of this, recent trends focus on sensor fusion techniques. There are two distinct alternatives to the problem of fusing information from two or more sensors. The first option is to process the data from each sensor separately, drawing conclusions for each indivudual analysis, and then fuse the informations generated by the analysis of the data of each sensor. This is called late fusion. The second option is to merge the raw data from all sensors, and then try to take conclusions from a holistic analysis of the entire data. This is called early fusion. Althouhg there are advantages and disadvantages for each of the approaches, in the work presented in this thesis an early fusion strategy is endorsed. In particular, the work that will be presented focused on the developement of several strategies for obtaining representations of the raw data that can be usefull for subsequent processing algorithms. Questions on which should be an adequate representation are addressed, as well as methodologies to cope with data assyncrony, large size, and of different nature. These proposed representations are referred to as intermediate representations, in the sense that they are intended to stand between the raw sensor data and the subsequent processing algotithms.

Section 1.1 provides an overall view of the impact of road traffic accidents in society. It also refers the concerns of global organizations such as the United Nations, the World Health Organization, or the European Union on this topic. Section 1.2 formulates the objectives of the thesis. Finally, the structure of this document is described in section 1.3.

1.1 Motivation

The General Assembly of the United Nations resolution 64/255 proclaimed 2011-2020 the Decade of Action for Road Safety [United Nations 2010]. The goal is to stabilize and then reduce the fore casted level of global road fatalities by increasing activities conducted at national, regional and global levels. Resolution 64/255, requested the World Health Organization and the United Nations regional commissions, in cooperation with the United Nations Road Safety Collaboration and other stakeholders, to prepare a Plan of Action for the Decade as a guiding document to support the implementation of its objectives.

The reasons for this concern are unfortunately very obvious: each year nearly 1.3 million people die as a result of a road traffic collision. Worldwide, twenty to fifty million more people per year sustain non-fatal injuries from a collision, and these injuries are an important cause of disability worldwide [United Nations & World Health Organization 2011]. Road traffic injuries are among the three leading causes of death for people between 5 and 44 years of age, and they are predicted to become the fifth leading cause of death in the world, resulting in an estimated 2.4 million fatalities each year [United Nations & World Health Organization 2011]. Other estimates for the future are also very concerning. The evolution of DALY (Disability-adjusted life year, a health-gap measure

Table 1.1: Source [World Health Organization 2004]. Change in rank order of DALYs for the ten leading causes of the global burden of disease.

	1990		2020
Rank	Disease or injury	Rank	Disease or injury
1	Lower respiratory infections	1	Ischaemic heart disease
2	Diarrhoeal disease	2	Unipolar major depression
3	Perinatal conditions	3	Road traffic injuries
4	Unipolar major depression	4	Cerebrovascular disease
5	Ischaemic heart disease	5	Chronic obstructive pulmonary disease
6	Cerebrovascular disease	6	Lower respiratory infections
7	Tuberculosis	7	Tuberculosis
8	Measles	8	War
9	Road traffic injuries	9	Diarrhoeal disease
10	Congenital abnormalities	10	HIV

that combines information on the number of years lost from premature death with the loss of health from disability) shows that road traffic accidents will become one of the major causes for loss of quality of life [World Health Organization 2004] (Table 1.1).

In the European Union, although the numbers have been decreasing in the last decade, the fact is that in 2010, road traffic accidents were responsible for 35000 fatalities and 1.5 million injured. Figure 1.1 shows the numbers of fatalities, injuries and accidents in the European Union over the last decade. With regards to the situation in Portugal, in 2010, in spite of the 0.2% slight decrease in the number of road accidents with victims in mainland roads (35426), the resulting number of victims increased also slightly, to 47302 (+0.3% when compared with 2009). In 2010 there were 937 fatalities, 2475 seriously injured victims and the 43890 lightly injured victims [Instituto Nacional de Estatística 2011].

The economic consequences of motor vehicle crashes are also devastating. The economic losses have been estimated between 1% and 3% of the Gross National Product, reaching a total over 500 billion US dollars per year worldwide. Reducing road casualties and fatalities will reduce suffering, unlock growth and free resources for more productive use [United Nations & World Health Organization 2011].

Figure 1.2 provides some more detailed information on the problem. It shows that the majority of accidents occur in rural roads (Fig. 1.2 (*a*)), involve cars or taxies (Fig. 1.2 (*c*)), and that about four out of every five accidents occur in dry weather (Fig. 1.2 (*b*)).

According to [European Commission, Directorate-General for Energy and Transports 2001], the main causes for accidents have been clearly defined:

- Excessive or inappropriate speed, the cause of about a third of fatal and serious accidents.
- The consumption of alcohol and drugs or fatigue. Drinking and driving is responsible for about 10000 deaths each year.
- Failure to wear a seat belt or crash helmet is a major aggravating factor in accidents. If the rate

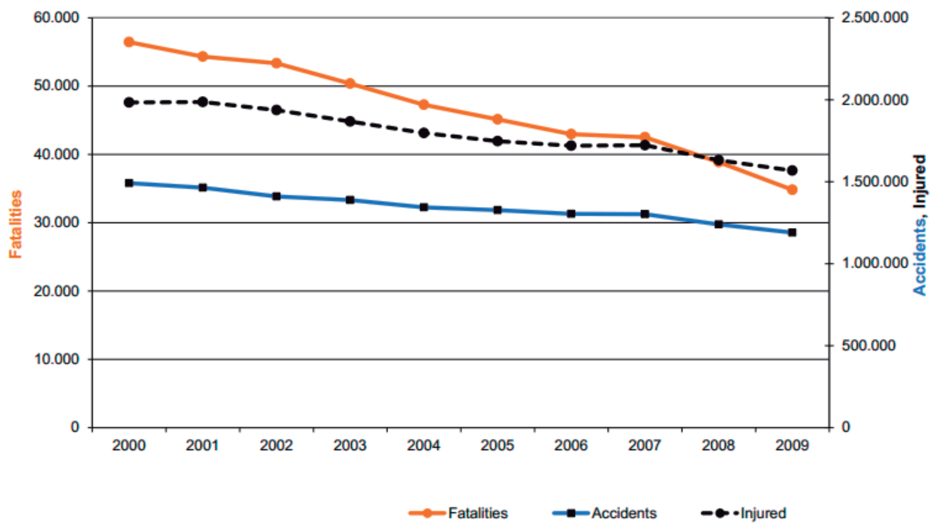


Figure 1.1: Source [European Road Safety Observatory 2011]. Fatalities, accidents and injured per year in the EU countries.

of seat-belt use could be increased everywhere to the best international rate, more than 7000 lives would be saved each year.

- The lack of sufficient protection provided by vehicles in the event of an impact. Analysis of accidents shows that, if all cars were designed to provide protection equivalent to that of the best cars in the same class in the event of an accident, half of fatal and disabling injuries could be avoided.
- High-risk accident sites (black spots).
- Non-compliance with driving and rest times by commercial drivers.
- Poor visibility of other users or an insufficient field of vision for the driver. The lack of visibility in the blind spot towards the rear of vehicles alone causes 500 deaths a year.

The Global Plan for the Decade of Action for Road Safety 2011-2020 [United Nations & World Health Organization 2011] was the result of a combined effort of the United Nations, the World Health Organization and other stakeholders to address the problem of road safety. It serves as a tool to support the development of national and local plans of action, while simultaneously providing a framework to allow coordinated activities at regional and global levels. The plan suggests activities to be taken under five pillars of action:

- Pillar 1, Road safety management;
- Pillar 2, Safer roads and mobility;

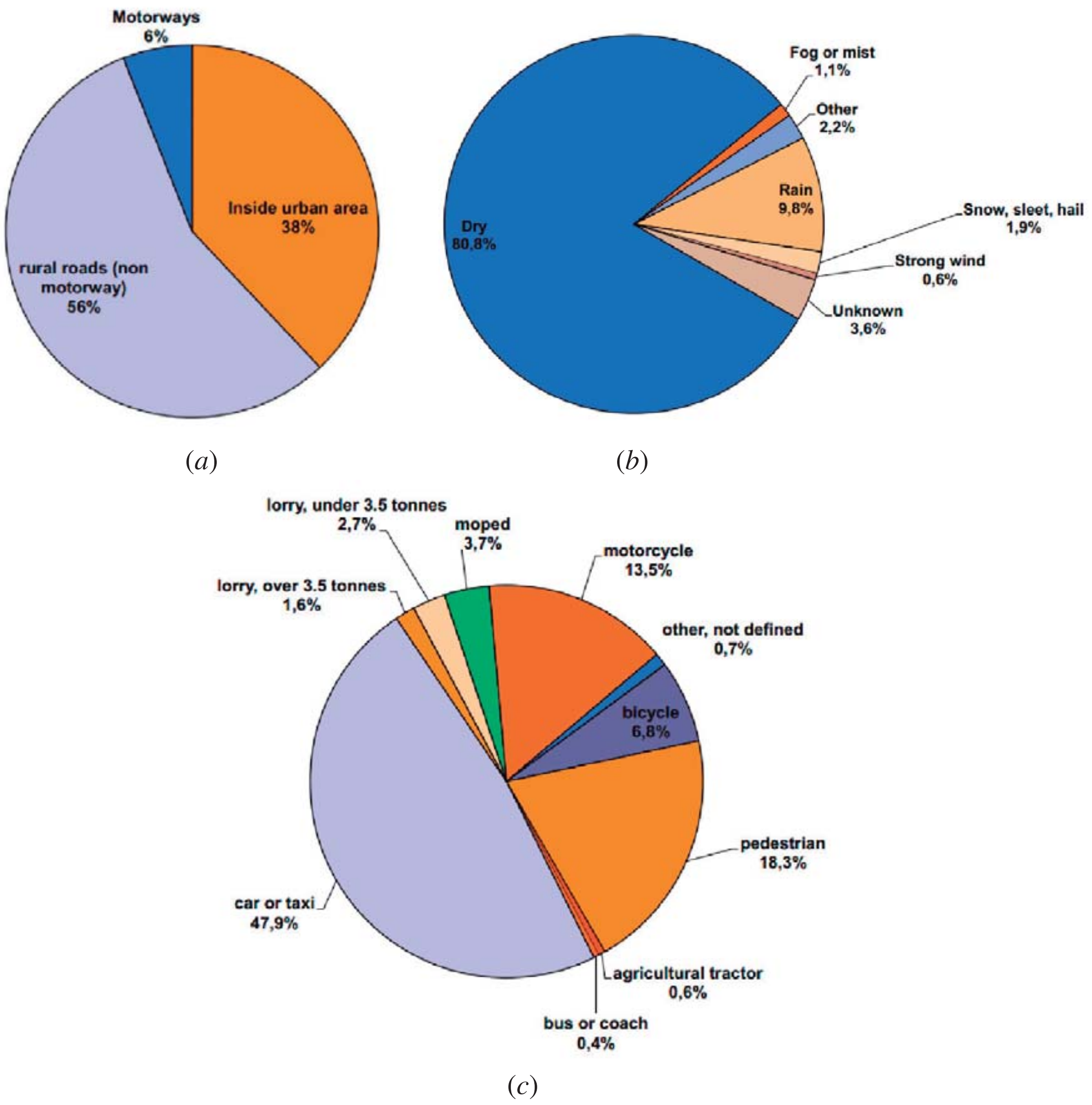


Figure 1.2: Source [European Road Safety Observatory 2011]. Share of fatalities as a function of: (*a*) type of road; (*b*) weather conditions; (*c*) type of vehicle. Data for EU countries, 2009.

- Pillar 3, Safer vehicles;
- Pillar 4, Safer road users;
- Pillar 5, Post-crash response.

The problems addressed in this thesis are embodied on Pillar 3. We discuss and propose state of the art technologies that ultimately may enhance the safety of vehicles. The plan is organized so that each Pillar is broken into seven actions. Actions number six and seven recommend:

- Action 6, encourage application of pedestrian protection regulations and increased research into safety technologies designed to reduce risks to vulnerable road users;
- Action 7, encourage managers of governments and private sector fleets to purchase, operate and maintain vehicles that offer advanced safety technologies and high levels of occupant protection.

In sum, the main motivation of this work is that it addresses a very significant problem worldwide. It is also interesting to note that the work developed is aimed at objectives declared as important by the international community.

1.2 Objectives

As described in section 1.1, the magnitude of the problem of road accidents, as well as the human and economic consequences, are all very concerning. In the scientific community, there is a discussion on whether a technological development effort should be applied to the road infrastructures or to the vehicles themselves. Considering the statistics given in Fig. 1.2 (*a*) one finds that the majority of accidents occur in rural (non highway) roads. In those roads the technological development of the infrastructures seems less likely: it would be expected that the first areas to receive *smart* infrastructures would be cities and highways. This supports the approach of developing onboard technology that requires no changes in the infrastructures. In Fig. 1.2 (*b*) it is possible to see that 80% of fatalities occur in dry weather conditions. Discussions on the robustness of onboard sensors in rough weather conditions like snow or rain are thus less important, since that a system that works well in dry weather conditions would have a very large impact. Finally, Fig 1.2 (*c*) shows that the majority of fatalities involve cars, taxis and other large size vehicles as trucks or lorries. This is also an important factor, since that onboard sensors and computational resources can easily be installed on these type of vehicles. Smaller vehicles like motorcycles or motor-less vehicles present a problem for the technologies that will be discussed. Given all the topics discussed before, it is plausible to assume that technology that can be placed onboard vehicles and contrives to make them safer, will certainly have a large positive impact of the problem of road accidents. Such technologies are now at an early stage of development. Current ADAS commercial applications include several systems, but none that focuses on computing a global description of the scene around the vehicle. The ultimate goal of this work is thus to develop new algorithms that are capable of processing onboard sensor data and provide a high level understanding of the road scene.

The objectives of the work presented in this thesis were the following:

- To develop perception based algorithms that process onboard sensor data and generate information about the road scene;
- To develop and integrate basic software functionalities in the autonomous driving robotic prototypes that allows them to participate in autonomous driving competitions;
- To test and assess the effectiveness of the proposed algorithms on a full scale vehicle;
- To develop alternative data representations that may cope with multiple sensors and that improve the effectiveness of subsequent processing algorithms.

The first three objectives are addressed in chapters 3, 4 and 5. The fourth objective is addressed in chapters 6 through 11.

1.3 Thesis Structure

This thesis is composed of twelve chapters. Chapter 2 provides a state of the art on autonomous vehicles. Then, the topics addressed in the following chapters may be divided into two parts. The first part contains chapters 3 and 4. Chapter 3 describes three robotic prototypes. The author has contributed to the design, construction and development of these robots. The robots are designed to execute autonomous driving tasks, and have participated in several autonomous driving competitions. Chapter 4 addresses the software architectures used in those robots. Some of the software architectures were developed at the Laboratory of Automation and Robotics, where the author has carried out the work. Some other software architectures were just adapted and implemented on several robotic prototypes.

The second part of the thesis proposes algorithms for computing scene representations from onboard sensors. Chapter 5 presents an Inverse Perspective Mapping (IPM) methodology for computing a bird's eye view of the road from multiple cameras, using a Laser Range Finder (LRF) sensor to assist the computation. Chapter 6 presents several algorithms for performing the color correction of images. The objective is to obtain good quality, artifact free image mosaics. Chapters 7, 8, 9, 10 and 11 are partial descriptions of a complete scene representation algorithm that was developed. The algorithm is designed to compute a scene representation using as input data from a 3D laser and several images. The algorithm uses the 3D data to generate a geometric model of the scene, and then applies texture to the model using the images. Additionally, the model of the scene may evolve if newer (and better) data is received. Chapter 7 describes the dataset that is used, along with some 3D data preprocessing algorithms. In chapter 8 the algorithm that generates a geometric model of the scene is described. The refinement mechanism for the geometric model is then presented in chapter 9. Techniques for texture mapping the model are discussed in chapter 10, and chapter 11 addresses the problem of how to evolve the photometric representation when new images are received. Finally, conclusions are drawn in chapter 12.

Some additional remarks should be made on the conventions used in this document. The core chapters of this document (chapters 3 through 11) are structured with a small preamble at the beginning, that briefly describes the chapter. Then, there are always two sections: introduction and related work, that describe the specific problem for each chapter. At the end of each chapter there is section where specific conclusions on the topic are drawn. Throughout the thesis, some figures and graphs present results describing the processing time of algorithms. Unless otherwise noted, those results were obtained using a computer with an *Intel* i7 processor. There are several figures that contain an indication of the coordinate frames. Unless otherwise noted, coordinate frames represented in figures use the following convention. The red color line represents the X axis, the green line, the Y axis, and the blue, the Z axis. Axes are not represented by an arrow, but rather by a line. The positive direction of the axes may be inferred from the origin of the coordinate system and the tip of the line that represents the axis.

Chapter 2

State of the Art

As discussed in chapter 1, this thesis focuses on two main subjects. The first part contains a description of the developed robotic prototypes, and the second part presents intermediate scene representation algorithms. Throghout the thesis, the problems associated with Autonomous Driving (AD) and Advanced Drivers Assistance Systems (ADAS) are the common ground. Because of this, the choice was to provide a state of the art on autonomous vehicles in this chapter, and to include a related work section inside each chapter that adresses more specifically each particular topic. Hence, the descriptions made in this chapter are limited to real size vehicles that can perform autonomous driving tasks.

2.1 Milestones Achieved in Autonomous Driving

Driverless vehicle concepts and trials have been created and run by dozens of companies and universities over the last 35 years. Below is a brief overview of some of the major milestones and more successful efforts, based on [Keyes 2011].

- 1977: Tsukuba Mechanical Engineering Lab in Japan ran a driverless car on a dedicated, marked course. It navigated by following white street markers and traveled at speeds of up to 30 km/h.

- 1980s: Mercedes-Benz tested a vision-guided robot van, designed by Ernst Dickmanns and his team at the Bundeswehr University Munich, that achieved 100 km/h on trafficless streets.

- 1987-1995: The European Commission funded the 800 million Euro EUREKA Programme for European Traffic of Highest Efficiency and Unprecedented Safety (PROMETHEUS) Project on autonomous vehicles.

- 1980s: The Defense Advanced Research Projects Agency (DARPA) funded Autonomous Land

Vehicle in the United States achieved the first road-following demonstration using laser, radar and computer vision. The vehicle drove at speeds up to 30 km/h.

- 1987: HRL Laboratories demonstrated the first offroad map and sensor based autonomous navigation on the ALV vehicle. The vehicle traveled over 200 meters at 3 km/h on complex terrain with steep slopes, ravines, large rocks and vegetation.
- 1994: Daimler-Benz and Ernst Dickmanns put two semi-autonomous vehicles, VaMP and Vita-2, on the road which drove more than 1000 Km on a three-lane highway in heavy traffic at speeds up to 130 km/h. The vehicles were capable of autonomous driving in free lanes, convoy driving, lane changing and passing.
- 1995: A Mercedes-Benz Ernst Dickmanns autonomous car took a 1600 Km trip from Munich to Copenhagen and back. The car reached speeds exceeding 175 km/h on the German Autobahn. Its longest stretch of driving without human intervention was approximately 158 km.
- 1995: The Navlab project by Carnegie Mellon University traveled 5000 Km on a "No Hands Across America" trip. Only the steering was autonomous, the throttle and brakes were human-controlled.
- 1996-2001: Alberto Broggi of the University of Parma ran the ARGO Project, which allowed a driverless car to follow painted lane markings in a highway. The project ended with a six day, 2000 Km trip on the motorways of northern Italy named *MilleMiglia in Automatico*. The car averaged 90 km/h. The longest distance traveled without human intervention was 54 km. The vehicle used only two black and white video cameras and stereoscopic vision algorithms to track its surroundings.
- 2004: The 2004 DARPA Grand Challenge was held on March 13 in the Mojave Desert; it was the first long distance competition for driverless cars in the world. The contest sponsored by DARPA offered a prize of one million US dollars. The challenge consisted of a 240 Km route filled with twists, turns, and obstacles. None of the robot vehicles finished the route. Carnegie Mellon University's Red Team traveled the farthest distance, completing a 11 Km of the course.
- 2005: the 2005 DARPA Grand Challenge was held on October 8. Twenty three of the twenty four competitors traveled farther than the winner of the previous year did, and five challengers completed the full course. The winning team, based on time, was the Stanford Racing Team led by Sebastian Thrun.
- 2007: The 2007 DARPA Urban Challenge took place on November 3. The course was 96 Km long and had to be completed in less than 6 hours. Since this was an urban test, all vehicles were required to obey all traffic regulations while negotiating the course. Six vehicles completed the

course. The two million US dollar winner was Tartan Racing, a joint effort between Carnegie Mellon University and General Motors Corporation.

- 2008-2011: Rio Tinto and Komatsu run a fleet of five fully autonomous 300 ton dump trucks at the Rio Tinto mines in Australia. After running 24 hours a day for over two years the fleet was doubled to ten trucks.

- 2010: The VisLab Intercontinental Autonomous Challenge (VIAC) was run. It consisted of four autonomous vans traveling 13000 Km from Italy to China. A lead van first ran each route with significant human intervention. The following vans used the data collected by the lead van in order to run the course with less human intervention.

- 2010: Google driverless cars travel over 480000 Km on California roads, including toll roads, rural roads, Lombard Street and city streets.

2.2 Research Groups in Autonomous Driving

This section lists some of the most relevant research groups in the scope of AD worldwide. Obviously, the selection of which projects should be included in the following description is not consensual. There are several tenths of projects worldwide that researched on autonomous driving. Because it is not possible to describe them all in detail, we have decided to describe only those projects that have somehow pioneered AD research. Another criteria is based on choosing the projects that have won major robotic competitions or that have achieved significant landmarks in the field.

The Robotics Institute, Carnegie Mellon University

Carnegie Mellon University has pioneered the research in AD with the Navlab project. Navlab [Thorpe *et al.* 1991] is a series of autonomous and semi-autonomous vehicles developed by teams from The Robotics Institute at the School of Computer Science, Carnegie Mellon University. The vehicles were mainly semi-autonomous, though some were fully autonomous and required no human input. Navlab 1 (Fig. 2.1 (*a*)) was built in 1986 using a Chevrolet panel van. The van had 5 racks of computer hardware, including 3 Sun workstations, video hardware, GPS receiver, and a Warp supercomputer. Navlab 2 [Coulter & Mueller 1994] was built in 1990 using a US Army HMMWV. Computer power was uprated for this new vehicle with three Sparc 10 computers, for high level data processing, and two 68000-based computers used for low level control. The Hummer was capable of driving both off or on road. Several additional prototypes followed, adding up to a total of eleven prototypes.

The project became well known in 1990 after the Navlab 5, a Pontiac Trans Sport completed a trek named "No Hands Across America" driving autonomously more than 98% of the way

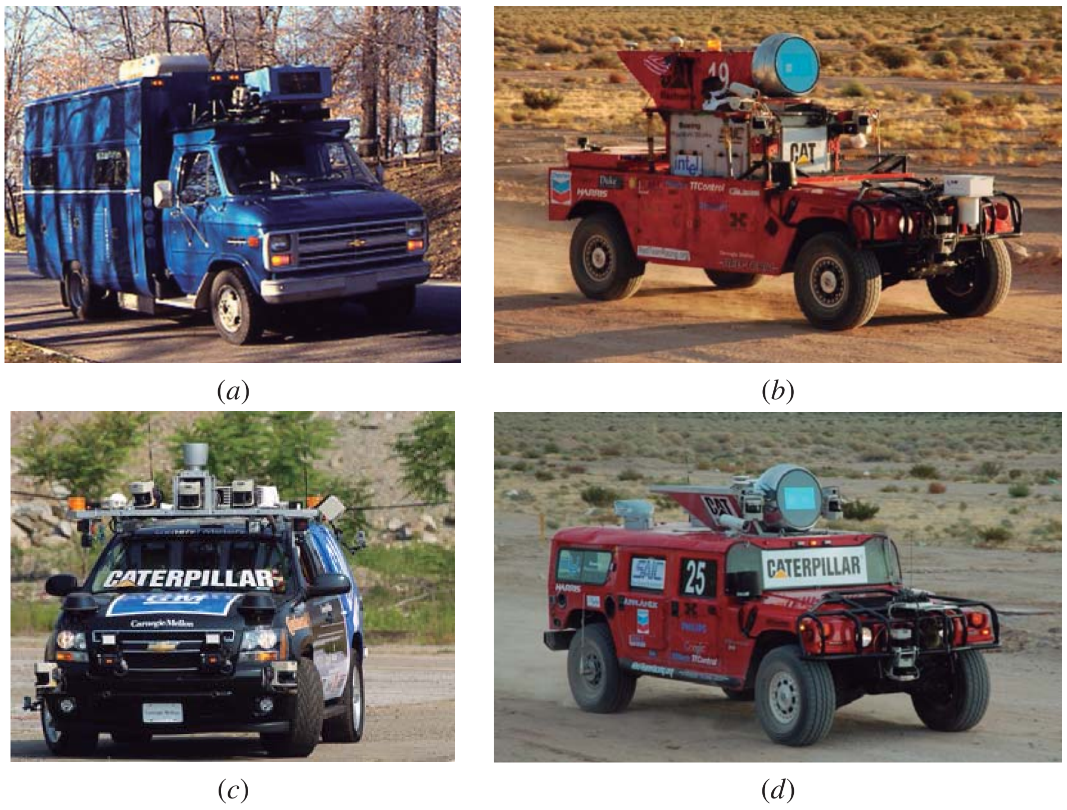


Figure 2.1: Four prototypes from Carnegie Mellon University: (*a*) Navlab I; (*b*) Sandstorm; (*c*) Boss; and (*d*) Highlander.

across the United States from coast-to-coast. The vehicles were equipped with cameras and the focus of the project was on road following [Kanade *et al.* 1986], navigation and obstacle avoidance [Thorpe *et al.* 1988]. Some data sets have been released and are now publicly available [Wang *et al.* 2004]. This project has been thoroughly funded by the United States Military.

In 2004, with the advent of the DARPA Challenge, the Tartan Racing team was created to participate in the competition [Urmson *et al.* 2004]. Two prototypes were developed, Sandstorm Fig. 2.1 (*b*) and Highlander Fig. 2.1 (*d*).

Sandstorm is a heavily modified 1986 HMMWV. It competed in the DARPA Grand Challenge in 2004 and 2005. Although it did not complete the challenge, Sandstorm qualified in first position in the 2004 DARPA Grand Challenge. It traveled the fastest and farthest (12 Km) during the 2004 race before colliding with a sand embankment. The sensors used by Sandstorm in 2004 included three fixed Laser Range Finder (LRF) units, one steerable LRF (in the globe on top), a Radar unit (developed in collaboration with the Duke University Robotics Team), and a pair of cameras for stereo vision. Sandstorm also had a GPS and an inertial navigation system for determining the geographical position; the system was an Applanix POS LV system. Since the 2004 race, Sandstorm has been continuously tested and modified, logging data along hundreds of kilometers. The 2005 version of Sandstorm used six fixed LRF units, the steerable LRF, and short and long range Radars. Sandstorm competed in the 2005 DARPA Grand Challenge. It finished the race in 7 hours and 5 minutes, in the second position out of the five vehicles that completed the course [Urmson *et al.* 2006].

Highlander is a heavily modified 1999 HUMMER H1. It competed in the 2005 DARPA Grand Challenge [Urmson *et al.* 2007]. The sensors used by Highlander include several LRF units, one steerable LRF, GPS and an inertial navigation system (Applanix POS LV). Highlander competed in the 2005 DARPA Grand Challenge, and finished in 7 hours and 14 minutes, placing third out of the five vehicles to complete the course. It was preceded, in second place, by Sandstorm, its sister vehicle.

For the 2007 DARPA Urban Challenge, the Tartan Racing team developed a new prototype called Boss (Fig. 2.1 (*c*)) [Urmson *et al.* 2008]. It was a Chevy Tahoe with over 500000 lines of code designed to autonomously navigate in town and in traffic [Urmson *et al.* 2009]. Boss used perception, planning and behavioral software to reason about traffic and take appropriate actions while proceeding safely to a destination. It was equipped with more than a dozen LRF, cameras and radars to view the world. High level route planning determined the best path through a road network [Likhachev & Ferguson 2009]. Motion planning requires consideration of the static and dynamic obstacles detected by perception [Darms *et al.* 2009], as well as lane and road boundary information, parking lot boundaries, stop lines, speed limits, and similar requirements. Defensive driving skills allowed Boss to avoid crashes [Ferguson *et al.* 2008b] [Ferguson *et al.* 2008c]. According to the Tartan Racing team, Boss is capable of performing the following maneuvers: follow rules of the road; detect and track other vehicles at long ranges; find a spot and park in a parking lot; obey intersection precedence rules; follow vehicles at a safe distance; and react to dynamic conditions like blocked roads or broken down vehicles. Boss has won the 2007 DARPA Urban Challenge [Urmson & Whittaker 2008].

University of Munich

The VaMoRs project, which stands for *Versuchsfahrzeug fuer autonome Mobilitaet und Rechnersehen*, Test Vehicle for Autonomous Mobility and Computer Vision, was also one of the first projects to have successfully achieved autonomous driving experiments. The project initiated in the eighties decade, where the team from the Bundeswehr University of Munich equipped a Mercedes-Benz van with cameras and other sensors (Fig. 2.2 (*a*)). The 5-ton van was re-engineered such that it was possible to control the steering wheel, throttle, and brakes through computer commands based on real-time evaluation of image sequences. Software was written that translated the sensory data into appropriate driving commands. For safety reasons, initial experiments in Bavaria took place on streets without traffic. Since 1986 the Robot Car VaMoRs managed to drive autonomously at speeds up to 96 km/h. One of the challenges in high-speed autonomous driving arises through the rapidly changing visual street scenes. In the eighties and nineties, computers were much slower

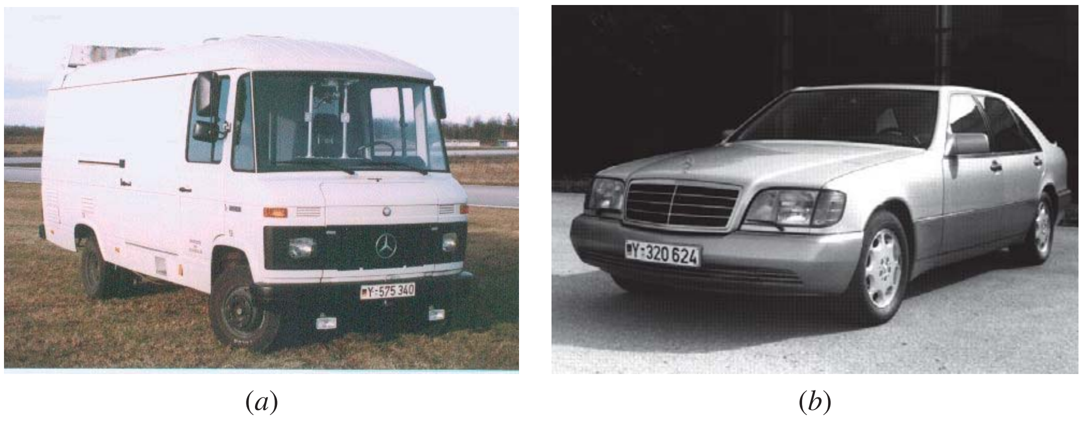


Figure 2.2: Two prototypes from the VaMoRs project: (*a*) VaMoRs; (*b*) VaMP.

than they are today. Therefore, sophisticated computer vision strategies were necessary to react in real time. The team approached the problem through an innovative approach to dynamic vision. Spatio temporal models were used right from the beginning, in what was called a 4D approach [Dickmanns & Mysliwetz 1992]. The system did not require to store previous images but was able to yield estimates of all 3D velocity components of several agents on the road.

The focus of the research was done in vision systems, particularly multi camera active perception systems [Dickmanns 2004] [Dickmanns 2012]. The EMS vision system, Expectation-based, Multi-focal, Saccadic vision, [Gregor *et al.* 2002] included attention control with artificial saccadic movements of the platform carrying the cameras and allowed the system to focus its attention on the most relevant details of the visual input. In 1987, the European car manufacturing industries launched the PROMETHEUS. The initially planned autonomous lateral guidance by buried cables was dropped and substituted by the much more flexible machine vision approach proposed by Dickmanns, and partially encouraged by his successes. Most of the major car companies participated and so did the VaMoRs team, with the Daimler-Benz. Substantial progress was made. In particular, robot cars learned to drive in traffic under various conditions. An accompanying human driver with a "red button" made sure the robot vehicle could not get out of control and become a danger to the public. Since 1992, driving in public traffic was standard.

In 1994, two new prototype autonomous S-Class Mercedes-Benz achieved an important landmark. During the final presentation of the PROMETHEUS project in October 1994 on Autoroute 1 near the airport Charles-de-Gaulle in Paris, with guests onboard, the twin vehicles of Daimler-Benz (VITA-2) and the Bundeswehr University VaMP (Fig. 2.2 (*b*)) drove more than 1000 Km on the three-lane highway in standard heavy traffic at speeds up to 130 km/h [Thomanek & Dickmanns 1995]. Driving in free lanes, convoy driving with distance keeping depending on speed, and lane changes left and right with autonomous passing have been demonstrated.

A second landmark was a 1758 Km trip in the fall of 1995 from Munich in Bavaria to Odense in Denmark to a project meeting and back. Both longitudinal and lateral guidance were performed autonomously by vision. On highways, the robot achieved speeds exceeding 175 km/h, with an average human intervention every 9 km. The longest autonomously driven stretch reached 158 km. The system used black and white video cameras. In total, 95% autonomous driving (considering the distance) was achieved.

In the years that followed, the VaMoRs prototype was used to develop the capabilities needed for driving on networks of minor and unsealed roads and for cross-country driving including avoidance of negative obstacles like ditches [Baten *et al.* 1998].

VisLab, Parma University

At the Dipartimento di Ingegneria dell'Informazione, Università di Parma, the VisLab group started their research activities on the topic of AD in the early nineties. At that time, only a very few laboratories worldwide were investigating the applicability of artificial vision onboard of moving vehicles. Also, no hardware architecture was able to deliver sufficient processing power to run real-time image processing algorithms. To cope with this issue the group developed their own hardware architecture, named PAPRICA, for PArallel PRocessor for Image Checking and Analysis, based on 256 single-bit processing elements working in SIMD fashion, and installed it onboard of a mobile laboratory to develop and test some initial concepts in the field of intelligent vehicles [Bertozzi & Broggi 1999]. These activities were funded by PROMETHEUS, which ended with a demonstration in October 1994, in France. The experience not only provided the ground for new ideas on computer vision techniques, but the typical problems of the automotive environment were also investigated, creating a strong know-how on the application of artificial vision in the real world. In 1996, the group developed a real vehicle prototype named ARGO [Broggi *et al.* 2000] (Fig. 2.3 (*a*)), a Lancia Thema passenger car which was equipped with vision sensors, processing systems, and vehicle actuators. In parallel, they developed the necessary software and hardware that made it able to drive autonomously on standard roads. In 1998, the six day long demonstration of the ARGO capabilities was a milestone *MilleMiglia in Automatico*. In that experience, 94% of a 2000 Km journey were driven autonomously. ARGO was able to follow the road, overtake slower traffic, locate obstacles, and follow the vehicle in front. This was the first experiment of a vehicle running autonomously with vision as the only sensor and a low cost off-the-shelf processing system.

Anchored in these results, VisLab was offered a series of research opportunities by many car manufacturers or automotive suppliers. VisLab has conducted research on applications such as pedestrian detection and night vision to increment road safety, using thermal imagery. In 2004, VisLab partnered with the TerraMax truck corporation to take part in the DARPA Grand Challenge and DARPA Urban Challenge. In 2005, the TerraMax vehicle [Chen *et al.* 2009], a 14 ton truck (Fig. 2.3 (*b*)), was one of only 5 vehicles worldwide to complete the challenge and reach the end of the 220 Km

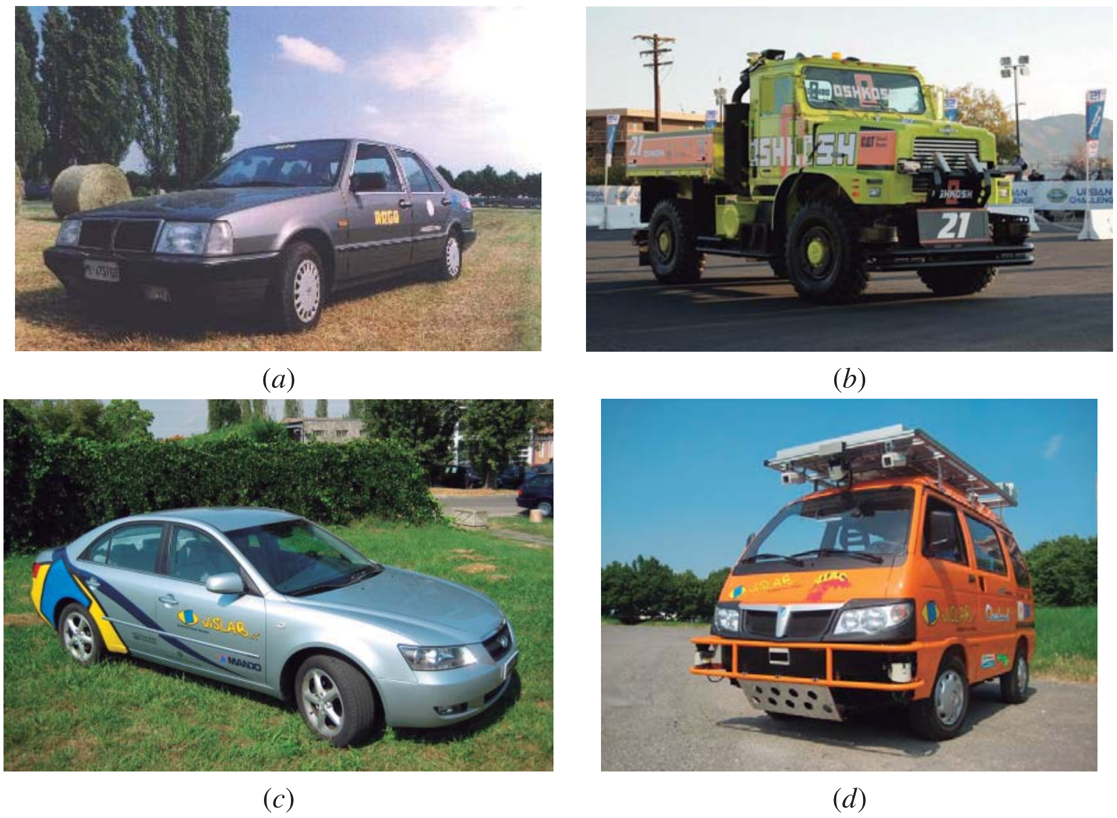


Figure 2.3: Four prototypes from the VisLab group: (*a*) ARGO; (*b*) TerraMax; (*a*) BRAiVE; and (*d*) one of the four vehicles used on the VIAC challenge.

off road race in the Mojave desert. In 2007, TerraMax was qualified for the DARPA Urban Challenge [Broggi *et al.* 2010]. In January 2006, at the US Army base in Yuma, Arizona, VisLab, in partnership with Oshkosh Truck Corporation and Rockwell Collins, demonstrated a new prototype vehicle to the US military: a concept vehicle able to reach a predefined destination, unload, and get back to the starting point with no human intervention.

In 2007 a local non profit organization, *Fondazione Cassa di Risparmio* decided to partially support VisLab activities, which allowed VisLab to acquire a new vehicle, equip it with sensors, and continue the research. The new prototype's name is BRAiVE (Fig. 2.3 (*c*)). It was equipped with 10 cameras, 5 LRF, 16 laser beams, GPS, INS, and complete x-by-wire provided by partner MANDO, from South Korea. BRAiVE flew to China in 2009 for the official presentation at the IEEE Intelligent Vehicles Symposium 2009, where it demonstrated fully automatic features.

In 2008 VisLab was selected by the European Research Council (ERC) to receive an advanced grant (approximately 2 million US dollars) to support research in the following 5 years. As a result, in 2010 the group embarked on driving four vehicles autonomously from Italy to China with no human intervention. This challenge was called VIAC [Bertozzi *et al.* 2011]. Soon after this, VisLab was awarded a second ERC grant to industrialize some of the results obtained and successfully tested on the VIAC vehicles.

Artificial Intelligence Lab, Stanford University

Stanford Racing Team was Stanford University's response to the DARPA Grand Challenge launched in 2004. The Stanford Racing Team started its work in July 2004, supported by a consortium of companies located in the San Francisco Bay Area, as well as car manufacturer Volkswagen.

In 2004, the Stanley robotic vehicle (Fig. 2.4 (*a*)) was operational. Stanley's original frame was a standard European diesel model Volkswagen Touareg provided by Volkswagen for the competition. The Stanford Racing Team choose the Touareg for its drive by wire control system which could be adapted (which was done so by Volkswagen) to be run directly from an onboard computer without the use of actuators or servo motors. It is important to note however, that the steering wheel was driven by an electric motor and the gear shifting accomplished with a hydraulic piston. To navigate, Stanley used five roof mounted LRF units to build a 3D map of the environment, complementing the position sensing GPS system. An internal guidance system utilizing gyroscopes and accelerometers monitored the orientation of the vehicle and also served to supplement GPS and other sensor data. Additional guidance data was provided by a video camera used to observe driving conditions out to eighty meters, beyond the range of the LRFs and to ensure room enough for acceleration. Stanley also had sensors installed in a wheel to act as an odometer in case of loss of GPS signal, such as when driving through a tunnel. To process the sensor data and execute decisions, Stanley was equipped with six low-power 1.6 GHz Intel Pentium M based computers in the trunk, running different versions of the Linux operating system. The software was composed of over 100000 lines of code, run by Stanley to interpret sensor data and execute navigation decisions. Stanley was characterized by a machine learning based approach to obstacle detection. Data from the LRFs was fused with images from the vision system to perform more distant look ahead. If a path of drivable terrain could not be detected for at least 40 meters in front of the vehicle, speed was decreased [Stavens *et al.* 2007] and the LRFs used to locate a safe passage [Dolgov *et al.* 2010]. To correct a common error made by Stanley early in development, the Stanford Racing Team created a log of "human reactions and decisions" and fed the data into a learning algorithm tied to the vehicle's controls [Patel *et al.* 2005]. This action served to greatly reduce Stanley's errors. The computer log of humans driving also made Stanley more accurate in detecting shadows, a problem that had caused many of the vehicle failures in the 2004 DARPA Grand Challenge. In 2005, Stanley competed in the 2005 DARPA Grand Challenge. It won the competition [Montemerlo *et al.* 2006] [Thrun *et al.* 2006b] [Thrun *et al.* 2006a].

In 2007, the Stanford Racing Team competed in the DARPA Urban Challenge with Junior [Montemerlo *et al.* 2008] (Fig. 2.4 (*b*)). Junior was a modified 2006 Passat wagon, equipped with a four-cylinder turbo diesel injection engine. The 140 horsepower vehicle was equipped with a limited torque steering motor, an electronic brake booster, electronic throttle, gear shifter, parking brake,

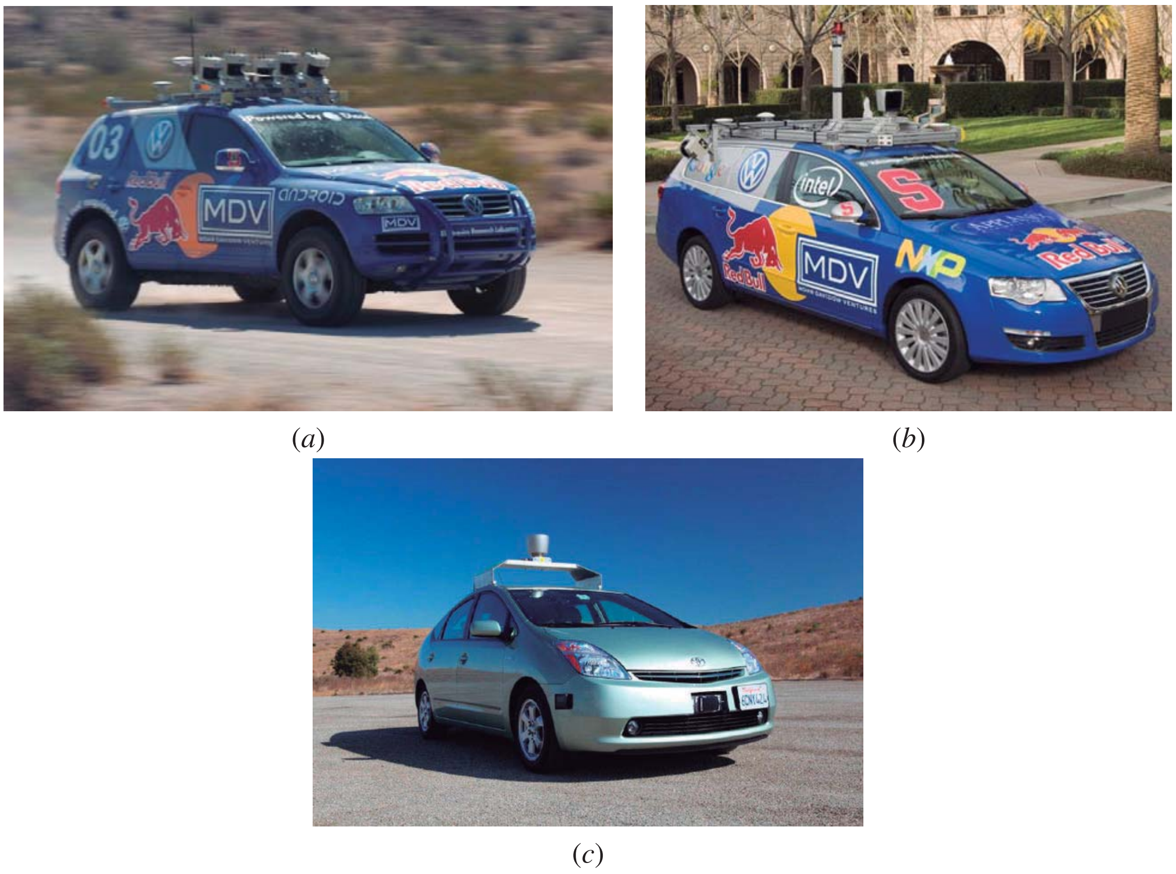


Figure 2.4: Three prototypes from the Stanford University: (*a*) Stanley; (*b*) Junior; and (*c*) Google Driverless Car.

and turn signals. A custom interface board was developed to provide computer control over each of these vehicle elements. The engine provided electric power to Junior's computing system through a high-current prototype alternator, supported by a battery-backed electronically controlled power system. For development purposes, the cabin is equipped with switches that enable a human driver to switch on / off various electronic interface components. For example, a human developer could choose the computer to control the steering wheel and turn signals while retaining manual control over the throttle and the vehicle brakes. For inertial navigation, an Applanix POS LV 420 system provides real-time integration of multiple dual-frequency GPS receivers, including a GPS azimuth heading measurement subsystem, a high performance inertial measurement unit, wheel hodometry via a distance measurement unit, and the Omnistar satellite-based Virtual Base Station service. The real-time position and orientation errors of this system were typically below 100 cm and 0.1 deg, respectively. Two side-facing and a forward-pointed LRF laser sensors provide measurements of the adjacent three-dimensional road structure and infrared reflectivity measurements of the road surface for lane marking detection. In addition, a 3D laser range finder provided a large number of range measurements around the vehicle.

Junior's software architecture was designed as a data driven pipeline in which individual modules process information asynchronously. The same software architecture was employed successfully by Junior's predecessor Stanley in the 2005 challenge. Each module communicates with other modules via an anonymous publish subscribe message passing protocol, based Inter Process Communications (IPC) [Simmons & Apfelbaum 1998]. Modules subscribe to message streams from other modules, which are then sent asynchronously. The result of the computation of a module may then be published to other modules. In this way, each module is processing data at all times, acting as a pipeline. The time delay between entry of sensor data into the pipeline to the effect on the vehicle's actuators was approximately 300 milliseconds. Junior finished second in the DARPA Urban challenge.

These achievements led to Stanford's cooperation with Google, and ultimately development of Google Driverless Car (Fig. 2.4 (*c*)). Google Driverless Car is a project by Google that involves developing technology for driverless cars.

Other Institutions

As discussed before, it is a very difficult task to list all the research groups that are working or have worked in AD. Boosted by the support of the very strong German automotive industry, relevant work has also been carried out by several other German Universities, namely the Artificial Intelligence Group of the Freie University of Berlin. These have carried out a demonstration where an autonomous vehicle travelled the city of Berlin [Wang *et al.* 2011]. The FZI lab from Karlsruhe University is developing what they call cognitive cars, which can execute risk assessments in complex traffic situations [Kumpakeaw & Dillmann 2007] [Vacek *et al.* 2007].

In France, the National Institute for Research in Computer Science and Control has also addressed several problems connected with autonomous driving for some years now [Diosi *et al.* 2011]. In the United Kingdom, the University of Oxford also has their own robotic car prototype [Mathibela *et al.* 2012], while in Spain the University of Alcalá, in Madrid, has done several demonstrations with an autonomous vehicle [Sotelo *et al.* 2004].

In the United States, several other institutions have also researched on autonomous driving: the Massachusetts Institute of Technology has also participated in the DARPA Challenge competitions, finishing fourth in the 2007 edition with the Talos vehicle [Huang *et al.* 2011]. This vehicle will be described in detail in chapters 7, since the data sets provided by the vehicle will be used for validating some of the proposed algorithms. The team from Cornell University has also competed in the 2007 Urban Challenge, using novel probabilistic inference algorithms. These technologies were integrated into a modified stock SUV chassis for the Urban Challenge. Several other participations in the DARPA Challenge included a team from University of Central Florida, Virginia Tech, Caltech, and many others.

The first, full scale autonomous / semi autonomous vehicle operational in Portugal was developed at the Laboratory of Automation and Robotics of the University of Aveiro. The *AtlasCar* is a modified 1995 Ford Escort, equipped with several LRF and cameras, and adapted to have drive by wire capabilities. Since a portion of the development of this vehicle was included in the current Ph.D. work, the vehicle will be described in detail in chapter 3.

2.3 Autonomous Driving Competitions

This section is devoted to describe some of the most important autonomous driving competitions featuring full scale vehicles. The reason for the highlight of this topic is that robotic competitions have largely contrived for the development of the field of AD. While traditional project application and selection funding mechanisms also contribute for the advances in the field, the fact is that competitions such as the DARPA Grand Challenges have boosted the effort of the research community, both in quantity and quality, like never before. Because of this, the following lines describe some of the most significant robotic competitions designed for full scale autonomous vehicles. Additionally, the Autonomous Driving Competition (ADC) of the Portuguese Robotics Open (PRO) will be described, since that some of the following chapters describe algorithms designed to solve the challenges posed by that competition.

The DARPA Challenge was the first long distance competition for driverless cars in the world. In the past, research efforts in the field of driverless cars followed a more traditional commercial or academic approach with individual funding to research institutions. The United States Congress authorized DARPA to offer a prize money of one million US dollars for the first Grand Challenge to facilitate robotic development, with the ultimate goal of making one third of ground military forces autonomous by 2015. Following the 2004 event, the director of DARPA, announced that the total prize money had been increased to two million for the next event. The competition was open to teams and organizations from around the world, although with the limitation of having at least one US citizen on the roster. Teams have participated from high schools, universities, businesses and other organizations. More than 100 teams registered in the first year, bringing a wide variety of technological skills to the race. In the second year, 195 teams from 36 US states and 4 foreign countries entered the race. In 2005, the challenge was composed of three narrow tunnels and several sharp left and right turns. The race concluded through Beer Bottle Pass, a mountain pass with a cliff on one side and a rock wall on the other. The vehicles were given a set of GPS way points, just a few hours before the race start. The robots were expected to pass all way points, transversing obstacles and selecting the best path between way points. Table 2.1 lists the teams that were able to complete the challenge in 2005.

The DARPA Urban Challenge competition took place on November 3, 2007 at the site of the now decommissioned George Air Force Base, in California. The course involved a 96 Km urban

Table 2.1: Results of the DARPA Grand Challenge, 2005 edition [Wikipedia 2004]. The Table only shows the teams that completed the challenge.

Vehicle	Team Name	Team Home	Time Taken (h:m)	Result
Stanley	Stanford Racing	Stanford University, California	6:54	First place
Sandstorm	Red Team	Carnegie Mellon University, Pittsburgh	7:05	Second place
Highlander	Red Team	Carnegie Mellon University, Pittsburgh	7:14	Third place
Kat-5	Team Gray	The Gray Insurance Company, Louisiana	7:30	Fourth place
TerraMax	Team TerraMax	Oshkosh Truck Corporation, Wisconsin	12:51	Fifth place

Table 2.2: Results of the DARPA Urban Challenge, 2007 [Wikipedia 2004]. The Table only shows the teams that completed the challenge.

Vehicle	Team Name	Team Home	Time Taken (h:m)	Result
Boss	Tartan Racing	Carnegie Mellon University, Pittsburgh	4:10:20	1st Place
Junior	Stanford Racing	Stanford University, California	4:29:28	2nd Place
VictorTango	Odin	Virginia Tech, Virginia	4:36:38	3rd Place
Talos	MIT	MIT, Massachusetts	6:00:00	4th Place
Little Ben	Ben Franklin	University of Pennsylvania, Philadelphia	(1)	(2)
Skynet	Cornell	Cornell University, New York	(1)	(2)

(1) No official time;
(2) One of six teams to finish the course;

area course, to be completed in less than 6 hours. Rules included obeying all traffic regulations while negotiating with other traffic and obstacles and merging into traffic. Unlike previous challenges, the 2007 Urban Challenge organizers divided competitors into two groups. All teams from each group travelled the race circuit simultaneously. Teams were given maps sparsely charting the way points that defined the competition courses. At least one team, Tartan Racing, enhanced the maps through the insertion of additional extrapolated way points for improved navigation. Table 2.2 lists the teams that were able to complete the challenge in 2007. Figure 2.5 shows a satellite view of the scenario in which the DARPA Urban Challenge competition took place.

European Land Robot Trial (ELROB) is a European event which demonstrates the abilities of modern robots (www.elrob.org). The ELROB is not designed as a competition, like the US DARPA Grand Challenge, but a pure demonstration of what European robotics is able to achieve today. The scenarios are designed to simulate real world missions, be it military or civilian ones. There are no artificial constraints set to this scenarios to ease the task for the robots like for example very visible road markings. The first ELROB was organized in 2006 by the German Federal Armed Forces and took place on the infantry training area near Hammelburg. The goal of the first trial was to boost the development of unmanned ground vehicles that could be used in military missions on short notice. The ELROB is setup as an annual event and alternates between a military and a civilian focus each year. European Robotics and the NATO Research Task Group *Military Applications for Multi-Robot*

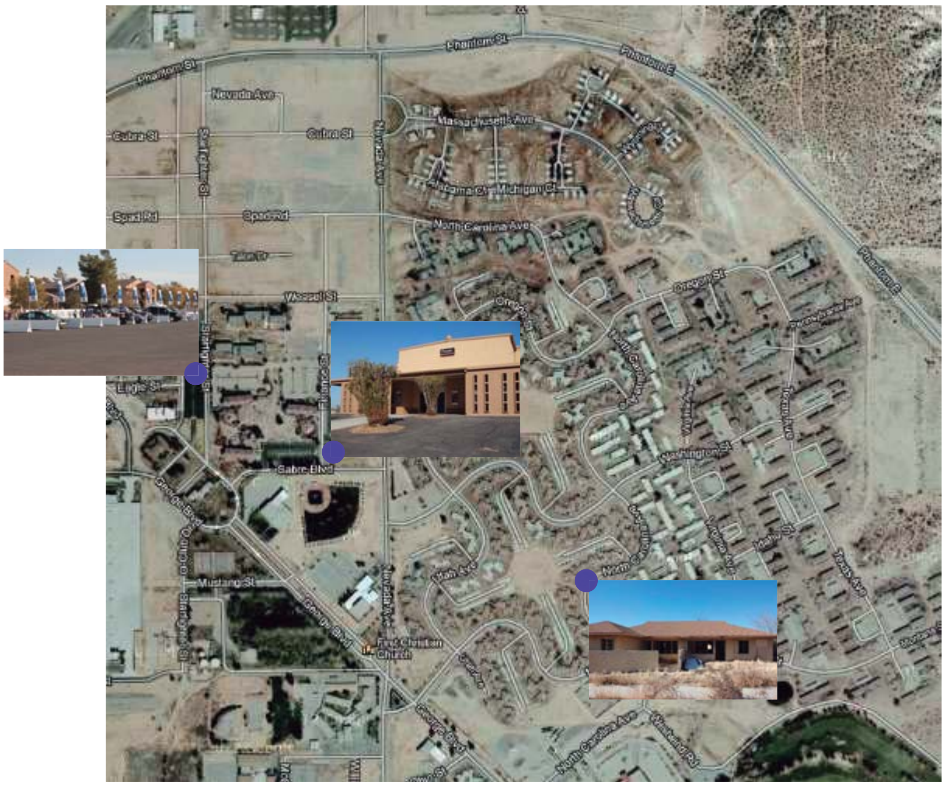
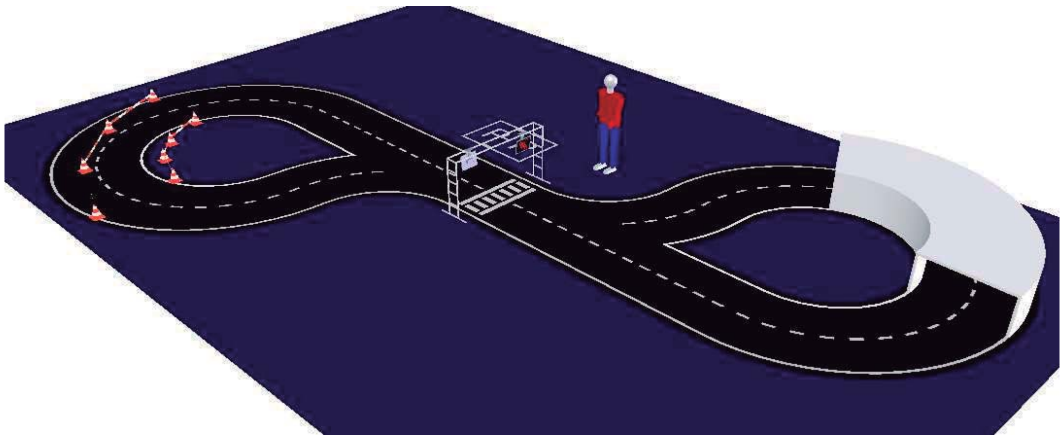
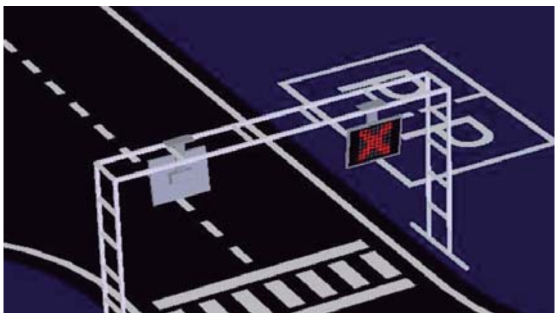


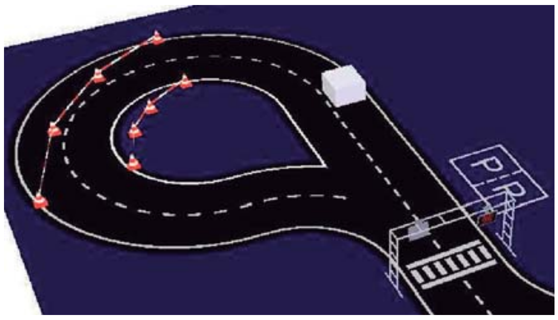
Figure 2.5: A satellite view of the Darpa Urban Challenge competition area, the George Air Force Base in Victorville, California. Pictures of some locations in the scenario are also displayed. The image on the left shows the race start and finish site.

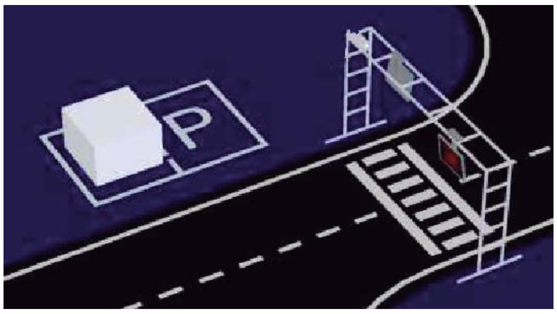
Systems came up with the idea for ELROB in the year 2004. European Robotics aims to bridge the gap between defence and security users, industry and research in the field of ground robotics. Only teams from Europe are allowed. Both teams of commercial and academic backgrounds have participated. In the previous editions of the trial, the participating teams are mostly composed of teams from Germany. The competition started in 2006, and is now holding its eight event in September 2013, in Germany.

The Portuguese Robotics Open (PRO) is a robotic competitions event that takes plane every year in Portugal. One of the major competitions within the event is the Autonomous Driving Competition (ADC). The ADC is a race like challenge composed of three rounds. Robots must drive through the road-like scenario (Fig. 2.6 shows schematics, Fig. 2.7 shows photographs) as fast as possible. However, they must do so abiding to several rules. When robots fail to comply with the rules, they are given a penalty time corresponding to the infraction. The final score is the time taken to transverse the course added to the total penalty time. The three rounds of the competition are organized so that the complexity of the challenge increases from round to round. The first round is a speed trial, where robots must travel the course without leaving the road and stop at the cross area after completing two laps. In the second round, robots must turn left or right at the intersection based on the commands given by the traffic lights panels (Fig. 2.6 (*b*)). Additionally, an obstacle in placed on the track on an unknown position (Fig. 2.6 (*c*)). Having completed to entire course, the robot must stop at the parking spot (Fig. 2.6 (*d*)). The challenge may also contain traffic signs (Fig. 2.6 (*e*)) that indicate, for example, that the robot should drive in a particular lane. In the final round, a road maintenance area (Fig. 2.6 (*c*)) and a tunnel (Fig. 2.6 (*a*)) are added to the scenario. Throughout the years of history of the competition very few teams have successfully completed all three rounds.


(*a*)


(*b*)


(*c*)



(*d*)

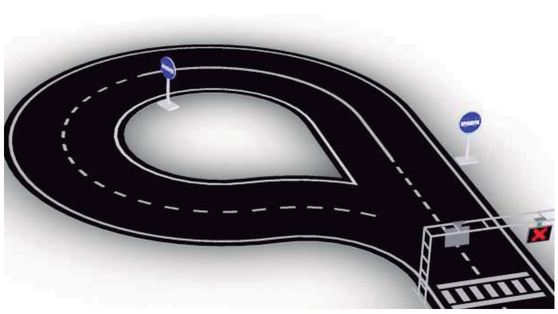
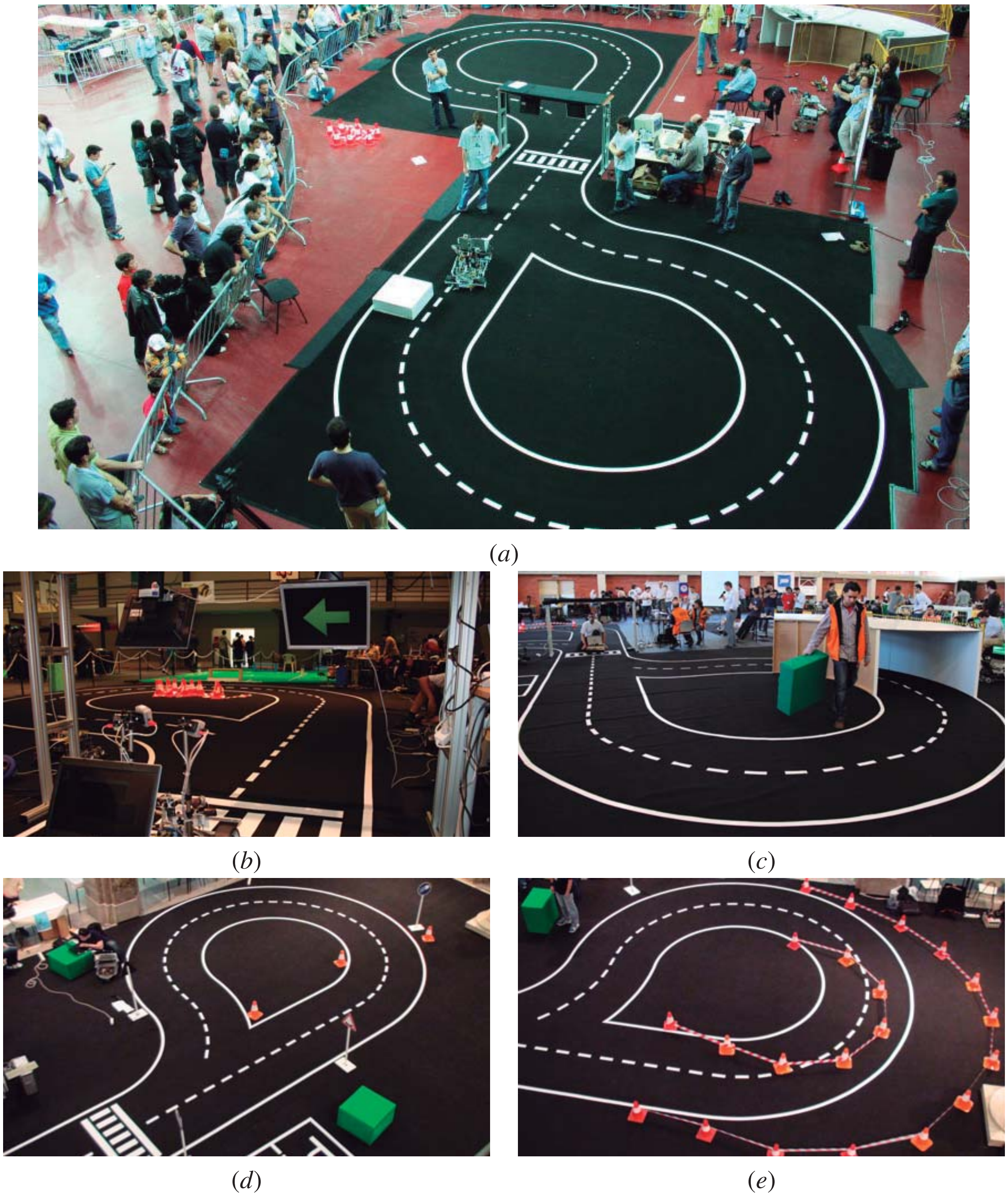

(*e*)

Figure 2.6: Schematics of the ADC: (*a*) the full scenario; (*b*) traffic lights; (*c*) road maintenance and obstacle; (*d*) cross walk and parking zone; (*e*) traffic signs.



(*a*) (*b*) (*c*) (*d*) (*e*)

Figure 2.7: Pictures of the ADC: (*a*) the scenario at PRO 2006; (*b*) traffic lights; (*c*) tunnel; (*d*) obstacles and traffic signs; (*e*) road maintenance area.

2.4 The Near Future of Autonomous Driving

Previous sections have shown that there is a vast number of research institutions working on AD and ADAS systems. Associated with them, and also doing independent research, almost all car manufacturer has also been addressing the problem for several years now. Furthermore, several automotive component manufacturers like Bosch or Omrom have also been funding their own research. The question is why, after several decades, thousands of dedicated researchers and significant funding, are autonomous vehicles not a reality in nowadays roads. Yes, there have been significant advances in the field. From the milestones that have been achieved, it seems plausible to assume that a fully autonomous vehicle should be in grasp of our scientific and technological capabilities within the next decade, if not at present time. However, up to this moment, the fact is that this is still not a reality. The most recent vehicles are equipped with several ADAS systems, but there is no commercially available vehicle with full AD capabilities.

Although AD and ADAS are sometimes presented as separate fields of research, in fact they are tightly coupled. Typically, in ADAS, a human driver interference or cooperation with the systems is considered. In the other case, AD systems do not count on the driver to assist or overrun decisions made by the autonomous systems or subsystems. However, in both cases the core objective is to develop algorithms that make a computer system *understand* a road traffic scenario. It is only in the applications scope that these two fields of research vary: while ADAS uses these algorithms to assist or help a human driver in his task, in AD these are a sub group of a larger computer program that makes use of the information they produce for driving a car without human intervention. ADAS systems are designed to operate in different ways, according to the state of the vehicle and the surroundings. Three layers of operation or system status may be defined, each signaled with a color:

- Green status: assigned when the system does not detect or predicts any dangerous situation;
- Yellow Status: occurs whenever the system computes a dangerous situation that requires a mandatory action from the driver within the next few seconds. In this case, an alarm should be conveyed to the driver;
- Red Status: in this case, the dangerous situation is sure to occur. The system is aware that, given the dynamic capabilities of the vehicle, a crash is eminent. In such cases the systems may take control of the vehicle to try to mitigate as much as possible the consequences of a crash.

As an example, suppose that a vehicle is travelling on a highway. The road is a straight lane, and the vehicle is equipped with a range measuring sensor that provides a real time estimate of the position and velocity of the obstacles in front of it. The driver is driving at a constant speed. The road ahead is blocked by another vehicle which is stopped due to a malfunction. At a large distance, the driver may continue at present speed. Hence, the system is in green status. At a certain point, the system realizes that the driver should have began to decrease speed, in order to approach the stopped car at a low speed. At this point, the system shifts to the yellow status, and a warning can be granted to the driver, asking him to decrease speed. If, regardless of the warning, the driver does not decrease the vehicle's speed, the situation will come to a point where the crash between both vehicles is unavoidable because the maximum braking deceleration is not sufficient to stop the vehicle before the crash. In this case, the system enters a red status, and takes control of the brake, decelerating the vehicle as much as it can, in order to mitigate the consequences of the accident.

Obviously, unlike in AD, in the field of ADAS, there is the additional problem of defining the thresholds for these system status. Also, there is the problem of defining which are the adequate actions for a given situation. Suppose the example described above: during the yellow status, should the vehicle warn the driver of an eminent danger situation. And, if so, how should the warning be conveyed to the driver. There are several possibilities, from a sound warning to a flash light in the instrument pannel, etc. Such a system must be very carefully devised, because there is the chance that the driver gets further distracted by the warning itself and, because of it, is unable to execute the necessary actions to avoid the red status, and therefore, the crash. It is in this area of research, the human vehicle interface, that ADAS most distinguishes itself from AD. However, the point that we wanted to make is that the core technologies are the same for both fields. Imagine that an ADAS system, that constantly monitors the driver's actions, is so efficient that it always can devise the appropriate driving actions for all situations. In such a case, the replacement of the human driver by a fully autonomous system would be a very easy task.

Another question that often is raised in the AD and ADAS communities is that any AD or ADAS system must be fault free, before it can be applied to commercial vehicles. Ultimately, no system is completely fault free. Yet the general opinion is that those that could be installed on autonomous vehicles should have such properties. Obviously, commercial systems should have a very low failure probability, and special efforts should be endeavored to avoid faults as much as possible. But in our opinion, the fault probability threshold necessary to consider their application to commercial vehicles is not of zero failure probability, but rather a smaller failure probability than that of the average of human drivers. Of course that a death caused by a failure of an autonomous system would have a very large impact on the media and in the general public opinion and this is a concern to the researchers in the field. However, from a statistical standpoint, if autonomous systems would cause a smaller amount of accidents, they would in fact be saving lives.

Given all these considerations, the question is why have not the automotive manufacturers yet introduced autonomous driving systems in commercial vehicles. There are several factors that have delayed or even stalled the process. Some of them are listed in the following lines.

Sensors

Grasping a complete or at least sufficient amount of data from the road scenario and from the agents involved requires a very large set of sensors. Recent AD robotic prototypes display a vast array of radars, LRF, sonars, monocular visible and infrared cameras, stereo cameras, 3D lasers, time of flight cameras, GPS, inertial measurement units and other sensors. In general, all these sensors have undergone significant advances in recent years, making them more effective, precise and robust. This has certainly been a factor that delayed the development of AD capabilities. One symptomatic example is the Velodyne LIDAR [Velodyne 2012]. The Velodyne is a 3D LRF that produces 1.3 million range measurements per second, all around the vehicle. Apart from the notable exceptions from the DARPA challenges of 2004 and 2005 (although these competitions did not involve coping with urban traffic) and the VisLab group, the fact is that Velodyne seems to be a standard in current autonomous vehicles. It should be noticed that this sensor was developed relatively recently, in 2007.

Hardware

As discussed, AD applications require that the vehicles are equipped with vast amounts of sensors. As a consequence, the amount of data received is also very large. Also, because the road scenarios are highly dynamic environments the AD systems have real time demands. The conjunction of both these factors led to the necessity of having very powerful computers onboard the vehicles, and to the fact that, for many years, the available hardware capabilities were insufficient to comply with these demands.

Recognition Systems

In order to *understand* the traffic, autonomous systems employ several algorithms for recognizing the agents that move about the road scene. In the field of pattern recognition, many algorithms have been proposed throughout the years to perform visual detection of pedestrians, lane markings, other vehicles, traffic signals, etc. There have been significant advances, and the systems now have very high hit rates (number of detected entities over the total number of entities), sometimes of over 99%. The problem has been related with the number of false positives, that is, the number of falsely detected entities over the total number of detection attempts. By today's standards, values under 1% are considered very interesting. However, within the context of AD, these performances could be still not sufficient. Suppose a pedestrian detection system that has a 0.1% false alarm rate. Whenever a pedestrian is detected, the system warns the driver of the danger associated with a nearby pedestrian. With a false alarm rate of 0.1%, the system is expected to falsely detect a pedestrian every 1000 detection attempts. If we consider that the system is processing images streaming from a camera at 30Hz, this means that a false detection would occur every 35 seconds. No driver would buy a system that would wrongly warn him every 30 seconds. In conclusion, the current state of the art on visual recognition of the road entities, in particular the level of false alarms, is still orders of magnitude away from what would be necessary for their application to autonomous systems. These problems have been partially solved by the inclusion of additional sensors like LRF, or the conditional display of the warning by monitoring the drivers gaze, for example, the system issues warnings only when a pedestrian is detected and the driver's gaze is not aimed in that direction.

Legal Issues

One thing that has certainly hampered the release of commercial vehicles with AD capabilities is the fact that it is forbidden by law in the entire European Union and most of the United States. In June 2011, the United States state of Nevada passed a law permitting the operation of driverless cars in Nevada. Google has been lobbying for driverless car laws. The Nevada law went into effect on March 1, 2012, and the Nevada Department of Motor Vehicles issued the first license for a self driven car in May 2012. The license was issued to a Toyota Prius modified with Google's experimental driverless technology. In August 2012, the team announced that they have completed over 480000 km autonomous driving with an accident free record. As of September 2012, three states have passed laws permitting driverless cars: Nevada, Florida and California. In Europe, with the lobbying from the powerful automotive industries, it is very possible that several countries will follow.

Another problem that is difficult to handle is that of legal responsibility, in particular for dealing with insurance contracts. In the case of an accident caused by an AD system, who is juridically accountable: the human passenger (that was not driving), the vehicle owner, the system manufacturer, the car manufacturer. It is a delicate issue that still remains to be solved.

2.5 Conclusions

This chapter presented several AD concepts that were proposed throughout the last three decades. Although at first sight it seems that there are several AD projects that are in conditions of developing an autonomous vehicle for a commercial application, the fact is that most experiences described before have been achieved under special conditions and unrealistic constraints. Many involve trafficless or controlled traffic setups, and others have a human monitoring the system and intervening when necessary. A commercial solution for an everyday autonomous driving vehicle is very near, but there are still some issues mostly related with the robustness of the systems that need to be addressed.

Chapter 3

Robot Prototypes

This chapter presents three robotic prototypes developed at the Laboratory of Automation and Robotics of the University of Aveiro. Section 3.2 describes the *Atlas2000* prototype; section 3.3 describes the *AtlasMV*, and finally, section 3.4 describes the *AtlasCar*. Additionally, some of the algorithms that were developed specifically for the competitions are presented in section 3.5. Section 3.6 presents the results the robots have obtained in the competitions and section 3.7 draws some conclusions.

3.1 Introduction

During the course of the work, several perception based algorithms were developed. They were implemented on the robots for assessing their performance in realistic scenarios. In this way, the prototypes proved to be very valuable test beds for assessing the efficiency of the proposed algorithms. Over the past years, these prototypes have also participated in several robotic competitions integrated in the Portuguese Robotics Open (PRO) [PRO 2012]. For each prototype, a description of the mechanical and electronic components is given. Additionally, some of the algorithms that were developed for the robots are briefly discussed.

3.2 *Atlas2000*

The *Atlas2000* robot was designed and built in 2005 [Oliveira *et al.* 2005]. At that time, it was the first robot presented at the Autonomous Driving Competition (ADC) to have a structure similar to that of a car, with two rear wheels providing traction and two front wheels for steering. The platform was adapted from a one to four scale model. One of the advantages of the robot were the stability provided by the car like structure, as well as a new redesigned Linux based software. On the side of disadvantages, the *Atlas2000* has mostly its size. It is often the largest vehicle in the ADC, which

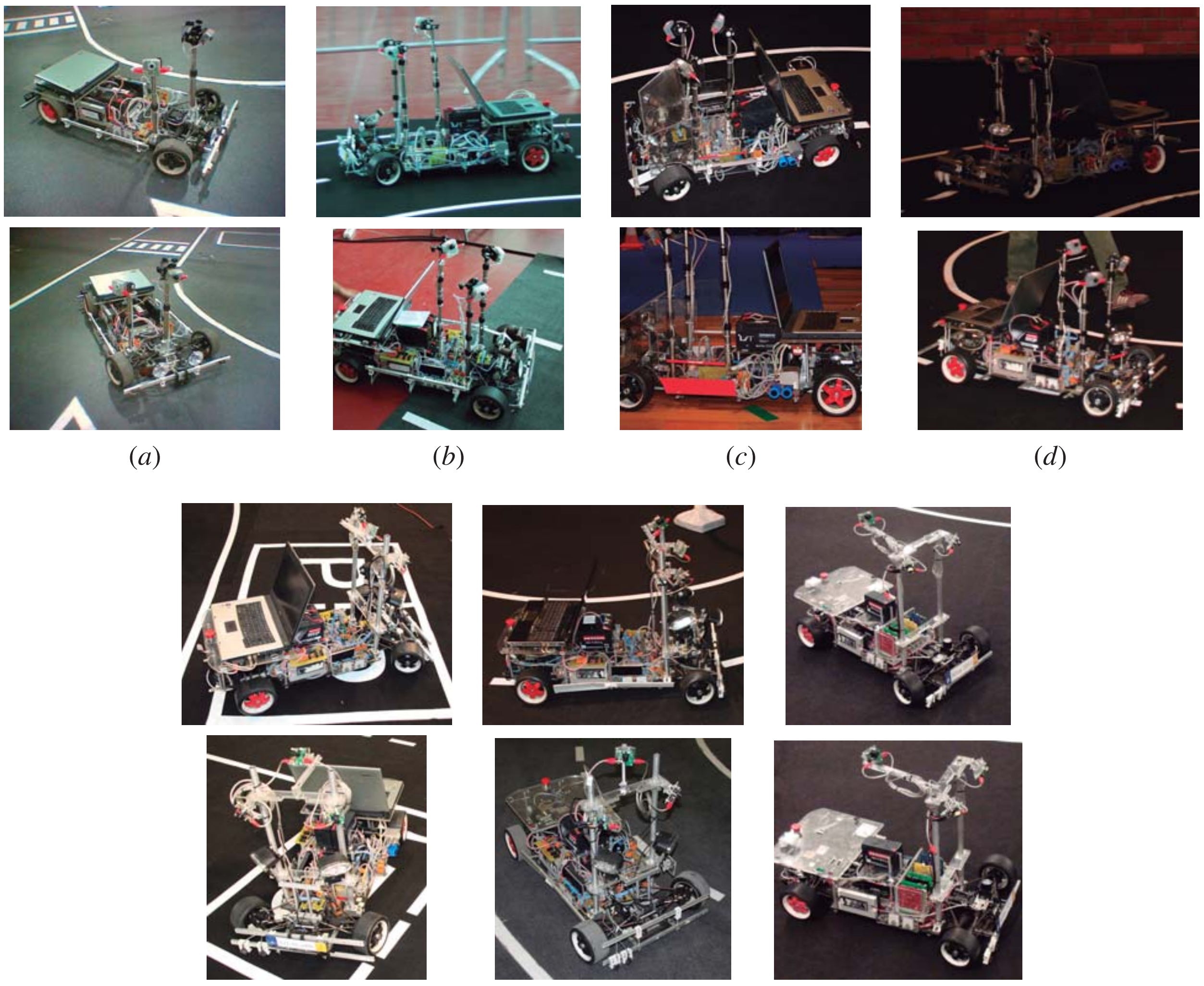


Figure 3.1: The *Atlas2000* autonomous robot. The robot participated in the PRO of 2005 (*a*), 2006 (*b*), 2007 (*c*), 2008 (*d*), 2009 (*e*), 2010 (*f*), 2011 (*g*).

makes it harder to execute obstacle avoidance maneuvers or road maintenance area navigation. Nevertheless, throughout its long seven years record of participations in the ADC, the *Atlas2000* achieved notable positions: it was first in 2006, 2007, 2008 and 2011. Also, this robot never finished worse than second place. Figure 3.1 shows the various versions of the *Atlas2000* throughout the years it participated in the competitions. The next sections will briefly discuss the mechanical, electronic, and sensorial components of the *Atlas2000*.

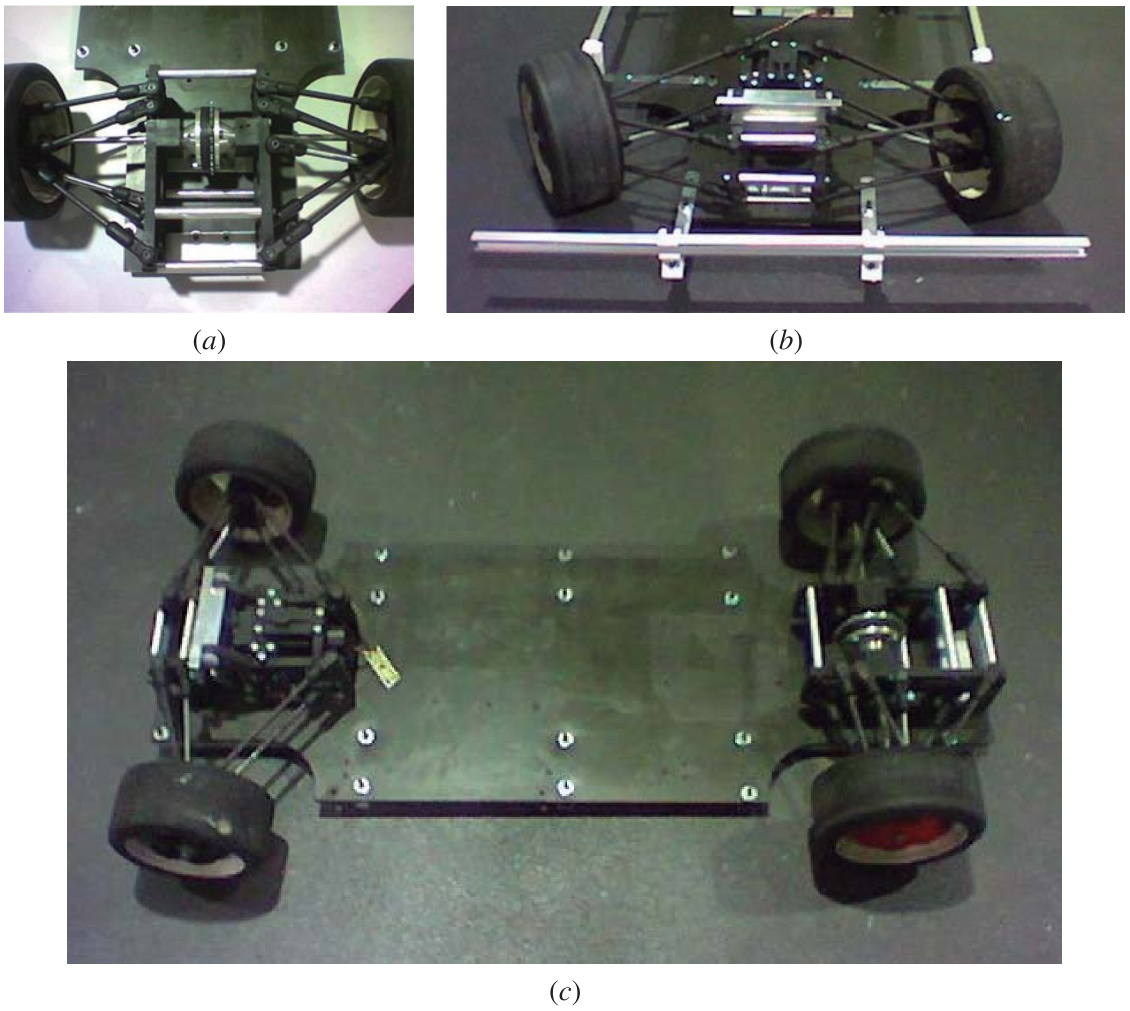


Figure 3.2: An overview of *Atlas2000*: (*a*) The basic structural platform; (*b*) traction system; (*c*) steering system.

3.2.1 Mechanical Components

As said before, the *Atlas2000*'s basic structure was adapted from a one to four scale model. It is shown in Fig. 3.2 (*c*). A 300W DC motor is used for traction coupled to a mechanical differential (Fig. 3.2 (*a*)). The Ackerman like steering is powered by a DC servo motor (Fig. 3.2 (*b*)). A braking system is installed near the mechanical differential in order to ensure sufficient braking capabilities (Fig. 3.3). The system is composed of a brake disk and a brake piston, which is actuated by a solenoid. Hence, unlike in the *AtlasMV* in which the brake command has several command positions, the system of the *Atlas2000* can only brake or not brake. The mechanical structure suffered only minor changes throughout the years, since it proved to be quite robust and effective.

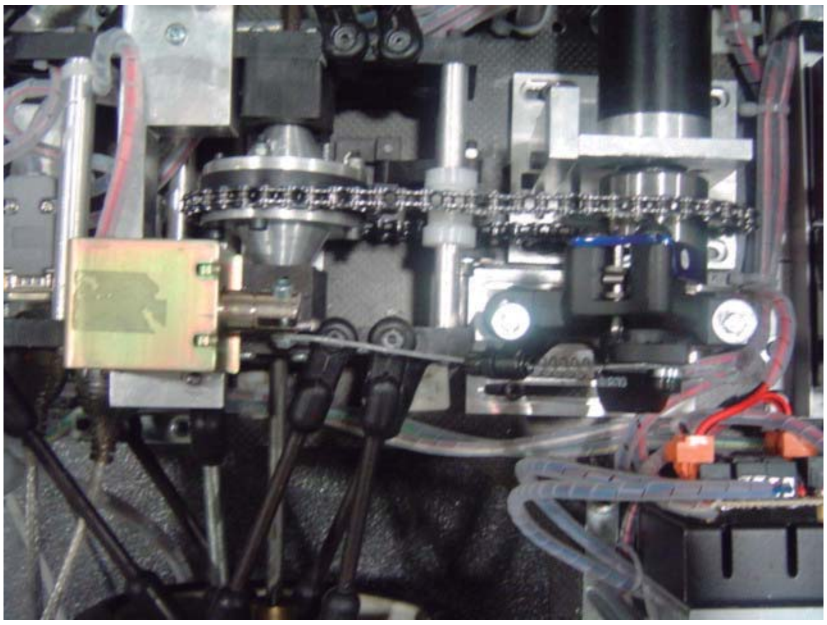


Figure 3.3: The braking mechanism of the *Atlas2000*.

3.2.2 Electronic Components

For the interface with the traction and steering motors, a custom designed micro controller board is used. It connects to the computer using RS232 communication. The digital input and outputs are interfaced using an additional micro controller board with a second RS232 connection to the computer. The digital outputs are responsible for the brake and lights activation. Digital inputs, in other hand, receive readings from digital auxiliary sensors. Figure 3.4 shows the electronics components of the robot.

3.2.3 Sensors

The initial version of the *Atlas2000* robot used two cameras. One was directed towards the front of the robot so that it could capture a view of the road. The second camera was pointed at the traffic lights. This early version of the robot's sensors is shown in Fig. 3.5 (*a*).

In 2006, the ADC was changed to a scenario where the road contained two lanes instead of the previous single lane. Since the road became much wider, a single camera was not sufficient (even when equipped with wide angle lenses) to grab an image of the entire track. Because of this, a second camera was added to the road viewing system. The two road facing cameras provided a complete view of the entire road. Figure 3.5 (*b*) shows the structure containing the two road facing cameras. In 2011, driven by the successful Laser Range Finder (LRF) approach of the *AtlasMV*, the *Atlas2000* was equipped with a LRF. This equipment is also shown in Fig. 3.5 (*b*).

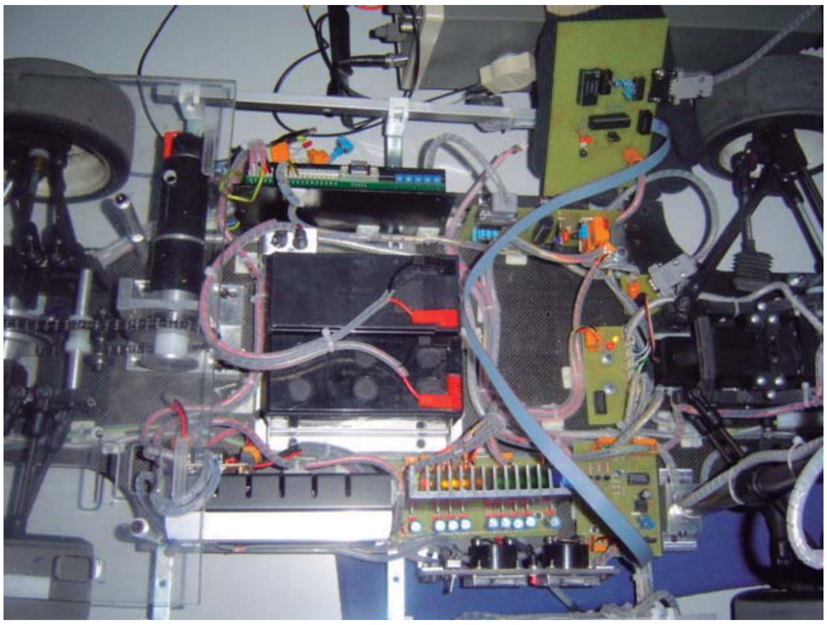
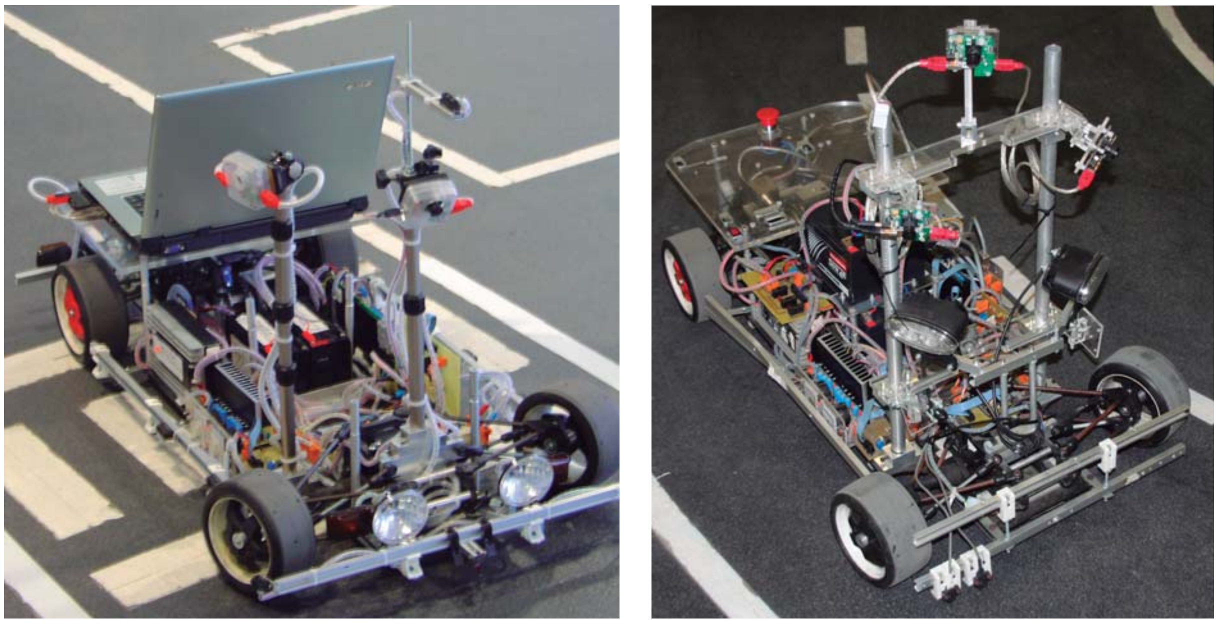


Figure 3.4: The electronic components of the *Atlas2000*.



(*a*) (*b*)

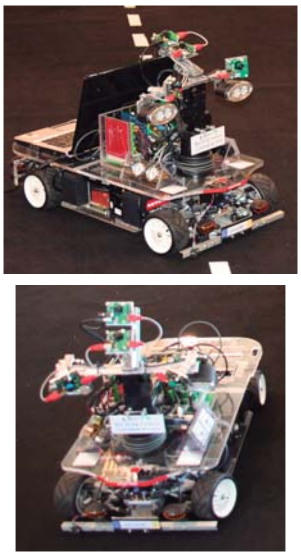
Figure 3.5: The sensors onboard the *Atlas2000*: (*a*) 2005; (*b*) 2011.

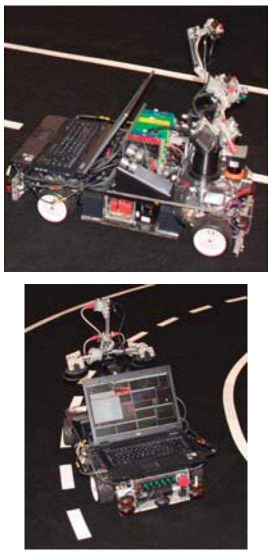
3.3 *AtlasMV*

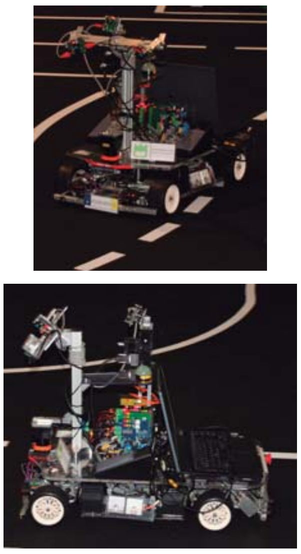
The *AtlasMV* is a small scale robotic prototype designed to compete in the PRO. Its project and design had the objective of building a smaller are more maneuverable robot than the *Atlas2000*, since the latest had significant disadvantages in the ADC due to its size. The *AtlasMV* showed significant improvements both in the mechanical as well as in the electronic components with respect to its predecessor. Its smaller design and improved steering mechanism led to a larger maximum turn radius, and the modular electronics design proved more robust than previous solutions.

The design and development of the *AtlasMV* began in 2007, and the robot had its first appearance in the PRO of 2008, where it achieved the third overall place in the competition. Since then, the *AtlasMV* proved to be one of the most advanced robots in the competition, winning several editions of the ADC and bringing many technological innovations to the competition. Some examples are the multi camera based inverse perspective mapping framework which significantly improved the robustness of the perception algorithms, new lane marker detection algorithms running on the top view images, a laser range finder to detect obstacles and, perhaps most importantly, a new software architecture based on Carnegie Mellon Robot Navigation Toolkit (CARMEN). The robot participated in the ADC editions 2008, 2009, 2010 and 2011. Section 3.6 will provide a detailed description of the results obtained by the robot. Throughout the years, the robot underwent several transformations. In Fig. 3.6 it is possible to see pictures of the robot in each of the versions presented for competition.

Sections 3.3.1 and 3.3.2 will describe the several mechanical and electronic components of the *AtlasMV*. In section 3.3.3 the sensors that are installed onboard the robot are presented.


(*a*)


(*b*)


(*c*)

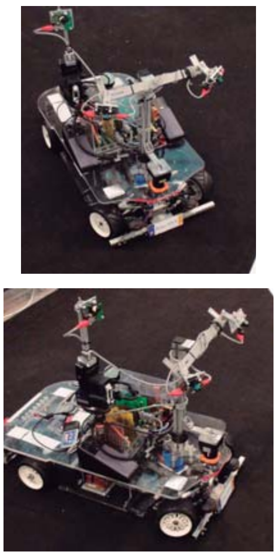

(*d*)

Figure 3.6: The *AtlasMV* autonomous robot. The robot participated in the PRO of 2008 (*a*), 2009 (*b*), 2010 (*c*) and 2011 (*d*).

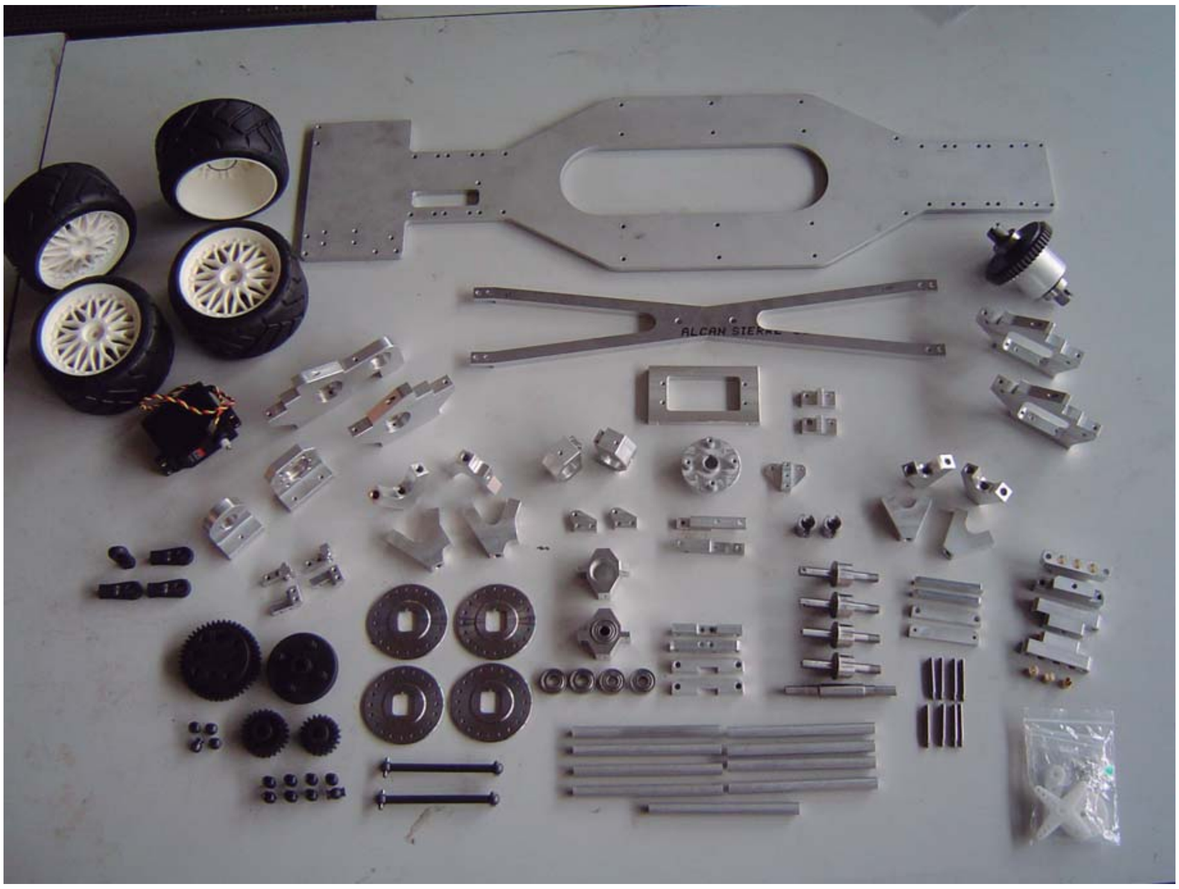
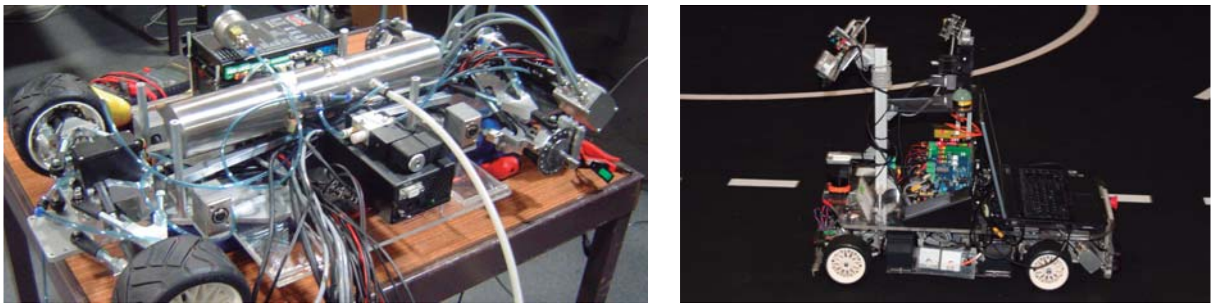


Figure 3.7: An overview of the custom made parts of the *AtlasMV* robot.

3.3.1 Mechanical Components

The *AtlasMV* project started in 2007. During that year, development of both the mechanical and electronic components of the robot was initiated. Unlike the *Atlas2000*, the *AtlasMV* was entirely developed from scratch. Several dozens of mechanical parts were machined purposely for the robot. Figure 3.7 shows some of them. The robot is designed as a two layer structure, as shown in Fig. 3.8. The bottom layer accommodates the steering system, the traction system, the batteries and several electronic controllers. The upper layer accommodates the electronic boards, the cameras support structure, the LRF, and the laptop.

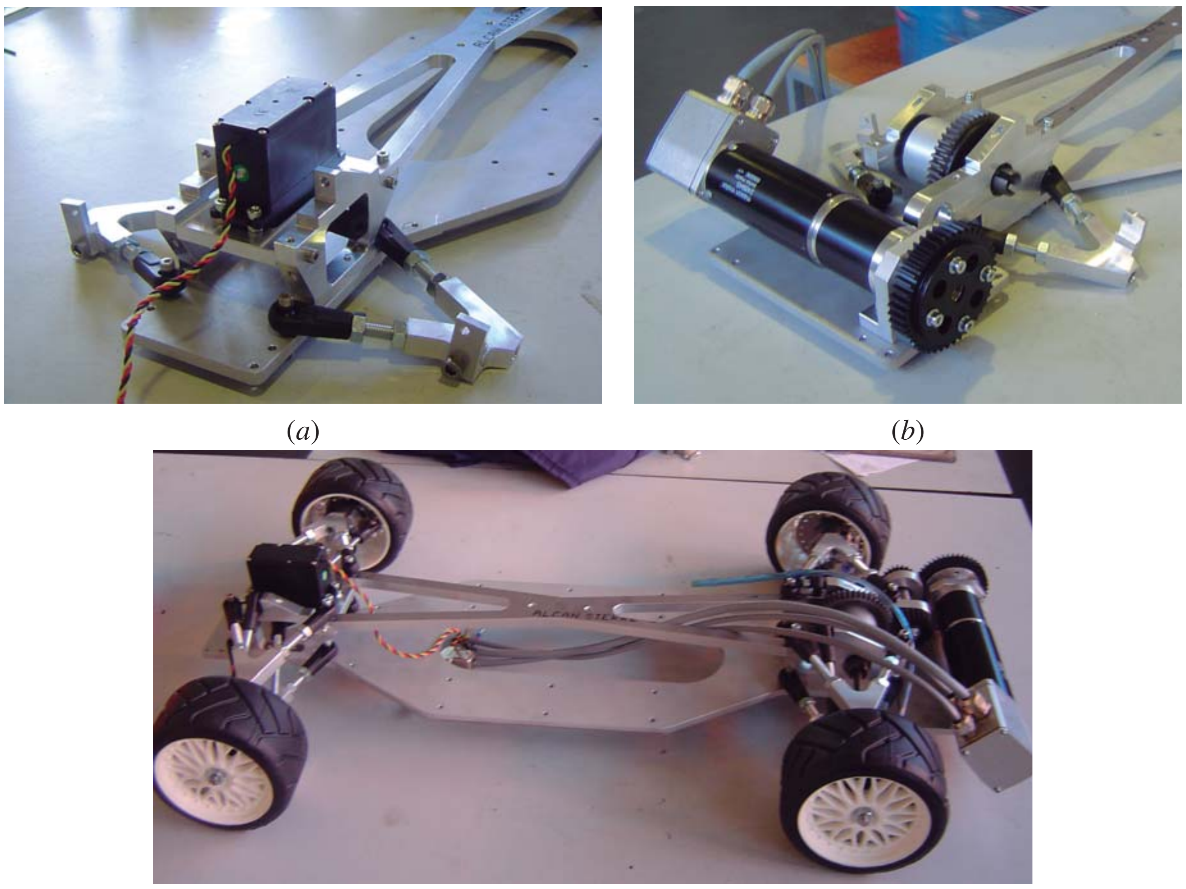
The steering system is an Ackerman inspired system with two wheels in front, and is powered by a servo motor. It is shown in Fig. 3.9 (*a*). The traction system (Fig. 3.9 (*b*)) is powered by a 300W DC motor, coupled to a planetary differential system. The vehicle is capable of reaching speeds of around three meters per second. It weights around 30Kg, which is quite heavy when compared to other robots from the competition. Because of this it does not have the same acceleration capabilities as other robots. Figure 3.9 (*c*) shows both systems mounted on the platform.



(*a*) (*b*)

Figure 3.8: An overview of *AtlasMV*: (*a*) bottom layer. (*b*) upper layer.

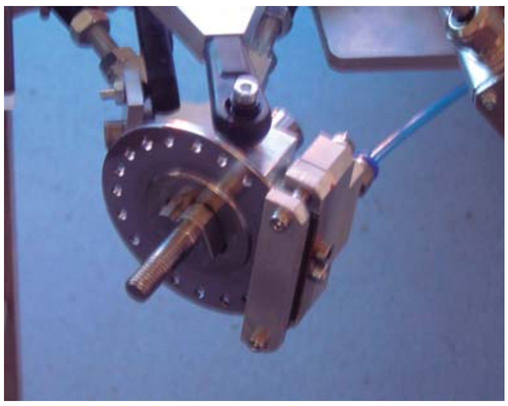
Due of the large weight, one of the main problems was that the braking capabilities of the robot were not adequate. For this reason a braking system was designed and added to the robot. The



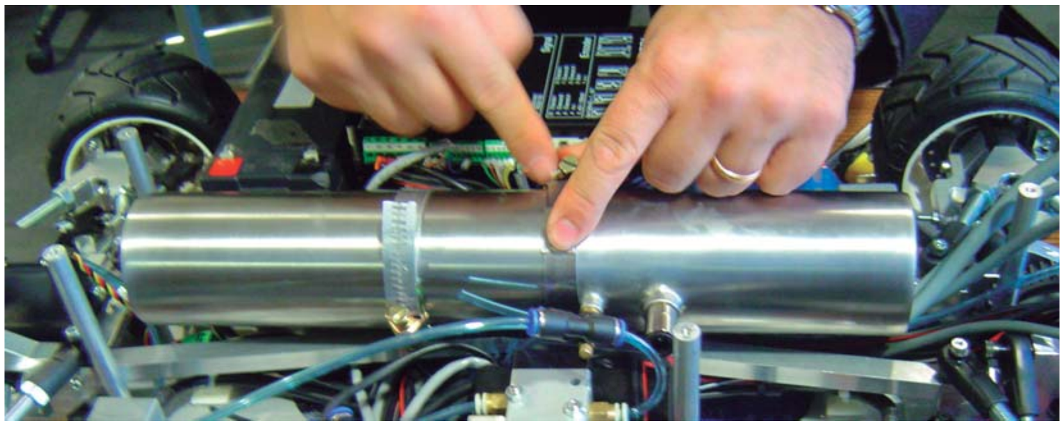
(*a*) (*b*) (*c*)

Figure 3.9: The mechanical components of the *AtlasMV*: (*a*) steering mechanism; (*b*) traction mechanism: (*c*) both systems mounted on the platform.

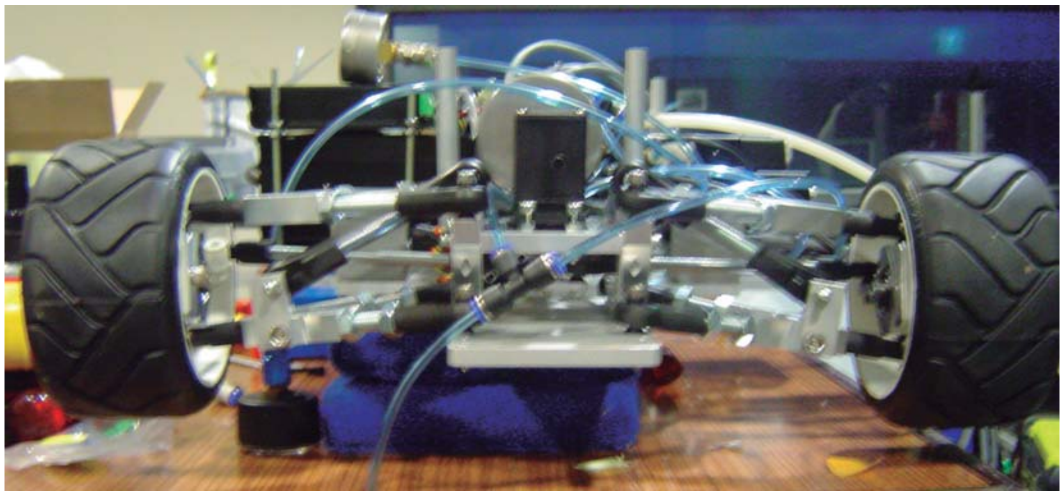
(*a*)



(*b*)



(*c*)



(*d*)

Figure 3.10: The compressed air braking system of the *AtlasMV*: (*a*) pistons and supporting frame; (*b*) structure already mounted; (*c*) the compressed air reservoir; (*d*) the compressed air circuit (blue tubes).

system was initially composed of two brake disks mounted in the front wheels (Fig. 3.10 (*b*)). The disk pads (Fig. 3.10 (*a*)) were actuated by a pressurized air system. The system was controlled by an electro valve that pressurized the circuit and the braking pads, allowing the robot to quickly decrease speed. One of the problems with this system was that air pressure was lost in the pads which meant the vehicle required a compressed air reservoir in order to operate over time. This is shown in Fig. 3.10 (*c*). Figure 3.10 (*d*) shows the pneumatic system onboard the robot. Later, in 2009, the pneumatic system was replaced by a hydraulic one. Also, two additional brake disks were mounted on the rear wheels. These changes solved the autonomy problem. In the previous version, the compressed air reservoir had to be refueled every half an hour. With the hydraulic system, there is only a need for a yearly maintenance procedure. Also, because the oil is much less compressible than air, the hydraulic system proved more efficient. Currently, the *AtlasMV* is capable of braking very fast, which contributes to the good overall performance of the system.

Throughout the years, the mechanical structure proved very reliable. Apart from the braking system, the structure suffered very few changes. The reliability of the mechanical structure of the *AtlasMV* was a key factor to the good performances accomplished by the robot.

3.3.2 Electronic Components

The electronic components onboard the *AtlasMV* include of the shelf components as well as custom designed boards. The off the shelf electronic components are located in the bottom layer of the robot and include the Pan and Tilt Unit (PTU) controller and the traction motor controller. The custom boards are located in the upper layer. The motherboard was especially designed to provide a modular philosophy to the system (Fig. 3.11 (*a*)). It provides common electric signals to other boards, including power and electric signals. Other boards are plugged in as shown in Fig. 3.11 (*b*). The boards serve mainly as an interface between the computer and the motors or sensors. Motor control is achieved by RS232 serial communication between the computer and a microcontroller, which then controls the motors according to the higher level commands. Other boards are designed to stabilize power lines, control relays, acquire analogic and digital signals. Due to constant problems with custom designed electronics equipment, in more recent versions of the robot, the trend has been to remove as much as possible complex tasks from the microcontrollers. In 2011, the latest version of the *AtlasMV* performed direct RS232 communication between the laptop and both the traction and sterring motors. Custom electronics is now mainly used for the aquisition of electric signals, and control of custom equipments. One example is the light signals control board, in which the computer communicates to the microcontroller what should be the state of the lights of the robot, e.g., turn, brake and head lights. The microcontroller then acts on relays that turn on or off the lights. Compared to previous years, the 2011 version was far more robust to communication and control problems of the motors. In total, there are six custom boards onboard the *AtlasMV*. Motor control, lights signal control and activation boards, power board, relay board, and distance sensors board.

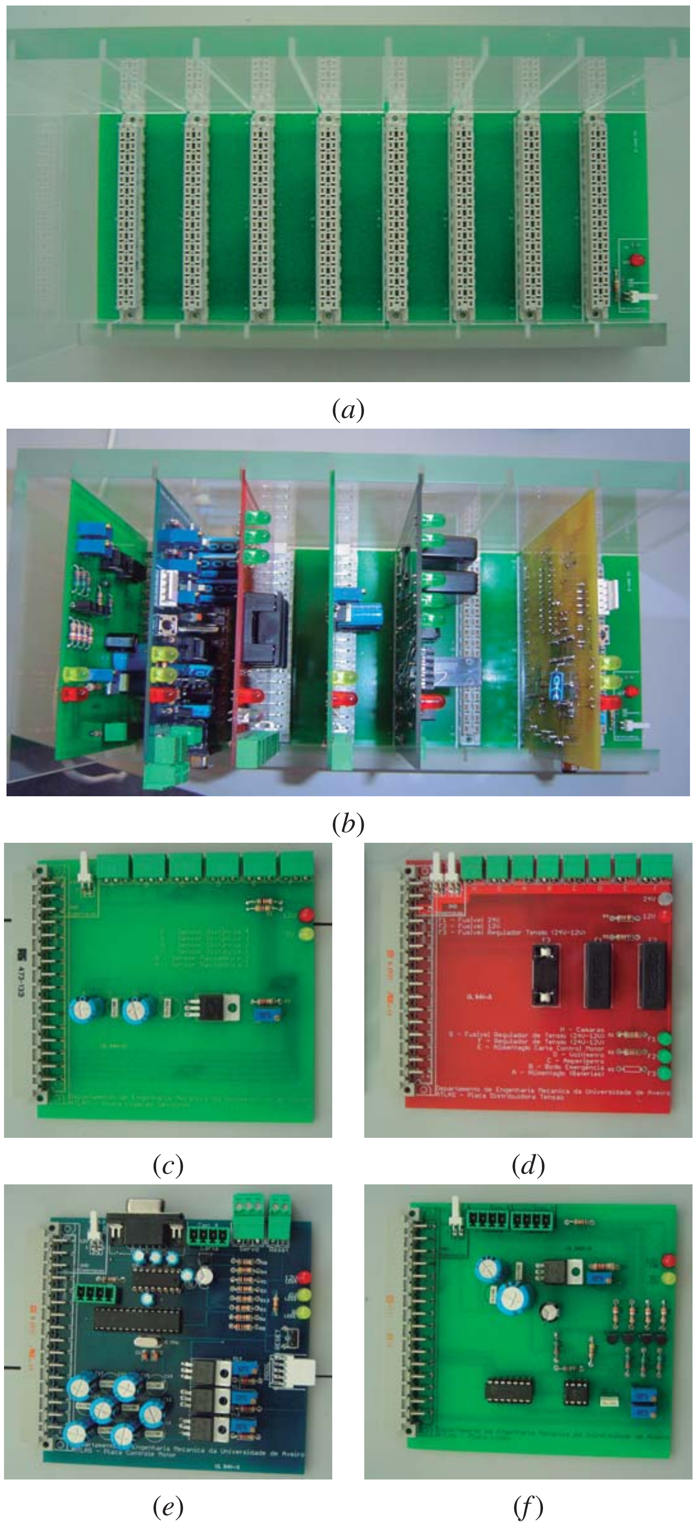


Figure 3.11: The electronic boards of the *AtlasMV*: (*a*) main board; (*b*) main board with other boards mounted; (*c*) distance sensors board; (*d*); power distribution board; (*e*) motor control board; (*f*) lights control board.

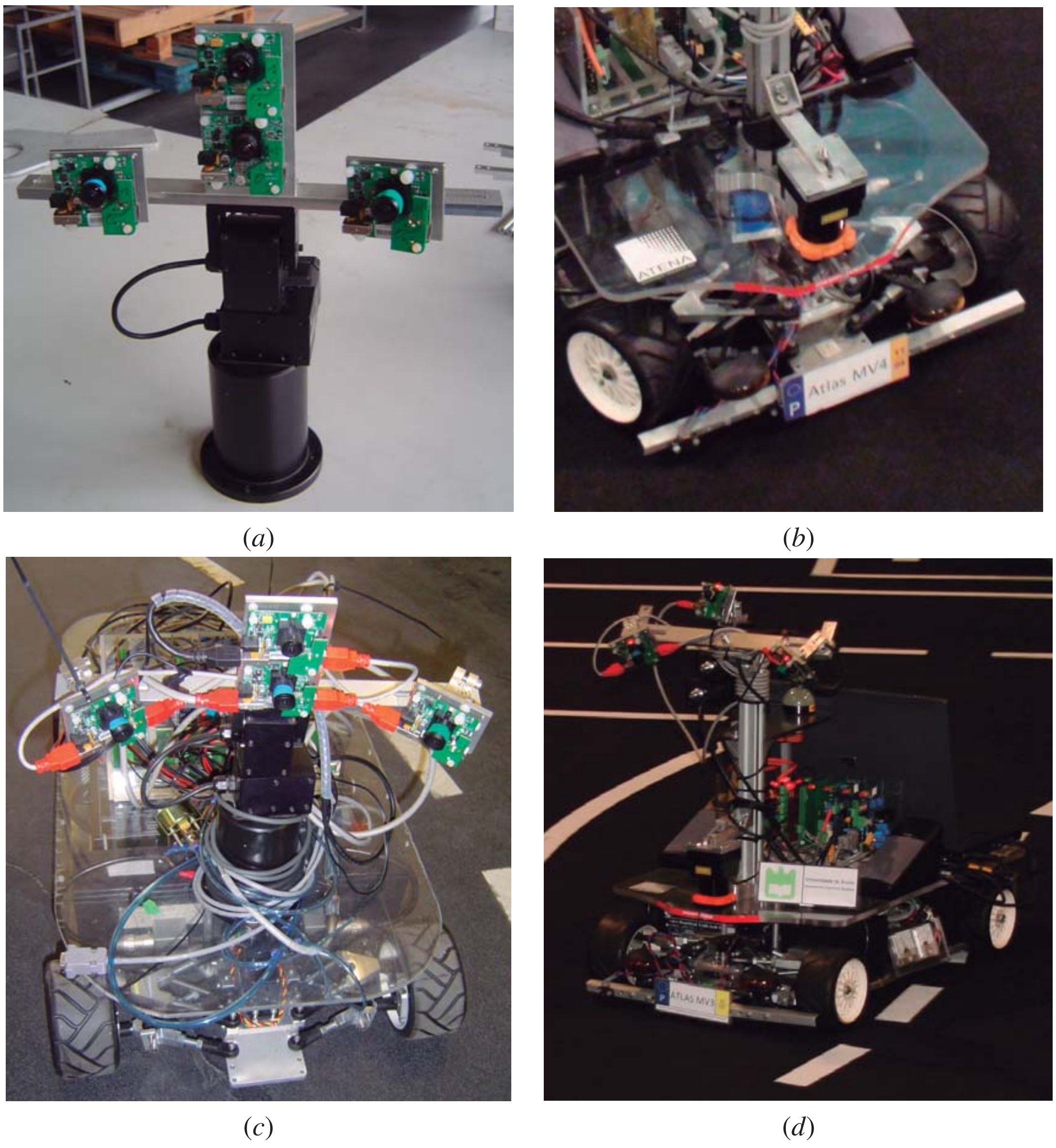


Figure 3.12: The sensors onboard the *AtlasMV*: (*a*) motor controlled camera support; (*b*) laser range finder; (*c*) early version, 2008; (*d*) recent version, 2011.

3.3.3 Sensors

For perceiving the environment around it, the *AtlasMV* uses both vision and a LRF. In the 2008 version, the robot was equipped with four Firewire cameras, mounted on top of an active perception PTU. This structure is shown in Fig 3.12 (*a*), and mounted on the robot in Fig. 3.12 (*c*). The two cameras mounted on the center of the structure, vertically aligned, compose the traffic lights recognition system. The cameras have different focal length distances, which enables one of the cameras to have a peripheral view of the scene and another to collect a high resolution foveal view of the scene. The peripheral camera is used to locate the traffic lights panel and then, using this information, the foveated camera is pointed towards the traffic lights for recognition. Regarding the recognition and control algorithms, more details are provided in section 3.5. The two cameras positioned on the side of the structure are used to obtain a complete view of the road. Given the dimensions of the vehicle and the road, it is not possible to use a single camera to view the entire lateral extension of the road. Therefore, the two side cameras are used to provide partial views of the road. These views are then fused into a single image, using a multi camera Inverse Perspective Mapping (IPM) algorithm. More details on this are provided both in section 3.5 and chapter 5.

In 2009 the *AtlasMV* was equipped with a LRF. The sensor is mounted on the front of the robot (Fig. 3.12 (*b*)), and provides range measurements on a plane parallel to the road, approximately 25 centimeters higher. The LRF equipment gave the *AtlasMV* obstacle avoidance capabilities which had not been seen until then. The robot was capable of not only avoiding obstacles but also of tracking and pursuing moving obstacles. The sensor provided a much more robust sensing of the obstacle, the tunnel and the road maintenance area pins. Previous vision based color segmentation approaches were replaced by the LRF based approach.

In 2010, the side cameras used for viewing the road were mounted on a fixed structure. The traffic lights recognition system was composed of a single camera mounted on a PTU.

3.4 *AtlasCar*

The *AtlasCar* (http://atlas.web.ua.pt/) [Santos *et al.* 2010] is an autonomous vehicle prototype developed by the group for Robotics and Automation from the Department of Mechanical Engineering at the University of Aveiro. It is a common vehicle equipped with several sensors and actuators. Sensors are used to perceive the environment in and around the vehicle, and actuators to control the vehicle in accordance with commands provided by the computer.

The challenge in developing such a system comprises many scientific issues in data acquisition, processing, fusion and interpretation, as well as efficient storage and data flow, but also many other engineering concerns have risen such as selecting an adequate software architecutre for the system. The purpose of such an equipped vehicle is to collect multi sensor data from road scenarios, which is then used to create models for enhanced perception and data fusion. The main points driving this project are:

- Sensorial redundancy using different physical principles;
- Capability of massive data logging, including multiple sensor time and spatial registration;
- Scalability of hardware and software solutions;

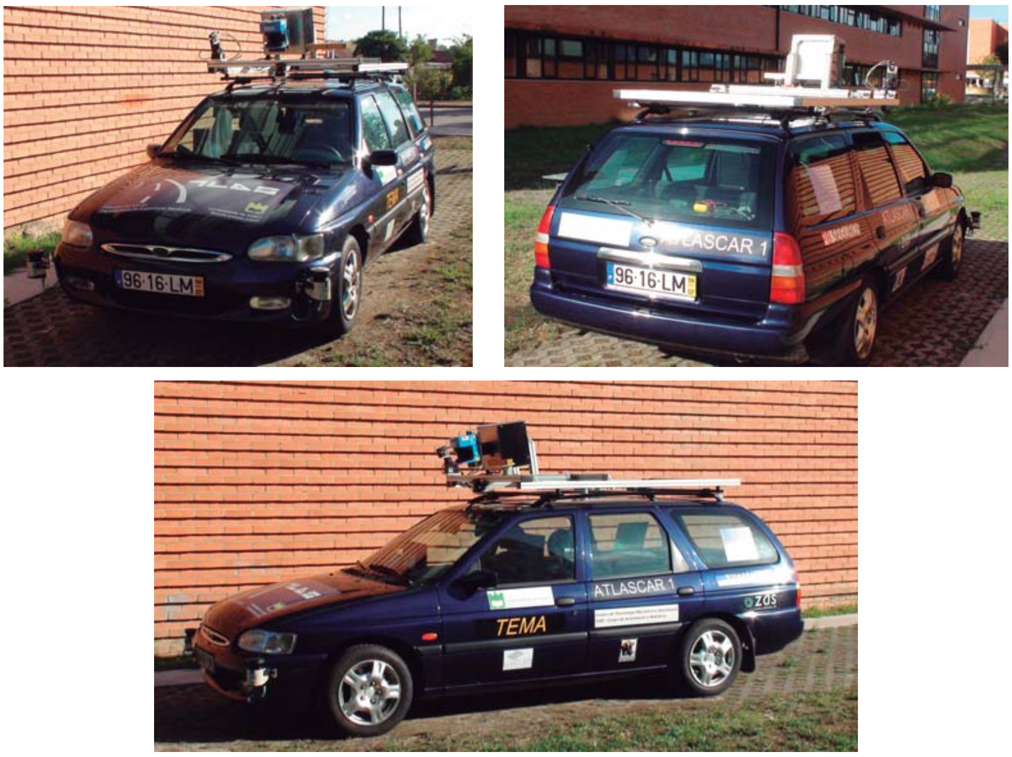


Figure 3.13: The *AtlasCar* autonomous vehicle.

- Power autonomy and safety both by surging mechanical power from the car engine and proper interface for using wall socket power when parked;
- Maintain the cars legal conformity and compatible with human driving.

Before reaching real world autonomous driving, intelligent vehicles must first have robust perception capabilities. This accounts both for information from the external environment and agents, and also the vehicle own status. Besides these sources of data, intelligent vehicles will also monitor the driver actions and, to the extent possible, his state of awareness. All this concurs to create unprecedented safety and assistance during road driving. Due to the complexity of the dynamics of the environment, sensorial redundancy must be used. The first step is then to equip a vehicle with a large amount of sensors and collect data from road like environments. The *AtlasCar* is capable of logging massive amounts of multi sensor data, to be processed offline. This opens many fronts of research in data interpretation, fusion and integration. When the algorithms are developed, the *AtlasCar* will serve as a first test platform. Ultimately, the *AtlasCar* is expected to demonstrate autonomous driving capabilities.

The project started in 2009, with the installation of a power generation and management module and an array of sensors. The chosen platform was a standard gasoline-powered Ford Escort Wagon with 75 *horse power*, a 460 liter trunk, manual gear and autonomy of more than 500 km. The first

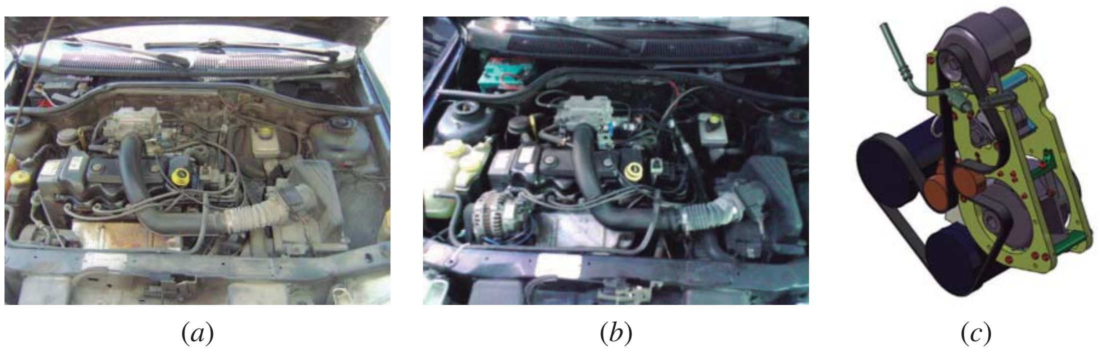


Figure 3.14: (*a*) the engine compartment before the installation of the secondary alternator; (*b*) the engine compartment with the new alternator installed, visible in the bottom left part of the image; (*c*) a CAD drawing of the alternators in their new configuration. Source [Santos *et al.* 2010].

public appearance of the *AtlasCar* was at the PRO 2010 in Leiria, Portugal, on April of 2010. From there onward, the vehicle has made several demonstrations at a varied set of events. It is the first full scale autonomous vehicle developed in Portugal. Figure 3.13 shows the *AtlasCar*.

The objectives of the *AtlasCar* project consist in the research and development of Advanced Drivers Assistance Systems (ADAS) and of Autonomous Driving (AD) technologies. In this sense, the *AtlasCar* is a mobile laboratory for multi sensor data acquisition of real road scenarios. These data sets are publicly available to the scientific community. Since this prototype was used in many experiments, either for acquiring data or to test algorithms, it makes sense to provide a detailed description of the platform.

3.4.1 Mechanical and Electronic Components

To ensure power supply, a second alternator driven by the vehicle propulsion engine was installed on the engine compartment. This was achieved by redesigning the mechanical electric power generation inside the engine compartment, as shown in Fig 3.14. This redesign consisted of relocating the original alternator to a higher position in order to get room for the new generator, which supplies 1.2 to 2.5 kW, depending on engine rpms [Santos *et al.* 2010].

The DC output from the alternator is directed to a battery which serves as a buffer, and is then converted to a higher-voltage line (220V, AC) through an inverter. Finally, the power chain ends at an UPS to ensure a properly stabilized power throughput. This stabilized power output is driven to the electric panel. The electric panel is composed of three modules, two power regulators at 12V DC and 24V DC which feed different types of devices and sensors and a PLC. Currently, the task of the PLC is mainly to allow a software-based reset and power switching capabilities of sensors and other hardware. Computers and monitors are supplied directly from the UPS. In sum, onboard the vehicle there are 220V AC, 12V DC and 24V DC. These options are sufficient for all the sensors, computers and actuators installed. Figure 3.15 shows the power stabilization modules installed in the trunk. Figure

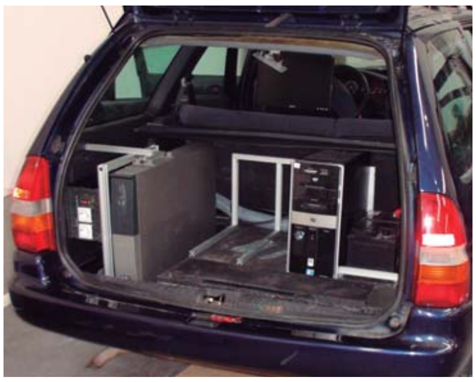
3.16 shows a diagram of the power distribution and switching onboard the vehicle along with the main component connections. To allow versatility, the system is also prepared to operate directly from a regular wall power socket for in-house developments [Rocha 2011]. In [Montemerlo *et al.* 2008] and [Miller *et al.* 2009], similar configurations were devised to provide power generation.

3.4.2 Sensors

As discussed before, one of the objectives of the *AtlasCar* project is to provide a mobile platform for obtaining vasts amount of multi sensorial data. Hence, the number of sensors onboard the vehicle should be as large as possible. Figure 3.17 shows a scheme of the sensors onboard the *AtlasCar*. It is equipped with five cameras, four lasers, a GPS and an inertial measurement unit. The following sections will describe in detail the sensors mounted onboard the *AtlasCar*.

Lasers

There are two Sick LMS151 Lasers mounted on each side of the front bumper, such that the scan plane is parallel to the ground plane. They are shown in Fig. 3.17 (*C*). These sensors measure in a 270 degrees plane at 50 Hz, and have a maximum range of 50 meters. The Hokuyo UTM30LX laser is mounted on the roof of the car pointing towards the road. It is marked in Fig. 3.17 (*H*). The laser takes measurements has a 30 meters range on a 270 degrees span, at a maximum output frequency of 40 Hz. Finally, a Sick LMS200 (Fig. 3.17, (*D*)) is mounted on top of a rotating platform in the roof of the car. The laser is capable of taking range measures on a 180 degree plane, up to 20 meters and 20 Hz. When the laser rotates, so does the scan plane. This enables the vehicle to obtain 3D measurements all around its frontal hemisphere. Although the number of lasers onboard the *AtlasCar* is scarce when compared for example with the *Darpa Challenge* competitors, they do


(*a*)

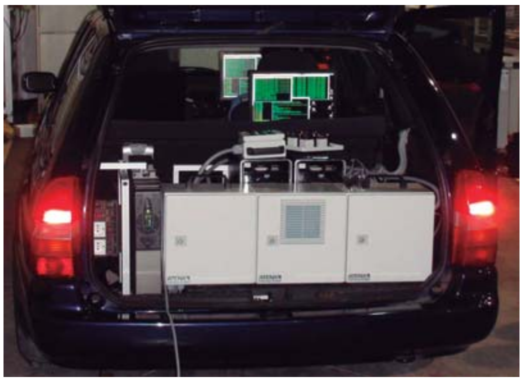

(*b*)

Figure 3.15: (*a*) The power stabilization equipment onboard the *AtlasCar*: from left to right, the inverter, the UPS, the buffer battery; (*b*) The final installation of the *AtlasCar* trunk. Inside the white boxes are the AC-DC power converters and the PLC.

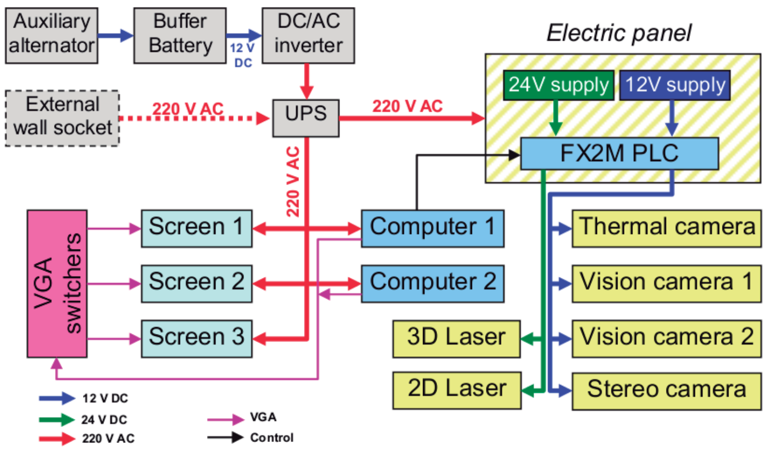


Figure 3.16: A schematic of the power supply lines of the *AtlasCar*. Source [Santos *et al.* 2010].

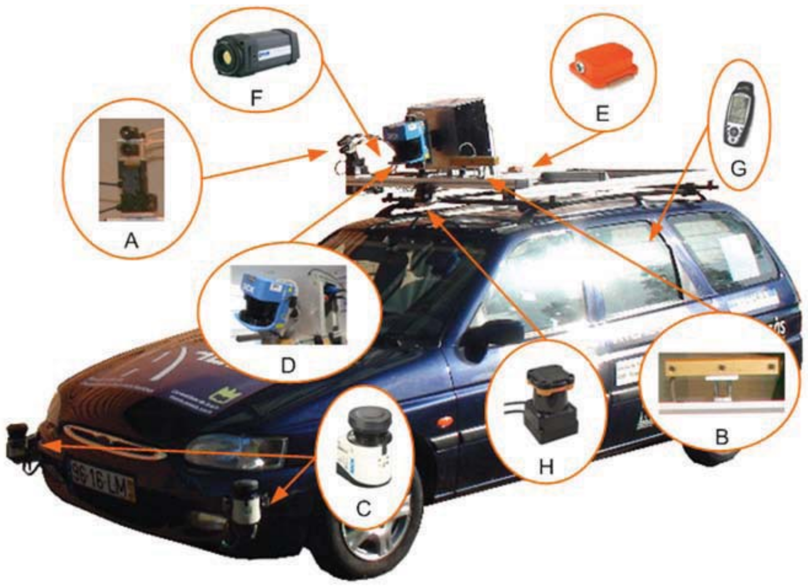


Figure 3.17: The *AtlasCar* full scale robotic platform. It is equipped with an active perception unit (*A*), a stereo rig (*B*), four LRF (*C, D, H*), a thermal vision camera (*F*), GPS (*G*) and an inertial measurement unit (*E*).

provide a reasonable cover of range measurements around the car. Figures 3.18 and 3.19 show the coverage of each of the lasers.

Cameras

There are three cameras mounted on the *AtlasCar*. They are all facing the front of the vehicle. Figure 3.17 (*B*) shows a Point Grey Research Bumblebee XB3 Stereo Camera (XB3). The three cameras

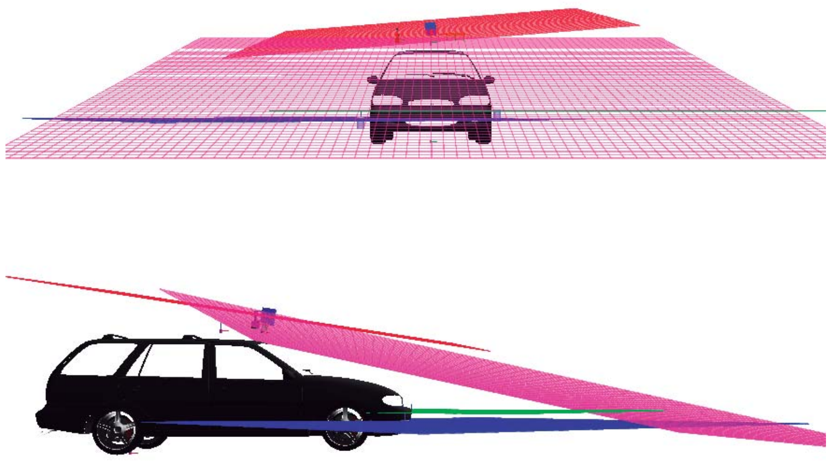


Figure 3.18: Coverage of the lasers mounted onboard the *AtlasCar*. The scan plane of each laser is depicted in different colors: (blue) right bumper; (green) left bumper; (magenta) center top roof; (red) rotating laser mounted on the roof.

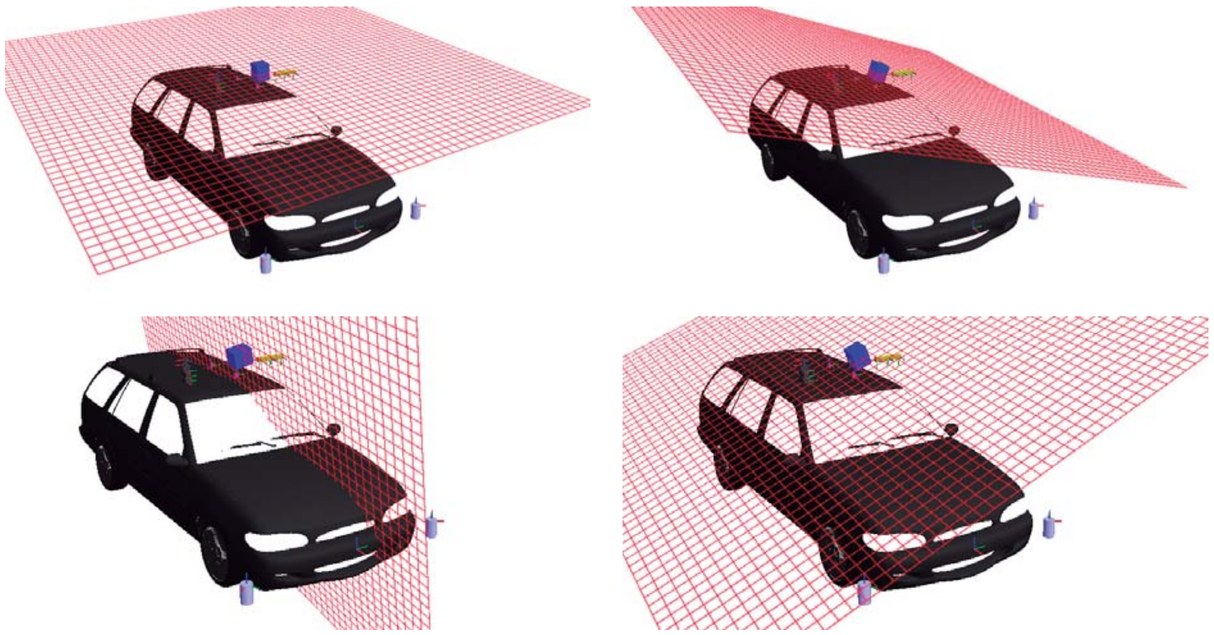


Figure 3.19: Coverage of the 3D laser scanner mounted on the roof. Several possible positions are displayed.

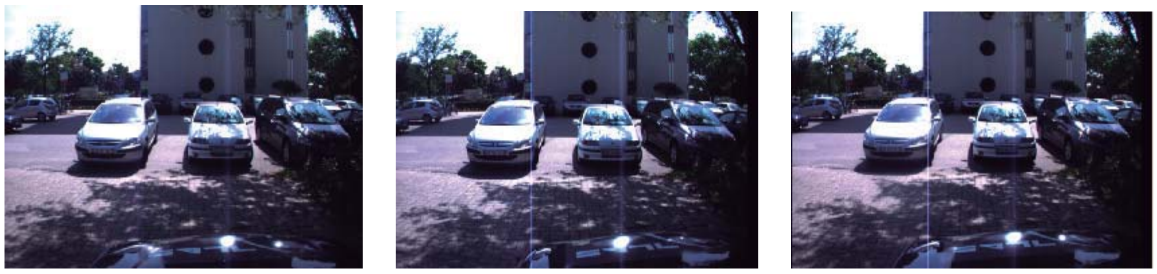


Figure 3.20: Images from the XB3 camera. These three images are used for two stereo pairs.

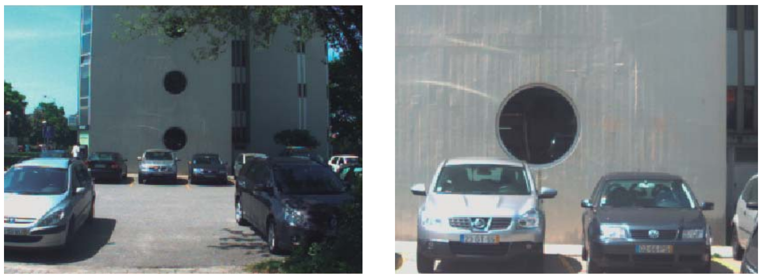


Figure 3.21: Images from the peripheral and foveated cameras

on the stereo rig provide two stereo pairs. Images have resolution of 1280×960 at a frequency of 15 Hz. Using these two stereo pairs, additional range measurements are made available to subsequent perception algorithms. The stereo rig uses 3.2 millimeters lenses which provides a complete view of the road. Figure 3.20 shows images from the cameras on the stereo rig.

Two *Point Grey Research Flea2* cameras are mounted on top of a *Directed Perception PTUD46* PTU. These form what is called the active perception unit (Figure 3.17 (*B*)). The two cameras provide images of 1280X980 resolution at 30 Hz and have different lenses, 4 and 12 millimeters focal distance. Hence, while one camera displays wide view images of the whole scene, the second camera outputs high detail images of small areas. The PTU is able to, given an order by the computer, point the cameras towards any object of interest in front of the car. Figure 3.21 shows images from both these cameras.

Proprioceptive Sensors

These sensors include all the equipments taking measurements of the internal status of the vehicle. A *Magellan GPS* is employed to obtain a global position Figure 3.17 (*G*), while an *Xsens MTI* inertial measurement unit provides estimates for the vehicle's egomotion. A second PLC is installed in the *AtlasCar*. It is used to communicate with several hardware modules installed all around the vehicle. They measure, amongst others, the position of the wheels and the angle of the steer, for odometry estimation, the engines RPMs, the positions of the pedals and the status of the lights.

3.5 Algorithms

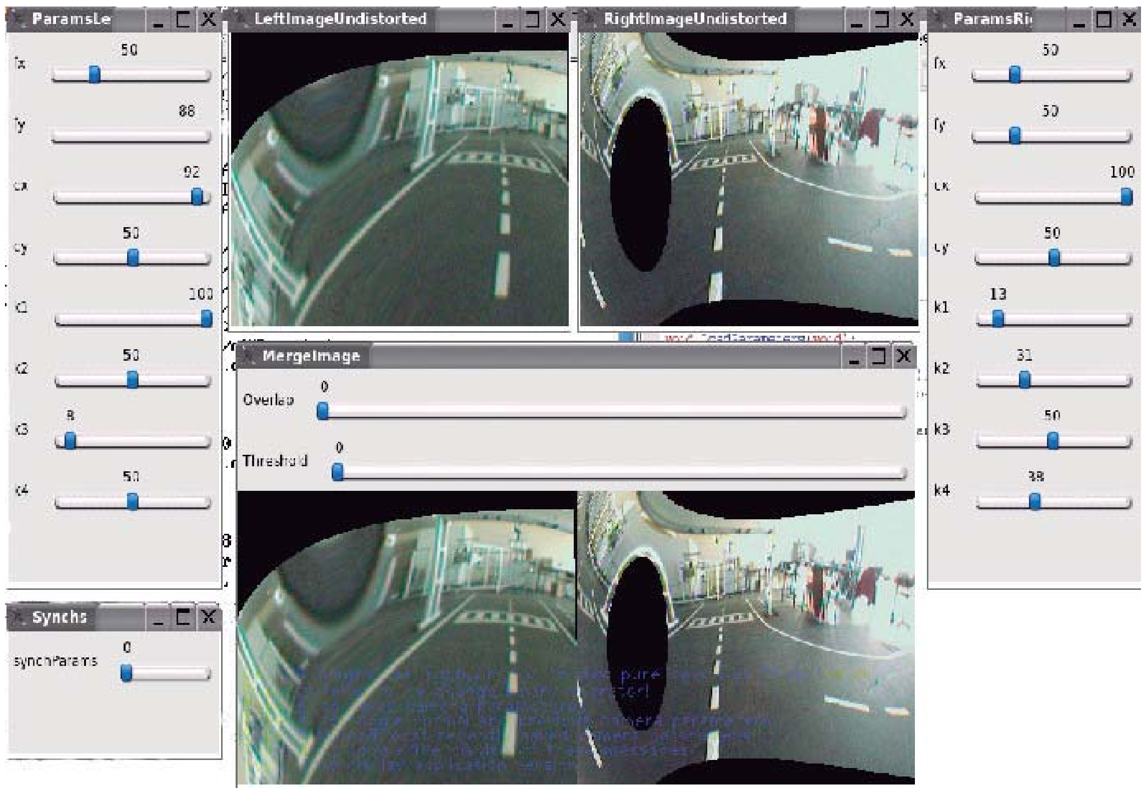
This section presents some of the algorithms developed for the *Atlas2000* and *AtlasMV* robots, particularly for the ADC competition. Four topics are addressed: fusion of vision and LRF sensors, section 3.5.1, road detection, section 3.5.2, navigation of the road maintenance area, section 3.5.3 and path planning, section 3.5.4.

3.5.1 Sensor Fusion

In the context of the ADC, the problem of fusing information from multiple sensors became an issue when it was necessary to use two cameras to view the road. Later, with the addition of a LRF, it was also necessary to obtain a representation where both visual and laser information could be processed.

The first approach to the problem, in 2006, was quite simplistic: to perform a manual calibration [Oliveira & Santos 2007]. An application was developed that permitted the user to change several parameters from both cameras in real time and see the resulting mosaic obtained from merging the images from both cameras. The parameters consisted of a description of each camera's pin hole model parameters, along with the barrel distortion model coefficients. An additional parameter defined the horizontal distance (in pixels) of the overlap of both images. Figure 3.22 (*a*) shows the calibration application. After calibration, the set of values for those parameters was saved and used during the competition. During runtime execution, the algorithm produces a mosaic of both cameras in real time. An example is shown in Fig. 3.22 (*b*). Once computed, the mosaic is used for detecting the road, the cross walk, obstacles, etc. All other recognition algorithms are computed using the mosaic as input. As shown in Fig. 3.22 (*b*), the images from the two cameras do not entirely overlap. Also, there is no absolute metric information. Obviously the size of objects in the image provides hints of their real size. However, it is not possible to have exact size or distance measurements using this mosaic. Although this image fusion approach is a very simplistic approach, the fact is that that the road recognition algorithm was capable of using this information as input to accurately detect the road. The *Atlas2000* used this approach from 2006 until 2010.

In 2008, a new algorithm for the fusion of the navigation cameras on the *AtlasMV* was introduced [Oliveira & Santos 2008b]. The algorithm uses the IPM technique to map images from two cameras onto a single, bird's eye view of the road. The technique consists of transforming the images taken into a new reference frame where the perspective effect is corrected. This reference frame is usually defined on the road plane, so that the resulting images become a top view of the road. One of the advantages of IPM is that the subsequent perception algorithms can be computed in a 2D synthesized world, which significantly eases the tuning of convolution filters size [McCall & Trivedi 2005], the stability of neural network's inputs [Pomerleau 1995], or the detection of features of interest [Bertozzi *et al.* 1997]. The technique requires some a priori knowledge, namely, the geometric transformation relating the cameras' and road reference frames. This is equivalent to state that the camera's position, orientation and intrinsic parameters must be known before hand. Commonly, the IPM technique also assumes that the road ahead is flat, that is, all pixels from the input image are views of points in the real world from the XoY plane of the road's reference frame. This assumption is a core issue of IPM. If undertaken by mistake, due to the pres-



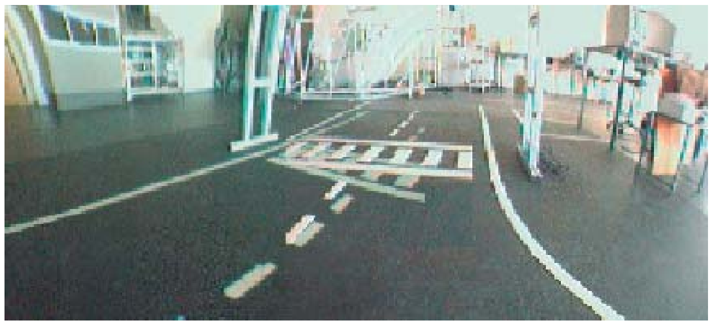
ParamsLe: fx 50, fy 88, cx 92, cy 50, k1 100, k2 50, k3 8, k4 50

ParamsRi: fx 50, fy 50, cx 100, cy 50, k1 13, k2 31, k3 50, k4 38

MergeImage: Overlap 0, Threshold 0

Synchs: synchParams 0

(*a*)



(*b*)

Figure 3.22: Non metric fusion of images: (*a*) manual calibration application for configuring the non metric fusion; (*b*) a mosaic obtained by fusion of two images. Source [Oliveira & Santos 2007]

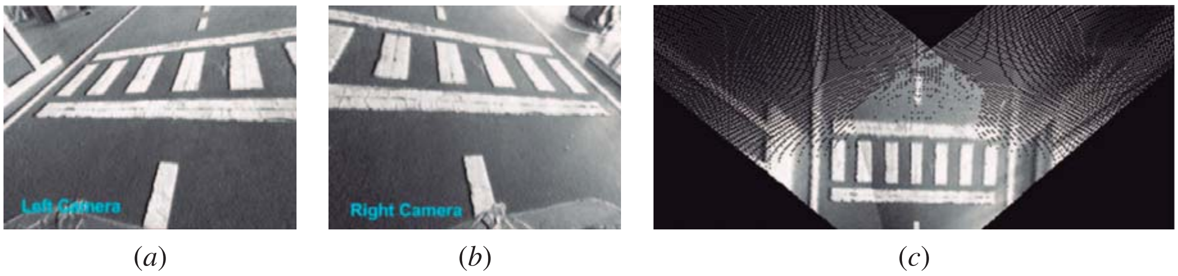
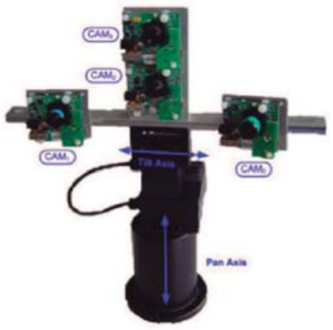


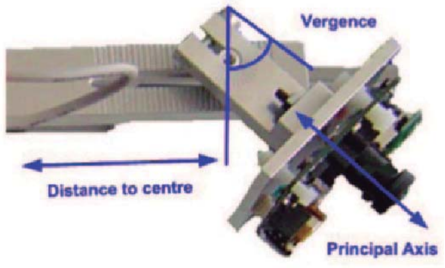
Figure 3.23: Metric fusion of images using IPM: (*a*) image from the left camera; (*b*) image from the right camera; (*c*) bird's eye view of the road.

ence of other vehicles, pedestrians, obstacles, or steep slopes in the road, the IPM produces wrong representations in the undistorted image. This problem is addressed by several researchers on this field [Bertozzi & Broggi 1998] [Broggi *et al.* 2006] [Labayrade *et al.* 2005]. Figure 3.23 shows an example of an IPM operation using the two navigation cameras onboard the *AtlasMV*. The advantage of the composite image obtained is that it is possible to accurately relate a distance in pixels to a metric distance in the real world. This technique was a large step forward with respect to the previous non metric image fusion, since it became possible to assess metric distances and use these to execute more complex motion planning algorithms. Also, with this technique the images from both cameras are accurately registered. The phantom artifacts that appeared due to a wrong registration in the previous technique (see Fig. 3.22 (*b*)) are solved with this approach. Another advantage is that, since IPM corrects distortion in the images associated with perspective, the road painted patterns appear now unchanged in the bird's eye image. The perspective corrected images show the road markings with the same shape, regardless of the point from which they are observed. This opened new possibilities for the detection of road markings (lane delimiters, cross and park zone markers) using template matching techniques. Some of them will be presented in the next sections.

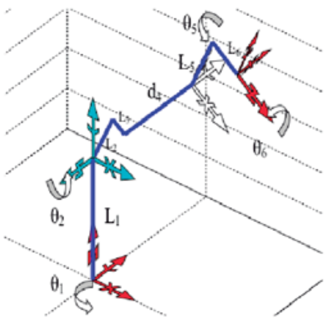
During the first two years of its participation in the ADC competition, the cameras onboard the *AtlasMV* were mounted on a PTU unit. This system was described in section 3.3.3. It is shown in Fig. 3.12 (*a*) and (*c*). The PTU changes position according to commands provided by an RS232 communications link. This means that the camera's orientation with respect to the road could be changed in real time. However, IPM technique requires as input the position of each camera with respect to the road. Moving the PTU meant moving the cameras, which in turn changed the inputs of the IPM processing. To solve this issue, a Denavit Hartenberg [Hartenberg & Denavit 1955] [Hartenberg & Denavit 1964] kinematic chain was defined that enabled the computation of each camera's position and orientation, given the PTU's position. Figure 3.24 (*c*) shows a diagram of the kinematics chain. The advantage of this technique was that it was possible to change the orientation of the cameras (by moving the PTU) and still produce a bird's eye view of the road. Figure 3.25 shows three examples of IPM images obtained from different positions set in the PTU.



(*a*)

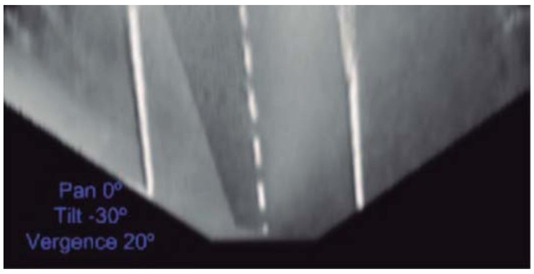


(*b*)

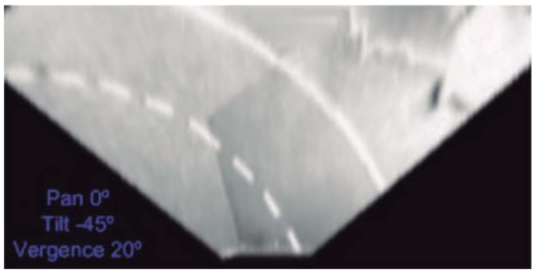


(*c*)

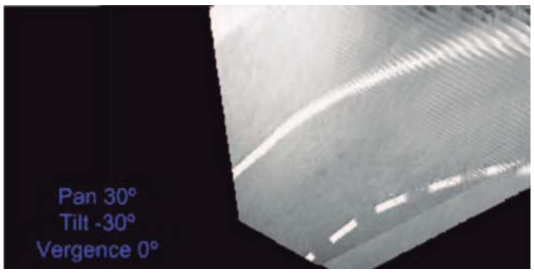
Figure 3.24: The Denavit Hartenberg kinematics chain for one of the cameras of the *AtlasMV*: (*a*) the cameras support structure; (*b*) a detail of the camera's mounting; (*c*) a diagram of the kinematics chain.



(*a*)



(*b*)



(*c*)

Figure 3.25: Images produced by IPM using different positions of the PTU: (*a*) pan angle 0 degrees, tilt angle -30 degrees; (*b*) pan angle 0 degrees, tilt angle -45 degrees; (*c*) pan angle 30 degrees, tilt angle -30 degrees.

Another advantage of the IPM technique is that, since the image has a metric nature, it is possible to overlay range measurements onto it. These measurements are provided by the range measuring sensor, which is registered with the cameras. This was done in 2009, where the measurements of the LRF onboard the *AtlasMV* were fused with the bird's eye view representation of the road. Figure 3.26 shows the output of this algorithm. It is possible to see that the legs standing in front of the robot

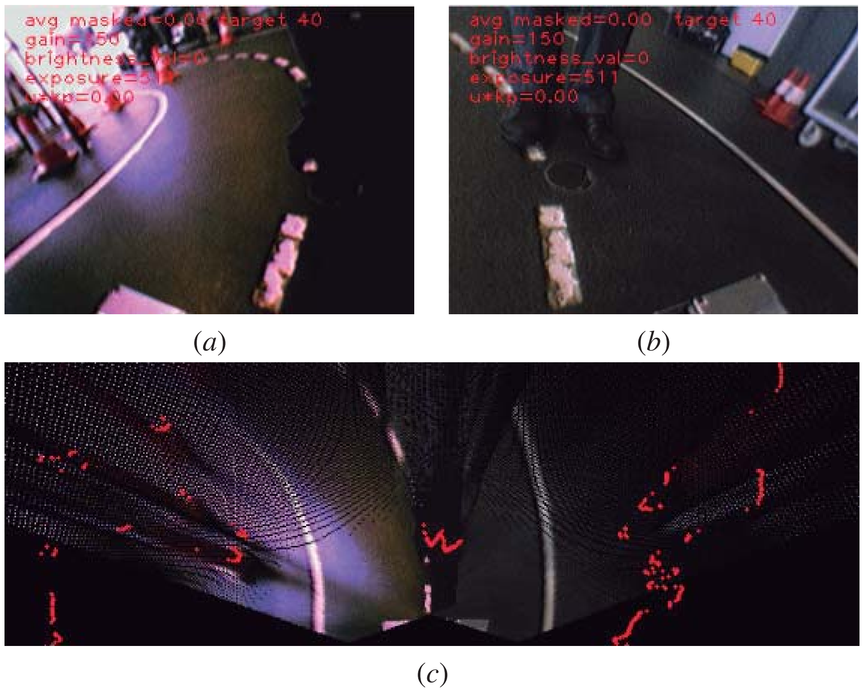


Figure 3.26: Overlaying laser range measurements onto the IPM image: (*a*) image from left camera; (*b*) image from right camera; (*c*) bird's eye view with laser range measurements (red dots).

are not only viewed by the cameras but also signaled (as red dots in Fig. 3.26 (*c*)) by the LRF. This algorithm will be detailed in chapter 5. It was also presented in [Oliveira & Santos 2008a].

3.5.2 Road Detection

This section describes some of the approaches that were attempted to perform the task of detecting the road. Two algorithms will be described. First, a flood fill based algorithm that searches for the area inside the road. Second, a line detection algorithm that was used by the *AtlasMV* in 2008 and 2009. Unlike in the previous approach, in this case the approach is to detect the lane markers that delimit the road, instead of the region contained by them. Only a brief description of the algorithms is provided in this section. Further details are given in [Oliveira *et al.* 2005] and [Oliveira & Santos 2007].

The first algorithm uses a set of sequential flood fill operations to filter the area that is contained inside the road. It relies on the connectivity of the lane markers that define the boundaries of the road. First, the image from the road (Fig. 3.27 (*a*)) is binarized (Fig. 3.27 (*b*)). Then, a line is added to the image to cap the upper part of the road, and a flood fill operation is initiated from inside the road (Fig. 3.27 (*c*)). The region above the line is erased (Fig. 3.27 (*d*)) and a second flood fill operation is executed, this time using the outside of the road as a seed (Fig. 3.27 (*e*)). Finally, the negative of this image is used to mask the area contained inside the road (Fig. 3.27 (*f*)).

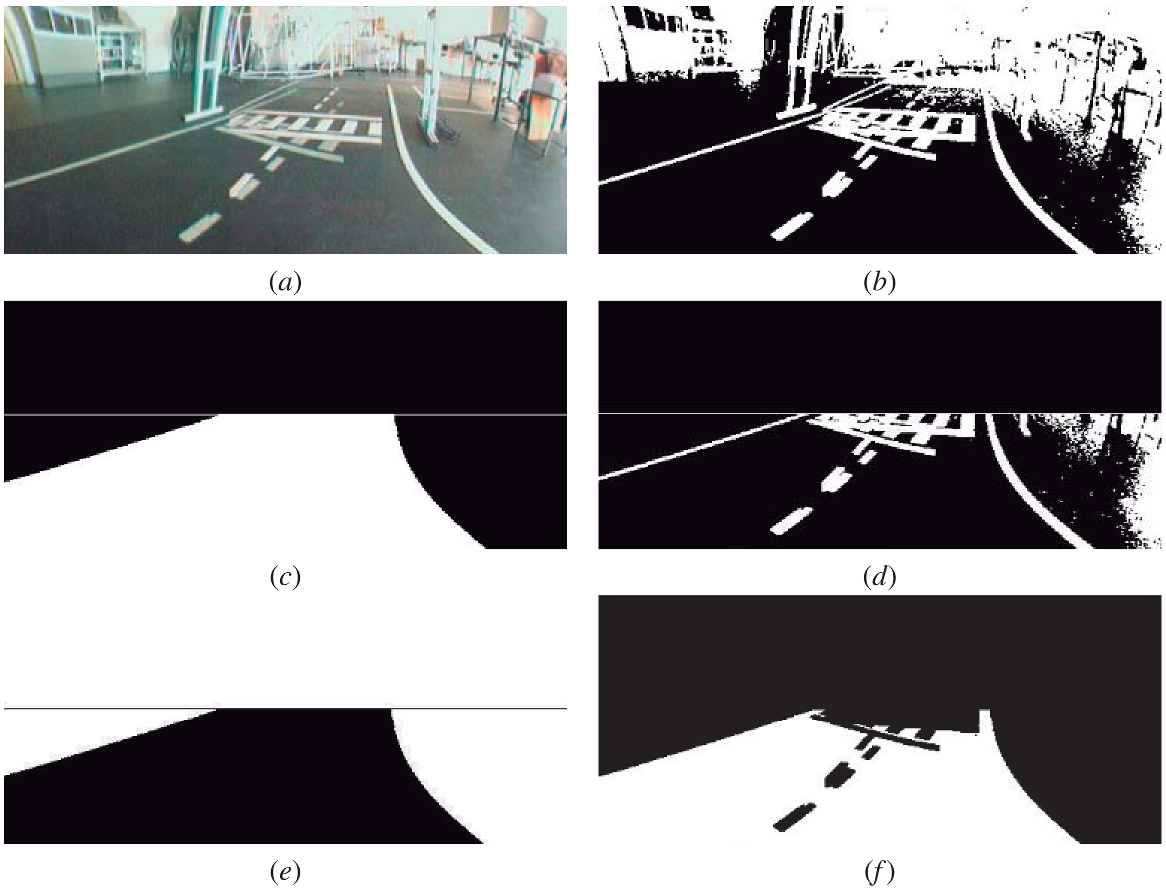
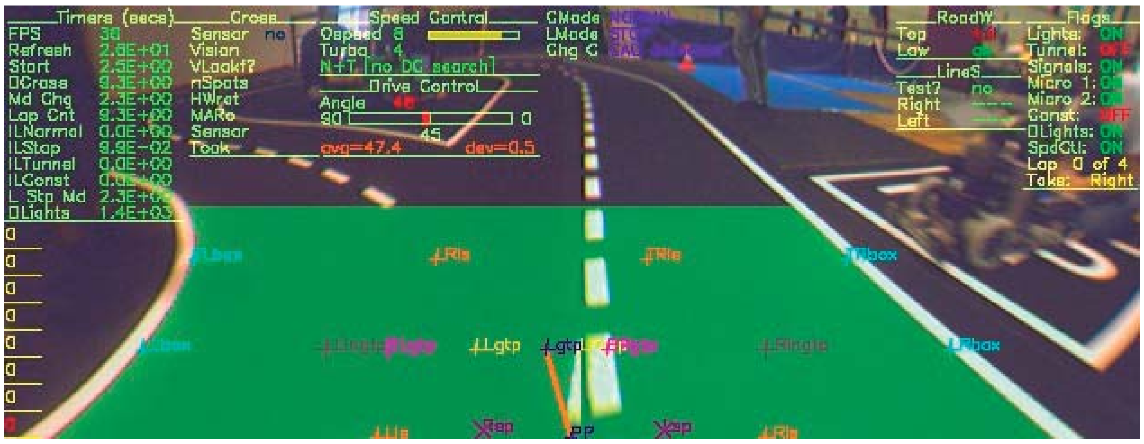


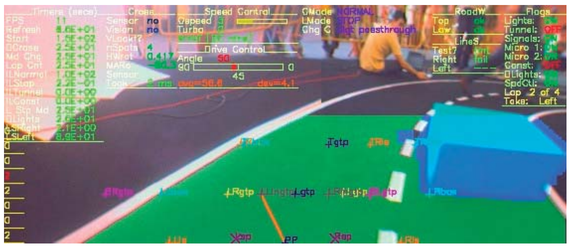
Figure 3.27: Flood fill based road detection: (*a*) original image; (*b*) binarized image; (*c*) flood fill from inside the road; (*d*) erase upper region of the image; (*e*) flood fill from outside the road; (*f*) final mask of the region contained by the road.

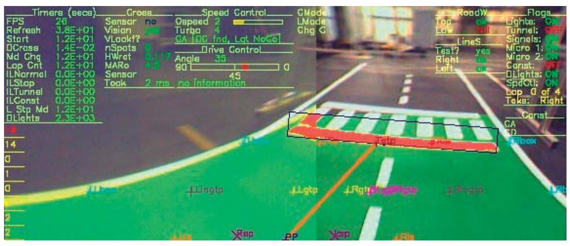
The flood fill based algorithm was used in the *Atlas2000* until 2010, and was a cornerstone of the achievements of that robot. Figure 3.28 shows the output of the algorithm for several road configurations.

The second algorithm here described approaches the problem of road detection from a different angle [Oliveira & Santos 2008c]. Instead of detecting the region contained inside the road delimiting lines, it tries to find the lines. The algorithm works on top of the IPM image described in section 3.5.1. The reason for that is that the removal of the perspective effect makes the lines to have a constant width. There are many other examples of the usage of IPM for easing the road detection. Some examples are [Bertozzi & Broggi 1998], [Kim & Pollefeys 2008]. The IPM image is processed using a morphological top hat operation. The top hat enables the extraction of the lines in demanding illumination conditions, such as shadows. The size of the structuring element used in the top hat operation was defined considering that the IPM image represents accurate geometrical information: it is defined with the radius slightly larger than the road lines' width. Figure 3.29 (*a*) shows the result of



(*a*)





(*b*) (*c*)

Figure 3.28: Road detection using a flood fill based algorithm: (*a*) intersection; (*b*) right turn; (*c*) approaching the cross.

a top hat operation over an IPM image. The next step is to generate what is called brightness profile, in order to search for possible lane marker candidates in the image. An horizontal line is scanned over the top hat image. The scanned line is used to extract an intensity profile in the top hat image. This signal is then clustered into several groups after the definition of a maximum gradient value. Figure 3.29 (*b*) shows the brightness profile and the clusters of the top hat image shown in Fig. 3.29 (*a*). The clustered groups are then filtered to check whether their average brightness is higher than their left and right neighbors. Groups that do not have this property are discarded.

Each group that was not discarded represents a possible intersection of a road line marker with the scan line. The middle point of each group is then employed as a seed point to a flood fill operation, reconstructing the candidate line from that point. This operation generates sets of line candidates, that is, Boolean images of the same size as the IPM image, where the pixels that have been filled are marker as true, and all others as false. The images of the candidate lines go through a search routine that finds several relevant points on the given line candidate. The search routine consists of finding, for a set of equally vertically spaced scans, the coordinate of the first white pixel that is found on a right to left scan. The output of this procedure is a vector of points that are on the right border of the line. Then the segment defined by each two consecutive border points is computed and the orientation

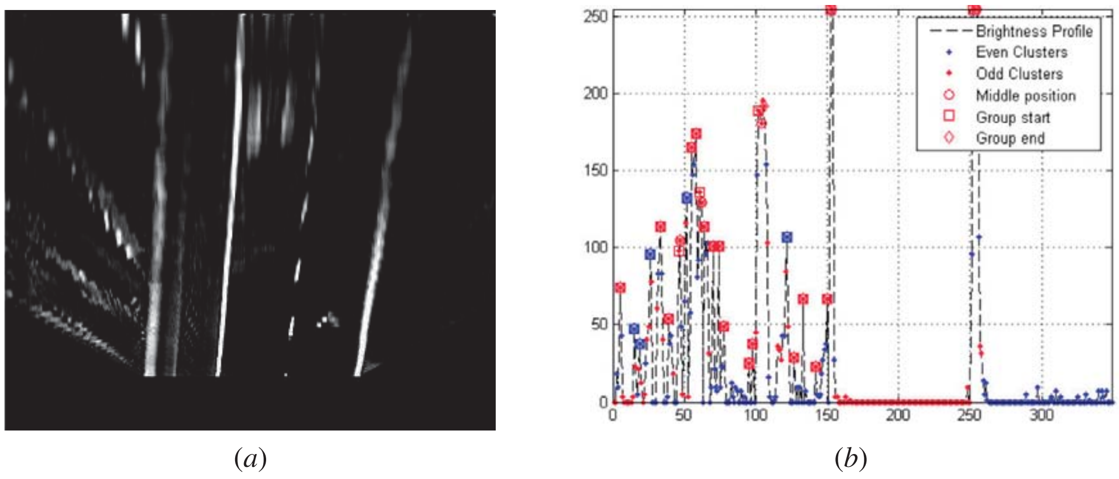


Figure 3.29: (*a*) the result of a top hat operation on the IPM image; (*b*) the brightness profile generated after the top hat image.

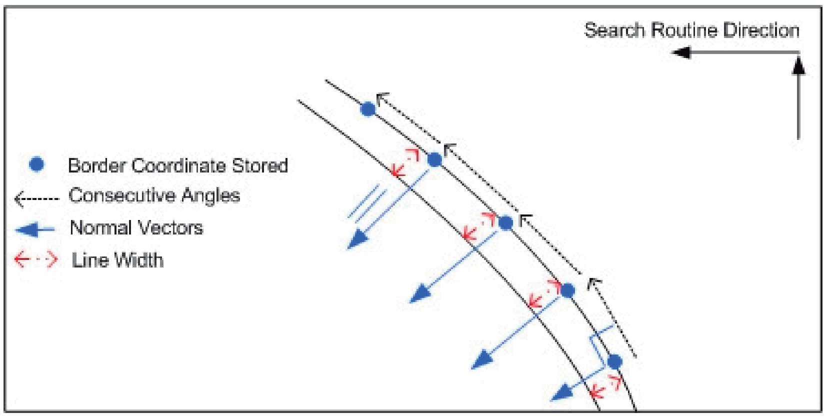
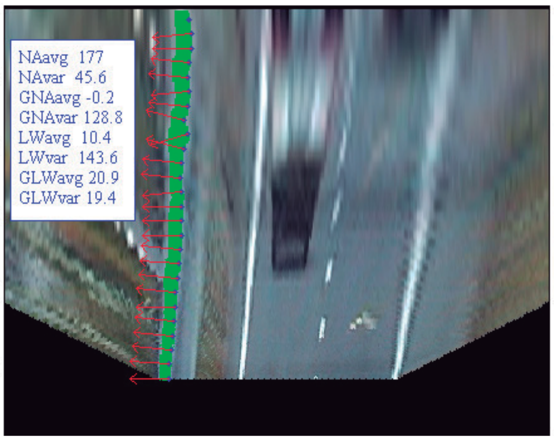


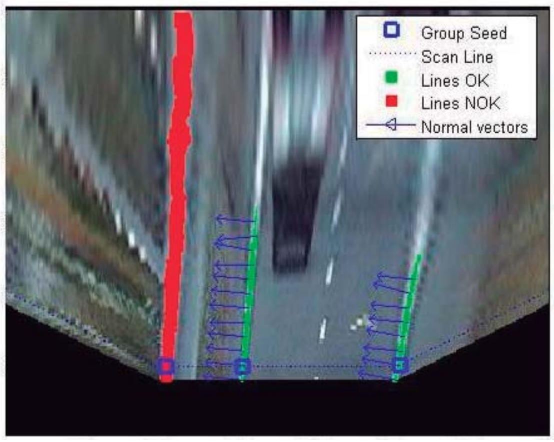
Figure 3.30: A diagram of the search routine that in ran over each candidate group.

angles are calculated. The vectors normal to those directions (angles) define the line orientation trend throughout the image. Finally, for each normal vector, a correspondent one-pixel wide line segment is defined up to the adequate image limits and its intersection with the Boolean mask is computed, providing an indication of the line width at each point. Figure 3.30 shows a diagram of the search routine.

The final stage of the algorithm is to compute a large set of statistics from the description of each line candidate, and to compare them with standard values, thus deciding whether or not the candidate line is validated as a line. One example of a relevant statistic is the average standard deviation of the width of the line. A lane marking does not change its width. Hence, there should be a typical value for the average that should match the standard size for the line's width, and the standard deviation should be small, since the line's width does not change significantly. Figure 3.31 (*a*) shows one of the

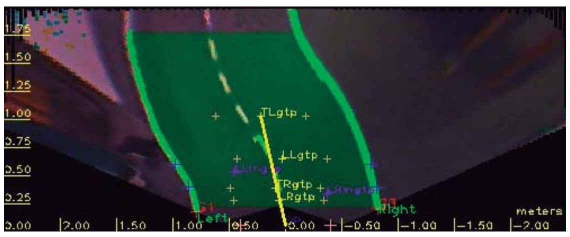


(*a*)

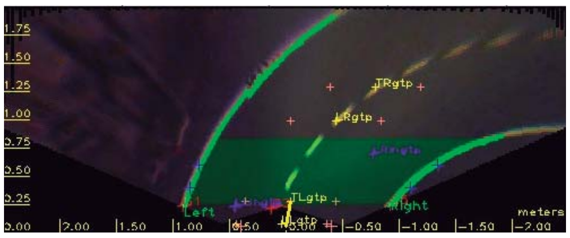


(*b*)

Figure 3.31: Testing a line candidate: (*a*) computing several statistical descriptors; (*b*) testing a candidate (green accepted as lane marker, red discarded).



(*a*)



(*b*)

Figure 3.32: Lane marker detection on the ADC: (*a*) a left turn; (*b*) a right turn.

candidate lines of the image, and the values of the statistical descriptors computed for that line (for further details on these descriptors see [Oliveira & Santos 2008c]). Figure 3.31 (*b*) shows three lines: the two signaled in green are considered lane markers, while the one in red is discarded.

This algorithm was used in the *AtlasMV* robot in 2008 and 2009. Figure 3.32 shows the results of the line detection algorithm in the *AtlasMV*. The algorithm was also tested in images of real roads with satisfactory results. Figure 3.33 shows some results of the algorithm in these cases.

3.5.3 Road Maintenance Area

One of the most challenging tasks of the ADC is to deal with the road maintenance area. This area is defined by a set of road maintenance cones, coloured orange and white (see Fig. 2.6 (*c*) and Fig.2.7 (*e*)). Because the cones have a white stripe in the middle, it is not possible to segment them completely in the image by using a simple color segmentation. As a consequence, the road maintenance navigation algorithm must be able to cope with partial cone detection. Two algorithms

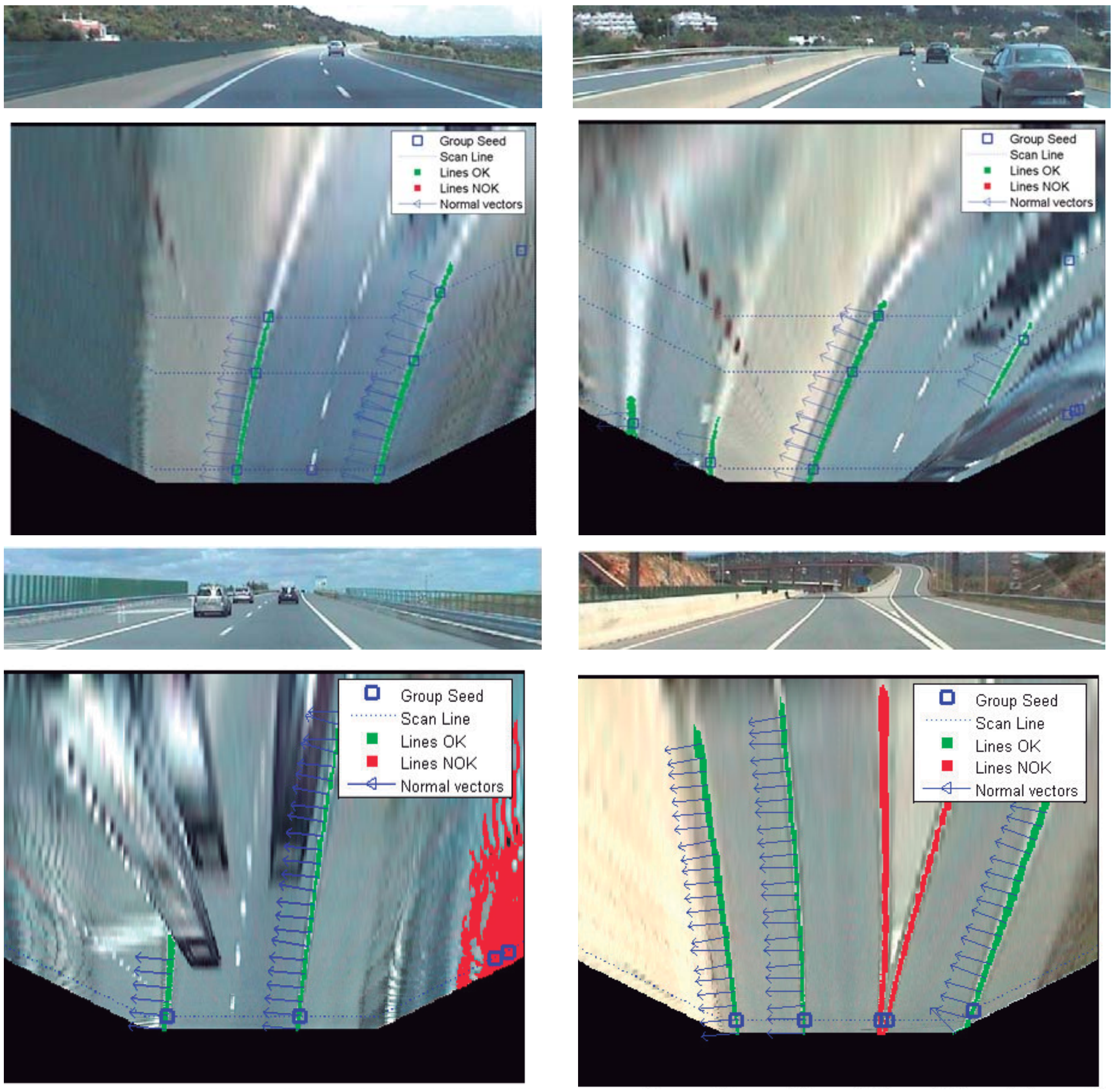
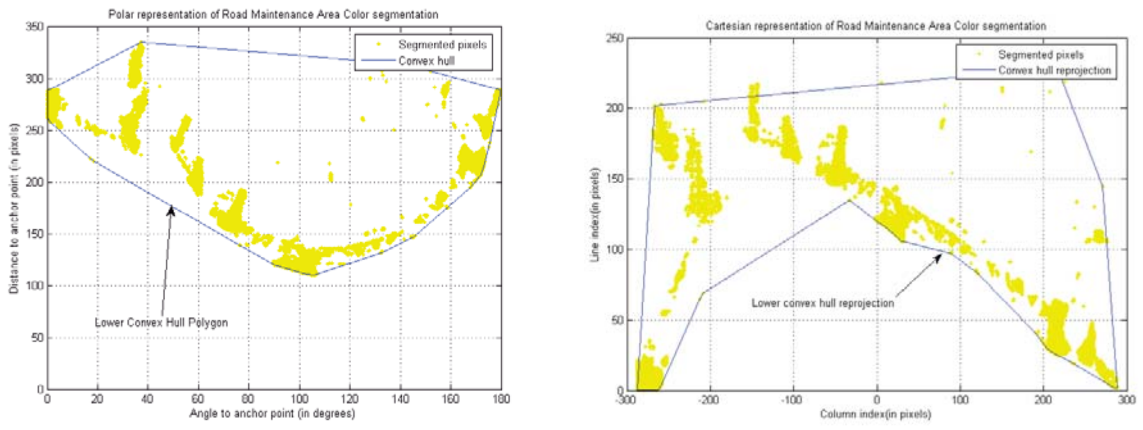


Figure 3.33: Examples of the lane marker detection in images from real roads.

have been developed to address this problem: one that uses the original perspective deformed image (non metric sensor fusion) [Oliveira & Santos 2011], and another that takes advantage of the geometry corrected top view image of the road (metric sensor fusion).

The main problem is that, after orange color segmentation, the base of the cones appear unconnected from top of the cones. Because of this, a simple search from left to right, for segmented orange, will not keep a geometrically accurate representation of the road when regions of the top of the cones are found. To solve this, a method that discards the top region of the cones was implemented. It is assumed that the robot is lying inside the road, that is, that it is in between the path defined by the cones. A polar transformation of the orange segmented pixels is computed, using as anchor the as-



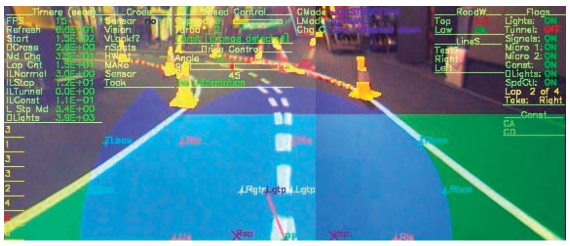
(*a*) (*b*)

Figure 3.34: Road maintenance area detection: (*a*) polar representation of the orange segmented mask (in yellow) and convex hull (blue); same information reprojected to Cartesian space.

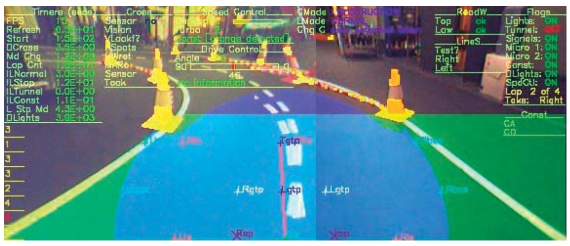
sumed robot position (the middle bottom of the image). The polar representation of the colour mask is shown on Fig. 3.34 (*a*). The next step is to compute the convex hull of that polar representation (Fig. 3.34 (*b*)). The lower polyline of the convex hull always stops at the base of the cones. By extracting this line and then converting it back to Cartesian coordinates, it is possible to obtain a representation of the base of the cones. The region that is bounded by the reprojected convex hull is an obstacle free region, which the robot uses to plan the motion. Figure 3.35 shows a sequence where the robot navigates through the road maintenance area. The detected obstacle free region is shown in blue.

A second algorithm was developed to execute road maintenance area navigation from bird's eye images. The objective is to classify the segmented orange pixels as belonging to right or left side cones. To achieve this, a simple dilation-based algorithm is performed. Dilation of the colour segmentation mask is performed a certain number of times, until only two regions exist. The outcome produces a new image with two separate blobs. These blobs result from the merging, through dilation, of the colour segmented portions of the cones to the left and to the right of the robot. A radial search for the left and right masks is then executed in order to find the base of the cones. Figure 3.36 (*a*) shows a schematic of the radial search for the left side mask. Figure 3.36 (*b*) shows the detection of the left side (blue) and right side (red) regions signaling the bases of the cones.

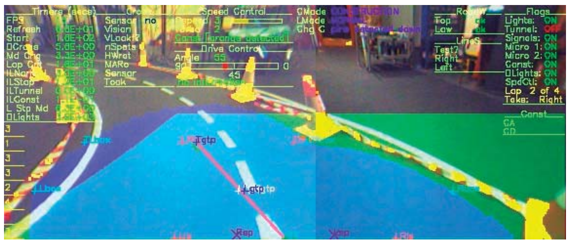
These algorithms were used until 2008. In 2009, with the inclusion of a LRF onboard the *AtlasMV*, the vision based detection of road maintenance pins was replaced by the direct LRF measurements.



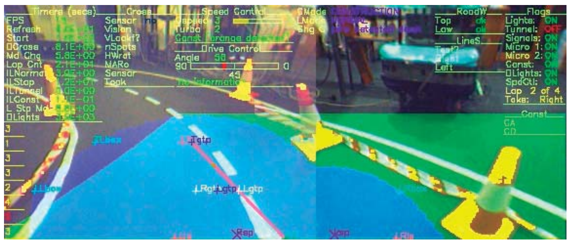
(*a*)



(*b*)

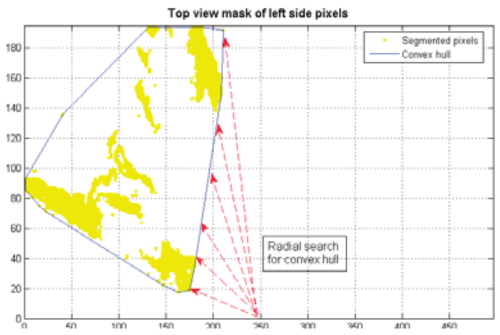


(*c*)



(*d*)

Figure 3.35: A sequence of navigation inside the road maintenance area, from (*a*) to (*b*) to (*c*) to (*d*). In the images, the obstacle free region is shown in blue and the orange detected color is marked in yellow.



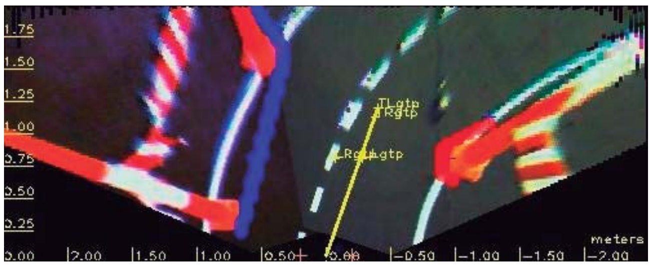


Figure 3.36: (*a*) A radial search over the left side dilated cone mask; (*b*) the detection of the bases of the cones on the left (blue) and on the right (red).

3.5.4 Path Planning

This section describes an algorithm that was used in the *Atlas* robots to execute path planning. Path planning is the task of computing where the robot should go, given a description of the scene around it. In the case of a road following robot, there should be of course a representation of the road's position with respect to the robot. Additionally, information on the position (and possibly the velocity) of obstacles is also important. The objective of a path planning algorithm is to compute a path that avoids collisions with obstacles while at the same time brings the robot to a predefined position and orientation.

In general, path planning has been thoroughly studied by the robotics community. In

[Laumond 1998] and [Ferguson *et al.* 2008b], extensive reviews on this topic are provided. However, while the state of the art in path planning in general has reached a high level of maturity, its adjustment to the problem at hand is cumbersome. In fact several details make most of the classic path planning methods unfit to tackle the ADC, namely:

- The fact that the robot travels at high speed (in relation to the scale of the scenario) demands that the path planning is fast to process;
- The reduced field of view of the robot (2 to 3 meters to the front) discards the usage of complex paths based on splines [Ferguson *et al.* 2008b] [Ferguson *et al.* 2008a], clothoids [Khosla 2002], or others;
- The fact that classic obstacle avoidance techniques such as Vector Field Histograms do not account for the non-holonomic nature of a car-like robot;
- The nonexistence of a global map (or of a large enough local one) in addition to the fact that the information present in it is sparse discards techniques based on occupation grids and similar techniques (most of the cells would have an unknown state).

Because of these specifications, we have developed a custom path planning algorithm for the robots [Oliveira *et al.* 2012]. Unlike traditional path planning algorithms, where an exact path from the starting position to the goal is defined, the proposed approach used a top down approach, in the sense that it found, from a discrete set of paths, the one that takes the robot to the goal. Obviously, this approach will generate an approximate solution. However, given the real time constraints of the competition along with the need to re-plan very frequently, this was found to be a very good solution to the problem of path planning. As discussed, this is not a path planning algorithm but more a local, short term evaluator of the best path for a robot. The algorithm starts by generating a set of possible paths using a non-holonomic vehicle model for the robot. Each path represents a possible heading of the robot. In a first stage of development, to account for simplicity and fast processing, these paths were first order curves, i.e., circumference arcs. In a second stage, composite curves were added. The algorithm builds up a snapshot of the current view of the world appropriated for subsequent processing. It does this by creating a desired path on the desired lane of the road in the form of a list of position and heading values, which are called attractor points. These points are not connected using splines or other methods. They can be viewed as beacons sparsely positioned on the lane, indicating the preferred position and heading of the robot. At the same time, laser obstacles are represented as a list of repelling points. In sum, the path planning algorithm will receive the position of the lane markers and of laser obstacles from the perception modules. Lane markers and laser obstacles are all registered in the instantaneous vehicle reference frame.

In this section, we will use two kind of descriptors for computing the optimum trajectory. These descriptors are in one case defined as a unitary vector and in the other as a 2D point. All descriptors are

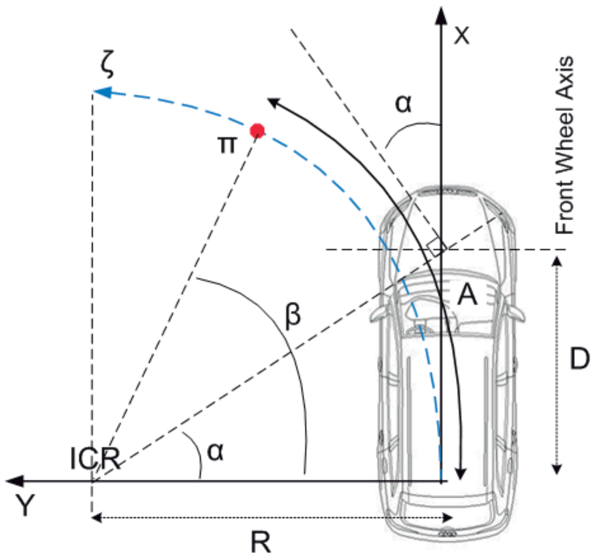


Figure 3.37: The non-holonomic model of the vehicle is used to define the paths the robot will execute as a function of the angle imposed on the steering wheels.

defined in the instantaneous vehicle reference system. Bold symbols will represent unitary vectors, i.e., $\mathbf{X} = [x \quad y \quad \theta]$, while 2D points, will be represented as $\mathtt{X} = [x \quad y]$. Throughout this section we will use the sub-indices notation to refer to the components of the vector or point. For example, $\mathbf{X}_\theta$ refers to the angle θ of vector $\mathbf{X}$, while $\mathtt{X}_y$ refers to the y component of point $\mathtt{X}$. Left super indexes will notate the trajectory to which the variable belongs to. Right super indexes notate the index of the variable.

The non-holonomic vehicle model (Fig. 3.37) is used to generate a set of possible trajectories for the robot. Each trajectory or path is defined by a set of trajectory nodes. From the observation of Fig. 3.37 it is clear that:

$$R = \frac{D}{tan(\alpha)}. \tag{3.1}$$

Hence, based on the steering direction α, it is possible to calculate the path that will be executed by the vehicle. Let ζ represent a path. The radius R of the instant center of rotation ICR. Given the arc length A, it is possible to calculate the angle β by:

$$\beta = \frac{A}{R}. \tag{3.2}$$

Let a unitary vector Π represent the cartesian coordinates that lie on path ζ. The coordinates are given by:

$$\Pi = \begin{bmatrix} \Pi_x \\ \Pi_y \end{bmatrix} = \begin{bmatrix} R \cdot sin(\beta) \\ R - R \cdot cos(\beta) \end{bmatrix}, \tag{3.3}$$

combining equations (3.1), (3.2) and (3.3) results in:

$$\begin{bmatrix} \Pi_x \\ \Pi_y \end{bmatrix} = \begin{bmatrix} \frac{D}{tan(\alpha)} \cdot sin\big(\frac{A \cdot tan(\alpha)}{D}\big) \\ \frac{D}{tan(\alpha)} \cdot \Big(1 - cos\big(\frac{A \cdot tan(\alpha)}{D}\big)\Big) \end{bmatrix}. \tag{3.4}$$

For paths steering to the right, i.e., when $\alpha < 0$, the value of Π_y becomes symmetric of what is given in (3.4). The expression is only valid for $\alpha \in [-90, +90]$, which is perfectly adequate for most of the vehicles. Equation (3.4) is used to generate all paths and path nodes. The planner generates a set of paths, each is referred to as ${}^{j}\zeta$. For computation simplification purposes, each path is segmented into linear pieces and represented by a set of m nodes, notated as ${}^{j}\Pi \in \{{}^{j}\Pi^0, {}^{j}\Pi^1, ..., {}^{j}\Pi^m\}$. The discrete array of paths and their nodes are defined by setting values for the number of paths, the wheel angle of the first path α_0 and the angular spacing between paths ($\Delta\alpha$). Hence, the steering angle of a path ${}^{j}\zeta$ will be defined by:

$${}^{j}\alpha = {}^{0}\alpha + j \cdot \Delta\alpha. \tag{3.5}$$

The number of nodes for each path is defined by specifying the number of nodes and the arc length (A) between consecutive nodes. The *ith* node ${}^{j}\Pi^i$ will have an associated arc length segment given by:

$${}^{j}A^i = i \cdot A. \tag{3.6}$$

Applying ${}^{j}\alpha$ and ${}^{j}A^i$ into equation (3.4) provides the (x, y) coordinates of all path nodes. To determine the orientation of the node we compute the angle formed between the line segment defined by the current and the next node, and the vertical direction. Hence, the orientation of node i, ${}^{j}\Pi^i_\theta$ is given by:

$${}^{j}\Pi^i_\theta = atan\left(\frac{{}^{j}h\Pi^{i+1}_y - {}^{j}\Pi^i_y}{{}^{j}\Pi^{i+1}_x - {}^{j}\Pi^i_x}\right). \tag{3.7}$$

Figure 3.38 shows an example of a set of paths and their nodes. Although the trajectories, the number of nodes in each trajectory, the $\Delta\alpha$ and *A* may change in runtime, they are considered equal for all paths, that is, all paths are equally spaced in wheel angle and will have the same number of nodes, which are in turn equally spaced by a constant arc length. Paths and their nodes are uniformly distributed across the provided limits. The methodology here proposed allows the user (or other higher level processes) to easily generate a large set of trajectories, using these small number of parameters. Different road scenarios have distinct requirements in terms of possible paths. For example a highway scenario, where a vehicle is traveling at high speed surely requires long trajectories to be tested, but the angular span of the vehicle's steering wheel is limited: in this case, a large value for *A* and number of nodes would generate long trajectories, while a small number of trajectories with

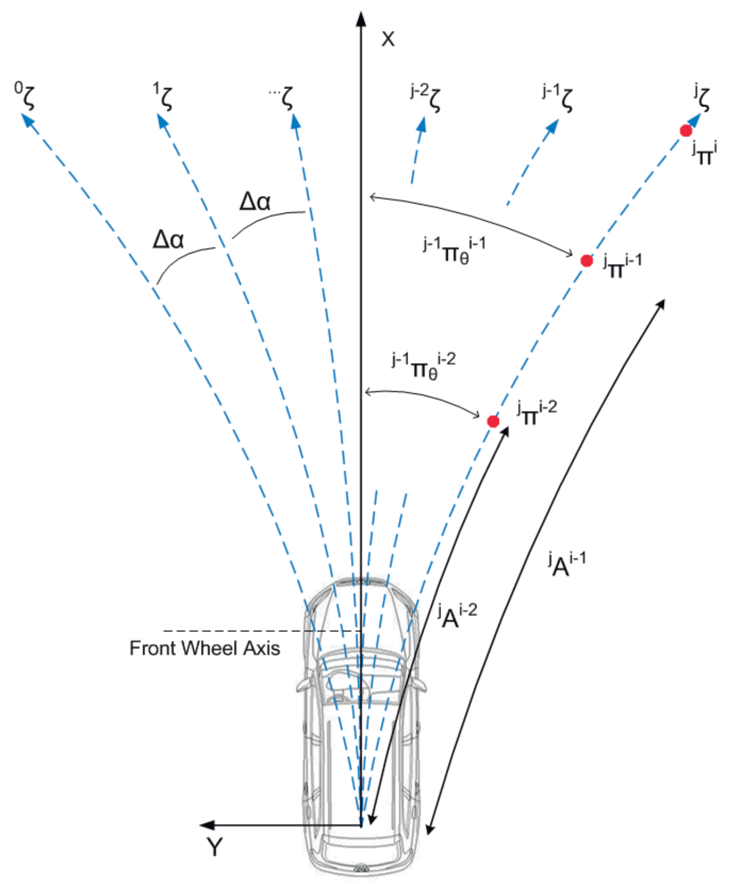


Figure 3.38: An example of a set of paths defined using equally spaced wheel angle and arc length between nodes.

the proper value for $\Delta\alpha$ would create a small angular span. In urban scenarios, small speed and hard turns are required: a small value of A will create short paths, while a large number of trajectories would provide trajectories with hard turns.

While the path planning module is executing, there are several perception modules running in parallel. When a perception module identifies a laser obstacle or the position of a lane marker, this information is is sent to the path planning module. However, the position of obstacles or lane markers must be represented in a way that is adequate for the path planning module. The conversion of the data into a suited representation generates what are called the navigation markers. They are computed from a given description of the scenario around the robot, i.e. from the information provided by the perception modules, and from a set of driving directives, which define what should be the navigation behavior. They are then used to compute several scores that will describe how good is a path for the current scenario. This process is an intermediate layer between the perception of the environment and the subsequent evaluation of all paths and selection of the optimum path.

There are two types of Navigation Markers: repelling and attractor points. Repelling points are

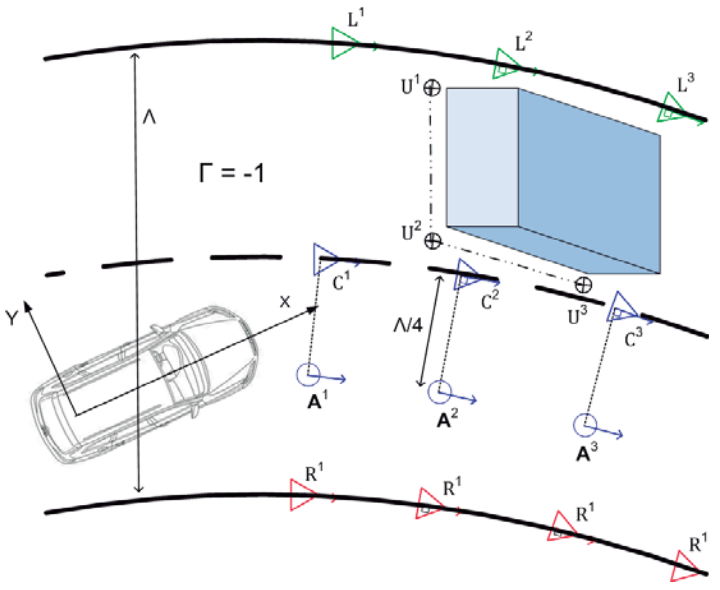


Figure 3.39: The several navigation markers for a common road scenario. The road is described by the left, central and right points, $\mathtt{L}^k$, $\mathtt{C}^k$ and $\mathtt{R}^k$, represented by the triangles in the figure. The attractor points ($\mathbf{A}^k$, represented as circles in the figure) are generated using the central lane marker points combined with the road width (Λ) and the desired lane positioning behavior directive (Γ). Laser obstacles generate repelling points ($\mathtt{U}^k$) and are represented by circles with a cross.

computed after the laser obstacles positions generated using the data reduction algorithm. They have no information regarding orientation. For a given iteration of the path planner, there will be a list of $\mathtt{U}$ repelling points, defined as:

$$\mathtt{U}^k = [\mathtt{U}^k_x, \mathtt{U}^k_y]. \quad , \forall k = 0, ..., N, \tag{3.8}$$

where N is the number of repelling points. Attractor points are, on the other hand, generated after the feature detectors that produce information regarding the road's positioning. The lane marker detection algorithms output a description of the road. This description consists of representing each of the three lane markings (left, central and right) as a list of points in the robot's reference frame. Let $\mathtt{L}^k = [\mathtt{L}^k_x, \mathtt{L}^k_y]$, $\mathtt{C}^k = [\mathtt{C}^k_x, \mathtt{C}^k_y]$ and $\mathtt{R}^k = [\mathtt{R}^k_x, \mathtt{R}^k_y]$ represent the lane marking descriptions. While the repelling point's position is obtained directly from the obstacles present in the laser scans, the location of the attractor points is dependent on the driving directive. The path planner module is informed constantly by another higher level process of the desired lane positioning behavior (Γ). This directive is defined as: $\Gamma = -1$, to drive on the left lane, $\Gamma = 0$, to drive on the center of the road and $\Gamma = 1$, to drive on the right lane. Since all the lane marker detection modules produce a standardized description of the road, the information of the left central and right lane markers is for now redundant. Currently, we use the central lane marker's description $\mathtt{C}$ to generate the attractor points. Let the unitary vector **A** notate the attractor points. Using the directive Γ combined width the road width Λ (assumed to be known a priori) the attractor points are defined as:

$$\mathbf{A}^k = \begin{bmatrix} \mathbf{A}_x^k \\ \mathbf{A}_y^k \\ \mathbf{A}_\theta^k \end{bmatrix} = \begin{bmatrix} \mathrm{c}_x^k + \frac{\Gamma \cdot \Lambda}{4} \cdot cos(\mathbf{A}_\theta^k) \\ \mathrm{c}_y^k + \frac{\Gamma \cdot \Lambda}{4} \cdot sin(\mathbf{A}_\theta^k) \\ atan\left(\frac{\mathrm{c}_y^k - \mathrm{c}_y^{k-1}}{\mathrm{c}_x^k - \mathrm{c}_x^{k-1}}\right) \end{bmatrix}. \tag{3.9}$$

Figure 3.39 depicts the Navigation Markers in a typical right turn scenario.

Once Navigation Markers are generated after the perception and several possible paths have been computed, the goal now is to find which of those paths is the more adequate for the current scenario. To perform this task, several evaluation functions Ω are employed. Each evaluation function returns normalized score $\hat{\Omega}$ that ascertains how well the path suits the criteria evaluated by the function. There are four evaluation functions, which will be described bellow. In many of the following evaluations it will be necessary to compute, for a given trajectory node Π^i, the closest navigation marker: for example, what is the closest attractor point to a certain trajectory node. Let function $map(i)$ be the function that retrieves the index k of the closest navigation marker to node i. It is defined as:

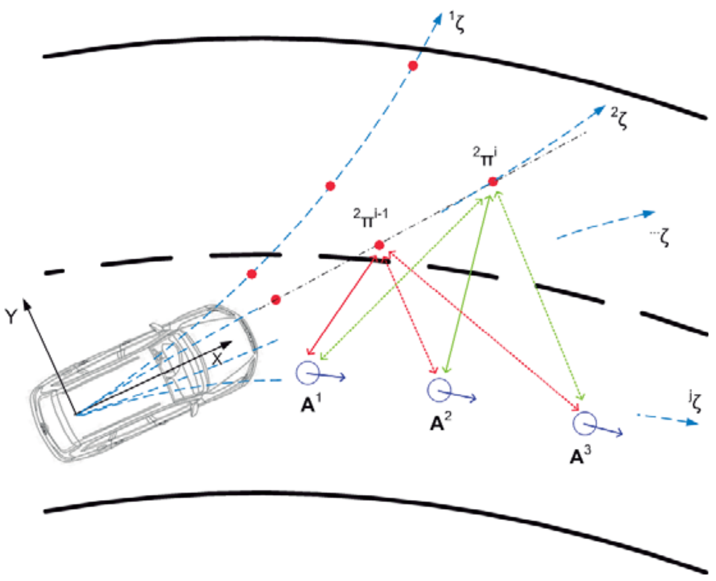


Figure 3.40: An example of the calculation of Ω^1 for nodes $^2\Pi^{i-1}$ and $^2\Pi^i$. For each node, the minimum distance to all attractor points is selected (represented with a solid line). Then the average of these minima is calculated. In this figure the Ω^1 criteria will have the following score arrangement: $\hat{\Omega}^1(^j\zeta) > \hat{\Omega}^1(^{\cdots}\zeta) > \hat{\Omega}^1(^2\zeta) > \hat{\Omega}^1(^1\zeta)$.

$$map(i) = argmin_k \left(\sqrt{({}^j\Pi_x^i - \mathbf{A}_x^k)^2 + ({}^j\Pi_y^i - \mathbf{A}_y^k)^2} \right). \tag{3.10}$$

Ω^1: Average distance to attractor points

This criteria measures a path's average distance to the attractor points and evaluates how close a path is to the attractor points. A perfect score means that the path is coincident with all the attractor points. For each path ${}^j\zeta$, the criterion is calculated as a function of the average distance between all path nodes and the closest attractor point to each node:

$$\Omega^1({}^j\zeta) = \frac{\sum_{i=0}^{N} \left(\sqrt{({}^j\Pi_x^i - \mathbf{A}_x^{map(i)})^2 + ({}^j\Pi_y^i - \mathbf{A}_y^{map(i)})^2} \right)}{N}, \tag{3.11}$$

where N is the number of trajectory nodes. The normalized evaluation score $\hat{\Omega}^1$ is obtained using the maximum admissible value as a normalizing factor. In our implementation, this value is set as the road's width Λ:

$$\hat{\Omega}^1({}^j\zeta) = 1 - \frac{max\Big(\Omega^1({}^j\zeta), \Lambda\Big)}{\Lambda}. \tag{3.12}$$

Figure 3.40 shows an example of the Ω^1 calculation.

Ω^2: Average angular difference to attractor points

This criterion measures the path average angular difference to the attractor points (Ω^2) and evaluates how compliant the path is to the attractor points heading. A perfect score means that the path drives the robot along the direction defined by the attractor points. For each path ${}^j\zeta$, the criterion is calculated as follows by measuring the average angular difference between each node's orientation and the closest attractor point's orientation:

$$\Omega^2({}^j\zeta) = \frac{\sum_{i=0}^{N} |({}^j\Pi_\theta^i - \mathbf{A}_\theta^{u(i)})|}{N}, \tag{3.13}$$

where N is the number of nodes of the path. To obtain the final score for this evaluation a normalization is computed using the maximum admissible angular difference. In our case, we set this value to 180 degrees:

$$\hat{\Omega}^2({}^j\zeta) = 1 - \frac{max\Big(\Omega^2({}^j\zeta), \pi\Big)}{\pi}. \tag{3.14}$$

Figure 3.41 depicts how $\Omega^2({}^j\zeta)$ values are computed.

Ω^3: Average laser obstacle clearance

This criterion measures a path's average distance to the laser obstacles repelling points (Ω^3). It evaluates how close path ${}^j\zeta$ is to the laser obstacles repelling points $\mathtt{U}^k$:

$$\Omega^3({}^j\zeta) = \frac{\sum_{i=0}^{N} \mathbf{f}(i)}{N}, \tag{3.15}$$

where N corresponds to the number of nodes and $\mathbf{f}(i)$ is a function that avoids local minima that typically occurs in the middle of two obstacles, the minimum distance from each path node to all repelling points is compared to a predefined variable that defines the saturation threshold for the obstacle clearance Φ:

$$\mathbf{f}(i) = \begin{cases} \mathbf{d}(i), & if \quad \mathbf{d}(i) > \Phi \\ 0, & otherwise \end{cases}, \tag{3.16}$$

where $d(i)$ is a function that retrieves the value of the distance from the path node i to the closest repelling point:

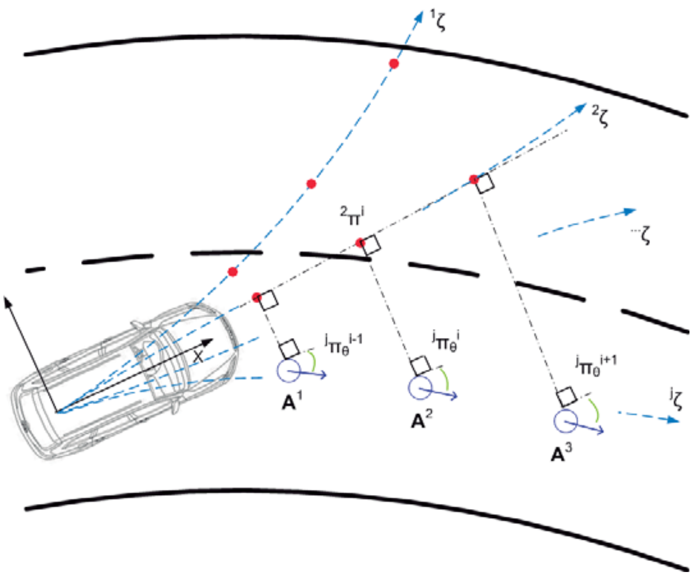


Figure 3.41: Calculation of Ω^2 scores. For each node, the minimum angular difference to the closest attractor points is selected. Then the average of these minima is calculated. In this figure, because higher index trajectories suit better the attractor points heading, the following occurs: $\hat{\Omega}^2({}^j\zeta) > \hat{\Omega}^2({}^{\cdots}\zeta) > \hat{\Omega}^2({}^2\zeta) > \hat{\Omega}^2({}^1\zeta)$.

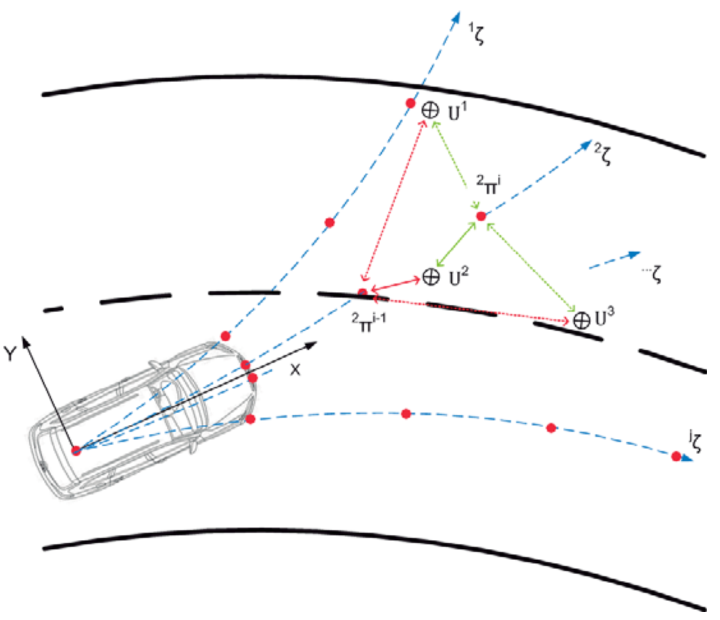


Figure 3.42: Calculation of Ω^3 scores. For each node, the minimum distance to the closest repelling points is selected. Then the average of these minima is calculated. In this figure the $\hat{\Omega}^3({}^j\zeta)$ score will be the highest of all trajectories, since path ${}^j\zeta$ is the one with best laser obstacle clearance.

$$\mathbf{d}(i) = \sqrt{\left({}^j\Pi^i_x - \mathtt{U}_x^{map(i)}\right)^2 + \left({}^j\Pi^i_y - \mathtt{U}_y^{map(i)}\right)^2}. \tag{3.17}$$

The normalized score is obtained as usual:

$$\hat{\Omega}^3({}^j\zeta) = 1 - \frac{max\left(\Omega^3({}^j\zeta), \Phi\right)}{\Phi}. \tag{3.18}$$

Figure 3.42 shows an example of the calculation of this evaluation criteria.

Ω^4: Free space

The final evaluation criteria is used to guarantee that the selected path is collision free. This is done by measuring the possible collisions of a path with the laser obstacles. In order to do so, a polyline representation of the robot for each path node is created using the width of the robot as a reference. As described, laser obstacles are defined by a set of points. Figure 3.43 shows a laser obstacle composed of three points, $\mathtt{U}^1$, $\mathtt{U}^2$ and $\mathtt{U}^3$. A set of line segments for each path node ${}^j\Pi^i$ is defined as ${}^j\Upsilon^i$. In Figure 3.43, set ${}^2\Upsilon^1$ is composed of the lines: $\overline{P_1P_2}$, $\overline{P_2P_3}$, $\overline{P_3P_4}$, $\overline{P_4P_1}$ and also of the line $\overline{{}^2\Pi^1 {}^2\Pi^2}$. The free space analysis consists of searching for intersections between the lines defined by

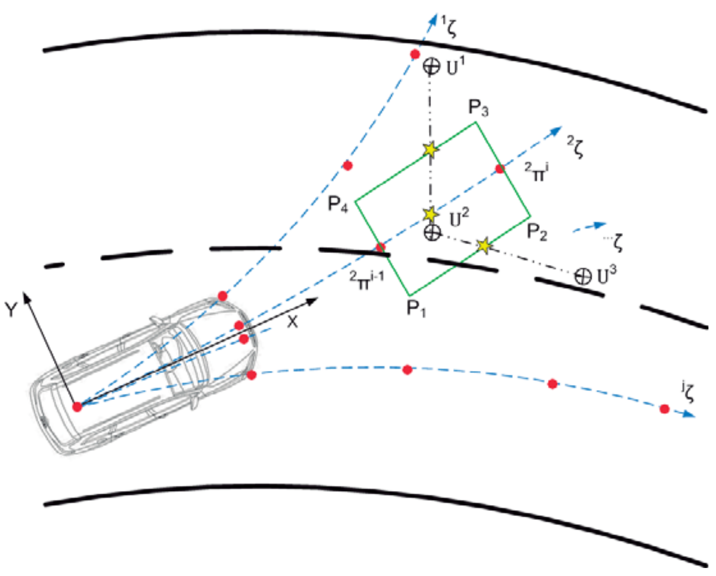


Figure 3.43: Polyline representation of the robot using the robot width as an estimate for defining several line segments. They are then tested for intersection with the lines defined by consecutive laser obstacles repulsor points U. In this case, path ${}^2\zeta$ intersects with the laser obstacle (marked by stars), which leads to a score $\hat{\Omega}^4({}^j\zeta) = 0$.

consecutive laser obstacle points with the lines in the set ${}^j\Upsilon^i$:

$$\Omega^4({}^j\zeta) = {}^jA^u, \tag{3.19}$$

where ${}^jA^u$ is the arc length of node index u (see eq. (3.6)), and u is the index of the node with maximum arc length that does not collide with an obstacle:

$$u = \begin{cases} argmax_i(\{{}^j\Pi^i\}), & if \neg intr(\overline{L_{i-1}L_i}, \overline{\Upsilon^i}) \\ 0, & otherwise \end{cases}, \tag{3.20}$$

where $intr$ is a function that tests for the intersection of the two groups of line segments and returns true if an intersection occurs. The evaluations returns maximum arc length distance that each path may accomplish before it collides with an obstacle. If the trajectory is collision free, the arc length of the most distant node is returned. Since we employ only first order paths, sometimes a path that will lead to a collision very far away from the robot may still be interesting to follow, up to a certain moment. In these cases, the path should not be discarded immediately. To handle this, the free space evaluation employs the notion of minimum safety distance Ψ which can be ascertained as the minimum arc length distance for a path where no collisions occur, in order to validate this path. The normalized value of this criteria is given as:

Table 3.1: Path planning parameters values used on the AtlasMV.

Parameter	Equation	Value on *AtlasMV*
Φ	(3.18)	110% of half the robot's width
Ψ	(3.21)	1.5 meters
w_1	(3.21)	0.25
w_2	(3.21)	0.25
w_3	(3.21)	0.5

$$\hat{\Omega}^4({}^j\zeta) = \begin{cases} 1, & if \quad \Omega^4({}^j\zeta) > \Psi \\ 0, & otherwise \end{cases} \tag{3.21}$$

The final score of a path is defined using a weighted average of the three first evaluations. In the case of the free space analysis, it is used to discard a path that will lead to a collision. We propose the following expression for obtaining a path overall score (${}^j\hat{\zeta}$):

$${}^j\hat{\zeta} = \hat{\Omega}^4({}^j\zeta) \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \cdot \begin{bmatrix} \hat{\Omega}^1({}^j\zeta) \\ \hat{\Omega}^2({}^j\zeta) \\ \hat{\Omega}^3({}^j\zeta) \end{bmatrix}, \tag{3.22}$$

where w_i is the weight corresponding to each criteria. In order to have a normalized overall score, the weights must add up to 1. The ratio between w_1 and w_2 defines the reactivity of the path, i.e., when $w_1 >> w_2$ the paths that rapidly bring the robot close to the attractor points will have better scores. The trade off is that the robot may be close to the desired position but without the desired orientation.

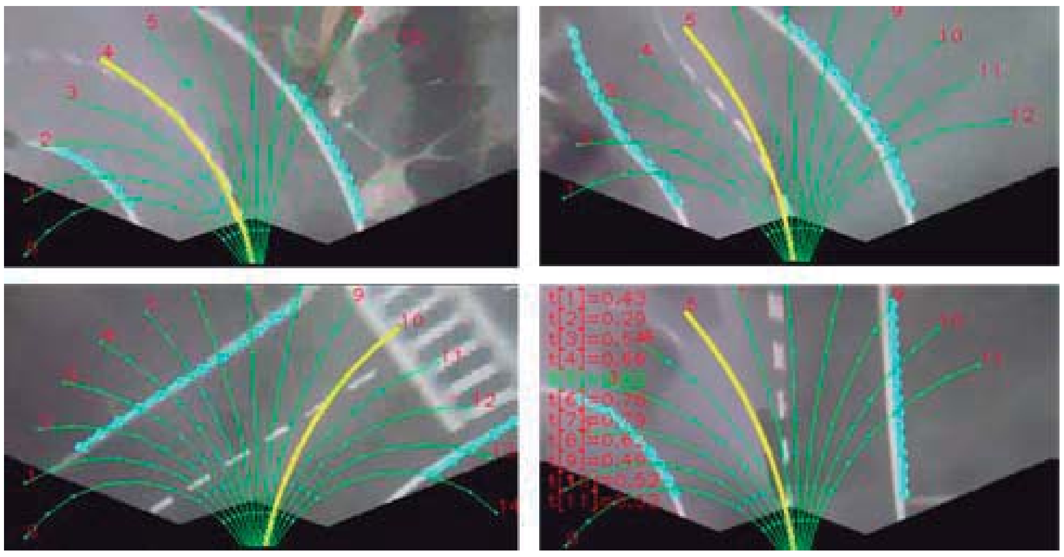


Figure 3.44: Some examples of path evaluation. The paths are represented by the green arcs. The highest scoring path is highlighted in yellow.

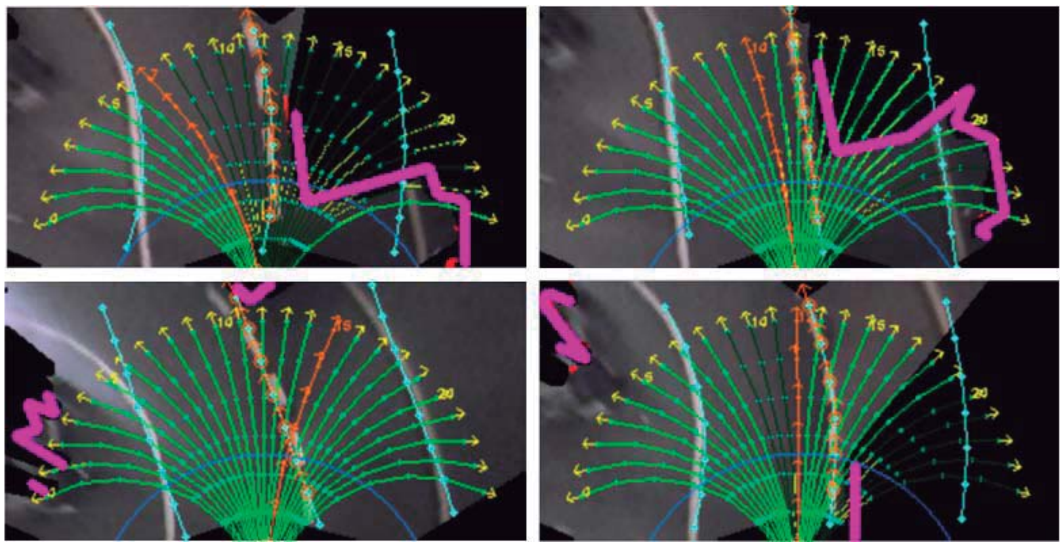


Figure 3.45: Some examples of path evaluation in obstacle populated scenarios. The paths are represented by the green arcs. The highest scoring path is highlighted in orange, and the laser obstacles are represented by the thick magenta lines.

On the other hand, if $w_1 << w_2$ then the best scored paths will have an orientation similar to that of the attractor points, and the algorithm will bring the robot smoothly to the attractor points but with more guarantees of having the desired orientation. The selection of the path to impose on the robot is obtained by finding the highest score amongst all paths. These weights where empirically tuned as presented below.

The path planning algorithm described in the previous sections was successfully applied to the *Atlas2000* and *AtlasMV*. The parameters of the motion planning algorithm were tuned empirically. The values are of course dependent on the scale of the robots as well as other characteristics such as maximum velocity or maximum wheel turn angle. However, we found that tuning these parameters is not a very difficult task. They have meaningful functionalities and clear influence on the robot's navigation behavior. This helps an user to quickly find a good compromise for the values of these parameters. In fact, the same algorithm was used for the two robots, without requiring much effort in tunning the parameters. For reference, Table 3.1 shows the values of the parameters that are used in the *AtlasMV*. Figure 3.44 shows some examples of the path planner in obstacle free environments. Figure 3.45 shows the performance of the path planner in a scenario populated by obstacles. In both cases the algorithm is able to select the most adequate path.

The algorithm has been thoroughly tested in the ATLAS robots during the ADC of the PRO. The Atlas robots have won the last six editions of the competition, i.e., from 2006 to 2011. This shows that these robots are very capable of navigating in complex scenarios. The presented path planing algorithm is a cornerstone of these capabilities. The algorithm is computed in less than 5 milliseconds (on a Dual Core 2.5GHz HP 8510p) which enables real time execution.

The algorithm was later extended to use composite trajectories, and implemented on a Robot Operating System (ROS) framework. Figure 3.46 shows examples of the paths that are typically predefined for testing in the ADC competition. Figure 3.46 (*a*) shows paths that are used during turns. Figure 3.46 (*b*) shows paths that are used to change lanes. Figure 3.46 (*c*) shows emergency paths, where the robot moves backward to gain space. Several of these paths are also used. Finally, Fig. 3.46 (*d*) shows a similar set of paths to what is shown in Fig. 3.46 (*b*), with the difference that the number of nodes per path is much higher. Figure 3.47 shows some examples of the path planning algorithms in action. The blue circle represents the attractor point. The path highlighted in green is the path that is chosen as the best path. In Fig. 3.47 (*a*) the attractor point is on the left side of the vehicle. Because of this, the path that leads the robot closest to the attractor point is the path on the left. In Fig. 3.47 (*b*), the attractor point is positioned on the right side. Since both the middle trajectory and the right side trajectory are both at the same distance to the attractor point, the criteria that defines which trajectory is chosen is the desired orientation of the robot. In Fig. Fig. 3.47 (*b*), the desired direction

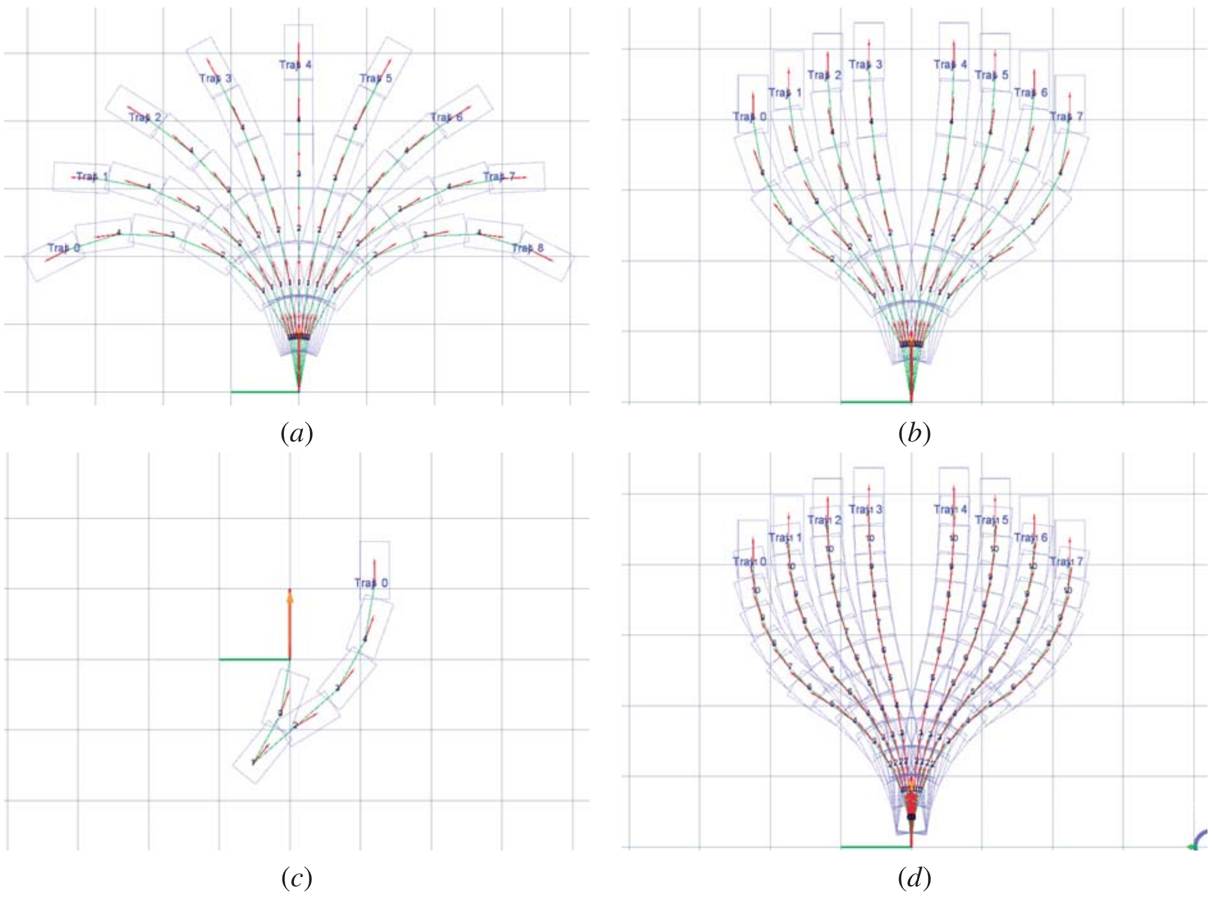


Figure 3.46: Typical predefined paths used in the ADC: (*a*) turn scenarios; (*b*) lane change scenarios; (*c*) emergency maneuvers; (*d*) same as in (*b*) but paths have twice as many nodes.

(indicated by a green arrow near the attractor point) is aimed vertically. Hence, the middle trajectory is the most adequate. In Fig. 3.47 (*c*), the attractor desired orientation is different, which makes the right side trajectory to be elected. Finally, in Fig. 3.47 (*d*) an example with an obstacle is shown. The obstacle is represented by the red lines to the front right of the vehicle. Although the attractor point still remains on the right side of the vehicle, the fact is that both the middle and right side trajectories would lead to a collision with the obstacle. Because of this, the left side trajectory, the only without collision, is selected.

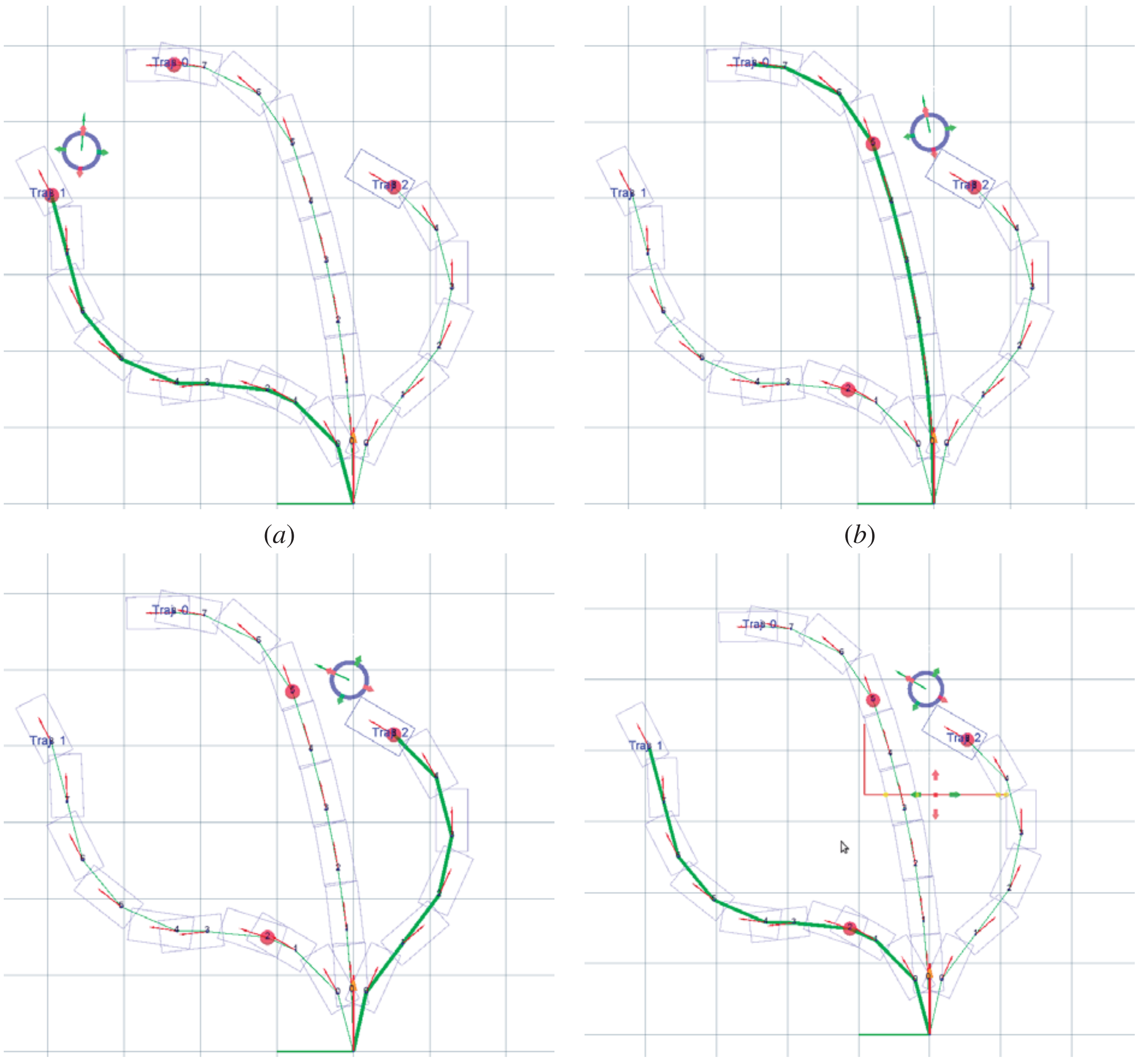


Figure 3.47: Some examples of path evaluation, attractor point is represented by the blue circle, obstacle by the red lines.

3.6 Results

Previous sections have presented the *Atlas* robotic prototypes. This section presents the results that those robots have achieved throughout the eight years of participations in the competitions. The *Atlas2000* and the *AtlasMV* have participated in the ADC since 2005. The *AtlasCar* participated in another competition in the PRO 2010 called Freebots.

Regarding the ADC, Table 3.2 lists the robots that finished in first, second and third place in the competition from 2001 to 2012. As can be seen some of the most relevant Portuguese institutions have participated in the competition. It is also interesting to note that, during the twelve years of the history, the competition has always been won 11 times by a team from the University of Aveiro. The exception is the year 2012, where a private team has achieved first place. The Atlas robots entered competition in 2003, with the *AtlasII*. This robot is not described in this chapter. It consisted of a different platform, with a single steering wheel at the front. This prototype achieved fourth place in 2003 and then third place in 2004. In 2005, the new *Atlas2000* entered the competition achieving second place. In the following six years, the Atlas robots have dominated the competition always achieving first place and inclusively achieving first and second place in 2009, 2010 and 2011.

Table 3.3 summarizes the participations of the *Atlas2000* throughout its long seven year record of participations. The *Atlas2000* robot entered competition in 2005, achieving a second place. In 2006,

Table 3.2: Robots positioned in the first three places, ADCs since 2001. Robots signaled in blue were developed by the Laboratory of Automation and Robotics, University of Aveiro.

Year		1^{st} place	2^{nd} place	3^{rd} place
2001 [9]	[PRO 2001]	Cyclop [3]	IQ 2001 [5]	Bender [8]
2002	[PRO 2002]	Capicua [3]	Prometeu [8]	Quinamawheel
2003	[PRO 2003]	Charrua [3]	MAde in Agueda [1]	Runner [2]
2004	[PRO 2004]	Made in Agueda [1]	Runner [2]	ATLASII
2005	[PRO 2005]	Made in Agueda [1]	ATLASIII	RobIEETA[3]
2006	[PRO 2006]	ATLASIV	PROJECTO VERSA [4]	ROTA2006 [3]
2007	[PRO 2007]	ATLAS 2007	ROTA 2007 [3]	GFORCE[5]
2008	[PRO 2008]	ATLAS 2008	RASTEIRINHO [5]	ATLAS MV
2009	[PRO 2009]	ATLAS MV2	ATLAS 2009	BEAGLE [6]
2010	[PRO 2010]	ATLAS MV3	ATLAS 2010	Zinguer 2010[3]
2011	[PRO 2011]	ATLAS 2011	ATLAS MV4	Zinguer 2011 [3]
2012 [9]	[PRO 2012]	Quattro[7]	Formula UM TD1 [8]	Formula UM TD2[8]

(1) Escola Superior de Tecnologia e Gestão, Águeda, Universidade de Aveiro;
(2) Laboratório de Sistemas Autónomos, Instituto Superior de Engenharia do Porto;
(3) Instituto de Engenharia Electrónica e Telemática da Universidade de Aveiro;
(4) Faculdade de Engenharia, Universidade do Porto;
(5) Instituto Superior Técnico, Universidade Técnica de Lisboa;
(6) Instituto de Educação e Desenvolvimento da Maia;
(7) Private team;
(8) Universidade do Minho;
(9) The Atlas robots did not participate in the 2001 and the 2012 competitions.

Table 3.3: Results of the participations of the *Atlas2000* in the ADCs.

	Time per round (sec)				
Year	1st	2nd	3rd	Total	Position
2005 [1]	74	140	– [2]	214	2nd
2006	56	160	108 [4]	324	1st
2007	43	141	122	306	1st
2008	51	173	197	421	1st
2009	42 [3]	158	587	787	2nd
2010	47	117	271	435	2nd
2011	86	132	136	354	1st

[1] The road scenario had a different configuration in 2005, with a shorter circuit and a single lane http://robotics.dem.uc.pt/web/.
[2] The time for this round was not available in 2005.
[3] First round all time fastest time.
[4] Third round all time fastest time.

Table 3.4: Results of the participations of the *AtlasMV* in the ADCs.

	Time per round (sec)				
Year	1st	2nd	3rd	Total	Position
2008	51	177	634	862	3rd
2009	54	176	170	400	1st
2010	45	102 [1]	130	277[2]	1st
2011	68	126	308	502	2nd

[1] Second round all time fastest time.
[2] Total of the three rounds all time fastest time.

2007 and 2008 it won the first place, achievement repeated also in 2011. It is also interesting to note that the *Atlas2000* has never finished less than in second place. Up to this day the *Atlas2000* still retains the all time record for fastest first and third rounds.

The *AtlasMV* entered competition in 2008, achieving an overall third place. Then, in 2009 and 2010, the robot won the ADC competition. Up to this day, it holds the record for fastest second round time as well as fastest overall time in the history of the competition. Table 3.4 summarizes the results achieved by the *AtlasMV*.

Finally, the *AtlasCar* has participated in the Freebots competition of the PRO in 2011, and achieved the first place (http://robotica2011.ist.utl.pt/en/competitions/freebots/).

3.7 Conclusions

This chapter presented in detail, the *Atlas2000*, *AtlasMV* and *AtlasCar* robotic prototypes. The *Atlas2000* and *AtlasMV* have participated in the ADC having achieved the first place a total of six times.

The *AtlasCar* was the first full scale autonomous vehicle created in Portugal. Additionally, it won the Freebots competitions at the PRO 2011.

The participation in the development of these prototypes provides a broad insight of the challenges of designing an autonomous driving platform. Also, the robots were used to test and validate several of the perception algorithms also presented in this chapter.

Chapter 4

Software Architectures

This chapter will discuss the software architectures used in the robots from the *Atlas* series. In section 4.2, an overview of the related work is provided. Then, section 4.3 gives a detailed description of the Laboratory of Automation and Robotics Toolkit (LARtk). The implementations of the LARtk on the *AtlasMV* robot and of the Robot Operating System (ROS) on the *AtlasCar* are presented in sections 4.4 and 4.5. Finally, conclusions are given in section 4.6.

4.1 Introduction

One of the most important components of a robot is the software. A common problem in the programming of autonomous robots is that programs tend to be very large, sometimes reaching tenths of thousands of code lines. Hence, the development and maintenance of very large codes becomes difficult. This section describes the software architectures used is the *Atlas* robots and how they were used to facilitate the development of algorithms. The first solution, implemented in 2004 on the *Atlas2000* and used until 2009, was a binary file compiled from dozens of source and header files. It was a single, monolithic code, where tasks were defined in sequence. The code had about 8 thousand lines of code, and the interdependency between software modules was very high: for example, a change in the image preprocessing stage could have unexpected results in the programs state machine. Furthermore, only one of the team members was able to rewrite or improve the code. Simultaneous development was almost impossible. Another disadvantage was that, since a single executable was running with all the required tasks, if a runtime crash occurred, the robot would loose all its functionalities. Since tasks were executed in a sequential order, a procedure for manually tunning the time each task would take was necessary and very intricate.

The solution for the maintenance and development of very large programs is not to have large programs. Keep them simple and relatively small. In 2008, a new software infrastructure was implemented on the *AtlasMV*. It was used until 2011. This software infrastructure is based on the Carnegie Mellon Robot Navigation Toolkit (CARMEN) [Montemerlo *et al.* 2003a]. Some of the functionalities of CARMEN have later been extended in the LARtk, developed at Laboratory of Automation and Robotics, University of Aveiro. A more recent software architecture was implemented on the *AtlasCar*. It is called ROS.

4.2 Related Work

The need for a modular software framework has long been viewed as necessary in the robotics community. One of the reasons is that a modular framework allows very simple code reuse. Another is that it has a very good potential for mixing real time execution with simulation. Since modules exchange information in the form of messages, it is very simple to record the output of a module by recording the messages it sent. If the recorded data is then played back, a receiving module will run just as if the other was also running. In this way, it is possible to mix runtime execution with offline simulations. There have been several attempts to come up with a standard framework, that the majority of people can use. Since the idea is to separate a robot's functionalities into several programs or modules, one of the important parts of any framework is how data is exchanged between modules. This is commonly referred to as inter process communications. Inter process communication is an extensively studied topic with broad applicability. There are several recurring themes in existing systems. Publish subscribe models are the most commonly used as described in [Newman 2003]. TCP socket communications are the most common transport. Most of these systems employ a centralized hub for message routing, although there is an exception in [Huang *et al.* 2010]. Some of the frameworks available will be described in the following lines. Special emphasis is given to the ROS toolkit, since it is nowadays a standard in robotic development and is the framework used in the *AtlasCar* prototype.

4.2.1 Lightweight Communications and Marshalling

Lightweight Communications and Marshalling (LCM) is a set of libraries and tools for message passing and data marshalling, targeted at real time systems where high bandwidth and low latency are critical. It provides a publish / subscribe message passing model and automatic marshalling / unmarshalling code generation with bindings for applications in a variety of programming languages. It was originally designed and used by the Massachusetts Institute of Technology (MIT) Darpa Urban Challenge Team as its message passing system [Huang *et al.* 2010]. LCM is designed for tightly-coupled systems connected via a dedicated local-area network. It is not intended for message passing over the Internet. It has been developed for soft real-time systems: its default messaging model permits dropping messages in order to minimize the latency of new messages. It uses UDP multicast as a low-latency but unreliable transport, thus avoiding the need for a centralized hub. Finally, the way in which LCM is perhaps most distinctive from other systems is in its emphasis on debugging and analysis. For example, while all systems provide some mechanism for delivering a message from one module to another, few provide a way to easily debug and inspect the actual messages transmitted.

4.2.2 Carnegie Mellon Robot Navigation Toolkit

CARMEN is an open source collection of software for mobile robot control . It is composed of modular software, designed to provide basic navigation primitives including: base and sensor control, logging, obstacle avoidance, localization, path planning, and mapping [Montemerlo *et al.* 2003b]. The idea is that robot tasks or functionalities are disassembled into smaller programs which are compiled and ran independently. In CARMEN, these programs are called modules. Information exchange is made through a central module that dispatches messages from one module to another, using an Inter Process Communications (IPC) [Simmons & Apfelbaum 1998]. The modules that receive the messages are called subscribers or listeners, while the ones that send messages are named publishers or servers. The advantage of a modular software architecture is that the code for each module is simple, compiled independently. The data flow is done based on standardized messages, which facilitates code development. For example, a developer writing an image processing algorithm does not have to write or even understand the code inside a camera acquisition driver module, it merely subscribes or listens to the image and processes it when a message is received. In CARMEN, messages are sent via socket connections, and are defined as a C/C++ structure in header files common both to the sender and the receiver modules. IPC links the sender with the receiver(s) of a particular message. The sender module call a send routine providing a pointer to the C/C++ structure where the message is defined. Then, the structure is parsed (marshalled) into a byte array and sent through the socket. The receiver module unparses (unmarshalls) the byte array and copies it to a similar structure, where the information is replicated. Marshalling is the process of transforming the message into a configurable easily reversible linear byte array. Messages are defined as C language structures, but before being sent must previously be transformed, i.e., marshalled, into byte arrays. To be marshalled, the format of the message structure must first be defined in a header file common both to the receiver and the sender.

4.2.3 Robot Operating System

ROS is a software framework for robot software development, providing operating system like functionality on a heterogeneous computer cluster. It was originally developed in 2007 under the name Switchyard by the Stanford Artificial Intelligence Laboratory. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It is based on a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. ROS is currently the standard robotic toolkit. Some application examples are [Zaman *et al.* 2011] and [DeMarco *et al.* 2011]. In ROS, a novel nomenclature was created to describe the information exchange and overall framework. A brief description of the framework and the nomenclature is given to ease the comprehension of the following sections. First a short description of how the code is organized and how it should be compiled and linked is given. Second, an explanation of the runtime execution, and how they transfer information is provided.

Programs are grouped into packages. A package is a collection of programs and/or libraries that perform a certain function. All the files of a package are contained by a parent directory with the name of the package. Inside the parent directory there are several subdirectories: *src*, where the source code is stored; *include*, for the headers that are to be viewed by other packages, which occurs if the package contains a library; *lib*, where the compiled libraries are stored; *bin*, which contains the binaries; and *msg*, which is the folder that contains the definition of messages. Inside the parent directory there are also a *manifest.xml* and a *CMakeList.txt* files. The first acts as a declaration of the package's dependencies of other packages. It is also where compiler and linker flags for other packages that depend on it are defined. The second file contains instructions for a standard *CMake* compilation. ROS automatically adds compiler and linker flags from other dependent packages declared in the manifest to the compilation execution. This is somewhat different from standard compilation methodologies, where a programmer that is developing a certain package must find which flags are required so that its code properly includes and links with another code. In ROS, a package that is going to be included or linked against other codes has the responsibility to declare in its manifest which are the required compilation flags and these are automatically inserted into the compilation execution of any dependent module. Hence, code dependencies are very clear and simple, which eases the task of a programmer. A *make* instruction automatically compiles the dependencies if it is required.

ROS proved to be a significant advance with respect to its predecessor, CARMEN IPC. Complex message exchange setups may be easily programmed. Deadlocks and program freezes, which occurred in CARMEN IPC, where never observed in ROS. A great number of packages are available to use in ROS, covering many topics. It also includes a very advanced visualisation package, the 3D visualization environment for robots using ROS (RVIZ).

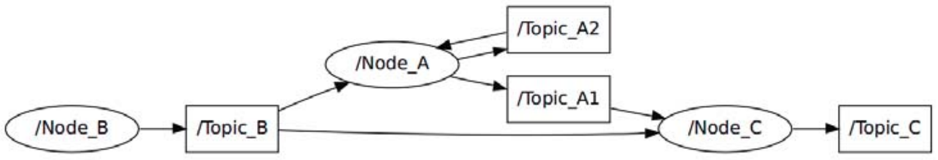
Information Exchange

In runtime execution, several binaries are executed. As discussed, in ROS binaries are grouped into packages. ROS keeps track of where the parent directories of every installed package are located. Hence, it is possible to execute binaries using a simple instruction *rosrun package binary* which calls the binary *binary* located in the binaries subdirectory of the package parent directory, i.e., */{package parent directory}/bin/*. A binary that is executed is designated as node. A node is a running instance of a binary that receives or sends information to other nodes. Note that a single binary may be executed several times and generate several nodes.

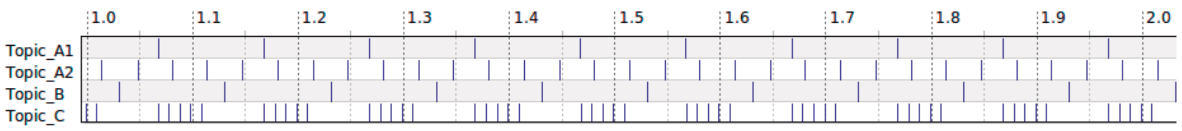
Two nodes exchange information when the first declared an advertisement of a particular message, via a topic name, and the second as subscribed to that topic name [Cousins *et al.* 2010b] [Cousins *et al.* 2010a]. A topic is an identification of a message within the runtime structure of ROS. Messages may also have different types, depending on the information they contain. The sender node is called a publisher node, and the receiver node is designated a subscriber node. To deal with the asynchronism of the nodes, ROS implements message queues both on the publisher and subscriber. Message queues are first in, first out buffers, that store messages until a node is ready to send or receive them. ROS showed a significant increase in robustness in regards to the transfer of large size messages when compared to CARMEN IPC. The message transmission mechanism involves marshalling and unmarshalling of the C++ structures from and to a byte array, like in IPC, but these operations are completely automated in ROS and are transparent for the programmer. There is a high level of flexibility and code re-usability inherent to this framework. To show this, a small but somewhat complex message exchange setup was programmed in ROS.

There are three nodes, A, B and C. Node B publishes messages on topic *Topic_B* and does not receive messages. In this sense, node *B* is a pure publisher node. Node C publishes *Topic_C* messages and subscribes to topics */Topic_A2* and */Topic_B*. Node *A* publishes two topics, */Topic_A1* and */Topic_A2*. It subscribes its own */Topic_A2* as well as a */Topic_B*. Nodes A and C may be considered hybrid publisher subscriber nodes. Figure 4.1 (*a*) shows the described setup. Now let the cycle frequencies of the nodes be 30, 10 and 20 Hz, for nodes A, B and C, respectively. Node B is programmed to publish */Topic_B* at its cycle frequency, 10 Hz. Node A publishes */Topic_A2* at its cycle frequency, 30 Hz, and is programmed to publish */Topic_A1* twenty milliseconds upon receiving a message on */Topic_B*. Node C is programmed to publish five messages on */Topic_C* in rapid succession but only when it has received at least one message from each of the topics */Topic_B* and */Topic_A1*.

Figure 4.1 (*b*) shows a record of the message traffic using this described setup. As expected, messages of *Topic_B* and *Topic_A2* occur at the respective nodes cycle frequencies. Messages on



(*a*)



(*b*)

Figure 4.1: An example of three nodes exchanging messages. *a*) the message exchange setup, with nodes signalled as ellipses and topics as squares; (*b*) a log of the message traffic for each topic. Horizontal scale in seconds.

topic *Topic_A1* are sent twenty milliseconds after a *Topic_B* is sent. Finally, five messages of *Topic_C* are sent on the set conditions. This example shows how fairly complex message exchange setups may be easily programmed in ROS, including hybrid publish subscribers and semaphores for message publishing.

Besides the publish subscribe paradigm presented before, ROS also supports a server client architecture, referred to as the server query method in section 4.3. There is, however a slight difference in the comparison. In CARMEN IPC, there was a centralized parameter server called *param_daemon*. This program served global parameters to all other nodes, which acted as clients. In ROS, the parameter server is decentralized. Every node acts as a server for its own parameters. This provides extra flexibility and the decentralized communications also improve performance.

Coordinate frames

A common issue in programming robots is how to handle the multitude of coordinates frames present in a robot. Data may be captured with respect to a certain coordinate frame, but an analysis on a different coordinate frame is commonly more interesting. This occurs quite often in sensor fusion applications. The problem is solved using geometric transformations. Geometric transformations are matrices, composed by a rotation and a translation components, that are able to transform a point or vector from one coordinate system to another. In many robotic applications, several coordinate frames are connected sequentially. For example, in a robotic arm with shoulder, elbow and wrist, the joint encoders provide transformations from the shoulder to the elbow, and from the elbow to the wrist. However, it is very common that other transformations are also required. Transformations that may be obtained from the assembly of sub transformations. In this example, the shoulder to the wrist transformation could be computed from the combination of the shoulder to elbow and elbow to wrist transformations. Geometric transformations may be assembled to generate global transformations: let ${}^{i}\mathbf{T}_{j}$ be the geometric transformation from reference frame i to j. Let three coordinate frames, i.e., $r1$, $r2$ and $r3$ be connected by transformations ${}^{r1}\mathbf{T}_{r2}$ and ${}^{r2}\mathbf{T}_{r3}$. It is possible to obtain the transformation from coordinate frame $r1$ to $r3$ with the following:

$$ {}^{r1}\mathbf{T}_{r3} = {}^{r1}\mathbf{T}_{r2} \times {}^{r2}\mathbf{T}_{r3} \tag{4.1} $$

The formulation in eq. (4.1) may be generalized for any number of coordinate frames, provided there is a tree that connects unequivocally every frame pair. In robotic applications, a second problem arises when the transformations change over time. It is the case of any any robot with moving parts. Let ${}^{i}\mathbf{T}_{j}^{t}$ be the transformation from frame i to j measured at time t. Equation (4.1) becomes:

$$ {}^{r1}\mathbf{T}_{r3}^{tq} = {}^{r1}\mathbf{T}_{r2}^{ta} \times {}^{r2}\mathbf{T}_{r3}^{tb} \tag{4.2} $$

note that eq. (4.2) is only valid if both transformations are taken at the same time, i.e., if $ta = tb$.

In this case, the resulting transformation is only valid for the same time, i.e., $tq = ta = tb$. If the sensors that measure the transformations over time are not synchronized, then, $ta \neq tb$. Therefore eq. (4.2) can only be applied when the transformation is static. Furthermore, the user might require the transformation at a particular time, one that does not correspond to any of the measurements. To solve this, interpolation techniques on quaternions are applied. Note that quaternion and geometric transformations matrices are representations that can be bidirectionally converted from one to the other. Therefore, the notation ${}^{i}\mathbf{q}_{j}^{t}$ is equivalent to ${}^{i}\mathbf{T}_{j}^{t}$, and refers to the equivalent unit quaternion to the transformation matrix. The Spherical Linear Interpolation (SLERP) [Eberly & Shoemake 2004] refers to constant-speed motion along a unit-radius great circle arc, given the ends. Let ${}^{i}\mathbf{q}_{j}^{0}$ and ${}^{i}\mathbf{q}_{j}^{1}$ be two unit quaternions, and $\mathbf{slerp}$ a function that retrieves the interpolated quaternion, given by:

$$\mathbf{slerp}({}^{i}\mathbf{q}_{j}^{a}, {}^{i}\mathbf{q}_{j}^{b}, \alpha) = {}^{i}\mathbf{q}_{j}^{a}(({}^{i}\mathbf{q}_{j}^{a})^{-1} \cdot {}^{i}\mathbf{q}_{j}^{b})^{\alpha} \tag{4.3}$$

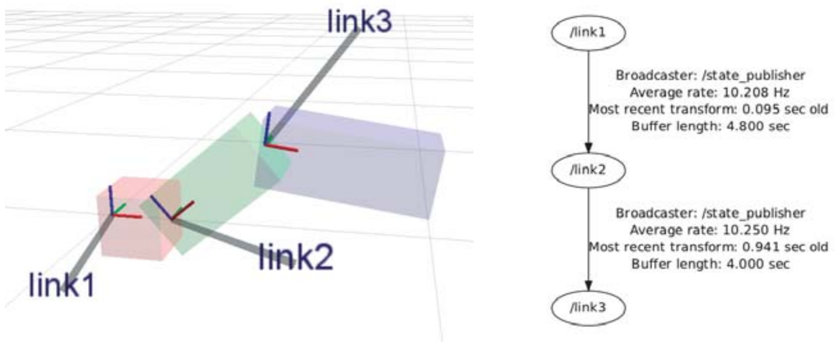
where α is an interpolation parameter, defined from 0 to 1. If $\alpha = 0$ or $\alpha = 1$ the returned quaternions are ${}^{i}\mathbf{q}_{j}^{a}$ or ${}^{i}\mathbf{q}_{j}^{b}$, respectively. With this tool it is possible to sought a transformation at a particular time, as long as at least two transformations from before and after the query time tq are stored. Equation (4.2) is adapted to:

$${}^{r1}\mathbf{T}_{r3}^{tq} = \mathbf{slerp}({}^{r1}\mathbf{T}_{r2}^{ta1}, {}^{r1}\mathbf{T}_{r2}^{ta2}, tq) \times \mathbf{slerp}({}^{r2}\mathbf{T}_{r3}^{tb1}, {}^{r2}\mathbf{T}_{r3}^{tb2}, tq) \tag{4.4}$$

where $ta1 < tq < ta2$ and $tb1 < tq < tb2$, which are the transformations that must be stored, and retrieved when an interpolation is desired. Using this methodology, the ROS *tf* package [Foote *et al.* 2012] lets the user keep track of multiple coordinate frames over time. It maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc., between any two coordinate frames at any desired query time.

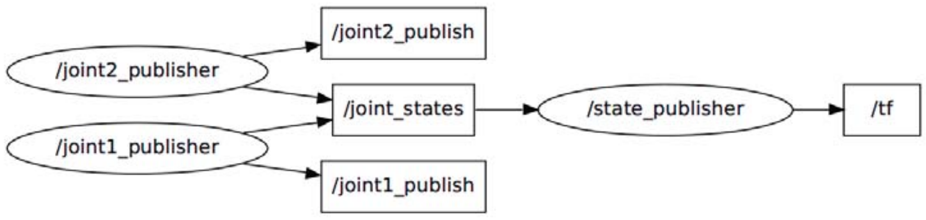
In ROS, there is a special group of messages for containing sensor data. They are called sensor messages. Sensor messages contain a header field. The header stores the time at which the message is generated as well as an identification string for the coordinate frame from where the data was collected. During runtime, several ROS nodes publish sensor data messages to other modules that are interested in those messages. Each sensor data message is attached to a particular coordinate frame. When necessary, a package may ask for a transformation between reference frames at a particular time. ROS reconstructs the connection tree using the frame signaled in the sensor message and the desired frame, and computes the sub transformations required for assembling the desired transformation. Then it collects the closest measured transformation messages for each sub transformation, uses SLERP interpolation and finally assembles the requested transformation.

Figure 4.2 (*a*) shows a small example of a robot with three links, *link1*, *link2* and *link3*. Two joints are defined: *joint1*, that connects *link1* with *link2* and *joint2*, that connects *link2* with *link3*, as depicted in the frames connection tree in Fig. 4.2 (*b*).

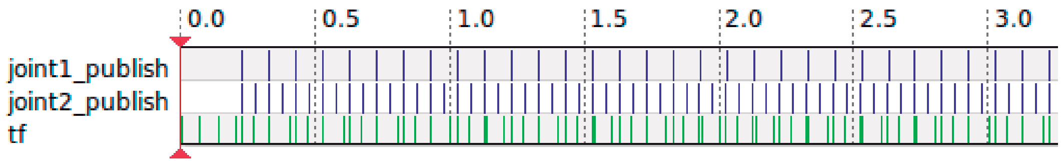

link3
link2
link1
/link1
Broadcaster: /state_publisher
Average rate: 10.208 Hz
Most recent transform: 0.095 sec old
Buffer length: 4.800 sec
/link2
Broadcaster: /state_publisher
Average rate: 10.250 Hz
Most recent transform: 0.941 sec old
Buffer length: 4.000 sec
/link3

(*a*) (*b*)

Figure 4.2: A small two joint robot for case studying the *tf* ROS package. (*a*) a scheme of the robot with the annotated links; (*b*) the connection tree of the links.



(*a*)



(*b*)

Figure 4.3: (*a*) A diagram of the nodes running; (*b*) the frequency of messages published by the nodes.

Figure 4.3 (*a*) shows a diagram of the nodes in execution. The state of each joint in published by a particular joint_publisher node. These nodes publish the current angle of the respective joint, in the *joint_states* message, as well as a second message signaling that they have published the joint state, the *joint_publish* messages. The nodes publish both messages at the same time. However, node *joint1_publisher* is publishing messages at 10 Hz, while node *joint2_publisher* is publishing at 20 Hz. A record of the traffic of messages is displayed in Fig. 4.3 (*b*). A third node, the state publisher, uses a description of the robot to compute the transformations from *link1* to *link2* and from *link2* to *link3* nodes based on the state of each joint.

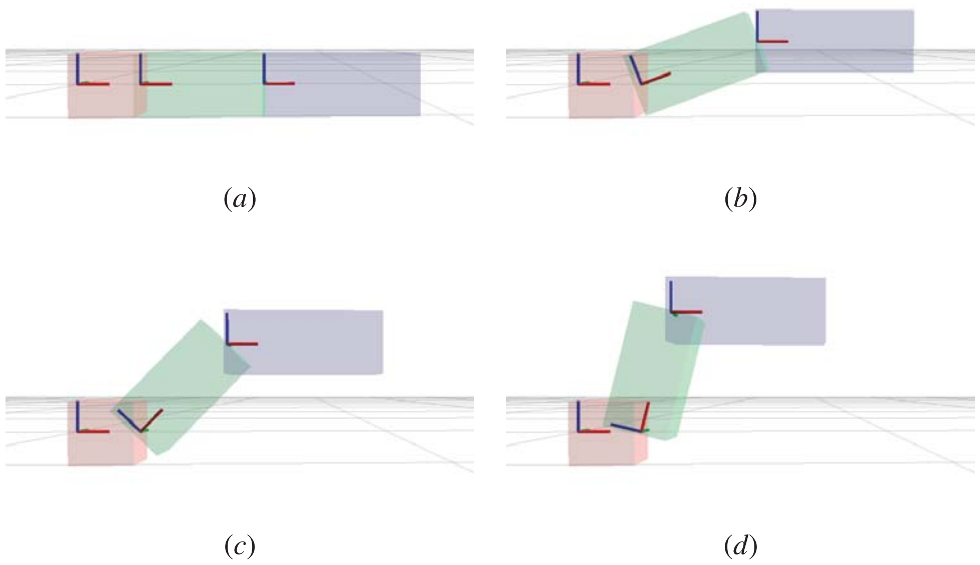


Figure 4.4: Movement of the joints: (*a*) Time 0 seconds, Joint 1 at 0 degrees, Joint 2 at 0 degrees; (*b*) Time 2 seconds, Joint 1 at 25 degrees, Joint 2 at -25 degrees; (*c*) Time 4 seconds, Joint 1 at 50 degrees, Joint 2 at -50 degrees; (*d*) Time 6 seconds, Joint 1 at 75 degrees, Joint 2 at -75 degrees;

During the test, the first joint, *joint1*, is programmed to change from 0 to 75 degrees, while *joint2* goes from 0 to -75 degrees. Figure 4.3 shows some images of the robot. Using SLERP sub transform interpolation and assembly for several query times, an animation is built that ROS is able to cope with asynchronous, partial transforms data and provides a very useful tool to handle most frame related problems in robotic applications.

Unified Robot Description File

The Unified Robot Description Format (URDF) is an XML specification to describe a robot [Meeussen *et al.* 2012]. The specifications are intended as general as possible, but there is a limitation that only tree structures can be represented, ruling out all parallel robots. Also, the specification assumes the robot consists of rigid links connected by joints; flexible elements are not supported. The URDF specification covers kinematic and dynamic description, a visual representation, and a collision model of the robot. The description of a robot consists of a set of link elements, and a set of joint elements connecting the links together. There are additional extensions to the format to handle transmissions, sensor and gazebo elements. The latest two are used for generating a robot simulation under the Gazebo 3D multi robot simulator [Koenig & Howard 2012]. The URDF of the robot is plugged into ROS as a global parameter, which can be consulted by any node. The advantage of an URDF is to have an unique, standard description of the robot, which is used both for runtime execution, simulation and visualization.

3D visualization environment for robots using ROS

RVIZ is a 3D visualization environment for robots [Hershberger & Faust 2012]. It is integrated into ROS. RVIZ can display customizable views of various types of robot data, and can show the difference between the physical world and what the robot is actually seeing. It draws a robot using the description contained in the URDF. It has several built in types of messages that can be subscribed for visualisation: 3D point clouds, camera data, maps, robot poses and inclusively it can display meshes created by other softwares (collada, stl, Ogre). Custom shapes like arrows, points, spheres or cubes are drawn using the markers special built in type, acting like a remote OpenGL node. RVIZ not only draws received messages but is also capable of providing a graphical user interface. This is done using interactive markers, which use bi directional ROS based communications between the node and RVIZ in order to let a user operate on the displayed environment.

Figure 4.5 shows a 3D view of the *AtlasCar* using RVIZ. A one Degree of Freedom (DOF) interactive marker is also visible near the roof mounted rotating laser, that is capable of setting the desired position of the laser.

4.2.4 Others

Other examples of open source robot control toolkits are Player Stage and Mobile Robot Programming Toolkit. Player Stage is a robot device server that provides network transparent robot control [Gerkey *et al.* 2003]. Player seeks to constrain controller design as little as possible. It is device independent, non locking and language and style neutral. Stage is a lightweight, highly configurable robot simulator that supports large populations. Player Stage is a community Free Software project. Mobile Robot Programming Toolkit [Blanco *et al.* 2012] provides C++ developers an extensive, portable and tested set of libraries and applications which cover the most common data structures and algorithms employed in a number of mobile robotics research areas. Key points in the design of MRPT are efficiency and re-usability of code. The libraries include classes for easily managing 3D geometry, probability density functions over many predefined variables, Bayesian inference, computer vision, SLAM, path planning and obstacle avoidance, and 3D visualization. Gathering, manipulating and inspecting very large robotic datasets efficiently is another goal of MRPT, supported by several classes and applications. The Mobile Robot Programming Toolkit was started at the University of Màlaga, Spain.

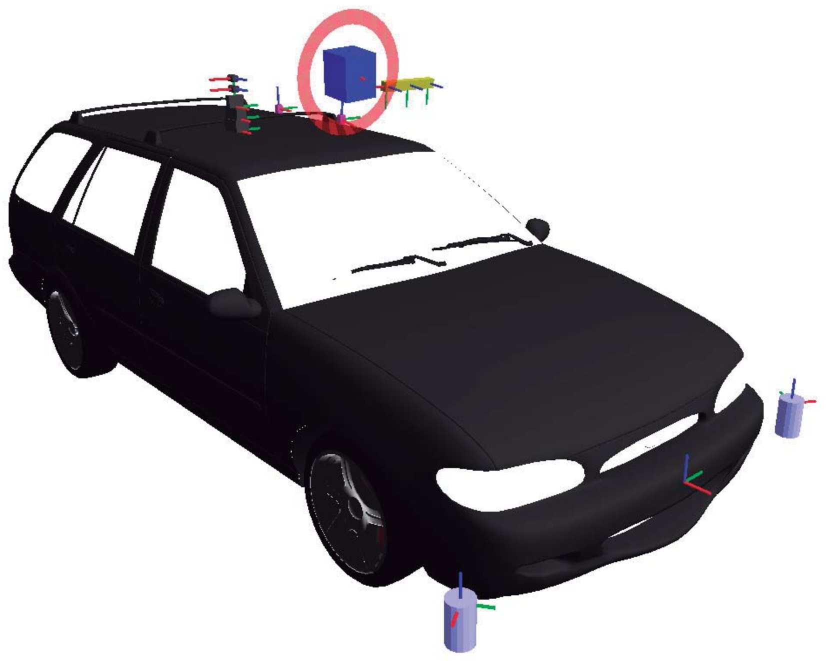


Figure 4.5: A 3D model of the *AtlasCar* and its onboard sensors visualised using RVIZ and the *AtlasCar* URDF. An interactive marker (red circle) enables the user to set the desired laser position.

4.3 Laboratory of Automation and Robotics Toolkit

The LARtk is an extension of CARMEN developed at the Laboratory of Robotics and Automation of the University of Aveiro. In LARtk, some functionalities have been added to the CARMEN package, especially concerning the transferring of large size messages. This chapter will describe the problems that existed in the original CARMEN version, and how those were solved in LARtk.

As in CARMEN, the system architecture relies on separate modules, i.e., computer processes that run in parallel. These modules were divided from the previous architecture bearing in mind that each would contain a simple task. Each module processes the information received and outputs the result. Hence, the way information must travel from one module to the other is a critical issue. Communicating with the sensors often requires constant monitoring by the process running on the computer. This new architecture uses small, dedicated modules that handle hardware and communicate with other parallel modules via IPC. The modular architecture is also more robust, because redundant parallel modules may compensate fails of others. Also, because the IPC based communication is performed through TCP/IP connections, messages can be easily exchanged between processes running in different machines. Hence, the entire program may be distributed in several computers, increasing the computational power of the robots, if so required. This is usually an important issue when running real time vision-based algorithms.

The LARtk software architecture has proven fully scalable since any modules can be added or suppressed without compromising the global operability. The encapsulation of information in predefined messages, by dividing the code into small task oriented modules and experimenting different forms of information exchange, proved to be a great improvement. The IPC framework available at the CARMEN community proved reliable except for some limitations regarding the transmission of large messages at a high frequency. This was overcome with the development of a mixed method that involved the well known shared memory intercommunication together with IPC structures and functions.

The modular architecture allows to continuously increase the system complexity without changing whatever was previously implemented. Besides this scalable nature, the resulting architecture represents a unified approach that makes it hardware-independent where each machine simply relies on small specific modules. The implementation of the LARtk described in this section will focus particularly on image efficient transfer among processes for real-time autonomous navigation.

The increasing affordability of more advanced sensors and devices induces researchers to include them on their mobile robots for more robust perception and navigation abilities. However, and due also to the disparity of standards and protocols, adding up off-the-shelf equipment along with custom designed boards or devices is not always a straightforward task. Moreover, keeping the pace in software development when there are changes in the team of programmers is often a nuisance for project managers.

4.3.1 Information Exchange

The information exchanged among modules must be classified so that a module receives only what it actually requires, and not everything else. This is accomplished by encapsulating information into messages. A module interested in some particular information can then ask to receive a specific message. On the other hand, a module that produces some specific output can periodically send a message containing a certain type of information. Message exchanging can be done in several ways. The simplest method is called publish/subscribe (Figure 4.6).

In this scenario, the publisher module generates information that is packed into a message of type "A". All modules interested in the information contained in message type "A" should subscribe to it. When that information is available, the publisher module publishes the message and it is dispatched by IPC to all modules that have previously subscribed to it. One disadvantage of this setup is that it

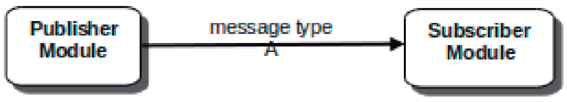


Figure 4.6: A simple publish subscribe setup.

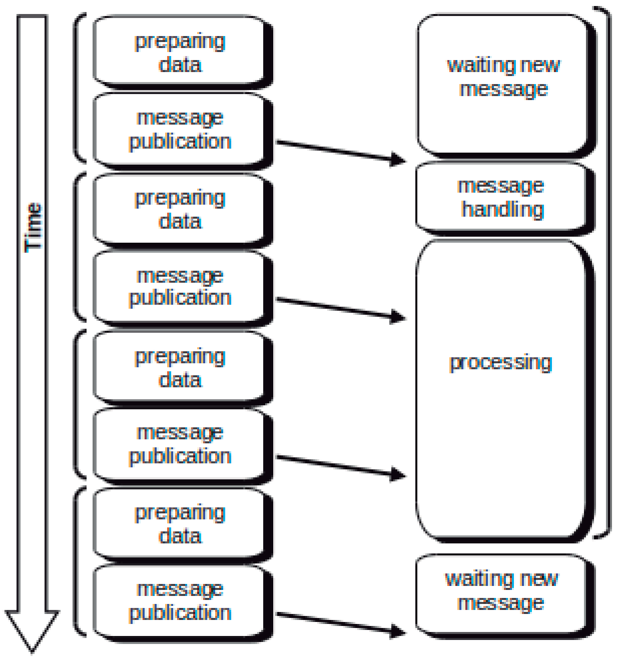


Figure 4.7: An example of a time sequence (vertical axis) of a publish/subscribe message exchange. The publishing module (on the left) has a faster cycle time (represented by the brackets) than the subscriber module (on the right). Because of this, the subscriber module receives two messages while it is still processing the first one.

is required that the subscriber module handles all the received messages. This may become critical if the publisher module has a shorter cycle time than the one of the subscriber. Figure 4.7 shows a case where this occurs.

In this scenario, since the subscriber takes very long to process the messages, there is going to be a growing queue of messages that, at some point, will overflow and cause the IPC central module to crash. This was observed in practice, especially for large messages (hundreds of kilobytes). The publish/subscribe message exchange methodology should be considered only if the subscriber module is faster than the publisher, or if the handling of the message reception performed by the subscriber is a very fast routine. For this reason, it is not advisable to send large messages containing laser scans or especially images using this methodology. The publish/subscribe method is appropriated for exchanging only small size messages or for fast processing modules. However this is not always the case. For example, the cameras installed on *AtlasMV* robot produce 320 $\times$ 240, 3 channels RGB images at 30 Hz. This means that a module performing image acquisition generates approximately 7 Megabytes of information every second. Also, the subscriber modules that perform image processing tend to be the slowest. In laboratory experiments, publishing such a large amount of information caused IPC to overload and crash during our tests. This occurred especially when the subscriber

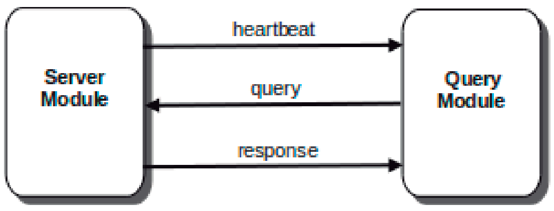


Figure 4.8: A query respond with heartbeat setup.

module was not able to handle the information flow on time. These limitations can be solved using a second methodology available in IPC: the server query paradigm. In this case, the server module will send a message only when asked for by the querier module. Hence, the frequency of message traffic is defined by the receiver module, the querier.

To support this message exchange configuration, three messages types are defined: a heartbeat, a query and a response message (Fig. 4.8). The heartbeat message is sent by the server module and indicates that new information is available. It it is a very small message, a mere notification to whomever is interested that new information is ready to be sent on request. Heartbeat messages are broadcasted by the server module, using the publish/subscribe methodology and because they are small they can be sent to many subscribers at high frequencies. The query message is also very small. It is sent by the querier module to the server in a peer to peer communication. After sending the query message, the query module waits for the response. The response message can be large (could contain an image, for example), and is sent by the server to the query module only in reply to a query. The response also works based on a peer to peer mechanism. The response message is where the actual information is sent, both heartbeat and query messages serve message traffic management purposes. Though more complex, this setup is particularly useful for transmitting large messages since that transmission only occurs when the querier module actually needs the information, reducing unnecessary message traffic. If the querier module is faster than the server module, heartbeat messages ensure that no query is done unless new information is available on the server side. The flow of messages is shown in Fig. 4.9. Heartbeat and queries are control messages that synchronize both modules so the information is exchanged only when actually needed. The complexity increase is compensated by the reduction of large message traffic. This setup should only be employed when the messages to be exchanged are large.

The disadvantage of the heartbeat/query/response method presented in Figs. 4.8 and 4.9 is that because messages are queried by a specific module, the server must send a response message for each specific query, since both the query and the response messages are implemented via a peer to peer communication. If the server module is meant to send large amounts of information to several querier modules, the message traffic would increase as many times as the number of receiving modules. The combination of the number of query modules and the message size may reach a point where the server

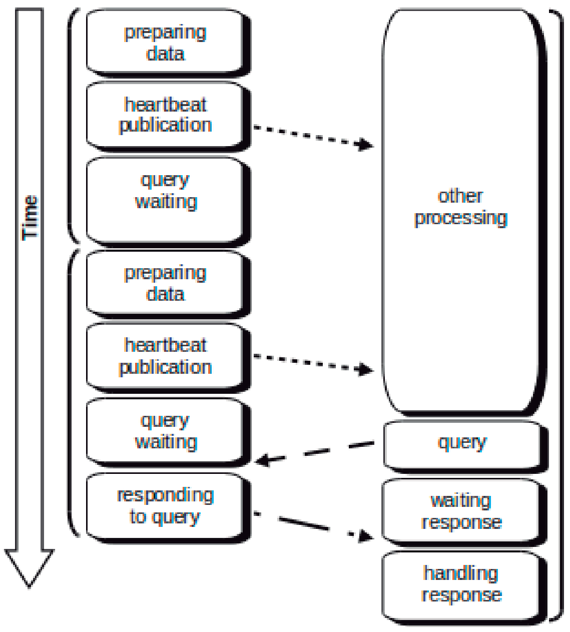


Figure 4.9: A query response time flow. Heartbeat messages (dotted line) are published whenever the server module (on the left) generates new information. The query module (on the right) queries the information (dashed line) only when it actually requires it. A query message is followed by a response message with the actual data.

module may not be able to respond to all queries on time.

To solve this particular problem, a new method has been devised using the functionalities of IPC's marshall unmarshall routines. In this particular information exchange setup, the server module allocates a shared memory segment with the size of the message, and then stores the message directly onto the shared memory by marshalling the data into the memory address. After this operation, a heartbeat message is published indicating that new information is available. The query module attaches to the memory segment and unmarshalls the information to local structure. For the query module to attach to the shared memory, the address and size of this memory must be known. For this purpose a new message is defined containing the required shared memory information. This message is queried to the server during the initialization procedures of the query module. Afterwards, whenever a heartbeat is received, the query module may, whenever it requires new information, unmarshall the data from the shared memory address and work with a copy of the data. This procedure is shown in Fig. 4.10.

A limitation of this method is that, because it is not based on TCP/IP, it does not work when the processes run on separate machines. Another glitch is that on large messages, there is some possibility that the query module may be still reading part of the message while the server is writing. This may lead to reading messages that are actually a combination of two different messages. In our particular case, since we use this setup mostly to transfer images, this is problematic. Nonetheless, this

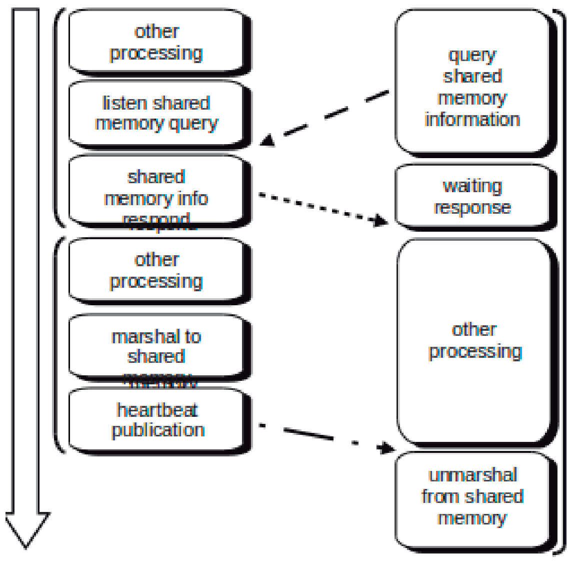

other processing
listen shared memory query
shared memory info respond
other processing
marshal to shared memory
heartbeat publication
query shared memory information
waiting response
other processing
unmarshal from shared memory

Figure 4.10: A query response time flow using shared memory. The query module (on the right) queries for the shared memory id (dashed arrow). The server module (on the left) responds with the information (dotted arrow). The query process then attaches itself to the shared memory and unmarshalls the data whenever a heartbeat is received.

method is very efficient for transferring large messages across multiple processes, since it addresses the majority of the problems present in the previous setups.

4.3.2 Data Logging and Playback

In order to allow simulation with real data, a logger/playback software was developed. The software logs IPC messages that are transmitted by the sensor modules into data files; the software was inspired in the CARMEN logger module but uses a different data storage method. Each log consists of two files, the first serves as a header file (logheader) containing only witch messages were logged and their timestamps, the second file contains the contents of the messages (logdata). Messages are first marshalled into to a byte array and then saved on the data file, the position in the file where the byte array starts is also saved in the log header file. The logger is able to log simple publish-subscribe messages and also more complicated query-server messages such as the ones exchanged through a shared memory (when efficiency is required). The playback module starts off by indexing all messages in the log header file and once playback starts it reads the necessary message data, converts it back from byte array to message format, and publishes it. Using this method, modules that subscribe the logged messages do not know that they are using logged data and not real time data, and thus allows for simulation with real data.

4.4 The LARtk on the *AtlasMV*

The LAR toolkit software architecture was implemented on the *AtlasMV* in 2009. By 2011, the robustness and efficiency of the architecture led to the implementation on the *Atlas2000*. This section describes the modules, i.e., the processes, that are usually run on the *AtlasMV*, using the LARtk infrastructure for message exchanging. The modules can be roughly divided into three categories: hardware interface, perception and planning. The first is responsible for the interaction with the hardware: data acquisition and motor command. The perception layer processes the acquired data. The last category is dedicated to the reasoning capabilities of the robots. It makes use of all the detected features and, based on the context, will plan and execute the robot's behavior.

The camera acquisition module (Fig. 4.11) communicates with IEEE 1394 Firewire cameras using the libcd1394 library. This library provides a complete high level application programming interface for developers who wish to control IEEE 1394 based cameras that conform to the 1394-based Digital Camera Specifications. All three message exchange formats were developed for the LARtk. The most commonly used is the query respond based on shared memory.

This module also controls the camera parameters. Brightness, saturation, white balance, shutter speed and others can be set in real time. The module is also capable of handing out distortion corrected images taken from wide angle lens cameras, given a chessboard calibration output. On-board the *AtlasMV* three instances of this code are running simultaneously, one for each camera.

The laser acquisition module (Fig. 4.12) acquires laser data from the laser and sends the information to other modules. The laser on-board the *AtlasMV* uses SCIP2.0 communication protocol standard. The manufacturers API was used to develop this module.

Several parameters can be set at startup: angular resolution, start/end scan angle, among others. It is also possible to use any one of the three information exchange methods.

The PTU control module (Fig. 4.13) is slightly different from the two previous ones. While the laser and camera acquisition modules only send information to other modules, the PTU control, besides sending the status of the device, also receives command messages. The PTU status messages indicate the position speed and acceleration of the joints. Command messages are used to command the PTU to a particular position. Hence, this module is capable of commanding the pan and tilt unit

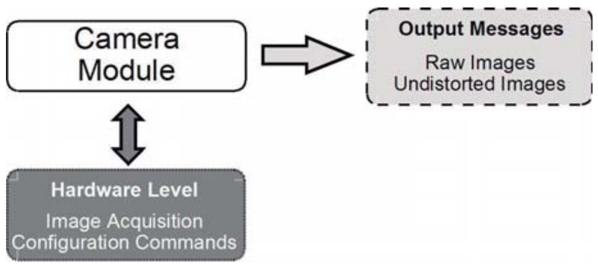


Figure 4.11: The Camera acquisition module.

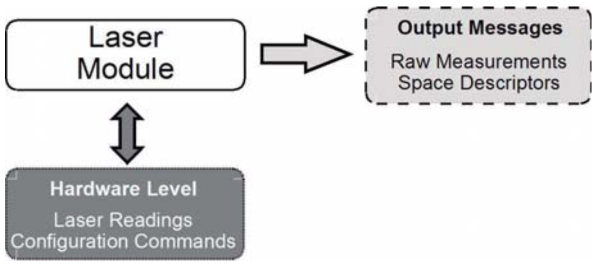


Figure 4.12: The laser acquisition module.

based on orders received from other modules.

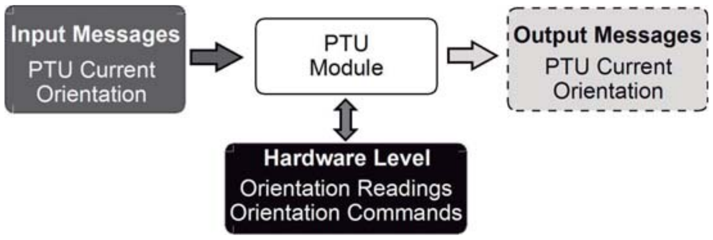


Figure 4.13: The PTU control module.

Because both the status and command messages are small, only the publish/subscribe method has been implemented. If a module requires information on the state of PTU it subscribes to the state messages. If, on the other hand, a module wishes to command the PTU position or speed, it should publish a command message. During startup, the PTU control module subscribes to the command message. It is actually possible to have different modules competing for the command of the PTU if they both send command messages. The PTU module will receive all the messages and execute them sequentially by order of arrival.

The base module (Fig. 4.14) is responsible for interfacing with the robot motors and electronics. Similarly to the PTU control, this module is also a publisher since it sends robot status messages and a subscriber because it abides to command messages.

Command messages define the steering and speed of the robot. The *AtlasMV* base module com-

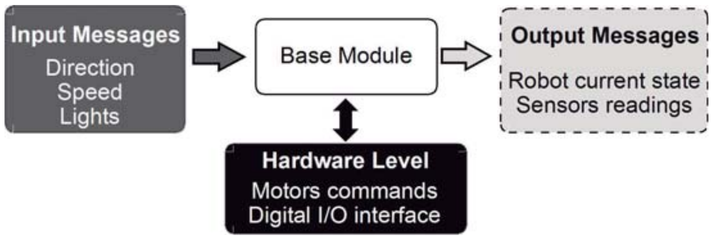


Figure 4.14: The base module.

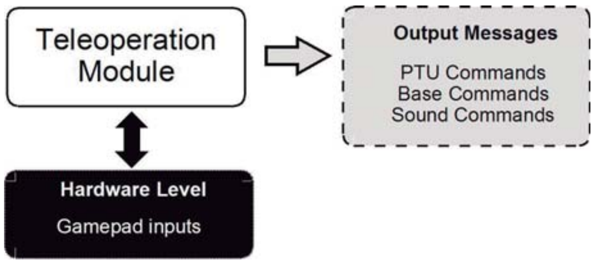


Figure 4.15: The remote operation module.

municates with robot electronics using RS232 communications protocol. It translates that information given in the command messages to the specific robot hardware communication protocols. In this way, if a different robot is used, most of the code remains unchanged, and only a different base module is written to translate command messages to hardware specific orders. This module also publishes a status message containing information about the robot current speed, steering angle, lights' state and digital inputs readings.

The remote operation module (Fig. 4.15) is a driver module for a gamepad or joystick. It allows a user to remote control the robot. Actions on the gamepad are translated to a robot command message. The module does not have any specific publication/subscription routines. It simply makes use of the routines implemented by each of the modules it wants to control. Using this module it is possible to remote control the speed, steering, lights and the Pan and Tilt Unit (PTU).

The Sound Player module (Fig. 4.16) is capable of generating audio output. For synchronization purposes, it can also inform other modules if it is busy playing a sound. Any module can require a sound message to be played by publishing a sound command message which defines identification of the media to be played. Upon receiving the command, the sound player makes use of Libao library functions to reproduce it. The sound player module indicates its status (busy or available) by publishing a status message. This module allows improved user/robot interactivity and also provides debug facilities.

The inverse perspective mapping module (Fig. 4.17) subscribes image from two cameras and the PTU status. It fuses the images from the cameras by projecting them to the ground plane. The PTU

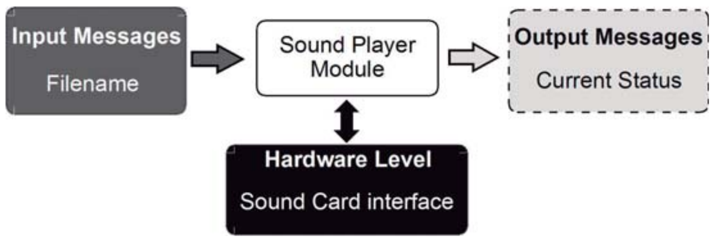


Figure 4.16: The sound player module.

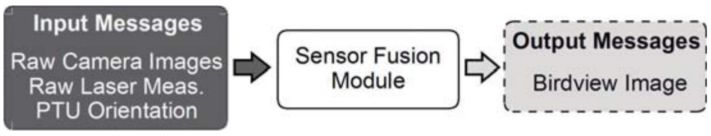


Figure 4.17: The inverse perspective mapping module.

status is used to assess the position of the cameras with respect to the ground plane: because cameras are mounted on the PTU, camera positions are a function of the PTU's orientation. In a late version, this module also subscribes to laser messages to produce a laser assisted inverse perspective mapped image. It creates a common reference representation of the measurements taken, whether they are images or range scans.

The module can fuse several images captured from multiple cameras along with laser information, generating enhanced images of the road, in a birds eye view perspective. This algorithm will be described in detail in chapter 5. Figure 4.18 shows an inverse perspective mapped image obtained by merging the images of two different cameras. This module uses a rectangular region, to define the area where the robot is interested on receiving sensory data. If one particular sensor collects information from outside this area, this information is clipped from the fused image. The ultimate goal of this task is to find a common representation for a multitude of sensor types and/or configurations. This ensures that, no matter the specific sensorial setup of a given robot, it is reshaped into a common reference. The advantage here is that subsequent modules (like feature extractors, road detectors, obstacle detectors) can rely on a constant, predefined representation of the data and work without need for reconfiguration.

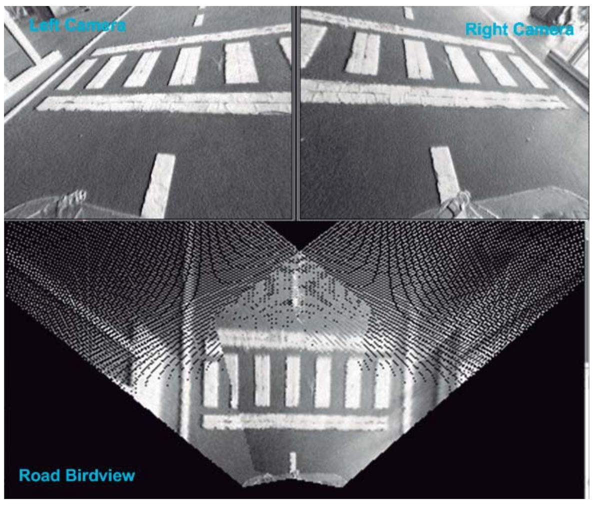


Figure 4.18: The output of the inverse perspective mapping module.

4.5 The ROS on the *AtlasCar*

For the purpose of launching multiple nodes, the ROS *launch tool* is used. *Roslaunch* is a tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the Parameter Server. It includes options to automatically respawn processes that have already died. *Roslaunch* takes in one or more XML configuration files (with the .launch extension) that specify the parameters to set and nodes to launch, as well as the machines that they should be run on. *Roslaunch* launches local processes and kills them using POSIX signals, but it does not guarantee any particular order to the startup of nodes.

For the moment, the software running on the *AtlasCar* runs on a single machine. The software nodes are organized in groups according to the functionalities they provide. It launches all modules that are related to the vehicle's electronics, as well as some core functionalities that are required to be running before other nodes are launched. For convenience, nodes and messages names are also organized using name spaces. There are several name spaces defined, for instance: */vhc* the vehicle name space is for entities related with the vehicle's electronic modules or mechanical structure; */trf* the transformations name space, which is employed where information is related to the coordinate frames of the robot.

4.5.1 Coordinate Frames

The *AtlasCar* uses the ROS framework. Because it is equipped with many sensors, several coordinate frames were defined in the vehicle. Some extra coordinate frames were defined to facilitate the measuring of transformations with respect to others, or to incorporate ego motion estimation.

The on-board coordinate frames represented in Fig. 4.19 are described bellow:

- Coordinate frames from sensors are contained in the sensor messages frame identification fields:

  /atc/laser/right_bumper, defined at the center of the Sick LMS 151 laser mounted on the right side of the vehicle;

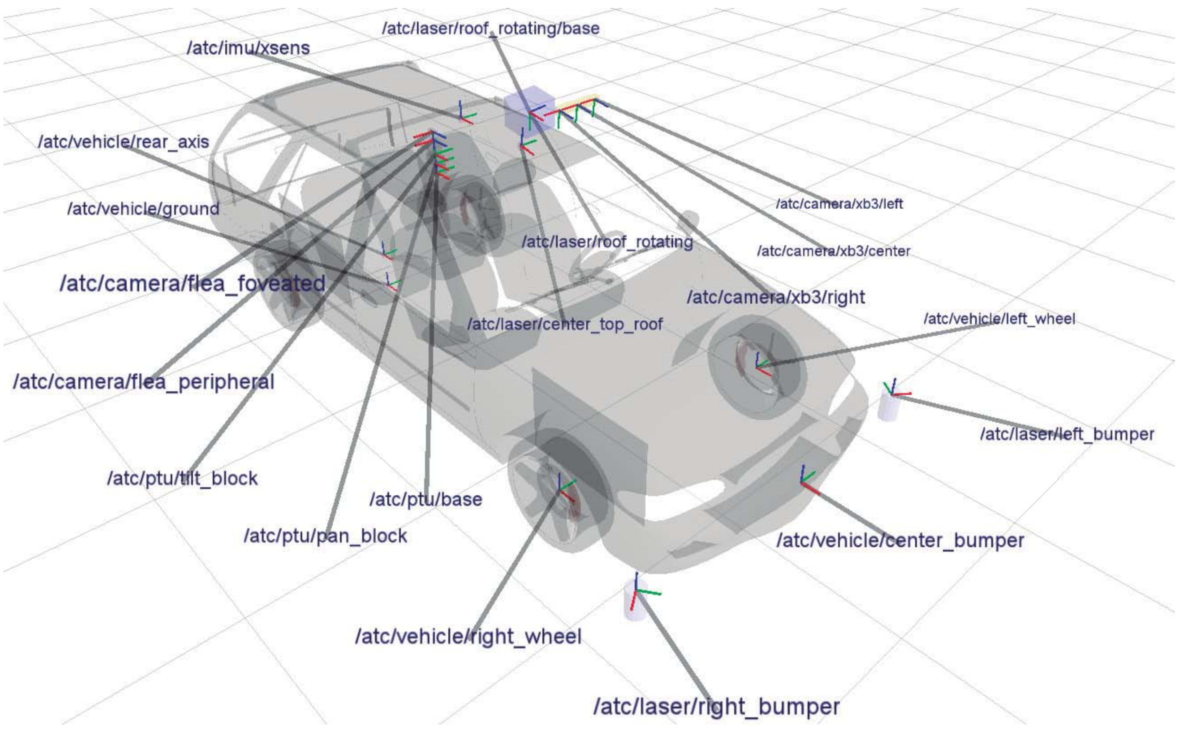
  /atc/laser/left_bumper, defined at the center of the Sick LMS 151 laser mounted on the left side of the vehicle;

  /atc/laser/center_top_roof, defined at the center of the Hokuyo UTM30LX laser mounted on the roof of the vehicle, pointing towards the road;

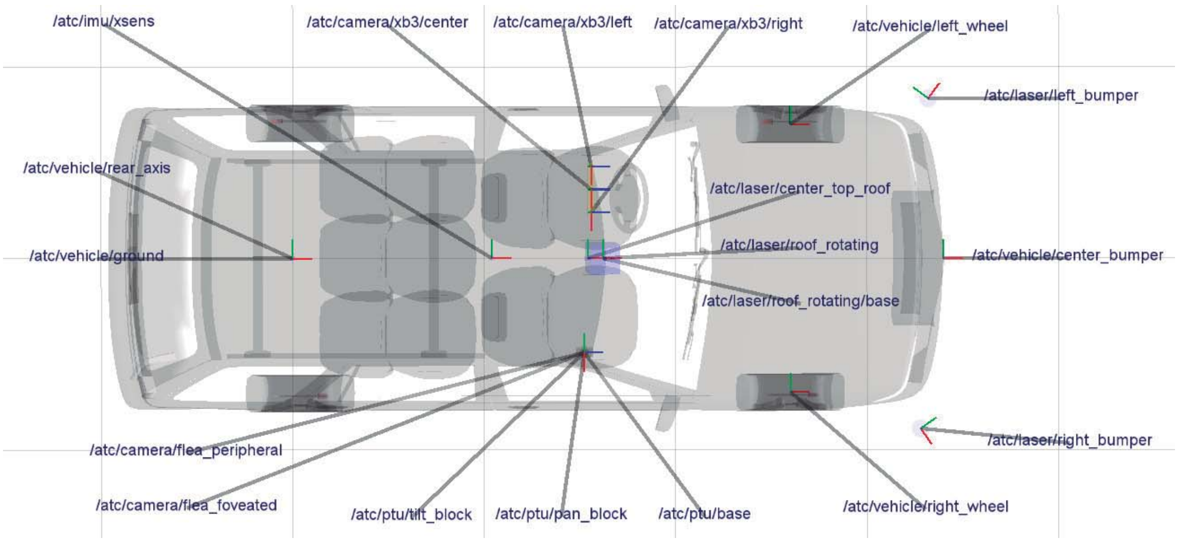
  /atc/imu/xsens, defined at the inertial measurement unit, located on the roof of the vehicle;

  /atc/camera/xb3/right, the right camera of the trinocular stereo pair, on the roof of the vehicle facing the road in front;

  /atc/camera/xb3/center, the center camera of the trinocular stereo pair, on the roof of the vehicle facing the road in front;



(*a*)



(*b*)

Figure 4.19: A diagram of the several coordinate frames defined on the *AtlasCar*.(*a*) isometric view; (*b*) top view; The XYZ axis are represented in red green and blue colors respectively.

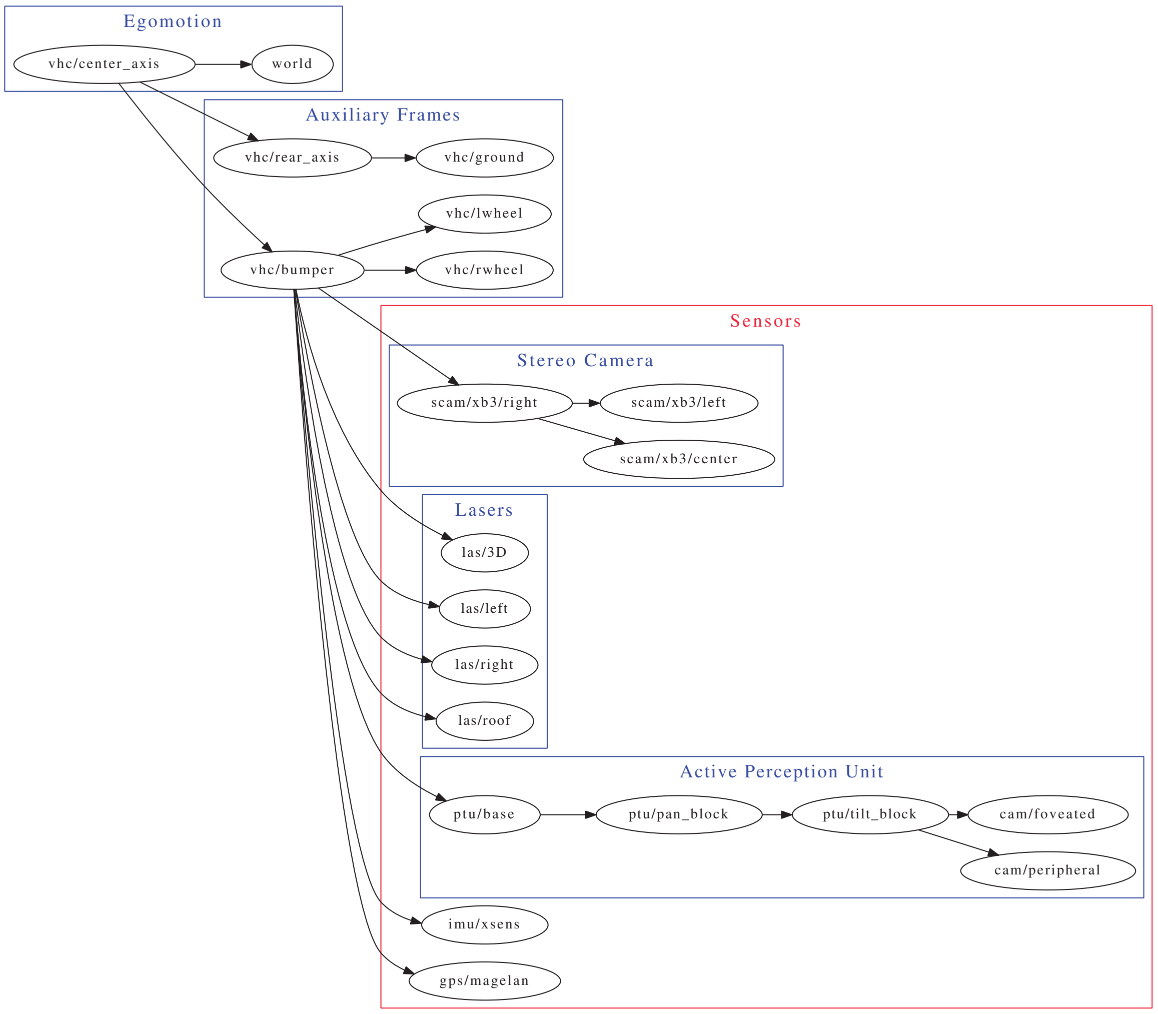


Figure 4.20: A diagram of the tree structure of all the coordinate frames define on-board the *AtlasCar*.

/atc/camera/xb3/left, the left camera of the trinocular stereo pair, on the roof of the vehicle facing the road in front;

/atc/camera/flea/peripheral, the wide angle camera mounted on top of the pan and tilt unit;

/atc/camera/flea/peripheral, the tele-objective camera mounted above the peripheral camera;

- Mobile Coordinate frames may move over time and provide greater coverage of the scene by the sensors mounted on top of these platforms:

  /atc/laser/roof_rotating, defined at the center of the Sick LMS 200 laser mounted on the rotating platform located on the roof of the vehicle. This frame rotates around the */atc/laser/roof_rotating_base* frame;

/atc/ptu/pan_block, rotates around the */atc/ptu/base* according to the pan angle;

/atc/ptu/tilt_block, rotates around the */atc/ptu/pan_block* according to the tilt angle;

/atc/vehicle/left_wheel, defined at the left wheel of the vehicle, changes position according to the steering wheel.

/atc/vehicle/left_wheel, defined at the right wheel of the vehicle, changes position according to the steering wheel.

- Auxiliary coordinate frames are frames defined for convenience, they facilitate measuring static transforms or are useful for analysing coordinates of objects with respect to the vehicle:

  /atc/ptu/base is the support coordinate frame for the PTU equipment;

  /atc/laser/roof_rotating/base, the support coordinate frame for the rotating laser equipment;

  /atc/vehicle/center_bumper, useful for assessing how distant are objects from the front of the vehicle;

  /atc/vehicle/ground, useful for assessing if objects lie on the road plane;

  /atc/vehicle/rear_axis, used for ego motion estimation.

The connection tree that defines every sub transformation between pairs of coordinate frames is shown in Fig. 4.20. As seen, there are many coordinate frames defined for the *AtlasCar*. The ROS framework helps to keep track of every defined coordinate frame and makes transformations from any coordinate frame transparent to the programmer.

4.5.2 Vehicle Interface and Control

The first group of modules is the vehicle and transformations layer, represented in Fig. 4.21. The launch files for this layer sets a global parameter of interest to many other modules, the location in the operating system of the URDF file. The following nodes are launched:

- *device_mapper* is a node which maps the devices connected to the computer. Many of the mounted sensors communicate via USB. The Linux operating system mounts the USB devices in */tty/USB{number}* ports. Since the order of mounting is arbitrary a specific equipment may be mounted on any of the ports. The *device_mapper* node reads a description of the robots sensors and compares it with the ports mounted in the operating system. The description is done in a xml file where several attributes may be specified for each sensor. Normally, only the serial number of the sensor is required. The *device_mapper* compares each of the sensors description against all mounted ports, when serial numbers match, it creates a global parameter indicating that the equipment should connect to the identified USB port;

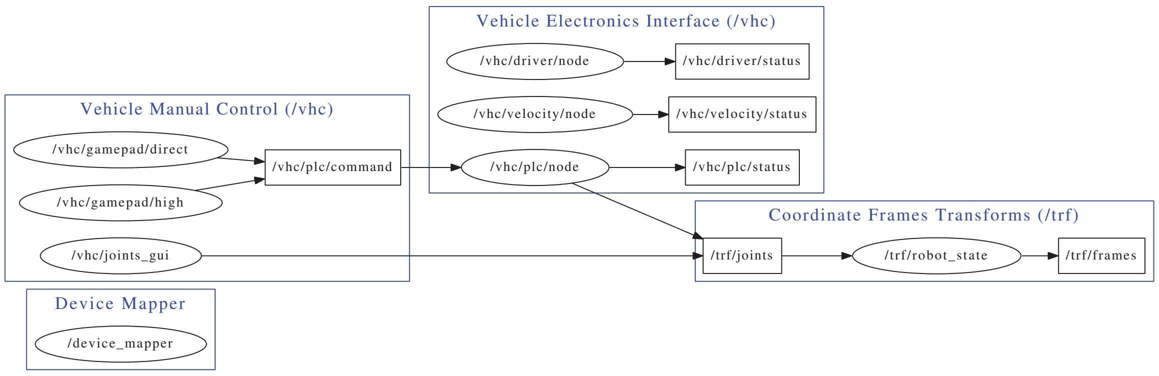


Figure 4.21: The Computation graph of the Vehicle Electronics Interface, Vehicle Manual Control, Coordinate Frames Transforms and Device Mapper layers. These are basic functionalities of the *AtlasCar*. In this diagram, nodes are represented as ellipses and messages as rectangles.

- Vehicle Electronics Interface, */vhc* name space, groups all the interface nodes that communicate with the vehicle electronics:

  /vhc/plc/node is the node responsible for communicating with the Programmable Logic Controller (PLC) device. Communications are based on Ethernet protocol. The node publishes */vhc/plc/status* messages, where all the information collected from the PLC is contained. The PCL provides information about the pedals, steering wheel and gearbox actuator systems: pedals and steering wheel positions, gearbox position, the state of the headlights, tail lights, turn signals and emergency light; This node also sends command orders to the PLC, to set the vehicle's speed, direction, and others. For this purpose, the node subscribes to the */vhc/plc/command* message. Hence, any node may directly control the vehicle by publishing these control messages;

  /vhc/driver/node is a node that communicates through Ethernet protocol with a custom designed electronics board. The board collects data about the drivers behaviour, namely it measures the force that the driver is using to push the pedals and steering wheel. It is designed to collect data from the human driving behaviour. The node publishes */vhc/driver/status* messages;

  /vhc/velocity/node is a node that communicates with an electronic board that reads the encoder mounted on the rear left wheel of the car. Communications are based on Ethernet, and the node publishes the measured velocity of the vehicle in the */vhc/velocity/status* message;

- Vehicle Manual Control, */vhc* name space, groups all the nodes that enable a manual control of the vehicle:

  /vhc/gamepad/direct is a node that communicates with common joystick gamepad and sends the user inputs to the */vhc/plc/node* through the */vhc/plc/command* message. The position

of the sticks in the gamepad are mapped to the desired position of the pedals and steering wheel;

/vhc/gamepad/high is a node that communicates with common joystick gamepad and sends the user inputs to the */vhc/plc/node*, just like the */vhc/gamepad/direct* node. The difference is that this node implements a high level control of the vehicle for the desired speed. The position of the sticks is not directly linked to the position of the pedals. Instead, the user input defines the desired speed and accordingly the */vhc/plc/node* computes the desired clutch, throttle and brake pedal positions;

/vhc/joints_gui is a node that enables the user to manually define the position of the joints;

- Coordinate frames transforms, */trf* name space, groups all nodes that manage the transformations between the on-board coordinate frames. The */trf/robot_state* is a robot state publisher node [Meeussen 2012]. The robot state publisher package computes the transformations associated with every coordinate frame of a robot. The package takes the joint angles of the robot (in this case in the form of */trf/joints* messages) as input and publishes the 3D poses of the robot links as */trf/frames* messages, using a kinematic tree model of the robot defined in URDF file. This node runs an instance of the robot state publisher configured for the *AtlasCar*.

4.5.3 Cameras

The cameras mounted on top of the PTU, the right roof peripheral and right roof foveated cameras, are grouped into the cameras layer, name space */cam*. Figure 4.22 shows the computation graph. Each camera functionality contains a */snr/cam/node*, a *camera1394* package [O'Quin & Tossell 2012] that is responsible for capturing images from the camera, setting image resolution, white balance, satura-

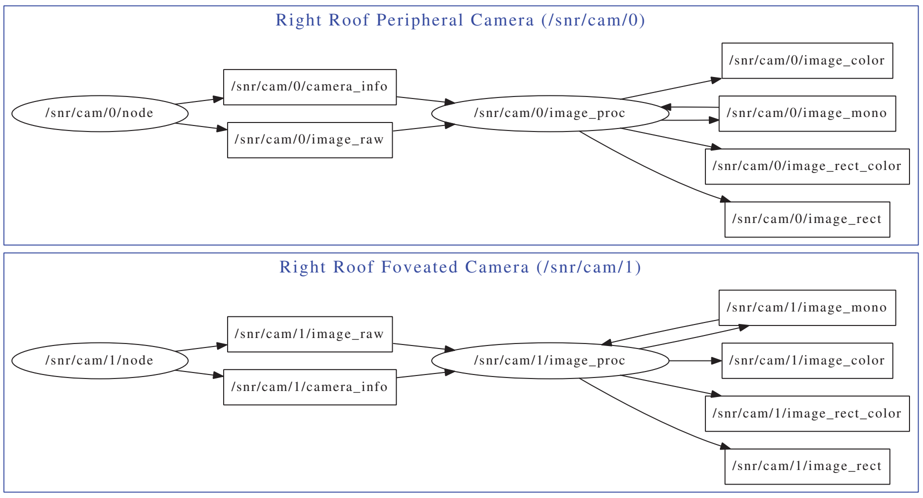


Figure 4.22: Camera layer computation graph.

tion, and every other camera parameters. This node publishes the images in the */snr/cam/image_raw* messages, as well as information about the calibration of the camera in the */snr/cam/camera_info* message. The calibration information contains the intrinsic parameters of the camera, as well as the pinhole model distortion coefficients. It is obtained using an Open Source Computer Vision Library (OPENCV) [Asbach *et al.* 2012] chessboard calibration, available in the camera calibration package [Bowman & Mihelich 2012], and stored in a calibration file. In runtime, the calibration file is read and */snr/cam/camera_info* messages are published.

The */snr/image_proc* node [Mihelich *et al.* 2012a] is meant to lay between the camera driver and vision processing nodes. It removes camera distortion from the raw image stream, publishing */snr/cam/image_rect* messages, and, if necessary, converts Bayer or YUV422 format image data to color. It publishes both color and mono images, and also supports real time compression of images before publishing.

4.5.4 Stereo Camera

The stereo camera layer uses the */scam* name space. It is composed by two stereo pairs based on the three cameras of the XB3 camera. The first stereo pair is constituted by the left and center Point Grey Research Bumblebee XB3 Stereo Camera (XB3) cameras. It has a smaller baseline and is called short (name space */snr/scam/short*). The second stereo pair, designated wide (name space */snr/scam/wide*) is composed by the left and right cameras of the XB3 device. Both stereo pairs are calibrated using the camera calibration package [Bowman & Mihelich 2012]. Figure 4.23 shows the computation graph of the stereo processing layer. The */snr/scam/xb3* node communicates with the camera using IEEE1394 protocol. It publishes four image messages, two for each stereo pair, as well as *camera_info* messages for each image. The short and wide stereo processing nodes (*/snr/scam/short/stereo_image_proc* and */snr/scam/wide/stereo_image_proc*) [Mihelich *et al.* 2012b] compute the rectified images using the information of the *camera_info* messages, and use them to compute the disparity images (*/snr/scam/ short/stereo_image _proc/disparity* and */snr/scam/wide /stereo _image_proc/disparity*) and 3D point clouds (*/snr/scam/short/stereo _image_proc/points2* and */snr/scam/wide/ stereo _image_proc/points2*).

Figure 4.24 shows the disparity maps obtained using the stereo processing layer. Figure 4.25 shows the 3D point clouds obtained using the stereo processing layer.

4.5.5 Pan and Tilt Unit

The PTU layer is organized as depicted in Fig. 4.26. The */snr* name space refers the sensors, the */snr/ptu/ctrl* is related to the control of the PTU. There are two distinct methods for controlling the PTU position. The first is to directly impose the angle of the joints. This is referred to as direct control. The second is called foveation control, where the cameras mounted on top of the PTU must point towards a given a 3D position, the target. This functionality is useful when detailed visual

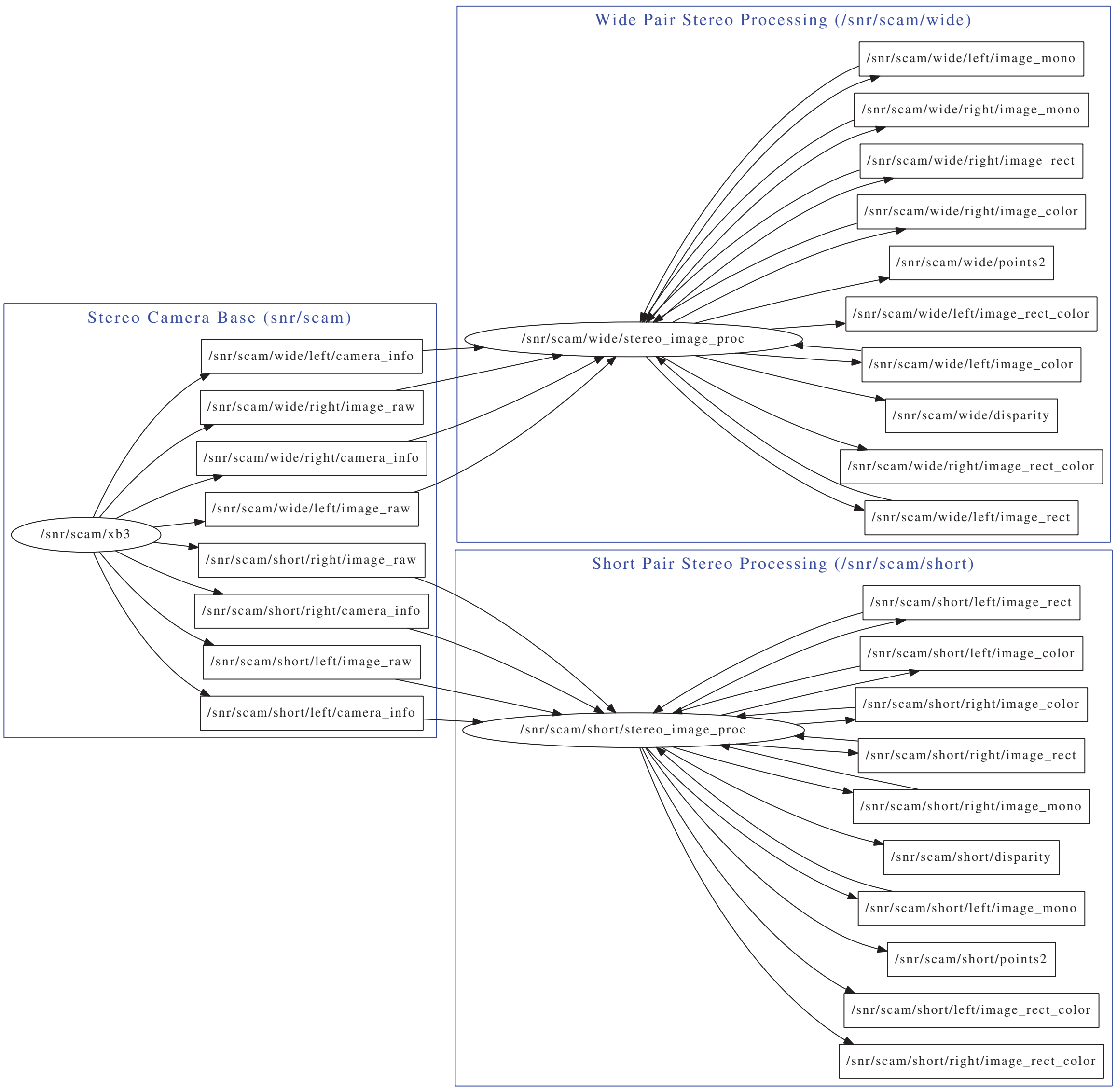

Wide Pair Stereo Processing (/snr/scam/wide)
/snr/scam/wide/left/image_mono
/snr/scam/wide/right/image_mono
/snr/scam/wide/right/image_rect
/snr/scam/wide/right/image_color
/snr/scam/wide/points2
/snr/scam/wide/left/image_rect_color
/snr/scam/wide/stereo_image_proc
/snr/scam/wide/left/image_color
/snr/scam/wide/disparity
/snr/scam/wide/right/image_rect_color
/snr/scam/wide/left/image_rect
Stereo Camera Base (snr/scam)
/snr/scam/wide/left/camera_info
/snr/scam/wide/right/image_raw
/snr/scam/wide/right/camera_info
/snr/scam/wide/left/image_raw
/snr/scam/xb3
/snr/scam/short/right/image_raw
/snr/scam/short/right/camera_info
/snr/scam/short/left/image_raw
/snr/scam/short/left/camera_info
Short Pair Stereo Processing (/snr/scam/short)
/snr/scam/short/left/image_rect
/snr/scam/short/left/image_color
/snr/scam/short/right/image_color
/snr/scam/short/stereo_image_proc
/snr/scam/short/right/image_rect
/snr/scam/short/right/image_mono
/snr/scam/short/disparity
/snr/scam/short/left/image_mono
/snr/scam/short/points2
/snr/scam/short/left/image_rect_color
/snr/scam/short/right/image_rect_color

Figure 4.23: Stereo camera layer

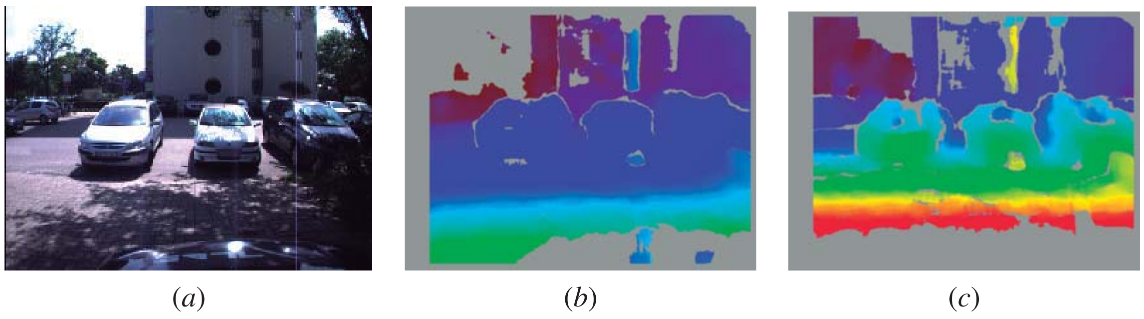


Figure 4.24: Stereo processing on the *AtlasCar*. (*a*) the image from the left camera, common to both stereo pairs; (*b*) the disparity image of the short stereo pair; (*c*) the disparity image of the wide stereo pair.

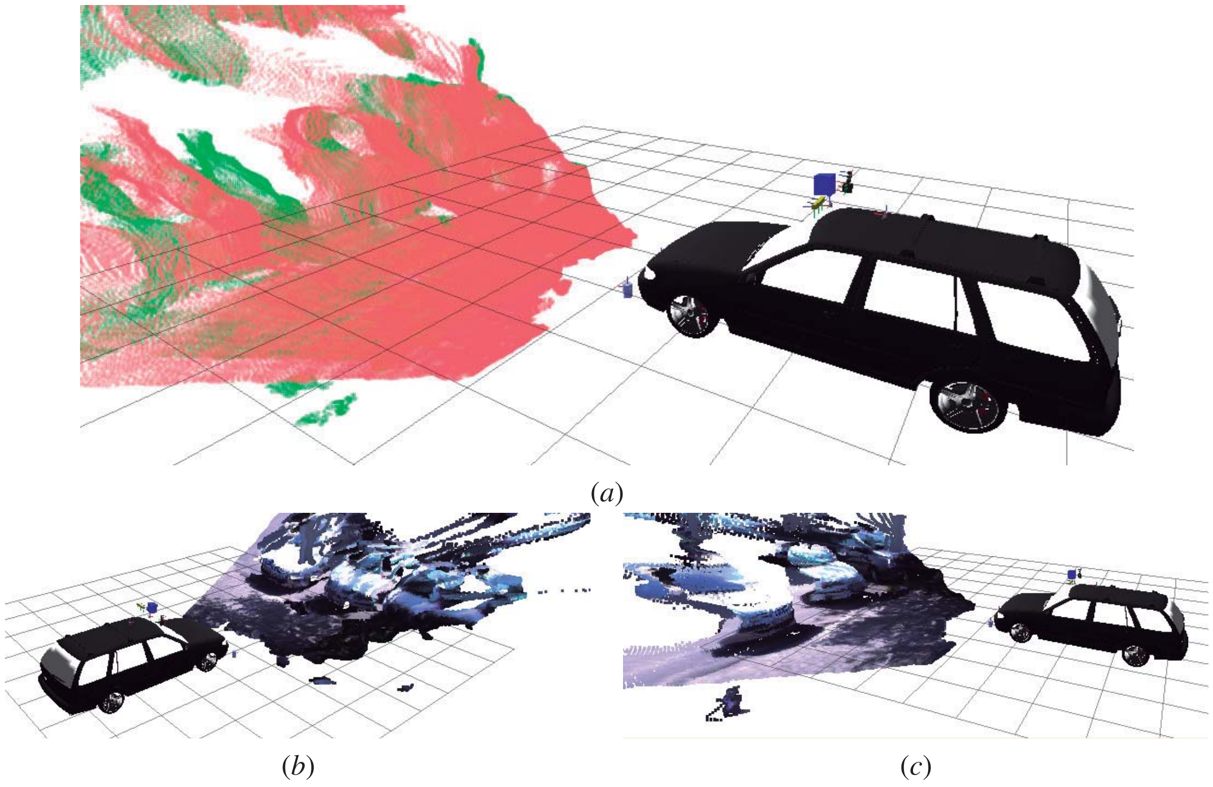


Figure 4.25: Point clouds obtained from stereo. (*a*) fake colors showing the stereo pair provenience of each point (*red* short, *green* wide pair) ; (*b*) and (*c*) the same scene with true textures.

information of an object is required. The object might be detected by the lasers, for example, and given its 3D position, the foveation controller can position the cameras to capture high level of detail images of the object.

- */snr/ptu/node* is the node that communicates through RS232 protocol with the PTU. It controls the PTU position according to */snr/ptu/command* messages, which may come from the direct

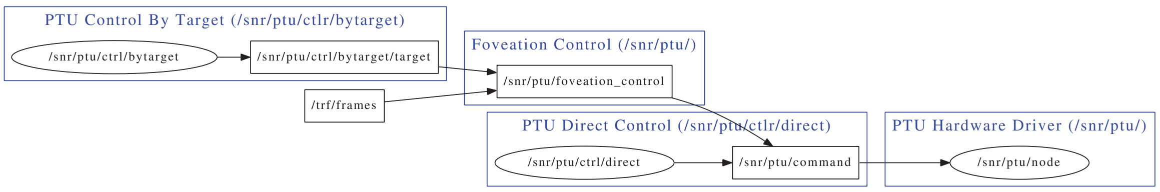

PTU Control By Target (/snr/ptu/ctlr/bytarget)
/snr/ptu/ctrl/bytarget
/snr/ptu/ctrl/bytarget/target
Foveation Control (/snr/ptu/)
/snr/ptu/foveation_control
/trf/frames
PTU Direct Control (/snr/ptu/ctlr/direct)
/snr/ptu/ctrl/direct
/snr/ptu/command
PTU Hardware Driver (/snr/ptu/)
/snr/ptu/node

Figure 4.26: The computation graph of the PTU control layer.

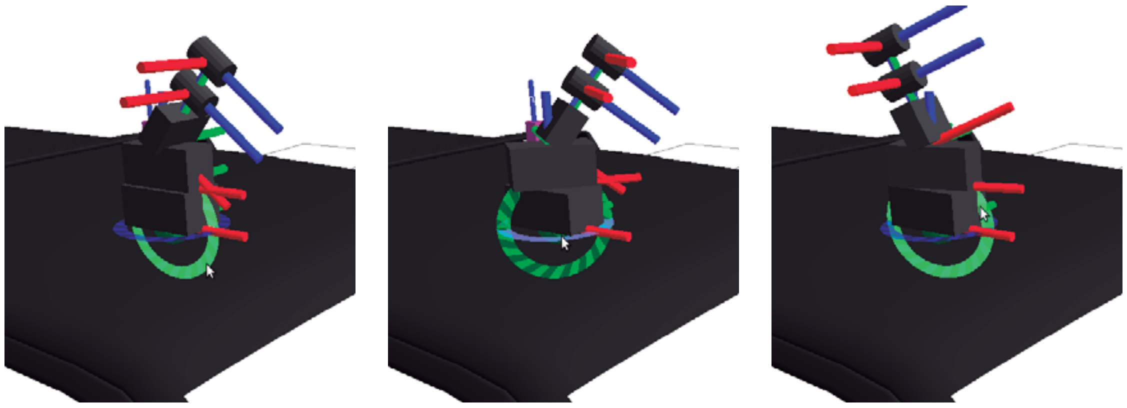


Figure 4.27: Direct control of the PTU.

control or the foveation control nodes.

- */snr/ctrl/direct* sets the PTU joint angles directly through the */snr/ptu/command* message.
- */snr/ctrl/foveation_control* sets the PTU joint angles through the */snr/ptu/command* message so that the cameras mounted on top of the PTU point towards a 3D position given by the */snr/ptu/ctrl/bytarget/target* message.
- */snr/ctrl/bytarget* defines the target position for the */snr/ptu/foveation_control*

Figure 4.27 shows some images where the PTU is controlled using direct control, and Fig. 4.28 shows a foveation control sequence.

4.5.6 Planar Laser Range Finders

The laser processing layer includes the three fixed lasers: bumper left and right and center top roof. The Sick LMS 151 lasers, name spaces */snr/las/2* and */snr/las/3*, use the LMS1xx package [Banachowicz 2012]. Communications are base of Ethernet protocol. The Hokuyo UTM30LX laser, name space */snr/las/0*, uses the ROS Hokuyo laser driver [Gerkey *et al.* 2012]. Figure 4.29 shows the computation graph of the lasers layer.

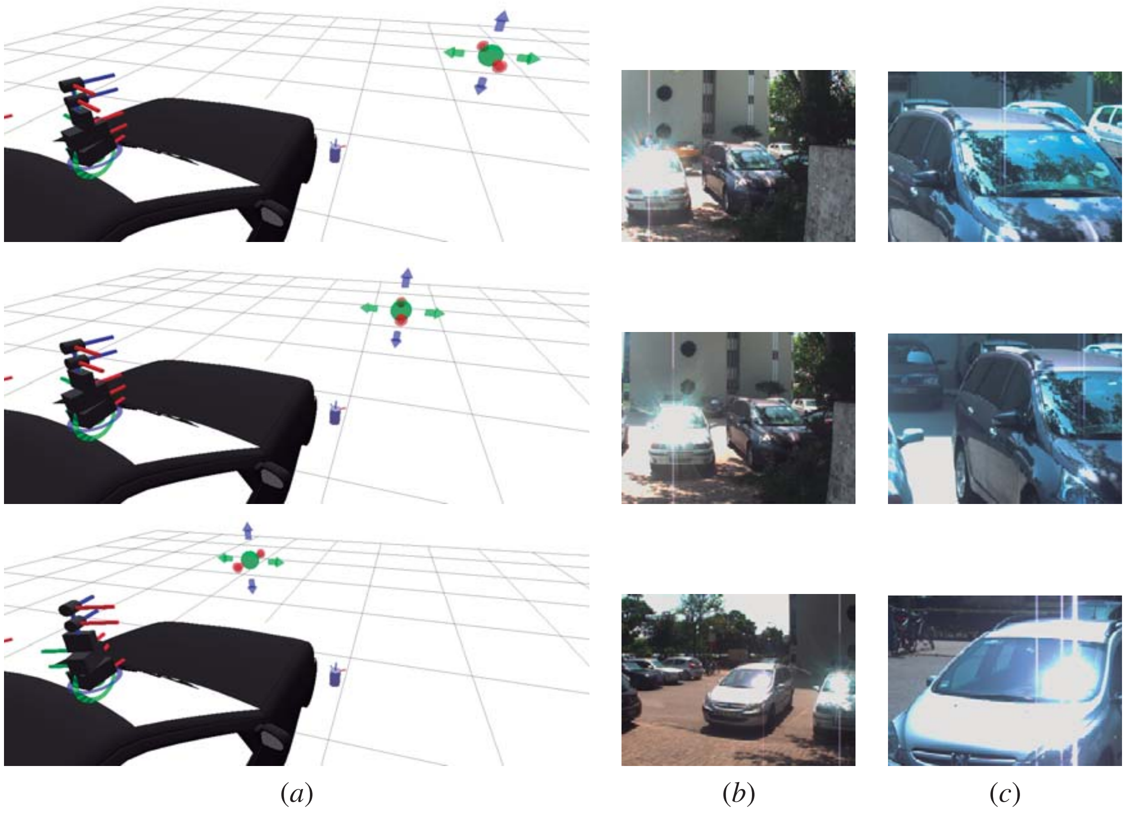


Figure 4.28: PTU foveation control. In this sequence, the target marker (green ball) is moved from right to left (*top* to *bottom* sequence in the figure); (*a*) a 3D visualisation of the PTU and the marker; (*b*) the image from the peripheral camera; (*c*) the image from the foveated camera.

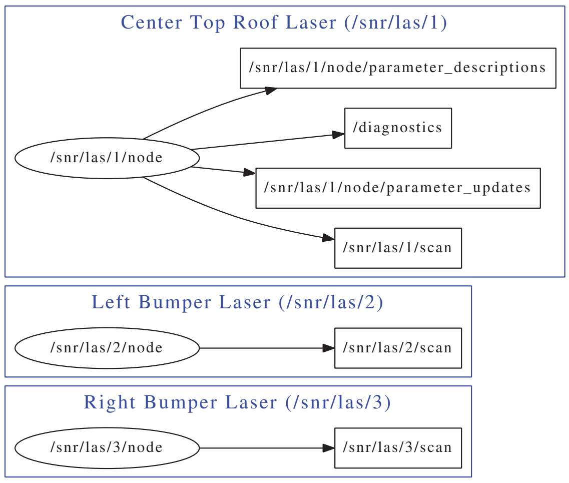

Center Top Roof Laser (/snr/las/1)
/snr/las/1/node
/snr/las/1/node/parameter_descriptions
/diagnostics
/snr/las/1/node/parameter_updates
/snr/las/1/scan
Left Bumper Laser (/snr/las/2)
/snr/las/2/node
/snr/las/2/scan
Right Bumper Laser (/snr/las/3)
/snr/las/3/node
/snr/las/3/scan

Figure 4.29: Planar laser range finders layer.

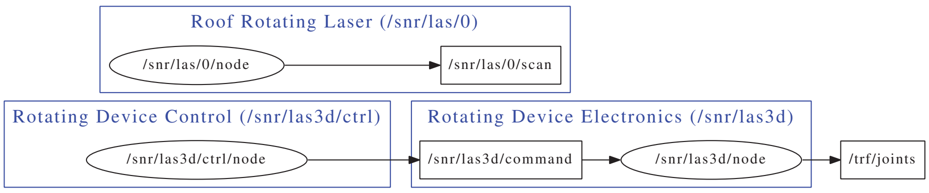

Roof Rotating Laser (/snr/las/0)
/snr/las/0/node
/snr/las/0/scan
Rotating Device Control (/snr/las3d/ctrl)
/snr/las3d/ctrl/node
Rotating Device Electronics (/snr/las3d)
/snr/las3d/command
/snr/las3d/node
/trf/joints

Figure 4.30: The computation graph of the 3D laser range finder layer.

4.5.7 3D Laser Range Finder

The 3D laser is mounted on the roof of the *AtlasCar*. Figure 4.30 shows the corresponding computation graph. Laser scan messages (*/snr/las/0/scan*) are provided by the */snr/las/0/node*, a driver that communicates with the laser using RS422 serial protocol [Derenick 2012]. The rotating device is controlled by the */snr/las3d/node* using RS232 serial protocol. This node publishes the rotation angle as a */trf/joints* message. It subscribes to *snr/las3d/command* messages that define the desired position or a constant rotation speed. The */snr/las3d/ctrl/node* uses a RVIZ graphical user interface to let the user select the desired position or velocity. Figure 4.31 shows four laser scans of a scene obtained with different rotations on the device.

Using the ROS framework, the reconstruction of 3D point clouds is straightforward. Since the */snr/las3d/0/node* is continuously publishing */trf/joints* messages, the */trf/robot_state* node (see Fig. 4.21) uses this information to periodically publish transforms from the *las/3D* coordinate frame to the */vhc/bumper* coordinate frame (see Fig. 4.21). A node that is performing reconstruction accumu-

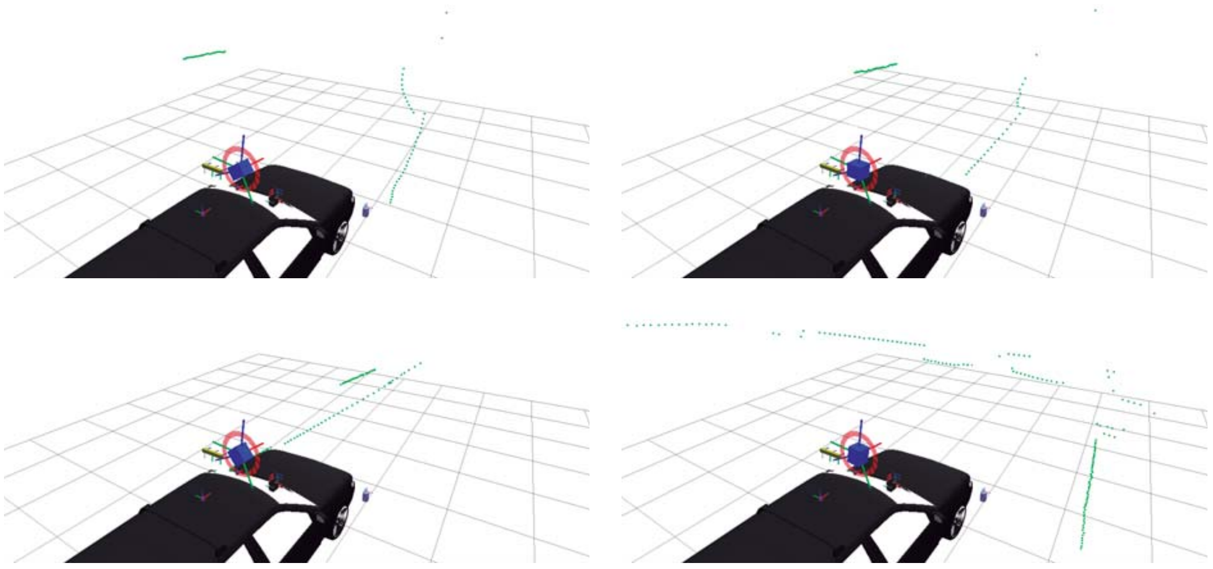
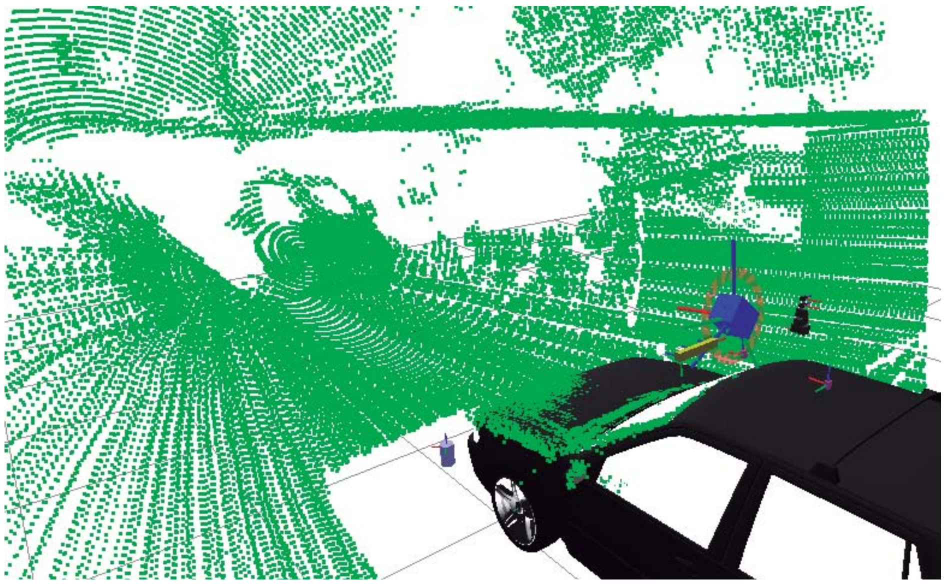


Figure 4.31: The laser scans obtained with the 3D laser on different positions.


(*a*)

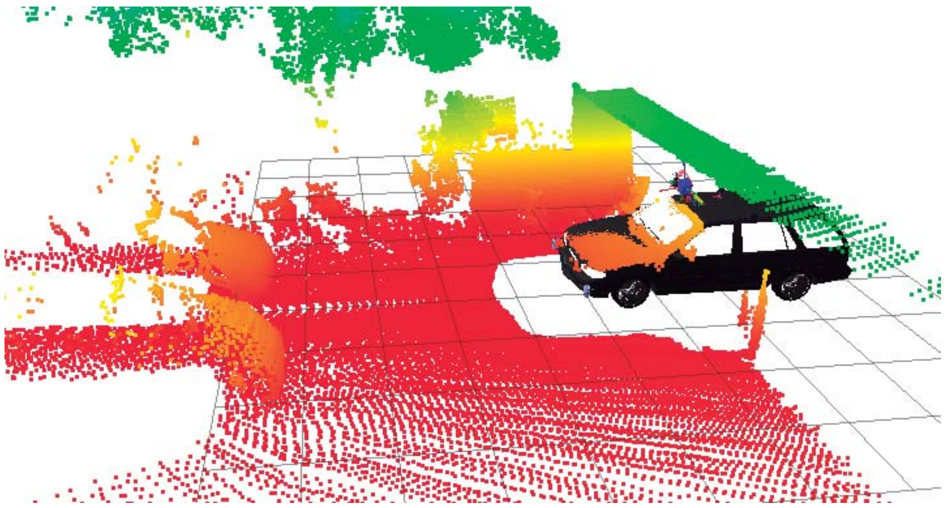

(*b*)

Figure 4.32: 3D reconstruction using the 3D laser on-board the *AtlasCar*. (*a*) reconstructed scene obtained by the accumulation of several scans show in Fig. 4.31; (*b*) the same scene with fake colors.

lates laser scans and transforms each scan to a particular vehicle coordinate frame (for example the */vhc/bumper*). Since each laser scan message has an associated time stamp, the transformation of each scan will be performed with respect to the corresponding the laser position at that time, generating a 3D point cloud (section 4.2.3). Figure 4.32 shows the reconstructed scene obtained by accumulating several scans.

4.5.8 Multi Target Tracking

The *AtlasCar* uses an adaptation of a Kalman filter based algorithm [Almeida & Santos 2010], that was developed for tracking objects on a single laser scan, i.e., on a single plane. In the *AtlasCar*, four different laser scanners produce different scans in different planes, as shown in Fig. 4.33. Figure 4.34 shows the computation graph of the multi target tracking layer. All laser scan messages provided by the laser layer are subscribed by the */pcp/fus/planar_pc/node* node. This node will fuse the information from the laser scans based on some parameters. First, a tracking plane is defined. The tracking plane is the plane where laser measurements are considered, drawn in Fig. 4.33. The node filters out all points from the laser scans that have a distance greater than a predetermined threshold, and accumulates the remaining laser scans points into a single 3D point cloud (*/pcp/fus/planar_pc/points*). Thus, it is possible to make use of portions of laser scans that have scanning planes not coincident with the tracking plane. Figure 4.35 shows and example of the filtering mechanism of the planar point cloud fusion algorithm. The */pcp/trk/mtt/node* uses the */pcp/fus/planar_pc/points* message to track all obstacles viewed in a scene. Figure 4.36 shows a sequence where a group of persons moving in front of the vehicle are tracked. The group of persons first appear behind the white vehicle and are

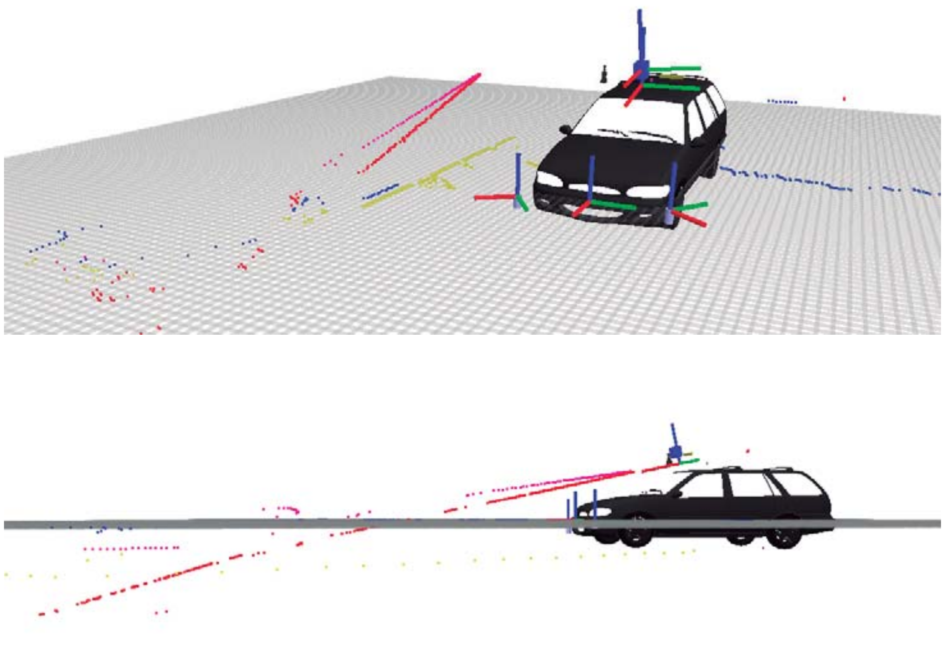


Figure 4.33: Multi target tracking. The four on-board laser scanners provide laser scans on different planes: (*yellow*) right bumper laser; (*blue*) left bumper laser; (*red*) center roof laser; (*magenta*) rotating laser; The tracking is performed on a plane, in this case defined on the XY plane of the vehicle center bumper frame, represented by the grey grid.

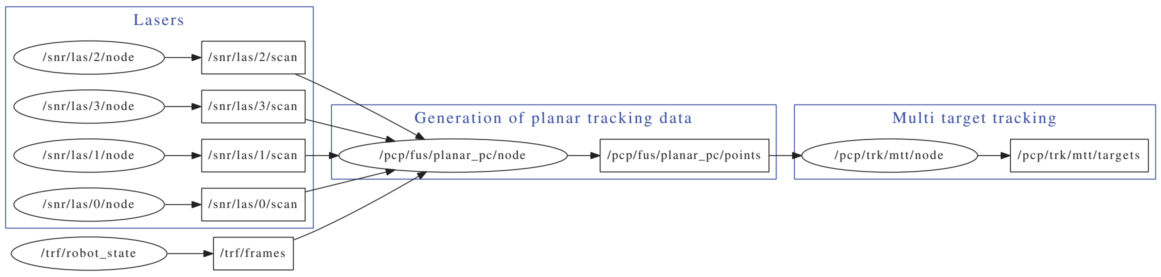


Figure 4.34: The computation graph of the multi target tracking layer.

classified as a single obstacle label 2155, since one of them is occluding the others (*top row*). In the following sequences, the id of the group is kept since the obstacle 2155 is tracked.

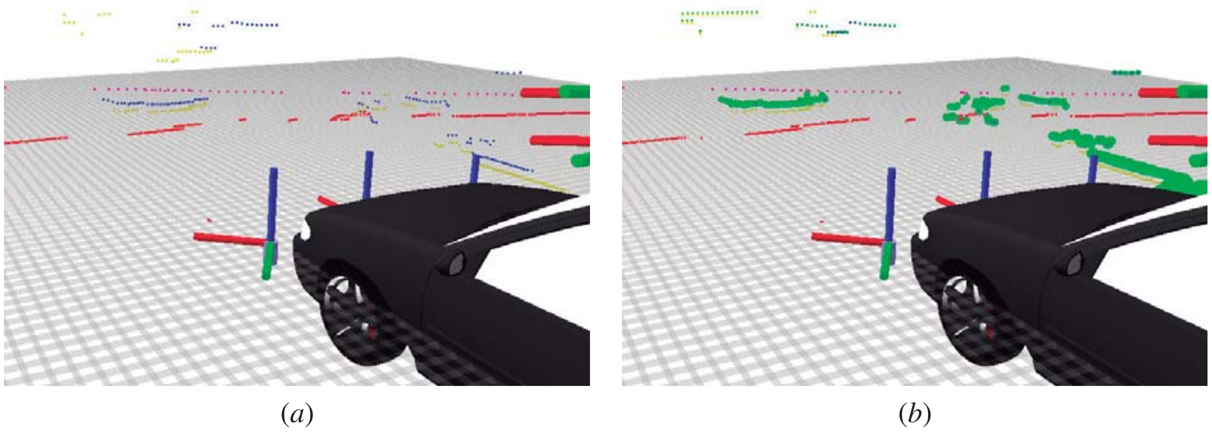


Figure 4.35: An example of the planar point cloud fusion. (*a*) a scene showing all the laser scans and the tracking plane; (*b*) the same scene with the points included in the fused point cloud signaled as green spheres.

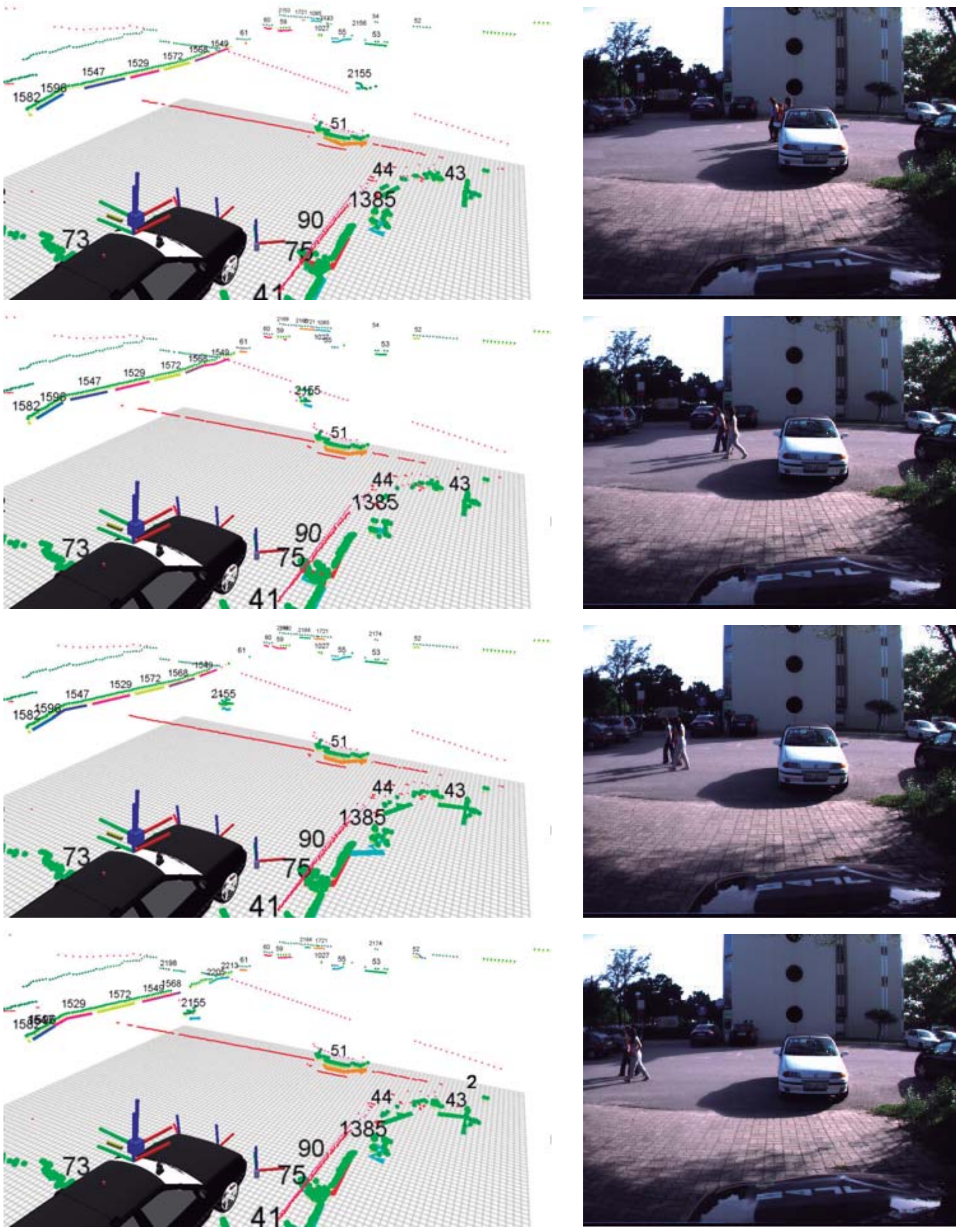


Figure 4.36: A tracking sequence (*top* to *bottom*). (*a*) A 3D representation of the scene; (*b*) the image taken from the scene.

4.6 Conclusions

This chapter has described the CARMEN, LARtk and ROS software architectures. The LARtk software architecture was employed in the *AtlasMV* robot series from 2009 onward, and the ROS framework is now being used as the base architecture for the development in the *AtlasCar*. Both software architectures have shown to be a significant advance in terms of code reuse, simultaneous development, and debugging tools. Nowadays, ROS may be viewed as a standard for robotic development. The application of this toolkit to the *Atlas2000* and *AtlasMV* series has already started in 2012.

Chapter 5

Inverse Perspective Mapping

The current chapter proposes a solution to address the limitations of the Inverse Perspective Mapping (IPM) methodology. The algorithm works by finding the mappable pixels in the image through a novel multi-modal framework. A polygon in the 3D road plane is obtained by the combined use of image properties and a Laser Range Finder (LRF). The 2D and 3D information is then employed to define the projection boundaries in the image and hence, the region of mappable pixels.

This chapter begins with an introduction to IPM (section 5.1), followed by a brief review of the state of the art on the topic of IPM (section 5.2). Then, section 5.3 presents the classical IPM algorithm. Relevant coordinate frames are introduced in section 5.4. The proposed approach is described in detail in sections 5.5. Results and conclusions are given in sections 5.6 and 5.7.

5.1 Introduction

Over the past years IPM has been successfully applied to several problems in the field of Intelligent Transportation Systems. In brief, the method consists of mapping images to a new coordinate system where perspective effects are removed. The removal of perspective associated effects facilitates road and obstacle detection and also assists in free space estimation. The current chapter addresses two significant limitations of classical approaches for Inverse Perspective Mapping:

- the presence of obstacles on the road;
- the loss of mapping accuracy during hard turns or demanding brake or acceleration maneuvers.

These limitations can be solved using a multi-modal approach, where information from a laser range finder is fused with the data from the camera. In this chapter, we will present an approach that is also able to cope with several cameras with different lenses or image resolutions. Furthermore, it can deal with dynamic viewpoints, i.e., to map images taken from moving cameras. As will be shown, the proposed algorithm reduces computation time and increases the mapping accuracy when compared with the classical IPM.

The past decades have witnessed a high pace of development in video based road or lane detection algorithms. These have achieved considerable improvements over the existing methods of traffic data collection and road traffic monitoring as well as produced extensive applications in the related field of autonomous vehicle guidance, mainly for determining the vehicle's relative position in the lane and for obstacle detection [Kastrinaki *et al.* 2003]. Over the last years, many researchers have employed the IPM technique as a part of the presented algorithms.

IPM uses information from the camera's position and orientation towards the road to produce a bird's eye view image where perspective effects are removed. The correction of perspective allows much more efficient and robust road detection, lane marker tracking, or pattern recognition algorithms to be implemented. In fact, it has been employed not only with the purpose of detecting the vehicle's position towards the road but also with many other applications (e.g., obstacle detection, free space estimation, pedestrian detection).

Despite the dedicated attention from the research community, the classic IPM method still presents some limitations when applied to the context of onboard road mapping. It works under three core assumptions: flat road, rigid body transformation from the camera to the road and road free of obstacles. Since the road plane must be coincident with the vehicle reference system (or a rigid body transformation from one to the other is assumed), pitch and roll variations from the host vehicle are neglected. The presence of obstacles such as other vehicles, buildings or pedestrians disrupt the mapping of IPM, because all projected pixels are assumed to be on the road plane, including the ones from the obstacles.

As will be shown, the inverse projection (image to 3D world, i.e., pixels to 3D points) has a much higher computational cost than the direct projection (from 3D world to image). The classical approaches presented for IPM make no considerations on this topic (e.g., [Aly 2008], [McCall *et al.* 2004], [Muad *et al.* 2004]). In the current chapter, the calculations for finding the mappable pixels are made in the real world and then directly projected to the image. Results will show that by computing a priori which pixels are to be inverse projected saves computation time as well as increases the accuracy.

5.2 Related Work

Over the last decades, IPM has been successfully applied to several problems especially in the field of Intelligent Transportation Systems. Although it was some years ago that authors began to mention the advantages of IPM (e.g., [Mallot *et al.* 1991], [Pomerleau 1995]), several recent publications (e.g., [Ehlgen *et al.* 2008], [Fang *et al.* 2009], [Li & Hai 2011]) show that this is still a topic of interest to the robotics, computer vision and intelligent transportation systems communities. The core application of IPM is the determination of the vehicle's position with respect to the road, commonly referred to as road detection or lane marker detection. There are several examples of using IPM for assisting road detection in the literature (e.g., [McCall & Trivedi 2005], [Muad *et al.* 2004], [Dornaika *et al.* 2011], [Guo *et al.* 2009]). The usage of IPM onboard a vehicle may also aid other automatic detection systems such as generic obstacle detection [Bertozzi *et al.* 1997] [Bertozzi & Broggi 1998], free space estimation [Tuohy *et al.* 2010] [Cerri & Grisleri 2005] pedestrian detection [Ma *et al.* 2007] [Broggi *et al.* 2008] or optical flow computation [Tan *et al.* 2006].

The IPM method receives as input the image from the camera, the 6D position of the camera with respect to the road reference system (i.e., extrinsic parameters), and a description of the properties of the lens (i.e., intrinsic parameters). Under the assumption that the road ahead of the vehicle is flat, that there is a fixed rigid body transformation from the camera to the road's reference frames, and that there are no obstacles present, the input image pixels are mapped to the road reference system and a new image is produced where perspective effects are removed. The image that is produced by the IPM will be henceforward named simply IPM image. Considering on-board road detection setups, cameras are usually mounted somewhere close to the rear view mirror inside the vehicle, facing to the road in front of it. The camera's position and orientation induces perspective associated effects to the captured road images. The IPM technique consists of transforming the images, mapping the pixels to a new reference frame where the perspective effect is corrected. This reference frame is usually defined on the road plane, so that the resulting images become a top view of the road. Figure 5.1(a) shows an example of a road scene; Figure 5.1(b) depicts the input image captured by the camera; and Fig. 5.1(c) represents the image produced using IPM.

One of the advantages of IPM is that the subsequent perception algorithms can be computed in the IPM resulting image, that is defined in a new reference system in which the geometric properties of road painted patterns are independent from the perspective of the camera, i.e., from the position of the camera. In [Pomerleau 1995], the authors claim that the parallelization of road features is crucial for curvature determination. Another advantage is that since the perspective effect associates different meanings to different image pixels, depending on their position in the image, after the removal of the perspective effect, each pixel represents the same portion of the road, allowing a homogeneous distribution of the information among the pixels of the resulting IPM image [Bertozzi & Broggi 1998]. Other authors have also employed steerable filters for lane markings detection and sustain that filtering on the IPM image allowing a single kernel size to be used over the entire area of interest [McCall & Trivedi 2005]. Furthermore, since images are mapped to a new reference system, several cameras may be used to produce a single IPM image mosaicing. There are some multi-camera IPM applications described in the literature [Bertozzi & Broggi 1998] [Guo *et al.* 2009]. It should also be noted that IPM requires no explicit feature detection, which contributes to the robustness and also that some special dedicated hardware systems are being developed to perform this

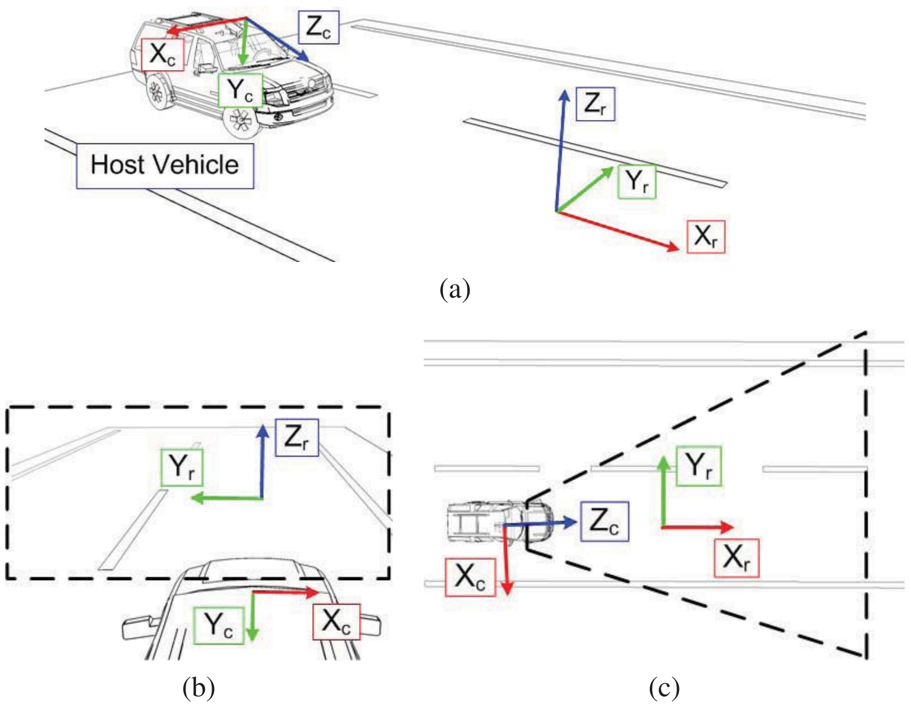


Figure 5.1: (a) A typical road scene with a camera mounted on the host vehicle facing the road. The camera reference system is labelled $X_cY_cZ_c$ and the road reference system is labelled $X_rY_rZ_r$. (b) An example of an image captured by the camera. This image is used as input to IPM. (c) The output image of IPM. Since the road is viewed from above no perspective distortion is present.

operation [Luo *et al.* 2009].

Given this, it is fair to say that IPM is a keystone in the development of on-board video processing systems. It assists or is very frequently a primary step in road modelling, obstacle and pedestrian detection, free space estimation and many other advanced drivers assistance systems.

Despite the advantages of IPM, the current state of the art on this method has some significative limitations, especially due to the following classical assumptions:

- **Assumption 1**: there must be a fixed rigid body transformation from the camera to the road's reference system.
- **Assumption 2**: the road must be free of obstacles.
- **Assumption 3**: the road must be flat.

Regarding **Assumption 1**, since the road plane must be coincident with the camera reference system (or a fixed rigid body transformation from one to the other is assumed), pitch and roll variations from the host vehicle are neglected. Pitch variations occur during demanding brake or acceleration maneuvers, while roll changes are expected to appear during hard turns. When the vehicle

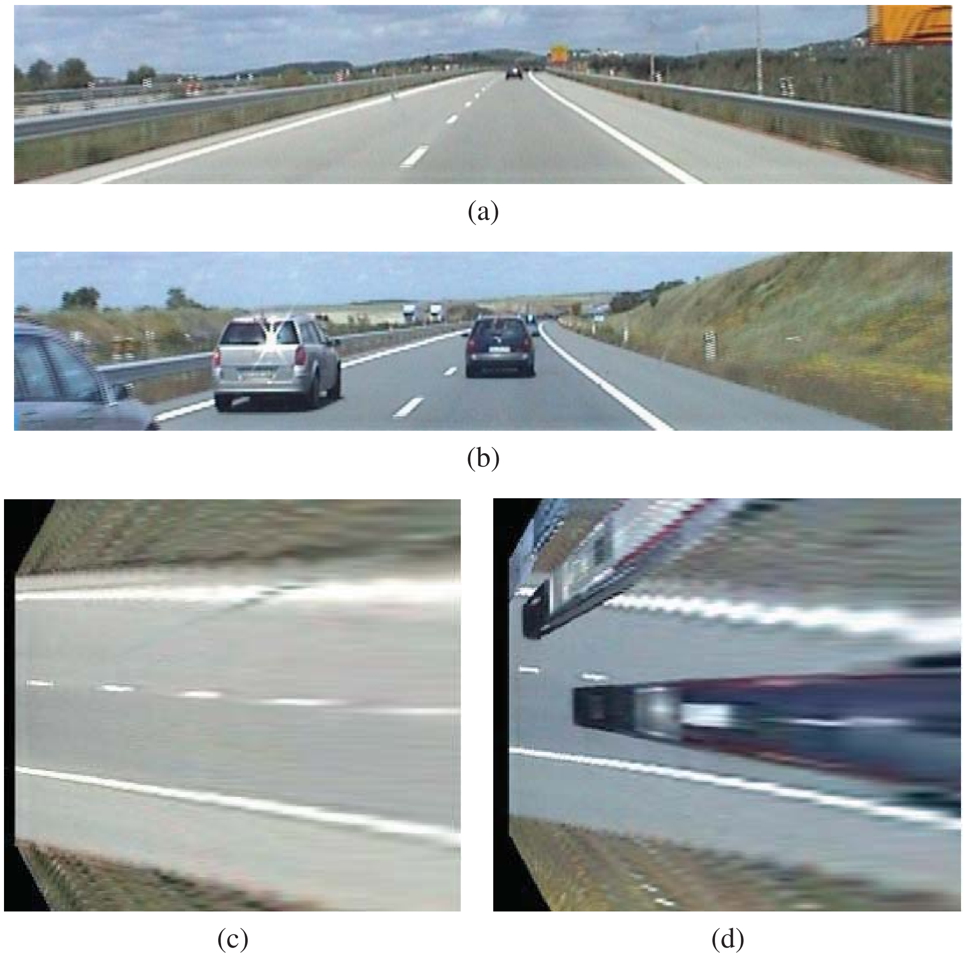


Figure 5.2: Some examples of problems with the current IPM method. The input image *(a)* is projected with a small error in pitch estimation, which results in an IPM image where lines are not parallel *(c)*. In *(b)*, the presence of other vehicles also disrupts the resulting IPM image *(d)*.

rolls or pitches the cameras 6D position with respect to the road changes. This problem has been identified in [Coulombeau & Laurgeau 2002], [Labayrade & Aubert 2003], [Dornaika *et al.* 2011] [Nieto *et al.* 2007]. Hence, during these maneuvers, IPM's effectiveness is expected to drop. In fact, some authors claim that even a small error in the vehicle's roll/pitch estimation leads to a massive terrain classification error, forcing the vehicle off the road [Thrun *et al.* 2006a].

In respect to **Assumption 2**, the presence of obstacles such as other vehicles, buildings or pedestrians will disrupt the mapping of IPM, because all pixels from the input image are assumed to be on the road plane. In automotive applications it is unfeasible to assume an obstacle free scenario.

Concerning **Assumption 3**, the approximation of the road surface to a plane is more acceptable. Nonetheless, in some roads, this may also be a factor for low IPM accuracy. However, this problem is less influent than the other two described before and is out of the scope of this chapter.

Figure 5.2 shows IPM images produced with an error in pitch estimation (a) and (c). It also shows a typical IPM image when other vehicles are present (b) and (d).

In sum, the IPM requires prior knowledge of the camera's 6D position, namely, the geometric transformation relating the cameras and road reference frames. This is equivalent to state that the cameras position, orientation and intrinsic parameters must be known before hand. IPM is also computed under the assumption that no other obstacles are viewed by the camera. These assumptions are a core issue of IPM. If undertaken by mistake, due to the presence of other vehicles, pedestrians, obstacles, or steep slopes in the road, the IPM produces wrong representations in the corrected image.

Let ${}^{c}\mathbf{R}_r$ be the classical 3×3 rotation matrix in 3D and ${}^{c}\mathbf{T}_r$ be the 3×1 translation vector in 3D that relates two reference systems. Their combination maps a point in the 3D road reference system $\mathbf{Q}_r = [\ X \quad Y \quad Z\ ]^T$ to a point in the camera's coordinate system $\mathbf{Q}_c = [\ X \quad Y \quad Z\ ]^T$:

$$\mathbf{Q}_c = {}^{c}\mathbf{R}_r \cdot \mathbf{Q}_r + {}^{c}\mathbf{T}_r. \tag{5.1}$$

Let $\mathbf{K}$ be the intrinsic parameters matrix of a given camera, represented as:

$$\mathbf{K} = \begin{bmatrix} \alpha_x & \beta & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{5.2}$$

where α_x and α_y are the lens scaling factors in both directions, x_0 and y_0 the principal point coordinates in pixels and β the skewness factor. These parameters can be obtained by an offline calibration since they are constant for each camera-lens setup.

The projection of an arbitrary 3D point $\mathbf{Q} = [\ X \quad Y \quad Z\ ]^T$ to a point $\mathbf{q}_h = [\ u \quad v \quad w\ ]^T$ in the camera's homogeneous image coordinate system, is described as:

$$\mathbf{q}_h = \mathbf{K} \cdot {}^{c}\mathbf{R}_r \cdot \mathbf{Q} + {}^{c}\mathbf{T}_r, \tag{5.3}$$

Finally, the coordinates of a pixel $\mathbf{q} = [\ x \quad y\ ]^T$ are obtained by adjusting the homogeneous coordinates with the scaling factor w:

$$\mathbf{q} = \frac{\mathbf{q}_h}{w}. \tag{5.4}$$

For simplification purposes, the current chapter will use the following notation:

$$\mathbf{K} \cdot {}^{c}\mathbf{R}_r = \mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}, \tag{5.5}$$

The above formulation may describe the projection of a point to a pixel in the image (*direct projection*), or it may be used to obtain the 3D point from the pixel coordinates (*inverse projection*).

The *direct projection* (dp) may be formulated as $dp : \mathbb{R}^3 \rightarrow \mathbb{Z}^2$, $\mathbf{Q} \rightarrow \mathbf{q}$. In the case of IPM, what is sought is the 3D coordinates of a given pixel. This is the inverse projection (ip), defined as $ip : \mathbb{Z}^2 \rightarrow \mathbb{R}^3$, $\mathbf{q} \rightarrow \mathbf{Q}$.

5.3 Classical Inverse Perspective Mapping

This section introduces the mathematical models used for IPM projections. Both the direct and inverse projections are reviewed. the direct and inverse projections.

As discussed in 5.2, the direct projection aims at obtaining the pixel coordinates of a 3D world point projected to the image. Equation (5.3) may then be rewritten as:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}. \tag{5.6}$$

which can be arranged as:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \alpha_x \mathbf{D}_1 + x_0 \mathbf{D}_3 + X(\alpha_x \mathbf{R}_{12} + x_0 \mathbf{R}_{31}) + Y(\alpha_x \mathbf{R}_{12} + x_0 \mathbf{R}_{32}) + Z(\alpha_x \mathbf{R}_{13} + x_0 \mathbf{R}_{33}) \\ \alpha_y \mathbf{D}_2 + y_0 \mathbf{D}_3 + X(\alpha_y \mathbf{R}_{21} + y_0 \mathbf{R}_{31}) + Y(\alpha_y \mathbf{R}_{22} + y_0 \mathbf{R}_{32}) + Z(\alpha_y \mathbf{R}_{23} + y_0 \mathbf{R}_{33}) \\ \mathbf{D}_3 + X\mathbf{R}_{31} + Y\mathbf{R}_{32} + Z\mathbf{R}_{33} \end{bmatrix}, \tag{5.7}$$

Using (5.4), we get the two equations that define the direct projection dp

$$\begin{cases} x = \dfrac{p_{11}X + p_{12}Y + p_{13}Z + t_1}{p_{31}X + p_{32}Y + p_{33}Z + t_3} \\ y = \dfrac{p_{21}X + p_{22}Y + p_{23}Z + t_2}{p_{31}X + p_{32}Y + p_{33}Z + t_3} \end{cases}, \tag{5.8}$$

this system of equations defines the direct projection of a point in the world reference system $\mathbf{Q} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ to a pixel in image coordinates $\mathbf{q} = \begin{bmatrix} x & y \end{bmatrix}^T$.

The inverse projection is the problem of obtaining the real world coordinates of a point from a pixel in the image. The problem is under-defined, since the three real world coordinates are sought from only two pixel coordinates. In IPM, the system is completed by defining the plane onto which the pixel is projected. Let an arbitrary plane, defined as:

$$\Pi : aX + bY + cZ + d = 0, \tag{5.9}$$

be the plane that contains the projection of the pixel. The system of equations in (5.3) may be extended to include the constraint of the projection plane, defined in (5.9):

$$w\begin{bmatrix} x \\ y \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & 0 \\ p_{21} & p_{22} & p_{23} & 0 \\ p_{31} & p_{32} & p_{33} & 0 \\ a & b & c & d \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ 0 \end{bmatrix}, \tag{5.10}$$

rearranging this formulation, the equations for inverse perspective mapping can be obtained. First, variable d may be moved inside the translation vector:

$$w\begin{bmatrix} x \\ y \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & 0 \\ p_{21} & p_{22} & p_{23} & 0 \\ p_{31} & p_{32} & p_{33} & 0 \\ a & b & c & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ d \end{bmatrix}, \tag{5.11}$$

then, equation (5.11) may be rearranged:

$$\begin{bmatrix} -t_1 \\ -t_2 \\ -t_3 \\ -d \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & 0 \\ p_{21} & p_{22} & p_{23} & 0 \\ p_{31} & p_{32} & p_{33} & 0 \\ a & b & c & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} - w\begin{bmatrix} x \\ y \\ 1 \\ 0 \end{bmatrix}, \tag{5.12}$$

and finally, the vector of pixel coordinates can be embedded inside the projection matrix:

$$\begin{bmatrix} -t_1 \\ -t_2 \\ -t_3 \\ -d \end{bmatrix} = \underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & -x \\ p_{21} & p_{22} & p_{23} & -y \\ p_{31} & p_{32} & p_{33} & -1 \\ a & b & c & 0 \end{bmatrix}}_{\text{A}} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}, \tag{5.13}$$

rearranging the system of equations results in the inverse projection (ip) of a pixel to a known plane:

$$\begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & -x \\ p_{21} & p_{22} & p_{23} & -y \\ p_{31} & p_{32} & p_{33} & -1 \\ a & b & c & 0 \end{bmatrix}^{-1} \begin{bmatrix} -t_1 \\ -t_2 \\ -t_3 \\ -d \end{bmatrix}, \tag{5.14}$$

this is a valid solution whenever matrix A is invertible and not singular. In other words, the projection formulation is invalid when the projection plane and the image plane are parallel and the projection plane is behind the image plane. The term *behind* will be clarified in section 5.5.1 with the introduction of the half space of projection.

5.4 Road and Vehicle Coordinate frames

In the classic IPM formulation the camera and road reference systems have a known fixed transformation between them (**Assumption 1**). The IPM projection will transform the pixels from the camera to the road reference system. When the road and camera reference systems are assumed to be coincident (or with a fixed transformation), pitch and roll variations from the vehicle towards the road are neglected. Pitch variations occur during demanding brake or acceleration maneuvers, while roll changes are expected to appear during hard turns (e.g., [Coulombeau & Laurgeau 2002], [Labayrade & Aubert 2003], [Dornaika *et al.* 2011]). Hence, in between these maneuvers, IPM's effectiveness is expected to drop.

In the current chapter we propose to add an additional reference system, the vehicle reference system. The vehicle reference system is fixed to the host vehicle. It is the reference system to which all sensors on the vehicle are related. Therefore, a fixed, rigid body transform is used to perform transformations between the camera and the vehicle reference systems. Hence, three reference systems are used: the camera system $X_cY_cZ_c$, the road reference system $X_rY_rZ_r$ and the vehicle reference system $X_vY_vZ_v$. Figure 5.3 shows the reference systems for the vehicle, road and camera.

The general camera to road reference systems transformation was introduced in eq. (5.1). Let the rotation and translation matrixes of equation (5.1) be assembled into a global transformation matrix ${}^c\mathbf{H}_r$ in homogenous format, so that:

$$\mathbf{Q}_c = {}^c\mathbf{H}_r \cdot \mathbf{Q}_r, \tag{5.15}$$

the global transformation from the camera to the road is obtained as the product of a fixed camera to vehicle transformation and a dynamic (pitch, roll, therefore time dependent) vehicle to road transformation.

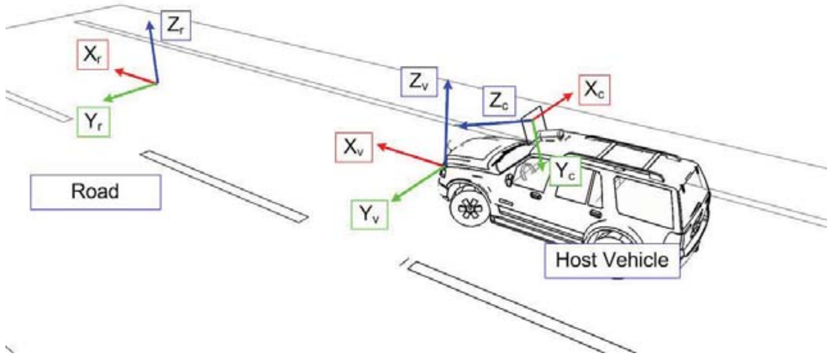


Figure 5.3: A typical road scene. The host vehicle has a camera mounted on the roof. Note that the figure shows the reference systems of both the vehicle and the road, since they may not coincide.

$$\mathbf{Q}_c = {}^{c}\mathbf{H}_v \cdot {}^{v}\mathbf{H}_r(t) \cdot \mathbf{Q}_r. \tag{5.16}$$

In the general mathematical model proposed here, the classic IPM approach may still be used: ${}^{v}\mathbf{H}_r(t)$ is constant for all values of t, i.e., the coefficients of (5.9) are defined to represent the X_vY_v plane $\Pi_{road} : a_r = b_r = d_r = 0$ and $c_r = 1$; or the road plane may be actually detected, if ${}^{v}\mathbf{H}_r(t)$ is estimated over time using stereo or laser sensors pointed towards the road, i.e., some estimation function of the parameters in (5.9) is running continuously. An example of real time estimation of road to vehicle transformation is presented in [Sappa *et al.* 2008].

5.5 Proposed Approach

IPM is the application of (5.14) to the pixels in the image. However, in a given image, not all pixels may be interesting or even possible to project. The current work addresses this problem by using a laser sensor to detect mappable regions, together with a set of criteria to select which pixels should be mapped. The following subsections present the different criteria used to find which pixels in an image are possible to be projected.

5.5.1 Half Space of Projection

Expression (5.14) is the mathematical solution of the intersection of the optical ray of a given pixel with the road plane. Because of this, a pixel above the horizon line in the image will be projected to the back of the camera's plane. Figure 5.4 shows the projection rays of two pixels, one is projectable and the other should be discarded.

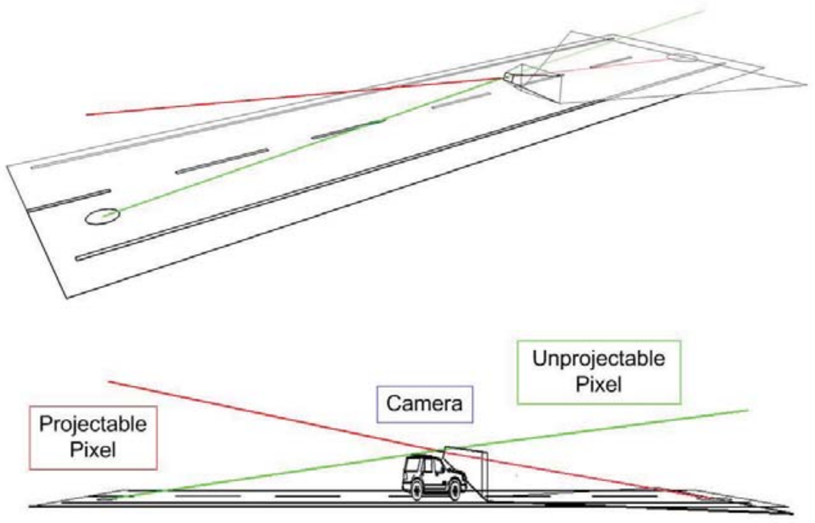


Figure 5.4: An example of a pixel that cannot be projected (*green*) since its optical ray intersects the road plane on the back of the image plane. Inversely, the pixel in *red* is projectable.

Although the presented solution is a valid mathematical solution, for the proposed model, however, the unprojectable pixels must be handled in accordance. This is done by first computing the image plane. The image plane divides the three-dimensional Euclidean space into two parts. One of them is called half space of projection. It is defined as the region of the Euclidian space where all points contained by it may be virtually projected into the image plane. The image plane is defined as $\Pi_{image} : a_i \cdot X + b_i \cdot Y + c_i \cdot Z + d_i = 0$; it is obtained as follows: Let M_0, M_1 and M_2 be three non collinear points in the X_rY_r plane of the road reference system. As an example $M_0 = [\begin{array}{ccc} 0 & 0 & 0 \end{array}]^T$, $M_1 = [\begin{array}{ccc} 1 & 0 & 0 \end{array}]^T$ and $M_2 = [\begin{array}{ccc} 0 & 1 & 0 \end{array}]^T$. The points are projected from the cameras reference frame by means of the transformation matrix defined in (5.1). In the camera's reference system, those points are contained by the image plane and may be used to define two vectors whose cross product defines the vector normal to the image plane:

$$\begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix} = \left[{}^{C}\mathbf{R}_V \cdot M_0 - {}^{C}\mathbf{R}_V \cdot M_1\right] \otimes \left[{}^{C}\mathbf{R}_V \cdot M_0 - {}^{C}\mathbf{R}_V \cdot M_2\right], \tag{5.17}$$

where $\otimes$ denotes the cross product. The remaining image plane parameter d_i is obtained by substituting in the plane equation one of the projected points:

$$d_i = -(a_i \cdot X_0 + b_i \cdot Y_0 + c_i \cdot Z_0). \tag{5.18}$$

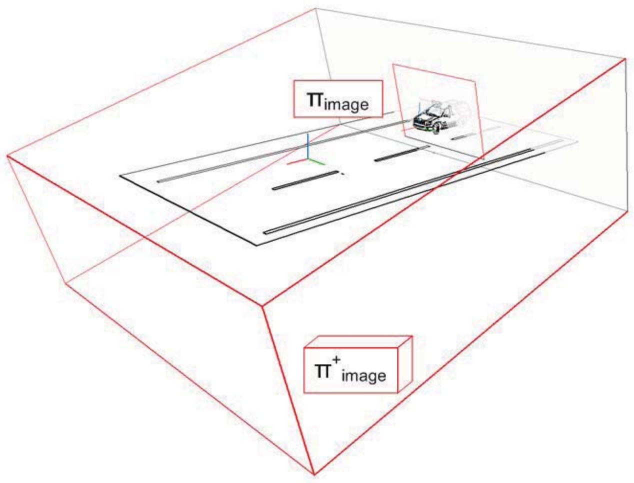


Figure 5.5: The half space of projection computed after the image plane.

Having the parameters of the image plane, and a test point $\mathbf{Q}_t = [\ X_t \quad Y_t \quad Z_t\ ]^T$ that is sure to be inside the half space of projection (for example a point a couple of meters in front of the host vehicle), a test is devised to assess if a point $\mathbf{Q} = [\ X \quad Y \quad Z\ ]^T$ belongs to the half space of projection (denoted as Π^+_{image}):

$$\begin{cases} \mathbf{Q} \in \Pi^+_{image}, & if \quad (a_i \cdot X_t + b_i \cdot Y_t + c_i \cdot Z_t + d_i) \times (a_i \cdot X + b_i \cdot Y + c_i \cdot Z + d_i) > 0 \\ \mathbf{Q} \notin \Pi^+_{image}, & if \quad (a_i \cdot X_t + b_i \cdot Y_t + c_i \cdot Z_t + d_i) \times (a_i \cdot X + b_i \cdot Y + c_i \cdot Z + d_i) \leq 0 \end{cases} . \tag{5.19}$$

The half space of projection in (5.19) is shown in Fig 5.5. It is used to define projectable polygons in 3D, as detailed in the following sections.

5.5.2 Desired Area of Perception

For an autonomous system it is important to define the area of perception that it requires to effectively navigate. A very large perception area increase the computational cost, while a small perception area might make the system unfit to handle quick variations in the road scenario. This section addresses the desired perception limits: how the programmer can effectively set an area of interest for the host vehicle to perform the IPM operation. In the case of a vehicle travelling in urban scenarios for example, perhaps 30 meters of view range are sufficient. The desired area of perception is formally defined as a polygon ψ_{dap} in the road's projection plane. This polygon must be contained in the half space of projection ($\psi_{dap} \subset \Pi^+_{image}$). Figure 5.6 shows an example of an area of perception.

Currently, ψ_{dap} is set as a four vertices polygon, defining, in the road plane, a rectangle in front of the host vehicle. The rectangle's side in the direction of the vehicle's movement may dynamically increase size in depending on the vehicle speed.

5.5.3 Image boundaries

Besides the desired area of perception, other regions of the road plane must be defined in order to perform an effective IPM operation. The camera lens properties and orientation towards the road plane define a possible area of projection. Let γ be the list of pixels in the image boundaries, obtained from all image pixels $\mathbf{q}$ that are in accordance with:

$$\mathbf{q} = \begin{bmatrix} x \\ y \end{bmatrix} \in \gamma, \quad \forall x \in \{1, W\} \vee y \in \{1, H\}, \tag{5.20}$$

where W, H are the image width and height respectively. The boundaries of the image are then projected to the road plane using the inverse projection ip from (5.14), and the real world coordinates of the image boundary pixels $\mathbf{\Gamma}$ are obtained. The half space of projection is again used to assert the validity of 3D points.

$$\mathbf{\Gamma} = ip(\gamma), \quad \forall ip(\gamma) \in \Pi^{+}_{image} \tag{5.21}$$

The list of world points $\mathbf{\Gamma}$ are used to form the vertices of the polygon ψ_{Γ} (an illustration is shown in Fig. 5.7).

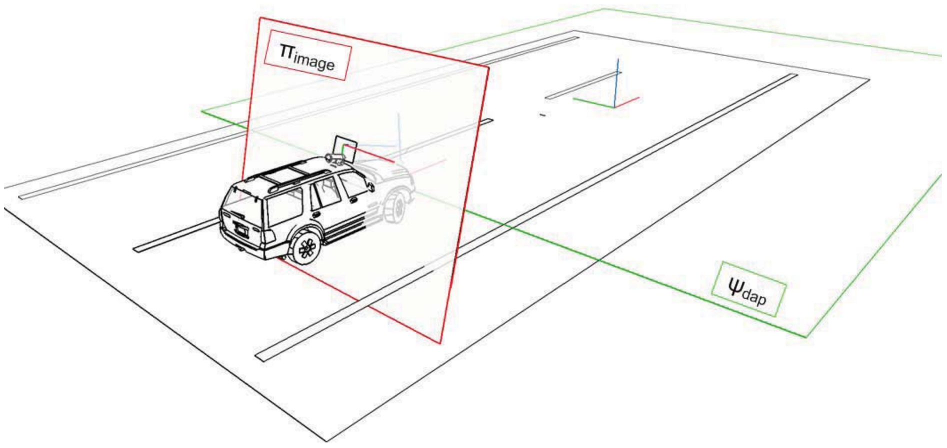


Figure 5.6: The desired area of perception, polygon (ψ_{dap}) in *green*. All vertices of this polygon should contained by the half space of projection, according to (5.19).

5.5.4 Laser Generated Polygon

The IPM technique requires that the road surface seen from the cameras is flat. This might not always be the case, particularly when other vehicles or obstacles lie on the road, as shown in the IPM resulting images published by some authors [McCall & Trivedi 2005, Bertozzi & Broggi 1998, Aly 2008]. In these examples, artifacts are generated in the regions of the image where the flat road assumption fails. Vehicles are mapped as if they had been painted on the road (see Fig. 5.2 (*b* and *d*)). Some authors have taken advantage of this phenomenon to detect obstacles in the road, by using the differences in two IPM images, from a pair of stereo cameras [Bertozzi & Broggi 1998]. This method is called stereo IPM. Although the latest is a valid approach, the fact is that calibration issues tend to disrupt the perfect mapping of stereo images. Because of this, it may sometimes be difficult to distinguish if disparities in the IPM stereo are due to a sub-optimum calibration or to an obstacle that lies on the road surface. There is also work related to sensor integration using both vision and laser in autonomous vehicles [Broggi *et al.* 2006], but in this case the objective was to enhance obstacle detection. Figure 5.8 shows a typical urban road scenario with several obstacles near the host vehicle.

Let $\mathbf{Q}_{laser} = [\ X_{laser} \quad Y_{laser} \quad Z_{laser}\ ]^T$ be the 3D points obtained by the LRF, referenced in the world coordinate system. Assuming that objects picked up by the laser are vertical, the coordinates where obstacles touch the floor, i.e., the object baseline $\mathbf{Q}_{bln}$, is obtained by the vertical projection of laser points onto the road plane:

$$\mathbf{Q}_{bln} = \begin{bmatrix} X_{laser} \\ Y_{laser} \\ -\dfrac{(a_r \cdot X_{laser} + b_r \cdot Y_{laser} + d_r)}{c_r} \end{bmatrix}. \tag{5.22}$$

The laser generated polygon ψ_{laser} is defined by the list of vertices at generic coordinates given

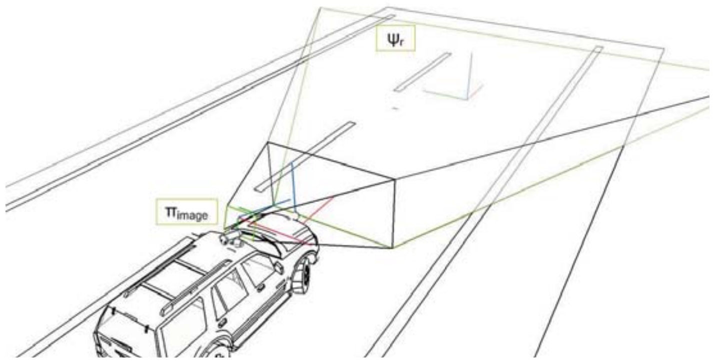


Figure 5.7: The projection of image boundary pixels onto the road plane results in the image boundaries polygon (ψ_Γ).

by $\mathbf{Q}_{bln}$.

5.5.5 Image Projection Polygon

As stated in the beginning of this chapter, the core of IPM method is the application (5.14) to the pixels in the image that are known to be on the projection plane. The objective is to be able to define for the input image which pixels are possible (and desirable) to map. The proposed approach defines three polygons in the road plane: a polygon defining the desired area of perception (ψ_{dap}), a polygon corresponding to the boundaries of the image (ψ_{Γ}) and a polygon defining the laser scanned objects (ψ_{laser}). The resultant projection polygon ($\psi_{projection}$) is obtained by the intersection of the three other polygons:

$$\psi_{projection} = \psi_{dap} \bigcap \psi_{\Gamma} \bigcap \psi_{laser}, \tag{5.23}$$

where $\bigcap$ represents polygon intersection. The projection polygon is composed of a list of vertices, i.e., 3D points defined the road reference system. The vertices defined in the road reference system are direct projected to the image plane using eq. (5.8). The result is a list of 2D vertices that define a polygon in the image plane. This is called the image projection polygon. Inside the polygon are all pixels that should be mapped using IPM. Since perspective transformation is an affine transformation, the image projection polygon is calculated as the direct projection of the vertices of the projection polygon in the road plane.

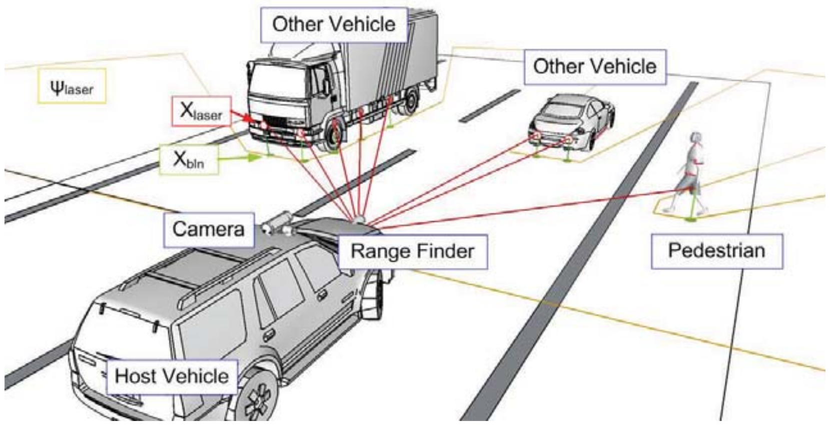
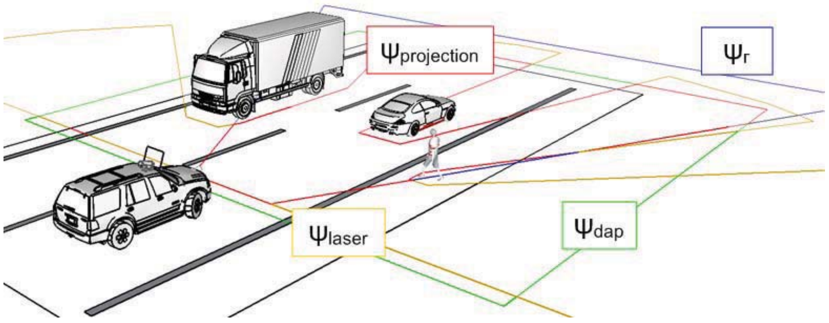
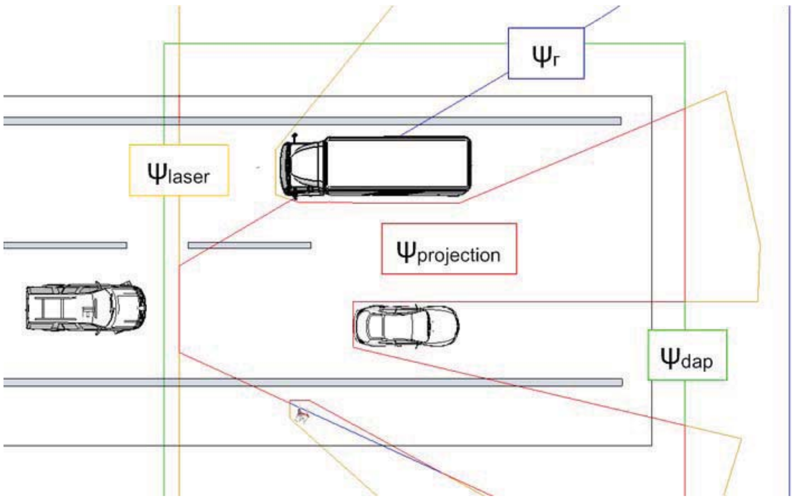


Figure 5.8: A typical urban road scenario with several obstacles near the host vehicle.



(a)



(b)

Figure 5.9: A road scenario with several obstacles: isometric view (a) and top view (b). The projection polygon ($\psi_{projection}$) is shown in red. It is obtained by the interception of the desired area of perception (ψ_{dap}) in green, the image boundaries polygon (ψ_{Γ}) in blue and the laser generated polygon (ψ_{laser}) in yellow.

5.6 Results

Several experiments have been devised to obtain quantitative results of the proposed IPM methodology. First, the platforms used to obtain the results are presented: a dual camera pan and tilt unit (*PTU*), a small scale robot and finally, a full scale autonomous vehicle. The computational performance of the proposed approach is compared to the classic IPM using a measure of the accuracy of IPM. Results are presented for the accuracy of the proposed approach. Also, a comparative study of the classic IPM versus the laser assisted IPM shows that the accuracy of the latest is much better when obstacles appear in the area of projection. Finally, this section ends with some qualitative results, providing images of IPM from onboard cameras of a full scale vehicle.

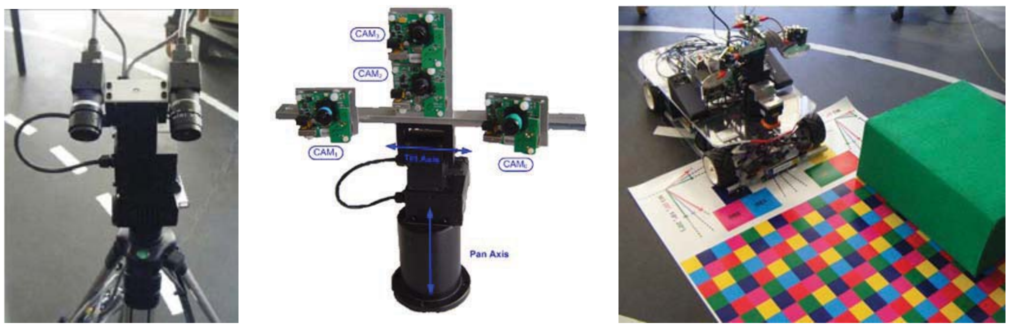
5.6.1 Test Platforms

In order to assess the performance of the proposed methodology, the *AtlasMV* and *AtlasCar* test platforms where used (Fig. 5.10).

Figure 5.10 *(a)* shows the dual camera PTU. The servo actuated PTU controlled through RS232 serial protocol was used so that the IPM is tested when the cameras move into different positions. The cameras have different lenses and also different image resolution. *Camera 0* has a wide angle lenses and a resolution of 800×600 pixels, while *Camera 1* has a tele lens and a resolution of 320×240 pixels. This platform is used to assess the computational performance of the proposed approach. The time taken to perform IPM on both cameras is measured, during a test where the PTU moves the cameras to different positions.

Figure 5.10 *(c)* shows the *ATLASMV* robotic platform. It is a small scale autonomous robot built for participating in an autonomous driving competition. It is equipped with four cameras and a LRF. The side cameras (Fig. 5.10 *(b)*), used to map the road in front of the robot, have wide angle lenses and produce images with a resolution of 320×240 pixels. The *ATLASMV* is used in two tests: one for measuring the accuracy of IPM, another to assess the effects of using the LRF to assist IPM. The quantitative results obtained from the accuracy of IPM are calculated using a color calibrated grid (shown in Fig. 5.10 *(c)*, below the robot and, in Fig. 5.11, viewed from the cameras).

The grid is a 3×1 meters sheet of chapter marked with a special colored pattern. The grid is positioned in a known position in front of the cameras. Using the position and rotation of the cameras with respect to the calibration grid, a virtual image of the grid is produced to overlap the resultant IPM image. This virtual image of the grid serves as a test mask for measuring the IPM accuracy (η_{IPM}): after projection using IPM, pixels are labelled with a color that should match the color of the virtual image. The accuracy of the projection is obtained as the ratio between correctly projected pixels and



(a) (b) (c)

Figure 5.10: Two of the test platforms used for testing the proposed approach: dual camera PTU unit (a); the ATLASMV small scale robot (c), equipped with a LRF and a multi-camera perception unit (b).

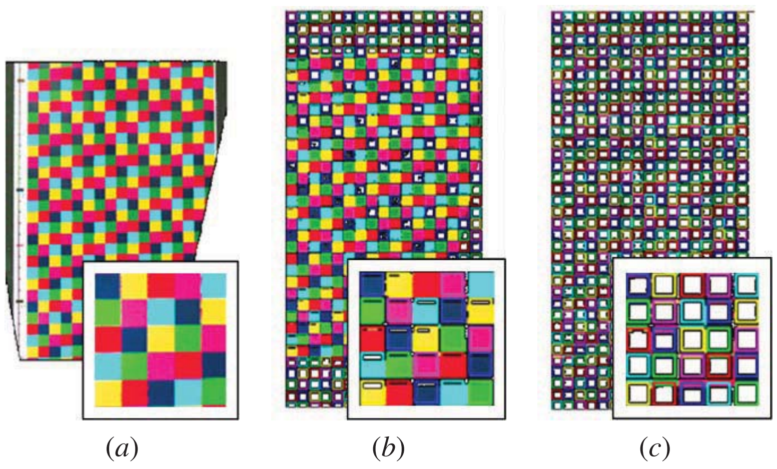


Figure 5.11: The entire projection obtained using *Camera 0* of the *ATLASMV* test platform *(a)*. The correctly projected pixels *(b)*. Pixels that where incorrectly projected *(c)*. Bellow each image, an enlarged region of the pixels in shown.

the total projected pixels:

$$\eta_{IPM} = \frac{number\_of\_correct\_projections}{total\_number\_of\_projections} \tag{5.24}$$

Figure 5.11 *(a)* shows all the pixels of a given projection. Pixels classified as correctly and incorrectly projected are displayed in Fig. 5.11 *(b)* and *(c)* respectively. Also, the virtual grid is overlayed onto the images.

5.6.2 Computational Performance

The computational performance of the IPM transformation has been a concern of some authors [Bertozzi *et al.* 1997] [Evans & Kim 1998]. Its implementation on onboard systems requires real time performance from the systems. In order to test the performance of the proposed approach, the dual camera PTU setup was used (Fig. 5.10 *(a)*).

In a classic IPM, all pixels in a given image are inverse projected, i.e., eq. (5.14) is applied to all pixels. On the other hand, the proposed approach first computes the image projection polygon, and then applies (5.14) only to the pixels that should be projected. The computational demand of an IPM operation depends on the amount of projected pixels, which in turn depends on the camera's pose towards the projection plane. For example, a camera pointing to the sky will have only a small amount of pixels viewing the road plane.

To compare the performance of classic IPM with the proposed approach a 14 second test sequence was devised. Since the orientation of the camera's towards the road plane changes the amount of projectable pixels, during the 14 seconds of the test, the PTU is ordered to go to specific positions:

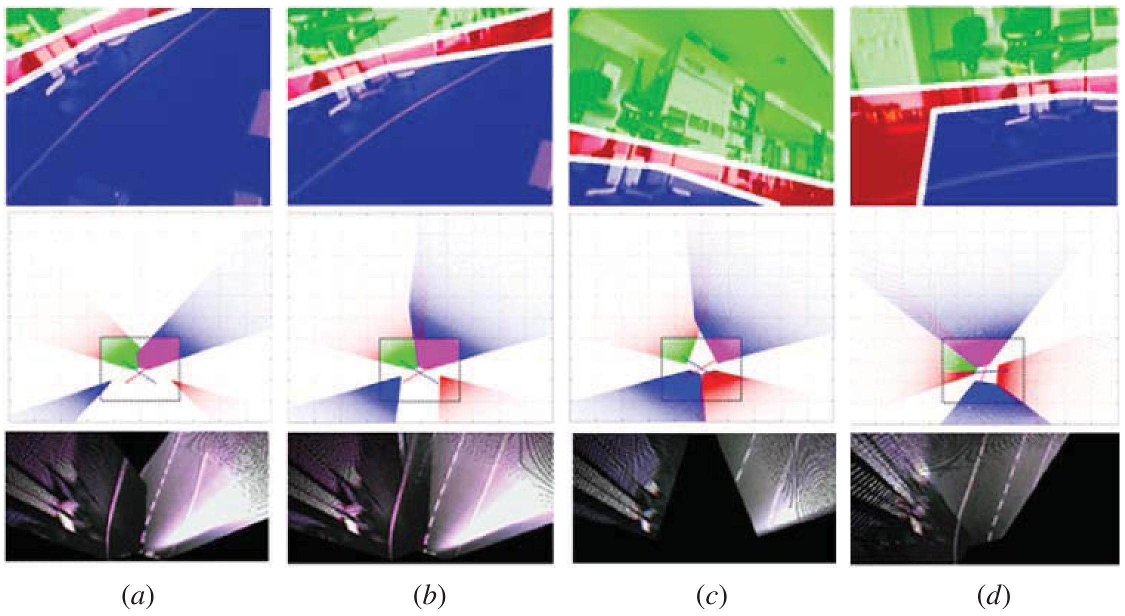


Figure 5.12: Some key frames of the test sequence. *First row*: images taken from *Camera 0*, the blue area is the area of projected pixels, the red is the area outside the desired area of perception and the green area is the area outside the half plane of projection; *Second row*: a map of the projection. Projected/unprojected pixels from *Camera 0* in green/red. Projected/unprojected pixels from *Camera 1* in magenta/blue; *Third row*: the IPM resulting image after mapping both cameras. In columns, different snapshots of the test sequence: 0 (*a*), 2 (*b*), 5.5 (*c*) and 12 (*d*) seconds.

- State 1 (0-5.5 seconds) the PTU is moving upwards. This causes an increasingly smaller amount of mappable pixels for both cameras;
- Stage 2 (5.5-8.5 seconds) moves the PTU down and the inverse phenomena occurs;
- Stage 3 (8.5-14 seconds) maintains a fixed tilt and the PTU pans increasingly to the left, which will make *Camera 1* to have increasingly less mappable pixels.

Figure 5.12 shows some IPM resulting images of key points in the test sequence. Figure 5.13 (*a*) compares the projection time of both cameras using the classic IPM and the proposed approach. Figure 5.13 (*b*) indicates the amount of projected pixels and the time saved using the proposed approach in relation to the classic IPM.

From 0 to 5.5 seconds, the PTU is moving upwards and so the pitch angle of the cameras is changing. This is observable in the difference of mapping in Fig. 5.12 (*a*), (*b*) (*c*). Figure 5.13 (*b*) shows a reduction in the number of projected pixels for each camera. In Fig. 5.13 (*a*), a reduction of IPM projection time using the proposed approach is clearly noted. *Camera 0* takes more time to project than *Camera 1* because the resolution of the images is different (800 $\times$ 600 pixels and 320 $\times$ 240 pixels, respectively).

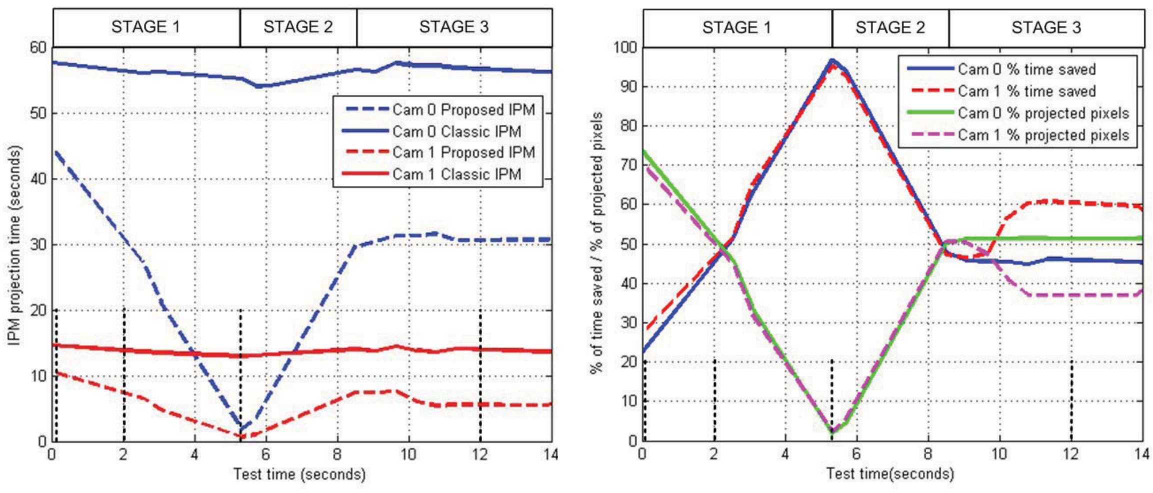


Figure 5.13: (*a*) Time taken to perform the IPM projection for both cameras. The classic IPM time and the time of the proposed approach are shown. (*b*) Percentage of time saved and number of projected pixels. Proposed approach compared to the classic approach. Key frames of Fig. 5.12 signaled as the vertical black lines.

From 5.5 to 8.5 seconds the PTU is moving downwards and the effects are the inverse.

From 8.5 to 14 seconds the change in pan angle causes *Camera 1* to view increasingly less of the desired area of perception. Figure 5.13 (*b*) shows a decrease in the number of projected pixels during this period.

5.6.3 IPM Accuracy

Although many researchers have employed the IPM operation in order to ease the road recognition process, e.g., [Bertozzi & Broggi 1998], [Bertozzi *et al.* 1997], [Broggi *et al.* 2006] or [Broggi *et al.* 2008] , the fact is that no reporting of the accuracy of each implementation was found in the literature. Despite some insights on the topic of accuracy measurement for general projective geometry [Evans & Kim 1998], [Fang *et al.* 2009], a method had to be devised for this particular application to provide a quantitative analysis of the proposed method. For this experience the dual camera PTU setup was used (Fig. 5.10 *(a)*). The calibration grid presented in section 5.6.1 was employed and an accuracy of $\eta_{IPM} = 0.85$ was achieved for the system. Because the current chapter is the first to present such quantitative results, a measure of the quality of this value is not possible.

The second experiment is intended to assess how important is to have accurate measures of the camera's position and orientation with the road plane. In other words, how does the uncertainty of the camera pose estimation reflect on the final accuracy of the projection. For this purpose, errors in the yaw and pitch angles of the camera were introduced, and the IPM accuracy was calculated. Figure 5.14 shows the resulting IPM of mappings with some errors (*a*), (*b*), (*d*) and (*e*) and the resulting

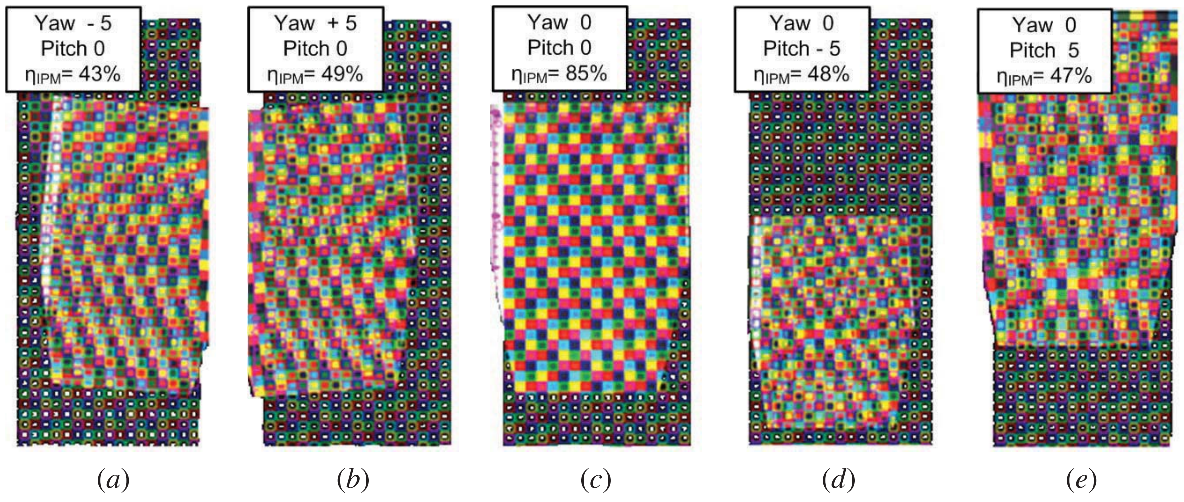


Figure 5.14: The resulting IPM projection when errors in yaw (*a*), (*b*) and in pitch (*d*), (*e*) are introduced in the calculation. The reference projection, where no errors were introduced, is shown in (*c*).

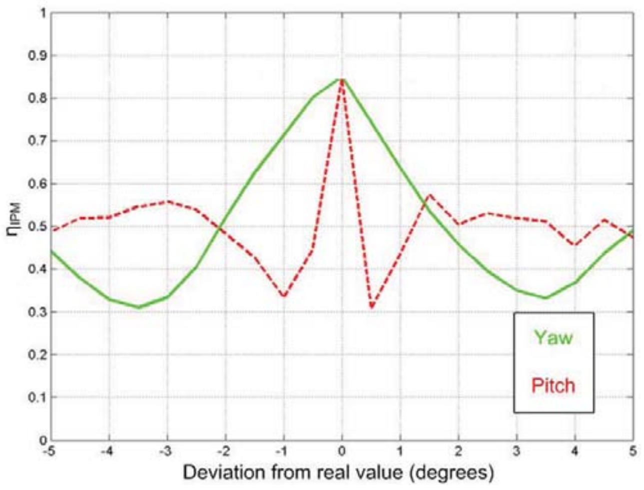


Figure 5.15: IPM accuracy (η_{IPM}) scores for errors in camera pose. Results are presented for errors in yaw and pitch angles.

image with no errors (*c*).

Figure 5.15 shows the decrease in IPM accuracy with the increase of error in yaw and pitch. The pitch angle is the most relevant for the projection accuracy, since a half degree error changes the accuracy from 0.85 to 0.30. Variations in yaw also drop the accuracy value to 0.30, but only after a 3.5 degree deviation. This is consistent with the concerns of several researchers worldwide that mention onboard camera's pitch estimation to be a cumbersome problem. In [Thrun *et al.* 2006a], for example, it is stated that "a small error in the vehicle's roll/pitch estimation leads to a massive terrain classification error forcing the vehicle off the road. Such situations occur even for roll/pitch errors below 0.5 degrees".

5.6.4 IPM Accuracy using Laser Range Finder

In order to test the usage of the LRF on the IPM projection the *ATLASMV* robot was used. An obstacle with 0.2 meters height (green box in Fig. 5.10 (*c*)) was placed over the calibration grid in front of the robot at several distances and in several positions (to the left, right or in front of the robot). For each

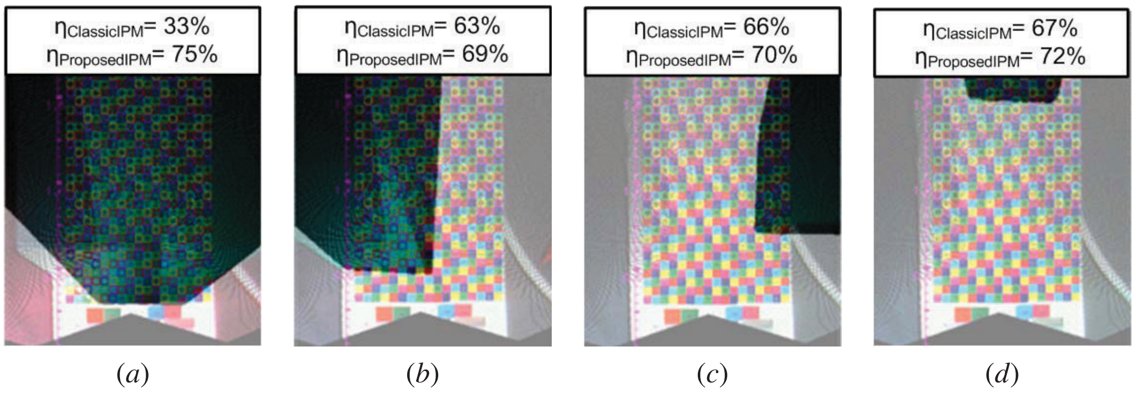


Figure 5.16: Obstacle positioned at: (*a*) 0.3 meters in front; (*b*) 0.5 meters to the left; (*c*) 0.75 meters to the right; and (*d*) 1.5 meters in front.

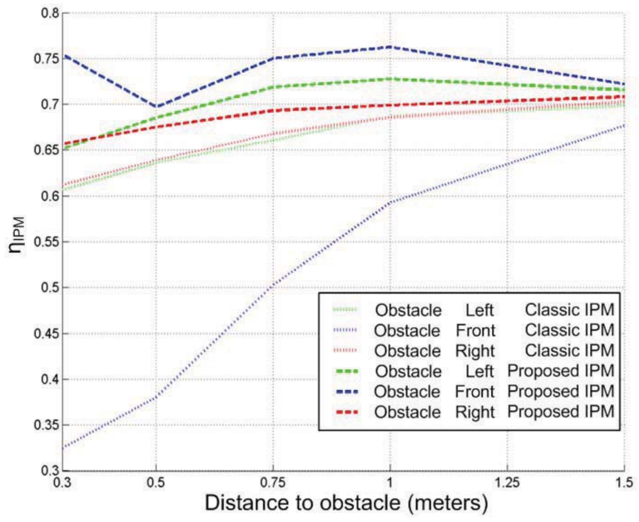


Figure 5.17: IPM accuracy (η_{IPM}) for the classic IPM (*dotted lines*) and the proposed approach (*dashed lines*)

obstacle position the accuracy was computed. Figure 5.17 shows the η_{IPM} results both for the classic and the proposed IPM approach.

The laser polygon introduced in section 5.5.4 is able to classify pixels that view the obstacle as unmappable. Because of this, the proposed IPM approach (Fig. 5.17, dashed lines) consistently gets better accuracy results than the classic IPM (Fig. 5.17, dotted lines). When the obstacle is very close (0.3 meters, Fig. 5.16 (*a*)), using a classic IPM operation would be catastrophic (0.33 accuracy ratio) but the proposed approach remains accurate enough (0.75). Figure 5.16 shows the IPM resultant images for some of the tested scenarios.

5.6.5 Tests in Real Environments

For the final validation of the proposed approach, several tests in real road scenarios with a full scale vehicle were done. The test platform used was the *AtlasCar* (Fig. 3.17). The platform is equipped with three cameras (each with a different focal distance lens) and several lasers. Several hours of data from urban and highway roads were used for validating the algorithm. The proposed approach is less time consuming, is able to deal with pitch/roll variations due to brake/turning maneuvers, and using the LRF copes with obstacles present in the projection area. Figure 5.18 provides a qualitative comparison of the classical IPM with the proposed multi-modal IPM. It is possible to observe that in the presence of other vehicles or obstacles, the classical IPM produces several artifacts on the resultant image. On the contrary, the multi-modal approach to IPM is able to cope with obstacles and removes them from the resultant image. Even in free road scenarios, as is the case of Fig. 5.18 (*fourth line*), the artifacts produced by the parked cars could reduce the effectiveness of a road detection approach.

The flexibility of the proposed approach can also handle the usage of several input cameras. In Fig. 5.19, the three cameras onboard the *AtlasCar*, each with different focal distance lenses, are used to obtain a more detailed mapping of the road. Figure 5.19 (*a*) shows images from the three cameras. The IPM is mapped to the road plane and the distribution of pixels supplied by each camera is shown in Fig. 5.19 (*b*). Using a single camera to map the road (the green camera), shows the classical problems of lack of accuracy at long distances (the yellow traffic pattern in Fig. 5.19 (*c*)). However, if multiple cameras are employed, the tele-camera (blue camera) can provide a high resolution view at long distances, which leads to a high resolution view of the yellow pattern of the road (Fig. 5.19 (*d*)).

Several video sequences showing results from classical and proposed IPM can be found at http://lars.mec.ua.pt/public/Media/mriem/ipm/.

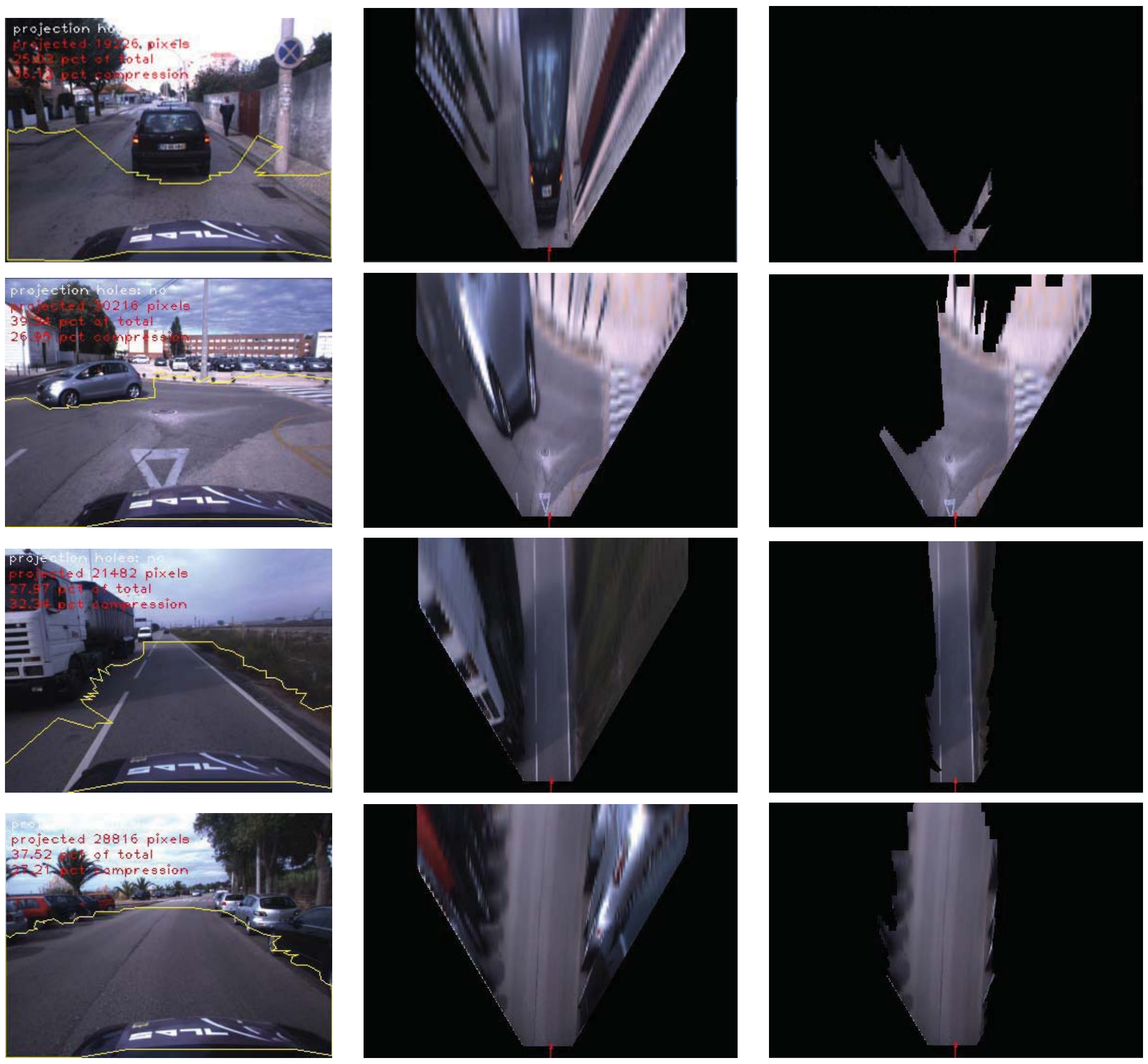


Figure 5.18: Comparison of the classical IPM (*middle column*) with the proposed multi-modal IPM (*right column*). The input image (*left column*) shows the image projection polygon highlighted in yellow. Four examples are shown, one in each row.

5.7 Conclusions

The current chapter presents a flexible mathematical model to perform IPM. The model is able to cope with pitch and roll interferences from the host vehicle by decoupling the road reference frame from the vehicle reference frame. Polygons generated from the image boundaries, the laser obstacles and the desired area of perception are computed in the road plane. The combination of these polygons, which is called the projection polygon, is then directly projected to the image plane and the image projection polygon is obtained. The image projection polygon indicates, in the image coordinate

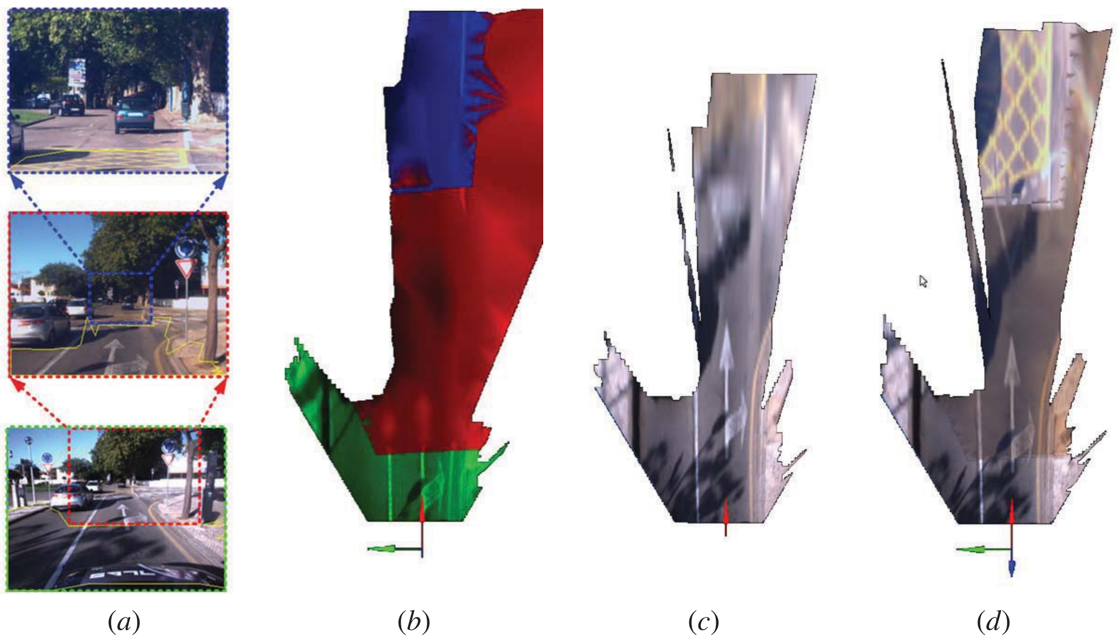


Figure 5.19: Using the proposed IPM approach in real scenarios. (*a*) Images of the three cameras onboard the *AtlasCar*; (*b*) The distribution of mapping for each camera ; (*c*) IPM using just green camera; (*d*) IPM using all cameras

system, which pixels are to be mapped through IPM, saving computation time.

Different test platforms (from a small scale robot to a full scale vehicle) were used to obtain quantitative and qualitative results. Results show that the proposed approach is computed in less time than the classic IPM, and that the proposed approach has higher IPM accuracy in the presence of obstacles. A study of the influence of errors in the camera's pose estimation to the IPM projection accuracy is also presented. Finally, several hours of data both from urban roads and highways were qualitatively analyzed and show that the proposed approach is robust and efficient.

Chapter 6

Photometric Calibration

This chapter presents three alternative methodologies for computing a photometric calibration between image pairs. Photometric calibration, also known as color correction, is an important step towards the objective of obtaining fine quality multi-image mosaics.

This chapter starts with an introduction to the problem of photometric calibration (section 6.1), followed by the state of the art on the topic of photometric registration (section 6.2). Then, three alternative approaches to the problem of color correction are presented. In section 6.3 a Mean shift based approach is proposed, section 6.4 presents an alternative method based on the usage of 3D Gaussian mixture models and, finally, section 6.5 proposes a probabilistic approach to the problem of color correction using sets of truncated Gaussians. In all three sections, results will be presented to compare the effectiveness of each algorithm with the state of the art. Finally, conclusions are given in section 6.6.

6.1 Introduction

Image mosaicking applications require both geometrical and photometric registrations between the images that compose the mosaic. In this chapter, several different approaches to the problem of correcting the photometric disparities of the images that compose a mosaic. Recent years have proven the importance of image mosaicking. This area of research and other similar variations such as image compositing and stitching have found a vast field of applications ranging from satellite or aerial imagery [Soille 2006] to medical imaging [Duncan & Ayache 2000], street view maps [Vincent 2007], city 3D modeling [Micusik & Kosecka 2009], super-resolution [Ben-Ezra *et al.* 2005], texture synthesis [Lempitsky & Ivanov 2007] or stereo reconstruction [Li *et al.* 2004], to name a few. In general, whenever merging two or more images of the same scene is required for comparison or integration purposes a mosaic is built. Two problems are involved in the computation of an image mosaic: the geometric and the photometric correspondence [Levin *et al.* 2003]. The geometric correspondence is usually referred to as image registration and is the process of overlaying two or more images of the same scene taken at different times, from different viewpoints and by different sensors. The procedure geometrically aligns two images [Zitová & Flusser 2003]. This problem has been extensively studied and is out of the scope of the this chapter. In this chapter it is assumed that the given images are geometrically registered. It should be noted, however, that in most cases the alignment that is produced by a registration method is never accurate to the pixel level. Hence, a pixel to pixel direct mapping of color is not a feasible solution. On the other hand, the photometric correspondence between images deals with the photometric alignment of image capturing devices. The same object, under the same lighting conditions, should be represented by the same color in two different images. However, even in sets of images taken from the same camera, the colors representing an object may differ from picture to picture. This poses a problem to the fusion of information from several images. Hence, the problem of how to balance the color of one picture so that it matches the color of another must be tackled. This process of photometric alignment or calibration is referred to as color correction or between images.

6.2 Related Work

The general problem of compensating the photometric disparities between two coarsely geometrically registered images is referred to as color correction. In other words, color correction is the problem of adjusting the color palette of an image using information from the color palette of another image. Let two images of the same scene be referred to as source ($\mathbf{S}$, Fig. 6.1 (*a*)) and target ($\mathbf{T}$, Fig. 6.1 (*e*)) images. Now let a geometric registration between these images be given, so that it is possible to build a mosaic of the scene. Note that in the overlapping area of the mosaic in Fig. 6.1 (*f*), the color of a pixel is given by the average of the $\mathbf{S}$ and $\mathbf{T}$ image pixels. Nonetheless, a clear photometric miss registration between the images in the overlapping area can be appreciated. Color correction tackles the problem of how to adjust the colors of $\mathbf{T}$, so that they resemble to the colors in $\mathbf{S}$. The color adjusted $\mathbf{T}$ image is called corrected image and is referred to as $\hat{\mathbf{T}}$. A mosaic built using $\mathbf{S}$ and $\hat{\mathbf{T}}$ should present smooth color transitions, as in the example of Fig. 6.1 (*g*). Let two new images, $\mathbf{S}_p$ and $\mathbf{T}_p$, be the portions of $\mathbf{S}$ and $\mathbf{T}$ that overlap the area containing mutual information, shown in Fig. 6.1 (*b*) and (*c*), respectively. From this overlapped area a color correction method computes an estimation of one or more color palette mapping functions. By applying these functions to all the pixels of $\mathbf{T}$, the color corrected image $\hat{\mathbf{T}}$ is obtained. The objective of any color correction approach is to provide an estimation for this mapping function, denoted as color palette mapping function, that maximizes the similarity between the $\mathbf{S}$ and $\hat{\mathbf{T}}$ images.

In most of the proposed color correction algorithms, the color correction operation is performed independently for each color channel. In this sense, color correction (correcting the three color channels) and brightness correction (correcting a single grayscale image) are similar. Throughout this paper, the methodology for correcting a single channel is explained. All the examples shown in graphs and plots are given for the red channel of the images. The process should be applied similarly for the green and blue channels. The term color or pixel color will be used throughout the remainder of the paper referring to the intensity value of that pixel for a particular color channel. Since color correction is done independently for each channel, some authors have proposed to use color spaces where cross channel artifacts are less prone to occur. For example in [Reinhard *et al.* 2001] and [Tai *et al.* 2005], the images are first transformed to the $l\alpha\beta$ color space before performing independent channel color correction. In [Zhang & Georganas 2004], the CIECAM97 color space is employed and in [Fecker *et al.* 2008] the chosen color space was YCbCr. On the other hand, many other proposals use the standard RGB color space [Jia & Tang 2005] [Xiao & Ma 2006] [Kim & Pollefeys 2008] [Pitie *et al.* 2005]. In the current paper, the RGB color space is used. Despite this, it should be noted that the methodology here proposed works also on other color spaces. However, an analysis of the effects of using different color spaces is out of the scope of the current paper.

The estimated color palette mapping function should be as similar as possible to an ideal mapping function. However, since the ideal mapping function is unknown in most cases, it is not possible to measure the quality of a color correction method by comparing the estimated and the ideal mapping functions. The alternative is to analyze the output of the algorithms, i.e., to compare photometrically the corrected image with the source image. This comparison is only possible in the overlapping regions of the images.

In general, the color correction methods proposed in the literature can be divided into model-based parametric approaches, i.e., where the color distribution of the images is assumed to have some statistical distribution, and modeless, non-parametric approaches, i.e., where no assumptions about

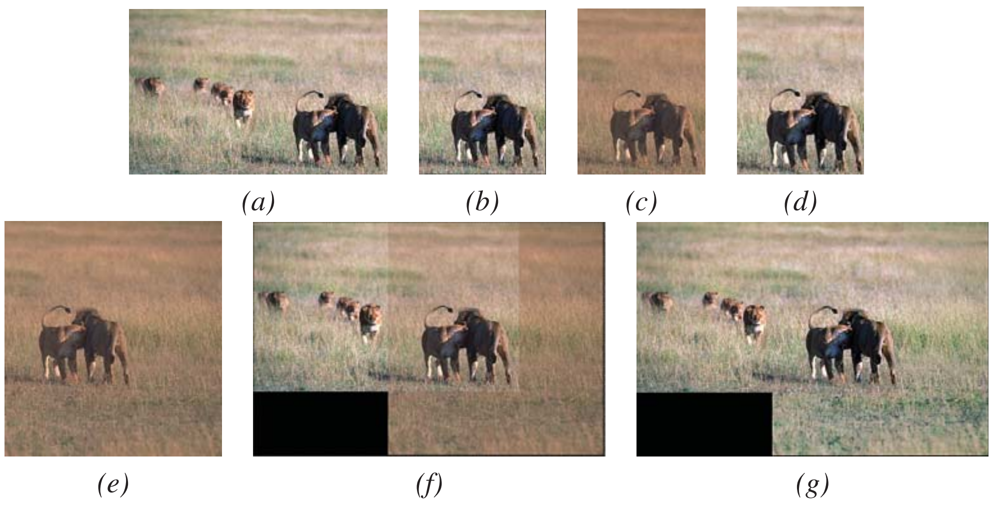


Figure 6.1: Images involved in a color correction procedure: (*a*) source image ($\mathbf{S}$); (*b*) overlapping area of the source ($\mathbf{S}_p$); (*c*) overlapping area of the target ($\mathbf{T}_p$); (*d*) overlapping area of the corrected image ($\hat{\mathbf{T}}_p$); (*e*) the target image ($\mathbf{T}$); (*f*) mosaic of source and target; (*g*) mosaic composed of the source and corrected images.

the nature of the color distribution are taken [Xu & Mulligan 2010].

In [Reinhard *et al.* 2001], a simple statistical distribution transfer methodology was proposed. The color distribution of the **S** image is transferred to the **T** image by scaling and offsetting according to the mean and standard deviations. It was also in [Reinhard *et al.* 2001] that an alternative color model, namely the $l\alpha\beta$ color-space, was proved to be more effective for calculating the color transfer functions than the usual RGB color-space. It is successfully employed since it minimizes the cross channel correlation, which is present on many color spaces. This work has been extended in [Xiao & Ma 2006], where tools that permitted RGB color space to be used with similar effectiveness were presented. These global methods use the entire image to collect statistical information i.e., they assume a single color palette mapping function. In complex scenes this assumption does not hold due to differing optics, sensor characteristics, and hardware processing employed by video cameras [Yin & Cooperstock 2004]. Because of this, the approach presented in [Reinhard *et al.* 2001] was latter improved with the proposal of segmenting the images into several regions [Tai *et al.* 2005] [Hsu *et al.* 2005] [Xiang *et al.* 2009] [Tai *et al.* 2007]. Each region provides the data to compute a local color palette mapping function, which is then applied to the pixels of those regions. An Expectation Maximization color segmentation algorithm was proposed in [Tai *et al.* 2005] and latter extended in [Xiang *et al.* 2009]. The local approaches improve the statistical parameters required for the mapping proposed by [Reinhard *et al.* 2001]. In [Oliveira *et al.* 2011], an alternative Meanshift [Comaniciu & Meer 2002] based color segmentation method was proposed. Note however that because the accuracy of the geometric registration between **S** and **T** does not allow a direct mapping of color between pixels, there is a limit for the number of regions that may be segmented. Very small regions or, by absurd, one pixel sized regions, will fail to properly color correct the image because they will not be able to cope with the color mapping noise produced by a less than perfect registration. Other model based approaches include Principal Component Analysis [Zhang & Georganas 2004] [Abadpour & Kasaei 2007], and gain compensation methodologies [Brown & Lowe 2007].

Some authors suggested to solve the problem by using non parametric approaches. For instance in [Jia & Tang 2005], color correction is done through the estimation of global and local color transfer functions. The complex estimation problem is reduced to a robust 2D tensor voting in the corresponding voting spaces. In [Sajadi *et al.* 2010] higher-dimensional Bezier patches were used to represent color transfer functions. Histogram based approaches were presented in [Fecker *et al.* 2008] and [Tian *et al.* 2002]. In [Pitie *et al.* 2005] and [Pitie *et al.* 2007] the entire probability density function is mapped without making assumptions on its nature. In [Qian *et al.* 2010] and [Abadpour & Kasaei 2004], fuzzy logic based methodologies are proposed. In [Siddiqui & Bouman 2008] a multi-layer hierarchical stochastic framework is presented. Other approaches make use of multi channel image blending [Brown & Lowe 2007] [Li *et al.* 2008]. The modeling of the vignetting effects has also been the concern of several authors [Kim & Pollefeys 2008] [Zheng *et al.* 2009] [Hasler & SÃijsstrunk 2004]

[Litvinov & Schechner 2005b] and [Litvinov & Schechner 2005a]. Finally, some other authors proposed to use learning approaches (e.g., neural networks) to find the appropriate color palette mapping functions for performing color correction [Nayak & Chaudhuri 2006] [Yin & Cooperstock 2004]. However, learning approaches have the limitation of requiring a specific training for different setups.

The performance evaluation done in [Xu & Mulligan 2010] used more than 60 image pairs and compared several methodologies. In that evaluation, the methods that presented the best color correction performances were [Kim & Pollefeys 2008] and [Jia & Tang 2005]. The proposed methodology contains significant differences from both: [Kim & Pollefeys 2008] was devised to color correct mosaics of images taken from a single camera, while the proposed approach is designed to operate on mosaics of images that can be taken from several cameras. Under the assumption that both the source and target images are from the same camera it makes sense to attempt to model each camera's full radiometric behavior as done in [Kim & Pollefeys 2008]. In our case we do not assume images come from a single camera and therefore cannot expect to model the camera. Instead we try to model the observed color transformation and then apply this model of the transformation to color correct the images. We do not know whether the images in the tested data-sets are from single or multi cameras, but we believe that this extra flexibility from the proposed approach explains why it obtains better results that [Kim & Pollefeys 2008]. The method presented in [Jia & Tang 2005] performs an analysis of the joint image histogram. The same is proposed in this paper. However, the reason why our approach outperforms [Jia & Tang 2005] is that it employs a probabilistic approach to infer the color mapping functions, instead of a voting scheme. In a voting scheme the importance is given to the most common mapping, which may not always be the best solution. The following sections will provide the details of the proposed approach.

The problem of color correction has been widely studied during the last decade. Some authors suggested to solve this problem by using non parametric approaches, i.e., methods that make no assumptions about the nature of the color distribution. While some authors have attempted to model the radiometric response functions of the sensors and the exposures [Tsin *et al.* 2001, Mitsunaga & Nayar 1999, Mann & Mann 2001], others inclusively model the vignetting phenomenon [Yu *et al.* 2004, Zheng *et al.* 2008] and there are some others that provide techniques to model the combination of both [Kim & Pollefeys 2008, Litvinov & Schechner 2005b, Litvinov & Schechner 2005a]. This trend is usually addressed just for intensity images and so is not considered in color correction. In [Jia & Tang 2003, Jia & Tang 2005], color correction is done through the estimation of global and local color transfer functions. The complex estimation problem is reduced to a robust 2D tensor voting in the corresponding voting spaces. A cumulative histogram matching technique was presented in [Fecker *et al.* 2008], while in [Pitie *et al.* 2005] the entire probability density function is mapped without making assumptions on its nature. On the other hand, model based parametric approaches try to model the color distribution in the images and use tools that transfer the color distribution characteristics from one image to the other. One of the most important works in this scope is [Reinhard *et al.* 2001]. In that paper, a simple statistical distribution transfer methodology was proposed. It was also in [Reinhard *et al.* 2001] that an alternative color model, namely the $l\alpha\beta$ color-space, was proved to be more effective for calculating the color transfer functions than the usual RGB color-space. It is successfully employed since it minimizes the cross channel correlation, which is present on many color spaces. This work has been extended in [Xiao & Ma 2006], where tools that permitted RGB color space to be used with similar effectiveness were presented. Principal component analysis where implemented by [Zhang & Georganas 2004]; while in [Brown & Lowe 2003, Brown & Lowe 2007] a gain compensation algorithm and a multi-band blending post processing was proposed.

Although much work has been done in this topic, the fact is that there is still space for improvement of color correction methodologies. In general, the size of the data sets used for evaluating the performance of the proposed methods is relatively small. In fact, most proposed approaches show the results of their algorithm just for the images depicted in the paper, which are never more than five or six. As an exception, the performance evaluation done in [Xu & Mulligan 2010] used 40 synthetic and 30 real image pairs. For performance evaluation, the image pairs (the target and source) must be properly registered. For synthetic image pairs, the target image is clipped from the source image and then its colors are altered. Using the information from the clipped region, the transformation matrix from target to source image is accurate. In the case of real image pairs, manual or automatic registering [Brown & Lowe 2007] is used, which of course will never be entirely accurate.

6.3 Mean Shift Based Local Color Correction

The first proposed algorithm can e classified as a parametric approach to color correction. The algorithm is composed of four different stages, which are applied consecutively to the images. Initially, the target image is segmented into a set of regions according to their color information. In the current version the Mean shift algorithm [Comaniciu & Meer 2002] is used (section 6.3.1). Secondly, color transfer functions are computed for every region (section 6.3.2). They are obtained using color information from the given region and its corresponding pixels in the model image. The correspondence between regions and model image pixels results from the coarse registration, considered known a priori. Thirdly, a color influence map [Maslennikova & Vezhnevets 2007] is computed for every region, in order to mix the each of the computed local color transfer functions (6.3.3). Finally, the color from the current image is corrected using the local transfer functions weighted by the color influence maps (6.3.4). The next lines will describe in detail each one of these stages in detail. Results of this particular technique are described in section 6.3.5.

The color correction algorithm presented in this section does not present a way of color correction the non overlapping regions of the target image. Because of this, in all source target image pairs presented in this section, the **S** image cover the entire range of the **T** image. In other words, the overlapping portion of the target image ($\mathbf{T}_p$) is exactly the same as the target image, i.e., $\mathbf{T}_p = \mathbf{T}$. In this particular section, no distinction is made between $\mathbf{T}$ and $\mathbf{T}_p$

The proposed algorithm is a local approach to the global color correction algorithm proposed by [Reinhard *et al.* 2001], where the new color $\mathrm{c}'(i,j)$ of each pixel in the corrected image is obtained using:

$$\mathrm{c}'(i,j) = \mu_{\mathbf{S}} + \frac{\sigma_{\mathbf{S}}}{\sigma_{\mathbf{T}}} \times (\mathrm{c}(i,j) - \mu_{\mathbf{T}}) \tag{6.1}$$

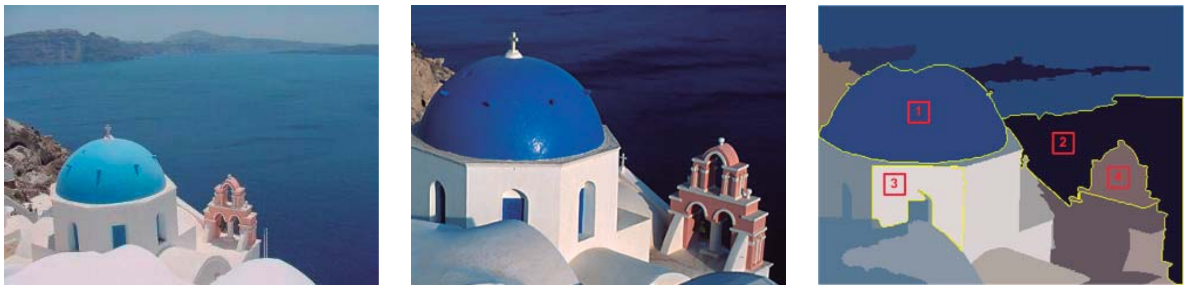
where $\mu_{\mathbf{T}}$ and $\mu_{\mathbf{S}}^k$ are the means colors of the target and source image respectively, $\sigma_{\mathbf{T}}$ and $\sigma_{\mathbf{S}}$ are the colors standard deviations of the target and source images, and $\mathrm{c}(i,j)$ is the color a pixel in the target image's *ith* line and *jth* column.

6.3.1 Image Segmentation

In order to segment the target image into a set of similar colored regions the Mean shift algorithm is used [Comaniciu & Meer 2002]. The Mean shift algorithm automatically segments the target image into a set of regions. Some care must be taken while tuning the Mean shift parameters: very large regions may include a large set of different colors and could result in very similar results for local color correction to its global counterpart. On the other hand, very small regions are sensitive to the lack of accuracy in image registration. In standard images the output of the Mean shift algorithm provides a good segmentation of the different colors present in the image. As will be shown, the color correction results using this technique are quite good, which means that the Mean shift algorithm does a good job at segmenting the several test images, in particular since that the same parameters are used in all the test images. This shows the robustness to the color segmentation stage of the proposed local color correction technique.

Figure 6.2 shows an example of a source and target image pairs. The result of the Mean shift algorithm applied to the target image is presented in Fig. 6.2 (*c*).

The approach presented in [Tai *et al.* 2005] was evaluated against nine other color correction approaches. It outperformed all other parametric approaches and is recommended as the first option to try for a general image and video stitching applications in practice [Xu & Mulligan 2010]. It consists of segmenting both the target and model images into several regions using an expectation maximization algorithm (local EM based color correction). Then, regions from the target image are matched to the model image. This is done by projecting each target image region onto the model image in order to assess the highest overlapping region in the model image. The match of regions from the target and the model image provides the statistical parameters required in (1). In addition to the segmented regions, the local EM based color correction algorithm also computes a weight mask for each region. These weight values indicate the probability that a given pixel belongs to that region. The final pixel color is obtained by adding up the contributions of each region's color transfer function weighted



(*a*) (*b*) (*c*)

Figure 6.2: An example of a source (**S**) and target (**T**) image pair: (*a*) source image; (*b*) target image; (*c*) the Mean shift color segmentation of the **T** image.

by its corresponding weight. Although it was the best performing algorithm in the evaluation done in [Xu & Mulligan 2010]. We believe the local EM based color correction algorithm can be improved in two major aspects. First, the expectation maximization segmentation stage is computationally demanding: authors state that this step takes four minutes to converge while segmenting a 512x512 image. Since the local EM based color correction must segment both the target and model image, the segmentation can take about 8 minutes. Second, the expectation maximization stage requires a parameter to define the desired number of regions: this is not interesting if unsupervised color correction is required. Figure 2 (right) shows the color corrected target image using this local EM based approach. Our proposed color correction approach addresses both aspects allowing unsupervised applications and lower computation times. Up to our knowledge, [Tai *et al.* 2005] is best algorithm published in the

One important difference from the Mean shift method to, for example the probabilistic segmentation proposed in [Tai *et al.* 2005], is that in the number of colors (or segmented regions) in the image is an input parameter in [Tai *et al.* 2005], while the Mean shift algorithm automatically estimates this value. Therefore, it seems feasible to assume that in Mean shift, changes in its input parameters do not result in a dramatic change of the color correction output, as should be the case in [Tai *et al.* 2005].

6.3.2 Local Color Transfer Functions

The output of the Mean shift color segmentation stage is a set of regions, each representing an area of the **T** image with similar colors. The objective now is to define a local color transfer function for each one of these regions. For this, statistics from each region (from the **T** image) are compared to a corresponding region in the **S** image. Let k be a given region from the target image. The coordinates of a pixel that belongs to that region is denoted as $\mathrm{x}_{\mathbf{T}}^{k}$, and the corresponding pixel $\mathrm{x}_{\mathbf{S}}^{k}$ in the **S** image is obtained by:

$$\mathrm{x}_{\mathbf{S}}^{k} = \mathbf{f}\big(\mathrm{x}_{\mathbf{T}}^{k}\big), \tag{6.2}$$

where **f** is the that obtains the coordinates of a pixel in the source image, given the coordinates of that pixel in the target image. It is obtained after the registration procedure. All the pixels inside region k are used for collecting local statistics, that is, to compute mean and standard deviation measures. For a given region k, let $\mu_{\mathbf{T}}^{k}$ and $\sigma_{\mathbf{T}}^{k}$ be the mean and standard deviations of color in that region. Using the transformation from target to source image, it is also possible to collect statistics over the source image's corresponding pixels for region k. These are referred to as $\mu_{\mathbf{S}}^{k}$ and $\sigma_{\mathbf{S}}^{k}$, respectively for the mean and standard deviation of color in the source image's corresponding region. From this statistics a local color correction function can be easily formulated by adapting eq. (6.1) to the local case:

$$\mathtt{c}'(i,j) = \mu_{\mathbf{S}}^{k} + \frac{\sigma_{\mathbf{S}}^{k}}{\sigma_{\mathbf{T}}^{k}} \times (\mathtt{c}(i,j) - \mu_{\mathbf{T}}^{k}). \tag{6.3}$$

A color transfer function, either it is global or local, can be expressed generically as a function f of several parameters:

$$\mathtt{c}'(i,j) = f\big(\mu_{\mathbf{S}}^{k}, \mu_{\mathbf{T}}^{k}, \frac{\sigma_{\mathbf{S}}^{k}}{\sigma_{\mathbf{T}}^{k}}, \mathtt{c}(i,j)\big). \tag{6.4}$$

A global color transfer function will have the first three parameters constant for the whole image, while the local approach maintains those parameters constant merely for every color segmented region. The initial hypothesis of this section was that due to different surface reflective properties and non uniform illumination, the global image color statistics would generate only a rough approximation of all the color transfer functions. Figure 6.3 plots the global and local function parameters for three channels ($l\alpha\beta$ color space) of the target and source images shown in Fig. 6.2 (*a*) and (*b*). Dashed lines represent the global color transfer function parameters for each channel. Dots represent the parameters of local color transfer functions computed from the regions depicted in Fig. 6.2 (*c*). Ellipsoids represent the average and standard deviations of the collection of local parameters. As expected, the parameters from the local color correction functions (represented by dots) are different for every region. It is also possible to observe that local color transfer functions sometime have differences to the global parameters of over 0.2 (20%, since the offset values are normalized from 0 to 1 on all image channels). Hence, it is possible to conclude that using a single global color transfer function as done in [Reinhard *et al.* 2001] is merely a rough approximation of the real color transfer functions. Another conclusion can be drawn by noting the larger size of the red ellipsoid when compared to the other two: luminance (l channel) has higher standard deviations than the α or β channels. This indicates that luminance is the channel with highest variability which reinforces the initial hypothesis that assuming a constant illuminant is not feasible for complex environments.

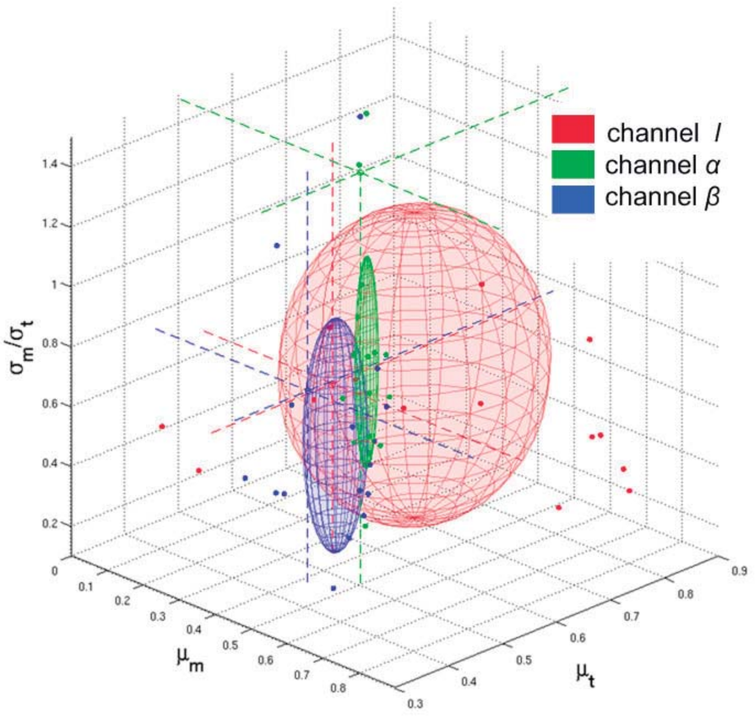


Figure 6.3: Distribution of the parameters of eq (6.3) for the regions segmented in the target source image pair displayed in Fig. 6.2.

6.3.3 Color Influence Maps

Although the importance of using multiple local color transfer functions has been established, the application of the color transfer functions to each pixel must be addressed in order to achieve natural color transition across regions. Hence, in this section we propose a methodology for combining the different local color transfer functions. It is based on the use of Color Influence Maps (CIM), which are computed for every region. The CIM [Maslennikova & Vezhnevets 2007] is a weight mask that measures the similarity between each color pixel and the mean color of that particular region. Since in the $l\alpha\beta$ color space the different channels are uncorrelated, the color similarity can be computed as an Euclidean distance:

$$\mathbf{CIM}^k(i,j) = g\big(\parallel \mathrm{c}(i,j) - \mu^k_{\mathbf{T}} \parallel\big). \tag{6.5}$$

where g is an arbitrary response function. In the current work the function proposed in [Maslennikova & Vezhnevets 2007] has been used:

$$g(x) = e^{-3\cdot x^2}. \tag{6.6}$$

Figure 5 shows the CIMs computed for the four regions highlighted in Fig. 6.2 (c). In these illus-

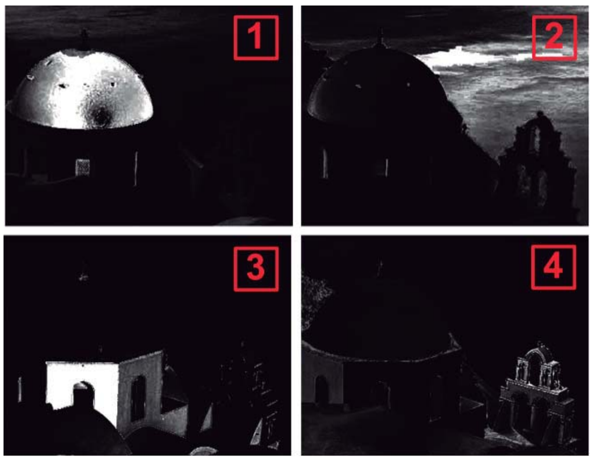


Figure 6.4: Color Influence Maps for the four regions highlighted in Fig. 6.2 (*c*).

trations pixels with a color similar to the mean color value of the considered region are represented with a high value (i.e., white), even though they may not belong to the same region.

6.3.4 Weighted Color Correction

Finally, in order to merge the different CIMs and get a single color correction value for every pixel we propose a weighted color correction scheme. The final color for a given pixel is obtained by adding the contributions of every color transfer function, weighted by the corresponding CIM:

$$\mathrm{c}'(i,j) = \frac{\displaystyle\sum_{k=1}^{N} \left(\left((\mu_{\mathbf{S}}^{k} + \frac{\sigma_{\mathbf{S}}^{k}}{\sigma_{\mathbf{T}}^{k}} \times (\mathrm{c}(i,j) - \mu_{\mathbf{T}}^{k})) \right) \cdot \mathbf{CIM}^{k}(i,j) \right)}{\displaystyle\sum_{k=1}^{N} \mathbf{CIM}^{k}(i,j)} \tag{6.7}$$

where N is the total number of segmented regions of the target image.

6.3.5 Results

The proposed approach has been applied to a set of images and compared with approaches from [Reinhard *et al.* 2001] and [Tai *et al.* 2005]. Figure 6.5 (*a*) shows the original target image and the corrected images using: (*b*) the global approach from [Reinhard *et al.* 2001], (*c*) the local approach from [Tai *et al.* 2005], and (*d*) the Mean shift based local approach. Figure 6.6 shows the mosaics obtained using the several target images: (*a*) original target image (*b*) corrected using the global approach from [Reinhard *et al.* 2001], (*c*) corrected using the local approach from [Tai *et al.* 2005], and (*d*) corrected using the Mean shift based local approach.

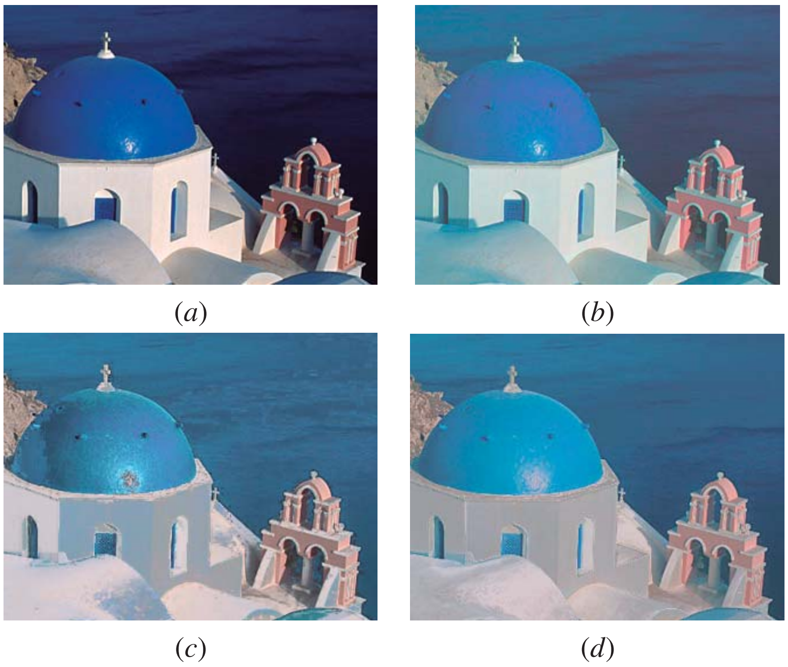


Figure 6.5: Color correction of the target image: (*a*) original image; (*b*) global approach from [Reinhard *et al.* 2001]; (*c*) local approach from [Tai *et al.* 2005]; (*d*) Mean shift based approach.

Figures 6.7 and 6.8 show the several qualitative results. Mosaics are shown without color correction, with global color correction, with local based color correction and with the Mean shift based approach proposed in the current section. It is visible that local approaches (the proposed approach and [Tai *et al.* 2005]) obtain better results in comparisson with the global approach from [Reinhard *et al.* 2001]. This is especially true in images where there is a great variety of colors. The Astro Clock (Fig. 6.7, *third column*) is a special case because the clock is in a different position in the source and target images. This example was included to try to assert how a small area that has an evident miss registration (different colors) is handled by the algorithms. In [Tai *et al.* 2005] approach, the blue region is not moved while in the proposed approach the blue is somewhat transferred to the left portion of the clock, where it should be. Regarding Ponte Vecchio (Fig. 6.8, *second column*), the proposed approach was able to correct the yellow houses on both sides of the bridge. In [Tai *et al.* 2005], both houses are painted white, which is a clear color correction failure. In Times Square (Fig. 6.8, *third column*) although the sky seems better using [Tai *et al.* 2005], the reflection of the yellow beer bottle on the left is supposed to disappear on the corrected image. In this detail, the proposed approach clearly does a better job than [Tai *et al.* 2005]. In Sagrada Familia (Fig. 6.8, *first column*), a single global color correction obtains the same results as both local techniques, because the target image has a low variety of colors. Finally, we can mention that the proposed approach requires on average about sixty seconds to compute the color correction, while [Tai *et al.* 2005]s approach requires, on average, over 10 minutes.

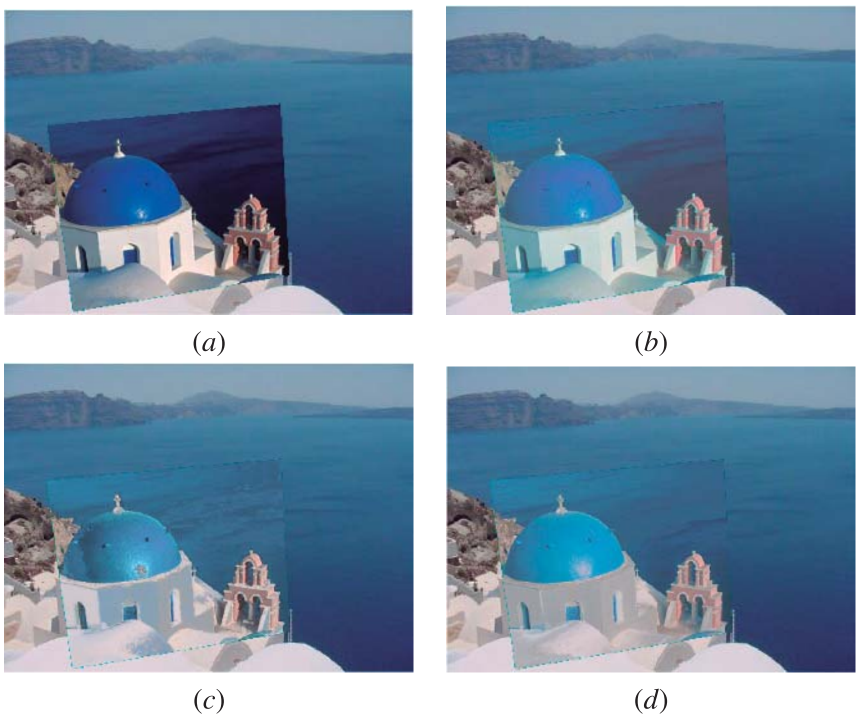


Figure 6.6: Mosaics obtained after color correction: (*a*) with original target image; (*b*) corrected using global approach from [Reinhard *et al.* 2001]; (*c*) corrected using local approach from [Tai *et al.* 2005]; (*d*) corrected using Mean shift based approach.

This section presents a new parametric method for local color correction of two coarsely registered images. The method uses the well known Mean shift algorithm and builds weight masks using CIM. Qualitative results show that the proposed approach produces visualy appealing mosaics, in most cases better that other approaches. Two reasons can be mentioned to explain this improvement. First the Mean shift algorithm performs better at segmenting the regions. Second, the local based method is more sensitive to registration inaccuracies, because it matches the blobs based on maximum region overlap, while in our approach we do not segment the model image but collect statistics directly from the projection of every region in the target image to the model image. Furthermore, the proposed approach is computed in about one tenth of the time when compared to that proposed in [Tai *et al.* 2005]. This section does not propose a method for infering color transfer functions when the target image is not entirely contained by the model image. In these cases, the overlapping region must provide information that must be extrapolated to the rest of the image.

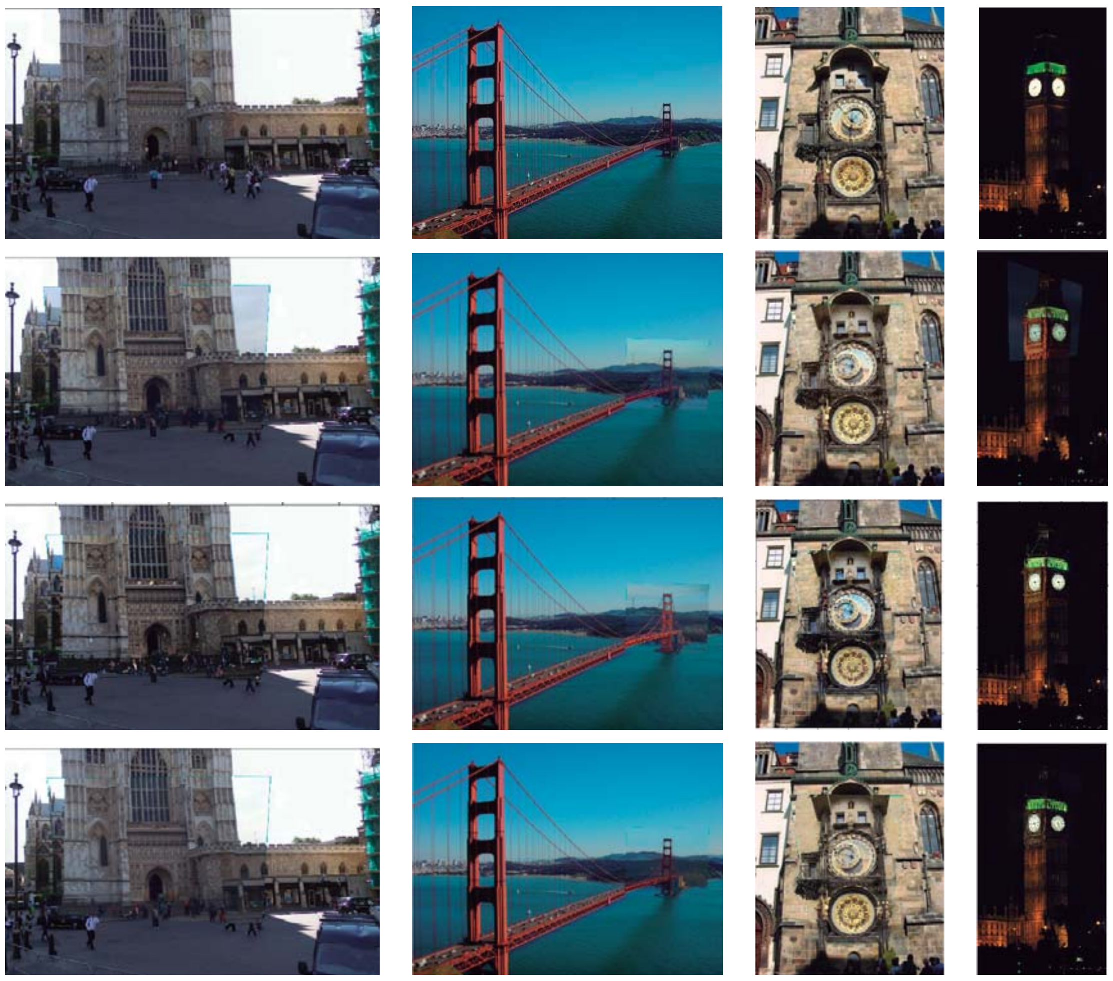


Figure 6.7: Color correction examples of Westminster Abbey, London (*first column*), Golden Gate bridge, San Francisco (*second column*), Astronomical clock, Prague (*third column*), and Big Ben, London (*fourth column*). The mosaic with the original source image (the ground truth) (*first row*), global color correction (*second row*), local approach (*third row*) and Mean shift based color correction (*fourth row*).

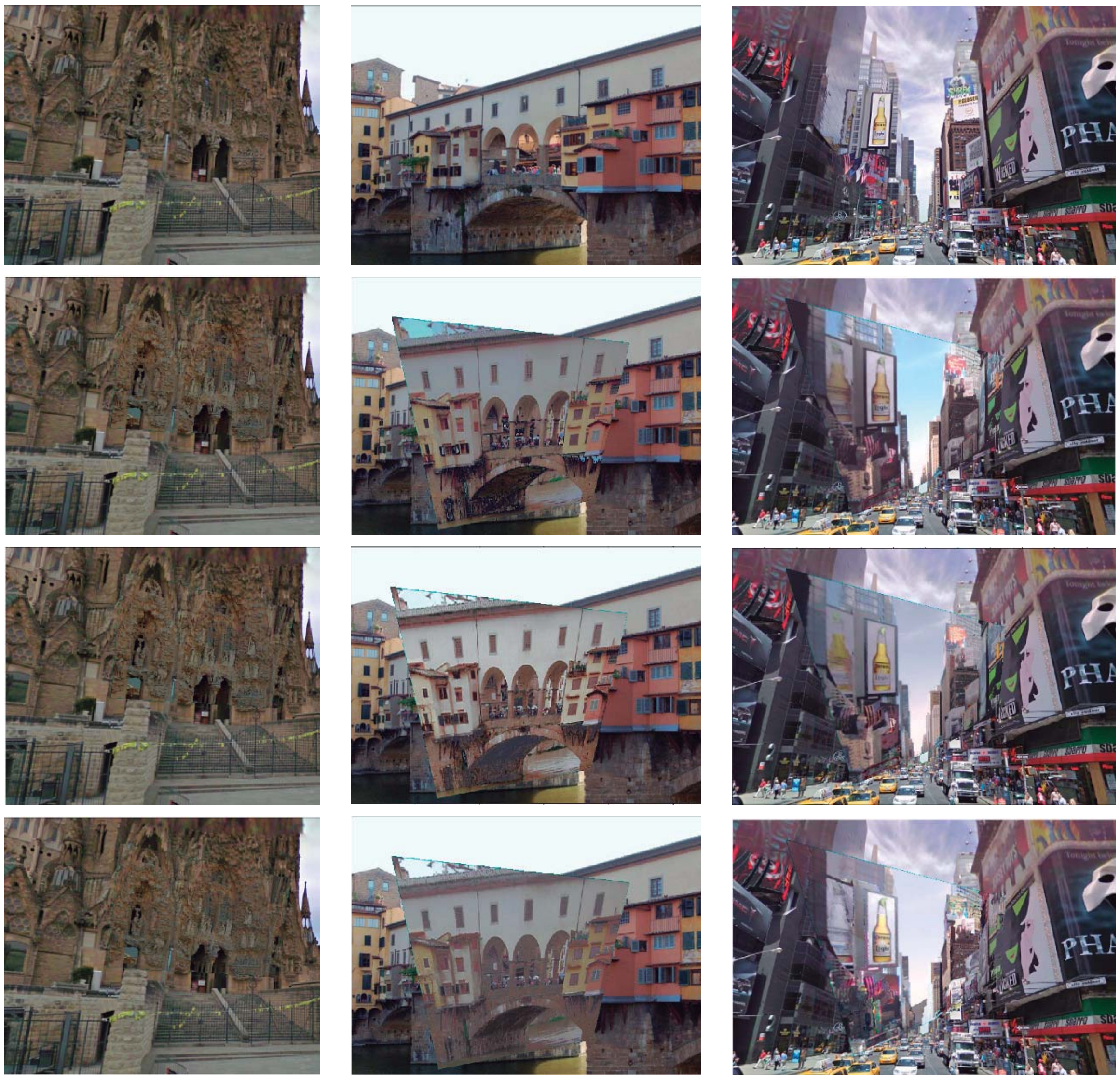


Figure 6.8: Color correction examples of Sagrada Familia, Barcelona (*first column*), Ponte Vechio, Florence (*second column*), and Times Square, New York (*third column*). The mosaic with the original source image (the ground truth) (*first row*), global color correction (*second row*), local approach (*third row*) and Mean shift based color correction (*fourth row*).

6.4 3D Gaussian Mixture Models

The section proposes an alternative color correction technique based on 3D Gaussian Mixture Models (3DGMM). To assess the effectiveness of the proposed algorithm, a very large set of images is used, and eight other state of the art algorithms for color correction are evaluated. The evaluation metric is taken from a recent performance evaluation of color correction algorithms [Xu & Mulligan 2010].

As discussed previously, the approach presented in [Reinhard *et al.* 2001] assumes a Gaussian distribution of color on both the source and target images. In other words, it uses a linear color transfer function. The Gaussian distribution based color transfer scheme, initially proposed in [Reinhard *et al.* 2001], has been defined in eq. (6.1). That equation can may be used to process single channel images (*gain compensation*) or color images (*color correction*). For color images eq. (6.1) is applied separately for the three color channels. However, in practical situations, the color distribution of the whole image is seldom a normal distribution. Global modelling of the color distribution fails in practice because it provides only a rough approximation of the color distribution. By computing, for several regions, a local color transfer function and assuming a separate Gaussian distribution for each region, the set of color transfer functions will provide a more consistent color correction output. This was proposed in [Tai *et al.* 2005], where the Reinhard's methodology was extended to the local scenario, namely through a color transfer scheme based on single channel probabilistic segmentation and region mapping using the EM algorithm. It was also shown in the results from the local Meanshift based approach presented in section 6.3.

With this algorithm, we propose to model the color distribution of the target and source images using 3DGMM for joint probabilistic segmentation of the three color channels. Then, several color transfer functions can be derived from the adaptation of equation (6.1), in a very similar way to that proposed in section 6.3. The methodology consists of three stages: probabilistic segmentation, computation of color transfer functions and finally, in the application of those color transfer functions to the target image. The next sections present these stages.

Like in the previous section, this section does not propose a method for infering color transfer functions when the target image is not entirely contained by the source image. Hence, also in this case, it is assumed that $\mathbf{T}_p = \mathbf{T}$, or that only $\mathbf{T}_p$ should be corrected (see Fig. 6.1).

Figure 6.9 shows the example of source and target images pairs that is used to demonstrate the algorithm.

6.4.1 Probabilistic Segmentation

The first stage consists of modelling the distribution of color in both images using 3DGMM. Gaussian Mixture Models (GMM) are among the most statistically mature methods for clustering. In this case they are used to model the color distribution of the pixels by segmenting the image. After segmentation, each cluster is a Gaussian component from the mixture model. The following details are given

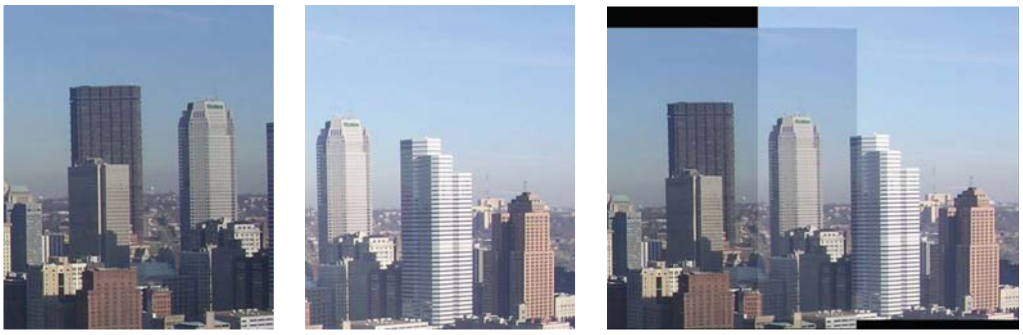


Figure 6.9: A mosaic of two images of a city landscape scene *(c)*. The colors of the target image *(a)* must be corrected using information from the source image *(b)* to avoid the artifacts present in the mosaic.

assuming an RGB image. However, the presented method is not restricted to this color space. In fact, alternative color spaces have been tested, although results have shown no particular advantages when compared to the traditional RGB. This is because the joint color correction of the three color channels avoids cross channel artifacts, as detailed below. The segmentation step is intended to cluster the image pixels into a set of colors. The assumption is that it is more feasible to represent color distribution as a Gaussian distribution in regions where only one color (or a more uniform set of colors) exist. The segmentation is done by defining the same number of Gaussian clusters NG both for the target and source images. This is done under the reasonable assumption that, since the images have the same view of the scene, they both should have the same number of colors in the overlapping regions.

Let NG be the number of Gaussian components that model the color distribution of an image. Let a given color be denoted as $\mathbf{c} = [r, g, b]$. A Gaussian component ω_k has mean $\boldsymbol{\mu}^k = [mean(r), mean(g), mean(b)]$ and standard deviation $\boldsymbol{\sigma}^k = [std(r), std(g), std(b)]$. The color distribution is modelled by the mixture of Gaussian components, so that the total density of pixels for a given color $D(\mathbf{c})$ is given by the weighted sum of the Gaussians:

$$D(\mathbf{c}) = \sum_{k=1}^{NG} mp^k \times P^k(\mathbf{c}), \tag{6.8}$$

where mp^k is the mixture proportion of Gaussian component ω_k, and $P^k(\mathbf{c})$ is the probability of Gaussian ω_k for color $\mathbf{c}$, which is given by the difference between two cumulative distribution functions of neighboring colors. The cumulative distribution function of Gaussian ω_k for a given color $\mathbf{c}$, denoted as $cdf(\omega_k, (c))$, is given by:

$$cdf(\omega_k, \mathbf{c}) = \frac{\sum_{i=1}^{3} \Big(\frac{1}{2}(1 + erf\Big(\frac{\mathbf{c}(i) - \boldsymbol{\mu}^k(i)}{\sqrt{2 \times (\boldsymbol{\sigma}^k(i))^2}} \Big) \Big)}{3}, \tag{6.9}$$

where (i) is the index of the color channel (i.e., $i \in \{r, g, b\}$), and *erf* is the error function, also known as probability integral, given as:

$$\mathit{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^\infty e^{-x^2} dx. \tag{6.10}$$

Hence, the probability of the Gaussian component k is computed as:

$$P^k(\mathbf{c}) = \mathit{cdf}(\omega_k, \mathbf{c} + \boldsymbol{\lambda}) - \mathit{cdf}(\omega_k, \mathbf{c} - \boldsymbol{\lambda}), \tag{6.11}$$

where $\boldsymbol{\lambda} = [cr/2, cr/2, cr/2]$ is given by half the image's color resolution (*cr*).

In [Tai *et al.* 2005], each channel of the image undergoes a segmentation procedure similar to this one. However, in that approach, since the probabilistic segmentation is performed independently for each channel, the probabilities of Gaussian components may be different from channel to channel, i. e., $P^k(\mathbf{c}(1)) \neq (P^k(\mathbf{c}(2)) \neq (P^k(\mathbf{c}(3))$. The methodology proposed in the current chapter uses a single probability function for all the three color channels, i.e., $P^k(\mathbf{c})$, as defined in eq. (6.11).

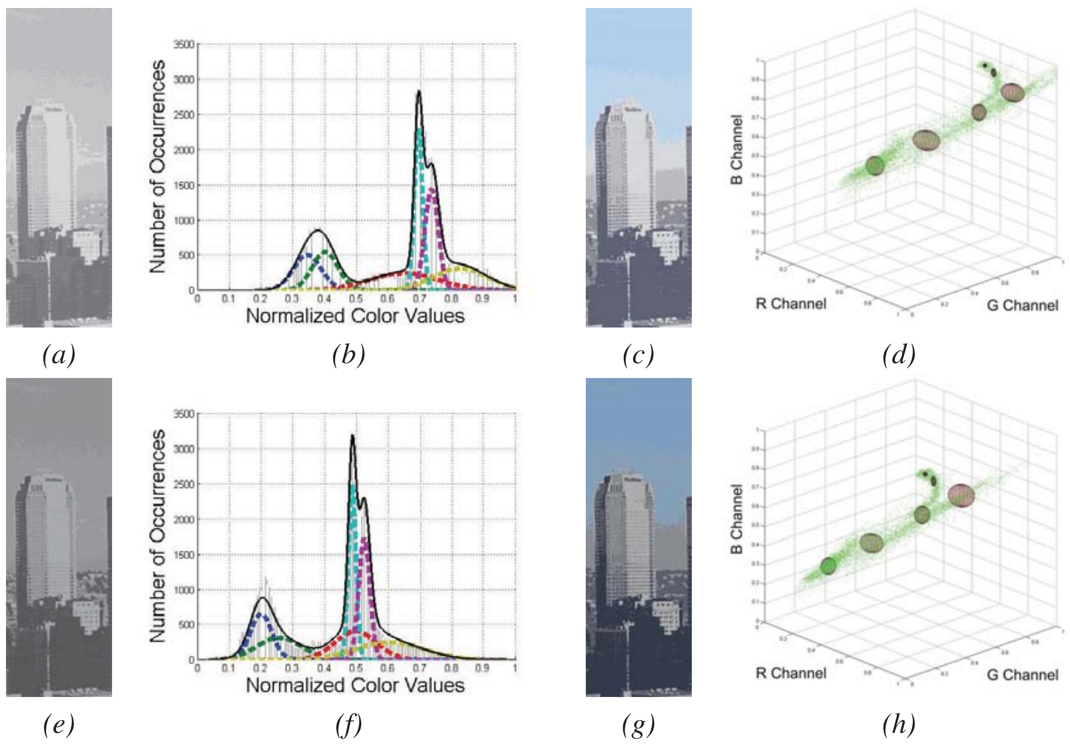


Figure 6.10: Single channel color segmentation of the Source *(a)* and Target *(e)* images. Single channel histograms, Gaussian components (*dashed*) and total Gaussian mass (*solid black*) of the source *(b)* and target *(f)* images. 3D color segmentation of the source *(c)* and target *(g)* images. Color distribution of all pixels (*green dots*) and 3D Gaussian components (*red ellipsoids*) of the source *(d)* and target *(h)* images.

This reduces the occurrence of cross channel artifacts that arise from color correction as is the case in [Tai *et al.* 2005]. As will be shown in section 6.4.4, by performing a 3DGMM of all three image channels in a joint segmentation step we are able to improve the color correction performance and reduce processing time. Figure 6.10 shows the GMM color segmentation both for the 1D and the 3D cases.

6.4.2 Color Transfer Functions

The current section proposes to perform a probabilistic segmentation of both the source and target images using 3DGMM. The result of the segmentation step is that both the target and the source images are segmented into NG clusters, each representing a Gaussian component for the inferred mixture model. It is then necessary to associate each Gaussian component from the target image to another of the source image. This association is referred to as matching of Gaussian components. When spatial information exists, which is the case since images are registered, the matching is performed based on the maximum spatial correlation of pixel probabilities. To compute the spatial correlation, let *color* be the function that retrieves the color of a pixel. The probability P that each pixel $\mathbf{x}$ has of belonging to Gaussian component ω_k is calculated using the color retrieval function. For simplification purposes, $P^k(color(\mathbf{x}))$ will be from now on denoted as $P^k(\mathbf{x})$. The matching of Gaussian components is computed as follows: let $m(k)$ be the matching function that outputs the index of the source image Gaussian component for target image Gaussian component k:

$$m(k) = argmax(r(k, j)), \forall j \in \{1, 2, 3, ..., NG\}, \tag{6.12}$$

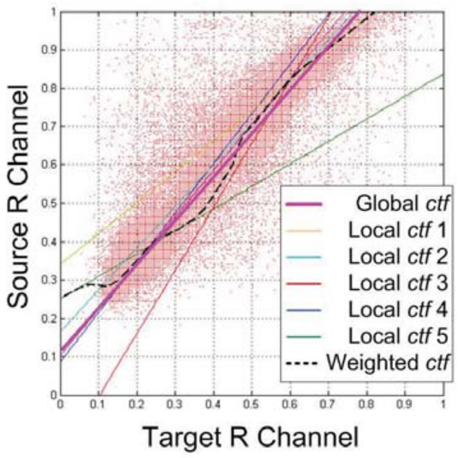


Figure 6.11: The mapping of the red color channel for the target / source image pair of Fig. 6.9. The global color transfer function is obtained using [Reinhard *et al.* 2001]. Several local color transfer functions are obtained using [Tai *et al.* 2005], which combined result in a non linear weighted color transfer function.

where r represents the spatial correlation between the probabilities of target image Gaussians P_t with source image Gaussians P_s, given by:

$$r(k,j) = \frac{\sum_{\mathbf{x}=[1,1]}^{[W,H]} (P_t^k(\mathbf{x}) - \bar{P}_t^k) \times (P_s^j(\mathbf{x}) - \bar{P}_s^j)}{\sqrt{\sum_{\mathbf{x}=[1,1]}^{[W,H]} \left(P_t^k(\mathbf{x}) - \bar{P}_t^k\right)^2 \sum_{\mathbf{x}=[1,1]}^{[W,H]} \left(P_s^j(\mathbf{x}) - \bar{P}_s^j\right)^2}}, \tag{6.13}$$

where $\bar{P}^k$ represents the probability of the average color of Gaussian k, i.e., $\bar{P}^k = P(\boldsymbol{\mu}^k)$, and W, H are the image's width and height respectively.

The color correction procedure will make use of NG color transfer functions, each one corresponding to a match between a region in the target with a region in the source image. The color transfer functions (ctf) are obtained by adapting (6.1) to the 3D case:

$$ctf^{k,m(k)}(i) = \boldsymbol{\mu}_s^{m(k)}(i) + \frac{\boldsymbol{\sigma}_s^{m(k)}(i)}{\boldsymbol{\sigma}_t^k(i)} \times (\mathbf{c}(i) - \boldsymbol{\mu}_t^k(i)), \tag{6.14}$$

Figure 6.11 shows a comparisson of several local color transfer functions, obtained for each Gaussian component match, with a single global color transfer function computed using a global approach like the one presented in [Reinhard *et al.* 2001]. A single global *ctf* (as proposed in [Reinhard *et al.* 2001]) is a poor approximation for the real color transfer. The computation of several local color transfer functions and the usage of a non-linear weighted *ctf* (as proposed in [Tai *et al.* 2005]) increases the effectiveness of color correction. However, as can be observed in Fig. 6.11, pixels with color 0.4 in the target image are mapped to the source image to values that range from 0.3 to 1. In other words, there is a high level of redundancy in the color pallet mapping. Inferring a function that maps the color on the target to the source will always be insufficient due to its non injective nature. By collectively using the three channels in a joint segmentation and color correction step as proposed in the current chapter the redundancy present in single channel mapping is reduced, which improves the color correction effectiveness.

6.4.3 Color Correction

Once the source and target images have been segmented into NG regions and the corresponding color transfer functions for each match are computed, the objective at this last stage is to correct the color of every single pixel. Because of the probabilistic nature of the proposed color segmentation, pixels may have non zero probability of belonging to more than one region. Hence, the proposed color transfer approach is defined as a weighted combination of all the computed color transfer functions:

$$\mathbf{c}'(i) = \sum_{k=1}^{NG} mp^k \cdot P_t^k(\mathbf{c}) \cdot ctf^{k,m(k)}(i), \tag{6.15}$$

where the bold symbol $\mathbf{c}'$ denotes the three channel color of the color corrected image and (i) is the index of the color channel. The expression formulated in eq. (6.15) is in all similar to the one proposed in section 6.3. Both are a weighted average of the locally computed color transfer functions. The difference here is that while before the CIM was used to compute the weights, in this case it is the mixture proportion of the Gaussian component that defines the weight.

6.4.4 Results

In order to validate the proposed technique, two very different data sets are used. The first is a standard data set with over 70 images of several scenes and places. The second data set was taken from the cameras onboard the *AtlasCar*. In the following lines, we will present results using both data sets.

Standard Data Set

In order to test the proposed algorithm, the two data sets of a recent performance evaluation [Xu & Mulligan 2010] were used. They consist of a synthesized data set of 40 image pairs and a real image data set of 30 image pairs. The registration of target / source was not provided by the authors of [Xu & Mulligan 2010]. Because of this, a manual process of hand labelling pixels in both images was done to obtain the registration. In order to compare the results of the proposed approach with the state of the art, eight of the nine algorithms used in [Xu & Mulligan 2010] were applied to the same data sets. Regarding the missing algorithm [Pitie *et al.* 2005], it was not possible to find a public implementation to guarantee a fair comparison. However, the algorithm presented in [Pitie *et al.* 2005] did not reach the best performance in none of the tests presented in [Xu & Mulligan 2010]. Table 6.1 lists all the algorithms tested in this comparison. Note that the cardinal references for each algorithm as presented in Table 6.1 are only valid within this subsection. Different comparison are made in sections 6.4.4 and 6.5, using an additional method. In that case, new cardinal references are attributed to each algorithm.

The evaluation parameters, i.e., *color similarity* (CS) and *structural similarity* (SS) were taken from [Xu & Mulligan 2010]. For a better comparison of the tested methodologies, the average processing time taken to correct one image is also presented. Although results of both CS, SS and time are presented, the CS score is the most important parameter, since it evaluates how well a color correction algorithm is able to balance the colors in the target image so that they match the ones in the source image (see details in [Xu & Mulligan 2010]).

Table 6.2 shows the average CS and SS scores of the seven methods used for comparison, as well as of the approach proposed in the current paper. Analyzing Table 6.2, two different classes of methods may be identified: fast algorithms #1, #2, #3, #4, #5 and #6, which have processing times under one second but have limited CS scores; and highly effective algorithms #7 #8 #9 and #10 (the proposed approach), which require more time to get the highest CS scores. Note that this CS score corresponds to a logarithmic scale (see details in [Xu & Mulligan 2010]). Results show that the proposed approach has the highest average CS scores for both the synthetic and real data sets. Furthermore, the proposed approach presents some of the lowest values of standard deviation of CS, which accounts for a smaller variation of CS scores throughout the images in the data sets. This is also a very important remark since it accounts for the reliability and robustness of our algorithm. The proposed approach is also much faster than two of the highest scoring methods (#7 and #8).

The results presented in Table 6.2 are different (in absolute values) from the ones presented in [Xu & Mulligan 2010] because there is a different registration. Nonetheless, the results presented in the current chapter are consistent with those in [Xu & Mulligan 2010], where the best average CS scores were also from algorithms #7 #8 and #9.

Table 6.5 gives some qualitative results. Here it is also possible to verify that the proposed approach shows the greatest similarity with the reference source image when compared to the other eight algorithms.

This chapter proposes to use a single step multi-dimensional probabilistic segmentation of the three color channels of an image in order to perform color correction. The color distribution of the images is modelled as a 3D mixture of Gaussian components. The proposed approach is compared with several state of the art algorithms used from color correction. In addition, a large set of images, previously used in [Xu & Mulligan 2010], are employed to assess the effectiveness of the color correction algorithms. Furthermore, the evaluation metric is taken from a recent performance evaluation in color correction. The joint segmentation of the three channel color reduces processing time from similar single channel methods and avoids cross channel artifacts that may appear due to an independent color correction of each channel separately. The proposed approach obtained the highest average CS scores and is amongst the lowest in CS standard deviation, which definitely makes it a technique

Table 6.1: A list of the algorithms used for comparing the proposed approach.

Name of the Approach	Reference	Alg.
Baseline (Target Image)	—	#1
Gain Compensation	Brown 07 [Brown & Lowe 2007]	#2
Global Color Transfer in RGB	Xiao 06 [Xiao & Ma 2006]	#3
Global Color Transfer	Reinhard 01 [Reinhard *et al.* 2001]	#4
Cumulative Histogram Mapping	Fecker 08 [Fecker *et al.* 2008]	#5
Principal Components Analysis	Zhang 04 [Zhang & Georganas 2004]	#6
Local Color Transfer	Tai 05 [Tai *et al.* 2005]	#7
Tensor Voting	Jia 05 [Jia & Tang 2005]	#8
Brightness Transfer Function	Kim 08 [Kim & Pollefeys 2008]	#9
3D Gaussian Mixture Models	section 6.4	#10

Table 6.2: Average and standard deviation of CS and SS scores for the two data sets. Average processing time per image is also provided. The proposed approach (# 10) obtains the highest average CS score.

Alg.	Synthetic					Real				
	CS		SS		Time	CS		SS		Time
#	μ	σ	μ	σ	(sec)	μ	σ	μ	σ	(sec)
#1	18.66	3.92	1.00	0.00	—	16.44	3.40	1.00	0.00	—
#2	21.35	2.56	0.97	0.02	0.53	19.89	3.27	0.96	0.03	0.51
#3	19.03	5.97	0.71	0.27	0.72	19.43	5.80	0.65	0.17	1.35
#4	19.86	4.92	0.79	0.15	0.43	20.25	6.78	0.67	0.17	0.94
#5	26.79	6.04	0.90	0.06	0.93	21.93	4.07	0.89	0.10	1.64
#6	26.29	6.65	0.91	0.07	0.70	20.41	3.56	0.87	0.13	0.93
#7	27.45	7.77	0.90	0.08	54.47	21.23	4.24	0.85	0.13	148.00
#8	27.71	7.33	0.91	0.06	233.50	21.82	4.05	0.85	0.17	235.70
#9	27.82	7.58	0.90	0.06	5.93	21.85	4.03	0.88	0.10	6.92
#10	**28.18**	**4.11**	**0.74**	**0.11**	**23.19**	**22.41**	**3.40**	**0.89**	**0.11**	**54.43**

to take into account for devising color correction algorithms. Results show that 3DGMM may be successfully applied to color correction with effectiveness that overcomes the current state of the art.

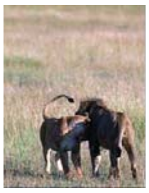
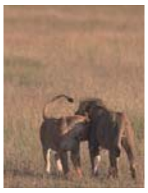
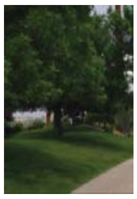
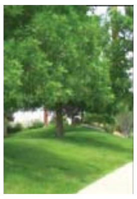
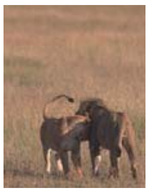
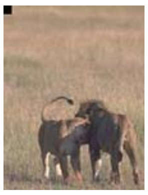
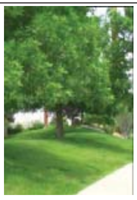
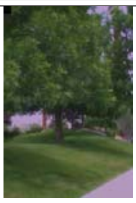
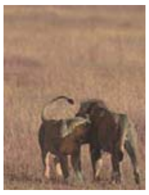
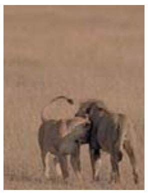
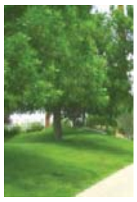
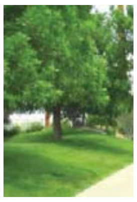
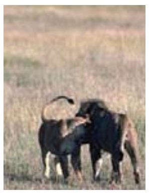
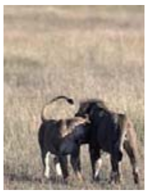
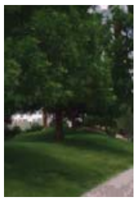
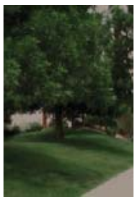
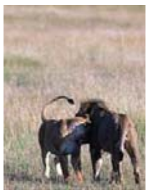
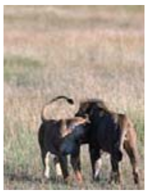
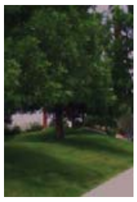
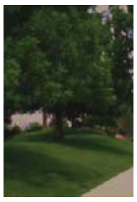
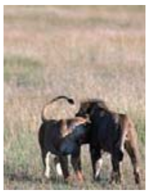
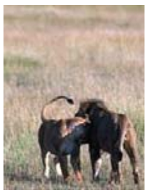
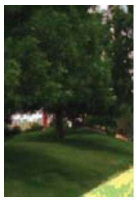
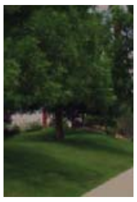
AtlasCar **Data Set**

In recent years, vision based sensors have been increasingly applied to autonomous vehicles and advanced driver assistance systems. They have key advantages over some other sensors, such as: being passive, obtaining vast amounts of information and being a low cost technology. Actually, the low cost and the impossibility to get a good view of the entire road around the vehicle using a single sensor, leads to the use of two or more of these devices. Many examples can be given, from the DARPA Challenge competitors (*www.darpa.mil/grandchallenge*).

The usage of more than one camera onboard of a moving platform poses new problems, which have not yet received enough attention from the research community. In fact, no assumptions can be made on key parameters, for example, scene illumination and contrast, which are directly measured by the vision sensor. If images from the cameras are to be merged into a mosaic or analyzed by some feature extractor algorithm, colors in both images should appear similar. This problem, called the photometric correspondence between images, has been addressed both by the computer graphics and the computer vision research communities.

Although many different algorithms have been proposed to perform color correction, no study has been published regarding the application of these algorithms to the vision sensors onboard autonomous vehicles. There are some characteristics in this particular application that make a specialized study necessary: real time requirements, since the processing time is coupled with the maximum vehicle speed; and the need to handle a great range of illumination conditions due to sun glare, tunnels, night or fog.

Table 6.3: The output of the comparative methods and the proposed approach (alg. #10) for three images in the data set. The image pairs are shown on the top of the table. Below each image the CS and SS scores are displayed. For image *Synthetic* #34, algorithm #9 achieves the highest CS score. In the case of *Synthetic* #23 and *Real* #6, the proposed approach outperforms all other methods.

Synthetic #34		**Real #6**	
			
Source	Target	Source	Target
Alg.; CS; SS	Alg.; CS; SS	Alg.; CS; SS	Alg.; CS; SS
			
#1 ; 17.3 ; 1.00	#2 ; 19.8 ; .98	#1 ; 12.2 ; 1.0	#2 ; 18.3 ; .88
			
#3 ; 16.9 ; 0.93	#4 ; 16.2 ; 0.94	#3 ; 13.1 ; 0.40	#4 ; 12.9 ; .38
			
#5 ; 32.8 ; 0.84	#6 ; 33.0 ; 0.86	#5 ; 21.9 ; 0.58	#6 ; 21.9 ; .63
			
#7 ; 35.4 ; 0.83	#8 ; 35.2 ; 0.83	#7 ; 22.2 ; 0.65	#8 ; 22.2 ; 0.59
			
#9 ; **36.9** ; 0.83	#10 ; 36.1 ; 0.82	#9 ; 20.5 ; 0.57	#10 ; **22.7** ; 0.58

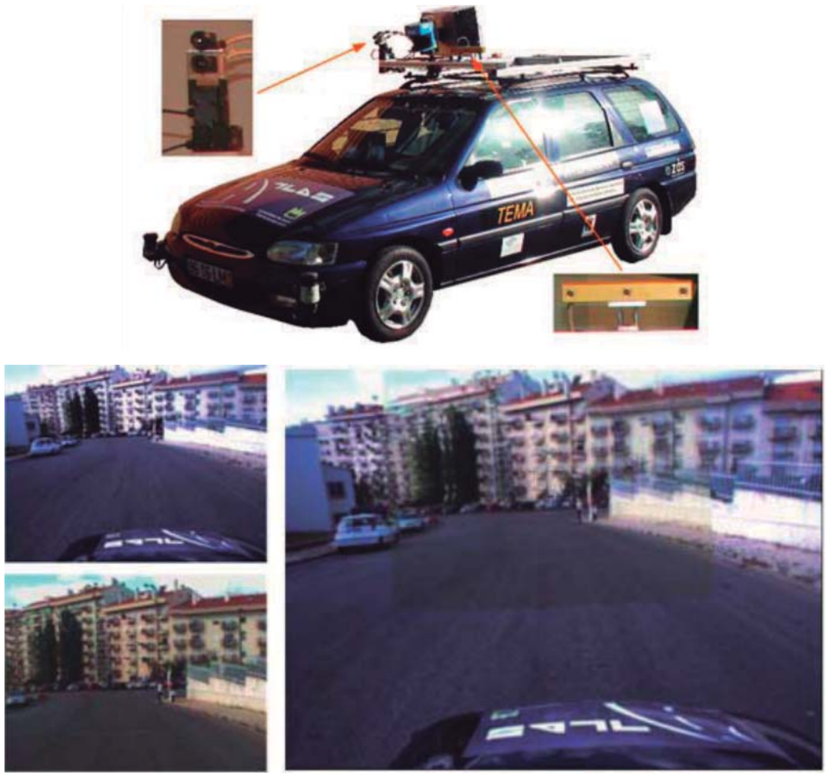


Figure 6.12: The *AtlasCar* robotic platform with several onboard cameras (*top*). Onboard images taken with two cameras from a typical road scene. Source image (*bottom-left-top*), Target image(*bottom-left-bottom*); the mosaic of both which shows a clear difference in colors (*bottom-right*); .

Table 6.4: Average and standard deviation of CS and SS scores for the set of selected images. It also shows the average processing time per image. The methods are sorted by average CS score. Note that the fastest algorithm [Brown & Lowe 2007] gets the worst CS score. The proposed algorithm obtains the highest average CS score, the second best on average SS score and is the fastest processing of the top three in CS scores.

			CS		SS		Time
Name of the Approach	Reference	Alg.	μ	σ	μ	σ	(sec)
Baseline (Non corrected Image)	none	#1	15.46	4.68	1.00	0.00	—
Gain Compensation	[Brown & Lowe 2007]	#2	15.51	2.47	0.98	0.02	0.21
Global Color Transfer in RGB	[Xiao & Ma 2006]	#3	17.42	5.72	0.68	0.13	0.34
Global Color Transfer	[Reinhard *et al.* 2001]	#4	17.56	6.23	0.70	0.13	0.22
Cumulative Histogram Mapping	[Fecker *et al.* 2008]	#5	20.98	5.14	0.73	0.23	0.53
Principal Components Analysis	[Zhang & Georganas 2004]	#6	22.53	4.71	0.77	0.23	0.40
Local Color Transfer	[Tai *et al.* 2005]	#7	23.28	3.92	0.75	0.20	4.48
Brightness Transfer Function	[Kim & Pollefeys 2008]	#8	24.15	4.94	0.75	0.20	5.09
3D Gaussian Mixture Models	section 6.4	#9	**24.30**	**4.06**	**0.77**	**0.21**	**4.10**

In order to test the proposed algorithm, the *AtlasCar* robotic platform [Santos *et al.* 2010] was used to acquire several images of typical road scenarios. The vehicle is used for research on autonomous driving and advanced driver assistance systems and it is equipped with several cameras (Fig. 6.12(*top*)). Although many video streams from the *AtlasCar* were tested, the results here presented refer to a set of 30 image pairs from two onboard cameras. In the case of this data set, the target images are entirely overlapped by the source images. Images from the stereo camera where selected to be the source images, while images from a teleobjective camera where the target images. All image pairs where hand registered using Matlab. Figure 6.12 shows one of the image pairs from the data set (*bottom-left*) and a mosaic composed of the two images (*bottom-right*).

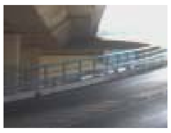
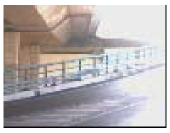
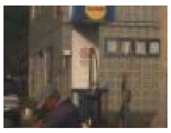
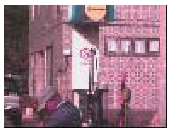
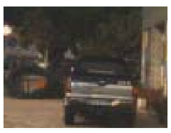
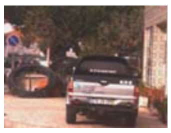
In order to compare the results of the proposed approach with the state of the art, seven of the nine algorithms used in a recent performance evaluation on color correction for image stitching applications [Xu & Mulligan 2010] were used in the same data set. The other two algorithms were not used since [Jia & Tang 2005] takes on average 140 seconds to correct a single pair of images and, regarding [Pitie *et al.* 2005], it was not possible to find a public implementation to guarantee a fair comparison.

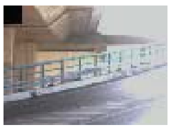
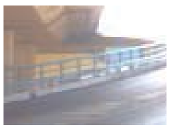
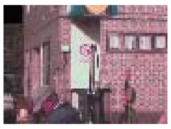
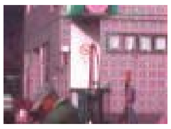
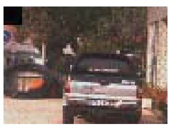
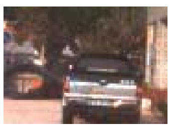
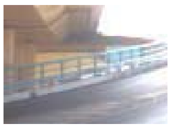
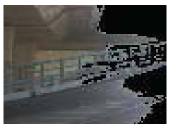
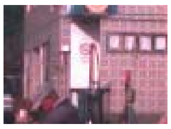
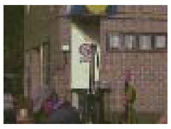
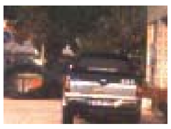
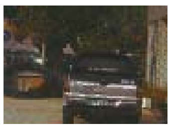
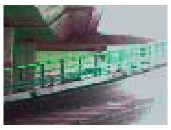
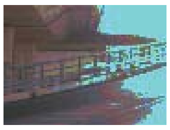
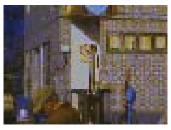
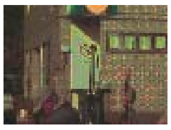
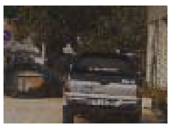
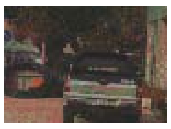
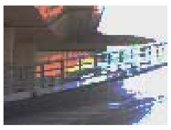
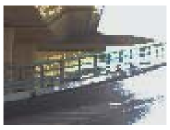
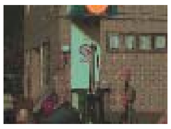
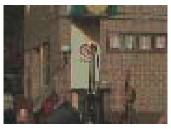
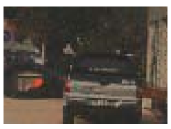
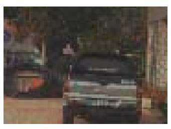
The evaluation parameters, i.e., *color similarity* (CS) and *structural similarity* (SS) were taken from [Xu & Mulligan 2010] (see reference for their meaning). For better comparison of the proposed methodologies, the average processing time taken to correct one image is also presented.

Table 6.4 shows the average CS and SS scores of the seven methods used for comparison, as well as of the approach proposed in the current paper. Analyzing Table 6.4, two different classes of methods may be identified: fast methods [Brown & Lowe 2007, Xiao & Ma 2006, Reinhard *et al.* 2001, Fecker *et al.* 2008, Zhang & Georganas 2004], which have processing times under one second but have limited CS scores; and highly effective methods [Tai *et al.* 2005, Kim & Pollefeys 2008] (and the proposed approach), which require about 10 times more time to get the highest CS scores. Note that this CS score corresponds to a logarithmic scale (see details in [Xu & Mulligan 2010]). Results show that the proposed approach has the highest average CS scores and is the second best in average SS score. Also, considering the second class of tested methods, the proposed approach is the fastest one. The values presented in Table 6.4 are consistent with the evaluation performed by [Xu & Mulligan 2010], where the best average CS scores were also from [Tai *et al.* 2005, Kim & Pollefeys 2008].

Table 6.5 gives some qualitative results. Here it is also possible to verify that the proposed approach shows the greatest similarity with the reference source image.

Table 6.5: The output of the comparative methods and the proposed approach (**GMM**) for three of the images in the data set. The image pairs are show on the top of the table. Bellow each image the algorithm reference, CS and SS scores are displayed. In the first two images the proposed approach obtains the best CS score, while in the third image it scores close to the best.

Image #1		Image #2		Image #3	
					
Source	Target CS=11.5	Source	Target CS=13.6	Source	Target CS=14.6

Alg.; *CS*; *SS*	Alg.; *CS*; *SS*	Alg.; *CS*; *SS*	Alg.; *CS*; *SS*	Alg.; *CS*; *SS*	Alg.; *CS*; *SS*
					
#2 ; 14.0 ; 0.9	#3 ; 13.0 ; 0.9	#2 ; 15.5 ; 0.9	#3 ; 14.3 ; 0.7	#2 ; 15.8 ; 0.9	#3 ; 15.2 ; 0.8
					
#4 ; 13.1 ; 0.6	#5 ; 8.5 ; 0.4	#4 ; 14.3 ; 0.7	#5 ; 24.9 ; 0.8	#4 ; 15.1 ; 0.8	#5 ; 26.0 ; 0.8
					
#6 ; 16.5 ; 0.7	#7 ; 17.0 ; 0.5	#6 ; 21.3 ; 0.7	#7 ; 23.7 ; 0.7	#6 ; 25.4 ; 0.8	#7 ; 26.7 ; 0.7
					
#8 ; 16.2 ; 0.7	#9 ; 17.1 ; 0.7	#8 ; 25.4 ; 0.7	#9 ; 25.9 ; 0.7	#8 ; 27.6 ; 0.7	#9 ; 27.2 ; 0.7

6.5 Probabilistic Color Correction using sets of Truncated Gaussians

This section proposes the third color correction approach. It is a probabilistic approach for modeling local color palette mapping functions. First, the overlapping portion of the target image undergoes a Mean shift based color segmentation process. Each of the color segmented regions is then mapped to a local joint image histogram. Then, a set of Gaussian curves is fitted to the local joint image histogram using a Maximum Likelihood Estimation process and truncated Gaussian curves as model. These Gaussians are then used to compute local color palette mapping functions. The next step is to expand the application of these functions from the overlapping area of target image to the entire target image. Finally, the entire color corrected image is produced by applying the color palette mapping functions to the target image. The following subsections will explain in detail the stages of the algorithm.

Although there are several methods proposed to deal with color correction, most involve strong assumptions, which are in general, difficult to fulfill in complex environments. Because of this we believe there is room for improving the effectiveness of color correction algorithms. Furthermore, the size of the data sets used for evaluating their performance is relatively small. In fact, most proposed approaches show results just for the few images depicted in the chapter and compare them to the baseline approach from [Reinhard *et al.* 2001]. This section proposes a novel color correction algorithm that presents several technical novelties when compared to the state of the art: (*i*) the usage of truncated Gaussians to model more accurately the color distribution; (*ii*) the cross modeling of mapping probabilities followed by the fusion of the set of Gaussians, enabling a more consistent inference of the color mapping functions inclusively for mappings of colors that are not observed; and (*iii*) a methodology to perform the expansion of the color palette mapping functions to the non overlapping regions of the images. To the best of our knowledge, this chapter also presents one of the most complete evaluations of color correction algorithms for image mosaicking published in literature. A very extensive comparison, that includes nine other color correction algorithms, two data-sets with over sixty image pairs and two distinct evaluation metrics, is presented. As will be shown, the proposed color correction algorithm achieves very good results when compared to state of the art algorithms.

6.5.1 Mean Shift Color Segmentation

The first step of the algorithm here presented is to perform a color segmentation of the overlapping portion of the target image $\mathbf{T}_p$ into several regions. In this case, only the portion of the target image that contains joint information with the source image is accounted for. Hence, image $\mathbf{T}_p$ is split into several regions, which will be treated independently by the color correction algorithm. This methodology is usually referred to as local color correction. Several previous works on color correction have shown the advantages of using a local approach when compared to a global one. Local methods presented in [Tai *et al.* 2005], [Xiang *et al.* 2009], [Oliveira *et al.* 2011] and in section 6.3, have shown better performance compared with the global approach presented in [Reinhard *et al.* 2001].

Also, section 6.3 has already shown several advantages of Mean shift with respect to other local color correction approaches.

The relevance of Mean shift [Comaniciu & Meer 2002] to perform color segmentation as a pre-processing step for color correction has been established [Oliveira *et al.* 2011]. Compared to other algorithms such as [Tai *et al.* 2005], where an expectation maximization (EM) probabilistic based color segmentation is proposed, Mean shift is better in two major aspects: first, the EM segmentation stage is computationally demanding: authors state that this step takes four minutes to converge while segmenting a 512x512 image. Since the local EM based color correction must segment both the source and the target image, the segmentation can take about eight minutes [Tai *et al.* 2005]; second, the expectation maximization stage requires a parameter to define the desired number of regions: this is not interesting if unsupervised color correction is required. Meanshift addresses both drawbacks allowing unsupervised applications and lower computation times.

Note that although the algorithm surely benefits from the usage of Mean shift as a preprocessing step, it may still run if no color segmentation is performed. In section 6.5.8 a study is presented that shows how Mean shift influences the performance of the proposed color correction methodology.

The following portions of the algorithm are applied independently for each of the segmented regions: region i in image $\mathbf{T}_p$ will be referred to as $\mathbf{T}_p^i$, and the projection of this region to the source image as $\mathbf{S}_p^i$.

6.5.2 Local Joint Image Histogram

The current chapter proposes to look into the problem of color correction as a mutual information problem. In probability theory the mutual information of two random variables is a quantity that measures the mutual dependence of the two variables. Other authors have used similar approaches by analyzing the joint image histogram [Jia & Tang 2005, Kim & Pollefeys 2008]. Let $\mathtt{X}$ and $\mathtt{Y}$ be the discrete random variables that correspond to the color values of the $\mathbf{T}_p^i$ and $\mathbf{S}_p^i$ regions respectively. The random variates of these random variable are denoted as x and y, respectively. Similarly to the standard one dimensional histogram, the joint image histogram is a two dimensional histogram or matrix, where each cell index (x, y) accounts for the number of times that the value x of color $\mathtt{X}$ in the $\mathbf{T}_p^i$ region is mapped to the value y of color $\mathtt{Y}$ in the $\mathbf{S}_p^i$ region. In this context, $\mathtt{X}$ and $\mathtt{Y}$ represent all the possible values of color in the target and source images, respectively. Hence, they are both defined in the discrete interval $\{0, 1, 2, ..., 2^n - 1\}$, with n being the bit depth of the images. The joint image histogram is built using the following eq.:

$$\mathbf{I}^i(x, y) = \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} \mathbf{q}(\mathbf{T}_p^i(u, v), x) \cdot \mathbf{q}(\mathbf{S}_p^i(u, v), y), \tag{6.16}$$

where u, v are the pixel coordinates, W and H the width and height of the images (note that only pixels belonging to region i are considered), $\mathbf{q}(a, b)$ is a function that returns 1 in case $a = b$ and 0 otherwise. In this paper, the normalized version of the joint image histogram is used.

The objective of any color correction algorithm is to propose one or several color palette mapping functions, expressed as:

$$\hat{\mathtt{Y}} = \mathbf{f}^i(\mathtt{X}), \tag{6.17}$$

where $\mathbf{f}^i$ is the estimated color mapping function for region i, and $\hat{\mathtt{Y}}$ is the color of the color corrected image $\hat{\mathbf{T}}$, for a given color $\mathtt{X}$ of the target image $\mathbf{T}$. A color palette mapping function cannot map one element of its domain to different elements of its codomain. However, from the observation of a typical joint image histogram of region (see Fig. 6.14 (*a*)) or (*d*)), it is possible to realize that there are several observations of $\mathtt{Y}$ for each of the values in $\mathtt{X}$. Therefore, it must be assumed that the joint image histogram represents a set of *considerably noisy* observations of the ideal mapping function. The noise is mainly due to lack of accuracy in the registration of the images, but other factors might be involved, such as: vignetting, changes in local illumination of the scene, or even different characteristics of the image acquisition device.

6.5.3 Modeling the Mutual Information

For a given region i, the observed mutual information between color $\mathtt{X}$ and color $\mathtt{Y}$ is modelled with a set of Gaussian curves. These curves will model the conditional probability that color $\mathtt{X}$ occurs, given a specific value for variable $\mathtt{Y}$ (notated $P(\mathtt{X} \mid \mathtt{Y} = y)$). In other words, we propose to represent the observed mutual information between variables (obtained by the normalized local joint image histogram) by a set of conditional probabilities, modeled by a Normal distribution:

$$\mathbf{I}^i(\mathtt{X}, \mathtt{Y}) \sim P^i(\mathtt{X} \mid \mathtt{Y} = y) \sim \mathcal{N}^i_y(\mu^i_y, \sigma^i_y), \tag{6.18}$$

where μ^i_y and σ^i_y are the mean and standard deviation of the fitted Normal, for a particular region i and value of $\mathtt{Y} = y$. For each admissible value of y, a Gaussian curve is estimated. Since y is defined in $\{0, ..., 2^n - 1\}$, a typical eight bit depth ($n = 8$) image will be modelled by 256 Gaussians. We refer to this as modeling the horizontal slices of the normalized joint image histogram. Each horizontal slice is modelled using the maximum likelihood method. Let $\mathbf{X}^i_y$ be the vector of observed color values of $\mathtt{X}$ for a given value of $\mathtt{Y} = y$, extracted directly from the local joint image histogram of the region:

$$\mathbf{X}^i_y = \mathbf{I}^i(\mathtt{X}, \mathtt{Y}), \forall X \in \{0, 1, .., 2^n - 1\} \quad and \quad \mathtt{Y} = y. \tag{6.19}$$

The goal is to fit a Gaussian curve to the vector of observations $\mathbf{X}_y$:

$$\mathbf{X}_y \sim \mathcal{N}_y(\mu_y, \sigma_y). \tag{6.20}$$

We propose to model the vector of observations with a truncated normal distribution. In comparison with a standard Gaussian distribution, the truncated Gaussian distribution presents some advantages namely because it can model better the image sensor saturation phenomenons. Cameras often saturate the acquired image in the higher bounds, for example when facing direct sunlight or in the lower bound, when viewing a dark scenario or when the exposure / brightness control is not well adjusted. In practice, it is difficult to achieve the dynamic range of the human eye in a given scene using electronic equipment. This is why images captured by cameras are often over or under saturated. To best cope with these phenomenons, a truncated Gaussian distribution is employed. It can accurately model the saturation near to the sensor's high and low dynamic limits. Within the interval $[a, b]$, the probability density function for a truncated normal is defined as:

$$pdf(z; \mu, \sigma, a, b) = \frac{\phi(z)}{\Phi(b) - \Phi(a)}, \tag{6.21}$$

where z corresponds to the variable that we want to model (in this case, the color distribution of $\mathtt{X}$), a and b are the limits of the interval in which the Gaussian is defined, for images they correspond to $a = 0$ and $b = 2^n - 1$. Outside the interval $[a, b]$, the probability density function is equal to zero. The symbol ϕ represents the probability density function of a standard Gaussian distribution:

$$\phi(z; \mu, \sigma) = \frac{e^{-\frac{(z-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}, \tag{6.22}$$

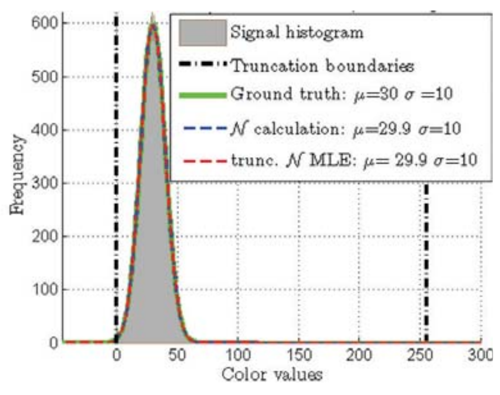
and Φ represents the cumulative distribution function of a standard normal distribution at point t:

$$\Phi(t; \mu, \sigma) = \frac{1}{2} + \frac{1}{2}\mathbf{erf}\Big(\frac{t-\mu}{\sqrt{2\sigma^2}}\Big), \tag{6.23}$$

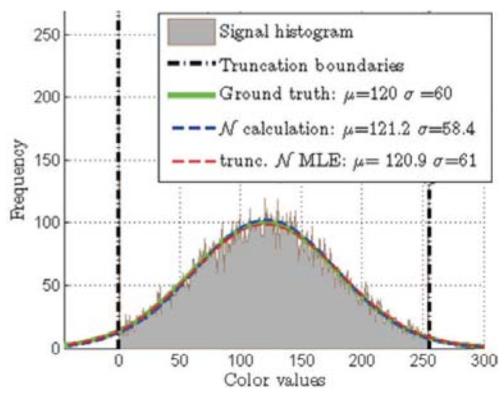
where **erf** is the error function. Rearranging eqs. (6.21) (6.22) and (6.23), the probability density function of a truncated Gaussian distribution is expressed as:

$$pdf(z; \mu, \sigma, 0, 2^n - 1) = \frac{e^{-\frac{(z-\mu)^2}{2\sigma^2}}}{\frac{\sqrt{2\pi\sigma^2}}{2}\Big(\mathbf{erf}(\frac{2^n-1-\mu}{\sqrt{2\sigma^2}}) - \mathbf{erf}(\frac{-\mu}{\sqrt{2\sigma^2}})\Big)}. \tag{6.24}$$

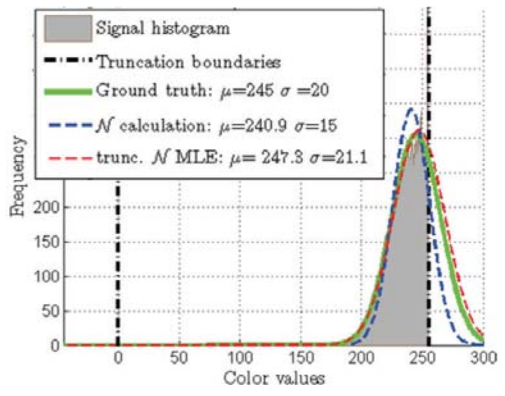
Using the probability density function presented in (6.24), a maximum likelihood estimation (MLE) procedure is used to fit the truncated Gaussian to the vector of observed values ($\mathbf{X}_y$). The method selects values of the model parameters that produce a distribution that gives the observed data the greatest probability, i.e., that maximize the likelihood function. The fitting will return the model parameters μ_y^i and σ_y^i. Figure. 6.13 compares the truncated Gaussian fitting (trunc. $\mathcal{N}$ MLE in Fig. 6.13) with standard Gaussian calculation ($\mathcal{N}$ calculation in Fig. 6.13), i.e., where μ and σ are directly calculated from the vector of observed values. Four cases are presented. In each, a ground truth Normal distribution (Ground Truth in Fig. 6.13) was used to sample the vector of observations (Signal histogram, in Fig. 6.13). The sampling is obtained by adding 10% random noise and saturating the



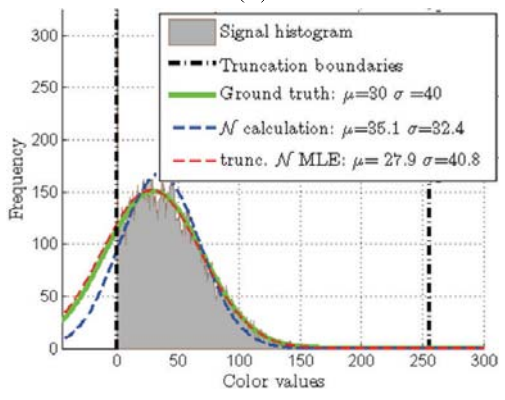
(*a*)



(*b*)



(*c*)



(*d*)

Figure 6.13: Comparison of standard $\mathcal{N}$ calculation (*dashed blue*) with truncated $\mathcal{N}$ maximum likelihood estimation (*dashed red*). A ground truth Gaussian model (*solid green*) was used to generate the observed data sets for color values (*signal histogram*).

signals that lie beyond the truncation limits.

From observing Fig. 6.13, it is possible to realize that in the cases where large portions of the area of the underlying Normal distribution disappear (due to the saturation discussed before), the truncated Gaussian estimation models the Ground Truth distribution more precisely. On the other hand, the standard $\mathcal{N}$ calculation fails to accurately model the signal. This is especially visible in Fig. 6.13 (*c*) and (*d*). In the case of Fig. 6.13 (*a*) and (*b*), a very small portion of the signal was saturated and because of this, both methods have a similar effectiveness. Nonetheless, since the goal is to model color distribution in images, it is expected that the image sensor saturation phenomenons occur often. Hence, the usage of a truncated Gaussian MLE estimation will increase the capability of the algorithm to model the joint image histograms and, as a consequence, improve the performance of the color correction.

Figure 6.14 shows examples of the modeling procedure for two images (*top* and *bottom*). Since the example on the *top* has a more scattered joint image histogram (Fig. 6.14 (*a*)), the modelled Gaussians have considerably larger standard deviations, i.e., they are wider (Fig. 6.14 (*b*)). On the

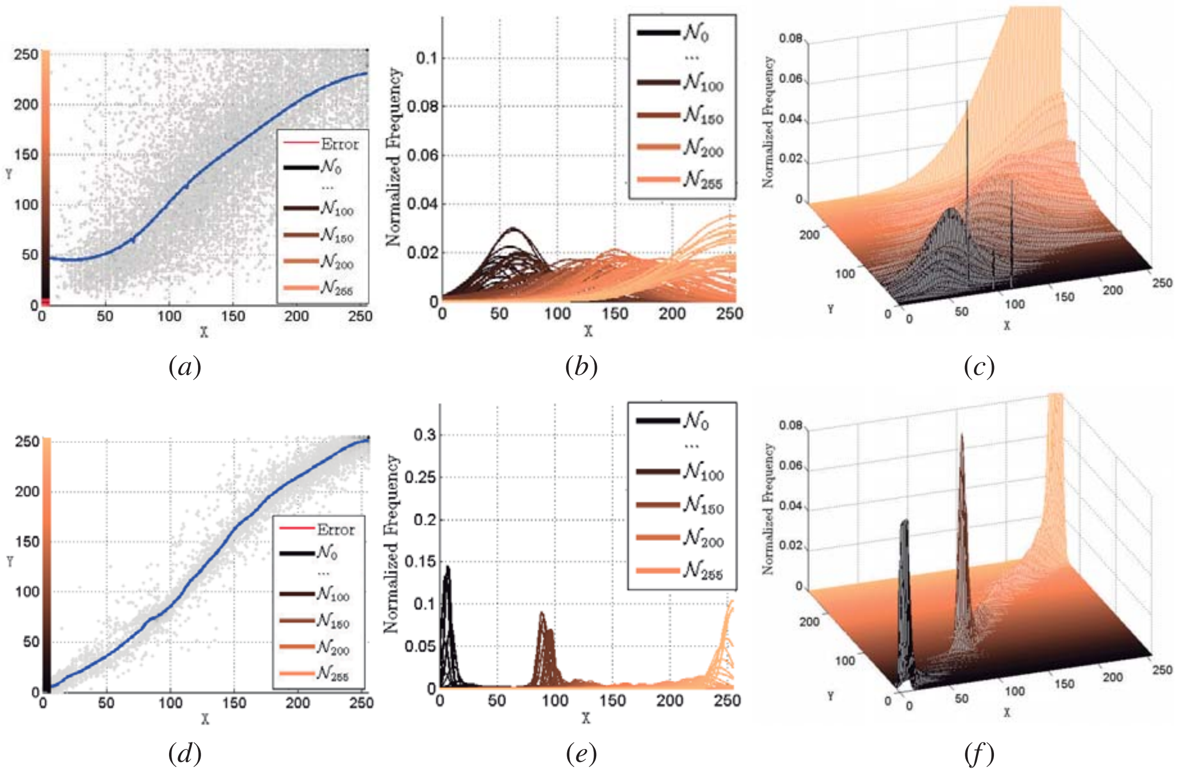


Figure 6.14: Examples of the modeling procedure for two images: case 1, *top* and case 2, *bottom*. The joint image histograms with the estimated color pallete mapping function in blue (*a*) and (*d*): the modeled Gaussians (*b*) and (*e*); and a 3D view of the modeled Gaussians (*c*) and (*f*).

other hand, the example on the bottom shows an image where the joint image histogram is more compact (Fig. 6.14 (*d*)), which leads to peak-like Gaussians (Fig. 6.14 (*e*)). A 3D representation of the joint image histogram as well as the fitted Gaussians is displayed in Fig. 6.14 (*c*) and (*f*). It is also possible to realize the difference in the modeled Gaussians in both cases, wide Gaussians on (*c*) and peak like Gaussians on (*f*).

The limitations of the truncated Gaussian fitting method are related to the number of observations required for properly estimating a probabilistic distribution. However, this is a common restriction to any fitting methodology. In order to infer a reliable statistical model, a reasonable number of observations must be given. This is observable in Fig. 6.14 (*a*). The fitting fails (represented by the red lines near to the lower values of $\mathtt{Y}$) because the joint image histogram contains almost no observations in the interval $\mathtt{Y} \in [0; 10]$. An analysis of the scattering of mappings in Fig.6.14 (*a*) shows that a voting scheme (as proposed in [Jia & Tang 2005]) will ignore a lot of information that could be used for inferring a more accurate mapping function. By fitting a model to the observed data it is expected that the color mappings are more acuratelly modeled.

6.5.4 Color Palette Mapping Functions

The joint image histogram represents the observable mutual information between $\mathtt{X}$ and $\mathtt{Y}$. The color palette mapping function should somehow be inferred from the information present on the joint image histogram. However, the observations present in the joint image histogram must be regarded as considerably noisy. Because of this, the current chapter proposes to model these observations as a set of horizontal Gaussians estimated in 1D space. In section 6.5.3, the s.pdf for probabilistic modeling of the joint image histogram are presented. With this procedure, 256 Gaussian curves (one for each value of $\mathtt{Y}$) are fitted to describe the probability of $\mathtt{X}$ given $\mathtt{Y} = y$. In this section, we will focus on how to compute the color palette mapping function from those modeled Gaussians. Compared to the modeling formulation in section 6.5.3, eq. (6.18), the problem is now reversed: the objective is to find $\mathtt{Y}$, given a value of $\mathtt{X} = x$, which can be written in probabilistic formulation as $P(\mathtt{Y} \mid \mathtt{X{=}x})$. Since each modeled Gaussian should proposes a probability that $\mathtt{X} = x$ is mapped to a given color $\mathtt{Y}$, the sum of the contributions of all these Gaussians is computed to obtain the final color palette mapping function:

$$\mathbf{f}^i(\mathtt{X} = x) \equiv \sum_{y=0}^{2^n-1} y \cdot P^i(\mathtt{Y} = y \mid \mathtt{X} = x), \tag{6.25}$$

using Bayes rule, the conditional probability in eq. (6.25) is expanded:

$$P^i(\mathtt{Y} = y \mid \mathtt{X} = x) = \frac{P^i(\mathtt{X} = x \mid \mathtt{Y} = y) \cdot P^i(\mathtt{Y} = y)}{P^i(\mathtt{X} = x)}, \tag{6.26}$$

where $P^i(\mathtt{X} = y)$ and $P^i(\mathtt{Y} = y)$ are the prior probabilities estimated from the normalized joint image histogram:

$$P^i(\mathtt{X} = x) = \sum_{y=0}^{2^n-1} \mathbf{I}^i(x, y),\ P^i(\mathtt{Y} = y) = \sum_{x=0}^{2^n-1} \mathbf{I}^i(x, y), \tag{6.27}$$

and $P^i(\mathtt{X} = x \mid \mathtt{Y} = y)$ is given by the probability density function of Gaussian y and value $z = x$ (see eq. (6.21)):

$$P^i(\mathtt{X} = x \mid \mathtt{Y} = y) = \frac{e^{-\frac{(x-\mu_y^i)^2}{2{\sigma_y^i}^2}}}{\frac{\sqrt{2\pi{\sigma_y^i}^2}}{2}\left(\mathbf{erf}(\frac{2^n-1-\mu_y^i}{\sqrt{2{\sigma_y^i}^2}}) - \mathbf{erf}(\frac{-\mu_y^i}{\sqrt{2{\sigma_y^i}^2}})\right)}. \tag{6.28}$$

By applying eq. (6.28) to all values of $\mathtt{X}$, a value of $\hat{\mathtt{Y}}$ is computed for each $\mathtt{X} = x$ and thus the color palette mapping function is computed for the given region i. The usage of the $P^i(\mathtt{Y} = y \mid \mathtt{X} = x)$ in eq. (6.25) instead of $P^i(\mathtt{X} = x \mid \mathtt{Y} = y)$ directly, is motivated by the reason that the marginal probabilities that appear in the expansion using the Bayes rule (eq. (6.26)) will weight the importance of each modeled Gaussian based by the ratio $P^i(\mathtt{Y} = y)$ over $P^i(\mathtt{X} = x)$, which will improve the quality of the estimation of the color palette mapping function.

As discussed in section 6.5.3, the fitting of some Gaussians may fail due to insufficient number of observations. In this case, the Gaussian is not valid and for that $\mathtt{Y} = y$, the value of $P^i(\mathtt{X} = x \mid \mathtt{Y} = y)$ is set to zero. One of the advantages of this approach is that the modeled Gaussians are defined in the whole range of admissible values of $\mathtt{X}$. Therefore, eq. (6.28) is defined for the whole range of possible values of $\mathtt{X}$. Another advantage is that observed data is fitted to a statistical Normal distribution model. Because of this, noise is considerably reduced. Noise in the observed data is mainly due to lack of accuracy in the registration between the source and target images, but other factors such as local changes in scene illumination may also be accounted. This is another advantage of the proposed approach when compared to others: model fitting increases the ability to cope with noise in the joint image histogram.

Figure 6.14 (a) and (d) show (in blue) the estimated color palette mapping function for the two cases presented. Both functions properly fit the input data present in the respective joint image histograms.

6.5.5 Expanding the Color Palette Mapping Functions

Once the color palette mapping functions are computed for all the regions in $\mathbf{T}_p$, they must be applied. If only the overlapping portion of the target image, $\mathbf{T}_p$, is considered, the operation is straightforward. However, to be complete, the process of color correction must correct the entire target image $\mathbf{T}$ (see Fig. 6.1 (e)). In other words, for each of the pixels in $\mathbf{T}$, even those which do not belong to $\mathbf{T}_p$, one of the computed color palette mapping functions must be selected. We propose to make use of the color resemblances between $\mathbf{T}$ and $\mathbf{T}_p$ to perform this expansion. The process starts with a second color segmentation, this time of image $\mathbf{T}$. Let $\mathbf{T}^j$ be the *jth* region of image $\mathbf{T}$. Each region has a mean color associated to it, denoted as $\overline{rgb}^j$ in the case of region $\mathbf{T}^j$, and $\overline{rgb_p}^i$, in the case of region $\mathbf{T}_p^i$. As discussed before, the proposed approach produces a color palette mapping function $\mathbf{f}^i$ for every region in $\mathbf{T}_p$. The objective is to find, for a given region j of the entire target image, the index i of the mapping function that should be applied to the pixels of that region. To do this, we propose to use the ith region that is, on average, photometrically closer to the mean color of the jth region:

$$\mathbf{map}(j) = \mathbf{argmin}_i(\| \overline{rgb}^j - \overline{rgb_p}^i \|). \tag{6.29}$$

In the target image $\mathbf{T}$ space, the selected color palette mapping function is given by the mapping presented in eq. (6.29). That is, for a given region $\mathbf{T}^j$ the selected color palette mapping function is $\mathbf{f}^{\,\mathbf{map}(j)}$.

It should be noted that very few of the methods presented in the literature propose methodologies for expanding the color palette mapping functions. Also, the result of the expansion, the image mosaic, or the whole corrected image, can only be analyzed qualitatively because the non overlapping portions of the image cannot be compared to the source image.

In the next sections, results are given. They are divided into three parts. First, a case study is presented to illustrate the details of the proposed algorithm. Second, qualitative and quantitative results are presented by comparing the proposed approach with nine other state of the art algorithms. Finally, an analysis on the effects of the Mean shift based color segmentation preprocessing step to the overall color correction performance of the proposed approach is presented.

6.5.6 A Case Study

In this section the s.pdf of the proposed algorithm are shown in detail. To start, a pair of source and target images of a bird are registered in order to obtain a mosaic (Fig. 6.15 *(a)*, source image on the *left* and target image on the *right*). As usual in pairs of images requiring color correction, the mosaic shows artifacts due to photometric miss calibration between the images. The overlapping areas (Fig. 6.15 *(b)* from the source image, and Fig. 6.15 *(c)* from the target image) of both images show clear differences in color. The three channels of the target image are processed separately. The following procedures are performed for each of the color channels.

The first step of the proposed algorithm is to color segment the image (section 6.5.1). This is done by applying Mean shift to the overlapping region of the target image (T_p), i.e, Fig. 6.15 *(c)*. In this case, the procedure segments nine regions from the image, as shown in Fig. 6.15 *(d)*. The segmentation step is crucial to the color correction process: since the image is split into several regions, each is used to compute the respective local color palette mapping function. To do so, for each region, the joint image histogram is computed. This is the second step of the algorithm, presented in section 6.5.2. We focus our analysis on just three of the segmented regions: region 1, the blue sky behind the bird (Fig. 6.16 *(a)*), region 5, the red beak of the bird (Fig. 6.16 *(d)*) and region 7, the brown tree branch near the bottom of the image (Fig. 6.16 *(g)*).

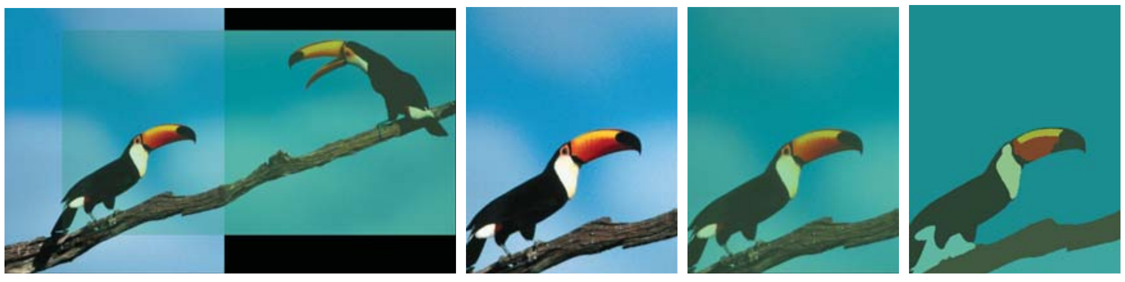


Figure 6.15: The images used in this case study: the mosaic of both source and target images is shown without color correction *(*a), the source image is on the *left* and the target image is on the *right* of the mosaic; the areas of the source and target images that overlap, respectively S_p *(b)* and T_p *(c)*; and the result of color segmentation of T_p using Mean shift *(d)*.

The normalized joint image histograms of regions 1, 5 and 7 are displayed in Fig. 6.16 *(b)*, *(e)* and *(h)*, respectively. Region 1 shows observations in the interval $0 < \mathtt{X} < 75$. Region 5, the red beak of the bird, has observations with higher values ($40 < \mathtt{X} < 180$), since we are looking at the red channel of the image. Region 7, the tree branch, shows observations is the $40 < \mathtt{X} < 110$ interval.

The third step of the algorithm (section 6.5.3) is to model the information present in the joint image histogram with a set of 256 Gaussian curves (one for each value of $\mathtt{Y}$), that describe the condi-

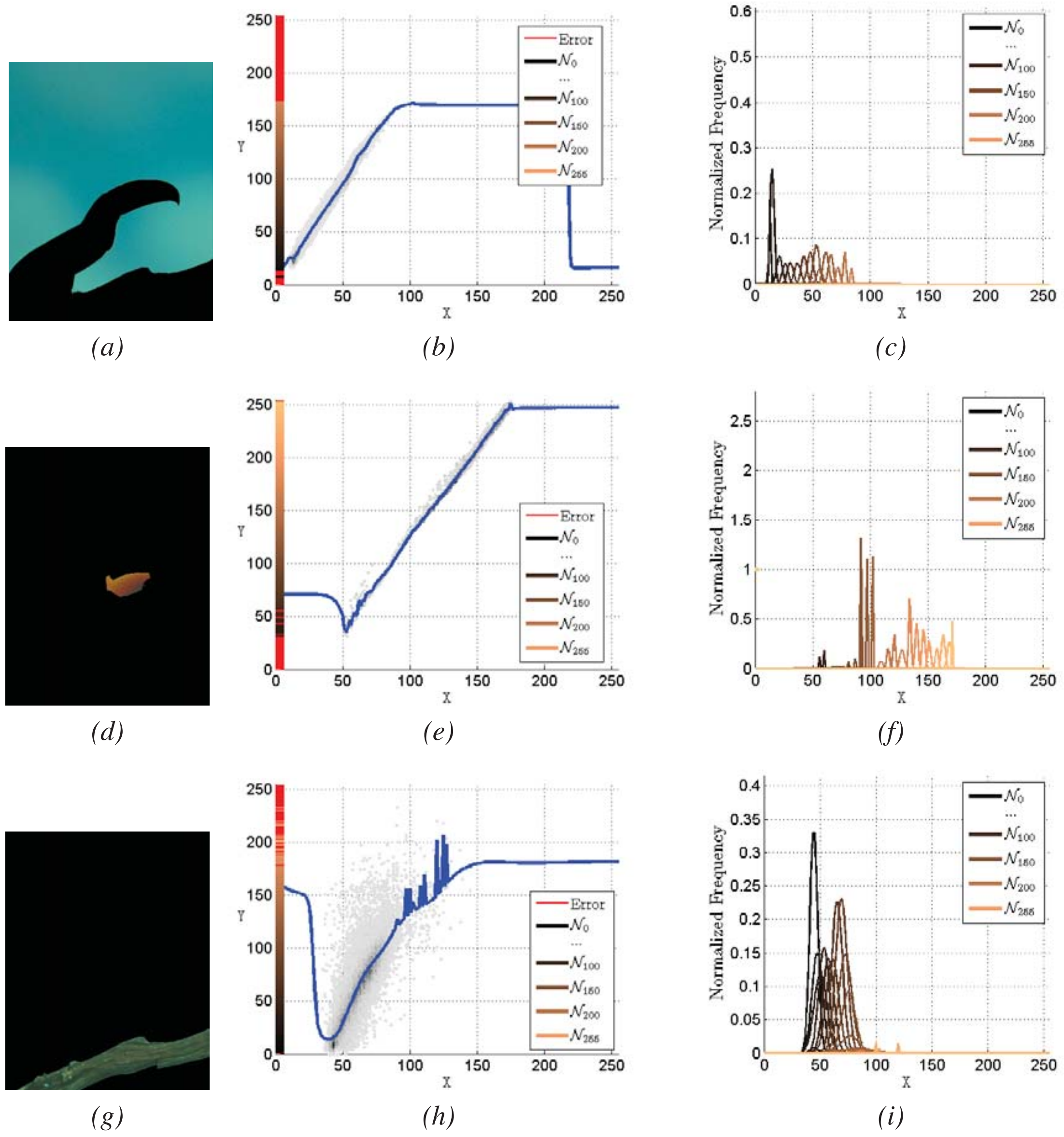


Figure 6.16: Graphs from region 1 (*first row*), 5 (*second row*) and 7 (*third row*): the regions, *(a)*, *(d)* and *(g)*; the joint image histograms with the computed local color palette mapping function in blue *(b)*, *(e)* and *(h)*; and the fitted Gaussians *(c)*, *(f)* and *(i)*.

tional probability $P(\mathtt{X} \mid \mathtt{Y} = y)$. Each segmented region is modelled independently according to this procedure. Figures 6.16 *(c)*, *(f)* and *(i)* show the Gaussian curves that are estimated for regions 1, 5 and 7, respectively. When the observations provided by the joint image histogram are more coherent, i.e, the values of $\mathtt{Y}$ are mapped to a short range of values in $\mathtt{X}$, there is a high probability of mapping $\mathtt{X}$ to a single $\mathtt{Y}$. As a result, the probability density functions of the estimated Gaussians resemble a peak (small value of σ and μ positioned where there is a large mapping probability). This is noted in the cases of regions 1 and 5: the coherent data in the joint image histogram (Figures 6.16 *(b)* and *(e)*), leads to the peak like Gaussians shown in Fig. 6.16 *(c)* and *(f)*. On the contrary, when the joint image histogram shows less coherent data, i.e, the values of $\mathtt{Y}$ are mapped to a large range of values in $\mathtt{X}$, the Gaussians tend to be wider (with a larger σ), since the probability of mapping is scattered through a large range of possible values. This is noticeable in the case of region 7, where the incoherent data in the joint image histogram (Fig. 6.16 *(h)*) results in the wider Gaussians present in Fig. 6.16 *(i)*.

The fourth step of the algorithm, detailed in section 6.5.4, is to obtain the color palette mapping function for each region. Figures 6.16 *(b)*, *(e)* and *(h)* show, in blue, the computed color palette mapping functions for regions 1, 5 and 7, respectively. It is possible to observe that these three mapping functions are very different. For example, target color $\mathtt{X} = 50$ is mapped to $\mathtt{Y} \simeq 90$ in region 1, $\mathtt{Y} \simeq 40$ in region 5, and $\mathtt{Y} \simeq 20$ in region 7; target color $\mathtt{X} = 100$ is mapped to $\mathtt{Y} \simeq 170$ in region 1, $\mathtt{Y} \simeq 125$ in region 5, and $\mathtt{Y} \simeq 140$ in region 7; finally, target color $\mathtt{X} = 150$ is mapped to $\mathtt{Y} \simeq 170$ in region 1, $\mathtt{Y} \simeq 200$ in region 5, and $\mathtt{Y} \simeq 180$ in region 7. These examples show that the mapping of color varies significantly, from region to region. This observation endorses the use of local approaches to color correction instead of global ones.

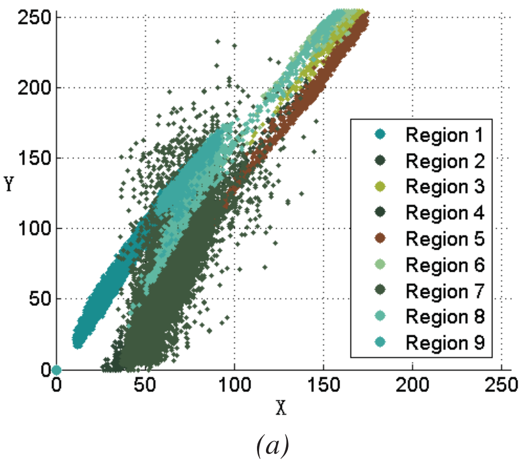
Figure 6.17 (*a*) shows a composite joint image histogram of all the regions. By observing the graph it is possible to realize that it is very hard to fit a single color palette mapping function to the global joint image histogram. For every value of $\mathtt{Y}$, there are wide ranges of mappings of $\mathtt{X}$. Using region local joint image histograms facilitates the computation of local color palette mapping functions. Figure 6.17 (*b*) shows the color palette mapping functions obtained in this case study. Although these functions are defined in the $\mathtt{X} \in \{0, ..., 255\}$ interval (see the examples in 6.16 (*b*), (*e*) and (*h*)), only portions of the functions are displayed for a better understanding of the graph. The color palette mapping functions fit the local region data very well, which explains the efficiency of the proposed color correction algorithm.

Finally, the fifth step of the algorithm is to apply the computed set of color palette mapping functions to the target image (section 6.5.5). Note that, as explained, these functions are computed from the region of the mosaic where mutual information between source and target color is observable. In other words, although the color palette mapping functions are computed based on the overlapping portion of the target image $\mathbf{T}_p$ (Fig. 6.15 *(c)*), they must be applied to the whole target image $\mathbf{T}$ (Fig. 6.18 *(a)*). To do so, a second Mean shift color segmentation is applied to $\mathbf{T}$, as shown in Fig. 6.18 *(b)*. Figure 6.18 *(c)* shows the corrected target image. Figure 6.18 *(d)* shows the mosaic obtained using the source and corrected image. Compared to the initial mosaic shown in Fig. 6.15 (*a*) it is possible to observe a much better photometric resemblance between both images. In fact, no artifacts are noticeable, which is a good indication of the color correction's effectiveness.

6.5.7 Comparison with other Algorithms

In this section quantitative results will be presented that show that the proposed approach outperforms nine other state of the art color correction methods. In section 6.5.1 the usage of Mean shift as a preprocessing step was explained. The *matlab* implementation available at http://coewww.rutgers.edu/riul/research/code.html was used for implementing color segmentation. As in most of the methods, there are several parameters in this algorithm that may change the outcome of the segmentation: spatial band width, range band width, gradient window radius, mixture parameter, edge strength threshold and minimum region area (see the toolbox for more detailed information). The results displayed in this section were obtained by using the toolbox's default values for all parameters. In this way, a more careful selection of those parameters could, to some extent, improve the results of the color correction algorithm here presented.

In order to test the performance of the proposed approach against other algorithms, two data sets of a recent performance evaluation [Xu & Mulligan 2010] were used. They consist of a synthesized data set of 40 image pairs and a real image data set of 23 image pairs. The synthesized image pairs were obtained by clipping the source and target images from a larger original image. Then, the color in the target image is altered using an image editing software. In the case of the real image data set, the target and source images are different shots of the same scene. They might or might not be



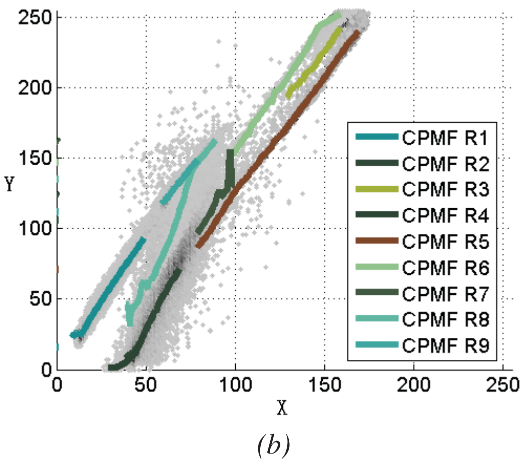


Figure 6.17: emph(a) Joint image histograms of the color segmented regions (shown in Fig. 6.15 (*d*)). *(b)* Color palette mapping functions (CPMF). Data is assigned to the corresponding region color (see Fig. 6.15 *(d)*).

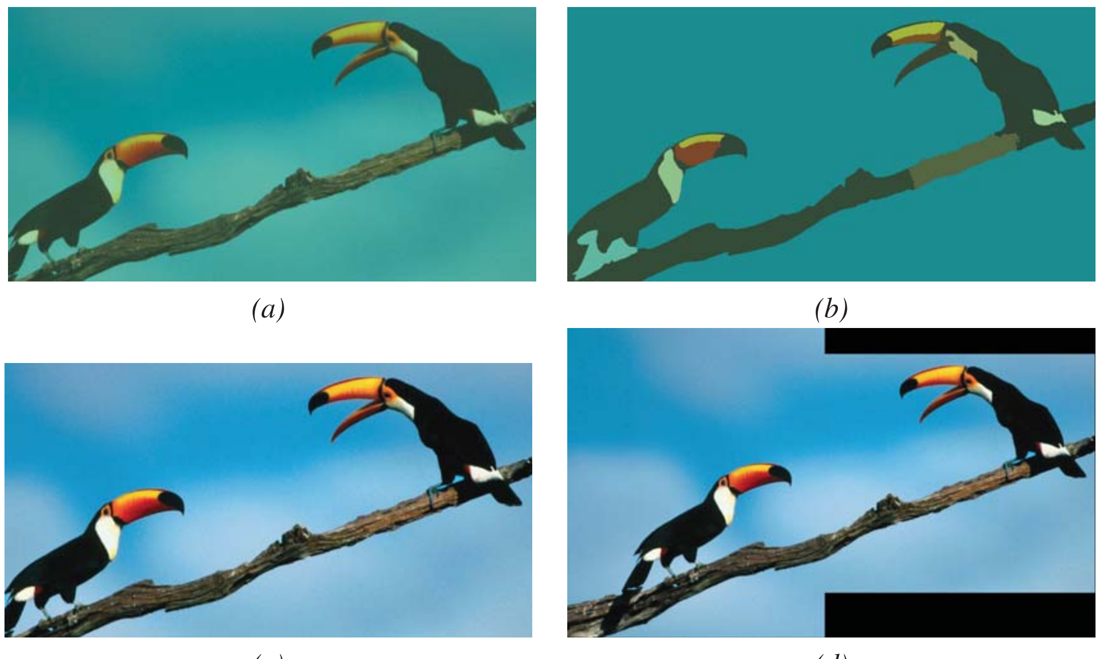

(a) *(b)* *(c)* *(d)*

Figure 6.18: Results of this case study: the target image (*a*); the Mean shift color segmentation of the target image (*b*); the corrected image (*c*); the mosaic obtained after performing color correction with the proposed approach (*d*).

taken using the same camera and/or at exactly the same time. While the real image pair emulates the conditions normally found by a color correction algorithm, the synthesized data-set is much easier to register and so, conclusions can be made whether an algorithm is more or less sensitive to registration accuracy.

Figure 6.19 shows the mosaics of some of the image pairs present on both data sets. The registra-

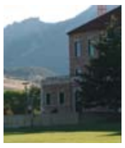
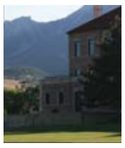
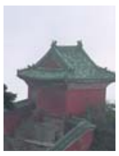
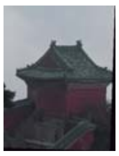
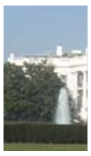
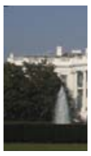
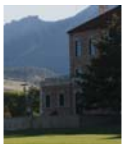
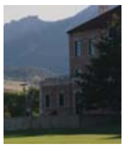
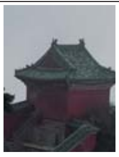
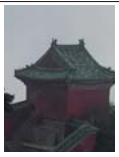
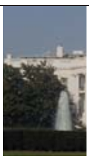
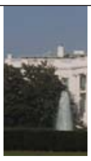
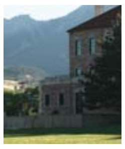
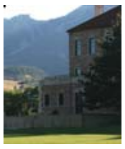
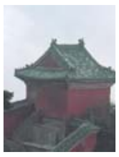
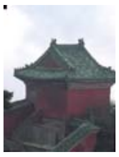
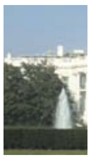
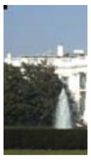
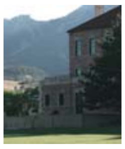
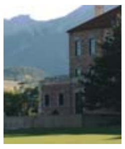
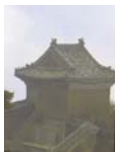
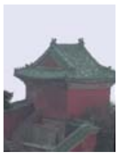
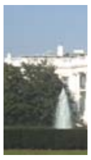
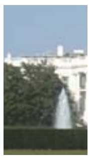
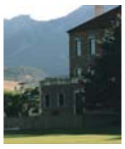
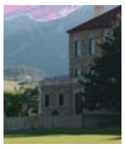
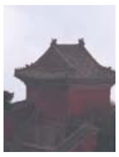
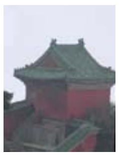
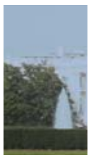
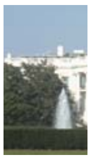
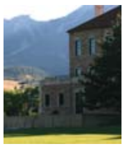
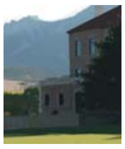
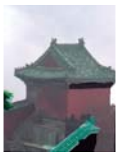
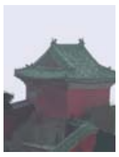
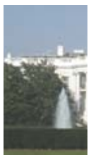
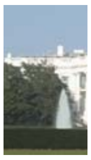
Table 6.6: Algorithms tested in this evaluation. The algorightm presented in this paper, will be referred to as #11.

Name of the Approach	Reference	Alg. #
Baseline (Target Image)	—	#1
Global Color Transfer in RGB	Xiao 06 [Xiao & Ma 2006]	#2
Global Color Transfer	Reinhard 01 [Reinhard *et al.* 2001]	#3
Cumulative Histogram Mapping	Fecker 08 [Fecker *et al.* 2008]	#4
Gain Compensation	Brown 07 [Brown & Lowe 2007]	#5
Principal Components Analysis	Zhang 04 [Zhang & Georganas 2004]	#6
Brightness Transfer Function	Kim 08 [Kim & Pollefeys 2008]	#7
Local Color Transfer	Tai 05 [Tai *et al.* 2005]	#8
Tensor Voting in Joint Image Space	Jia 05 [Jia & Tang 2005]	#9
Mean shift based Local Color Transfer	section 6.3	#10
Probabilistic modeling of LCPMF	section 6.5	#11

Table 6.7: Corrected overlapping areas of image pair numbers 6, 35 and 37 of the synthetic data-set. The **PSNR** (metric 1, *Mt 1*) and **S-CIELAB** (metric 2, *Mt 2*) scores are displayed. Best scores are highlighted in blue.

Synthetic image pair 6		Synthetic image pair 35		Synthetic image pair 37	
Source ($\mathbf{S}_p$)	Target ($\mathbf{T}_p$) or #1	Source ($\mathbf{S}_p$)	Target ($\mathbf{T}_p$) or #1	Source ($\mathbf{S}_p$)	Target ($\mathbf{T}_p$) or #1
#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*	#;*Mt 1*; *Mt 2*
#2 ; 22.7 ; 16.1	#3 ; 20.7 ; 18.4	#2 ; 21.7 ; 16.4	#3 ; 21.0 ; 15.7	#2 ; 21.1 ;11.9	#3 ; 21.3 ; 11.8
#4 ; 32.8 ; 5.1	#5 ; 23.2 ; 13.0	#4 ; 25.9 ; 3.7	#5 ; 23.3 ; 8.0	#4 ; 35.8 ;1.8	#5 ; 24.4 ; 7.7
#6 ; 34.7 ; 4.0	#7 ; 43.3 ; 0.5	#6 ; 25.2 ; 5.1	#7 ; 26.1 ; 2.7	#6 ; 31.3 ;4.3	#7 ; 33.6 ; 2.4
#8 ; 21.4 ; 9.5	#9 ; 42.0 ; 0.7	#8 ; 21.1 ; 5.7	#9 ; 26.1 ; 2.5	#8 ; 23.8 ;10.7	#9 ; 34.0 ; 2.4
#10 ; 33.2 ; 4.4	#11 ; 42.5 ; 0.4	#10 ; 25.6 ; 3.3	#11 ; 26.4 ; 2.6	#10 ; 30.1 ;4.4	#11 ; 37.7 ; 1.0

Table 6.8: Corrected overlapping areas of image pair numbers 1, 12 and 21 of the real data-set. The **PSNR** (metric 1, *Mt 1*) and **S-CIELAB** (metric 2, *Mt 2*) scores are displayed. Best scores are highlighted in blue.

Real image 1		Real image 12		Real image 21	
 Source ($\mathbf{S}_p$)	 Target ($\mathbf{T}_p$) or #1	 Source ($\mathbf{S}_p$)	 Target ($\mathbf{T}_p$) or #1	 Source ($\mathbf{S}_p$)	 Target ($\mathbf{T}_p$) or #1
#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*	#; *Mt 1*; *Mt 2*
 #2 ; 22.7 ; 8.1	 #3 ; 21.9 ; 8.6	 #2 ; 15.3 ;20.5	 #3 ; 15.4 ; 20.4	 #2 ; 15.1 ; 19.3	 #3 ; 14.9 ; 20.6
 #4 ; 21.8 ; 5.4	 #5 ; 21.8 ; 6.4	 #4 ; 24.4 ;3.9	 #5 ; 20.5 ;8.9	 #4 ; 25.0 ; 3.8	 #5 ; 20.3 ; 8.8
 #6 ; 22.0 ; 5.5	 #7 ; 22.5 ; 5.1	 #6 ; 22.5 ;10.7	 #7 ; 23.6 ;3.5	 #6 ; 24.6 ; 3.6	 #7 ; 24.9 ; 3.4
 #8 ; 21.7 ; 6.5	 #9 ; 21.0 ; 7.4	 #8 ; 23.1 ;5.8	 #9 ; 23.4 ;3.6	 #8 ; 21.0 ;7.4	 #9 ; 25.0 ; 3.5
 #10 ; 21.3 ; 7.3	 #11 ; 26.1 ; 3.5	 #10 ; 21.6 ; 7.2	 #11 ; 28.2 ;3.1	 #10 ; 23.8 ; 5.4	 #11 ; 26.2 ; 3.2

tion for each pair of images was not provided by the authors of [Xu & Mulligan 2010], just the image pairs. Because of this, a manual process of hand labeling pixels in both images was done to obtain a new registration. Since registration has some impact on color correction, the results that are presented are not exactly the same as those in [Xu & Mulligan 2010], although they are similar. Hence, the registration is never pixel accurate in both data sets although, as mentioned above, it is significantly more accurate in the synthetic data set. Note that most of the methods used in this evaluation propose no way of expanding the color palette mapping functions to the whole target image. However, to build a mosaic with the color corrected image, it is necessary to obtain a correction for the entire

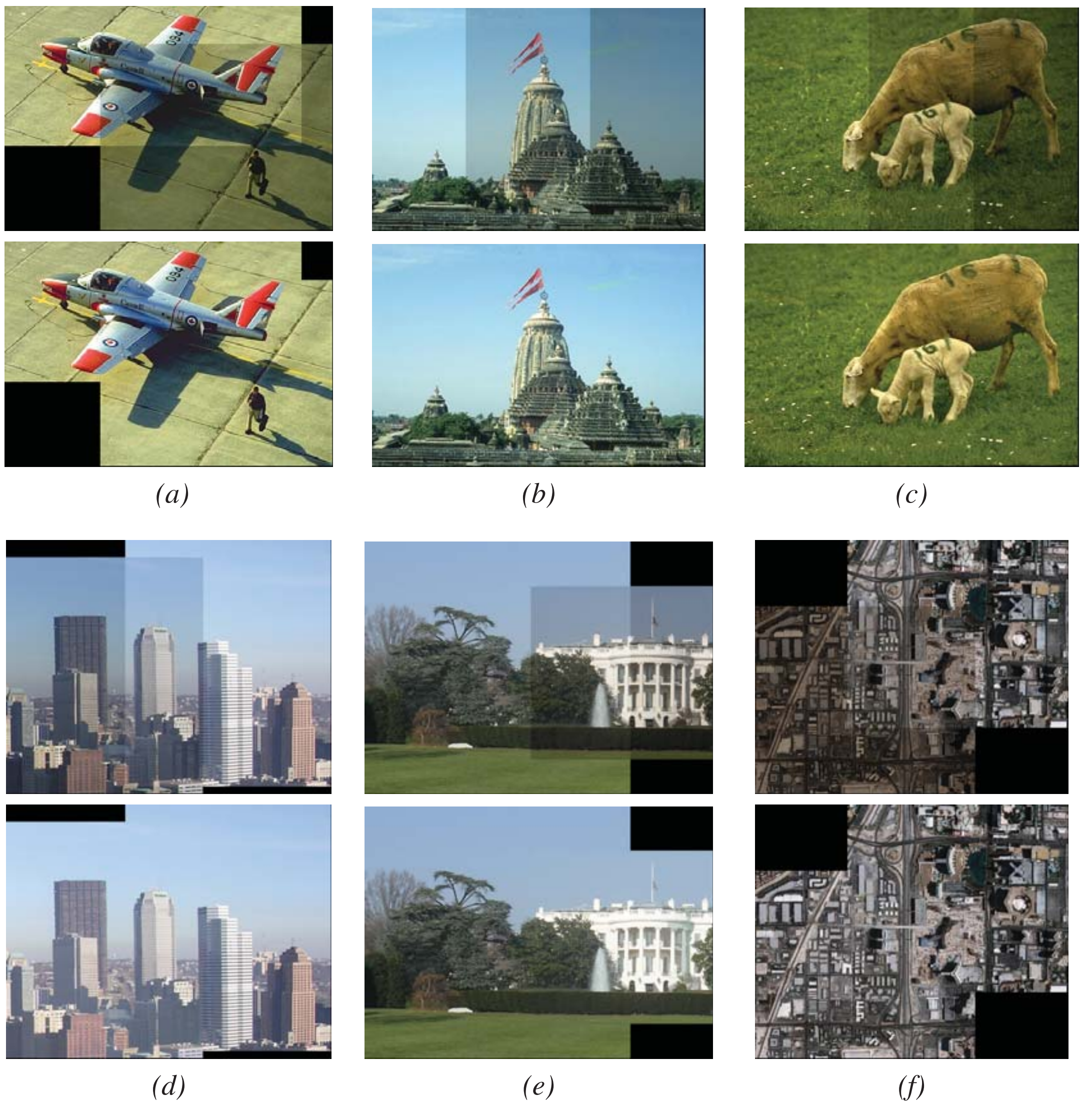


Figure 6.19: Mosaics obtained using the image pairs from the data set of [Xu & Mulligan 2010]. Synthesized data set: (*a*) image 17; (*b*) image 32 ; and (*c*) image 40. Real data set: (*d*) image 8; (*e*) image 21; and (*f*) image 23. (top) Original mosaics; $(bottom)$ Results from the proposed approach.

target image. Because of this, in the qualitative evaluation of the mosaics, only the results from the proposed approach are shown in Fig. 6.19. In all examples, the proposed approach removes most of the artifacts originally present and produces mosaics with smooth color transitions.

In order to compare the results of the proposed approach with the state of the art, eight of the nine algorithms used in [Xu & Mulligan 2010] were applied to the same data sets. The implementations of algorithms #2 through #9 were provided by the authors of [Xu & Mulligan 2010]. Regarding the missing algorithm [Pitie *et al.* 2005], it was not possible to find a public implementation to guarantee a fair comparison. However, the algorithm presented in [Pitie *et al.* 2005] did not reach the best performance in none of the tests presented in [Xu & Mulligan 2010]. Another recently published algorithm [Oliveira *et al.* 2011] was also included in the evaluation. Table 6.6 lists all the algorithms tested in the current paper. Each is attributed a number for easier reference in the discussion of results. Algorithm #1 is the baseline for comparison. In this case, no correction is performed. The corrected image is equal to the target image.

The performance of a color correction approach is quantitatively assessed by comparing the overlapping portions of the color corrected image and the source image. To test the images and evaluate the algorithms, two different image comparison metrics were used. The first, proposed in [Xu & Mulligan 2010], is called peak signal-to-noise ratio (**PSNR**). It measures the color similarity. The higher the value of this score, the more similar are the two images. The second evaluation metric is the spatial cielab (**S-CIELAB**), initially proposed in [Zhang & Wandell 1997]. This metric is well accepted as one of the standards for assessing the similarity between two images. Since it measures image dissimilarity, the lowest scores are, in this case, associated with the best performing color correction algorithms.

Table 6.7 shows four image pairs from the synthetic data-set. Pairs number 6, 11, 35 and 37 are displayed. The color corrected overlapping portion of the target image ($\hat{\mathbf{T}}_p$) is shown for all methods. Below, the **PSNR** and **S-CIELAB** scores are presented. In the case of image 6, a visual analysis concludes that methods #4, #6, #7, #9, #10 and #11 are able to produce a good color corrected image. Numerically, algorithms #7 and #11 obtain the highest similarity scores. The same occurs for image pair 11, where most methods effectivelly correct the image. However, the proposed approach (#11) obtains the highest similarity scores in both metrics. Pairs 35 and 37, on the other hand, seem harder to correct. In pair 35, several algorithms fail to properly correct the images, while in 37, algorithm #11 does the best job at recovering the color of the blue sky and the red shirt. This is reflected on the numerical scores as well.

Table 6.8 shows images 1, 4, 12 and 21 from the real data-set. In image 1, due to a poor registration, some of the algorithms fail. This is particularly observable at the top of the image. The colors of the sky and hills are very hard to correct for most of the algorithms. Algorithm #9 shows inclusively a *catastrophic* failure, painting the sky in magenta colors. The proposed approach is the algorithm that better corrects the image, obtaining the best scores in both metrics. It also achieves the best scores in the cases of images 4, 12 and 21.

Tables 6.9 and 6.10 show the quantitative results obtained by all algorithms on both the synthetic and real image data-sets. For the synthetic data-set, in the case of **PSNR** color similarity the proposed approach (#11) achieves an average score of 29.9, outperforming all other algorithms. The second best scoring algorithm is #7 (29.0 score), followed by #9 (28.8 score). Regarding the **S-CIELAB** evaluation of the synthetic data-set, the proposed approach also achieves the best average result (2.6 score), followed by algorithms #7 (2.7) and #9 (2.8). Algorithm #1 corresponds to the baseline approach, i.e, no color correction is performed in this case. Compared to the synthetic data-set, we can notice that, for the real data-set, the mean score of algorithm #1 is lower for **PSNR** and higher for **S-CIELAB**. This shows that in the real data-set, source and target images are, on average, photometrically more distant. As a consequence, the real data-set should be harder to color correct when compared to the synthetic data-set. In fact, this conclusion can be taken from the following analysis: for the synthetic data-set, the average **PSNR** score of the best 3 methods is 29.2, which accounts for a 10.4 improvement against the baseline 18.8 score. In the real data-set the improvement is of 6.5; this shows that the real data-set is harder to color correct when compared to the synthetic data-set. Analyzing the **PSNR** performance in the real data-set, the proposed approach (algorithm #11) achieves an average score of 24.7, followed once again by algorithms #7 and #9 with 22.5 score. In terms of the **S-CIELAB** metric, the proposed approach has again outperformed all methods

Table 6.9: Mean and standard deviations of the **PSNR** scores for each algorithm in both data sets. Best results are highlighted in blue.

	PSNR color similarity										
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
	Synthesized image data-set (40 images)										
μ	18.8	19.3	19.8	27.7	21.3	27.3	29.0	21.7	28.8	26.2	**29.9**
σ	4.0	5.7	4.9	6.7	2.5	7.4	8.7	3.2	8.2	5.7	9.2
	Real image data-set (23 images)										
μ	16.7	19.4	20.2	22.3	20.4	20.9	22.5	21.2	22.5	22.0	**24.7**
σ	3.4	6.0	6.8	4.1	3.3	4.9	3.9	3.0	4.1	4.0	3.7

Table 6.10: Mean and standard deviations of the **S-CIELAB** scores for each algorithm in both data sets. Best results are highlighted in blue.

	S-CIELAB color dissimilarity										
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
	Synthesized image data-set (40 images)										
μ	15.5	17.8	16.2	3.1	8.6	4.2	2.7	8.6	2.8	4.2	**2.6**
σ	7.3	10.1	7.6	1.6	3.1	3.7	1.7	3.4	1.6	1.8	1.8
	Real image data-set (23 images)										
μ	18.5	17.5	17.1	6.0	9.4	9.5	5.7	8.2	6.0	7.1	**4.8**
σ	9.5	9.7	10.3	3.3	4.8	8.3	3.4	4.2	4.0	3.8	2.5

with 4.8 score. Algorithm #7 is confirmed as the second best scoring method with 5.7, followed by algorithm #9 score of 6.0. Another interesting observation to make is that the difference in average score from the proposed approach to the other methods is larger for the real data-set. For example, the comparison to the second best scoring algorithm in each data-sets in each metric, results in the following: in the synthetic data-set the difference in scores is $29.9 - 29.0 = +0.9$ for **PSNR**, and $2.6 - 2.7 = -0.1$ for **S-CIELAB**. In the real data-set the differences are $24.7 - 22.5 = +2.2$ for **PSNR** and $4.8 - 5.7 = -0.9$ for **S-CIELAB**. The proposed approach not only scores more that all other methods, it outscores (on average) by a larger margin in harder data-sets. It seems that the proposed approach is more robust to difficulties in data-sets, such as the lack of accurate registrations between source and target images.

Table 6.11 compares the results obtained in this chapter with the evaluation from [Xu & Mulligan 2010]. Methods are ranked according to the average score of the metric. The results from this evaluation are consistent with those from [Xu & Mulligan 2010], since the sorting of the algorithms remains similar. For example, algorithms #7 and #9 were the best in [Xu & Mulligan 2010] and are still amongst the best. However, in the lower part of the ranking, the sorting is somewhat different from the evaluation from [Xu & Mulligan 2010]. This could be explained by the difference in registration between image pairs, i.e., algorithms may have different sensitivity to the source target image registration accuracy.

Finally, in Tables 6.12 and 6.13, a summary of the evaluation performed in this chapter is presented. All four cases, i.e., two data-sets and two evaluation metrics, are presented. The first four rows show the percentage of images (out of the total in that data-set) where the algorithm obtained the best (1^{st}), second best (2^{nd}), third best (3^{rd}), or worse ($> 3^{rd}$) score. Regarding the synthetic data-set, the proposed approach obtains the best **PSNR** score in 82.5% of the images, and the best **S-CIELAB** score in 80% of the images. In the case of the real data-set, the proposed approach obtains the best

Table 6.11: Algorithms sorted by their average score. #10 and #11 were not evaluated (*NE*) in [Xu & Mulligan 2010], since they are posterior.

	From Xu2010		Evaluation from the current paper			
	PSNR rank		PSNR rank		S-CIELAB rank	
Alg. #	Syn.	Real	Syn.	Real	Syn.	Real
#2	7^{th}	7^{th}	10^{th}	10^{th}	10^{th}	10^{th}
#3	8^{th}	6^{th}	9^{th}	9^{th}	9^{th}	9^{th}
#4	6^{th}	8^{th}	4^{th}	4^{th}	4^{th}	3^{rd}
#5	5^{th}	4^{th}	8^{th}	8^{th}	7^{th}	7^{th}
#6	3^{rd}	5^{th}	5^{th}	7^{th}	5^{th}	8^{th}
#7	1^{st}	1^{st}	2^{nd}	2^{nd}	2^{nd}	2^{nd}
#8	4^{th}	3^{rd}	7^{th}	6^{th}	7^{th}	6^{th}
#9	2^{nd}	2^{nd}	3^{rd}	2^{nd}	3^{rd}	3^{rd}
#10	*NE*	*NE*	6^{th}	5^{th}	5^{th}	5^{th}
#11	*NE*	*NE*	1^{st}	1^{st}	1^{st}	1^{st}

PSNR score in 73.9% of the images and the best **S-CIELAB** score in 87% of the cases. In all four cases, results show that the proposed approach obtains a better score (row 1^{st}) when compared to all other algorithms, at least in 73.9% of the images. Also, the second best average scoring algorithm, reference #7 (see Table 6.11), does not show the same consistency in the first or second best scoring position. Furthermore, the proposed approach is amongst one of the three best scoring algorithms in 97.5%, 92.5%, 100.0% and 95.7% of the images, respectively for the synthetic **PSNR**, synthetic

Table 6.12: Evaluation of the performance of the Algorithms according to the **PSNR** score. Rows represent: the % of images where the algorithm achieved the best (1^{st}), second best (2^{nd}), third best (3^{rd}), or worse ($> 3^{rd}$) score. The last row shows the % of times an algorithm failed to color correct the image. Best results are highlighted in blue.

	PSNR color similarity									
	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
	Synthetic image data-set (40 images)									
1^{st}	7.5	5.0	0.0	0.0	0.0	7.5	0.0	2.5	0.0	**82.5**
2^{nd}	5.0	7.5	15.0	0.0	12.5	45.0	0.0	32.5	15.0	2.5
3^{rd}	0.0	0.0	15.0	0.0	12.5	12.5	0.0	37.5	0.0	12.5
$> 3^{rd}$	87.5	87.5	70.0	100	75.0	35.0	100	27.5	85.0	2.5
Failures	20.0	35.0	**0.0**	7.5	5.0	2.5	20.0	2.5	2.5	**0.0**
	Real image data-set (23 images)									
1^{st}	13.0	13.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**73.9**
2^{nd}	8.7	13.0	26.1	4.3	13.0	17.4	0.0	26.1	13.0	4.3
3^{rd}	0.0	0.0	21.7	0.0	21.7	17.4	4.3	17.4	13.0	21.7
$> 3^{rd}$	78.3	73.9	52.2	95.7	65.2	65.2	95.7	56.5	73.9	0.0
Failures	4.3	**0.0**	4.3	8.7	13.0	4.3	13.0	8.7	13.0	**0.0**

Table 6.13: Evaluation of the performance of the Algorithms according to the **S-CIELAB** score. Rows represent: the % of images where the algorithm achieved the best (1^{st}), second best (2^{nd}), third best (3^{rd}), or worse ($> 3^{rd}$) score. The last row shows the % of times an algorithm failed to color correct the image. Best results are highlighted in blue.

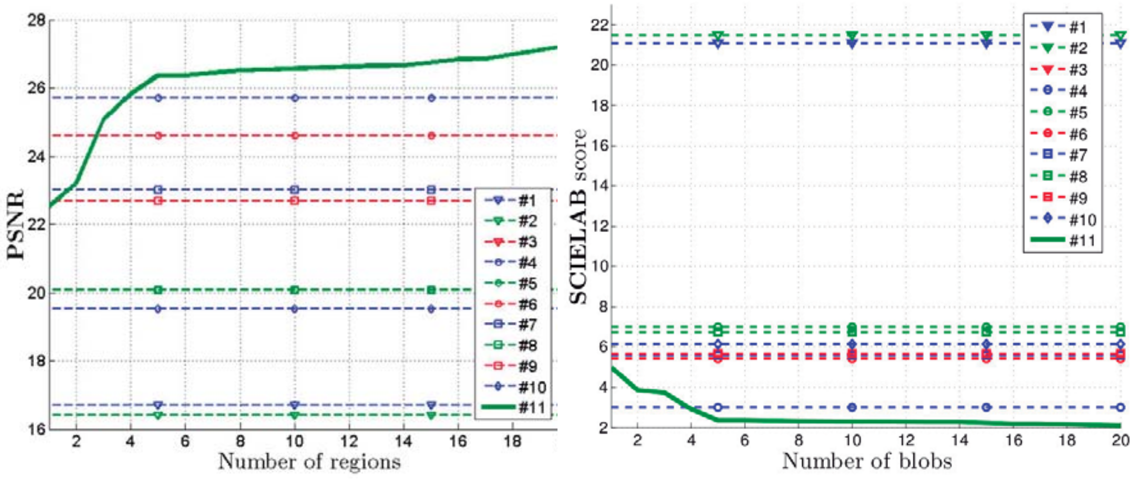
	S-CIELAB color dissimilarity									
	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
	Synthetic image data-set (40 images)									
1^{st}	0.0	2.5	5.0	0.0	15.0	30.0	0.0	17.5	0.0	**80.0**
2^{nd}	0.0	0.0	20.0	0.0	0.0	45.0	0.0	27.5	2.5	5.0
3^{rd}	0.0	0.0	7.5	0.0	5.0	17.5	0.0	35.0	2.5	7.5
$> 3^{rd}$	100	97.5	67.5	100	80.0	7.5	100	20.0	95.0	7.5
Failures	27.5	55.0	**0.0**	2.5	2.5	**0.0**	15.0	**0.0**	**0.0**	**0.0**
	Real image data-set (23 images)									
1^{st}	4.3	8.7	0.0	0.0	0.0	17.4	0.0	4.3	0.0	**87.0**
2^{nd}	0.0	0.0	13.0	0.0	0.0	52.2	8.7	26.1	0.0	4.3
3^{rd}	0.0	0.0	13.0	0.0	0.0	13.0	0.0	43.5	8.7	4.3
$> 3^{rd}$	95.7	91.3	73.9	100	100	17.4	91.3	26.1	91.3	4.3
Failures	17.4	13.0	**0.0**	**0.0**	13.0	**0.0**	8.7	4.3	8.7	**0.0**

S-CIELAB, real **PSNR** and real **S-CIELAB** cases. In sum, the proposed approach shows a very high consistency in scoring amongst the top three color correction algorithms. Tables 6.12 and 6.13, also show the percentage of failures for each algorithm. A failure occurs when an algorithm fails to color correct the target image. This is ascertained when the score obtained by the algorithm is worse (lower in **PSNR** or higher for **S-CIELAB**) than the score obtained by the baseline approach, algorithm #1. In other words, a failure occurs if an algorithm outputs a corrected image that is photometrically more distant to the source image than the original uncorrected target image. The proposed approach shows, in all cases, zero failures, which is a unique performance compared to other algorithms. This shows that the proposed approach is the most consistent and very robust.

6.5.8 Analysis of the effect of Mean shift to the overall Color Correction

As described in section 6.2, several color correction approaches make use of local color transfer methodologies. It was shown that they are more effective that global approaches. The color correction algorithm proposed in this chapter can also be viewed as a local color correction methodology, although it also works if no color segmentation preprocessing step is done. In this section an evaluation of the effect of color segmentation to the effectiveness of the proposed approach is presented. The parameters of the Mean shift were changed, so that, in different tests, the image is segmented into a distinct number of regions. Figure 6.20 shows the **PSNR** and **S-CIELAB** scores of the proposed approach in the different tests, i.e., with the Mean shift preprocessing step configured to have different number of blobs. Results refer to the overlapping area of the image presented in the case study, section 6.5.6. For comparison, the scores of the other algorithms used in this evaluation are also displayed. They are independent of the number of regions provided by Mean shift, which is why they show constant scores.

From the analysis of Fig. 6.20 it is possible to realize that the proposed approach achieves the best score when five or more regions are segmented by Mean shift, in the case of the **PSNR** score (Fig. 6.20 (*a*)), or when four or more regions are segmented in the case of the **S-CIELAB** score (Fig. 6.20 (*b*)). The effectiveness of the proposed algorithm clearly benefits from using a preprocessing color segmentation step. As expected, the effectiveness of color correction improves as the segmentation shifts from coarse to fine. However, there seems to be a saturation point (six regions, in the case of this image), where a larger number of regions produces only a very small improvement on color correction effectiveness. This is an expected behavior: as the size of the regions decreases, the color uniformity of the pixels in the region should contribute for a better estimation of the color palette mapping function, but in the opposite direction, the smaller number of pixels in the region also means that the statistical fitting of truncated Gaussians will be less effective and more prone to noise. Another interesting conclusion is that the proposed approach yields reasonable scores even when no preprocessing color segmentation is performed, i.e., when the number of regions in Fig. 6.20 is equal to one.



(a) *(b)*

Figure 6.20: Analysis of the effect of the color segmentation output to the effectiveness of the color correction performed by the proposed approach. (*a*) **PSNR** scores; (*a*) **S-CIELAB** scores.

6.6 Conclusions

This chapter proposed three alternative approaches to tackle the problem of color correction between images.

The first approach (section 6.3) uses a Mean shift color segmentation approach to perform local color correction. It was shown that due to the robustness of the Mean shift color segmentation stage, the proposed approach achieves better quality mosaics when compared to other local approaches.

The second approach (section 6.4) uses 3DGMM to model the color distribution in the images. The algorithm consists of performing a single step multi dimensional probabilistic segmentation of the three color channels of an image in order to perform color correction. A recent performance evaluation on color correction was used to adequately select other color correction methods and an evaluation metric. The joint segmentation of the three channel color was shown to reduce processing time from similar single channel methods: 4.1 seconds average processing time of the proposed approach versus 4.48 seconds from [Tai *et al.* 2005]. The proposed approach obtained very good average CS scores, which makes it a technique to take into account for devising color correction algorithms. In automotive applications, real time color correction would not be possible using this methodology, but if it is important to obtain a high color similarity score (that is, a very accurate color correction) , a strategy where a color palette mapping is built every four seconds could be devised. Results show that 3DGMM may be successfully applied to color correction in the context of multi-camera onboard systems, since it shows good results in the evaluation parameters and is faster to process than similar methods.

The third approach (section 6.5) proposes a probabilistic modelling of the joint image histogram using truncated Gaussians. Images are color segmented using Mean shift. Each color segmented region is then used to compute a local color palette mapping function, by fitting a set of truncated Gaussians to the observable mutual information. Finally, using an extension of the color palette mapping functions to the whole image, it is possible to produce mosaics where no color transitions are noticeable. For the proper assessment of the performance of the proposed algorithm, nine other color correction algorithms were evaluated. Each of the algorithms was applied to two data-sets of source target image pairs, with a total of 63 image pairs. Two different evaluation metrics were tested. Results show that the proposed approach outperforms all other algorithms, in most of the image pairs in the data-sets, using both evaluation metrics. Not only it obtains the best average scores but also shows to be more consistent and robust. Furthermore, the proposed algorithm works better in the harder data-set, which seems to lead to the conclusion that it is less sensitive to miss registrations between the source and target image pairs. Finally, although the usage of Mean shift improves the performance of the proposed approach, results have shown that the proposed approach achieves very good results even if no color segmentation preprocessing step is used.

Chapter 7

Data Sets and Preprocessing

This chapter introduces some data sets collected by autonomous vehicle platforms. Special emphasis is given to a detailed description of the nature and size of these data sets. Due to the large ammount of sensors onboard autonomous vehicles, the size of the data that needs to be processed is very large. A second part of this chapter presents 3D preprocessing algorithms, in particular those used for reducing the size of those data sets. The chapter begins with a small introduction to the problem (section 7.1) and a description of the related work (section 7.2). Then, in section 7.3, the Massachusetts Institute of Technology (MIT) Defense Advanced Research Projects Agency (DARPA) Urban Challenge data set is presented. Section 7.4, describes preprocessing algorithms suited for reducing the size of 3D data. Finally, conclusions are drawn in section 7.5.

7.1 Introduction

Chapters 8, 9, 10 and 11 will present in detail the algorithms used for creating a scene representation from 3D point cloud data. Before this, it is important to characterize the data produced by standard autonomous vehicle platforms. This chapter provides a description of the nature and size of the data produced both by the *AtlasCar* and the MIT *Talos* vehicles. Then, it introduces the data set which will be used to validate the 3D processing algorithm that will be proposed. The data sets from MIT are divided in two sequences. Each sequence is in turn marked by a number of locations. The second part of this chapter introduces some basic, 3D processing algorithms. These are conceptually simple and fast to process. Because of this they are referred to as preprocessing algorithms for 3D point clouds.

7.2 Related Work

Available 3D point clouds data sets collected from vehicles are scarce. The expensive price of acquisition platforms, as well as the lack of a standard format for the data hampers the appearance of an universal data set. However, recent years have witnessed the addition of several new data sets.

Table 7.1: 3D point cloud data sets available. Use soft copy for access to the hyper links.

Name	Institution	Laser	3D Laser	Video	Egomotion	Size
Victoria Park	University of Sidney $^{(1)}$	yes	no	yes	yes	>100MB
Radish	Several Institutions $^{(2)}$	yes	no	no	no	>1GB
Rawseeds	Albert-Ludwigs University $^{(3)}$	yes	no	no	yes	>1GB
New College	Oxford University $^{(4)}$	yes	no	yes	yes	>20GB
AtlasCar	University of Aveiro $^{(5)}$	yes	yes	yes	no	>20GB
Enpeda	University of Auckland $^{(6)}$	no	no	yes	no	>100MB
Malaga data set	University of Malaga $^{(7)}$	yes	no	yes	yes	>20GB
Karlsruhe	Karlsruhe Institute of Technology $^{(8)}$	no	no	yes	yes	>20BG
MIT Darpa	Massachusetts Institute of Technology$^{(9)}$	yes	yes	yes	yes	>200GB
Cheddae George	BAE Systems$^{(10)}$	yes	yes	yes	yes	>300GB

$^{(1)}$ Used by [Lina María Paz & Neira 2007].
$^{(2)}$ Described in [Howard & Roy 2003].
$^{(3)}$ Used in [Javier Civera & Montiel 2009], [Neira & Trinkle 2009], [Lina María Paz & Neira 2008] and [Piniés & Tardós 2008].
$^{(4)}$ Described in [Smith *et al.* 2009].
$^{(5)}$ Described in [Santos *et al.* 2010].
$^{(6)}$ Described in [Hermann *et al.* 2011] and [Schauwecker *et al.* 2011].
$^{(7)}$ Described in [Blanco *et al.* 2009].
$^{(8)}$ Described in [Geiger *et al.* 2012].
$^{(9)}$ Described in [Huang *et al.* 2011].
$^{(10)}$ Described in [Simpson *et al.* 2011]. Not available for free.

Table 7.1 lists some of the 3D data sets available nowadays. The table also indicates if the data sets have planar laser data, coming from a single or several LRF, 3D laser data, from rotating 3D laser sensors, video data, which includes both mono and stereo cameras, and egomotion data. In this case, egomotion data is of vital importance for the selection of a suited data set. The reason is that the approaches that will be proposed require periodic information about the 6DOF position of the vehicle with respect to a fixed world coordinate frame. Egomotion is typically computed at very high frequencies (100Hz) in comparison to other sensor data. It is computed using GPS, inertial measurement units, odometry or a combination of these. From the analysis of Table 7.1, the MIT data set seems to e one of the best data sets available. In fact, the MIT data set will be used to validate the algorithms that will be proposed to achieve scene reconstruction.

In this section, no review of point cloud preprocessing algorithms is made since these are generally simple algorithms, which will be described in section 7.4.

7.3 Data Sets

In order to evaluate the proposed 3D processing techniques a complete dataset both with 3D laser data, cameras and accurate egomotion is required. The MIT autonomous vehicle *Talos* competed in the Darpa Urban Challenge and achieved fourth overall place. The data logged by the robot is

Table 7.2: Several information on each of the logs.

	Size			
Description	Primary log	Camera log	Duration (sec)	Distance (km)
Sample log	83MB	none	10	-
Darpa finals mission 1	74GB	18GB	10947	30.0
Darpa finals mission 2	38GB	9.5GB	5428	21.5
Darpa finals mission 3	72GB	18GB	10414	41.2

publicly available [Huang *et al.* 2011]. The logs contain three missions recorded during the Darpa Urban Challenge competition plus a sample log. For each mission, the primary log contains data from all sensors although the five camera images are recorded with a 376×240 resolution, while in the camera logs the images are recorded with full resolution, that is 752×480. In the following tests, only the primary logs are used. It is assumed that the image resolution contained in the primary logs is sufficient. In total, the MIT logs sum up to 315GB of data. Table 7.2 displays information on each of the logs. Primary MIT log files are stored in the Lightweight Communications and Marshalling (LCM) log file format (see section 4.2.1 for further details). Camera log files are provided in two formats. The first is the *Camunits* [Huang 2012] log file format, and can be viewed using *Camview*, a visualisation software from *Camunits*. The second is a TAR archive of JPEG files. Conceptually, an LCM Log file is an ordered list of events. Each event has four fields: event number, monotonically increasing 64-bit integer that identifies each event. It should start at zero, and increase in increments of one; time stamp, monotonically increasing 64-bit integer that identifies the number of microseconds since the epoch (00:00:00 UTC on January 1, 1970) at which the event was received; channel, UTF-8 string identifying the LCM channel on which the message was received; and data, binary blob consisting of the exact message received. Each event is encoded as a binary structure consisting of a header, followed by the channel and the data. A software that reads specified portions of the MIT log files and writes a corresponding Robot Operating System (ROS) bag file was developed. In this way, it is possible to use the MIT Darpa dataset in the ROS software infrastructure. At the moment, only some of the sensor messages available in the MIT dataset, i.e., some event channels, are converted to the bag file. Nonetheless, the infrastructure is developed so it will be simple to add the missing channels, in the case they are necessary. For our tests in 3D processing and representation only some of the channels were required.

Table 7.3 lists all the event channels contained by the MIT logs and indicates if they are included in the conversion to bag tool. There are several event channels, including the five on-board cameras, vehicle egomotion estimation using an Applanix POS-LV 220 system which includes a GPS, an inertial measurement unit and a wheel encoder [Applanix 2012], 12 laser range finders and one Velodyne HDL64 lidar [Velodyne 2012].

In the work presented in this chapter, related to 3D processing, the option was to use the MIT data instead of the data provided by the *AtlasCar*. There are advantages and disadvantages on using either

Table 7.3: Event channels on the MIT logs and whether they are converted to ROS bag files or not.

Event channel	Description	Frequency (Hz)	Converted to ROS bag
POSE	Vehicle pose (local frame)	100	yes
GPS TO LOCAL	Vehicle pose (GPS)	100	no
CAM THUMB RFC	Camera front wide6 , low-res	10	yes
CAM THUMB RFC6mm	Camera front narrow 6mm low-res	10	yes
CAM THUMB RFR	Camera left, low-res	10	yes
CAM THUMB RFL	Camera right, low-res	10	yes
CAM THUMB RFR6	Camera rear, low-res	10	yes
BROOM L SICK	pushbroom left	75	no
BROOM CL SICK	pushbroom left-center	75	no
BROOM C SICK	pushbroom center	75	no
BROOM CR SICK	pushbroom right-center	75	no
BROOM R SICK	pushbroom right	75	no
SKIRT FL	SICK skirt front-left	75	no
SKIRT FC	SICK skirt front-center	75	no
SKIRT FR	SICK skirt front-right	75	no
SKIRT RC	HI SICK skirt rear-high	75	no
SKIRT RC	LO SICK skirt rear-low	75	no
VELODYNE	Velodyne	15	yes

one or the other, but those suggesting the usage of the MIT dataset are more relevant, namely those related to:

- number of sensors
- size of the 3D point clouds
- quality of the 3D point clouds
- vehicle egomotion

The MIT data set has an incomparably larger amount of sensors, when compared to those onboard the *AtlasCar*: while the *Talos* vehicle has 12 laser range finders, the *AtlasCar* has only four; the Talos has five cameras, the *AtlasCar* has only three; and above all the *MIT* has a Velodyne 3D laser. Because of this, the *Talos* as a much better laser coverage all around the vehicle when compared to the *AtlasCar*.

With respect to the size and quality of the 3D point clouds provided by the vehicles, the larger number of sensors in the *Talos* will obviously produce a much larger number of range measurements. However, some other factors also affect the size of the 3D point clouds, namely, the Velodyne sensor. Table 7.4 provides an analysis of the size of the point clouds provided by both vehicles. It shows that the *Talos* provides about 1.4 million points per second and that the *AtlasCar* provides approximately half of this number, about 650 thousand points per second. However, the great part of the *AtlasCar* 3D points is provided by the stereo cameras, which are much less accurate than lasers. Also, the XB3 has a limited field of view, which means that the 3D range measurements are not evenly distributed

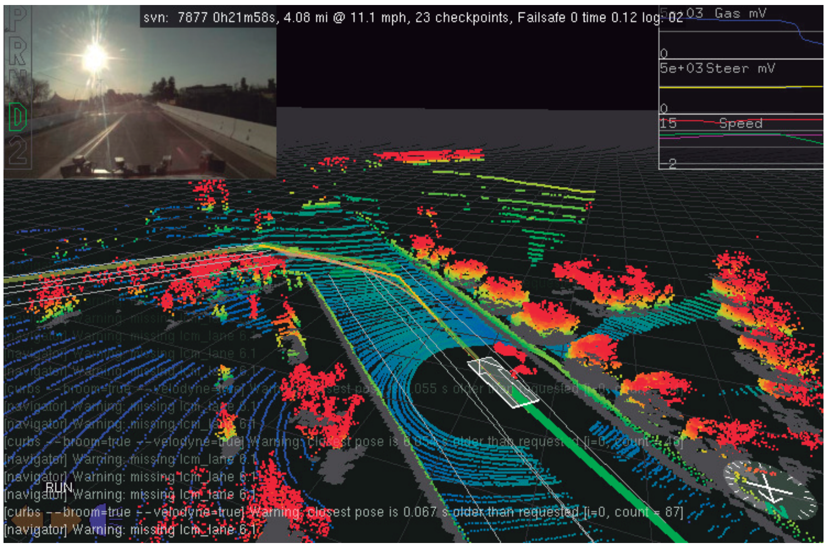
Table 7.4: Comparison of the size of the 3D point clouds provided by the *AtlasCar* and the *Talos* vehicles.

	Talos				*AtlasCar*			
sensor	num	fov	freq.(Hz)	p/sec($\times 10^3$)	num	fov	freq. (Hz)	p/sec ($\times 10^3$)
Sick LMS 291	12	180	75	162	0	-	-	-
Sick LMS 200	0	-	-	-	1	180	40	7.2
Hokuyo UTM30LX	0	-	-	-	1	270	40	10.8
Sick LMS 151	0	-	-	-	2	270	50	27
XB3	0	-	-	-	2[1]	30720[2]	10	601.4
Velodyne HDL64	1	1	1	1300	0	-	-	-
Total	-	-	-	**1462**	-	-	-	**646.4**

[1] Since the Point Grey Research Bumblebee XB3 Stereo Camera (XB3) has three cameras, two different stereo pairs are used.
[2] Although the images used for stereo are 320×240, which would give 76800 pixels and corresponding range points, the fact is that stereo requires texture uniqueness to be able to correspond pixels from the two cameras. In outdoor scenes of roads, typically only 40% of the total pixels are matched, which results in $76800 \times 0.4 = 30720$ 3D points per measurement.

all around the vehicle, as is the case of the *Talos*. In conclusion, the 3D point clouds provided by the *Talos* do have a better quality, which is one of the core reasons we decided to use the MIT dataset for assessing the proposed 3D reconstruction algorithms.

Finally, another key point that led us to use the MIT dataset was the vehicle egomotion. At the time of this decision, the *AtlasCar* did not have an egomotion functionality. Without an egomotion estimation, when the vehicle is moving, it is much harder to fuse or transform 3D point clouds captured



```
svn:  7877 0h21m58s, 4.08 mi @ 11.1 mph, 23 checkpoints, Failsafe 0 time 0.12 log: 02
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[curbs --broom=true --velodyne=true] Warning: closest pose is 0.055 s older than requested [i=0, count = [illegible]]
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[curbs --broom=true --velodyne=true] Warning: closest pose is 0.054 s older than requested [i=0, count = 46]
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[curbs --broom=true --velodyne=true] Warning: closest pose is 0.067 s older than requested [i=0, count = 87]
[navigator] Warning: missing lcm_lane 6.1
```

Figure 7.1: A snapshot of the MIT viewer application showing data from a log.

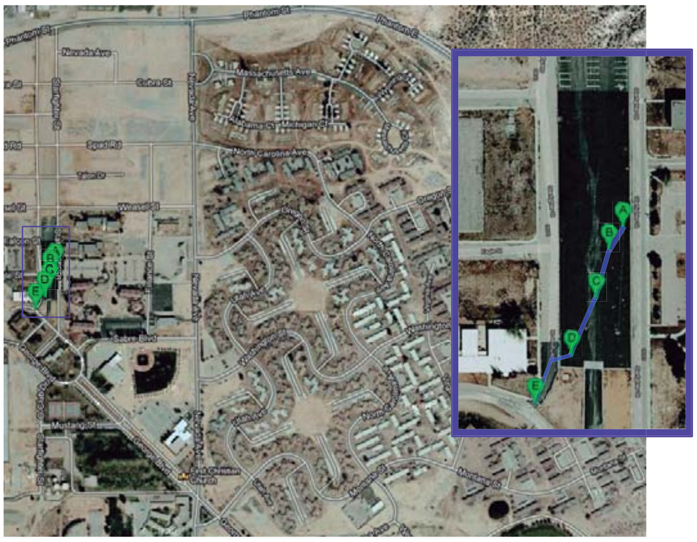


Figure 7.2: Path travelled by the robot during sequence 1. Key locations are annotated both in the map and the zoomed in image.

at different times to a single coordinate frame. In other words, if no egomotion estimation is available, it is not possible to perform analysis on 3D point clouds that were taken in different positions of the vehicle, since no information exists that relates these different positions. On the other hand, the *Talos* vehicle uses an *Applanix* system that estimates at a frequency of 100Hz the pose of the vehicle with respect to the world. This equipment uses a wheel encoder, an inertial measurement unit and a GPS to perform the estimation. The estimates are very accurate. The only downside is that these equipments

Table 7.5: Information on each of the locations in this sequence. Columns description: pt, number of points; size, memory size in mega bytes; t, mission time in seconds; d, traveled distance in meters.

Location		Location Snapshot		Sequence accumulated			
Name	Fig. (pag#)	pt ($\times 10^6$)	size (MB) [(1)]	pt ($\times 10^6$)	size (MB) [(1)]	t (s)	d (m)
A	A.1 (407)	1.3	15.6	1.3	15.6	1	0
B	A.2 (408)	1.3	15.6	13.0	156.0	11	75
C	A.3 (408)	1.3	15.6	26.0	312.0	21	125
D	A.4 (409)	1.3	15.6	39.0	468.0	31	140
E	A.5 (409)	1.3	15.6	52.0	624.0	41	190

[(1)] Computed from the number of points times the three xyz dimensions times the four bytes for each dimension (type *float32*). It is an approximate value since there are other informations on the message, such as the time stamp, the coordinate frame identification, etc.

cost about 30K Euros. Figure 7.1 shows a snapshot of the log, using the viewer application provided by MIT.

Using the MIT data sets and the bag conversion tool, several bag files where created. Each bag file represents a multi sensor sequence that will be used to assess and demonstrate the proposed algorithms. The two sequences will be described in detail in the following lines. In appendices A and B, examples of several locations in those sequences are shown. It is important to have a feeling of the environment the vehicle travels in these sequences, since it will be referred multiple times in the following chapters.

The first sequence is a small, forty second long sequence. The robot travelled 190 meters in total. It corresponds to the beginning of the first mission, and the robot is navigating through the paddock of the competition. Only the Velodyne 3D data is stored, Laser Range Finders (LRFs) are discarded. Five key locations are considered in this sequence: A, B, C, D and E. Figure 7.2 shows a map with the travelled path annotated. Table 7.5 displays some information on each of the locations of the sequence, including the number of points and memory size of each location, as well as accumulated throughout the sequence. In total, the sequence is 624 MB, considering 3D data only. If images are taken into account, the size sums up to tenths of gigabytes. Considering the relatively short forty second duration, the amount of information that is generated is immense. This shows the importance

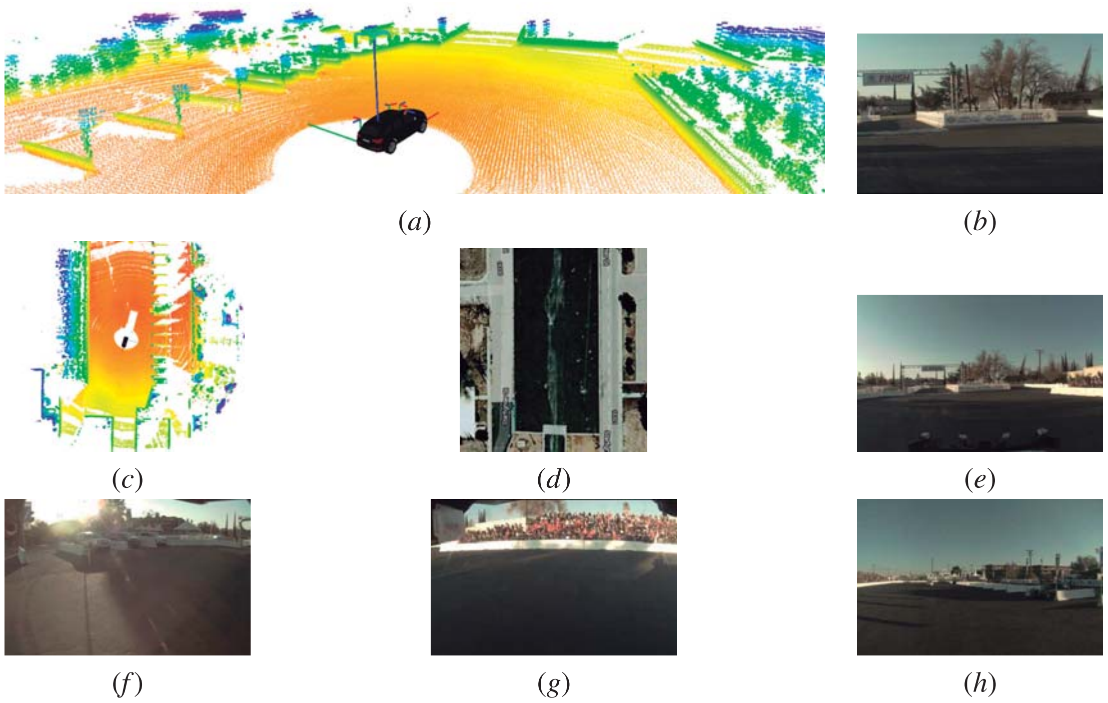


Figure 7.3: Location C of the sequence. Isometric (a) and bird's eye (c) view of the 3D data; front 6mm (b), front (e), rear (h), left (f) and right (g) camera images; (e) satellite view of the location.

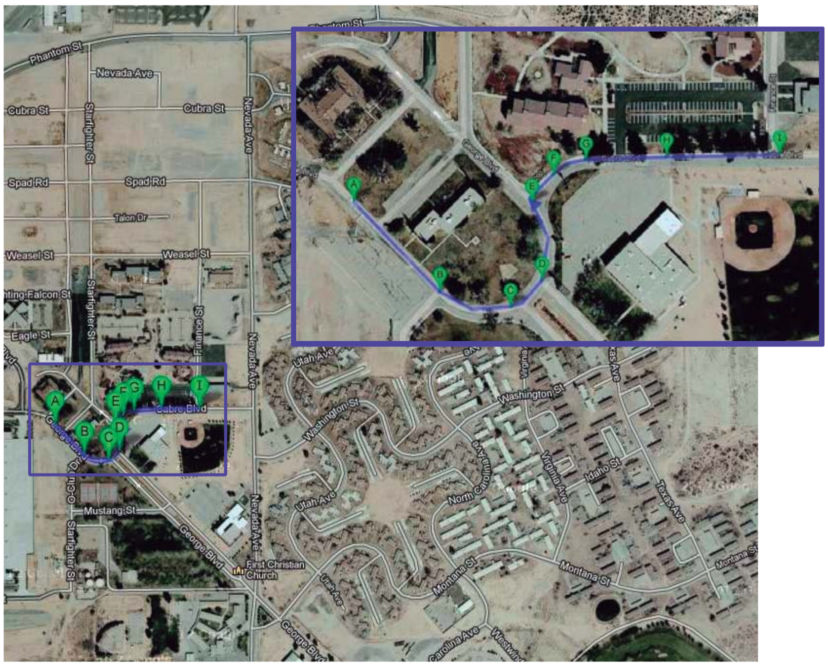


Figure 7.4: Path travelled by the robot during sequence 2. Key locations are annotated both in the map and the zoomed in image.

of researching effective data compression algorithms. Figure 7.3 shows images from all cameras, isometric and top views of the 3D data, and a satellite photograph of location *C*. In Appendix A, all locations are shown in detail.

The second sequence is larger than the first. It is 80 seconds long, and the distance travelled was 340 meters. Instead of the paddock, in this case, the robot navigates through streets in the DARPA scenario. As in sequence 1, only the Velodyne 3D data is stored. Nine key locations are considered in this sequence: *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H* and *I*. Figure 7.4 shows a map with each location annotated. Table 7.6 displays some information on each of the locations. Figure 7.5 shows an example of a location in this sequence. Appendix B provides detailed information on each of the locations.

Figure 7.6 (*a*) shows a photograph of the *Talos*. In Fig. 7.6 (*b*), a 3D model of the vehicle that will be used to show several results is shown.

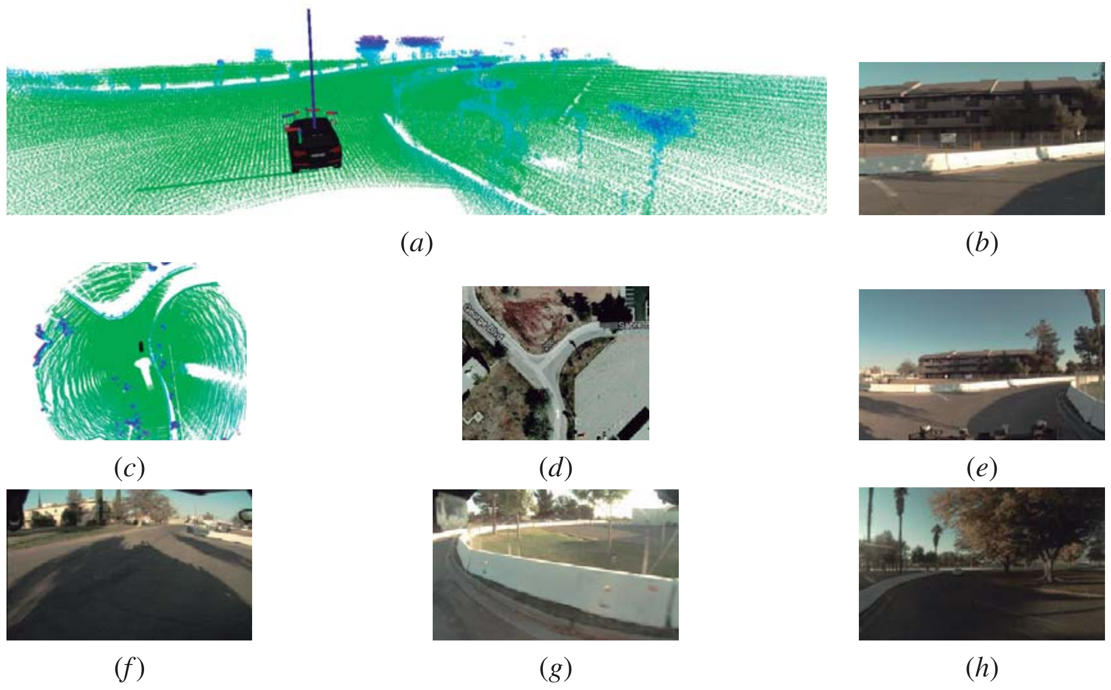


Figure 7.5: Location *E* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

Table 7.6: Information on each of the locations in this sequence. Columns description: pt, number of points after voxel grid filtering; size, memory size in mega bytes; ratio, compression ratio obtained by the voxel grid filter; t, mission time in seconds; d, traveled distance in meters.

Location		Location Snapshot		Sequence Accumulated			
Name	Fig. (pag#)	pt ($\times 10^6$)	size (MB)	pt ($\times 10^6$)	size (MB)	t (s)	d (m)
A	B.1 (411)	1.3	15.6	1.3	15.6	1	0
B	B.2 (412)	1.3	15.6	13.0	156.0	11	75
C	B.3 (412)	1.3	15.6	26.0	312.0	21	125
D	B.4 (413)	1.3	15.6	39.0	468.0	31	140
E	B.5 (413)	1.3	15.6	52.0	624.0	41	190
F	B.6 (414)	1.3	15.6	65.0	780.0	51	210
G	B.7 (414)	1.3	15.6	78.0	926.0	61	230
H	B.8 (415)	1.3	15.6	91.0	1092.0	71	280
I	B.9 (415)	1.3	15.6	104.0	1248.0	81	340

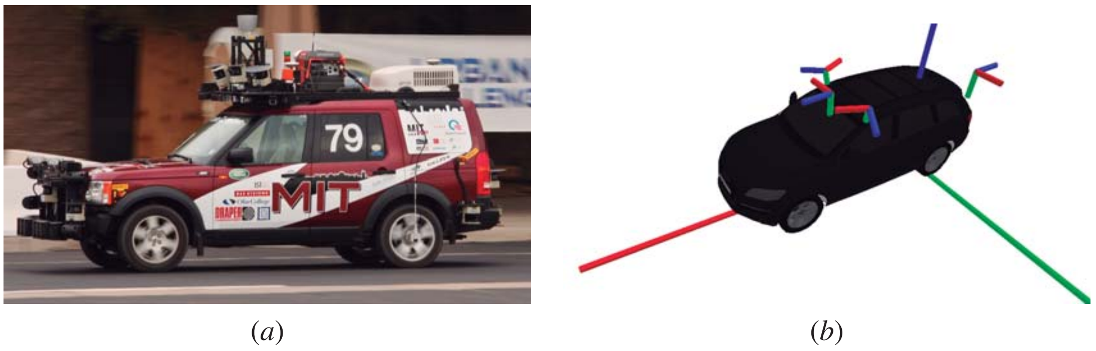


Figure 7.6: The *Talos* MIT autonomous vehicle. (*a*) a picture of the car; (*b*) the 3D model.

7.4 Preprocessing of 3D point clouds

From the amount of data generated every second by the *Talos*, presented in Table 7.4, it is possible to ascertain that some kind of data reduction mechanism must exist in order to compress the size of the point clouds. The large size of the MIT sequences also points to this conclusion. However, as pointed out in section 7.3, the solution cannot be to use a smaller amount of data as in the case of the *AtlasCar*, since the quality of the data is not adequate for 3D reconstruction. On the one hand it is required that the amount of 3D data is large enough to ensure a 360 degrees field of view and adequate spatial resolution, but, on the other hand, the large size of the data also makes it unsuitable for processing in real time. Therefore, the conclusion is that the amount of raw 3D data generated by a vehicle must be similar to *Talos*, since the data provided by the *AtlasCar* is not sufficient and does not yet have enough quality, but also that there must be a very efficient way to compress and discard some portion of the data. The choice of which data to discard and which to keep will of course depend on the application, computational power and other factors. In this work, the focus was to design 3D data processing algorithms as general and flexible as possible, not having any single application in mind, but rather to try to compress the data in a *lossless* way or with an intelligent filter for most of the typical subsequent applications, like, for example, obstacle detection, environment reconstruction, motion planning.

The following sections present some algorithms used to preprocess the input 3D point clouds.

7.4.1 Local and Spatial point filters

This section presents a very simple and fast mechanism to filter 3D data and significantly compress the data. It is composed by two filters: a local sensor filter and a spatial resolution filter. The filters assert the point cloud against a given criteria and discard points that do not comply with that criteria. These algorithms are meant to be used as a preprocessing stage.

A local sensor filter is proposed based on the maximum and minimum measured distance. Let $\mathcal{P}$

represent a point cloud. The point cloud is a list of n 3D points p_j, where j is the index of the point:

$$\mathcal{P} = [\mathrm{p}_0,\mathrm{p}_1...\mathrm{p}_n] = \begin{bmatrix} x_0 & x_1 & ... & x_n \\ y_0 & y_1 & ... & y_n \\ z_0 & z_1 & ... & z_n \end{bmatrix}. \tag{7.1}$$

The point clouds may also be represented in spherical coordinates, turning eq. 7.1 into:

$$\mathcal{P} = \begin{bmatrix} \rho_0 & \rho_1 & ... & \rho_n \\ \theta_0 & \theta_1 & ... & \theta_n \\ \phi_0 & \phi_1 & ... & \phi_n \end{bmatrix}, \tag{7.2}$$

where spherical coordinates variables ρ, θ and ϕ can be obtained with the typical transformation from Cartesian to spherical coordinates. The local sensor filter simply imposes a range of admissible ρ values (ρ_{min} and ρ_{max}) for a point to be kept. The remanding point cloud $\mathcal{P}^*$ is obtained using the following expression:

$$\mathcal{P}^* = \{\mathrm{p}_j \in \mathcal{P} \mid \rho_{min} < \rho_j < \rho_{max}, \quad \forall j \in \{0, 1, ..., n\} \quad \}. \tag{7.3}$$

The local nature of the filter implies that it must be computed in the 3D sensor local reference system. Let ${}^{\mathbf{v},t_j}\mathrm{p}_j$ represent the j^{th} point given by a range measurement taken from sensor coordinate frame **v** at time t_j. Since the sensor coordinate frame moves along with the vehicle then $\mathbf{v} = f(t_j)$. This is why the filtering must be done immediately after collecting new 3D data to ensure that all points have the same or very similar acquisition time t:

$$t_j \simeq t_{j+1}, \quad \forall j \in \{0, 1, ..., n-1\}. \tag{7.4}$$

If this is not the case, the information of where the sensor coordinate system was at for each point in the corresponding time of acquisition must be kept, which is not feasible because of memory restrictions. Empirical tests have shown that for normal vehicle speeds, the local sensor filter may be applied to Velodyne data at a frequency of 1 Hz. Hence, in the presented results, the maximum boundary can be formulated as:

$$\mathbf{max}(|t_j - t_k|) < 1, \quad \forall j, k \in \{0, 1, ..., n\}. \tag{7.5}$$

Figure 7.7 shows an example of the local sensor filter: all acquired points at a given time are plotted in blue. Points that are kept after applying the local filter are shown in green. It is possible to see that points that are far away from the vehicle are discarded, as well as those that are very near to the vehicle. Points very near to the vehicle are actually range measurements of the vehicle body. Figure 7.7 (*b*) shows several measurements on top of the vehicle that correspond to this case.

The second step of the preprocessing algorithm is the spatial resolution filter. The filter consists of

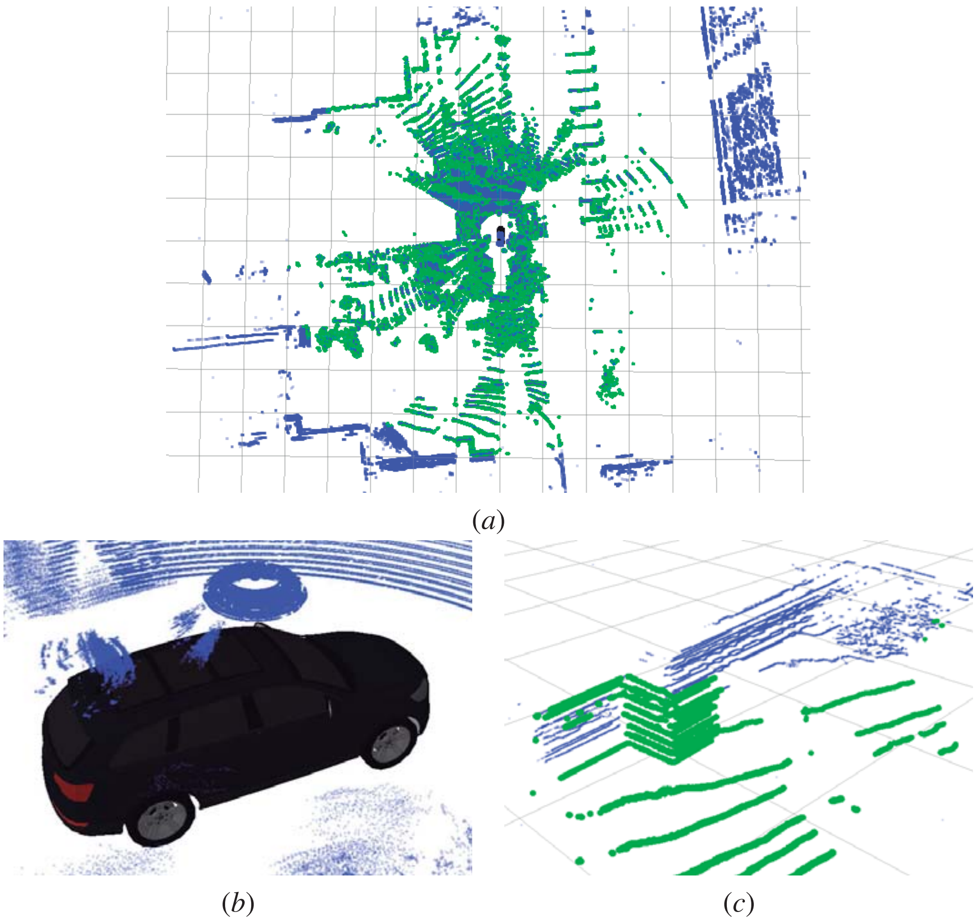


Figure 7.7: The local sensor filter. Blue points are all the acquired points, green points are the points kept by the filter. (*a*) a top view of the scene; (*b*) a detail of the points captured that belong to the vehicle; (*c*) a detail of a wall close to the ρ_{max} threshold.

dividing the 3D space into a grid, from which parallelepipeds are generated. Only a single 3D Point is stored inside each parallelepiped. This operation is commonly referred to as voxel filter. Figure 7.8 shows an example of the 3D grid as well as results obtained using this filter. It is possible to see that the kept points (in red) have a coarser spatial resolution (which was configured in the filter) when compared to the input data (in green).

The filter enables the control of the spatial resolution of the point cloud. The height, width and length of the parallelepipeds can be defined independently. Let **f** be a function that retrieves the index of the cube where a given point lies in. The spatial resolution filter can be defined as:

$$\mathcal{P}^* = \Big\{ \mathrm{p}_j \in \mathcal{P} \mid \big(\nexists\, k: \; k < j \;\wedge\; \mathbf{f}(\mathrm{p}_j) = \mathbf{f}(\mathrm{p}_k), \quad \forall j \in \{0, 1, ..., n\}\big) \Big\}, \tag{7.6}$$

where $\mathcal{P}^*$ is the remanding point cloud. Using eq. 7.6 it is possible to further reduce the size of the raw

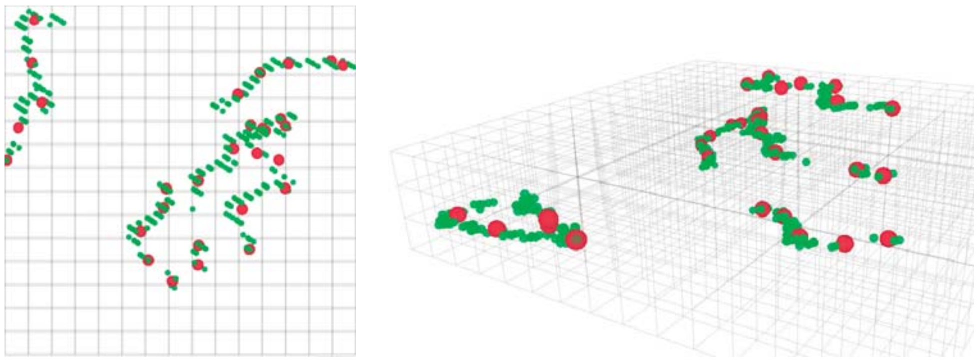
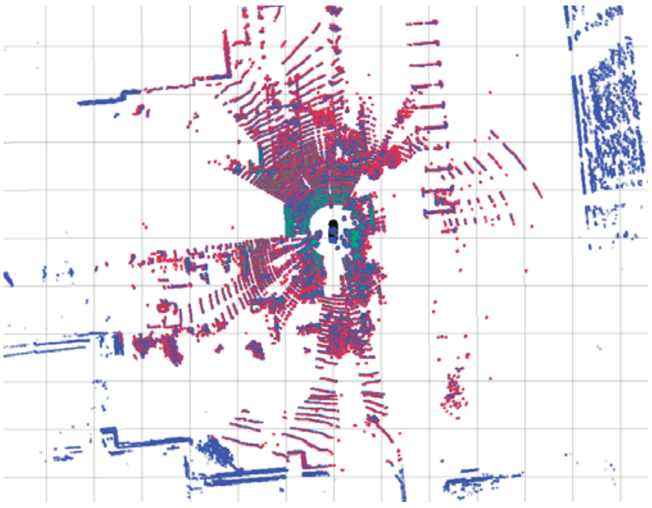

(*a*) (*b*)

Figure 7.8: The input point cloud (green points) is reduced using the from the spatial resolution filter. The 3D grid is shows b grey lines, while the kept points are signaled in red. *(a)* top view; *(b)* isometric view.

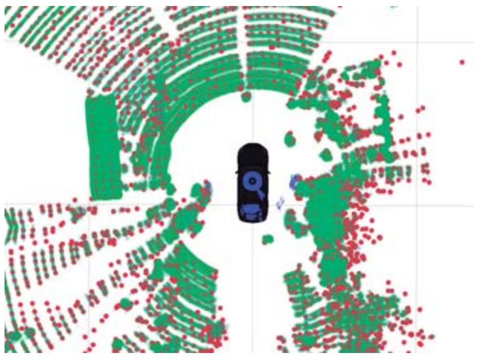
point clouds. The spatial resolution filter is applied only to the points that are kept by the local sensor filter. Hence a cascade configuration is adopted to setup both filters: first the local sensor and after the spatial resolution. This configuration speeds up processing time. We refer to the preprocessing of point clouds as the application of these two filters in a cascade like configuration to the raw point clouds. Figure 7.9 shows the results obtained after pre processing a Velodyne point cloud.

The presented preprocessing algorithm was applied to MIT sequences 1 and 2. Results are shown in Tables 7.7 and 7.8, for sequences 1 and 2, respectively. The parameters that were used are $\rho_{max} = 50$ and $\rho_{min} = 3$ meters with respect to the local sensor filter, and to a 3D grid size for the spatial resolution filter of 0.2, 0.2 and 0.01 meters. The values for these were obtained by empirical trial and error methodology, balancing the quality of the filtered point cloud with the compression ratio.

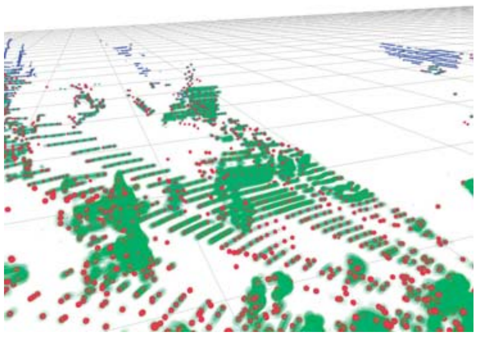
Table 7.7 shows that the preprocessing of point clouds is able to discard a large number of points. Typically, around 350K points are kept, which means that about 950K are discarded (see Table 7.5 for information on the raw point clouds). In terms of compression ratio, the typical value is around 0.25, both for the snapshot analysis of the locations and well as the accumulated sequence. The total memory size required to store the full pre processed sequence is 169.7MB (14.1$\times10^6$ points), compared to the 624MB (52.0$\times10^6$ points) for the raw data. The same conclusions can be taken for sequence 2 by the observation of Table 7.8. The average compression ratio is around 0.27, and the full sequence raw 104.0 $\times10^6$ points are compresses to total of 29.2 $\times10^6$ points. In total, the preprocessed sequence 2 has a size of 350.9 MB. In conclusion, the amount of data reduction provided by the filter is quite reasonable, especially taking into account the fact that there is no significant loss in the quality of the point cloud. This conclusion is drawn based on the observation of the output point clouds. As will be seen is subsequent sections, the preprocessed point cloud is enough for computing an accurate environment reconstruction.



(*a*)



(*b*)



(*c*)

Figure 7.9: Results from 3D point cloud preprocessing algorithm. In blue the raw 3D data, in green the point cloud after the local sensor filter, in red the point cloud after the spatial resolution filter.

Figure 7.10 compares, for both sequences, the raw and preprocessed sizes of the point clouds over time. From its observation it is possible to see that the preprocessing step here proposed dramatically decreases the size of the point clouds. Although from the observation of Fig. 7.10 the compression ratio seems constant, it is in fact environment dependent. In other words, the amount of discarded points is dynamically dependent on the environment. In the presented cases the compression ratio seems to be constant because the environment is very similar.

7.4.2 Surface Normals Estimation

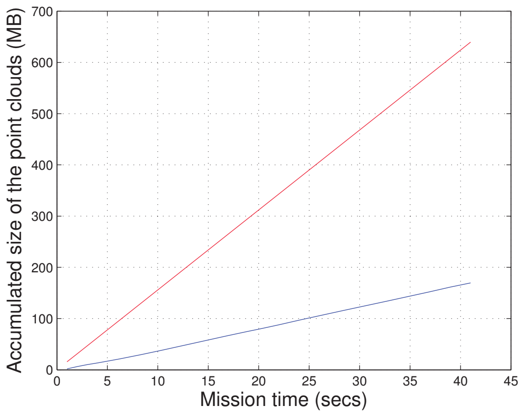
Surface normals are important properties of a geometric surface, and are heavily used in many areas such as computer graphics applications, to apply the correct light sources that generate shadings and other visual effects. The point clouds represent a discrete set of measurements taken from a

Table 7.7: Size of the point clouds of sequence 1 after applying the preprocessing algorithm. For comparison with raw values see Table 7.5.

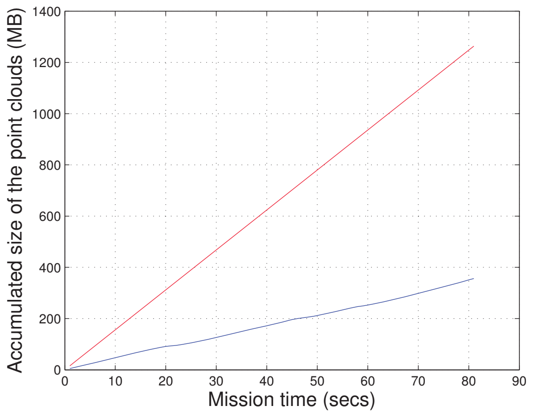
Location	Location Snapshot			Sequence Accumulated		
Name	pt ($\times 10^6$)	size (MB)	ratio	pt ($\times 10^6$)	size (MB)	ratio
A	0.156	1.8	0.12	0.156	1.8	0.12
B	0.372	4.4	0.28	3.4	40.9	0.23
C	0.348	4.1	0.26	6.9	83.5	0.25
D	0.342	4.1	0.26	10.5	126.7	0.26
E	0.344	4.1	0.26	14.1	169.7	0.26

Table 7.8: Size of the point clouds of sequence 2 after applying the preprocessing algorithm. For comparison with raw values see Table 7.6.

Location	Location Snapshot			Sequence Accumulated		
Name	pt ($\times 10^6$)	size (MB)	ratio	pt ($\times 10^6$)	size (MB)	ratio
A	0.402	4.8	0.30	0	0	-
B	0.404	4.8	0.31	3.9	47.0	0.30
C	0.166	1.9	0.12	7.6	91.4	0.29
D	0.375	4.5	0.28	10.5	126.6	0.27
E	0.388	4.6	0.29	14.3	172.0	0.27
F	0.353	4.2	0.27	17.6	211.7	0.27
G	0.323	3.8	0.24	21.0	252.7	0.27
H	0.418	5.0	0.32	24.8	298.3	0.27
I	0.490	5.1	0.33	29.2	350.9	0.28



(*a*)



(*b*)

Figure 7.10: Size of the accumulated point clouds using raw (red) and preprocessed data (blue) as a function of the mission time. (*a*) sequence 1; (*a*) sequence 2.

given surface. As will be shown in the next subsections, the knowledge of the normal vector to that surface is very useful especially for detecting the surface plane. There are two possibilities when addressing the surface normal estimation problem. The first option is to reconstruct the surface from the point cloud using surface meshing techniques, and then compute the normals of this surface. This option is not very suitable for handling real time large point clouds processing since that surface meshing techniques are computationally expensive. The second option is to use approximations to infer the normals from the point cloud directly. These approximations are computed locally, i.e., on the neighborhood of each point. Given the problem of real time reconstruction of road scenarios from large point clouds, this second option is the most adequate since it is faster to compute than the previous.

Since a local analysis for each point is used, the first concept that needs to be addressed is that of neighbourhood. Given a query point p_q it is possible to find its neighbor points using the expression:

$$\|\mathrm{p}_q - \mathrm{p}_k\| < R \quad \forall k \in \{0, 1, ..., n\}. \tag{7.7}$$

where $\| \ \|$ represents the Euclidean distance between the points, and R is a threshold parameter, i.e., the maximum allowed distance for a point p_k to be considered a neighbor of p_q. However, this is a brute force process and could take long to go through a large size point cloud. Because of this several authors have proposed approximate nearest neighbor solutions that can speed up computation times. Orthogonal decomposition methods have been proposed in [Arya & Mount 1993] [Arya *et al.* 1998] and [Muja & Lowe 2009], while in [Silpa-Anan & Hartley 2008] multiple randomized trees are created at once to help split the dimensions on which the data has the greatest variance. In [Nuchter *et al.* 2007] a cached Kd tree is used to provide a significant speedup of over neighbor point queries. Finally, in [Muja & Lowe 2009], the usage of the squared Euclidean distance is proposed to avoid the slower computation of the square rooted distances. By using eq. (7.7) it is possible to devise two different strategies to obtain the list of neighbor points:

- R radius search, where all points that lie inside a sphere of radius R centered on the query point are regarded as neighbours;
- K nearest neighbors search, where points are sorted according to their distance to the query point and then the K closest points are selected.

Both methods have advantages and disadvantages. While it might be interesting to use the radius search to list all points that are closer than a specified radius, this approach may suffer from extreme disparities in the number of selected neighbors of two different query points. This observation is particularly imminent when analysing sparse point clouds, which is the case. For example, if some statistical method is applied to the list of neighbor points, the method looses precision when there are few points in the radius neighborhood of p_q. It may also occur that there are no neighbors within the specified radius. It is sometimes important for feature vector computation and comparison that the number of neighbors is the same for all query points, so that the feature vectors have the same size. In these occasions, the K search is better suited. Obviously, in both methods, the parameters R or K

will significantly influence the estimation procedure. Although some works have been proposed to automatically select suitable values for these parameter [Lalonde *et al.* 2005] [Mitra *et al.* 2004], the fact is that these approaches rely on assumptions such as a constant density of the point cloud and are slow and inadequate for real time processing. In the tested datasets it was possible to empirically select acceptable parameter values after a few attempts.

There are several proposed methodologies to perform surface normal estimation. A Voronoi diagram based method is proposed in [Dey *et al.* 2005], while in [Kamberov & Kamberova 2007] a general orientability constraint is shown to improve the estimation of the normals. A comparison of existing methods for surface normal estimation with a special emphasis on the trade-off between quality and speed is provided in [Klasing *et al.* 2009]. The simplest method proposed is based on the first order 3D plane fitting [Berkmann & Caelli 1994]. In fact, the problem of estimating the normal is the same as that of determining the plane where the neighboring points lie, which in turn is a least square fitting problem. The implementation employed in this work is provided by the Point Cloud Library (PCL) [Rusu & Cousins 2011]. Further details are given in [Rusu 2009].

Figure 7.11 shows the normals estimation output using K nearest neighbours on the top and radius search on the bottom. Results are from location *D* of sequence 2 (see Fig. B.4). In Figure 7.11 (*a*), there is a significant noise in the normal estimation. This is significantly reduced when a more suited value of K is selected (Fig. 7.11 (*b*)). The same occurs for Fig. 7.11 (*c*), where the estimated normals are noisy. In Fig. 7.11 (*d*) the noise is decreased when different values of R are used.

7.4.3 Statistical Outlier Removal

The statistical outlier removal filter is presented in detail in [Rusu 2009]. The motivation is that depending on the acquisition sensor and even on the scenarios characteristics, some points may be the result of erroneous measurements. The idea is to discard points whose position is significantly different from that of its neighbors. To avoid situations where no neighbors are found, a K nearest neighbor search is performed. The proposed solution is based on a statistical analysis of the neighborhood. Let $\mathcal{P}^{\mathcal{N}}$ be the point cloud containing n neighbors of a given query point p_q. The mean distance from the query point to all neighbor points $\bar{\mathrm{p}}_q$ is computed as follows:

$$\bar{\mathrm{p}}_q = \frac{\sum_{k=0}^{n-1} \|\mathrm{p}_q - \mathrm{p}_k\|}{n}, \mathrm{p}_k \in \mathcal{P}^{\mathcal{N}}. \tag{7.8}$$

The next step is to compute the average distance from each neighbor point to all other points, and extract the mean μ and standard deviation σ of these values: The query point is considered an outlier (and not copied to the output point cloud $\mathcal{P}^*$) if it does not have an average distance to all neighbors similar to the one observed on average:

$$\mathcal{P}^* = \Big\{ \mathrm{p}_q \in \mathcal{P} \mid \mu - \sigma \cdot \alpha < \bar{\mathrm{p}}_q < \mu + \sigma \cdot \alpha, \quad \forall q \in \{0, 1, ..., n\} \quad \Big\}, \tag{7.9}$$

where α is a sensitivity parameters that defines how similar the query point must be to the neighborhood.

A second method to remove outliers is based on the consistency of the estimated normal vectors. Unlike the previous method, this one requires information about the normal orientation for each point. This is done using the method presented in section 7.4.2. It performs a statistical analysis of the estimated normals and removes points whose estimated normal does not follow the average trend of the neighborhood. Let $\vec{\mathcal{P}}$ denote a point cloud containing 3D points and the estimated normals $\vec{\mathrm{p}}$, defined as:

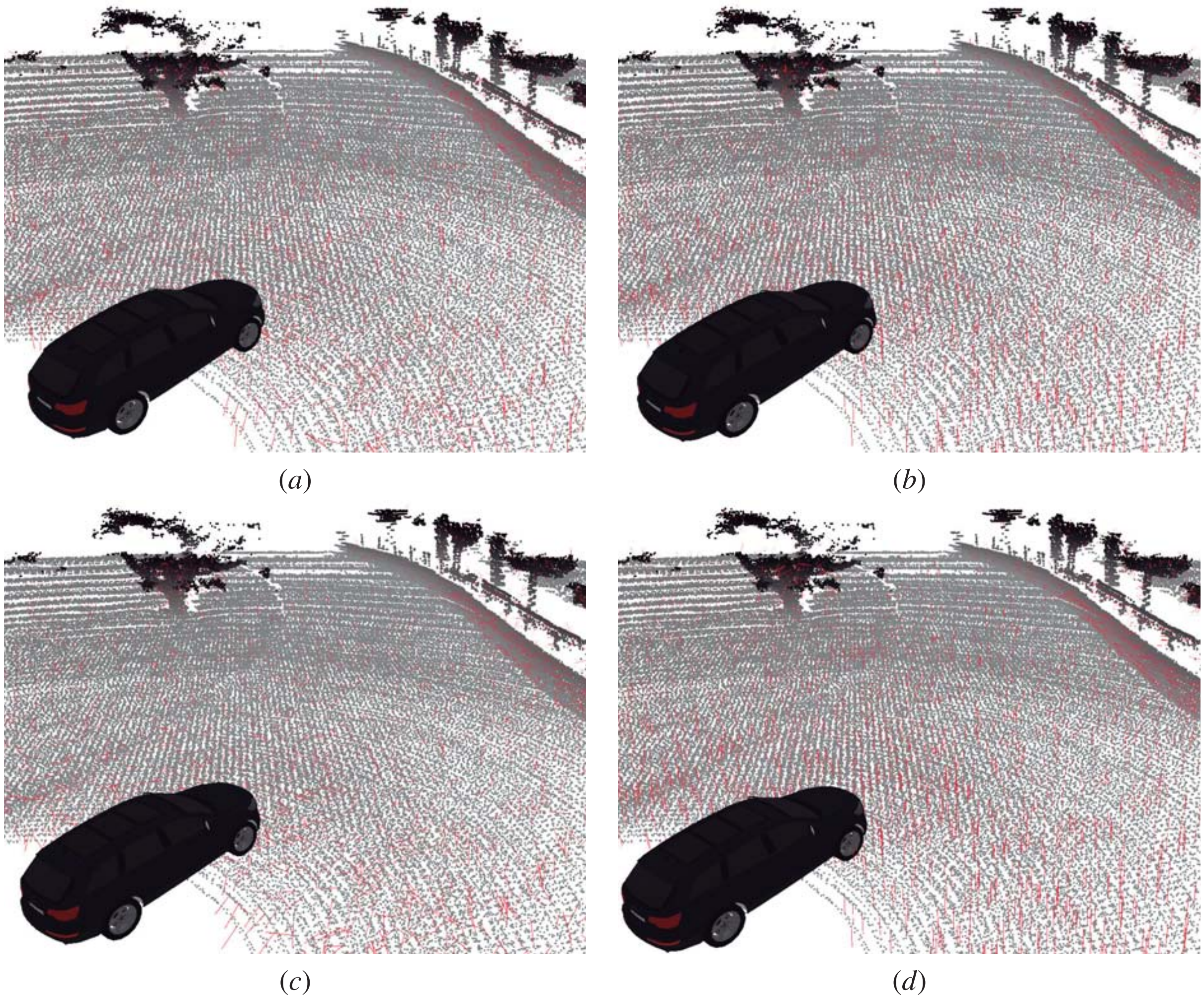


Figure 7.11: Normal estimation results using K nearest neighbours: (*a*) $K = 10$; (*b*) $K = 30$; results using radius search: (*c*) $R = 0.1$; (*d*) $R = 0.5$ meters;

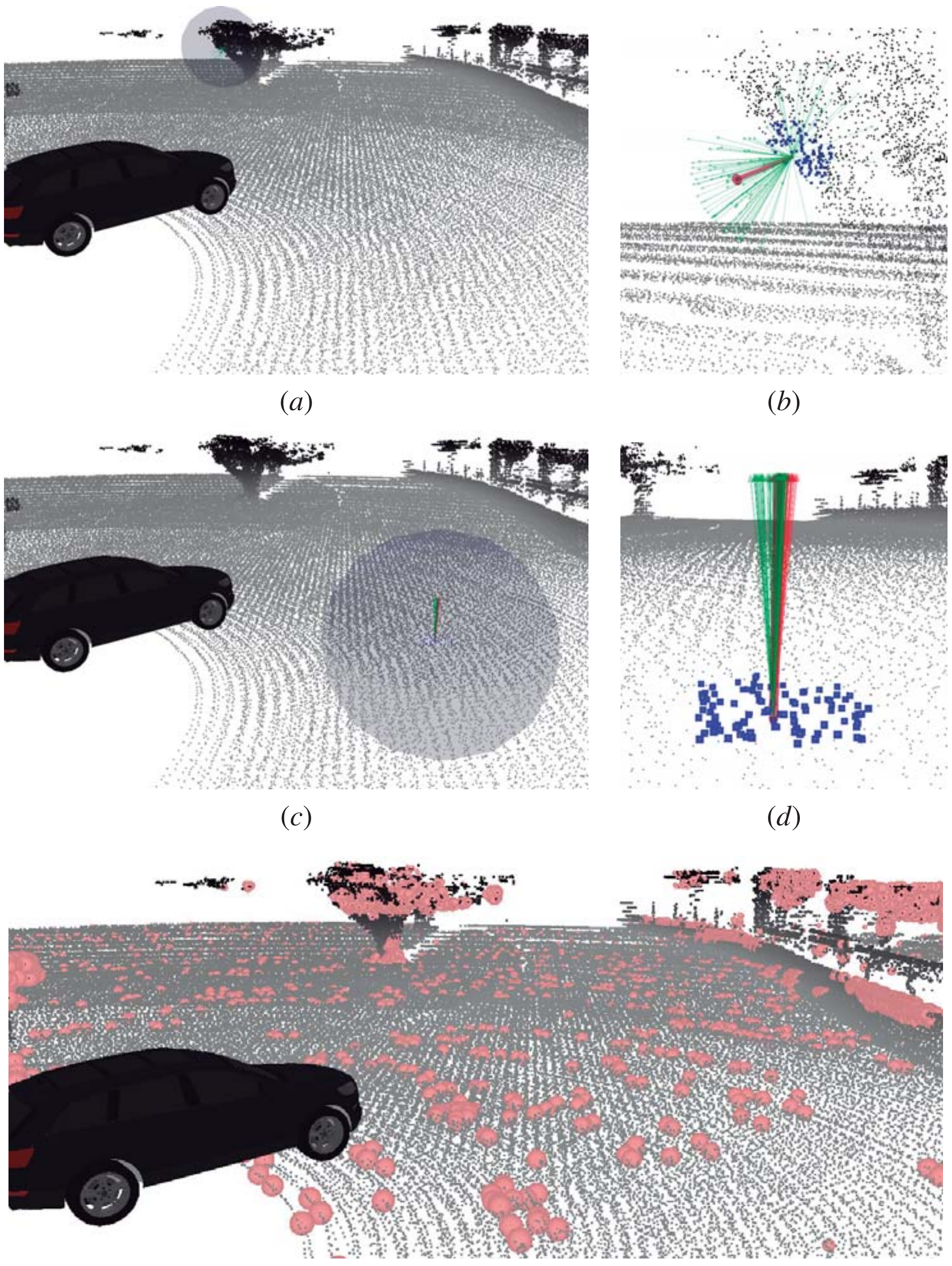

(*a*) (*b*) (*c*) (*d*) (*e*)

Figure 7.12: Results from the estimated normals consistency filter. Neighbor points are shown in blue, neighbor points orientations as green arrows and mean orientation vector in red. A query point in a tree canopy (*a*) and (*b*); query point on the road (*c*) and (*d*); (*e*) points excluded by the filter (shown in red).

$$\vec{\mathcal{P}} = [\vec{\mathrm{p}}_0,\vec{\mathrm{p}}_1...\vec{\mathrm{p}}_n] = \begin{bmatrix} x_0 & x_1 & \cdots & x_n \\ y_0 & y_1 & \cdots & y_n \\ z_0 & z_1 & \cdots & z_n \\ \mathbf{q}_0 & \mathbf{q}_1 & \cdots & \mathbf{q}_n \end{bmatrix}, \tag{7.10}$$

where $\mathbf{q}$ is a unit quaternion defining the orientation of the normal, $\mathbf{q}_i = \begin{bmatrix} q_{ix} & q_{iy} & q_{iz} & w_i \end{bmatrix}'$. Note that in this case since the normal only provides the orientation of a single axis and not of a full coordinate frame, the w component of the quaternion is not used and is set to zero. Let $\bar{\mathbf{q}}_j$ be the mean orientation vector of the points in the neighborhood of the query point. The standard deviation σ of the angle between each neighbor vector and $\bar{\mathbf{q}}_j$ are computed. The filter will exclude points that have neighborhoods where the orientation of the normals has a large standard deviation in the angle to the mean orientation vector.

$$\mathcal{P}^* = \Big\{ \mathrm{p}_q \in \mathcal{P} \mid \sigma < \epsilon, \quad \forall q \in \{0, 1, ..., n\} \quad \Big\}, \tag{7.11}$$

where ϵ is a fixed threshold value. The motivation behind the filter is to try to exclude from a point cloud the regions where the normals are erratic. This is because as will be shown in the next sections, planar polygonal primitives are used to extract geometric knowledge about the environment. For planes to be detected, some consensus in the normals orientation must exist. This filter immediately eliminates regions that lack a planar geometry. It is the case of tree canopies, where the leafs and branches generate a chaotic dispersion of the normals orientations. On the other hand, walls or roads are expected to have high consensus in the orientation of the normals.

Figure 7.12 shows some details of the filter. Results refer to location *D* of sequence 2 (see Fig. B.4). Two different query points are shown. The query point shown in Figs. 7.12 (*a*) and (*b*) is on a tree canopy, while the one presented in Figs. 7.12 (*c*) and (*d*) lies on the road. As expected, the first point shows a greater variation on the estimated normals (see Figs. 7.12 (*b*) and (*d*)). The filter extracts points from the point cloud using a maximum bond over a measure of this variation. Figure 7.12 (*e*) shows the excluded points. It is possible to see that, although some points lying on the road and walls are also excluded, the filter focuses more on discarding points on the tree canopies and other regions where the estimated normals are not expected to be consistent.

7.5 Conclusions

This chapter discussed the 3D point cloud data sets produced by vehicles equipped with multiple sensors. The MIT data set was described in detail, since it will be used to assess the performance of the algorithms proposed in the following chapters. Besides a description of several available data sets and, in particular, of the MIT data set, this chapter also presented several 3D point cloud preprocessing algorithms. These have the objective of reducing the original size of the raw point cloud, and it was shown that they can cut the original point clouds to about 30% of their original size.

Chapter 8

Geometric Scene Reconstruction

This chapter describes a new algorithm designed to perform geometric scene reconstruction. The idea is to describe a scene using sets of geometric polygonal primitives. Results will show that this approach can produce scene representations in considerably less time, compared to classical triangulation approaches. The chapter starts with a brief introduction to the problem (section 8.1). Section 8.2 provides a brief state of the art on scene reconstruction. Later, in section 8.3 the proposed approach is described in detail. Results are presented in section 8.4 and conclusions in section 8.5.

8.1 Introduction

Recent years have witnessed the arrival in the market of 3D sensors with improved performances. Some examples are the Velodyne Lidar [Velodyne 2012], the SR time of flight cameras (http://www.mesa-imaging.ch/) or the Microsoft Kinect (http://en.wikipedia.org/wiki/Kinect). Although these recent sensors provide highly accurate 3D data (see Figs, 1 and 2 in the application support document), they do pose new problems, mostly related to the amount of 3D data they generate, which baffles the performance of traditional 3D triangulation or surface reconstruction algorithms. Therefore, the scope of complex scene reconstruction is narrowed to off line applications, most of them restricted to the visualization of the 3D models.

In the meantime, in the robotics community, one of the most demanding challenges nowadays is the perception: how can robots cope with the environment and adequately interact with it? One of the cornerstones to that answer is perception. Without an accurate perception of the scene and the objects, a robot cannot be expected to properly execute complex tasks.

With the advent of new 3D sensing hardware, in particular those at very low cost, 3D perception gains more and more importance in robotics, as well as in other research domains. As a result, it is expected that, in the future, most robots will be able to *see* the world in 3D. However, there is still a gap between the data provided by these new sensors and the existing 3D processing algorithms. In this scope, one of the most important problems is 3D scene reconstruction, where traditional algorithms based on building triangular meshes are unable to operate in real time. The application domain can be several robotic applications, where real time is an nonnegotiable requirement, although 3D reconstruction in real time is a topic that falls into the scope of many other research fields. Some examples are reverse engineering, dimensional analysis, architecture and cultural heritage, medical robotics or 3D television.

This chapter proposes a novel scene reconstruction algorithm based on polygonal primitives. Unlike traditional scene reconstruction approaches, polygonal primitives do have the potential to be computed in real time. Furthermore, it is proposed that the reconstructed model is refined over time, that is, that the scene representation is incrementally improved when new 3D data arrives. Hence, the proposed representation will explore how a polygon based scene representation can be used to improve the effectiveness of some of the most common tasks of a autonomous robot, for example, trajectory planning, object detection or object recognition.

8.2 Related Work

Scene reconstruction is defined as the computation of a geometric 3D model from multiple measurements. These measurements could be obtained from stereo systems, range sensors, etc. It could also include the texture mapping of the generated model. Scene reconstruction methodologies are grouped into two different approaches: surface based representations or volumetric occupancy representations. In the first, the underlying surfaces of the scene that generated the range measurements are estimated, while in the second, the range measurements are grouped into grid cells which are then labeled vacant or occupied. Traditional surface based representations include several 3D triangulations methodologies, such as 3D Delaunay triangulation [Jovanovic & Lorentz 2011], or Ball Pivoting Algorithm (BPA) [Specht & Devy 2004]. There are also some alternative higher order surface representation methods such as Poisson surface reconstruction [Yin *et al.* 2010], Orientation Inference Framework [Chen & Lai 2011] or learning approaches [de Medeiros Brito *et al.* 2008]. However, most of these methods do not tackle well noisy range measurements and, above all, since these methods involve a large number of nearest neighbor queries, they are very slow to compute. One attempt to fasten the triangulation of point clouds was done in [Marton *et al.* 2009], but authors report they have only achieved near real time. Volumetric occupancy representations include occupancy grids [Weiss *et al.* 2007], elevation maps [Oniga & Nedevschi 2010], multi-level surface maps [Rivadeneyra *et al.* 2009] or octrees [Zhou *et al.* 2011]. While these representations are easier to compute, they do not provide accurate information about the geometry of the scene.

Recent research in the fields of pattern recognition suggest that the usage of 3D sensors improves the effectiveness of perception [Chen & Bhanu 2009] [Wang *et al.* 2007], "since it supports good situation awareness for motion level tele operation as well as higher level intelligent autonomous functions" [Birk *et al.* 2009]. However, in order to generate information from the 3D data, the majority of the robotics research community continues to build simplified 2D or 2.5D scene descriptions such as occupancy grids [Weiss *et al.* 2007], elevation maps [Oniga & Nedevschi 2010] or use discretized grid like approaches as in octrees [Zhou *et al.* 2011].

In conclusion, on the one hand, the new generation of 3D sensors are valuable since they provide more accurate data, but on the other hand traditional geometric reconstruction algorithms are not well suited to tackle such large quantities of data. On the one hand, perception and action planing in robots would surely benefit from a more detailed scene representation, while on the other the real time requirements of robotic applications have led to more simplified descriptions of the environment. A gap is evident.

8.3 Proposed Approach

It was shown in chapter 7 that the size of the point clouds acquired by autonomous vehicles is very large. Preprocessing mechanisms were proposed to reduce the size of the point clouds. The proposed mechanisms are fast to compute since they are based on simple condition tests which decide whether to keep or discard points. As shown in Tables 7.7 and 7.8, these preprocessing filters are able to reduce the point clouds to approximately 400K points per second. In Figure 7.10 it is shown that, although the preprocessing filters significantly reduce the size of the point clouds, if points are accumulated over time the size after only a few tenths of seconds becomes very large. In the examples shown, the accumulated points occupy about 10 MBs per second. If one considers for example that the Massachusetts Institute of Technology (MIT) DARPA mission 1 contains over ten thousand seconds (see Table 7.2), the resulting size would be of about 100 GBs, which is not a feasible size. If the goal is to have a representation of the whole environment, one must balance both the quality of the representation and the memory requirements. Furthermore, it does not seem a clever solution just to accumulate points. For example if the vehicle is stopped, the accumulated points would refer to the same scenario around the vehicle. In this case many of the stored information would be replicated. Even considering that the vehicle is moving, there is significant overlap between consecutive Velodyne scans. This insight provides a hint on how to compress the size of the point clouds. An alternative representation must be devised that is able to include information gathered over multiple scans, but that can also effectivelly discard range measurements of objects that are already included in the representation.

8.3.1 Representation

The term geometric primitive is used in various senses both in computer graphics and CAD applications, but the common basic meaning refers to the simplest, i.e., atomic or irreducible, geometric objects that the system can handle. Several geometric primitives are used in modern applications: cubes, spheres, planes, torus, etc. In the current work, we propose to explore the usage of polygonal geometric primitives, that is polygons. Polygons require only two entities to be described: the support plane and a list of points, lying on that plane, that define a closed polygon. Polygons may yield important advantages when compared to other classical approaches, namely: the fast detection of polygonal geometric primitives is simple when compared to the detection of other more complex primitives; the applicability, given that most of the reconstructed environments are structured of at least semi structured environments, it seems feasible to represent most of the 3D structure with polygons; and data reduction, since the scene is reduced to a set of polygons each with a support plane, further information on the polygons is represented in two dimensions, on a local reference system that lies on that support plane. Furthermore, unlike triangulation methods, one polygon may describe thousands of range measurements. These two considerations show how the novel scene representation will considerably reduce the amount of information to store and thus, be able to effectivelly compress the 3D data. This novel scene representation will be compared to other approaches. It is expected that this representation can:

- Perform scene reconstruction in real time;
- Have similar accuracy when compared to other approaches;
- Refine the representation when new 3D data arrives, i.e. to be dynamic;
- Be easily applicable as an input to standard obstacle detection, trajectory planning or pattern recognition algorithms;

This representation should be best suited to be used by most of the classical layers in a robots computation cycle: perception, obstacle detection, planning, tracking. Furthermore, given the high degree of detail in the representation and the fact that it is built after 3D data, it is expected that the performance of those layers actually increases. It is a novel approach, because it balances both the quality of the representation and the computational restrictions, having in mind the functional purpose of the representation. Here, the term *functional* accredits two interpretations: real time demands, i.e., from a robot's perspective, it is better to have a coarse representation of the environment in a few seconds than a very detailed one only after a few minutes; usability, i.e., the representation is easily binded to other algorithms like object detection and recognition or motion planning, in the sense that these may be effortlessly adapted to use the representation as an input. The system can be described as an intermediate layer between raw sensor data, given both by 3D range measurements as well as optical or thermal images, and the perception and planning layers. It integrates the data from all provided sensors and produces the best available representation within a task related reasonable amount of time.

The problem is how to compute such a complex scene representation within restricted time boundaries. We argue that although it is not possible to obtain a complex representation within seconds, it is possible to start from a coarse model of the scene and then update over time. We assume that the coarse model should have some practical usability, even if limited, and therefore should be made available to the subsequent robot classical functionality layers. The representation would then be refined as additional sensor information arrives. Hence, there is always a periodical availability of the best possible scene representation. Furthermore, the new raw 3D data is compared to the scene representation before being queued for refining the model. Portions of this data that are measurements of existing entities in the scene model are promptly discarded, saving computation time. Suppose a robot that is stopped or traveling very slowly trough a given scene with a wall on the right. At iteration zero, 3D range measurements indicate that a polygon can be used to describe the walls surface. In the next iteration, since the robot is moving slowly, a portion of the 3D data still refers to the wall. If this data is discarded with minimal computational requirements, then only the "unexplained" portions of the data would have to be scanned. This cascade like architecture is the key to achieve real time performance of the system.

Taking all previous considerations into account we propose to represent the environment using a set of polygonal geometric primitives, that is, polygons. Polygons require only two entities to be described: the support plane and a list of points, lying on that plane, that define a closed polygon. Each pair of consecutive points define a line segment and these, grouped together, bound the interior of the polygon. The choice of polygons instead of any other geometric primitives is explained for various reasons:

- Fast detection: the detection of polygonal geometric primitives is simple when compared to the detection of other more complex primitives;
- Applicability: given that road environments are structured, or at least semi structured, environments it seems feasible to represent the 3D structure with planes;
- Data reduction: since the scene is reduced to a set of polygons each with a support plane, further information on the polygons is represented in two dimensions, on a local reference system that lies on that support plane.

Figure 8.1 shows an example of a polygonal primitive. This example is located in sequence 1, location *C* (see Fig. A.3). In this case, there are 3920 range measurements from the wall panel. The polygonal geometric primitive is represented by a support plane and 22 polygon points. This simple example already shows the data reduction potential of the polygonal primitives.

The next sections will detail the algorithms employed in the detection and expansion of the polygonal primitives.

8.3.2 Support Plane

The previous section has described the representation primitives that are proposed. Geometric polygonal primitives are described by a support plane and a bounding polygon. This section will address the

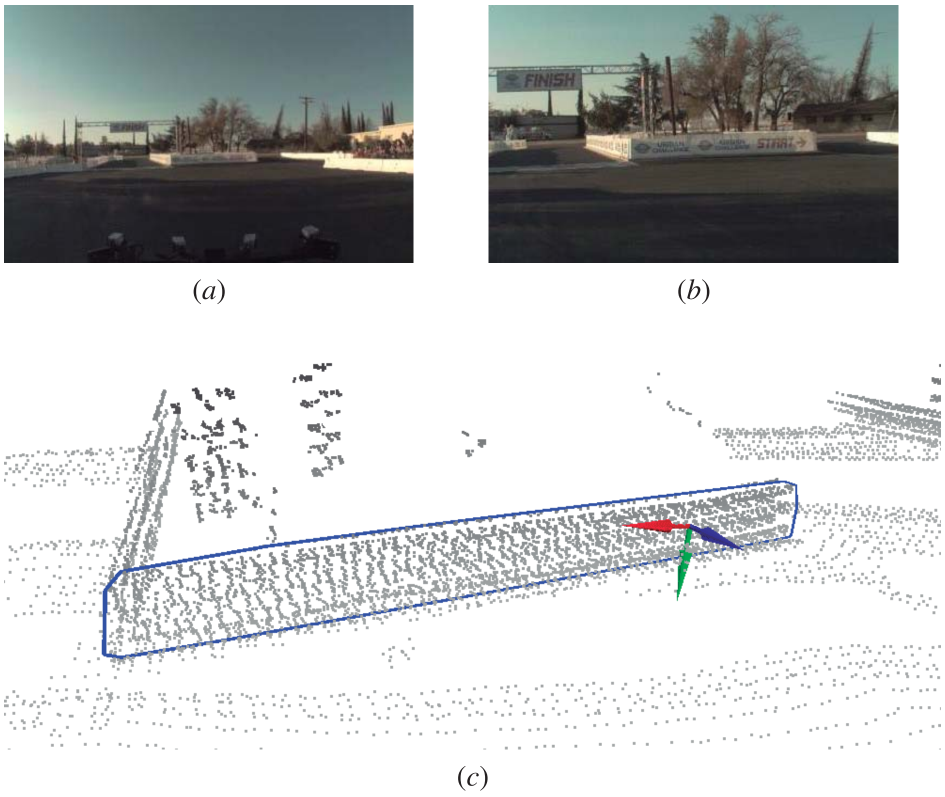


Figure 8.1: An example of a geometric polygonal primitive used to describe a wall panel, in sequence 1, location C (see Fig. A.3). (a) image from the front camera; (b) image from the front 6mm camera; (c) 3D view of the wall panel range points (in gray), the detected polygonal geometric primitive (in blue) and local primitive coordinate system.

detection of the support plane. Let $\mathcal{G}^i$ represent the *ith* polygonal geometric primitive of a given scene, with the support plane Hessian form coefficients is represented by $\mathcal{G}^i_p = \left[\begin{array}{cccc} a^i & b^i & c^i & d^i \end{array}\right]$. The search for the support plane is done on a given input point cloud $\mathcal{P}$ using a Random Sample Consensus (RANSAC) procedure. RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed. The algorithm was first published in [Fischler & Bolles 1981]. The assumption is that data consists of *inliers*, i.e., data whose distribution can be explained by some set of model parameters, and *outliers*, data that does not fit the model. The input to the RANSAC algorithm is a set of observed data values, a parameterized model which can explain or be fitted to the observations, and some confidence parameters. RANSAC achieves its goal by iteratively executing the following procedure:

- Select a random subset of the original data, i.e., the hypothetical inliers;

- Fit a model to the hypothetical inliers;
- Test the whole data against the fitted model, adding all data that fits well to the set of hypothetical inliers;
- Assess the quality of the estimated model by the amount of data that fitted well;
- Reestimate the model with the new set of hypothetical inliers;

This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are classified as inliers or a refined model together with a corresponding error measure. The model with the smaller error measure is selected as the output to the RANSAC.

In the specific case of detecting the polygonal geometric primitives support planes, there are two alternative methods. The first is the detection of a freely oriented plane, and the second of an oriented plane. These two alternative methods are better suited to detect different planes, as will be discussed. Both are explained in detail.

Let the point cloud $\mathcal{P}$ be the input data used for estimating the support plane of the ith primitive. The procedure for the freely oriented plane detection is the following shown in Algorithm 8.1.

This procedure is performed a number of times, and the support plane that had the higher number of points considered inliers is selected. This approach has good results when the ratio between the number of points that lie on the plane and the total number of points is high. This is usually true for the road plane, which is the first detected plane almost every time. Other smaller structures like

Algorithm 8.1 Detection of a support plane using RANSAC

```
Input: $\mathcal{P}$, the input point cloud, number of points $> 3$
Output: Plane parameters, $a^i, b^i, c^i, d^i$; list of inliers $\mathbf{I}$
  for $i = 0 \rightarrow$ maxiter do
      Randomly select three unique non collinear points $\mathrm{p}_l$, $\mathrm{p}_m$ and $\mathrm{p}_n \in \mathcal{P}$
      Compute the plane model parameters $a^i, b^i, c^i, d^i$ from $\mathrm{p}_l$, $\mathrm{p}_m$ and $\mathrm{p}_n$ [a]
      Add $\mathrm{p}_l$, $\mathrm{p}_m$ and $\mathrm{p}_n$ to the list of inliers, $\mathbf{I}^i \leftarrow \{\mathrm{p}_l, \mathrm{p}_m, \mathrm{p}_n\}$
      for all $\mathrm{p}_j \in \mathcal{P}$ do
          Compute the distance $d_j$ between the estimated plane and $\mathrm{p}_j$ [b]
          if $d_j < d_{threshold}$ then
              Add $\mathrm{p}_j$ to the list of inliers, $\mathbf{I}^i \leftarrow \{\mathbf{I}^i, \mathrm{p}_j\}$
          end if
      end for
      $n^i \leftarrow$ number of elements in $\mathbf{I}^i$
  end for
  return $a^k, b^k, c^k, d^k$; $\mathbf{I}^k$, where $k = \mathbf{argmax}_i(n)$
```

[a] The general problem of determining the plane coefficients given three non collinear unique points [Weisstein 2012a]
[b] The general problem of determining the distance between a point and a plane [Weisstein 2012b]

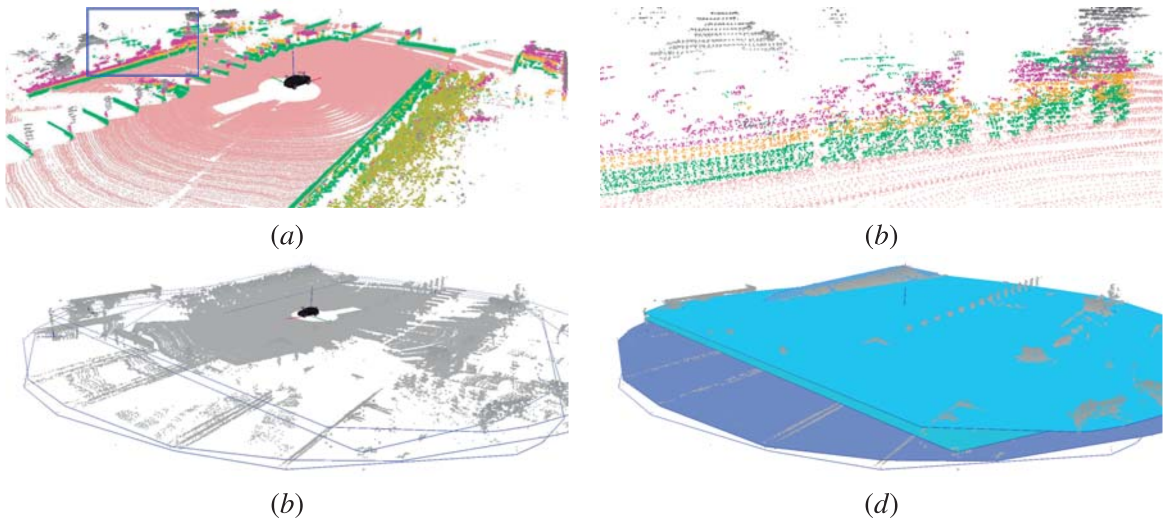


Figure 8.2: Plane detection examples using the freely oriented plane approach in sequence 1, location *C*. Five best RANSAC candidates are shown. (*a*) inliers of each plane candidate with different colors; (*b*) a detail of (*a*); (*c*) the bounding polygon of each of the five plane candidates; (*d*) the plane candidates with filled colors.

walls or panels do not have a large number of points and are therefore not well detected using this approach. If one plans to execute several RANSAC procedures in order to extract several planes as is the case, the walls should be well picked up by the method. Figure 8.2 shows some plane detection examples using the freely oriented plane approach. The first five candidates of a RANSAC procedure are shown. As discussed the road plane is well detected. However, the other best scoring hypothesis are not adequate: they consist of horizontal slices of the whole environment above the road plane (see Fig. 8.2). The reason is that RANSAC searches for the higher consensus, i.e., for planes with the highest number of inliers, and the 3D data is distributed across the horizontal dimension much more than the vertical dimension. Because of this, these horizontal planes will have higher probability (higher consensus most of the times) of being detected since they will always cross portions of several vertical structures in the 3D data. In conclusion, it seems that the freely oriented plane approach is well suited to detect the road plane but not adequate to find vertical oriented structures. The second method, the oriented plane RANSAC approach, offers a solution to this problem.

The oriented plane detection uses a very similar procedure to the freely oriented plane approach. However, the estimated orientation of 3D points is taken into account when detecting a plane. Section 7.4.2 described in detail the methodologies used to estimate a direction normal to the surface where each 3D point presumably lies on. Normal estimation receives the input point cloud $\mathcal{P}$ and generates a new point cloud $\vec{\mathcal{P}}$ with normal vectors associated to each 3D point. This method lets the user specify the orientation of the planes that are going to be searched. Let $\mathbf{q}_u$ be an user defined orientation and

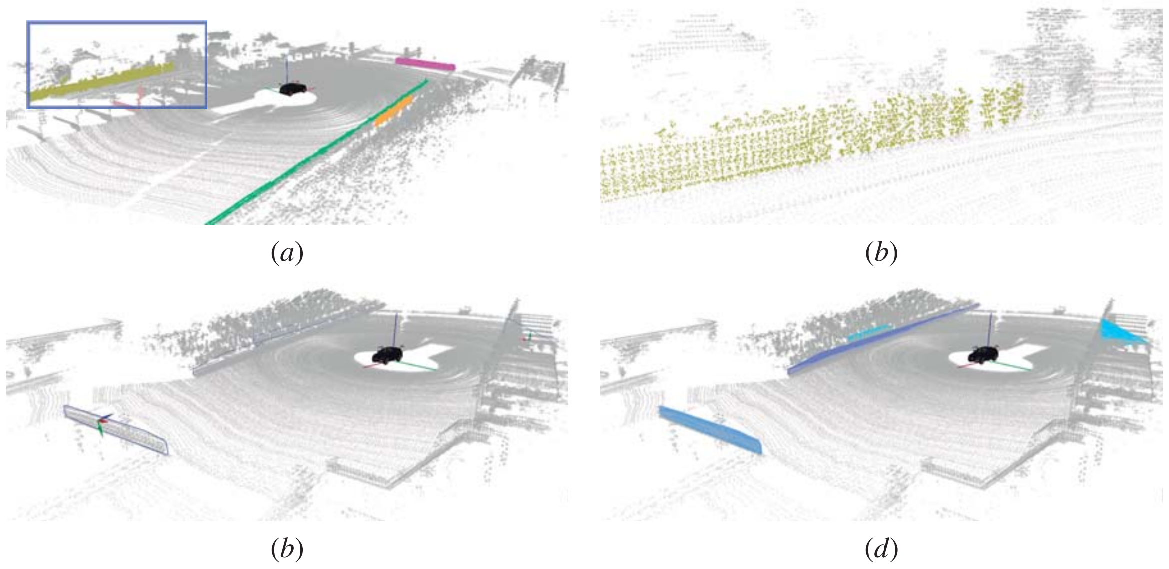


Figure 8.3: Plane detection examples using the oriented plane approach in sequence 1, location *C*. The user defined direction was $\mathbf{q}_u$ was defined as the vertical direction. The five best RANSAC candidates are shown. (*a*) inliers of each plane candidate with different colors; (*b*) a detail of (*a*) (the same as in Fig 8.3 (*b*)); (*c*) the bounding polygon of each of the five plane candidates; (*d*) the plane candidates with filled colors.

$f(\mathbf{q}_1, \mathbf{q}_2)$ a function that retrieves the smallest angle between two vectors $\mathbf{q}_1$ and $\mathbf{q}_2$. The point cloud $\vec{\mathcal{P}}^*$ used as input for RANSAC is obtained from the subset of points whose normals have similar orientation to $\mathbf{q}_u$:

$$\vec{\mathcal{P}}^* = \{\mathrm{p}_j \in \vec{\mathcal{P}} \mid f(\mathbf{q}_u, \mathbf{q}_j) < \alpha_t, \quad \forall j \in [0, n[\quad \}. \tag{8.1}$$

where α_t is a threshold parameter for the maximum angular distance between the user specified preferable direction and the estimated direction. The next steps of this method are in all similar to the freely oriented plane. However, because only 3D points with similar orientation to the user specified one are used for RANSAC, planes that where not easily detected using the previous method are well identified. Figure 8.3 shows an example of detection of oriented planes. It shows that the oriented plane RANSAC based detection is well suited to detect vertical structures.

8.3.3 From RANSAC Inliers to Primitive Support Points

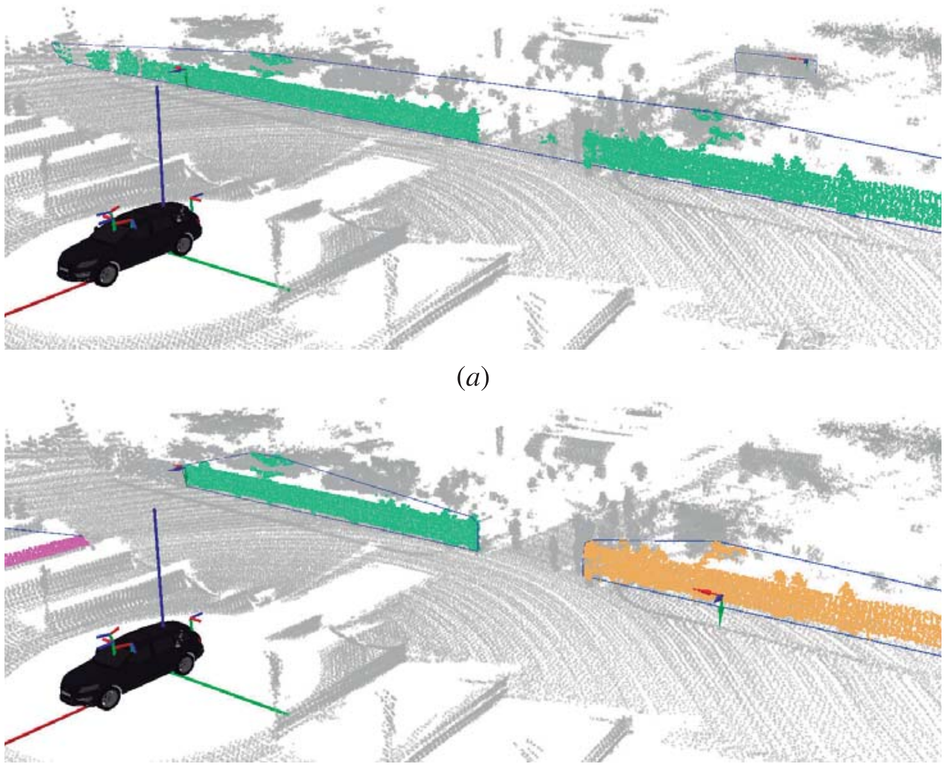
As described in section 8.3.2 a RANSAC procedure is used to extract a set of points that belong to a plane. These points are called inliers. When a plane is detected by RANSAC, the inliers are used as input to the clustering algorithm which is presented in this section. Clustering is required because

Algorithm 8.2 Clustering of point clouds

```
Input: $\mathcal{P}$, the input point cloud
Output: A list C={$C^0, C^1, ..., C^n$} of clusters, where $C^k$ ={$\mathrm{p}_0, \mathrm{p}_1, ..., \mathrm{p}_{m^k}$}
  Initialize number of clusters, $n \leftarrow -1$
  Initialize cluster list, $\mathbf{C} \leftarrow \{\}$
  while $\mathcal{P} \neg$ empty do
      $n \leftarrow n + 1$
      $\mathbf{C} = \{\mathbf{C}, C^n\}$                          ▷ starting a new cluster
      Initialize queue list Q with a random point $\mathrm{p} \in \mathcal{P}$, $\mathbf{Q} \leftarrow \{\mathrm{p}\}$
      Remove p from $\mathcal{P}$
      while Q¬ empty do
          Set seed point $\mathrm{p}_s$ as the first element in Q, $\mathrm{p}_s \leftarrow \mathbf{Q}(0)$
          for all $\mathrm{p}_j \in \mathcal{P}$ do
              Compute the distance $d_j$ between $\mathrm{p}_s$ and $\mathrm{p}_j$
              if $d_j < d_{threshold}$ then
                  Add $\mathrm{p}_j$ to the queue list, $\mathbf{Q} \leftarrow \{\mathbf{Q}, \mathrm{p}_j\}$
                  Remove $\mathrm{p}_j$ from the point cloud $\mathcal{P}$
              end if
          end for
          Add seed point to current cluster, $C^n \leftarrow \{C^n, \mathrm{p}_s\}$
          Remove seed point $\mathrm{p}_s$ from queue list Q
      end while
  end while
```

RANSAC finds a set of inliers that have similar orientations in the estimated normals and that are close to the hypothesized plane. In this case, an algorithm similar to the flood fill algorithm used in image processing [Samet & Tamminen 1988] [He *et al.* 2008] is applied. It is detailed in Alg. 8.2. The clustering algorithm performs the separation of a point cloud of inliers provided by RANSAC into several clusters. The code implementation that was used resorts to a Kd tree based search for distances between points [Rusu & Cousins 2011], which significantly improves the performance of the algorithm. Since several clusters may be segmented and only one must be chosen, we use a simple criteria that selects the cluster with the largest number of points. Points that belong to other clusters are reinserted into the raw data point cloud and are used in future support plane detections.

In Fig. 8.4 (*a*) it is possible to see the inliers (signaled in green) of a RANSAC plane detection. In this case, range measurements from two separate walls have been signaled as inliers to the detected plane since that the support planes of each wall are coplanar. Using the proposed clustering algorithm it is possible to separate the two walls into clusters and, in a first iteration, select the wall (cluster) with the largest number of measurements, i.e., the green wall on the left In Fig. 8.4 (*b*). The yellow wall on the right was detected in a second RANSAC iteration. In conclusion, clustering is a relatively simple mechanism. When combined with RANSAC, it is able to detect geometric polygonal primitives effectivelly.

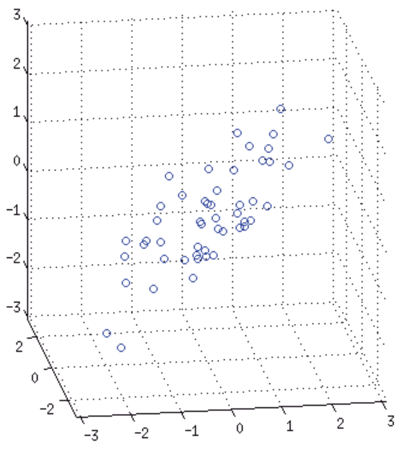

(*a*)

(*b*)

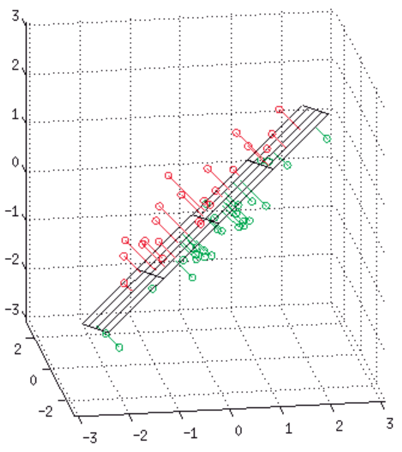
Figure 8.4: Using clustering of point clouds in order to separate inliers RANSAC. (*a*) detection without clustering; (*b*) detection with clustering.

8.3.4 Plane Model Estimation

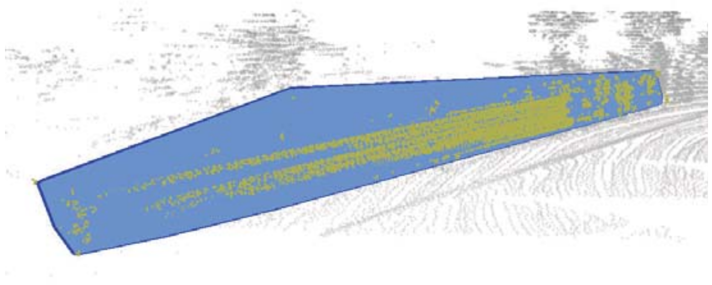
The last step described in the general RANSAC procedure list (see section 8.3.2) is the model re-estimation. It is performed only for the selected hypothesis and consists on a refinement of the model parameters. In fact, the RANSAC is used only to find a set of inliers on the input point cloud. Then, a clustering mechanism will extract a set of connected points. RANSAC does provide an estimate of the planes parameters but a very inaccurate one, since it was defined initially from the three randomly selected points. Hence, after selecting a plane, computing the set of inliers, and extracting the largest cluster, the next problem is how to accurately estimate the plane coefficients given a set with more than three 3D points. Given a set of 3D points, linear least squares fitting is the process of finding the linear sub-space which minimizes the sum of squared distances from all points that compose the set, to their projection onto this linear sub-space, i.e., the plane. Such linear sub-space is obtained using Principal Component Analysis (PCA). PCA is defined as a transformation that transforms the points to a new coordinate system such that the greatest variance by orthogonal projection of the objects comes to lie on the first coordinate, called the first principal component, the second greatest



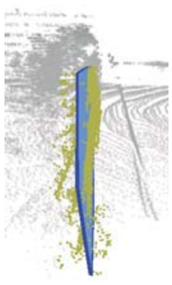
(*a*)



(*b*)



(*c*)



(*d*)

Figure 8.5: PCA for plane model coefficients refinement. (*a*) a set of 3D points; (*b*) the fitted plane and the input points (red or green depending on whether they are above or below the plane); (*c*) and (*d*) another example with the polygon shown in Fig. 8.3 (*b*).

variance on the second coordinate, and so on. The library referenced in [Alliez *et al.* 2012] was used to implement this functionality although there are other possibilities [Mathworks 2012]. Figure 8.5 shows examples of plane fitting with PCA.

In conclusion, this section presented details on the detection planes given a point cloud. It was shown that RANSAC is an appropriate method for extracting a set of inlier 3D points from the input point cloud. A slight variation of this method resorts to a point cloud with estimated normals to discover oriented planes.

8.3.5 Bounding Polygon

As stated in section 8.3.1 the proposed representation basis of the environment, the polygonal geometric primitives, require two descriptors in order to be defined: the support plane and a bounding polygon. The first was addressed in detail in section 8.3.2. The computation of the latest will be described in this section.

The computation of the bounding polygon is done after the detection of the support plane. In other words, for a given ith primitive, its support plane coefficients $\mathcal{G}_p^i = \begin{bmatrix} a^i & b^i & c^i & d^i \end{bmatrix}$ are known. The bounding polygon $\mathtt{P}^i$ is defined by a list of 2D points p:

$$\mathtt{P}^i = [p_0, p_1 ... p_n] = \begin{bmatrix} x_0 & x_1 & ... & x_n \\ y_0 & y_1 & ... & y_n \end{bmatrix} \tag{8.2}$$

where n is the number of points in the polygon. Each pair of consecutive points define a line segment and the group of line segments bounds the polygons interior. In this sense a polygon can also be represented as a set of line segments $\overline{\mathtt{P}}^i$:

$$\overline{\mathtt{P}}^i = \{\overline{p_0p_1}, \overline{p_1p_2}, ..., \overline{p_{n-2}p_{n-1}}, \overline{p_{n-1}p_0}\}. \tag{8.3}$$

In order to define the 2D points a special coordinate system must be computed. It is called the primitive local coordinate system. The local reference system must have its Z axis parallel to the normal of the support plane. The orientation of the remanding axes is arbitrary. We use the first two 3D points p_0 and p_1 in the inliers point cloud provided by the RANSAC search to define the X axis and the Y axis results from:

$$\vec{Y} = \vec{Z} \times \vec{X} \tag{8.4}$$

where $\vec{X}$, $\vec{Y}$ and $\vec{Z}$ are the normalized vectors coincident with each of the primitives local coordinate systems axes, and $\times$ is the cross product. The 2D bounding polygon points are converted to and from 3D coordinates in the world coordinate system as follows: let ${}^{\mathtt{W}}\mathbf{P}^i$ be the list of 3D positions of the ith primitives polygon points, defined in the world $\mathtt{W}$ coordinate system:

$${}^{\mathtt{W}}\mathbf{P}^i = [{}^{\mathtt{W}}\mathbf{p}_0, {}^{\mathtt{W}}\mathbf{p}_1, ... {}^{\mathtt{W}}\mathbf{p}_n,] = \begin{bmatrix} {}^{\mathtt{W}}x_0 & {}^{\mathtt{W}}x_1 & ... & {}^{\mathtt{W}}x_n \\ {}^{\mathtt{W}}y_0 & {}^{\mathtt{W}}y_1 & ... & {}^{\mathtt{W}}y_n \\ {}^{\mathtt{W}}z_0 & {}^{\mathtt{W}}z_1 & ... & {}^{\mathtt{W}}z_n \end{bmatrix}, \tag{8.5}$$

and ${}^{\mathtt{W}}T_{\mathtt{l}_i}$ be a function that transforms point ${}^{\mathtt{W}}\mathbf{p}$ in the world coordinate system to point ${}^{\mathtt{l}_i}\mathbf{p}$ in the local coordinate system:

$${}^{\mathtt{l}_i}\mathbf{P}^i = {}^{\mathtt{W}}T_{\mathtt{l}_i}\left({}^{\mathtt{W}}\mathbf{P}^i\right) = \left[{}^{\mathtt{l}_i}\mathbf{p}_0, {}^{\mathtt{l}_i}\mathbf{p}_1, ... {}^{\mathtt{l}_i}\mathbf{p}_n,\right] = \begin{bmatrix} {}^{\mathtt{l}_i}x_0 & {}^{\mathtt{l}_i}x_1 & ... & {}^{\mathtt{l}_i}x_n \\ {}^{\mathtt{l}_i}y_0 & {}^{\mathtt{l}_i}y_1 & ... & {}^{\mathtt{l}_i}y_n \\ {}^{\mathtt{l}_i}z_0 & {}^{\mathtt{l}_i}z_1 & ... & {}^{\mathtt{l}_i}z_n \end{bmatrix}, \tag{8.6}$$

Then, since the 2D polygon points must lie on the support plane, a projection of a 3D point ${}^{\mathtt{l}_i}\mathbf{p}_j$ is easily produced by discarding the z coordinate. Let $\mathbf{f}_p : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ be a function that projects a 3D point ${}^{\mathtt{l}_i}\mathbf{p}_j$ in the local primitive coordinate system to a 2D point ${}^{\mathtt{l}_i}p_j$ defined in a two dimensional coordinate system that is coincident to the X and Y axes of the local primitive coordinate system:

$$
\begin{aligned}
{}^{\mathtt{l}_i}p_j &= \mathbf{f}_p({}^{\mathtt{l}_i}\mathbf{p}_j) \\
\equiv \mathtt{P}^i = [p_0{,}p_1 ... p_n] = \begin{bmatrix} x_0 & x_1 & ... & x_n \\ y_0 & y_1 & ... & y_n \end{bmatrix} &= \begin{bmatrix} {}^{\mathtt{l}_i}x_0 & {}^{\mathtt{l}_i}x_1 & ... & {}^{\mathtt{l}_i}x_n \\ {}^{\mathtt{l}_i}y_0 & {}^{\mathtt{l}_i}y_1 & ... & {}^{\mathtt{l}_i}y_n \end{bmatrix},
\end{aligned}
\tag{8.7}
$$

There are two ways to compute a bounding polygon from a set of 2D points: convex hull and concave hull. Both are described in the following lines.

The first option is to compute the bounding polygon by performing a 2D convex hull operation on the 2D projections of the inlier 3D points provided by RANSAC. There are a great number of options to perform a convex hull operation: Gift wrapping [Jarvis 1973], Graham scan [Graham 1972], Divide and conquer [Amato & Preparata 1993], Monotone chain [Andrew 1979] and Incremental convex hull algorithm [Kallay 1984], Quick hull [Bykat 1978], among others. It is a well documented topic in the area of computation geometry. In this work the implementation provided in [Hert & Schirra 2012] is used to compute the 2D convex hull, based on a non recursive version of [Bykat 1978], presented in [Barber *et al.* 1996].

The second option is to compute the bounding polygon using a concave hull. Concave hulls, also known as alpha shapes, will provide a more accurate reconstruction of the boundaries of the polygon. There are tools available to compute both 2D [Kai & Da 2012] as well as 3D [Kai *et al.* 2012] alpha shapes. In the current work, an implementation provided by [Rusu & Cousins 2011] is used. Although the concave hull seems more interesting to use since it does a better job of preserving the contours of the data, the fact is that the convex hull does present several advantages when compared to the alpha shapes. First, convex hulls are considerably faster to compute and use in subsequent operations, and second, the computation of the alpha shapes does require an user selection of the α parameter. Because of these reasons, the remanding explanations and results will be shown only with the convex hull polygon. However, it should be noted that all the algorithms described in the next sections are also fitted to work with concave polygons.

Figure 8.6 shows a scene where a polygon is reconstructed using both the convex and concave hull. In Figs 8.6 (*a*) and (*b*) the inliers of the RANSAC procedure are shown in blue. The projection of these to the polygons support plane as described in eq. (8.7) results in the green points. In Figs 8.6 (*c*) and (*d*) the convex and concave hulls are shown in blue and red colors respectively.

8.3.6 Basic Geometric Properties

Although the algorithms proposed in the previous sections are very effective at extracting polygons from 3D data, some additional constraints are helpful in order to maintain a reduced number of polygons. These additional constraints are related to the size and the amount of 3D range measurements

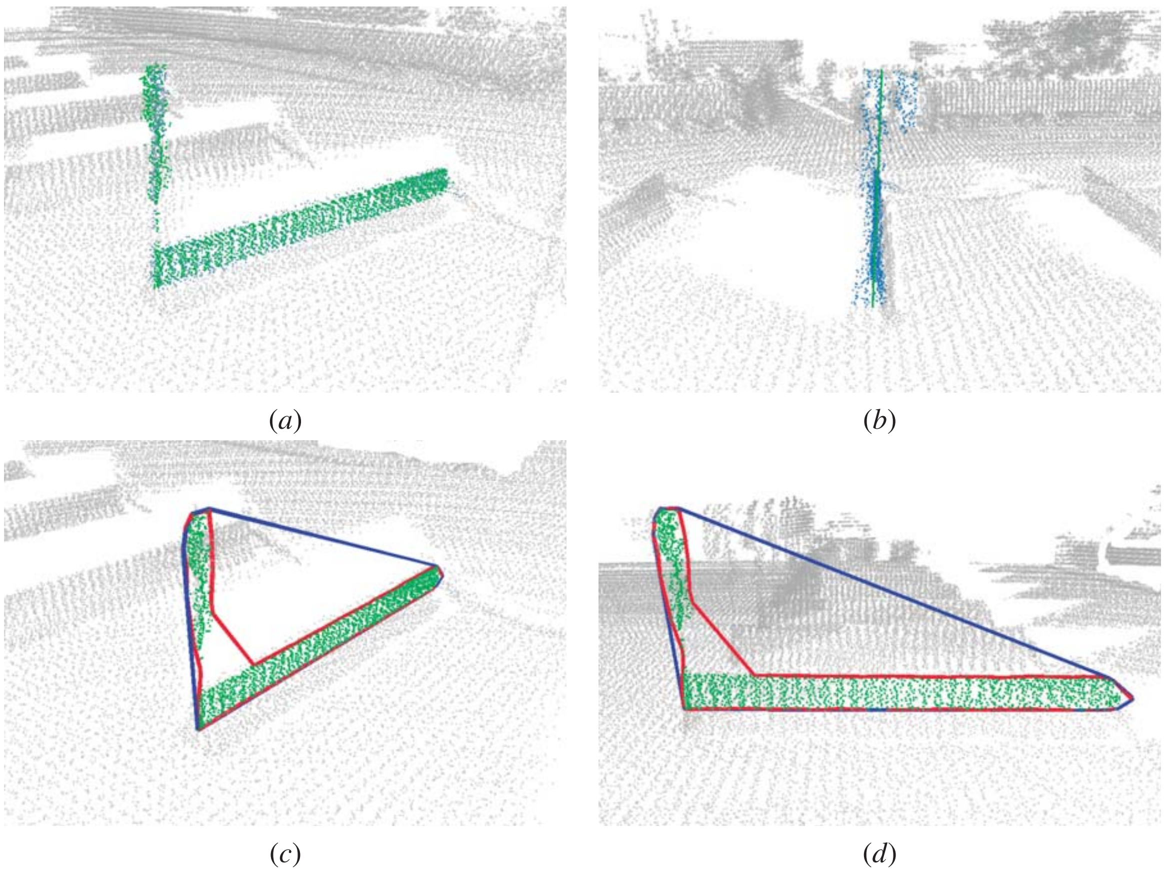


Figure 8.6: The computation of the bounding polygons: (*a*) and (*b*) the RANSAC inliers shown in blue, the projection of these to the support plane shown in green; (*c*) and (*d*) the convex and concave hull polygons in blue and red, respectively.

that support the plane. Both constraints are described below. The polygons computed using either the convex or concave hulls are defined as simple polygons, i.e., polygons whose edges do not intersect. The area of a simple polygon $\mathbf{A}(\mathtt{P})$ containing n vertices was proposed in [Branden 1986], and is given by:

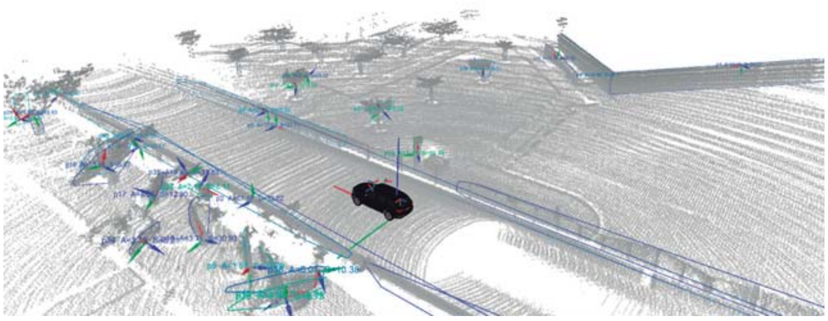
$$\mathbf{A}(\mathtt{P}) = \frac{1}{2}\sum_{i=0}^{n-2}\big(x_i \cdot y_{i+1} - x_{i+1} \cdot y_i\big), \tag{8.8}$$

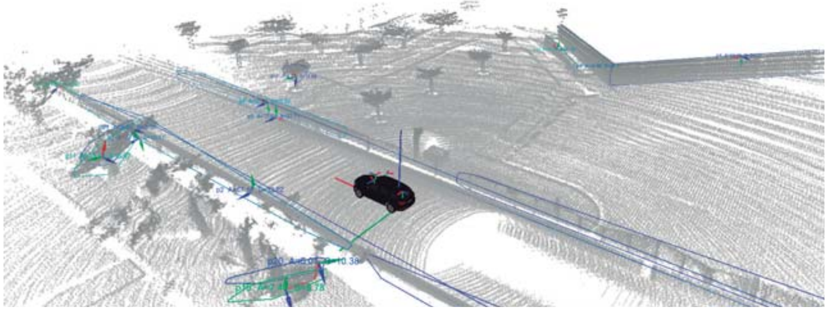
which will return a positive value when the polygons vertices are set in counter clockwise order [Giezeman & Wesselink 2012a]. The other criteria is solidity $\mathbf{S}(\mathtt{P})$, which is defined by the following expression:

$$\mathbf{S}(\mathtt{P}) = \frac{N}{\mathbf{A}(\mathtt{P})}, \tag{8.9}$$

where N is the number of points that are explained by the geometric polygonal primitive, i.e., the number of inliers provided by RANSAC.

These two criteria are used to discard some very small polygons or polygons with large areas but few support planes. Two minimum threshold values are set to decide whether a polygon is discarded or not:


(*a*)


(*b*)

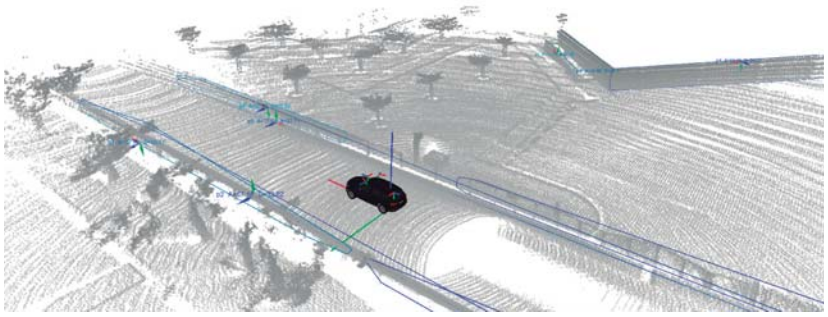
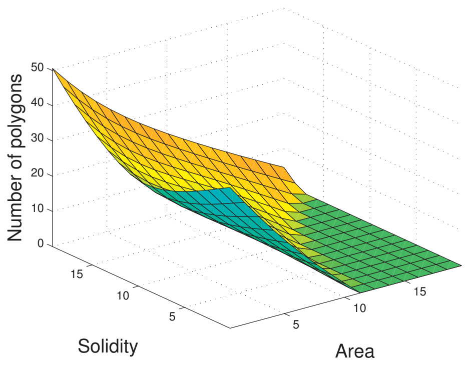
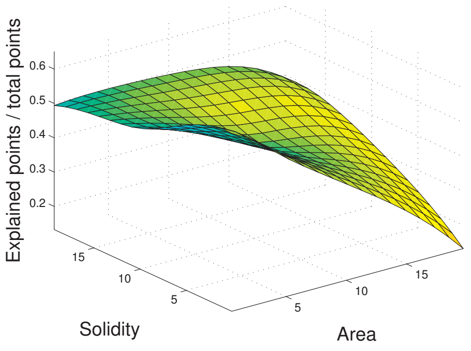

(*c*)

Figure 8.7: Location *H*, sequence 2 (see Fig. B.8, page 415). Examples of geometric polygonal primitives detection using different values of area ($\mathbf{A}_t$) and solidity ($\mathbf{S}_t$) minimum thresholds: (*a*) $\mathbf{A}_t = 0.5$, $\mathbf{S}_t = 0.1$; (*b*) $\mathbf{A}_t = 5$, $\mathbf{S}_t = 0.1$; (*c*) $\mathbf{A}_t = 10$, $\mathbf{S}_t = 10$.



(*a*)



(*b*)

Figure 8.8: Location *C*, sequence 1, influence of the minimum area and solidity thresholds on: (*a*) the number of polygons; (*b*) the number of explained points.

$$\begin{cases} keep \quad \mathtt{P}^i, \quad if \quad \mathbf{A}(\mathtt{P}^i) \geq \mathbf{A}_t \quad and \quad \mathbf{S}(\mathtt{P}^i) \geq \mathbf{S}_t \\ discard, \quad otherwise \end{cases}, \tag{8.10}$$

where $\mathbf{A}_t$ and $\mathbf{S}_t$ are the minimum area and solidity thresholds, respectively. Although very simple, this test will enable the system to discard small polygons and therefore save memory space. It will also not account for polygons which do not have few range measurements, and therefore a small value of solidity. It is also simple to do a manual tunning of these two parameters, since they have close relations to physical properties of the polygons, i.e., size and density.

Figure 8.7 shows the amount of geometric polygonal primitives detected for several values of $\mathbf{A}_t$ and $\mathbf{S}_t$. As expected, changes in the minimum area and solidity thresholds will affect the number of detected polygons. Also, the number of 3D points explained by the polygons is dependent on these parameters. We say that a 3D point *is explained* when there is a valid geometric polygonal primitive that contains the point. Hence, if the number of polygons reduces, it is expected that the number of explained points also decreases. The ratio between the number of explained points for all valid polygons with the total number of input points provides an estimate of how much of the environment is represented by the proposed model, or, in the other hand, how much of the scene remains to be explained.

Figure 8.8 shows two graphs: in (*a*) the influence of $\mathbf{A}_t$ and $\mathbf{S}_t$ to the number of polygons is shown; in (*b*) the influence of those two parameters on the number of explained points is displayed.

8.3.7 A Cascade Processing Configuration

The previous sections described the algorithmic tools used to perform the detection of geometric polygonal primitives from a given point cloud. This section intends to provide the global picture of the detection process, explaining how several primitives are extracted from a single input point cloud. The proposed processing architecture can be described as a cascade configuration. The cascade configuration is inspired in the cascading of classifiers, a technique very common in the field of pattern recognition. Cascades are based on the concatenation of several classifiers, using all information collected from the output from a given classifier as additional information for the next classifier in the cascade. Unlike voting or stacking ensembles, which are multi expert systems, cascading is a multistage one. Cascade setups are also frequently related to human-like attention mechanisms. The main advantages for cascade configurations are the processing speed as well as the admissible poor false positive effectiveness of the classifiers, when considered individually. In the problem at hand, a cascade configuration term refers to the processing of the input point cloud. The input point cloud contains several range measurements, i.e., 3D points. It is assumed that each point can only be used to represent a single geometric polygonal primitive. In other words, primitives may have many 3D

Algorithm 8.3 Cascade configuration for the detection of geometric polygonal primitives

```
Input: $\mathcal{P}^{it=0}$, the input point cloud at iteration 0
Output: A list of geometric polygonal primitives $\mathbf{G} = \{\mathcal{G}^0, \mathcal{G}^1, ..., \mathcal{G}^n\}$
  Initialize number of primitives, $k \leftarrow 0$
  Initialize number of iterations, $it \leftarrow 0$
  Initialize primitives list, $\mathbf{G} \leftarrow \{\}$
  Initialize cycle break flag, $cycle\_break \leftarrow \texttt{false}$
  while $cycle\_break = \texttt{false}$ do
      RANSAC search over $\mathcal{P}^k$, returns estimated plane $\hat{\mathcal{G}}_p^k$ (first guess) and inliers $\mathcal{I}^k$
      if RANSAC found a candidate then
          Cluster inliers point cloud $\mathcal{I}^k$ to cluster list $\mathbf{C}$=$\{C^0, C^1, ..., C^n\}$
          Find largest cluster, $max\_cluster = argmax_i(\mathbf{size}(C^i))$
          Set the primitive support points $\mathcal{S}^k$ to the largest cluster, $\mathcal{S}^k = C^{max\_cluster}$
          Compute accurate plane coefficients from support points, $\mathcal{G}_p^k \leftarrow$ PCA over $\mathcal{S}^k$
          Compute bounding polygon $\mathtt{P}^k$, its area $\mathbf{A}(\mathtt{P}^k)$ and solidity $\mathbf{S}(\mathtt{P}^k)$
          if $\mathbf{A}(\mathtt{P}^k)$>$\mathbf{A}_t$ and $\mathbf{S}(\mathtt{P}^k)$>$\mathbf{S}_t$ then
              Add to primitive list, $\mathbf{G} \leftarrow \{\mathbf{G}, \mathcal{G}^k\}$
              increment number of primitives, $k \leftarrow k + 1$
          end if
          Remove support points $\mathcal{S}^k$ from $\mathcal{P}^{it}$, compute $\mathcal{P}^{it+1}$
      else
          Finish search for primitives, $cycle\_break = \texttt{true}$
      end if
      increment number of iterations, $it \leftarrow it + 1$
  end while
```

points associated to them, but each of those points can only be associated to one of the primitives. This assumption is a core component of the cascade processing because it simplifies and fastens the detection of the primitives. Let $\mathcal{S}^k$ be the point cloud containing the support points of primitive k, and $\mathcal{P}^k$ be the input point cloud in which the primitive was searched. The input point cloud for the search of the next primitive, $\mathcal{P}^{k+1}$ is obtained by removing the support points of primitive k:

$$\mathcal{P}^{k+1} = \left\{ \mathrm{p} \in \mathcal{P}^k \mid \mathrm{p} \notin \mathcal{I}^k \right\}. \tag{8.11}$$

Since every iteration of primitive detection will perform a search on a smaller point cloud and that the smaller the point cloud is, the fastest the search, it is expected that the cascade configuration is capable of significantly reducing the processing time. Section 8.4 will show results concerning processing time. Let a geometric polygonal primitive $\mathcal{G}^k$ have support points $\mathcal{S}^k$, support plane $\mathcal{G}_p^k = \begin{bmatrix} a^k & b^k & c^k & d^k \end{bmatrix}$ and bounding polygon $\mathtt{P}^k$. Algorithm 8.3 describes in detail the overall process of detecting several primitives from an input point cloud.

8.4 Results

In the previous sections the details of the extraction of geometric polygonal primitives were described. In section 8.3.1 the representation of the scene and the data compression potential is discussed. Then, section 8.3.2 shows how the support planes are detected using RANSAC and section 8.3.3 shows how inliers are clustered and the largest cluster is extracted. In section 8.3.6 the computation of geometric properties of the polygons is described. The computation of the bounding polygons is illustrated in section 8.3.5. Finally, section 8.3.7 describes the overall configuration of the detection algorithm into a cascade-like processing. This section will present some results concerning the detection of geometric polygonal primitives using the proposed methodology. Since in this section only the detection of geometric primitives is addressed, the presented results are given for the locations of both MIT sequences, instead of for the entire sequences. The detection of geometric properties is provided as a snapshot analysis of a given location. For the moment, no considerations are formed on how a representation should evolve over time. This is the reason why only the snapshot locations are analysed in this results section. The input data for all the algorithms whose performance is evaluated are the preprocessed point clouds described in tables 7.7 and 7.8. The following subsections will show several results.

8.4.1 Qualitative Analysis

A qualitative analysis is shown in Figs. 8.9 and 8.10, where the detected geometric polygonal primitives are shown for some locations of sequences 1 and 2, respectively. As it is possible to observe in both figures, the majority of the relevant planes are picked up by the algorithm. Note that in Figs.

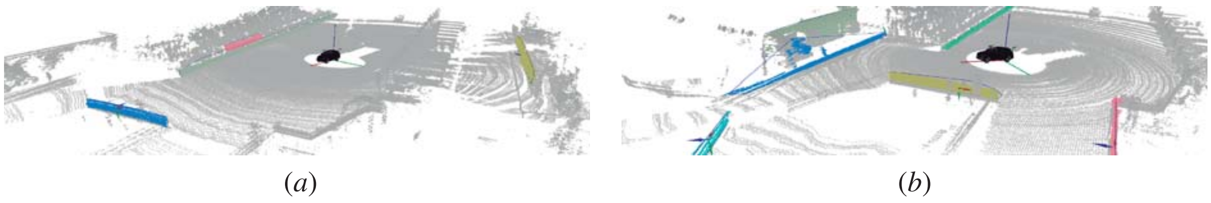


Figure 8.9: Detection of geometric polygonal primitives in the data sets of sequence 1: (*a*) location *C*; (*b*) location *D*.

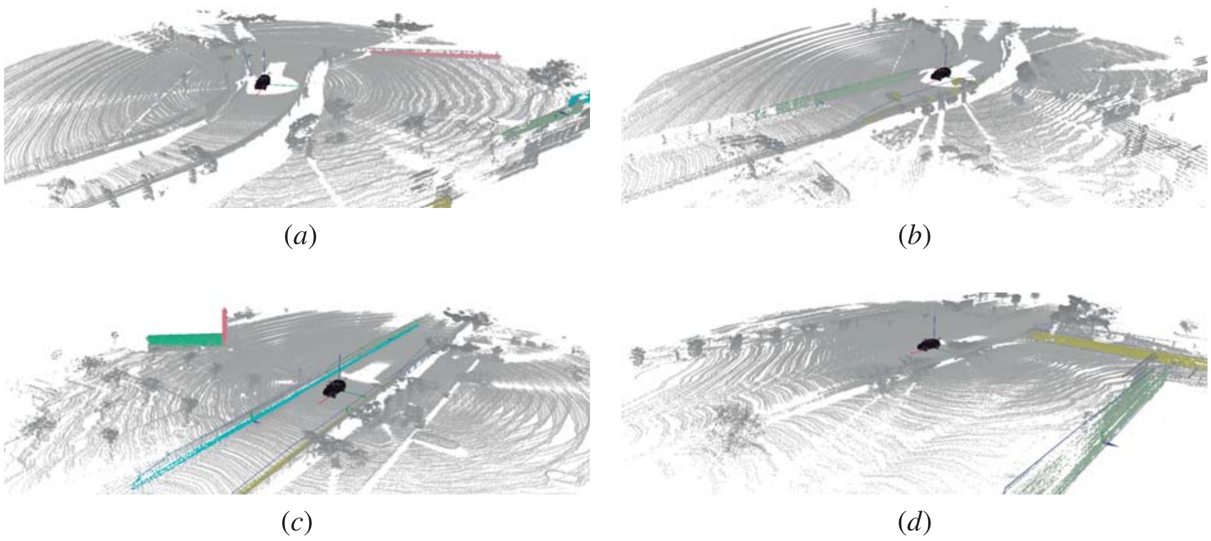


Figure 8.10: Detection of geometric polygonal primitives in the data sets of sequence 2: (*a*) location *F*; (*b*) location *G*;(*c*) location *H*;(*d*) location *I*.

8.9 and 8.10, the color of each primitive denotes the index of the primitive. From observing Fig. 8.9 (*a*) it is possible to see that the wall panel in front of the vehicle is marked in blue, while in Fig. 8.9 (*b*) the same panel is marked in yellow. This means that although the same planes are detected over consecutive locations, there is no guarantee that they are given the same label. In chapter 9 this problem will be addressed in depth. For now, what is relevant is that the proposed approach seems to be able to pick up a reasonable amount of planes in a great variety of situations.

8.4.2 Data Compression

If one agrees that the geometric polygonal primitives do show a good discriminative power for detecting the most relevant polygonal regions in a given point cloud, the next step would be to study the computational cost and accuracy of this representation. Regarding the computational cost, two metrics are important: memory size of the representation, which should be as small as possible; and processing time, which should also be reduced. One of the reasons for the choice of polygonal primitives was due to its data compression potential. A single polygonal primitive can represent thousands of range measurements, if all of them correspond to a wall for example. The memory size required by a geometric polygonal primitive is directly related to the number of vertices in the polygon (see eq. (8.2)). Figure 8.11 (*a*) shows a graph with the number of range measurements supported by each primitives versus the number of vertices of the bounding polygon. All of the computed polygonal primitives for sequence 1 are represented in the graph.

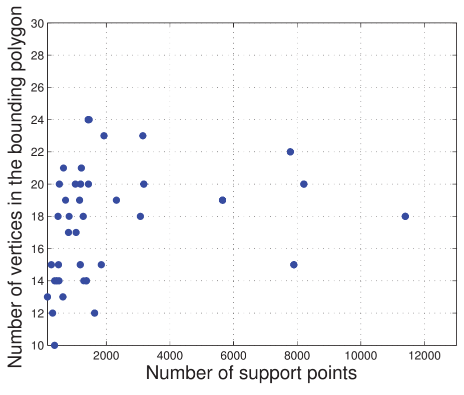
The first observation is that the typical compression ratio seems to be considerable. Let us assume, from Fig. 8.11 (*a*), that as a worst case scenario, on average, polygons have around 1000 support points for an average of about 30 vertices, Each support point is defined in the 3D space. Hence, the memory requirements for the support points (M_s) would be:

$$M_s = \underbrace{1000}_{support\ points} \times \underbrace{3}_{3D} \times \underbrace{4}_{typefloat} = 12Kb. \tag{8.12}$$

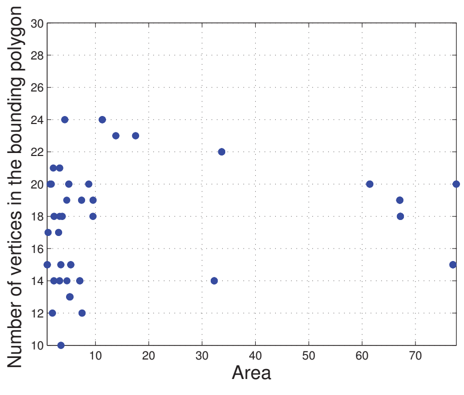
On the other hand, the memory requirements for a geometric polygonal primitive (M_p) are given by the support plane and the number of vertices represented in 2D space:

$$M_p = \underbrace{4}_{Hessian\ coef.} \times \underbrace{4}_{typefloat} + \underbrace{30}_{vertices} \times \underbrace{2}_{2D} \times \underbrace{4}_{typefloat} = 0.176Kb. \tag{8.13}$$

From this analysis it is possible to compute a worst case scenario compression ratio of $0.176/12 =$



(*a*)



(*b*)

Figure 8.11: Analysis of the geometric polygonal primitives data compression potential for all the detected primitives in sequence 1: (*a*) the number of vertices versus the number of support points; (*b*) the number of vertices versus the area represented by the polygon.

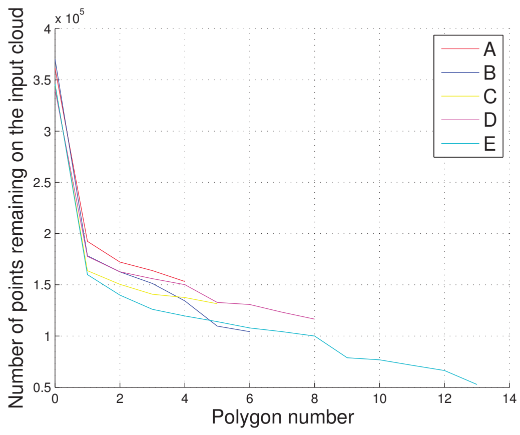
1.4%. In other words, in terms of memory size, geometric polygonal primitives take only about 1% of the size required by the raw range measurements. This is a very efficient compression ratio.

Another very interesting observation is that there seems to be an upper boundary to the number of vertices required to define the bounding polygon, and thus, to the memory size required by the representation. For example, there are primitives with 18 vertices that have from 500 to almost 12000 support points. Unlike in other approaches, for example triangulations, where the number of range measurements are directly related to the number of triangles and consequently to the memory size, in the case of the geometric polygonal primitives, since the contour of the polygon is the only information kept, the increase in the number of range measurements does not seem to directly affect the required memory size to store this representation. Figure 8.11 (*b*) shows a similar analysis where it is shown that the area of the bounding polygon also does not have a direct relation to the number of vertices in the polygon. In conclusion, geometric polygonal primitives show a very good compression ratio but also and perhaps more importantly, the memory size required by this representation is not directly related to the number of points that a given primitive represents. Furthermore, using this representation, it is possible to represent large areas of the scene with a very little amount of memory space.

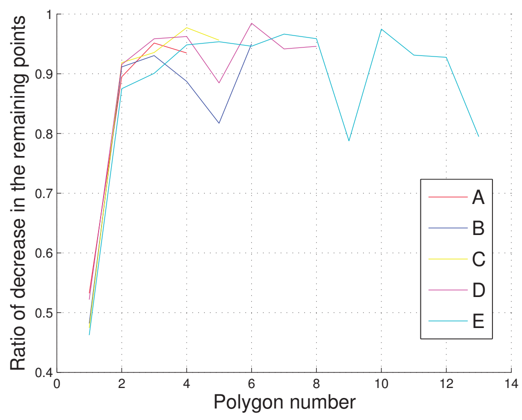
8.4.3 Computation Time

Having established the effectiveness of the polygonal primitives representation as far as data compression concerns, a second item is now analysed: the computation time. Computation time is very important since one of the expected applications is autonomous robot navigation. It is required that a representation is computed within a minimal amount of time, so that there is time for other algorithms to analyse the representation and act accordingly. In this scope, the expression to act accordingly means not only to perform a correct action but also to execute it in due time. As described in section 8.3.7 the setup for detecting several polygonal primitives is a cascade-like configuration. In other words, the algorithm will search for polygonal primitives on a given input point cloud. After the first primitive is found, all the range measurements that are explained by that primitive are removed from the input point cloud. The second primitive is then searched in a smaller point cloud and so on. Since the search for a primitive is done over a decreasing size point cloud, it is expected that the search becomes faster with the number of detected primitives. In Fig. 8.12 an analysis of the computation time of each primitive is displayed. Primitives with higher numbers are detected in posterior phases.

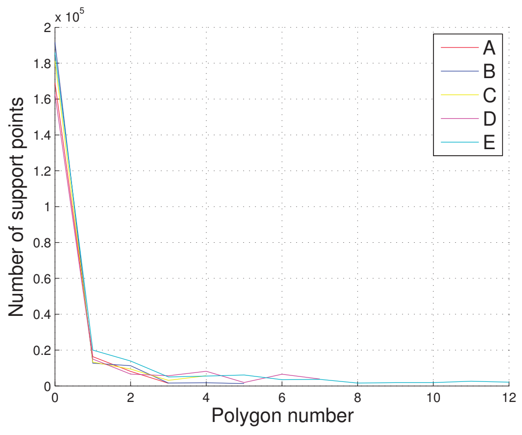
Figure 8.12 (*a*) shows the number of points remaining in the input point cloud as a function of the primitive number. Results are shown for all locations in sequence 1. The number of detected primitives varies from location to location. It is also possible to observe that, as expected, the number of remaining points decreases with the increase in the number of detected primitives. In all locations, the number of remaining points is monotonically decreasing. There is another very interesting observation to make with respect to Fig. 8.12 (*a*): the amount of reduction in the number of points is higher



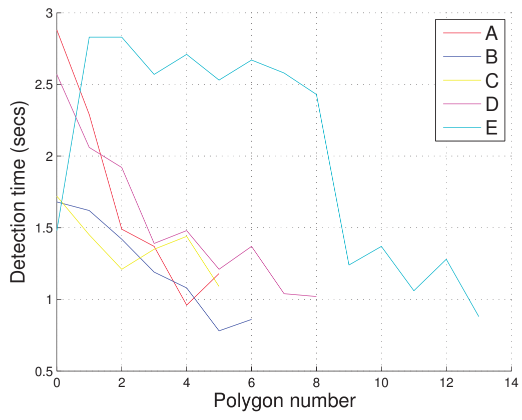
(*a*)



(*b*)



(*c*)



(*d*)

Figure 8.12: An analysis of the cascade processing: (*a*) a graph showing the number of points left to process for a given input point cloud, as a function of the index of the detected primitive; (*b*) the ratio of decrease of the remaining points from the input point cloud; (*c*) the number of primitive support points; (*d*) the time it takes to perform the detection of each of the geometric polygonal primitives as a function of the primitives index.

when there are few primitives. In other words, the rate of descent of the curve is higher for the first detected polygonal primitives. Figure 8.12 (*b*) shows the ratio of decrease in number of remaining points as a function of the polygon number. The decrease in number of remaining points for polygon number 1 corresponds to about 50% reduction. This value tends to increase (smaller reduction) as the number of primitives increases. Since early detections correspond to primitives with a low number, it is possible to state that the algorithm first finds primitives that account for a larger drop in remaining points. This is a very interesting conclusion because it is also expected that the detection time decreases with the decrease in number of points. Hence, since the algorithm tends to remove the largest portion of points at the early stages of the cascade processing, this means that the latter stages will also be more efficient to compute. The reason for this behaviour is the RANSAC algorithm. Because RANSAC will search for the larger consensus, it will most likely select planes that are supported by a greater number of points. This is shown in Fig. 8.12 (*c*). In this way, RANSAC tends to select first polygons with the largest amount of primitive support points. As a consequence, the largest decreases in the input point cloud occur early in the cascade, which in turn fastens subsequent detection stages of the cascade. The RANSAC suits the cascade processing perfectly, since it statistically tends to select the fastest strategy to decompose the input point cloud into geometric polygonal primitives. The detection time per primitive is shown in Fig. 8.12 (*d*). The detection time tends to decrease with the increase in polygon number, for the reasons that where previously reported. The optimization of the cascade is of course a statistical process, which derives from the random behaviour of RANSAC. Hence, it is not expected that the processing time decreases in all occasions, but only in the majority of them. In Fig. 8.12 (*d*) all locations except location *D* show a decreasing processing time. The maximum observed time to detect a primitive was about 3 seconds. This value seems not to be sufficient for real time processing but, as will be shown, they are quite good when compared to other surface reconstruction approaches.

8.4.4 Comparison to Other Approaches

As discussed in section 8.2, there can be several alternative representations to a given set of range measurements. In this section we will compare the proposed approach with three surface reconstruction methodologies.

The first is called ball pivoting algorithm triangulation. It was proposed in [Bernardini *et al.* 1999]. The BPA computes a triangle mesh interpolating a given point cloud. The principle of the BPA is very simple: Three points form a triangle if a ball of a user-specified radius touches them without containing any other point. Starting with a seed triangle, the ball pivots around an edge (i.e. it revolves around the edge while keeping in contact with the edge's endpoints) until it touches another point, forming another triangle. The process continues until all reachable edges have been tried, and then starts from another seed triangle, until all points have been considered. Although all range points are considered in the computation of the mesh, which accounts for the accuracy of the methodology, this fact also hampers the computational performance of the algorithm.

The second method used for comparison is an optimized version of a 3D triangulation, referred to as Greedy triangulation (GT). In this approach a method for fast surface reconstruction from large noisy data sets is proposed [Marton *et al.* 2009]. Given an unorganized 3D point cloud, the algorithm recreates the underlying surface's geometrical properties using data resampling and a robust triangulation algorithm, the authors claim to achieve near real time. For resulting smooth surfaces, the data is resampled with variable densities according to previously estimated surface curvatures.

One of the advantages of this method is that, since a greedy search is executed, it is expected to be faster than other standard triangulation approaches.

Finally, the third method is Poisson surface reconstruction. It was initially proposed in [Kazhdan *et al.* 2006]. In this approach, surface reconstruction of a point cloud with estimated normals is viewed as a spatial Poisson problem. The Poisson formulation considers all the points at once, without resorting to heuristic spatial partitioning or blending, and is therefore highly resilient to data noise. Unlike radial basis function schemes, a Poisson approach allows a hierarchy of locally supported basis functions, and therefore the solution reduces to a well conditioned sparse linear system.

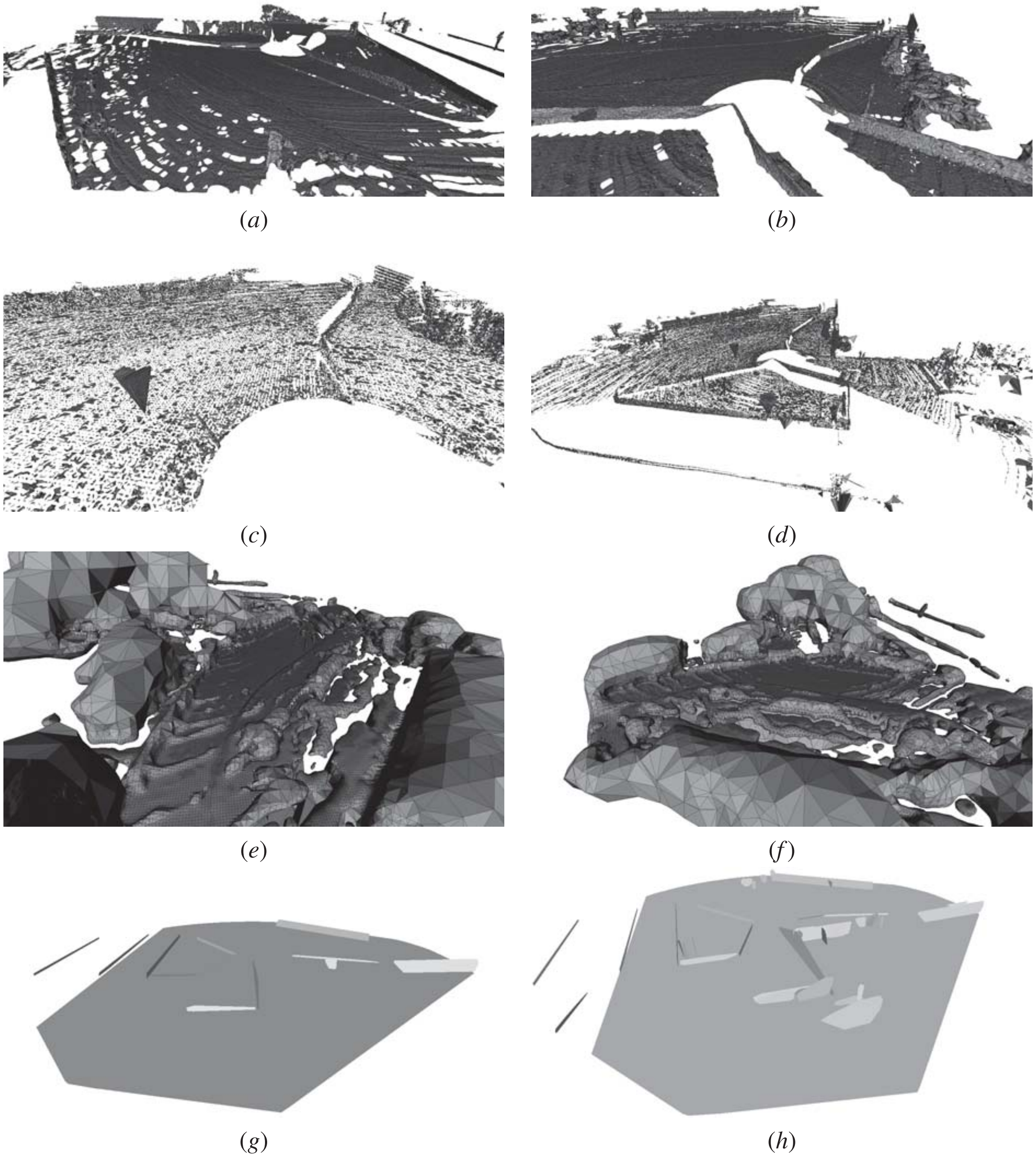
The three methods presented above will be compared with the proposed approach. Two different alternatives for the proposed approach are used: parameters set 1 and parameters set 2, Geometric Polygonal Primitives (GPP)1 and GPP2, respectively. In GPP1 the area and solidity thresholds (see section 8.3.6) are set so that only very large polygons are detected. Processing time is faster, since a lot of polygons are discarded but, on the other hand, accuracy or completeness of the scene representation is lost. GPP2 is a parameter set with a less strict threshold parameters and, as a consequence, provides a more accurate scene description at the cost of a higher computation time.

Figure 8.13 and shows qualitative results for all the described approaches. Location *E* of sequence 1 is used for all the images. In order to ensure a fair comparison, all approaches start from the preprocessed point clouds described in Tables 7.7 and 7.8.

Figure 8.13 (*a*) and (*b*) shows results obtained using the BPA method. It is possible to see that a very accurate reconstruction of the scene is made. Since the radius of the sphere was set to a high value (the default value) the ground surface is almost entirely connected by triangles. Since the method is not designed to cope with noisy range measurements, the surfaces are not smooth. This is an exhaustive method that performs searches on all data points. All input points are used and assumed to be precise, the algorithm just triangulates over them.

Figure 8.13 (*c*) and (*d*) shows results from the GT algorithm. Qualitatively, the algorithm seems to be less efficient than BPA in reconstructing the scene. Most of the ground is unconnected. This is also related to the parameters of the algorithm. However, several trials were made to try to improve by manually tuning the parameters. Results did not improve. It seems that the algorithm is more sensitive to holes in the data than BPA. As with the BPA, this approach is also very sensitive to noise since it does not consider that the input data points could have some noise. As a consequence, most of the surfaces that seem to be planar are represented by many triangles which are not coplanar, meaning the noise from the range measurements disturbs the effectiveness of the reconstructed triangles.

Figure 8.13 (*e*) and (*f*) shows results from the POIS algorithm. Results from this algorithm are the worst in terms of a visual analysis. The reason might be that POIS is not prepared to handle large holes in the data, since the algorithm assumes that the implicit surface belongs to a single object. In the current data set, this is not true. Results do not appear to be good because POIS tries to fit a single



(*a*) (*b*) (*c*) (*d*) (*e*) (*f*) (*g*) (*h*)

Figure 8.13: Qualitative comparison of several surface reconstruction methodologies in location *E* of MIT sequence 1: (*a*) and (*b*) BPA; (*c*) and (*d*) GT; (*e*) and (*f*) POIS; (*g*) GPP parameters set 1; (*h*) GPP parameters set 2;

implicit surface to the whole data set. Further tunning of the algorithm's parameters did not lead to better results in this data set. It is possible to observe that where the original data was dense, that is near regions of the ground plane that had no occlusions, the algorithm was able to model the data points properly. This algorithm however also has the advantage of fitting a surface to the data, process that accounts for noisy measurements. It also does not perform exhausting nearest neighbor searches as the other two triangulation methods do, which should make it faster that the previous two.

Finally, Figure 8.13 (*g*) and (*h*) shows results from the GPP. Parameters sets 1 and 2 are shown in (*g*) and (*h*), respectively. As expected, in Fig. 8.13 (*g*) the number of polygons used to represent the scene is small. Even though, it can be said that the most relevant polygons are part of the representation (see Fig. A.5 in appendix A, for the raw measurements of location *E*). In Fig. 8.13 (*h*), because the second parameters set is less strict in accepting polygon candidates, a lot more polygons are detected. At first sight, the representations obtained both by GPP 1 and GPP 2 seem to be very simplistic when compared to approaches BPA, GT and POIS. However, if one retains the core objective of the proposed reconstruction which is to be used by classical algorithms in autonomous robots, there are some interesting aspects. Planes in the scene, such as for example the ground plane, are automatically extracted as a single surface (one polygon supported by a single plane), instead of thousands of triangles with very different support planes. An autonomous robot that navigates based on a simple collision check between its body bounding box and all the primitives (polygons or triangles) in the reconstructed scene is a common approach to navigation. In this case for example, the very short number of polygons in the scene would be advantageous since it would enable a much fast collision check processing.

One very important component of the analysis of an algorithm is the computational performance of the algorithm, In other words, how long do the algorithms take to reconstruct the scene. Since robotic applications and driving assistance systems are the focus of this research, real time demands are an nonnegotiable requirement. Table 8.1 shows the computation times each algorithm took to reconstruct a scene. All locations from sequences 1 and 2 are presented.

The slowest algorithm is BPA. It takes on average about 600 seconds, i.e., 10 minutes to reconstruct the scenes. This is a very large amount of time, which definitely sets aside any possibility of real time reconstruction. On the other hand, the algorithm does seem to be the one that more accurately represents the scene. The amount of time taken by BPA should be related to the fact that nearest neighbor queries are computed for each of the input points. Nearest neighbor queries are very slow to process and the time they cost increases exponentially with the number of points to be tested, since all points are visited for each point in the point cloud.

In the case of the GT algorithm, although it uses a greedy search in order to optimize the nearest neighbor queries, it still takes a long time to reconstruct the scene. On average, the algorithm takes about 150 seconds (two and a half minutes). When compared with BPA, GT is in fact much faster. It is capable of reducing the processing time to a quarter. However, two and a half minutes are still not adequate for real time processing. Although the authors of [Marton *et al.* 2009] claim that the algorithm is capable of running in near real time, in this particular case of processing Velodyne point clouds this is not true. In fact, the algorithm is very distant from real time performance. The explanation for the high computational demand of both triangulation methodologies is related to the fact that all points are accounted for, i.e., no noise is considered, and especially because of nearest neighbor searches.

The POIS algorithm is faster than both triangulation methods. On average it takes around 60 seconds to reconstruct the scene. The algorithm relies on an octree for building the mesh. If the octree is set to have a smaller maximum depth, the computation time should decrease. However, as observed before, the reconstructed surface is not very accurate since the algorithm is sensitive to gaps in the 3D data. Hence, reducing the level in the octree would lead to a less accurate reconstruction. From the analysis of Fig. 8.13, it can be seen that POIS is the less accurate methodology.

The GPP algorithms are definitely the fastest ones. GPP 1 takes about 20 seconds to reconstruct

Table 8.1: Comparison of the computation time of several approaches for surface reconstruction on the MIT data sets.

Sequence/	Processing time (secs)				
Location	BPA [(1)]	GT [(2)]	POIS [(3)]	GPP 1 [(4)]	GPP 2 [(5)]
S1 *A*	659.0	154.0	63.2	16.3	27.3
S1 *B*	752.9	157.5	61.6	25.3	17.4
S1 *C*	488.2	156.3	56.3	13.5	49.4
S1 *D*	480.4	142.4	52.6	25.2	25.2
S1 *E*	558.8	149.0	57.9	47.4	58.1
S1 μ	585.9	151.8	58.3	25.5	35.5
S2 *A*	953.7	161.9	78.6	17.7	23.3
S2 *B*	1057.0	176.2	79.2	6.8	43.3
S2 *C*	63.8	71.7	35.7	7.3	18.9
S2 *D*	685.4	156.9	75.0	6.2	20.3
S2 *E*	614.2	165.5	77.8	18.2	10.2
S2 *F*	624.6	154.7	64.5	26.1	21.8
S2 *G*	456.3	135.5	64.2	9.2	29.9
S2 *H*	1107.7	171.3	81.1	42.8	28.5
S2 *I*	1315.5	204.3	79.3	14.8	88.8
S2 μ	764.2	155.3	70.6	16.6	31.7

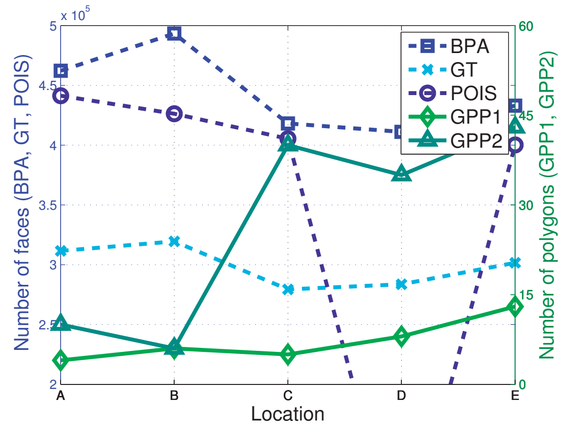
(1) BPA, proposed in [Bernardini *et al.* 1999], using the implementation from [CNR 2005].
(2) GT, proposed in [Marton *et al.* 2009], using the implementation from [Rusu & Cousins 2011].
(3) POIS proposed in [Kazhdan *et al.* 2006], using the implementation from [CNR 2005].
(4) Geometric polygonal primitives as proposed in this chapter, parameters set 1 (GPP 1). In parameters set 1, the area and solidity thresholds are set so that only very large polygons are detected.
(5) Geometric polygonal primitives set 2 (GPP 2), where the area and solidity thresholds are set so that even small polygons are detected.

the scene. As expected, in the case of GPP 2, since more polygons are detected, the average time is around thirty seconds. While it yields computation times that are not real time compliant, the GPP approach is the fastest of all tested algorithms. It takes only about a four percent (thirty to seven hundred seconds) of the time needed by BPA. Even considering the faster POIS algorithm, the proposed approach is capable of reconstructing the scene on average in half the time. Obviously, the processing of speed is directly related to the simplicity of the computed representation. Furthermore, the fact that RANSAC analyses only a small sample of points in the input point cloud means that not all input points are visited in order to reconstruct the scene, which is the case with the slower triangulation approaches. Also, the cascade configuration discussed in section 8.3.7 does help to speed up processing time. Regarding the real time demands, this matter will be further discussed in chater 9, where the updating of polygons is discussed. For now it is obvious that the initial detection of polygons in a scene is not possible in real time.

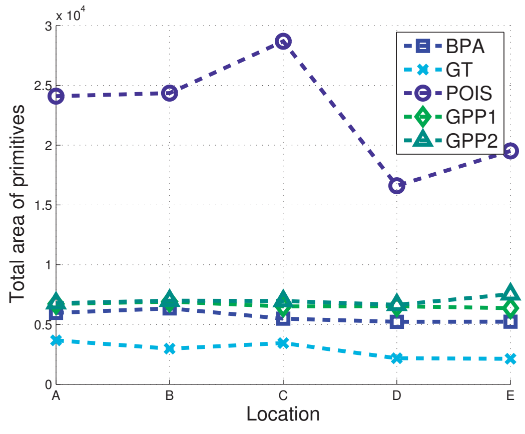
In Table. 8.1, it is possible to see that, for all approaches, the time taken to reconstruct location C of sequence 2 is much lower than in other locations. This is explained by the fact that this location has much less input data points, due to a large compression done by the preprocessing algorithms. Table 7.8 shows that unlike the usual 400K points, location C has only 166K points. Hence we conclude that the computation time of all algorithms is dependent on the number of input point, which was already an expected behaviour.

In Table 8.1 it is established that the proposed approach is much faster than other algorithms. However, when looking at Fig. 8.13 where qualitative results obtained by all approaches are compared, one may argue that the meshes generated by the GPP approaches are much more simpler than the other meshes, and that this is the reason why the GPP approaches are much faster, because they do not represent all the geometry that is underlying to the input point cloud. It is not possible to ascertain whether or not the meshes generated by GPP represent all the underlying geometry.

Fig. 8.14 (*a*) shows the amount of primitives each approach generates, i.e., triangles for BPA, GT and POIS, and polygons for GPP 1 and GPP 2. It shows that the number of polygons generated by GPP approaches is much smaller than the number of triangles generated by other approaches. This was expected. The idead behind the GPP approach is to represent the scene using only a few tenths or hundreds of polygons, instead of the hundreds of thousand triangle generated by other approaches. However, the number of primitives is not a good metric to describe the amount of scene represented by the algorithm. One triangle, if large enough, can represent the same surface described by many triangles. Since the objective is surface reconstruction, the area of the surface seems to be the best metric for assesing how much the amount of surface is covered by the algorithms. Hence, the total area of the mesh is used as the metric. It is computed as the sum of the areas of all primitives that belong to that mesh. Figure 8.14 (*b*) shows the total area of the meshes generated by the algorithms on each of the locations in scene 1. The POIS algorithm generates polygons with very large area, but this is because this approach fails to accuratly fit the surface in places where data is missing. When this



(*a*)



(*b*)

Figure 8.14: Comparison between the number of primitives (*a*) and total area (*b*) of the meshes generated by all surface reconstruction approaches. All locations of sequence 1.

occurs, the surface is extended with large blobs that are surely not coherent with the scenes structure but contrive to increase the total area of the mesh. However, the most important observation is that the GPP approaches generate polygons with roughly the same area as the triangulation approaches. Given this, it is possible to claim that the surface representations generated by GPP represent the same about of geometric information as do the triangulation approaches. In other words, GPP is much faster and it is not so at the cost of discarding or not detecting a large quantity of the scenes underlying geometric structure.

Since it is shown in the previous comparison that GPP approaches do not faulter in terms of amount of information generated, now a question may be posed in terms of accuracy. In other words, GPP approaches produce polygons with the same area as other approaches, and do it much faster. But is it at the cost of not detecting the scene with propper precision. This is a difficult question to answer. The main reason is that there is no ground truth for the MIT data sets. Actually, to the best of our knowledge, there are no 3D mesh data sets of road or city scenes with ground truth. In order to have quantitative results with respect to the accuracy of each algorithm the BPA algorithm is used. It is the longest taking algorithm, but at the same time it is the one that provides the more detailed mesh. Hence, the choice of using BPA as ground truth seems to be the most feasible one. The accuracy of the meshes generated by each algorithm are evaluated using a measure of distance (between meshes) from each of the approaches and the results from BPA.

There is a vast field of research in what regards establishing metrics to compute a similarity score between two meshes. In fact, the problem is not simple. For example, one can use the distance between the faces of the mesh, the vertexes of the mesh, the number of faces or points, the area or volume of the mesh, the difference between the orientation of the normals to the surfaces, and many other metrics. In the present case a variation of the Hausdorff distance is used. The Hausdorff distance was proposed by [Cignoni *et al.* 1998]. Let X and Y be two meshes. The Hausdorf distance between those meshes $d_{\mathrm{H}}(X, Y)$ is computed as:

$$d_{\mathrm{H}}(X, Y) = \max\{ \sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y) \}, \tag{8.14}$$

where $\sup$ and $\inf$ are the supremum and infinus, respectivelly. Since the Hausdorff distance is computed over a discrete set of points a Montecarlo sampling is used to select a statistically representative set of points for each mesh. These points are randomly selected from the entire mesh's surface, e.g., they can be vertexes, points liying on edges or on faces of the meshes. In this particular case a variation of the Hausdorff distance, called the one sided Hausdorff distance is used where only the only the $\sup_{x \in X} \inf_{y \in Y} d(x, y)$ part is computed, because we only wish to affer how distant is each approach to the ground truth and not the other way around. In this case, the X meshes are given by each of the algorithms and the Y mesh is supplied by the ground truth mesh BPA.

Table 8.2 shows the Hasudorff distance values obtained by GT, POIS, GPP 1 and GPP 2 using BPA meshes as ground truth. In fact, for every sampled point in the X mesh, the closest sampled point in the Y mesh is computed. By definition, the Hausdorff distance (max in Table 8.2) is the one sided supremum of that infinus distance. However in practice the mean or RMS measures are quite useful since they are much less sensible to outliers. These measures are also presented in Table 8.2.

The algorithm that obtains the best results is GT. The reason for this may be because of the

Table 8.2: Comparison of the accuracy of the several approaches using BPA results as ground truth and Hausdorff distance as metric.

Sequence/ Location	Hausdorff distance (meters)											
	GT			POIS			GPP 1			GPP 2		
	max	mean	RMS	max	mean	RMS	max	mean	RMS	max	mean	RMS
S1 *A*	11.7	0.15	0.41	14.0	1.39	2.98	7.6	1.02	1.71	7.6	0.87	1.55
S1 *B*	11.8	0.12	0.37	14.1	1.39	2.99	12.7	0.94	1.77	12.6	0.81	1.62
S1 *C*	12.7	0.18	0.44	13.9	1.06	2.59	8.9	0.87	1.54	8.9	0.69	1.32
S1 *D*	13.8	0.10	0.40	13.9	1.90	4.00	7.6	0.86	1.47	7.6	0.69	1.28
S1 *E*	12.5	0.14	0.49	14.0	1.42	3.03	14.0	1.25	2.56	14.0	1.11	2.39
S1 μ	12.5	0.14	0.42	13.9	1.43	3.12	10.2	0.99	1.81	10.1	0.83	1.63
S2 *A*	7.4	0.15	0.41	14.9	1.18	2.82	7.3	0.55	1.01	7.34	0.49	0.94
S2 *B*	6.7	0.07	0.27	14.5	1.09	2.73	10.9	0.59	1.18	10.9	0.48	1.04
S2 *C*	13.2	0.06	0.20	14.2	1.45	3.07	6.6	0.68	1.06	6.6	0.61	0.99
S2 *D*	14.1	0.11	0.33	14.6	1.19	2.88	6.6	0.52	0.86	6.6	0.42	0.75
S2 *E*	5.6	0.08	0.28	14.6	1.56	3.11	7.9	0.64	1.15	7.9	0.64	1.14
S2 *F*	12.4	0.05	0.21	13.9	1.66	3.66	5.6	0.57	1.03	5.7	0.53	0.99
S2 *G*	14.3	1.38	3.16	14.3	1.01	2.50	8.8	0.65	1.25	8.8	0.53	1.09
S2 *H*	9.5	0.09	0.41	14.7	0.97	2.62	11.3	0.57	1.22	11.3	0.54	1.15
S2 *I*	7.1	0.07	0.25	14.9	2.87	5.07	7.4	0.69	1.28	7.5	0.50	1.05
S2 μ	10.0	0.23	0.61	14.5	1.44	3.16	8.0	0.60	1.12	8.1	0.53	1.01

selection of the ground truth BPA. Both algorithms have a similar philosophy of analysing all points and triangulating using nearest neighbor queries. Hence, it is expected that results from these two approaches are very similar which is in fact the case, with an average error of 0.14 meters in sequence 1 and 0.23 meters in sequence 2. In the case of POIS the conclusions of the visual analysis are confirmed in this quantitative evaluation. POIS is the algorithm that less accuracy presents, as reported by the 1.43 meters mean distance in both sequences. The accuracy presented by the GPP 1 and GPP 2 approaches are about 0.8 and 0.65 meters, respectivelly. Altough these values are better than the POIS approach, they are still quite large. To have, on average, an error of over half a meter with respect to the ground truth in fact not very good. But it is interesting to further analyse where this error comes from. Figure 8.15 shows a graphical representation of the error for all of the approaches. For each approach, the points sampled of the output mesh are shown with color associated to the computed one sided Hausdorff distance of each point. A Red-Green-Blue colormap is used to code the distance. Red represents zero distance and blue maximum distance. In Fig. 8.15 (*a*), corresponding to the GT approach, almost all points have red color, resulting in low mean error. The POIS approach, represented in Fig. 8.15 (*b*), shows a lot of points in blue and green color, e.g., points whose minimum

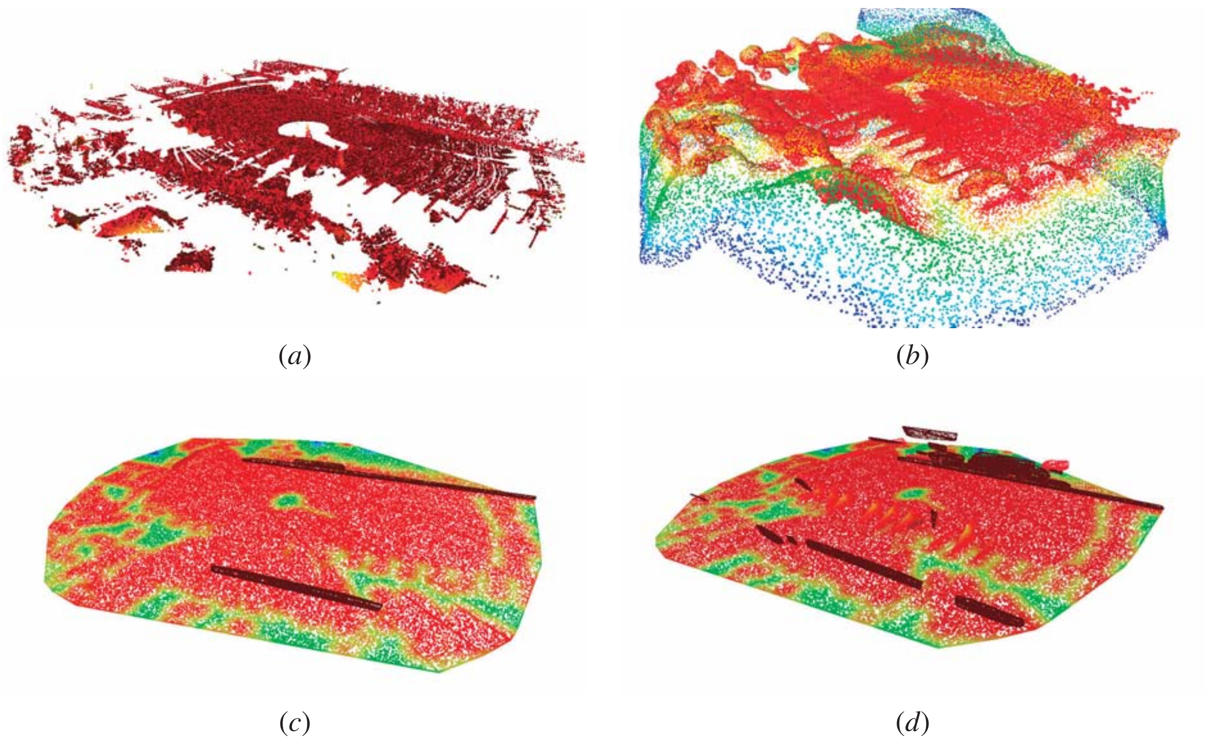

(*a*) (*b*) (*c*) (*d*)

Figure 8.15: Qualitative analysis of the one sided Hausdorff distance for the tested algorithms in location *C* sequence 1: (*a*) GT; (*b*) POIS; (*c*) GPP parameters set 1; (*d*) GPP parameters set 2; A Red-Green-Blue color map is used to code the distance. Red represents zero distance and blue maximum distance.

distance to the ground truth sampled points was very large. This is why POIS shows low accuracy values. In the case of the GPP approaches, 8.15 (*c*) and (*d*), some regions of the sampled points are more prone to have large error distances, while those in red seem to perfectly fit the ground truth mesh.

If a carefull observation is made on the regions where the GPP aproaches fail, the conclusion is that the regions that have large errors are those where range measurements do not exist because of obstacle occlusion. See Fig. A.3 in Appendix A to check where the vehicle was when this scene was captured and where are the occluded regions. Another interesting observation is that it is the polygon that represents the ground plane the one that shows a greater amount of error. Vertical polygons are not so much affected by large errors. The reason is that the ground plane is the one that suffers more from occlusion from other planes. It is because of this that GPP 2 shows greater accuracy when compared to GPP 1 as shown in Table 8.2: since GPP 2 detects more polygons, the overall weight of the polygon that represents the ground plane for the sampling of points and to the computation of the mean Hausdorff distance is smaller in GPP 2.

Figure 8.16 shows another example of what causes a low accuracy performance in the GPP approaches. In Fig. 8.16 (*a*), the ground truth for location *E* is shown. When these range measurements where taken, the vehicle was aproximatelly in the middle of the scene (see Fig. A.4, page 407), to the right of the scene, a road lies in between two walls. However, because of the height of the wall closest to the vehicle, the road surface is occluded and no range measurements of this region are collected. In the BPA approach (i.e., the ground truth), this region is left without information.

In the GPP approaches, Fig. 8.16 (*b*), the polygon that represents the road enconpasses this occluded region. The reason for this is that the convex hull was used to compute the bounding polygon (see section 8.3.5). When points from this occluded region are looked up in the sampled points from the ground truth mesh, there are no points near them, and the computed distances are large. Figure 8.16 (*d*) shows the sampled points in the GPP 2 and Fig. 8.16 (*c*) shows the corresponding closest points found in the ground truth sampled mesh. It is possible to see that points from the occluded road find correspondence only in the limits of that road, where ground truth data exists.

In conclusion, although GPP approaches do outperform the POIS algorithm, they show large accuracy errors with respect to the ground truth. The reason for this is that the BPA methodology, that was selected to serve as ground truth, does not perform interpolation over occluded areas, as the GPP approaches do. One may quibble about how exact is the ground truth given by BPA. In the example presented in Fig. 8.16 the occluded zone that was mentioned was indeed a road and the representation produced by the GPP approaches is correct. The choice of BPA could be discussed, but with the current data sets there was no alternative, since there is no real ground truth. So the first conclusion is that although the error shown by GPP approaches seems high, in reality it may be much smaller, it is caused because the ground truth is insufucient.

Second, the ground truth algorithm may also have problems in tackling phenomena like the laser

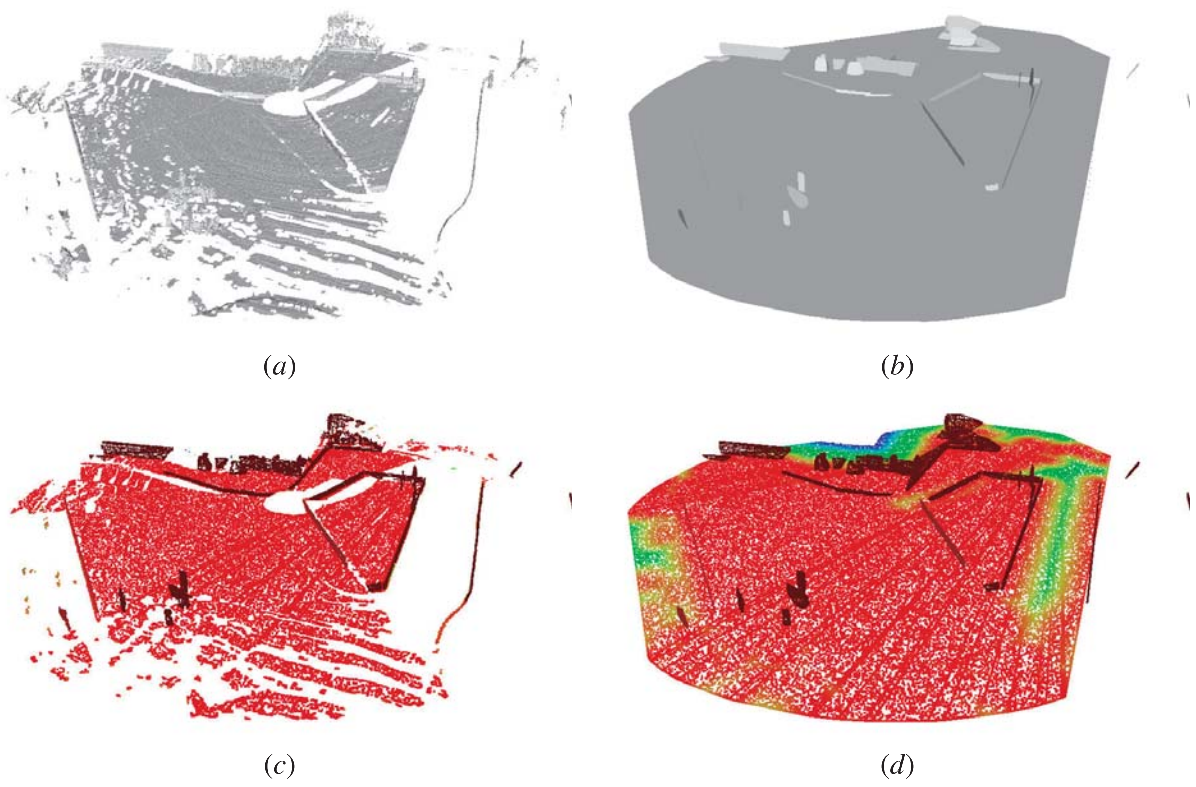


Figure 8.16: An example of the Hausdorff distance calculation for location *E*, sequence 1: (*a*) the ground truth, i.e., results from BPA; (*b*) the representation generated by GPP 2; (*c*) The points that were sampled in the ground truth mesh; (*d*) The points that were sampled in the GPP 2 mesh; A Red-Green-Blue colormap is used to code the distance. Red represents zero distance and blue maximum distance.

sensor accuracy. In Fig. 8.17 two examples are shown where a wall is reconstructed from 3D range measurements. Both the BPA and GPP representations are overlayed. It is possible to see that in the BPA representation, in grey, the surfaces of the walls are rugged. From the images taken from the cameras it can be seen that those surfaces are flat. Because laser range measurements have a limited accuracy, the input 3D points do not lie perfectly on the surface plane. In BPA, the masurements are considered to have no error, which cause the reconstruction to be rugged. The GPP approaches on the other hand, uses a PCA approach to fit a plane to the range measurements (see section 8.3.4), and represent the wall as a single surface, as it is in reality. Hence, the second conclusion is that GPP may show slight errors with respect to the ground truth that should not be accounted for, since the GPP is in fact more accurate that BPA.

It was shown that the results obtained by GPP in the Hausdorff distance comparison might not only have to do with problem in the approach, but also with limitations with the algorithm that pro-

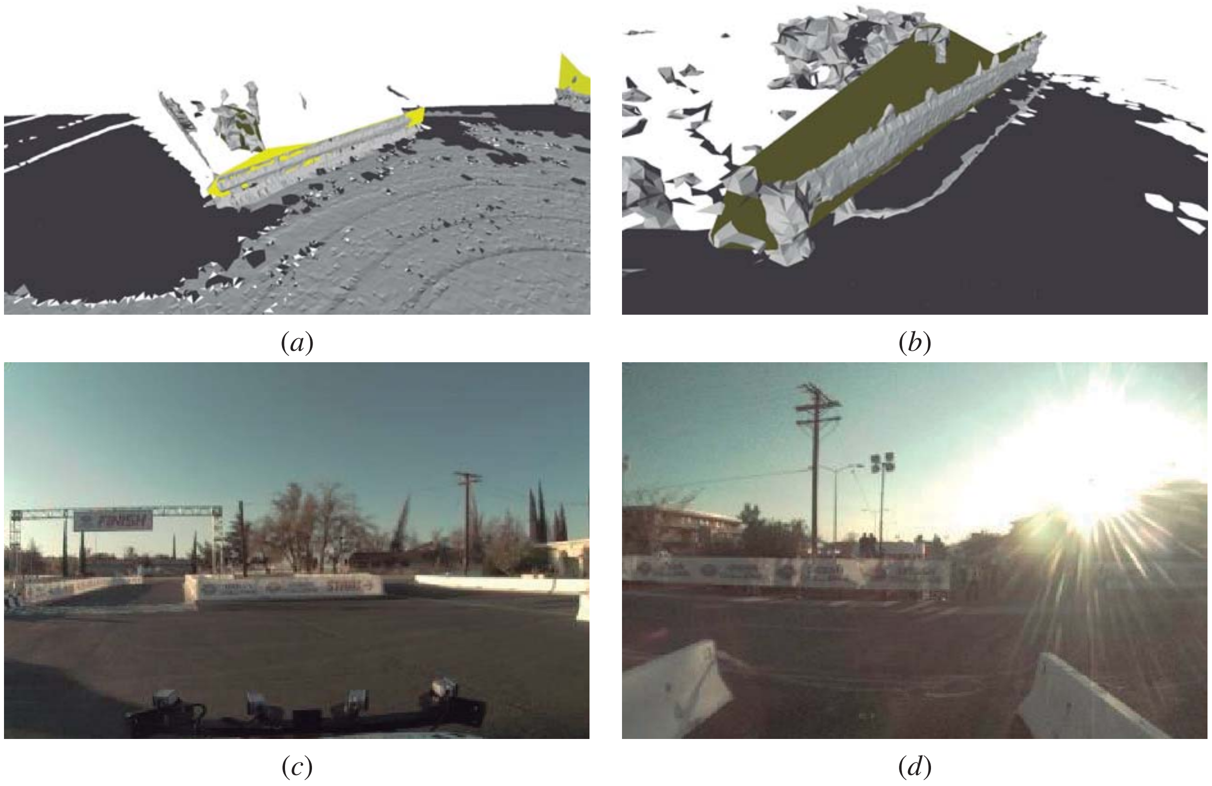


Figure 8.17: Comparison between the ground truth mesh (in grey) and the polygons generated by GPP (in yellow): (*a*) and (*b*) two different examples of wall surfaces; (*c*) and (*d*) images of the surfaces corresponding to (*a*) and (*b*), respectively.

vided the ground truth. When the reasons for having low accuracy results were analysed, it was found that they have mostly to do with the usage of the convex hull and are located especially in the polygon that represents the ground plane. Since the GPP methodology is somewhat flexible, it is possible to use three slight variantions of the GPP 2 algorithm whose results are shown in Table 8.2. These variantions are shown in Fig. 8.18. In Fig. 8.18 (*a*) results from the standard GPP 2 approach are shown. The first variation is to use the results discarding the polygon that represents the ground truth, since it is the one that shows the largest amount of errors (Fig. 8.18 (*b*)). In Fig. 8.18 (*c*) a second alternative is shown where the concave hull is used as a bounding polygon, instead of the convex hull. Finally, in Fig. 8.18 (*d*) the concave hull is used and the ground plane polygon is discarded. The motivation is to assess how much these two factors influence the lack of accuracy shown in Table 8.2. Table 8.3 shows the one sided Hausdorff distance results for these alternatives, with respect to sequence 1. The first alternative, discarded ground plane and convex hull (second column) shows a very high increase in the accuracy, from 0.83 meters to 0.13 meters of mean distance. In the second alternative, with ground plane and concave hull (third column), the mean distance is 0.53 meters. This explains that

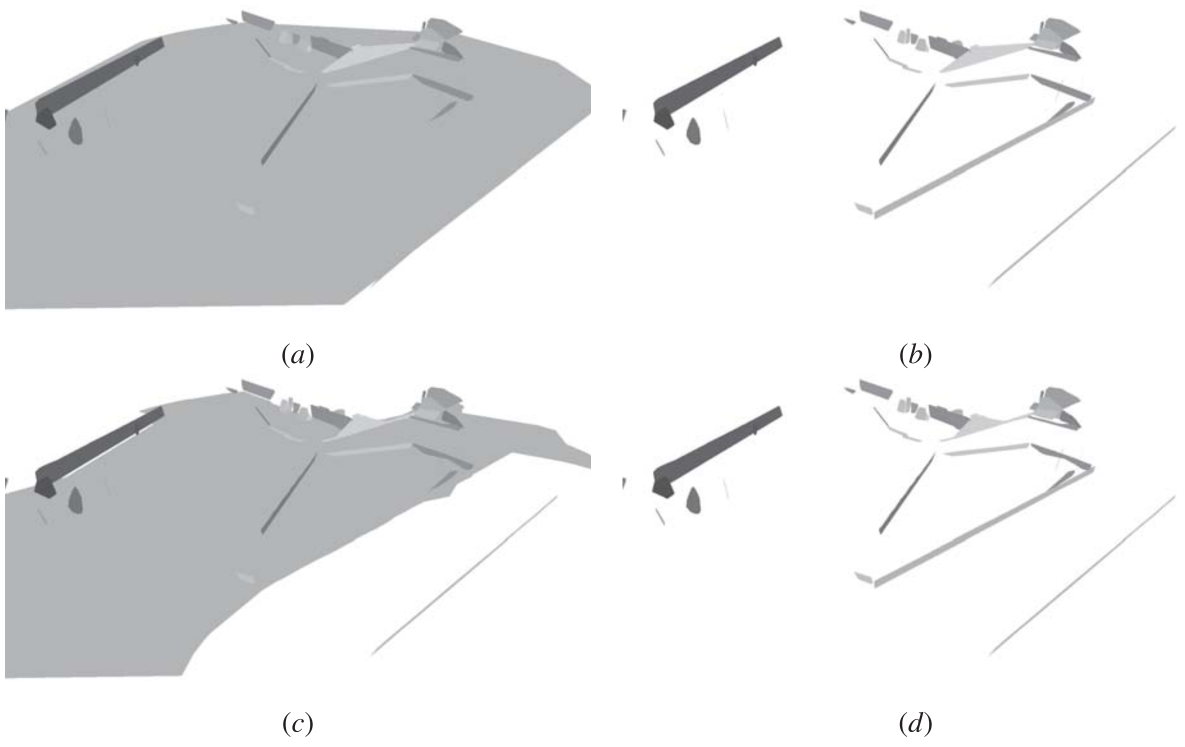


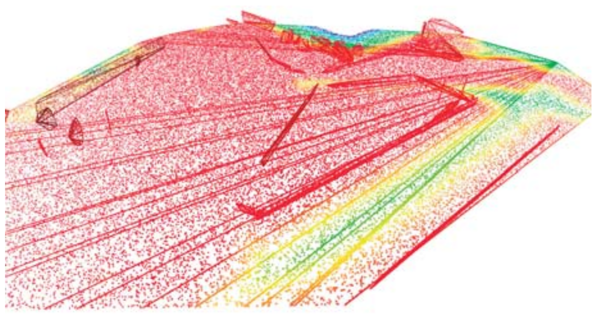
Figure 8.18: Representations obtained when using alternatives for the GPP 2 method for location *E*, sequence 1: (*a*) the standard GPP 2, with ground plane and convex hull; (*b*) discarded ground plane, convex hull; (*c*) with ground plane, concave hull; (*d*) discarded ground plane, concave hull.

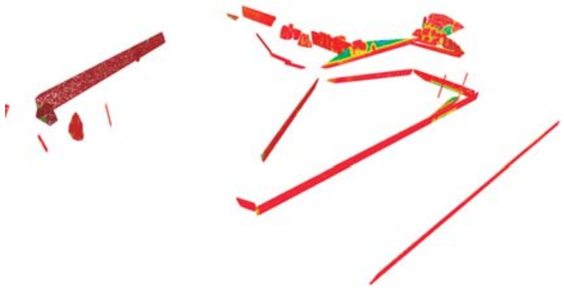
it is the inclusion of the ground plane polygon that most influences the lack of accuracy, but that the convex hull is also responsible for some of the errors. In the third alternative both are combined: a concave hull is used and the ground plane is also discarded. Note that the concave hull methodology is used for computing the bounding polygon of all polygons, not just of the ground plane polygon. The results shown by this third alternative are very good, with 0.1 meters of average error. In conclusion, if the representation accuracy is an inportant factor, concave hulls should be used, instead of convex. Also, if the ground plane is not required, its removal benefits the overall accuracy of the representation.

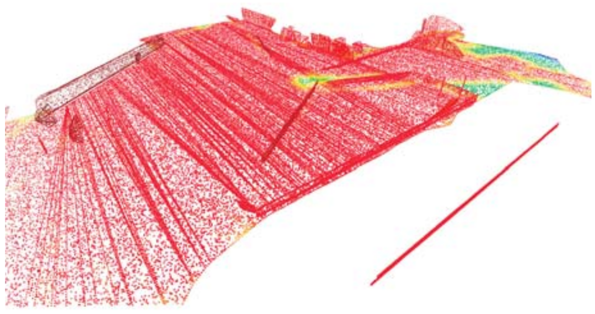
Figure 8.19 shows a visual analysis of the Hausdorff distance errors for these variations of GPP 2. It is possible to observe that regions with error, e.g., in blue and green, decrease considerably when the concave hull is used, but in particular when the ground plane polygon is discarded.

Table 8.3: Comparison of the Hausdorff distance accuracy of the GPP 2 approach using: the standard approach, convex hull and ground plane included (also in Table 8.2); the convex hull with no ground plane included; the concave hull with ground plane; and the concave hull without ground plane.

B. Polygon Ground plane	GPP 2 Hausdorff distance (meters)											
	Convex Included			Convex Not included			Concave Included			Concave Not included		
	max	mean	RMS	max	mean	RMS	max	mean	RMS	max	mean	RMS
S1 *A*	7.6	0.87	1.55	1.8	0.15	0.26	6.8	0.71	1.25	1.2	0.13	0.19
S1 *B*	12.6	0.81	1.62	1.5	0.11	0.19	12.6	0.53	1.09	1.1	0.08	0.14
S1 *C*	8.9	0.69	1.32	1.9	0.16	0.29	6.6	0.52	0.99	1.9	0.12	0.22
S1 *D*	7.6	0.69	1.28	2.2	0.14	0.26	7.3	0.59	1.13	2.1	0.11	0.21
S1 *E*	14.0	1.11	2.39	1.7	0.10	0.19	8.8	0.32	0.81	1.4	0.08	0.14
S1 μ	10.1	0.83	1.63	1.8	0.13	0.24	8.4	0.53	1.05	1.5	0.10	0.18


(*a*)


(*b*)


(*c*)

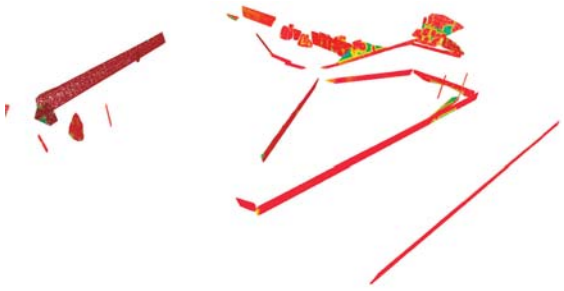

(*d*)

Figure 8.19: Results from the Hausdorff distance obtained when using alternatives for the GPP 2 method for location *E*, sequence 1: (*a*) the standard GPP 2, with ground plane and convex hull; (*b*) discarded ground plane, convex hull; (*c*) with ground plane, concave hull; (*d*) discarded ground plane, concave hull. A Red-Green-Blue color map is used to code the distance. Red represents zero distance and blue maximum distance.

8.5 Conclusions

This chapter has described in detail a new algorithm that performs surface reconstruction from point clouds. The representation is based on geometric polygonal primitives. Polygon support planes are detected using RANSAC and fitted using PCA. Then, a bounding polygon is computed using convex or concave hull algorithms. Finally, points that belong to a polygon are discarded from the subsequent searches, in a cascade like configuration.

The proposed representation based on sets of polygons is shown to have very good compression capabilities. The cascade configuration speeds up the detection of polygons in a scene.

The performance of the presented method is compared with three well known surface reconstruction methods. Results have shown that the proposed approach is much faster to reconstruct the scene, which makes it the best candidate for real time scene reconstruction applications. It was also shown that the better performance is not granted at the cost of a less representative scene representations. In terms of accuracy, at first sight the proposed approach seems to yield low accuracy. However, in a more detailed analysis, it was shown that the proposed approach can be easily configured, if so desired, to show very good accuracy. If a very accurate scene representation is required, concave hull polygons should be used. If, on the other hand, accuracy is not so important (in fact it was shown that sometimes the GPP is actually better than the ground truth), convex hull should be the choice to compute the bounding polygon. If the application does not require the detection of the ground plane, its removal may also increase the overall accuracy.

Chapter 9

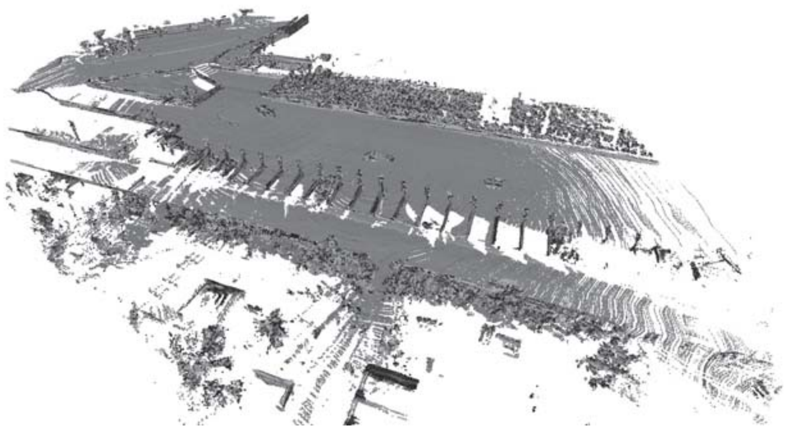
Geometric Scene Refinement

This chapter proposes a new algorithm for performing the refinement of the polygonal primitive based geometric scene representation. As will be shown, this mechanims is very effective, which enables a scene representation to be computed in even less time. The chapter describes how the scene representation evolves dynamically, changing the description of the scene as new 3D data is received. Section 9.1 introduces the problem. In section 9.2, the alternative methods for surface reconstruction are analysed. Then, section 9.3 presents the proposed approach. Finally, results are presented in section 9.4, and conclusions in section 9.5.

9.1 Introduction

Chapter 8 has described in detail the algorithm that is proposed to perform geometric scene reconstruction. The algorithm was compared to other surface reconstruction methods and it is shown that the geometric polygonal primitives are much faster than all other methods. However, despite being the fastest amongst all tested algorithms, the time they take to compute a scene representation is still far behind real time performance. In this chapter, a methodology for refining the scene representation over time is proposed.

We will use the distinction between the terms scene and scenario. Let scenario refer to a particular location that should be reconstructed. It can be a city, a road or anything else. By scene, we refer to the portion of the scenario that is viewed by the vehicle at a particular time. Hence, a scenario is the whole place to be reconstructed, and while the vehicle is travelling through the scenario it views many scenes. Of course that different objectives or necessities may require different sizes of scenarios. For example if a vehicle does not plan its movement more than twenty meters in front of it and it views a scene of over fifty meters around it, in this case then the instantaneous scenario would be composed of a single scene. Although one could argue that there is no need for large scenarios, since long term plans are prone to change often, the ability to store in memory (whatever the representation may be) a large portion of the environment should be helpful. The size of the scenario should be defined according to the requirements of the vehicle and, therefore, of the application at hand, but the focus here is to develop a scene representation mechanism that can accommodate a large scope of applications, those that require small sized scenarios but also those that necessitate large size ones. Since the difficulty here lies in the computation of large sized scenarios, we will focus on this aspect. If, for a given application, small size scenarios are recommended, it is straight forward to obtain a small size scenario from a large size one. In other words, the objective of this section is not to address the question of which should be the size of a scenario but instead to focus on the possibilities that emerge from fast computing and low memory representations of a given scenario provided by the geometric polygonal primitives, and whether they can contribute for easing the process of generating


(*a*)

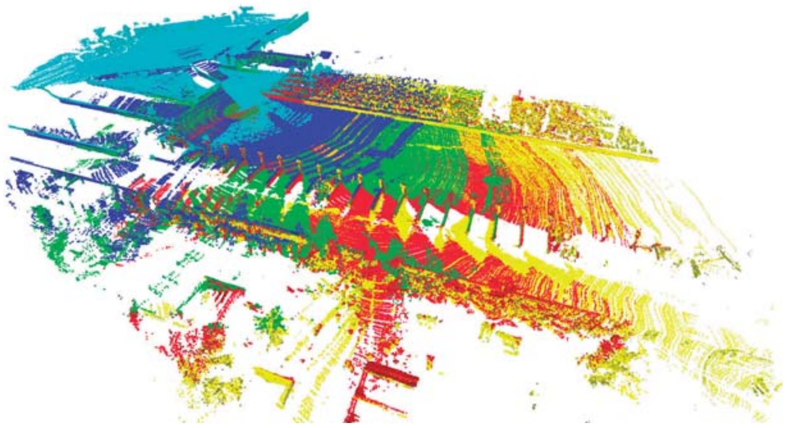

(*b*)

Figure 9.1: Raw 3D points for all locations of scenario 1: (*a*) accumulated point cloud; (*b*) same as in (*a*) but data points from locations are color coded: location *A* (yellow), *B* (red), *C* (green), *D* (blue) and *E* (cyan).

large scale scenarios.

For testing and performance evaluation purposes, we consider to be scenarios each of the sequences presented before in chapter 7.3. In the previous section, only particular locations of each of the scenario/sequences where used to evaluate the performance of the scene reconstruction algorithms. In this section, remind that scenario 1 does not only have five input point clouds in locations *A* through *E*. In fact each pair of consecutive locations has ten seconds in between them, and there is a 400K points input point cloud every second (see sequence accumulated columns in Tables 7.5 and 7.6). For comparison, evaluation and visualization purposes, however, only the point clouds that result from the accumulation of the annotated locations are displayed. It is quite slow to use all the input point clouds in sequence 1, this is why only the locations are considered for computing results. Figure 9.1 (*a*) shows an accumulated point cloud obtained by merging all point clouds from locations *A* through *E*, of sequence 1. In Fig. 9.1 (*b*), the 3D points are color coded according to the location from where they where taken. As can be seen, there is a very significant degree of overlap between consecutive locations. Of course the degree of overlap depends of the vehicle's speed, but if one considers that in between each two locations there is a gap of one second, it may be assumed that for standard vehicle speed there is always a considerable amount of overlap between consecutive input point clouds. In this sense, consecutive point clouds represent, to the extent of their overlap, which is large, exactly the same objects in the scene. Note also that it is considered that vehicle egomotion is being provided, that is, it is always straight forward to transform each input point cloud or reconstructed scene to a global reference system which is why it is possible to show images like those in Fig. 9.1.

9.2 Related Work

This section cannot be addressed as a typical state of the art. The reason is that, to the best of our knowledge, there are no alternative methods proposed in the literature to refine scene reconstructions. As an alternative, in this section we analyse the theoretical approaches that could be used in the refinement of a scene representation.

At first sight, there are three alternatives for performing scenario reconstruction:

- Store raw measurements and reconstruct in the end; store all input point clouds and accumulate into a super point cloud. Then, use surface reconstruction methods on the super point cloud, i.e., over the range measurements of the whole scenario;

- Reconstruct incoming point clouds and fuse all partial reconstructions; perform scene reconstruction for each input point cloud, create the reconstructed scenario from the accumulation or fusion of the several scene representations produced for each input point cloud;

- Reconstruct with the first point cloud and then make the representation evolve as new point clouds arrive; use the first input point cloud to produce a partial scenario representation and then use subsequent input point clouds to refine or update the scenario representation.

The first possibility is the most immediate approach. Since there are well known surface reconstruction algorithms and that these make use of a single point cloud to reconstruct the surface, this approach merely merges all input point clouds into a single one, the accumulated point cloud. Then, standard surface reconstruction algorithms may be used using the accumulated point cloud as input. This method has several problems: first, it only computes the surface reconstruction after all points are accumulated. In the example of sequence 1, the scenario reconstruction could only begin after the input point cloud of location *E* is received and accumulated. This would be a problem since one of the objectives, as stated in the beginning of this chapter, is that some kind of representation of the scene is made available to the vehicle in a short period of time. Only in this way it is possible to expect that the trajectory planning, obstacle detection, or pattern recognition algorithms may use the representation as input. If only at the end of the scenario it is possible to start computing its reconstruction, then this is typically what is called post processing and is not suited for usage in real time applications. Furthermore, the surface reconstruction methods are slow to process just a single input point cloud, as was proved in section 8.4, and the time taken grows exponentially with the number of points in the input point cloud. Therefore, using standard surface reconstruction techniques in the accumulated point cloud would take a very long time. Figure 9.2 shows an example of scenario reconstruction (with Ball Pivoting Algorithm (BPA)) using as input a point cloud that accumulates locations *C D* and *E* of sequence 1. Just to have an idea, the reconstruction of the accumulation of just these three point clouds took 7776 seconds.

The second alternative is to compute a scene reconstruction for each of the locations, and afterwards to merge all the reconstructed scenes into the scenario representation. Taking the example of Fig. 9.2, if location *C*, *D* and *E* are computed independently, the overall process would take $488.2 + 480.4 + 558.8 = 1487.4$ seconds, based on the data collected from Table 8.1. Although it is surely faster to reconstruct each location independently when compared to accumulating all point and reconstructing from the accumulated point cloud, there should be some computational overheads with the merging process. However, for now, let us assume that the merging process has zero computational cost. In this case, the time required to reconstruct the sequence 1 scenario would be equal to sum the time taken to reconstruct each of the scenes.

Figure 9.3 shows the reconstructed scenarios obtained using this alternative, for all tested algorithms. The geometric structure of the scenario is captured by most of the algorithms. The only problem is that algorithms take too much time to do so.

Using Table 8.1 as a reference it is possible to estimate the following computation times for each of the algorithms, if this scenario reconstruction alternative is used. These estimates are presented in Table 9.1. Two alternatives are shown: using only locations, i.e., the values are the sum of those presented in Table 8.1; or using all input point clouds, an estimate is made from the average processing

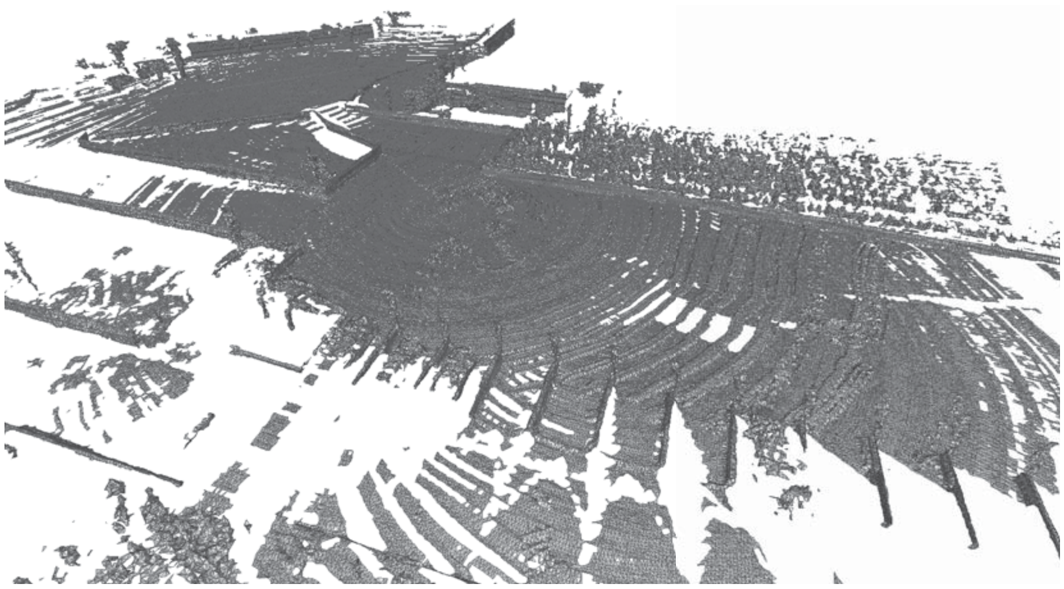


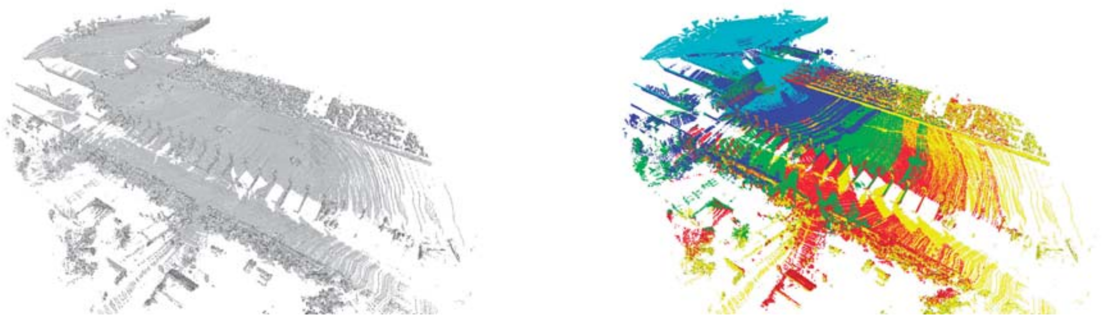
Figure 9.2: BPA surface reconstruction over the accumulated point cloud of all the locations in sequence 1.

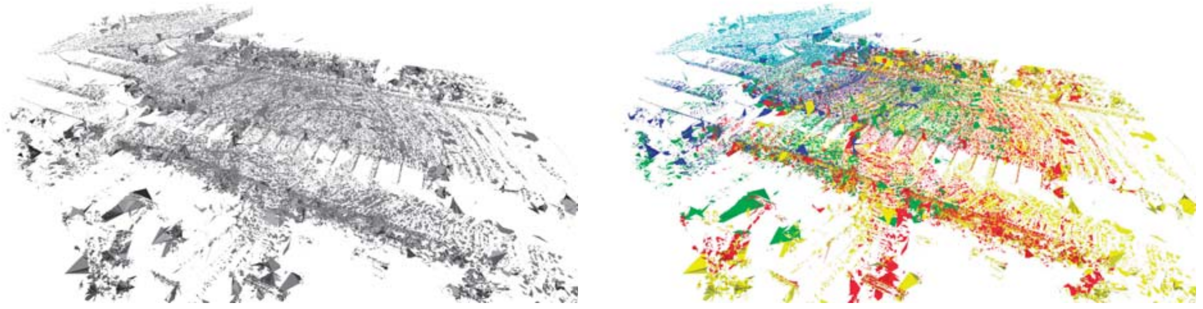
Table 9.1: Comparison of the computation time of several approaches for the reconstruction of the whole scenarios of the MIT data sets, when an approach of reconstructing each point cloud and then merging reconstructed meshes is used.

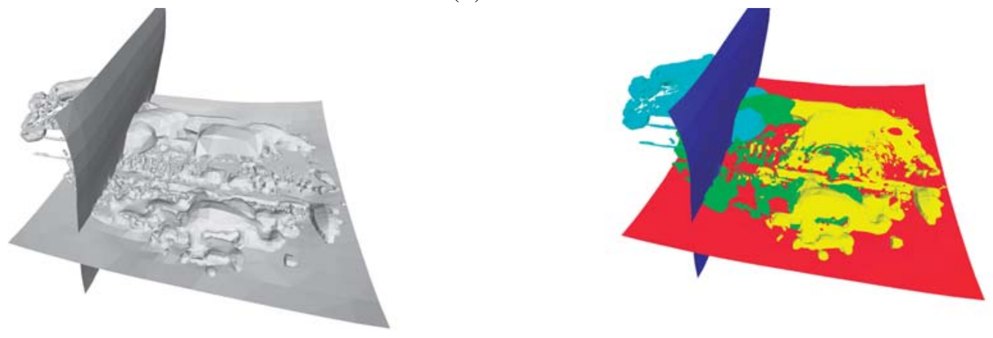
	Processing time (secs)				
Sequence	BPA	GT	POIS	GPP 1	GPP 2
S1 ($A + B + C + D + E$)	2941	759	291	127	177
S1 (all point clouds)	23426	6072	2332	1020	1420
S2 ($A + B + C + D + E$)	6878	1398	635	149	285
S2 (all point clouds)	61136	12424	5648	1328	2536

time per point cloud and the number of point clouds in the sequence. From Table 9.1 it is possible to see that it is not feasible to use none of these approaches to reconstruct the scenario. Even with the GPP 1 approach, which is the fastest, to reconstruct all locations takes 127 seconds or 1020 seconds if all input point clouds are used. Considering that the whole sequence 1 is only twenty seconds long, it seems impossible to achieve real time performance with this alternative.

Another disadvantage of this method is that, given that there is a significant overlap between neighboring input point clouds, each scene reconstruction is most of the times reconstructing portions of the scene that have already been reconstructed in previous locations. This occurs in all algorithms, since that each scene is reconstructed independently without knowledge about the other reconstructions. Some examples can be seen in Figure 9.4 where it is shown that using the partial reconstruction of each scene alternative leads to having primitives (triangles or polygons) representing the same


(*a*)


(*b*)


(*c*)

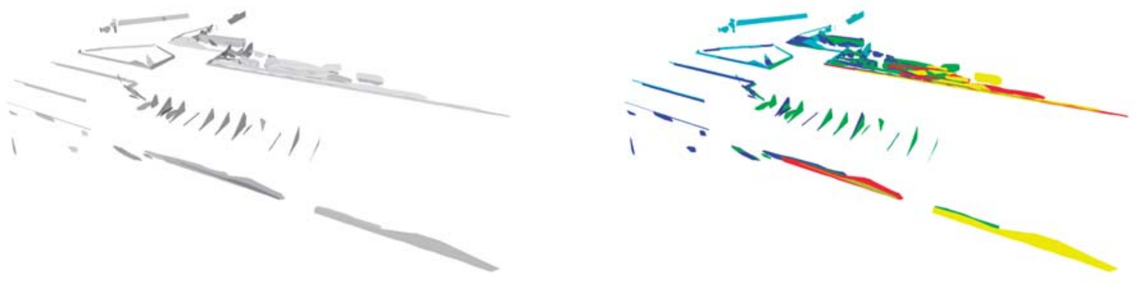

(*d*)

Figure 9.3: Accumulation of scene reconstructions of the location *A* (yellow), *B* (red), *C* (green), *D* (blue) and *E* (cyan), for the generation of the reconstructed scenario. Reconstructed scenario (*left*) and scenario with color coded scene primitives origin (*right*); (*a*) BPA;(*b*) GT;(*c*) POIS;(*d*) GPP 2, shown without the ground plane for an easier visualization.

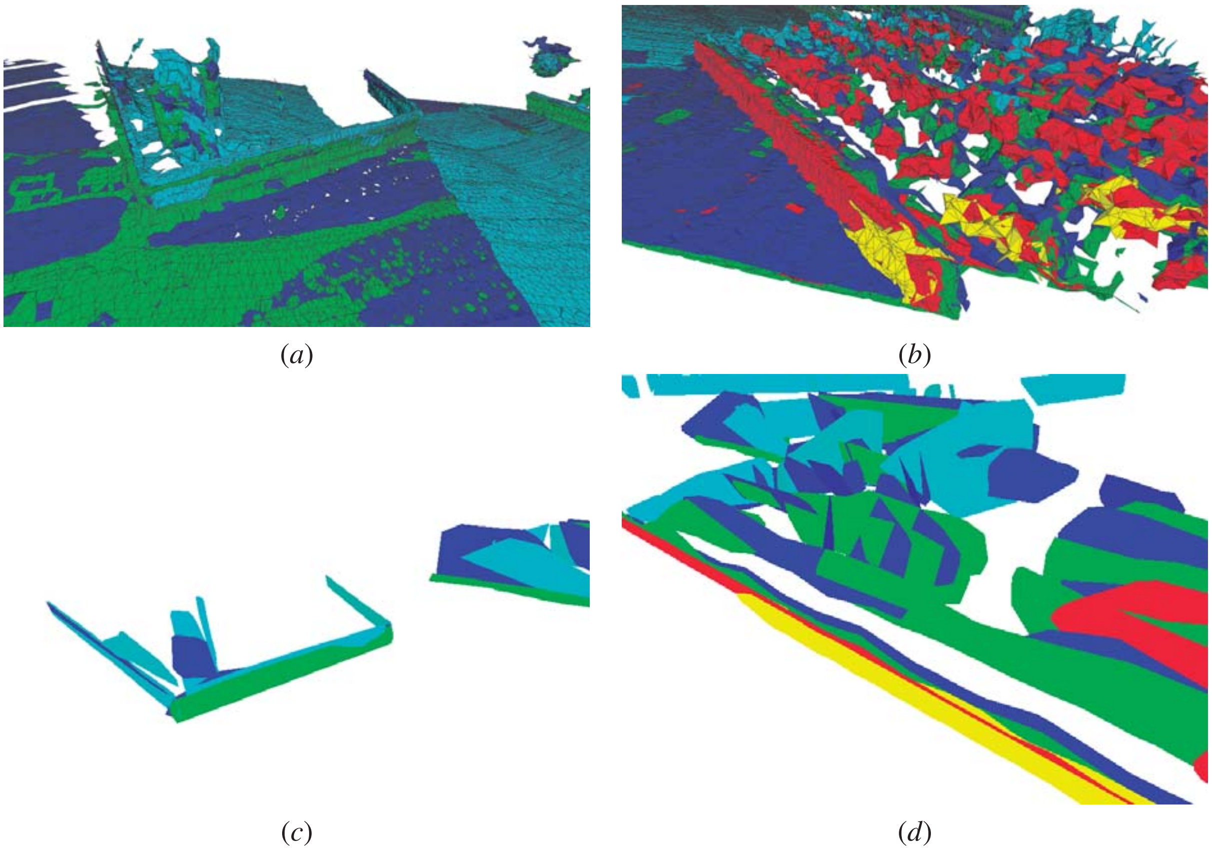


Figure 9.4: Details of images shown in Fig. 9.3: (*a*) and (*c*) BPA and GPP 2 reconstruction of an object, respectively; (*b*) and (*d*) the same for another object. Color code: location *A* (yellow), *B* (red), *C* (green), *D* (blue) and *E* (cyan).

objects. Hence, this alternative of reconstructing each scene, although faster than the previous, has the problem of waisting computational resources by executing multiple reconstructions of the same objects in the scene. The second major problem is the merging of the scene representations into a global scenario representation.

Even if it is considered that this process does not take a lot of time, conceptually it is not straight forward how several meshes are merged into a single one. The results presented in Figs. 9.3 and 9.4 use only the most basic alternative, which is to add all primitives in several meshes into a single mesh. However, problems like intersecting faces or duplicated vertexes could occur and should be dealt with. Figure 9.5 shows the problem of intersecting faces. In Fig. 9.5 (*a*) a detail of the BPA reconstruction obtained after accumulating the point clouds of locations *C*, *D* and *E* is shown. In 9.5 (*b*), the same detail is shown when using the alternative method where same locations are reconstructed independently and then local meshes are merged.

Finally, the third alternative is to perform some kind of update to the current scenario repre-

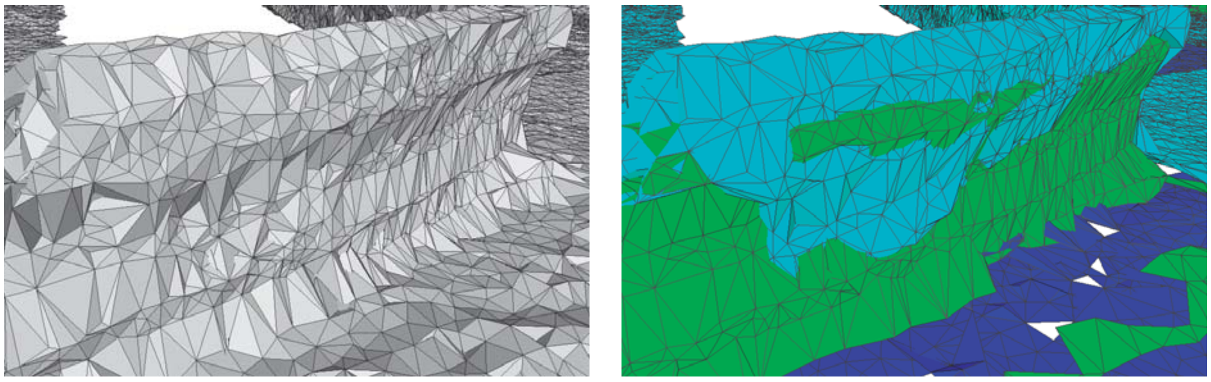

(*a*) (*b*)

Figure 9.5: Comparison of alternatives to scenario reconstruction: (*a*) reconstruction after accumulating all the point clouds; (*b*) reconstructing each point cloud and then merging local meshes. Color code: location *C* (green), *D* (blue) and *E* (cyan).

sentation as new input point clouds arrive. When the first input point cloud is received, a rough representation of the scenario is built using the available information. This representation is then refined and updated as new data arrives. Conceptually, this is the most interesting approach to scenario reconstruction. Since a representation of the scenario is built from the first point cloud, this representation, even if not very complete at this moment, can be of use for a vehicle to navigate. With this approach it is possible to always have the best available scenario representation and to make it available to other software layers. Another advantage is that, since the first created representation is refined, there is knowledge about what objects have been previously reconstructed and so it can avoid reconstructing them again, waisting computational resources. To the best of our knowledge, no algorithms have been proposed that perform scenario reconstruction under this paradigm. The next sections will discuss the methodology that is proposed to execute the update of the scenario using the geometric polygonal primitives.

9.3 Proposed Approach

It was shown in section 9.2 that there are several methodologies possible to perform scenario reconstruction. From all the methods, the one that certainly seems more interesting is the third, where an initial scenario representation is updated or refined when new sensor data arrives. To the best of our knowledge, there are no approaches that can handle the refinement of a given scenario representation in real time. This section will describe how the proposed representation, based on a set of geometric polygonal primitives, is updated. To update the representation an operation called expansion is performed. Note that there is no way to shrink a primitive, because the environment to be represented is considered static. If it is static that means that the geometry of the objects in the scene does not change, e.g., they can only enlarge due to information that appears in new data and that was not present in previous acquisitions. In a static environment this occurs because the vehicle is moving through the scenario, and extra portions of objects are seen because they are closer or stop being occluded by other objects. An illustrative example is when a vehicle is moving on a road and there is a long wall on the side of the road. At the beginning of the road, sensors see only a portion of the wall and the primitive assumes the respective shape. As the vehicle travels through the road, new parts of the wall are visible to the sensors. These additional range measurements of the wall should be used to update the already existing primitive that represented the wall, which means that the primitives shape should be update to encompass the new data.

The expansion operation implements this mechanism. Figure 9.6 shows a example: Let $\mathcal{P}$ represent an input point cloud that is received at a given time and that contains several points (triangles and diamonds in Fig. 9.6 (a)), and the scenario representation that was previously computed be composed of a single primitive $\mathcal{G}$ (black solid line polygon in Fig. 9.6 (a)). The primitive has a support plane, defined by the local coordinate system X and Y axes. The local coordinate system represented by red-green-blue lines, e.g., the XYZ axes. The first step is to compute a new point cloud $\mathcal{P}_{ort}$ that is given by the points of $\mathcal{P}$ whose distance to the plane is smaller than the perpendicular expansion threshold T_{ort}:

$$\mathcal{P}_{ort} = \{\mathrm{p}_j \in \mathcal{P} \mid d_j < T_{ort}\}, \tag{9.1}$$

where d_j is the distance of point p_j to the support plane of $\mathcal{G}$. Only points that lie close to the primitives support plane are stored in $\mathcal{P}_{ort}$ and used in the next steps. In Fig. 9.6 (a), some points are included in $\mathcal{P}_{ort}$ (triangles) and others discarded (diamonds). Then, the points in $\mathcal{P}_{ort}$ are projected to the primitives support plane and their coordinates transformed to the primitives local coordinate frame. In this local reference frame, the projected points always have z value equal to zero, which is why only the x and y coordinates are stored, i.e., points are defined in $\mathbb{R}^2$. Let $\mathcal{J}$ be the point cloud that contains the x and y coordinates of the projected points viewed from the primitive local coordinate system. Figure 9.6 (b) shows the projections of the triangles of Fig. 9.6 (a) to the support plane (circles). This process is called the orthogonal part of the expansion. From here onward, all computations are performed in $\mathbb{R}^2$, which significantly speeds up the computation.

The second part of the expansion is referred to as longitudinal expansion. Figure 9.7 shows an example point cloud $\mathcal{J}$ that contains several points. Let us consider that some of these points actually belong to the same object that the primitive represents (circles in Fig. 9.7), and others do not (squares in Fig. 9.7). Since these points are obtained from new data, not all of them are contained inside the bounding polygon of the corresponding primitives. Therefore, the primitive should expand to accommodate these new points. To do this, an iterative process is proposed. The first step is to offset the existing bounding polygon of the primitive. The algorithm that was used was introduced in [Aichholzer *et al.* 1995]. The implementation is from [Cacciola 2012]. The offsetting operation

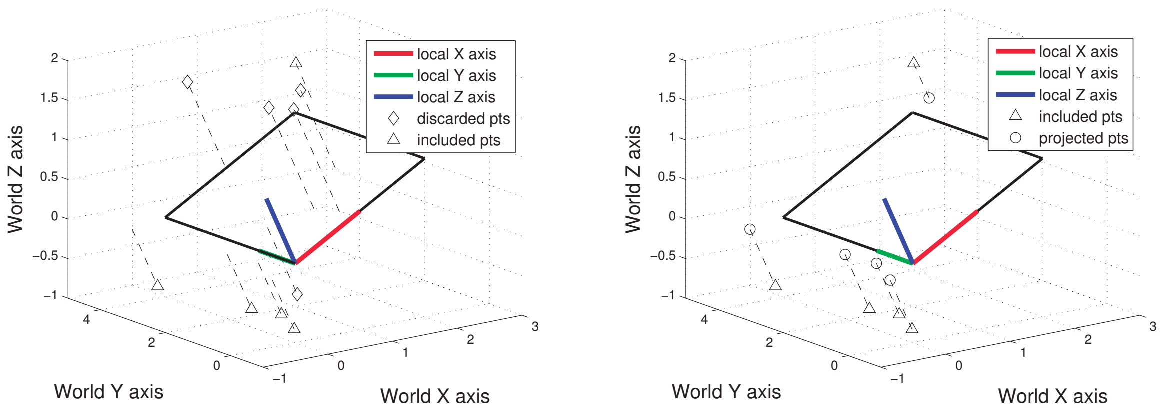


Figure 9.6: Orthogonal part of the geometric polygonal primitives expansion operation: (*a*) points are tested for their orthogonal distance to the support plane; (*b*) included points are projected to the support plane.

generates a new polygon from a given one. This new polygon is a grown or shrinked version of the input polygon, depending if the value of the offsetting is positive or negative, respectively. In this case only positive values of offsetting are used, i.e., the polygon can only grow. In every iteration, the polygon is grown as detailed next. The bounding polygon of the primitive is referred to as $\mathtt{P}$, and the grown or extended polygon is referred to $\hat{\mathtt{P}}$. Then, all points in $\mathcal{J}$ are tested to see if they are inside $\hat{\mathtt{P}}$ (implementation from [Giezeman & Wesselink 2012b]). The final stage is to compute a new convex hull. This new convex hull is computed from the point set that contains both the points of the previous convex hull and the points to which the polygon expanded to. The process repeats using the newly computed convex hull as starting hull. The iterative expansion stops when the extended polygon does not contain points inside it.

Figure 9.8 shows an example of an iterative longitudinal expansion. The initial situation for this expansion is depicted in Fig. 9.7. Remember that the circles represent points that are part of the object represented by the primitive, while squares are range measurements of other objects. Figure 9.8 (*a*) shows the start of the iterative process. The expanded polygon, (dashed line) is offset from the initial bounding polygon (solid blue line). Note that this offsetting has the same value under all dimensions, i.e., both vertical and horizontal. Since this is a search process and it is unknown where are the points that belong to the object, the best option is to expand likewise in all directions. All points belonging to $\mathcal{J}$ are tested to see whether they are inside the polygon. In the case of Fig. 9.8 (*a*), points that were inside are annotated with a blue cross. The second iteration is shown in Fig. 9.8 (*b*). The new convex hull is shown as a green solid line. It is computed as the convex hull of the previous iteration bounding polygon (solid blue line in Fig. 9.8 (*a*)) and the points found to be inside the previous iteration extended polygon (blue crosses in Fig. 9.8 (*a*)). The extended polygon is

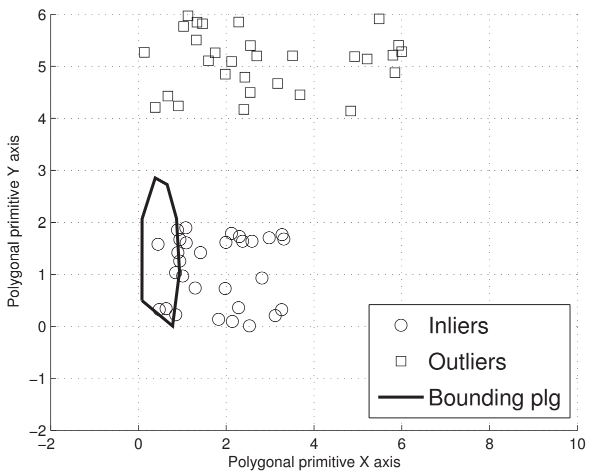


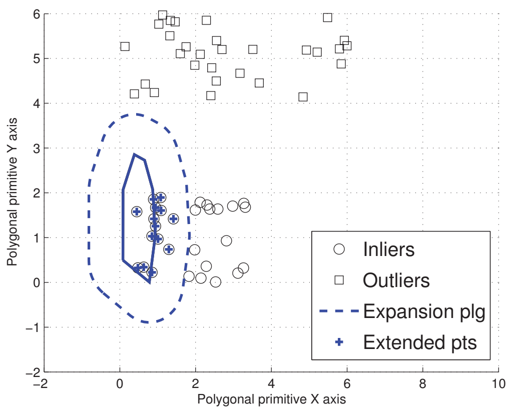
Figure 9.7: Expansion of geometric polygonal primitives. The polygonal primitive (polygon in black) that is to be expanded. In the next received range measurements, a point cloud containing candidates for expansion is defined. These have inliers (points that belong to the same object of the polygon, circles in Figure) and outliers (points that do not belong to the same object of the polygon, squares in Figure).

shown as the green dashed line. In this case, four additional points are found to be inside the polygon. The process repeats through the next two iterations, e.g., Figs. 9.8 (*c*) and 9.8 (*d*), until, in the fifth iteration (Fig. 9.8 (*e*)), no new points are found inside the extended polygon. This causes the iterative search to finish. The expansion operation changes the polygon from its initial state (blue solid line in 9.8 (*a*)) to a new shape, shown in the solid magenta line in Fig. 9.8 (*e*).

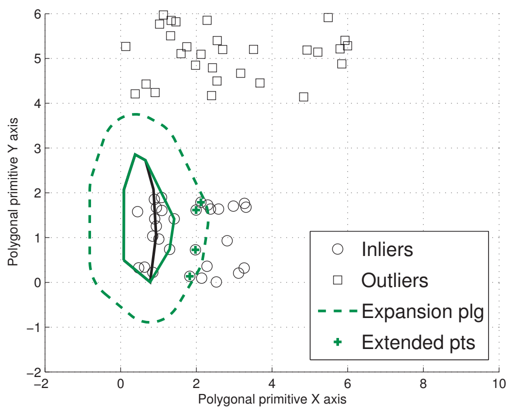
One of the advantages is that the convex hull grows only in the directions where new points are found inside the extended polygons, and remains unaltered in the directions where no new data is found. This is why, in the example shown in Fig. 9.8, the bounding polygon grew only in the horizontal (from left to right) direction. If the polygon grew in the vertical direction, from bottom to top, the outliers represented by the squares would eventually turn out to be inside an extended polygon and be included as belonging to the primitive.

Figure 9.9 (*a*) shows a diagram of the entire iterative expansion sequence, where it is easy to observe the horizontal expansion of the primitives bounding polygon.

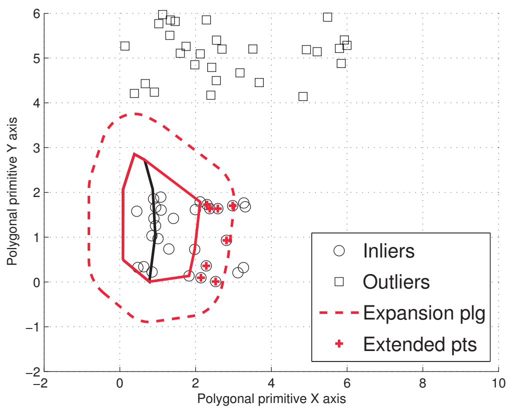
This property displayed by the algorithm of being able to selectively expand the bounding polygon in the right directions is very important because it allows the setting of small offset values. If these would be large, the consequence would be that outliers could also be included. Figure 9.9 (*b*) shows an iterative process where the offset value is set too high and the polygon expands to the outliers. Note also that if there was no iterative search, a value that would be sure not to find any outliers (squares) would also miss some inliers (circles). This is shown if Fig. 9.9 (*c*). If, on the other hand, the offset



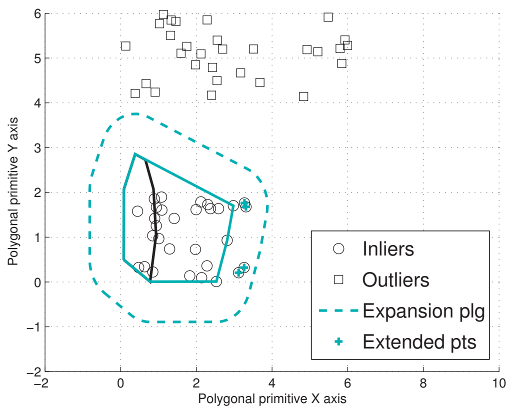
(*a*)



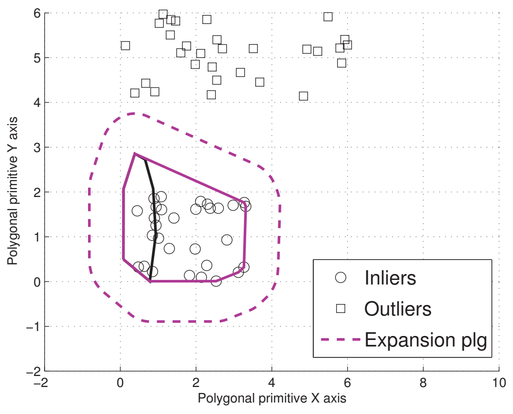
(*b*)



(*c*)



(*d*)

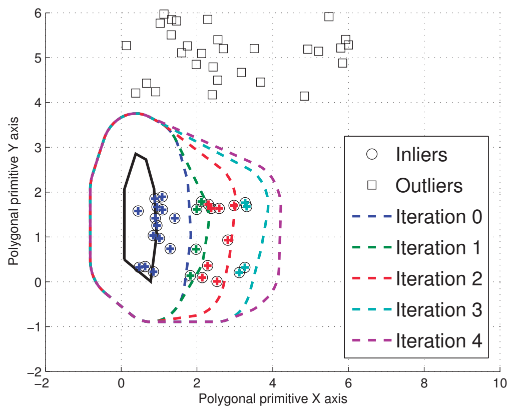


(*e*)

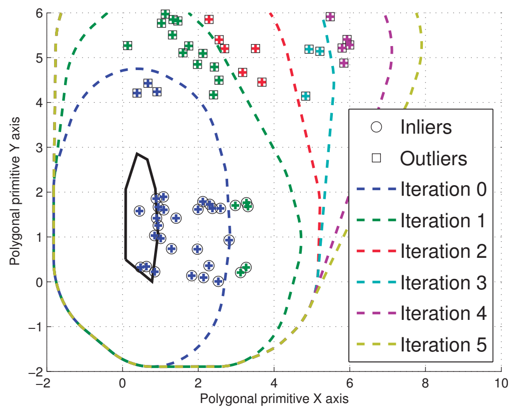
Figure 9.8: Expansion of geometric polygonal primitives. Solid lines in color represent the convex hull at the start of a given iteration, dashed lines the expanded polygon. Crosses over points mean they where added to the polygon in a given expansion: (*a*) iteration 0 (blue); (*b*) iteration 1 (green); (*c*) iteration 2 (red); (*d*) iteration 3 (cyan); (*e*) iteration 4 (magenta);

value is large enough to accommodate all inliers, it also includes some outliers, as shown in Fig. 9.9 (*d*). In sum, there is no value of offset that can grow the polygon enough to encompass all the circles and, at the same time, not encompass any of the squares. This is the reason why the iterative search is so important: it enables setting a very small offset value, to be sure that outliers are not found. The iterative propagation ensures that all points belonging to the same surface are found, even if the offset value is small.

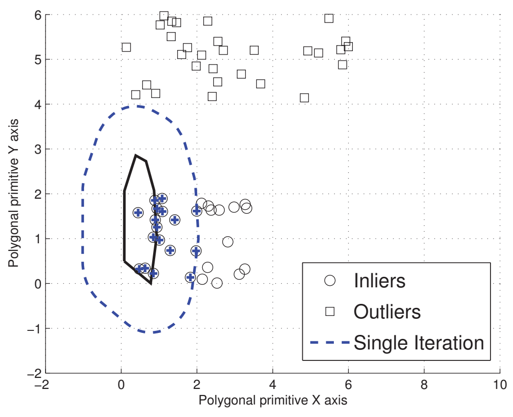
Since that the detection of polygons is quite slow, the expansion operation should be fast in order to compensate for it. One of the advantages of the proposed expansion mechanism is that most of the operations are executed in 2D space. Also, no nearest neighbor searches are executed



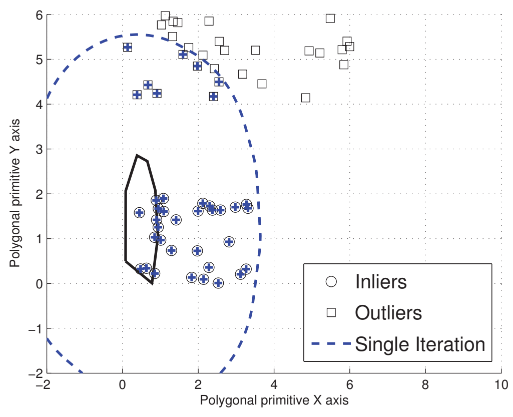
(*a*)



(*b*)



(*c*)



(*d*)

Figure 9.9: The effect of the longitudinal offset value, four examples: (*a*) the case shown in Fig. 9.8; (*b*) the offset value is too large and the expansion propagates to the outliers; (*c*) a non iterative example where the offset value is tunned not to include any outliers; (*d*) a non iterative example where the offset value is tunned to include all inliers.

when performing the expansion. As discussed in chapter 8, nearest neighbor searches are slow to process and tend to increase exponentially with the increase in the number of points. To avoid such searches is sure to speed up the algorithm. Obviously, the iterative process is slower than a single step process. However, in whole, the expansion algorithm is quite fast, considerably faster than the detection operations, which was one of the main objectives of the expansion operation. Section 9.4 will present results with the performance of the expansion algorithm.

9.3.1 Incremental Bounding Polygon Computation

Section 9.3 described the entire algorithm for the expansion of the geometric polygonal primitives. One of the key components of the algorithm is the iterative processing, and one of the steps that are performed in every iteration is the computation of a new convex hull. As discussed, at the end of every iteration, the objective is to compute a new convex hull that envolves both the convex hull of the previous iteration and the newly added points that were found to be inside the extended polygon. This section describes in detail how this process is performed. This section will only address the incremental computation of the bounding polygon when a convex hull is used. If a concave hull is employed the following deduction is not valid and a different solution must be found.

We will use the following notation for points or lists of points: when we refer to the 3D point, a handwritten character is used ,when we refer to the projection of those points to the primitives support plane, that is, to the 2D points defined on the primitives local coordinate system, then a typewritten character is used. For example, $\mathcal{S}$ refers to the 3D coordinates (defined in the global reference system) of the primitives support points, while $\mathtt{S}$ is the projection of those support points to the support plane (defined in the local coordinate system).

At expansion iteration j, let a given primitive have an initial bounding polygon $\mathtt{P}_{j-1}$ and support points $\mathcal{S}_{j-1}$. , which, if projected to the support plane become $\mathtt{S}_{j-1}$. Let the expansion process find the list of points $\mathcal{E}_j$, as points inside the extended polygon ($\mathtt{E}_j$ in the case of their projection). The objective is to update the bounding polygon at the end of iteration j, e.g., to obtain $\mathtt{P}_j$. At first sight, the solution is obvious: to compute the convex hull of a point cloud that contains all the support points of a primitive, plus the expanded points. Since the convex hull operation is performed on the primitives local coordinate system (this is done inside the longitudinal part of the expansion), only the 2D representations are used:

$$\mathtt{P}_j = \mathbf{conv}(\{\mathtt{S}_{j-1}, \mathtt{E}_j\}), \tag{9.2}$$

where **conv** is the function that returns the convex hull for a set of points and $\{A, B\}$ represents the union of groups A and B. The problem with the solution of eq. (9.2) is that, at iteration j, support points $\mathtt{S}_{j-1}$ should not be stored in memory. In section 8.4, results are shown that prove that polygonal primitives are a very memory effective representation. The representation is capable of compressing the raw data to 1% of its original size. In fact as shown in eqs. (8.12) and (8.13) and in Fig. 8.11, if all the data is stored, a few seconds are enough to have several gigabytes of data, which is not a feasible solution. In other words, before new data arrives, that is, before primitives are expanded, the memory containing the primitives support points must be freed. Hence eq. 9.2 cannot be used. However, convex hulls have some interesting properties which can be used to devise a solution. It is shown in [Krein & Smulian 1940] and [Schneider & Rolf 1993] that, for any given set of points $S1$ and $S2$, the convex hull of their Minkowski sum is the Minkowski sum of their convex hulls, that is:

$$\mathbf{conv}(\{S1, S2\}) = \mathbf{conv}(S1 + S2) = \mathbf{conv}(S1) + \mathbf{conv}(S2). \tag{9.3}$$

Then, eq. 9.2 can be changed to:

$$\mathtt{P}_j = \mathbf{conv}(\mathtt{S}_{j-1}) + \mathbf{conv}(\mathtt{E}_j). \tag{9.4}$$

$\mathtt{P}_{j-1}$ was obtained as the convex hull of the primitives support points at iteration $j-1$, which results in:

$$\mathtt{P}_{j-1} = \mathbf{conv}(\mathtt{S}_{j-1}), \tag{9.5}$$

and eq. 9.4 becomes:

$$\mathtt{P}_j = \mathtt{P}_{j-1} + \mathbf{conv}(\mathtt{E}_j). \tag{9.6}$$

Another important property of the convex hulls is the idempotence. Idempotence is the property of certain operations in mathematics and computer science, that they can be applied multiple times without changing the result beyond the initial application. That is, for a given set S1:

$$\mathbf{conv}(\mathbf{conv}(S1)) = \mathbf{conv}(S1), \tag{9.7}$$

which transforms eq. 9.6 to:

$$\mathtt{P}_j = \mathbf{conv}(\mathtt{P}_{j-1}) + \mathbf{conv}(\mathtt{E}_j), \tag{9.8}$$

and finally, using eq. (9.3), results in:

$$\mathtt{P}_j = \mathbf{conv}(\{\mathtt{P}_{j-1}, \mathtt{E}_j\}). \tag{9.9}$$

This deduction proves that it is possible to use just the bounding polygon of the previous iteration and the extended points in order to compute the new convex hull. It is very important since it makes possible to update the convex hull in an expansion operation without need to store all of the primitives support points, which would not be possible due to the very large memory size required.

If the concave hull is used instead of the convex hull this deduction does not hold. Some alternative method using for example point sampling over the previous iteration polygon would have to be implemented. However, such work was not endeavored, which is why the following sections only use convex hulls as bounding polygons.

9.3.2 Incremental Support Plane Computation

In order to properly update a primitive, not only its bounding polygon but also its support plane must be updated. The same problem is posed when, after a primitive expands to a set of points $\mathtt{E}_j$, a new support plane must be computed. In theory, to compute the new support plane that satisfies also the expanded points would be just a matter of running a Principal Component Analysis (PCA) operation over the set of support points plus the extended points. At expansion iteration j, the objective is to find the primitives new plane coefficients $\mathcal{G}_j^p$:

$$\mathcal{G}_j^p = \mathbf{pca}(\{\mathcal{S}_{j-1}, \mathcal{E}_j\}), \tag{9.10}$$

where **pca** is a function that retrieves the plane coefficients using PCA, $\mathcal{S}_{j-1}$ are the primitives support points at the previous iteration and $\mathcal{E}_j$ are the expanded points. Both are defined in $\mathbb{R}^3$, e.g., they are not the projections of the 3D points to the plane but the 3D points themselves. Like in section 9.3.3, the same problem is posed. For data compression purposes, the support points $\mathcal{S}_{j-1}$ should not be stored in memory. The idea is to replace these points by the representations of polygonal primitives. Even if it was feasible to store all support points in memory, the amount of support points (remind that a polygon can have hundreds of thousands of support points in just one iteration) after some iterations would make it impossible to execute a PCA in real time. To solve this issue, we propose to perform a PCA over a sampled set of points. The idea is to find a smaller set of points that are representative of the set $\{\mathcal{S}_{j-1}, \mathcal{E}_j\}$. The ratio between these should be maintained. Let $\hat{\mathcal{S}}_{j-1}$ and $\hat{\mathcal{E}}_j$ be the sampled sub sets of $\mathcal{S}_{j-1}$ and $\mathcal{E}_j$, respectively. Let $N_{\mathcal{S}_{j-1}}$ be the number of support points at iteration $j-1$, and $N_{\mathcal{E}_j}$ the number of points in the extended set $\mathcal{E}_j$. Similarly, $N_{\hat{\mathcal{S}}_{j-1}}$ and $N_{\hat{\mathcal{E}}_j}$ refer to the number of points in the sampled subsets. If the sampled sets are representative and the ratio between the support points and extended points is maintained over the real set of points and the sub sampled set of points, that is:

$$\frac{N_{\mathcal{S}_{j-1}}}{N_{\mathcal{E}_j}} \simeq \frac{N_{\hat{\mathcal{S}}_{j-1}}}{N_{\hat{\mathcal{E}}_j}}, \tag{9.11}$$

then, the result of PCA is approximately the same if we use either the complete set of points $\{\mathcal{S}_{j-1}, \mathcal{E}_j\}$ or the sampled set of points $\{\hat{\mathcal{S}}_{j-1}, \hat{\mathcal{E}}_j\}$:

$$\mathcal{G}_j^p = \mathbf{pca}(\{\mathcal{S}_{j-1}, \mathcal{E}_j\}) \simeq \mathbf{pca}(\{\hat{\mathcal{S}}_{j-1}, \hat{\mathcal{E}}_j\}). \tag{9.12}$$

There are many ways to sample representative subsets from real sets. In our case, we found that very simple sampling mechanisms suffice to provide accurate estimations of the new support planes coefficients. In practice the procedure is the following: a number $N_{\hat{\mathcal{E}}_j}$ of sampled extended points $\hat{\mathcal{E}}_j$ are randomly extracted from the total extended points list $\mathcal{E}_j$. Then, using eq. (9.11), the number of points $N_{\hat{\mathcal{S}}_{j-1}}$ that should be sampled from the primitive is computed. The sampled points $\hat{\mathcal{S}}_{j-1}$ are computed by randomly selecting points along the edges or inside the bounding polygon of the primitive. Using these two sampled sets, it is possible to compute the new plane coefficients. From test and trial it was found that if a total number of sampled points is around 1000 points, e.g., $N_{\hat{\mathcal{E}}_j} + N_{\hat{\mathcal{S}}_{j-1}} = 1000$, the sampled sets are representative enough to get accurate values of plane coefficients.

9.3.3 A Cascade Processing Configuration for the Expansion

The previous sections have described what is the proposed algorithm to expand geometric polygonal primitives. This section describes how the expansion mechanism is integrated into the entire scenario representation methodology. As in the case of the detection of geometric polygonal primitives (see section 8.3.7), a cascade processing configuration is used. The cascade configuration fastens the processing considerably. The underlying core assumption is that each range measurement can only be contained by a single primitive. In other words, the representation assumes that each range measurement is a measurement of a single object on the scene and therefore can be explained by a single primitive in the representation. Under this assumption, the points that have been attached to a given primitive by an expansion operation, can only belong to that primitive and not any other. Because of this, expanded points are removed from the input point cloud and are not a part of subsequent expansions (of other primitives) nor part of detections of new primitives. Since all the points that are taken by the expansion of a primitive are removed from the input point cloud, the subsequent expansions or searches are fastened since that less points will have to be analysed. In terms of configuration, a cascade processing recommends that the faster stages are computed first. The expansion of primitives is faster than the detection. Because of this, when a new input point cloud is received, all existing primitives are first expanded and only then the remainder non expanded points are used for searching new primitives. Algorithm 9.1 describes the architecture of the complete algorithm for the geometric polygonal primitives representation computation, including both the detection and expansion operations.

Algorithm 9.1 Cascade configuration for the expansion of existing geometric polygonal primitives and detection of new ones.

```
Input: The current list of geometric polygonal primitives $\mathbf{G} = \{\mathcal{G}^0, \mathcal{G}^1, ..., \mathcal{G}^n\}$
Input: $\mathcal{P}$, the input point cloud, containing new range measurements
Output: The updated list of geometric polygonal primitives $\mathbf{G} = \{\mathcal{G}^0, \mathcal{G}^1, ..., \mathcal{G}^n, \mathcal{G}^{n+1} ..., \mathcal{G}^{n+m}\}$,
    where $n$ is the number of primitives contained in the current representation and $m$ is the number of
    additional primitives that are found in this iteration
  for $i = 0 \rightarrow n$ do                                        ▷ Expansion of primitive $\mathcal{G}^i$
      Find the orthogonal distances $d$ from all points in $\mathcal{P}$ to the support plane of primitive $\mathcal{G}^i$
      Compute a new point cloud $\mathcal{P}_{ort}$ containing all points of $\mathcal{P}$ whose distance is smaller than the
        perpendicular expansion threshold ($T_{ort}$),    $\mathcal{P}_{ort} = \{\mathtt{p}_j \in \mathcal{P} \mid d_j < T_{ort}\}$
      Project all points in $\mathcal{P}_{ort}$ to support plane of primitive $\mathcal{G}^i$, obtain projected point cloud $\mathcal{J}$
        defined in $\mathbb{R}^2$
      Initialize cycle break flag, $cycle\_break \leftarrow$ false
      Initialize number of expansion iterations, $it \leftarrow 0$
      while $cycle\_break =$ false do                                  ▷ Iterative longitudinal expansion
          Expand bounding polygon, $\mathtt{P}^i_{it}$, and obtain expanded polygon $\hat{\mathtt{P}}^i_{it}$
          Compute $\mathcal{J}^{in}_{it}$, the points from $\mathcal{J}$ that are inside $\hat{\mathtt{P}}^i_{it}$
          if $\mathcal{J}^{in}_{it} = \{\}$ then                          ▷ no expansion occurred, break cycle
              $break\_cycle \leftarrow$ true
              Update primitive $\mathcal{G}^i$, by defining the updated bounding polygon given by $\mathtt{P}^i_{it}$, and
                recomputing the support plane using PCA over the old and the new inliers, e.g, the
                points whose projections are points contained in the list $\{\mathcal{J}^{in}_0, ..., \mathcal{J}^{in}_{it}, \}$
          else
              Remove all points in $\mathcal{J}^{in}_{it}$ from $\mathcal{J}$
              Remove all points from $\mathcal{P}$ whose projections are points in $\mathcal{J}_{in}$
              Compute the bounding polygon of the next iteration, $\mathtt{P}^i_{it+1}$, from the convex hull
                of $\{\mathtt{P}^i_{it}, \mathcal{J}_{in}\}$
          end if
          increment number of iterations, $it \leftarrow it + 1$
      end while
  end for
  From the input point cloud $\mathcal{P}$ from which expanded points were removed, continue to perform
    detection of new polygonal primitives $\mathcal{G}^{n+1} ..., \mathcal{G}^{n+m}$, and add them to the list of primitives $\mathbf{G}$,
    Algorithm 8.3
```

9.4 Results

This section presents several results and analysis of the expansion mechanism of the geometric polygonal primitives. The polygonal primitives algorithm without the expansion mechanism is compared against the same algorithm using the expansion mechanism. To guarantee a fair comparison, in both approaches, all the detection parameters are the same. These two algorithms will be refered to as with expansion and without expansion. Using this evaluation it is possible to assess what are the benefits or disadvantages of the expansion mechanism. All five locations of sequence 1 are used to obtain results. Parameters used in the detection are similar to the GPP 1 set.

9.4.1 Qualitative Analysis

Figure 9.10 shows a reconstruction of the scene using the expansion mechanism. The state of the reconstructed scenario at each location is shown. One clear advantage of this representation is that there are no overlapping primitives. This was discussed in section 9.2 and was expected to be one of the benefits of using a mechanism that was capable of updating the scenario reconstruction. A qualitative analysis of the results present in Fig. 9.10 also shows that the most important features of the scenario are contained in the representation, especially if the task in mind is navigation.

9.4.2 Computation Time

Of the important aspects, one that should be analysed is the effect of the expansion mechanism to the overall processing time of the scenario representation computation. As seen with the GPP 1 and GPP 2 parameters set, the values of the parameters drastically influence the computation time. This is because one can obtain a simplified representation in a small amount of time or, instead, a more complete representation taking longer to process. Hence, the time taken to compute a given representation is not, in itself, a measure of the computational requirements or of the quality of the algorithm. It depends on the completeness and / or quality of the produced representation. Given this, it is not adequate to compare directly the time taken to produce a representation with and without the expansion mechanism, if the quality of the representations produced by both does not play a role in the evaluation. Time to compute and quality of the representation, both of these variables should influence the evaluation and are dependent. To solve this issue, an alternative comparison is performed. The computation time is fixed to a given value for both algorithms, and the quality of the produced representation in that limited amount of time is measured. This method is in fact very similar to the one used for real time reconstruction. Let us consider sequence 1. For each of the five locations, there are 400 thousand input points. The algorithm can spend a great deal of time reconstructing a scene viewed at a particular location, but after some time new data from the next location is made available. In these circumstances, it does not make sense that the algorithm is still processing data from the first location. Instead it should drop the older data and start searching for primitives in the new data. In the results that follow, only the input points at each of the locations are used. Each location is separated by ten seconds.

Although the polygonal primitives method was proved to be the faster amongst all evaluated, it is still not fast enough for real time. Hence, we consider that the algorithm should spend twenty seconds processing time at each location, which is twice the amount of time available. A real time application would require faster computers and or that the vehicle would travel at half the speed. Although the geometric polygonal primitives are not able, in their current state, to run in real time, it should be

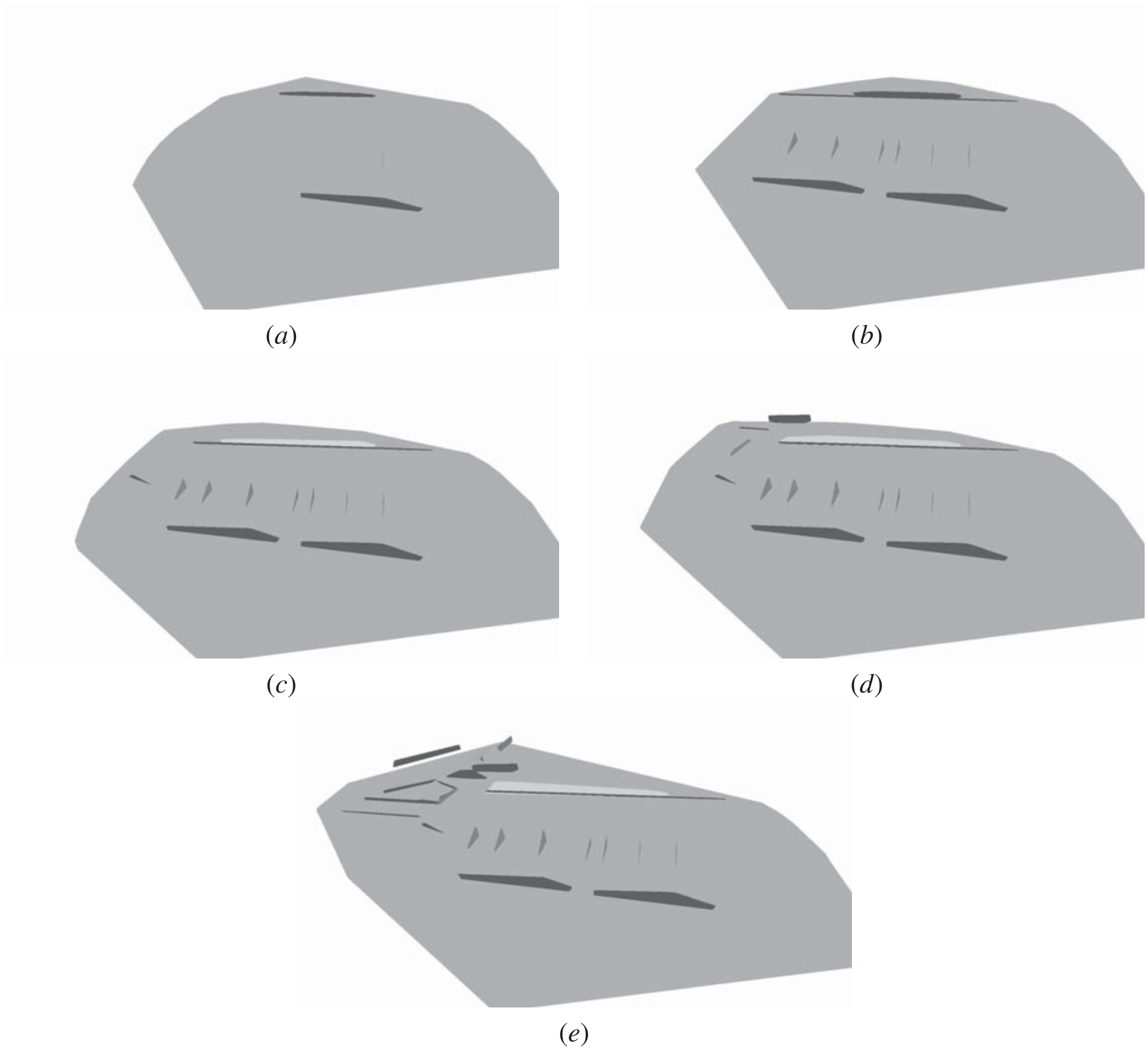


Figure 9.10: GPP 1 reconstruction for sequence 1: (*a*) The reconstructed scenario after the input point cloud of location *A* is received; (*b*) after location *B*; (*c*) after location *C*; (*d*) after location *D*; (*e*) after location *E*.

noted that from the analysis conducted in section 8.4 this methodology is much faster than all other evaluated alternatives and so, it should be signaled as the best candidate for real time applications running in better hardware. In sum, the algorithm is set to stop processing an input point cloud when twenty seconds pass from its arrival. Then it waits for the next input point cloud or starts processing it if it is already present in the message buffer.

Figure 9.11 shows the number of polygons created in both algorithms. Although at the beginning of the sequence both algorithms generate a similar number of primitives, after some locations

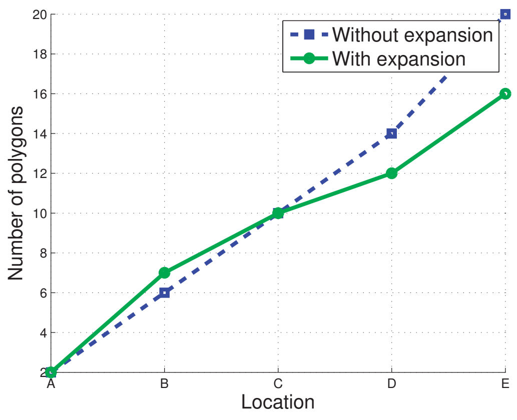


Figure 9.11: Comparison between the number of polygons generated by the algorithm using expansion and not using expansion through sequence 1.

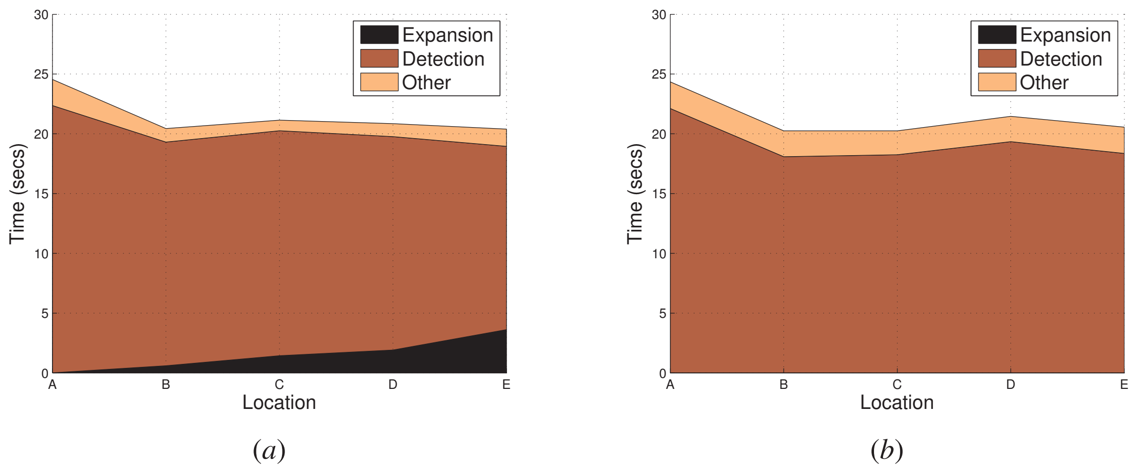


Figure 9.12: Comparison of the processing time used in the expansion, detection and other parts of the algorithm during a pass through sequence 1: (*a*) algorithm with the expansion mechanism; (*b*) algorithm without the expansion.

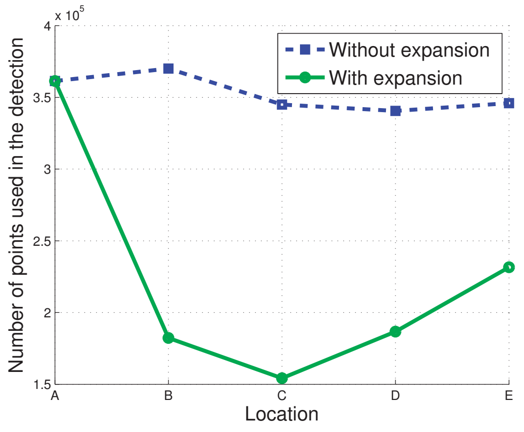
the algorithm without expansion shows a greater number of primitives. The explanation is that since the algorithm without expansion does not compare the stored primitives with the new data, it ends up duplicating primitives. On the contrary, when using the expansion mechanism, the duplication of primitives is avoided which leads to a smaller number of primitives. Note that just because a representation has more primitives it is not necessarily better. In fact, if there are duplicated primitives, the representation may be worse than another with a smaller number of non duplicated primitives.

Figure 9.12 shows graphs that display the usage of time by each of the mechanisms: detection, expansion, and other processing. Both with and without expansion algorithms are shown. Figure 9.12 (*a*) shows that, in the case of the algorithm with expansion, the largest portion of the time is spent detecting new primitives. In location *A*, the start of the processing, there are no primitives stored. Because of this no time is spent performing the expansion. As the vehicle moves and more primitives are stored, a higher number of primitives must be expanded (or tested for expansion). This leads to an increase in the time spent in the expansion part. Even though, the time spent processing the expansion mechanism is very small when compared to the detection time. This means that the algorithm with expansion does not spend a lot of time executing the expansion, having almost as much time available as the algorithm without expansion to search for and detect new primitives. Figure 9.12 (*b*) shows the same analysis for the case of the algorithm without expansion. Obviously, in this case, no time is spent in the expansion mechanism since there is none. An interesting observation is related to the other processes. Other processes consist in the pre-processing of the input point clouds, namely the estimation of normals (see section 7.4.2). It is observable that when the expansion mechanism is used the time consumed by other processes is reduced to about half. The explanation for this resides in the cascade processing described in section 9.3.3. When an input point cloud is received, the first mechanism that is called to process the point cloud is the expansion mechanism. Then, only the points that are not explained by the expansion are used in the following processes: preprocessing, computation of normals and finally detection. That is to say that it is left for the expansion mechanism the burden of processing the complete input point cloud (containing around 400K points). Pre-processing consisting of normal estimation is executed only in the remainding fewer points. This is why, in Fig. 9.12, the *other* part of the processing takes considerably less time when the expansion is used: because there are less points to process. The same phenomena should be observable with the detection mechanism, since it is also posterior to the expansion. However, since that a fixed amount of time is set for executing the whole reconstruction process, what happens is that the polygons are searched for the time that remains, until the twenty second limit is reached.

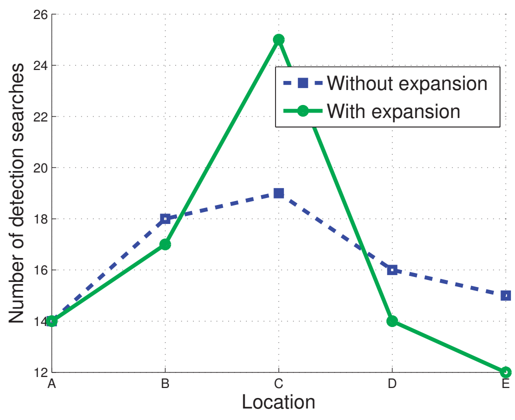
9.4.3 Comparison of Approaches With and Without Expansion

Since it was shown that the expansion mechanism does take some time, a question could be posed whereas it would not be better to use that time searching for polygons, i.e., performing detection. In fact, the objective is to use as much time as possible for the detection of new primitives. However, expansion is executed prior to detection and removes many points from the point cloud by assigning them to already existing primitives. As a result, the smaller amount of time available to detect polygons in the algorithm with expansion is compensated by the fact that a smaller point cloud is used for searching, which makes the process more effective.

Figure 9.13 (*a*) shows the number of points that are given as input for the detection mechanism. In the case of the algorithm without expansion, all of the input points are passed on to the detection, i.e., approximately 400K points. However, when the expansion is active, a large number of points are



(*a*)



(*b*)

Figure 9.13: Analysis of the expansion mechanism: (*a*) number of points to use as input to the detection; (*b*) number of detection searches.

explained by the expansion and are not feed into the detection. Figure 9.13 (*a*) shows that the number of points is reduced to approximately half the input points. In conclusion, the expansion mechanism takes a small amount of time when compared to the detection and other processes, and is able to quickly explain a large portion of the input points. This fact counterweights the fact that less time is made available to the detection, since that although detection searches must run in less time, they also run in point clouds with half the size. Note that expansion takes in the worst case scenario 25% of the time (under five seconds out of the twenty second window, see Fig. 9.12 (*a*)), and in return it explains 50% of the input points (200K points out of 400K points, see Fig. 9.13 (*a*)).

Figure 9.13 (*b*) shows the number of detection searches, e.g., the number of times the Random Sample Consensus (RANSAC) algorithm was executed. Since RANSAC is based on random sampling, a better scene representation should occur when there are more searches. Although the expansion mechanism leaves less time for detection, it reduces the size of the point cloud where primitives are searched for. Fig. 9.13 (*b*) shows that roughly the same number of searches are made. Hence, as a conclusion, it is possible to state that the expansion mechanism as a beneficial effect on the overall scene representation algorithm.

Previous results have shown that the expansion mechanism does not reduce the number of primitive searches. But a more detailed description of the scenario representations produces by both algorithms is in order. An analysis of the scenario representation produced with and without the expansion mechanism active. For this purpose, there a distinction between ground and vertical polygons that should be made. Ground primitives are primitives that are horizontal. They are typically collinear with the ground plane and are obtained using a freely oriented plane RANSAC search, as described in section 8.3.2. On the other hand, vertical entities are obtained from oriented plane RANSAC searches.

They describe vertical objects in the scenario such walls, trees, etc. Similarly, points and bounding polygons may also be designated as of ground or vertical nature. The reason for this distinction has already been advanced in section 8.3.2: typically, the number ground points exceeds by far those of vertical entities. Because of this, results from the ground primitive could mask all other vertical primitives. Results are presented separately for the ground and vertical entities.

Figure 9.14 shows, for sequence 1, the accumulated number of points assigned to ground primitives, vertical primitives, and not explained. Many points are left unexplained because these results refer to that twenty second time window for processing. Also, the area and solidity thresholds have discarded some polygons and, furthermore, there are always some points that cannot be fitted to planes in every scene. Naturally, at the beginning of the sequence, both the algorithm with expansion (Figure 9.14 (*a*)) and the algorithm without expansion (Figure 9.14 (*b*)) are almost identical. The reason is that at the beginning of the sequence there are no primitives stored in memory to expand, and the expansion has no effect. As the sequence moves forward, the number of unexplained points is approximately the same. It is the same to say that both algorithms explain the same amount of points in the environment. When we analyse the number of ground points, the algorithm with expansion obtains less points, since some portions of the road are seen behind walls. Because of this they are not connected to the existing ground polygon and, therefore, it does not expand towards them. It should be said that these portions of the road are somewhat unimportant for navigation purposes, and that the expansion methodology captures a great deal of the road's surface, as is shown in Fig. 9.10. In terms of vertical points, we can see that the algorithm with expansion is able to obtain more of these points. Since it picked up the ground primitive (as well as other primitives) at location *A*, in the following locations the twenty seconds where used to detect other polygons, which explains why it shows a greater number of points assigned to the vertical primitives. On the contrary, the algorithm without expansion searches (and detects) the ground polygon at every location. It does it marginally better, which is why there are more assigned ground points, but then lacks the time to detect many vertical polygons.

An analysis of the number of points assigned to ground primitives is shown in Fig. 9.15 (*a*). As discussed, the detection of the ground plane is more complete in the algorithm without expansion, which is why there are more points assigned to these primitives. In Fig. 9.15 (*b*), the total accumulated area of the ground nature primitives is depicted. Here, the difference between both algorithms is much larger than in Fig. 9.15 (*a*). This is because there is a duplication of area in the case of the polygon without expansion. This issue was addressed in section 9.2. The algorithm without expansion is in fact the second approach referred in that section, where scenario representation is performed by reconstructing each scene individually and then merging all scene representations. In this case the merging algorithm simply joins all detected polygons. Hence, when the same surface is detected in two separate locations, it is attributed to two different primitives. When the area of both is summed up there is an area duplication effect. In fact, in locations where the vehicle travels slowly, there will

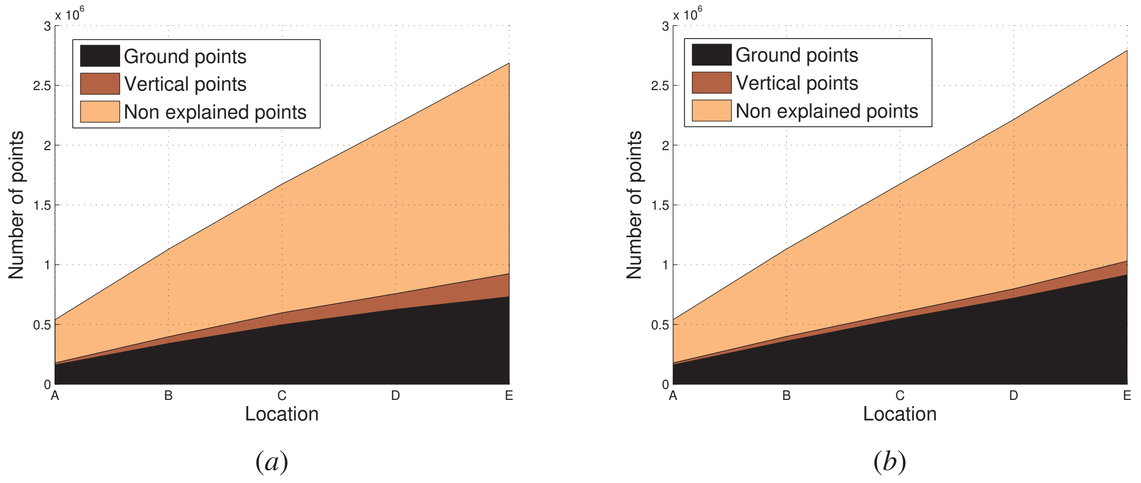


Figure 9.14: Number of accumulated points during sequence 1 according to their ground, vertical or non explained attributes: (*a*) algorithm with the expansion mechanism; (*b*) algorithm without the expansion.

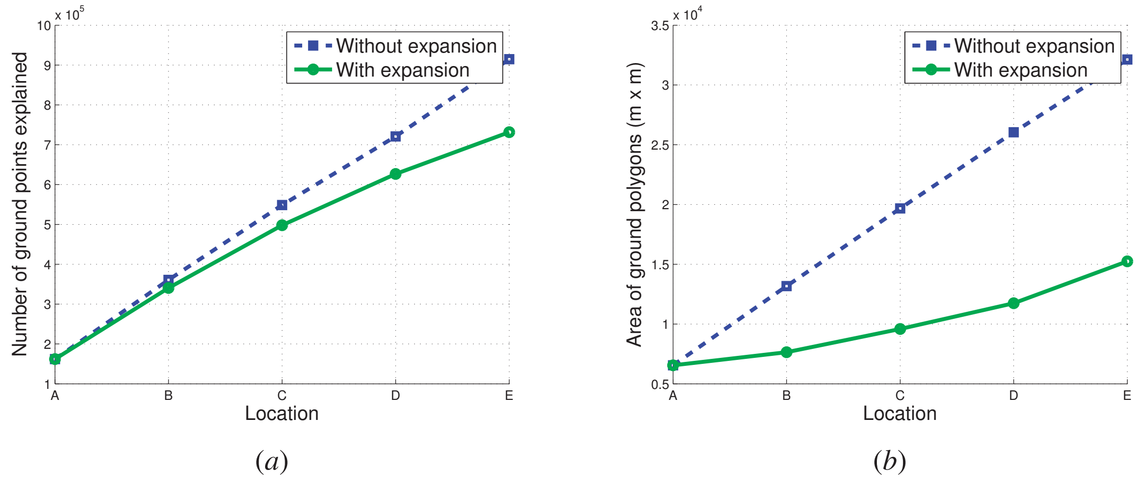


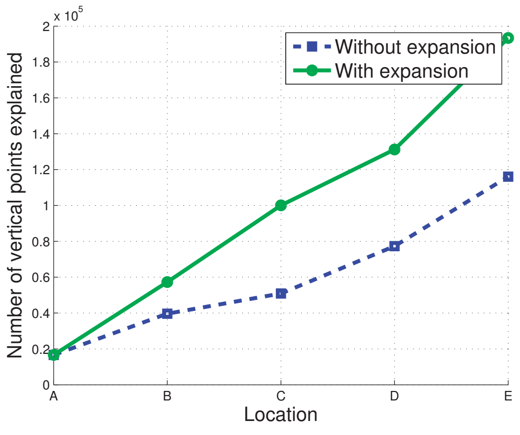
Figure 9.15: Analysis of the ground polygons data. Comparison between the algorithm using expansion and not using expansion through sequence 1. (*a*) number of ground points explained; (*b*) area of the ground polygons.

be a great overlap between the input point clouds of the locations and, to some extent, it can occur that the areas are even triplicated. This is what happens in Fig. 9.15 (*b*).

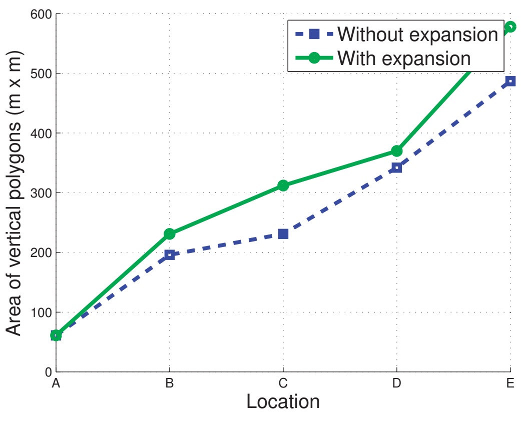
Having said this, it seems that the area is not a good metric to perform the comparison between with and without expansion algorithms. The answer is shown in Fig. 9.16. Figure 9.16 (*a*) shows that the algorithm with expansion obtains more points assigned as vertical. This had already been concluded from the analysis of Fig. 9.13. The most interesting conclusion comes from Fig. 9.16 (*b*). Although the algorithm without expansion has the advantage of duplicating the areas as mentioned above, the fact is that it that the algorithm equipped with the expansion mechanism is able to compensate for this advantage and achieve a larger total area contained in vertical polygons. As a conclusion, the algorithm with expansion is capable of detecting a larger amount of vertical primitives. Furthermore, in the algorithm without expansion, the same surfaces are detected multiple times, that is, computational resources are spent re detecting the same obstacles. This phenomena does not occur when the expansion is active and is one of the greatest advantages of this mechanism.

9.4.4 Evolution of Primitives

Next we focus our attention on how the primitives evolve. Only the with expansion algorithm is portrayed since in the other algorithm, primitives are static. Figure 9.17 (*a*) shows the number of support points assigned to each primitive. Only pair index primitives are shown to simplify the graph. Data from each primitives may start at different locations and signals where the primitive was detected. From observing Fig. 9.17 (*a*) we can see that at location *A* only primitive 0 was created, in location *B* primitives 2,4 and 6 are generated, in location *C* only primitive 8 is created. This is to say that since RANSAC is a random process there is no guarantee on the number of detected primitives at each location. In fact, this number may also depend on the underlying geometry of the location. The best that can be done is to execute as much searches as possible, to increase the chances of detecting all relevant polygons. Primitives 12 a 14 are created at location *E*, which is the last location of the sequence, and therefore do not evolve. Primitive 0 is the one that contains the greatest amount of



(*a*)



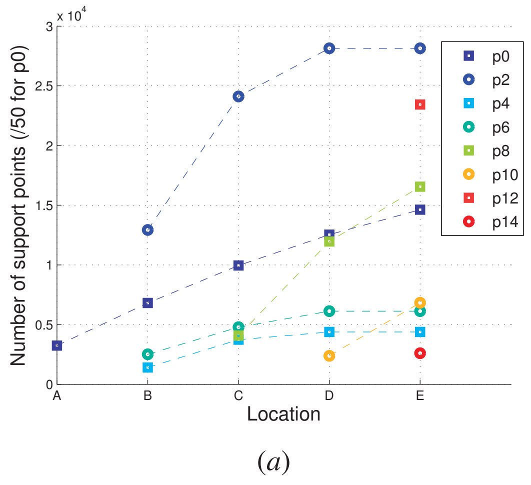
(*b*)

Figure 9.16: Analysis of the vertical polygons data. Comparison between the algorithm using expansion and not using expansion through sequence 1. (*a*) number of vertical points explained; (*b*) area of the vertical polygons;

support points (note that the number of support points is divided by 50 for this primitive, in Fig 9.17 (*a*)). It is of course the primitive that represents the ground plane. Another interesting observation is that most primitives significantly increase their number of support points throughout their evolution. Primitive 0 was detected at location *A* with $0.4 \times 50 \times 10^4 = 200 \times 10^3$ points, and at location *E* it already supported $1.4 \times 50 \times 10^4 = 700$K points. In other words, it increased the number of support points by 350%. Another example, primitive 10, detected at location *D* with 3K points, has at location *E* around 7K points. A 230% increase between consecutive locations.

Figure 9.17 (*b*) shows the area of the bounding polygon for the same primitives of Fig. 9.17 (*a*). Very significant increases in the area of the primitives bounding polygons is also present: primitive 0 (note that the area is divided by 200 for this primitive, in Fig 9.16 (*b*)), location *A*, $32 \times 200 = 6400m^2$, by the end of location *E* has $65 \times 200 = 13000m^2$, an area increase of 203%. Primitive 10, location *D*, area $11m^2$, location *E* area $29m^2$, a 260% increase in a single expansion. All these observations, both in number of support points as well as in terms of area, show that polygons grow considerably after being detected. If these primitives where not expanded, the additional support points and area would have to be handled by a detection mechanism. The expansion mechanism in one that can compute this growing of primitive in a simple and efficient manner, taking a lot of the burden of processing many points away from time consuming detection mechanisms.

It is also possible to see that some polygons stabilize their number of support points and area after some iterations. It is the case of polygons 2, 4, and 6. None of this had an increase in points or area, from locations *D* to *E*. It means either that the object surface represented by the primitive is completely contained inside the bounding polygon, or that the vehicle is far away from the polygon and does



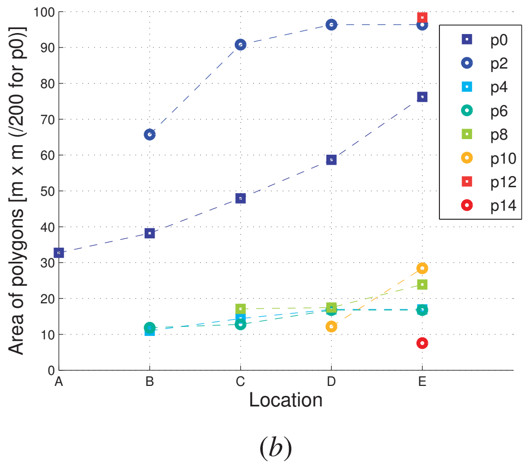


Figure 9.17: Analysis of the evolution of each of the polygons through sequence 1. Only pair index polygons are shown to simplify the graphs: (*a*) number of support points per polygon; (*b*) number of vertices in the convex hull; (*c*) area of the polygons.

not capture range measurements anywhere near that primitive. Whatever the reason, these values are good indicators to assess whether a polygon has become stable in the representation. A simple solution could be to label a polygon as stable after a few number of iterations where no evolution is noticeable. This could be important because stable primitives do not have to be tested for expansion. Going back to Fig. 9.12 (*a*), where the time spent performing expansion is shown, one can see that the expansion time is increasing as the sequence moves forward. If the tendency continues, it would come to a point where all of the twenty seconds are spent expanding already detected primitives. This occurs because at each new location there are more primitives listed to test for expansion. The time spent is increasing because the number of primitives is also increasing. If primitives can be signaled as stable, only the unstable primitives would be tested for expansion, and since the number of unstable primitives does not increase continuously, the time spent with expansion would just remain at a low value.

Finally, note that both the support points and the area never decrease. It is so because no collapsing or shrinking mechanism is used, the only possibility for a primitive is to enlarge. No such mechanisms where considered given the problem setup and the philosophy behind the proposed solution. Real time processing of such large point clouds is a demanding task. To create and evolve a representation is already computationally very expensive. The solution here proposed tries to make use of simple yet efficient mechanisms in order to create a representation. Given the amount of input data bandwidth, the option is to try to spend the time trying to explain new data other than revising previously explained data. Polygon shrinking would enter the category of representation revising, fine tunning, or post processing, which is why a solution was not developed.

Figure 9.18 shows the number of vertices of the bounding polygons of each primitive. The overall number is very low, e.g., under 30 vertices in all cases. Also, there is no direct connection between the

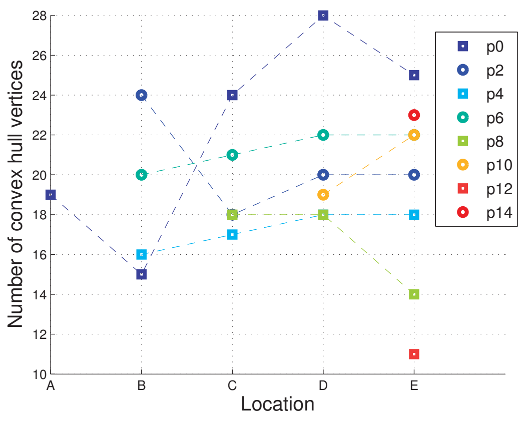


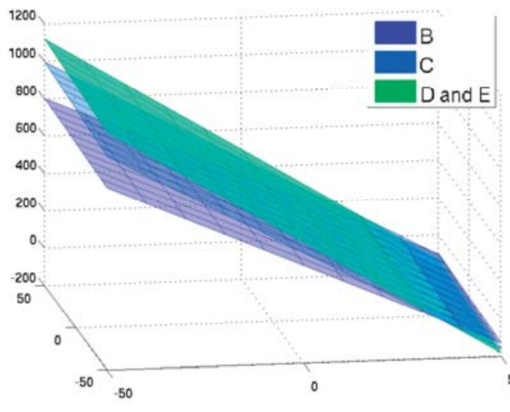
Figure 9.18: Analysis of the evolution of each of the polygons through sequence 1, in particular the number of vertices in the convex hull. Only pair index polygons are shown to simplify the graphs.

number of vertices and the number of support points or area, since some primitives actually decrease the number of bounding polygon vertices.

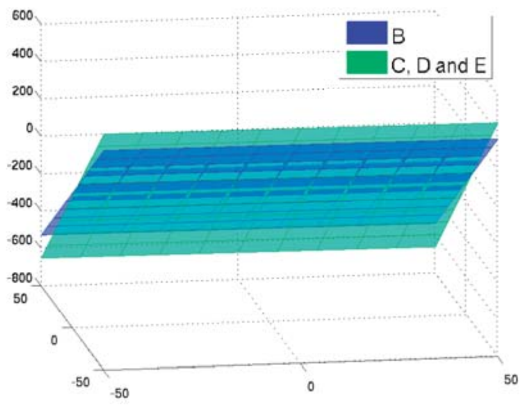
Figure 9.19 shows the evolution of the support planes throughout the sequence. Two examples are shown: polygon 2, Fig. 9.19 (*a*), and polygon 6, Fig. 9.19 (*b*). Both images show that the support planes can change their orientation as new data is received.

Figure 9.20 shows how primitive 0, i.e., the ground plane primitive, evolves over sequence 1. The primitive expands at every iteration to accommodate newly observed data points that belong to the ground plane.

Figure 9.21 shows two separate cases. In Fig. 9.21 (*a*) and (*b*) to consecutive locations in sequence 1 produce no noticeable changes in the primitives (marker in blue), since that the entire surface of the walls had already been completely covered by the primitive at the initial iteration. This shows that the expansion mechanism only grows the bounding polygon when it is actually necessary to accommodate new data. Figures 9.21 (*c*) and (*d*) show another example of two consecutive locations in sequence 2. Unlike in the previous case, this is an example of a noticeable growth of the primitives bounding polygon due to active primitive expansion. The primitives representing both walls alongside the road where the vehicle is travelling are expanded. The growth of these is clearly visible: from their initial state (Fig. 9.21 (*c*)) to the expanded state (Fig. 9.21 (*d*)).



(*a*)



(*b*)

Figure 9.19: Analysis of the updating of the support planes of primitives after an expansion. Sequence 1: (*a*) polygonal primitive 2; (*b*) polygonal primitive 6;

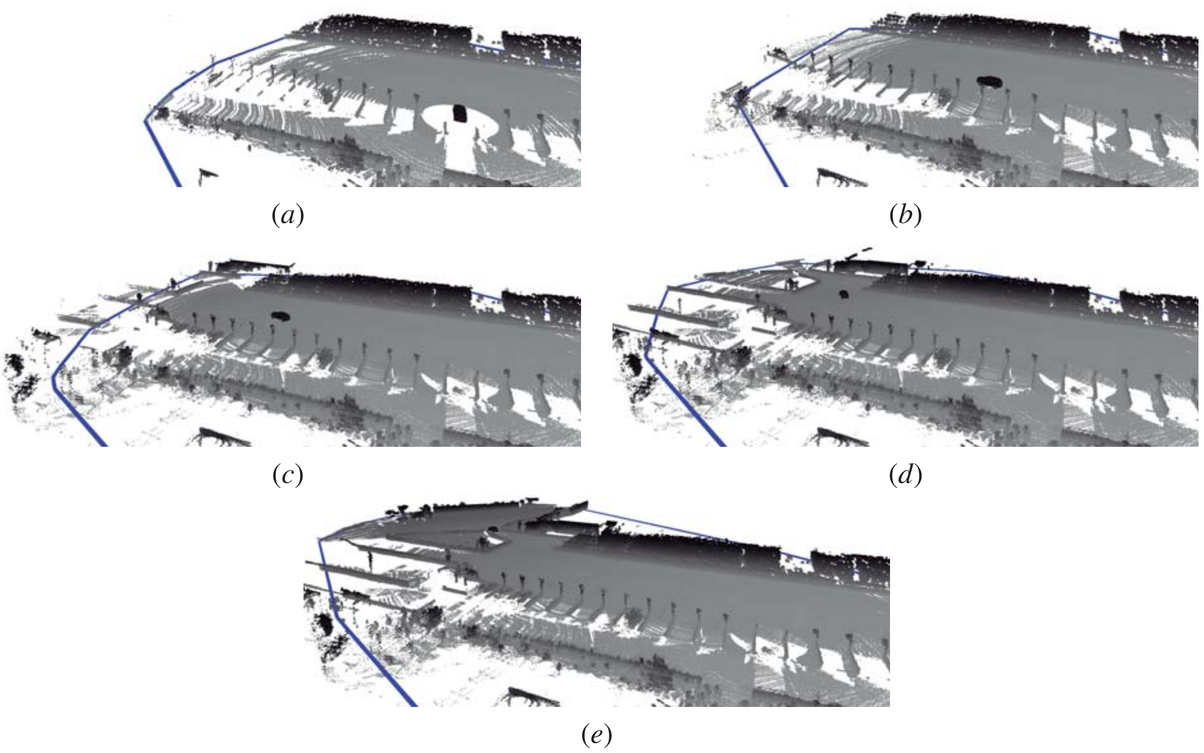


Figure 9.20: Evolution of polygonal primitive 0 (the ground plane) through sequence 1: (*a*) location *A*; (*b*) location *B*; (*c*) location *C*; (*d*) location *D*; (*e*) location *E*;

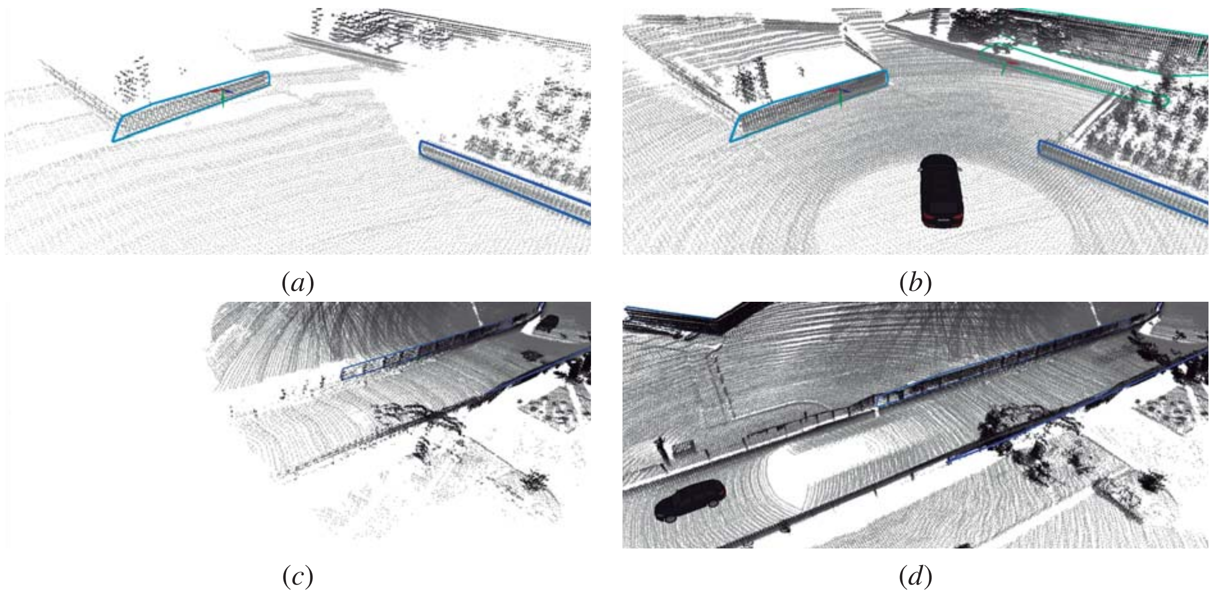


Figure 9.21: Evolution of polygonal primitives through sequence 1 (*top*) and sequence 2 (*bottom*): (*a*) location *C*, sequence 1; (*b*) location *D*, sequence 1; (*c*) location *G*, sequence 2; (*d*) location *H*, sequence 2.

9.5 Conclusions

This chapter presented a new approach to perform the refinement of geometric scene representations. A mechanism to expand the existing primitives according to new data was presented. The expansion mechanism is able to relieve processing load from the detection algorithms and efficiently grows existing primitives to accommodate new data. In conclusion, the proposed approach presents some very interesting characteristics when compared to standard reconstruction algorithms and is an interesting alternative to account for if real time applications are an objective.

Chapter 10

Photometric Scene Reconstruction

This chapter presents a methodology based on Data Dependent Triangulation (DDT) for the mapping of color from images onto the geometric polygonal primitive's based representation proposed in chapters 8 and 9. Several issues are addressed. An introduction to the problem is given in section 10.1. Section 10.2 describes traditional texture mapping techniques. In section 10.3, data dependent triangulations are proposed to solve the lack of accuracy of standard texture mapping techniques. Problems associated with computing the regions in the image that contain visual information about a given primitive are addressed. Finally, results are presented in section 10.4 and conclusions given in section 10.5.

10.1 Introduction

Chapter 8 presented a new approach that is capable of producing a geometric scene representation based on a set of geometric polygonal primitives. Furthermore, the proposed algorithms are also capable of updating the representation as new data arrives. The objective now is to increase the amount of information available to the scene representation. The reason for this is that 3D information is not the only information produced by sensors onboard a vehicle. While lasers, radars of sonars generate range (or 3D) information, there are other sensors that are capable of generating data of a different nature other than 3D. In general, there are many sensors that provide a measurement of some radiated energy along a given direction, and some of them provide useful information for navigation purposes. Some examples include RGB cameras (photometric data), thermal cameras (thermal data), infrared cameras (infrared spectrum data), ultraviolet cameras (ultraviolet spectrum data) amongst others. Typically, these sensors are passive sensors, meaning they measure energy emitted or reflected by objects or materials, but there are also examples of active sensors. It is the case of laser range finders that measure reflectance. It is also a measure of a material property along a defined direction. These sensors, or rather, the data they generate, have in common the fact that they capture some property of an object other than its 3D structure. In other words, they only provide directional information about the measured data.

In this chapter an algorithm for mapping photometric data to the geometric polygonal primitives is proposed. Photometric data is captured by RGB (or gray scale) cameras. The algorithm uses directional information along with the 3D representation of the scene in order to map color into the polygonal primitives. The algorithm employs the pinhole camera model (described in chapter 5) to compute the 3D direction from where each pixel has detected a given color. It then solves the so called inverse projection equation by resorting the support plane defined for each primitive. With this, it is possible to map color measured in the image to the geometric polygonal primitive in 3D.

The algorithm will be described in detail in the following sections. For now, we would like to stress that, as will be shown, the algorithm only requires a model of projection, and a geometric description of the scene, provided by the geometric polygonal primitives. Taking this into account, note that sensors that capture data of different nature may also be used to map other properties (other than the photometric) to the 3D representation. It is important to emphasize that, in theory, the algorithms that will be proposed in this chapter should also work with other cameras (thermal, infrared, ultra-violet, etc.), provided a model of projection is supplied for those cameras. In fact, in some cases, there is no need to provide a different projection model for the sensor. For example, it is shown in [Barrera *et al.* 2012b] and [Barrera *et al.* 2012a] that pinhole camera models may also be applied to thermal cameras with satisfactory results.

The perspective brings to mind a multi modal scene representation, where several different properties (geometry, photometry, thermal, etc.) are mapped into a single scene representation. For example, the algorithm would be not only able to perform typical texture mapping onto the geometry of the scene but also, for example, to map to the geometry of a given scene also the temperature at each point of the geometric surfaces. This broadens the potential of the proposed scene representation algorithm as well as the range of applications.

In conclusion, although images from RGB cameras will be used throughout the explanations in this chapter, with the proposed algorithm, it would be possible to map data of different nature to the geometry of the scene. In our opinion, this could be done with just minor little adaptations to the algorithm proposed here.

The Massachusetts Institute of Technology (MIT) data sets will be used to demonstrate the algorithm. The MIT *Talos* vehicle has five onboard cameras. For a detailed description, see Chapter 7.3.

10.2 Related Work

The first step involved in the extraction of the photometric properties is how to map images to the geometric primitives of a model. Images are obtained directly from the photometric sensors mounted onboard the vehicle, while the geometry description is provided by the algorithms described in chapters 8 and 9. Hence, given two inputs, an image and a geometry description, the objective is to obtain a new representation that not only has the geometry description but also the photometric description of the object it represents. All the geometric primitives described in chapter 8 are planar. In other words, both polygons (produced by the proposed approach) as well as triangles (produced by all other evaluated approaches) have each a corresponding support plane. This support plane of each primitive is used by the image projection algorithm, and is referred to as projection plane. Let a, b, c and d represent the coefficients of the Hessian form of the projection plane, and $\mathbf{I}$ be an image that is going to be projected. Image pixel coordinates are referred to as $\mathbf{u} = \{u, v\}$, respectively, columns and rows. The mapping of pixels to real world coordinates is called inverse projection, while the opposite operation is named direct projection. These topics were already addressed in chapter 5, so a detailed description will not be made. The important aspect is that, with a support plane and a description of the camera's position and intrinsic parameters, it is possible to map pixels to 3D points and vice versa. The mapping of 3D points ($\mathbf{X} = \{X, Y, Z\}$) into image pixels is called direct projection (`projection`):

$$\mathbf{u} = \texttt{projection}(\mathbf{X}), \tag{10.1}$$

while the mapping of pixels to 3D points is called inverse projection ($\texttt{projection}^{-1}$):

$$\mathbf{X} = \texttt{projection}^{-1}(\mathbf{u}). \tag{10.2}$$

If it is possible to map pixels to the projection plane, at first sight it seems trivial to obtain a representation with geometric plus photometric information: it should be a matter of applying eq. (10.2) to all the pixels in an image, and obtain all the corresponding 3D coordinates. This is partially true. There are two problems. The first is that the process is too slow, if all pixels are used. Images have hundreds of thousands of pixels and to apply the inverse projection to all pixels is not possible in real time. Second, the image represents a discrete set of measurements over the surface of the plane. Let us focus on the second problem for now, since the its solution also shreds some light into the unraveling of the first issue. Since image pixels are only discrete measurements of the photometry of the world, some interpolation must be made after they are projected to the projection plane. There are several interpolation techniques listed in the literature, such as nearest neighbor, bilinear or bicubic interpolations. We will not focus in detail on these techniques since they are considered standards in texture mapping applications. The key point here is that, no matter how high is the resolution of the camera, if the objective is to go from a discrete set of photometric measurements given by an image to a continuous representation of the photometry of the surface, then it is imperative that some interpolation takes place. Coming back to the first issue, we can say that even if all image pixels are projected, an interpolation would still have to be executed. This is an important observation because it leads to the following conclusion: if even when all image pixels are projected an interpolation must be conducted, then why not project less pixels as long as the interpolation can still obtain a good mapping of the color in the image.

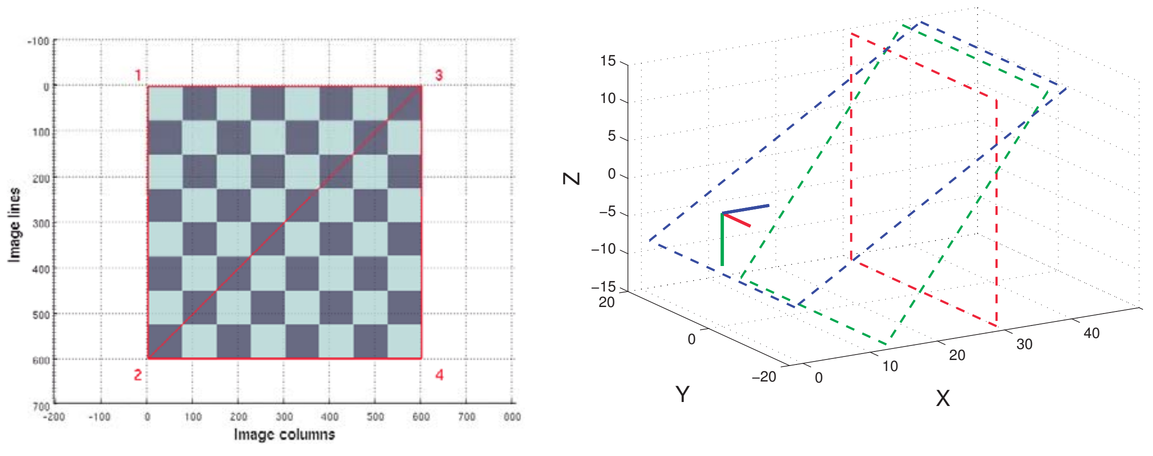
The standard technique to perform mapping of color from an image is called texture mapping. Texture mapping is a technique for mapping a 2D image onto a 3D surface by transforming color data so that it conforms to the surface plot. It allows the application of texture such as tiles or wood grain, to a surface without performing the geometric modeling necessary to create a surface with these features, or, in other words, without computing the projection of every pixel in the image onto the surface. The color data can also be any image, such as a picture taken by a camera. Texture mapping is performed over convex polygons, most commonly on triangles. Let $\mathbf{X}_1$, $\mathbf{X}_2$ be the coordinates of the vertices 1 and 2 in 3D space. The coordinates $\mathbf{u}_1$, $\mathbf{u}_2$ of the pixels that correspond to those vertices in the image plane can be obtained using direct projection:

$$\mathbf{u}_i = \texttt{projection}(\mathbf{X}_i), \quad \forall i \in \{1, 2\}. \tag{10.3}$$

Let α be a parameter $0 < \alpha < 1$, that indicates how a given vertex is positioned along the $\overline{\mathbf{X}_1\mathbf{X}_2}$ line segment. Texture mapping interpolates the color value for any vertices along the line segment as follows:

$$\mathbf{u}_\alpha = (1 - \alpha) \cdot \mathbf{u}_0 + \alpha \cdot \mathbf{u}_1, \tag{10.4}$$

which is of course a linear interpolation. When this kind of linear interpolation is used the texture



(*a*) (*b*)

Figure 10.1: Projecting a chessboard image onto several projection planes: (*a*) the image to be projected; (*b*) the projection planes and a representation of the camera's position as the red-green-blue reference system.

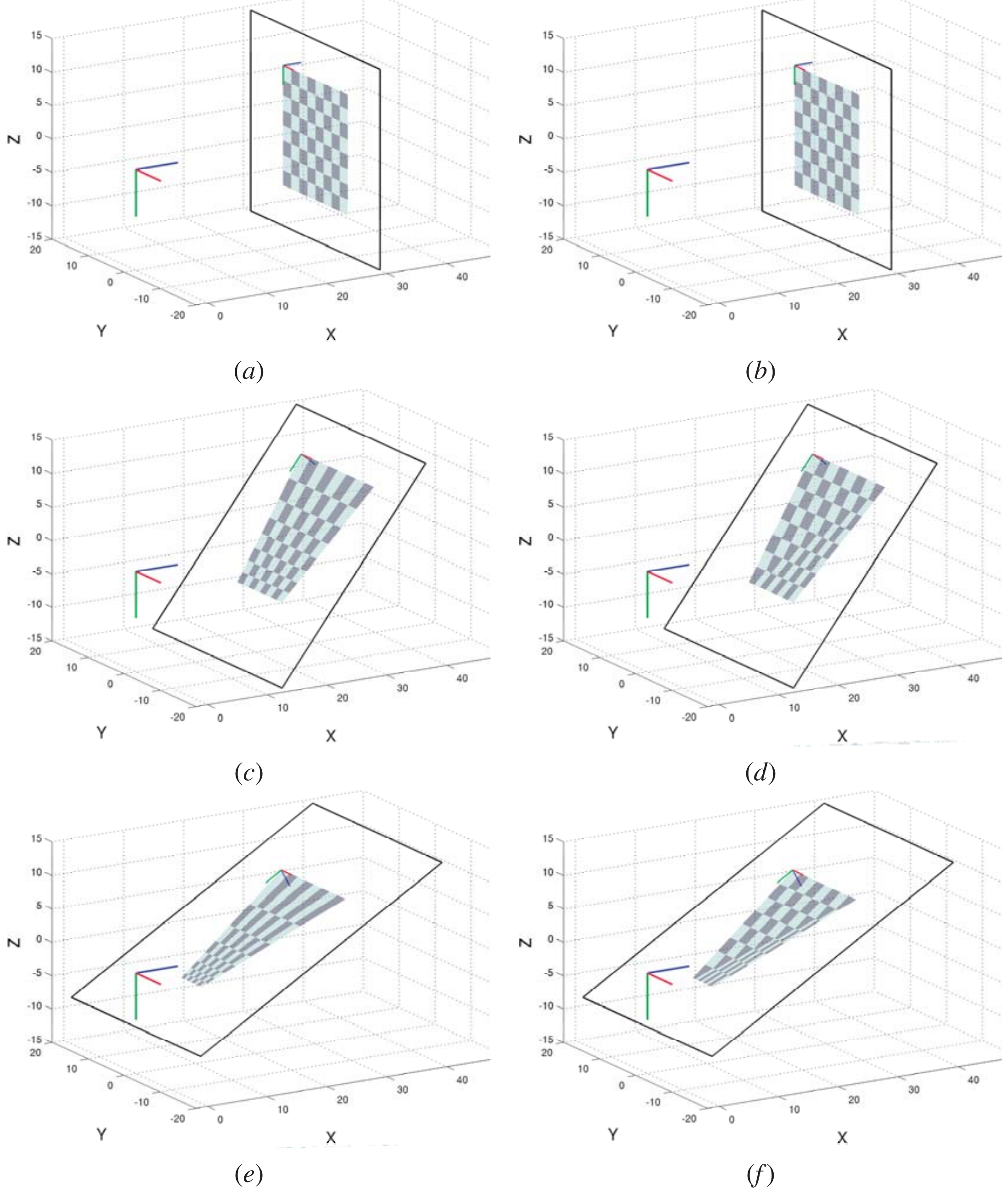


Figure 10.2: Projection using complete mapping versus texture mapping for the three projection planes displayed in Fig. 10.1 (*b*): (*a*) complete mapping, plane 1; (*b*) texture mapping, plane 1; (*c*) complete mapping, plane 2; (*d*) texture mapping, plane 2; (*e*) complete mapping, plane 3; (*f*) complete mapping, plane 3.

mapping is referred to as affine texture mapping. A linear interpolation works fine when the image plane and the projection plane are parallel. However when this does not occur, the projection shows some artifacts that derive from the assumption that a linear interpolation can be used. This is a well documented problem, and is discussed in several works [Segal *et al.* 1992] [Debevec *et al.* 1998]. Figure 10.1 (*a*) shows a chessboard image, which will be used in the forthcoming examples. The image is a square chessboard of 8 by 8, with a resolution of 800×800 pixels. Figure 10.1 (*a*) also shows how the image is divided into two triangles, e.g., triangle 1,2,3 and triangle 2,3,4. Both triangles are independently projected using affine texture mapping, to each of the three planes represented in Fig. 10.1 (*b*). The position of the camera is also shown by the red-green-blue lines, corresponding to the XYZ axes. We will label each of the planes as 1, 2 and 3, presented in Fig. 10.1 (*b*) in colors red, green and blue, respectively. Hence, plane 1 (in red) is parallel to the image plane, while plane 2 (in green) is slightly more oblique and plane 3 (in blue) is an even more oblique plane with respect to the image plane.

Figure 10.2 shows the projected images for each of the three planes. On the left column, the projection is obtained by computing the projections of all the pixels in the image. This will be referred to as complete mapping, which, as said before, is not adequate for real time processing. It is, however, important to be used as ground truth. On the right column of Fig. 10.2 the projections obtained using affine texture mapping are displayed. From the analysis of 10.2 it is possible to observe that when the projection plane is parallel to the image plane, both the complete mapping (Fig. 10.2 (*a*)) and the texture mapping (Fig. 10.2 (*b*)) present similar results. In the other two planes noticeable artifacts in the texture mapping appear. This is due to the linear interpolation discussed before. One good way to visualize the effect of linear interpolation is by the size of the squares in the case of plane 3. Since the plane is oblique with respect to the image plane pixels on the top of the image should represent a wider viewed area, which is why they are stretched in the complete mapping (Fig. 10.2 (*e*)), in compliance with projective laws. In the case of affine texture mapping (Fig. 10.2 (*f*)), this is not taken into account which makes the squares appear with a similar size, as in the original image.

10.3 Proposed Approach

This section presents the algorithms used to map texture onto the geometric polygonal primitive's representation.

10.3.1 Data Dependent Triangulation

It was shown if Fig. 10.2 that affine texture mapping is not capable of accurately mapping the texture of an image when the projection plane is not parallel to the image plane. However, there is a standard way to solve this problem. The solution is called view dependent texture mapping, and it consists of making texture mapping account for the position of the vertices in 3D space, rather than simply interpolating a 2D triangle. This achieves the correct visual effect, but it is slower to calculate. Instead of interpolating the texture coordinates directly, the coordinates are divided by their depth (relative to the viewer), and the reciprocal of the depth value is also interpolated and used to recover the perspective corrected coordinate. This correction operates so that parts of the polygon that are closer to the viewer, the difference from pixel to pixel between texture coordinates is smaller (stretching the texture wider), and in parts that are farther away this difference is larger (compressing the texture). View dependent texture mapping can be formulated as:

$$\mathbf{u}_\alpha = \frac{(1-\alpha)\cdot\frac{\mathbf{u}_0}{w_0} + \alpha\cdot\frac{\mathbf{u}_1}{w_1}}{(1-\alpha)\cdot\frac{1}{w_0} + \alpha\cdot\frac{1}{w_1}}. \tag{10.5}$$

The solution proposed in eq. (10.5) is capable of producing accurate mapping for texture. View dependent texture mapping is about 16 times the computational power of affine texture mapping. However, it is a standard solution implemented on current of the shelve graphics cards.

Even though there seems to be an established methodology, we propose an alternative solution. Some of the reasons for this will be detailed in the following lines, and some others will only be completely developed in chapter 11, when the mechanism for the expansion of photometric properties is introduced. In chapter 8, a new geometric scene representation was proposed. One of the major differences from the proposed method with respect to other well known methods (which were also evaluated in that chapter) is that the geometric primitives consist of polygons, instead of the traditional triangles. As explained before, the mapping of photometric properties is performed by mapping triangles in image space to 3D space. These procedures are executed in the graphics cards, and programmed using OpenGL [Opengl *et al.* 2005], Direct3D [Blythe 2006] or other graphics libraries. These libraries also have the functionalities of mapping convex polygons, but in fact these are mere high level functions that decompose the polygons into sets of triangles and then map the triangles. If we plan to use the geometry description from the polygonal primitives, the question is if it is preferable to have the graphics card/library decompose the polygons as it sees fit or if there is any advantage in performing triangulation of those polygons that describe the geometric polygonal primitives. We believe there is a clear advantage in performing the triangulation in a specially devised routine, since in this way it is possible to apply the most interesting triangulation criteria or methodology, given the objective. The objective, let us remind, is to compute a scene representation as efficiently as possible, so that real time is achievable. It is also very important that the memory required by such primitives is as small as possible. In other approaches, such as Ball Pivoting Algorithm (BPA), Greedy triangulation (GT) or Poisson Surface Reconstruction (POIS), the primitives that are used are triangles and hence it does not make sense to discuss how to triangulate again. So the mapping of texture onto geometry provided by other scene reconstruction approaches is straightforward: for each triangle in 3D space, compute the corresponding vertices in the image plane and execute texture mapping from this. We propose to execute the reverse procedure: to triangulate over the image space and then to map these triangles computed in image space to the 3D space. This is interesting because triangulation which serves only texture mapping purposes can be computed in the data space where texture is collected, the image. If triangulation is performed over the image it is possible to select the triangles that should be better suited for texture mapping purposes. In the next few lines, the algorithm will be described in detail, but for now some other questions may arise. One of them is why to spend computational resources in creating geometric polygonal primitives only to require in the end, as all other approaches do, a triangulated mesh. The answer is that, unlike in all other approaches, where a 3D mesh is defined, the triangulation over the image space is done in two dimensions. This factor considerably speeds up the computation. Also, because only a few pixels are defined to become vertices of the mesh, the number of points is considerably smaller than the amount of 3D data points that would be used for executing a 3D triangulation. Note that only geometric polygonal primitives provide the flexibility of leaving triangulation for later, which in turn can be used in the advantage of overall computational performance. It was established that it is required, in the case of geometric polygonal primitives, to execute a 2D triangulation step prior to texture mapping. The question now is how to choose these triangles. If the triangles are especially defined so that their faces represent smooth regions with constant color (as shown in Fig. 10.3 (*a*)), then, an affine texture mapping over these could in fact provide accurate projections. This procedure is referred to in the literature as DDT [Lehner *et al.* 2007], and the mapping of images using this technique will be referred to as DDT mapping as opposed to texture mapping.

Figures 10.3 (*b*), (*c*) and (*d*) show, respectively for planes 1, 2 and 3 (represented in Fig. 10.1 (*b*)), the DDT mappings performed over the triangles shown in Fig. 10.3 (*a*). It is possible to see that there are no artifacts as those shown for the affine texture mapping in Figs. 10.2 (*b*), (*d*) and (*f*). Note that the color inside the triangles is being obtained by linear interpolation, similarly to what occurs in affine texture mapping. The difference is that, because we are aware of the limitations of linear interpolation, triangles with smooth color transitions are chosen so that the linear interpolation does not create any artifacts, since it is performed over vertices with very similar color.

Figures 10.4 shows a comparison of the projections obtained by texture mapping (red diamonds) and DDT mapping (blue squares). Only the projections of the corners of the chessboard are shown. The background texture is provided by the complete mapping of all the pixels, so it should be considered as the ground truth. Planes 1, 2 and 3 are shown in Figs. 10.4 (*a*), (*b*) and (*c*), respectively. The DDT mapping is clearly more accurate than affine texture mapping. The lack of accuracy in affine texture mapping is higher in plane 3 and almost very low in plane 1. This is because, as discussed before, plane 1 is parallel to the image plane and no perspective effects are associated with this transformation.

In order to evaluate what is the influence of the orientation of the projection plane to the accuracy of projection, a test was conducted. In this test a set of sixteen projection planes was defined. For each, the texture and DDT mappings are compared to the complete mapping. Let the projection error (ϵ) be the average distance between vertices in 3D space obtained using one mapping that is

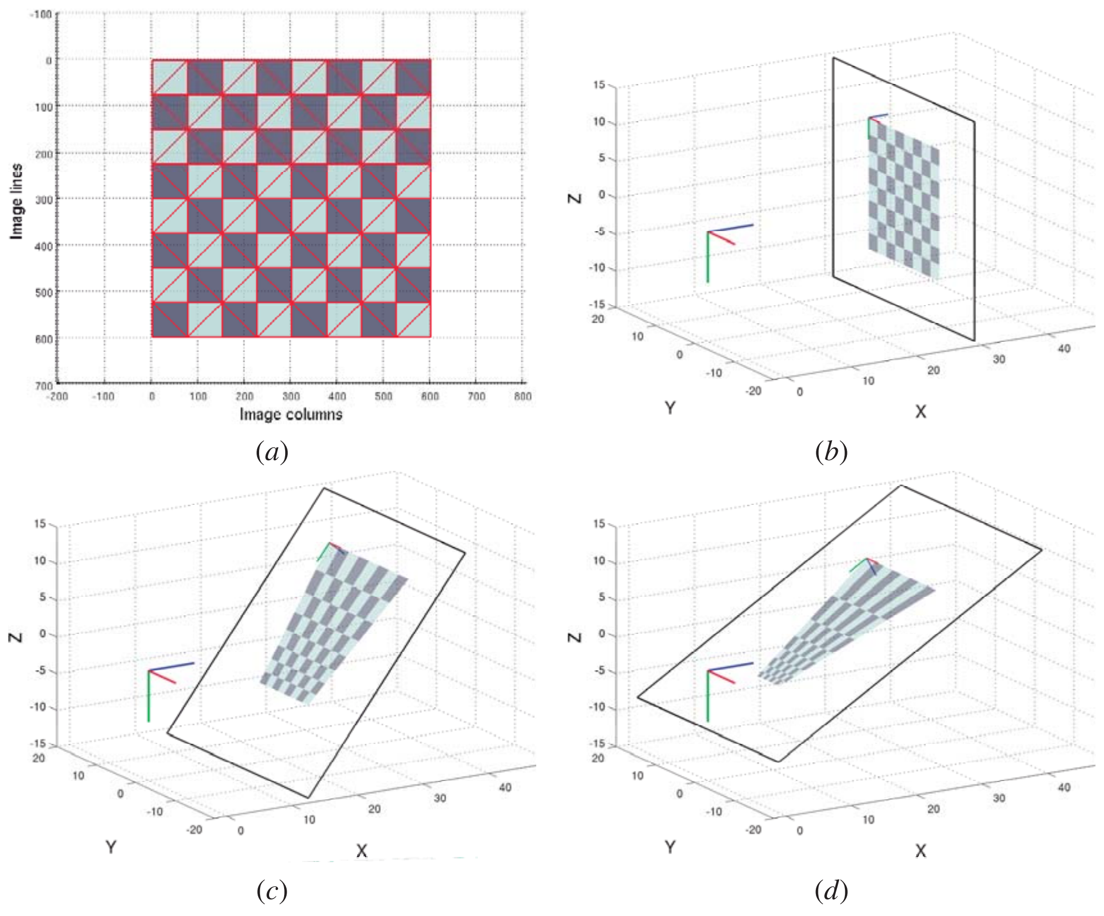
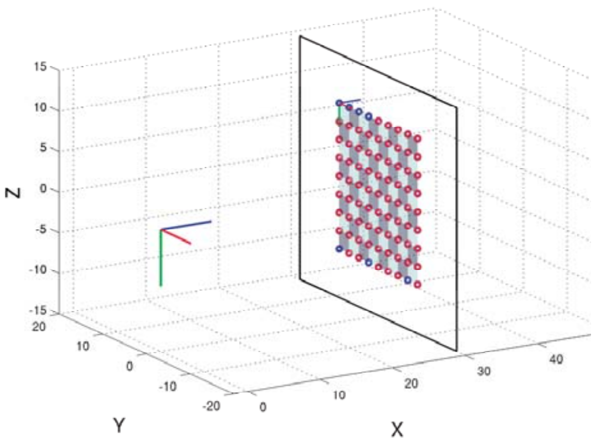


Figure 10.3: Projection of the chessboard image using DDT mapping: (a) DDT triangulation ; (b) DDT mapping, plane 1; (c) DDT mapping, plane 2; (d) DDT mapping, plane 3.

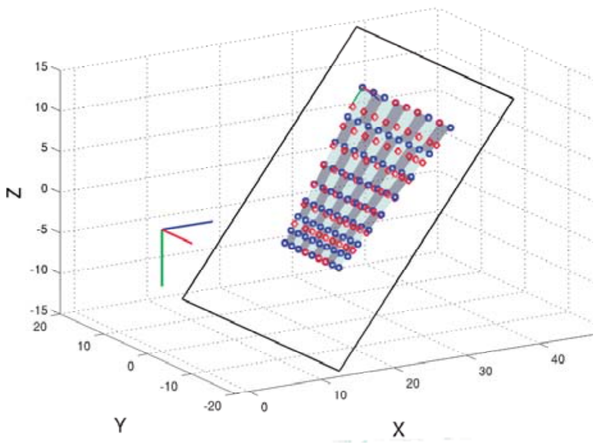
considered ground truth map_{GT} and any other mapping (map):

$$\epsilon = \frac{\sum_{\mathbf{u}=\{1,1\}}^{\{W,H\}} \mathbf{dist}\big(map_{GT}(\mathbf{u}), map(\mathbf{u})\big)}{W \times H}, \tag{10.6}$$

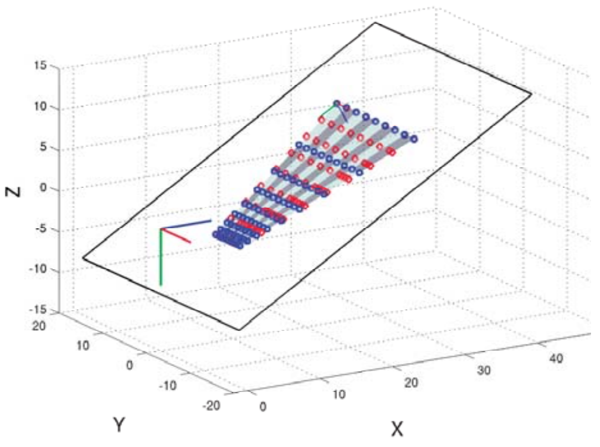
where W and H are the images width and height, respectively, and $\mathbf{dist}(a, b)$ is a function that retrieves the Euclidean distance between points a and b in 3D space. The projection of all pixels is considered the ground truth mapping. Hence, the projection error of both texture mapping and DDT mapping can be evaluated. Each tested plane is characterized by the angular difference its normal has to the images optical axis. The angular difference is regarded as the minimum angle between those vectors. Figure 10.4 (d) plots the projection error as a function of this angular difference. Both texture mapping and DDT mapping results are displayed. As was expected by the observations of the projections, DDT mapping shows a very low average projection error regardless of the plane's



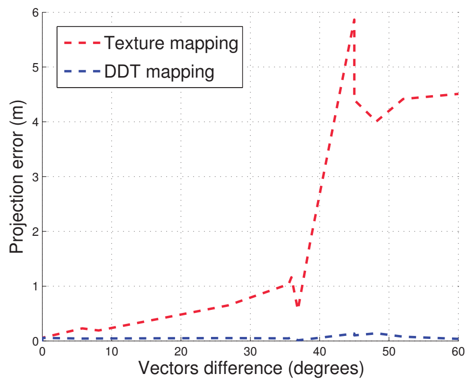
(*a*)



(*b*)



(*c*)



(*d*)

Figure 10.4: Comparison of the texture mapping (red diamonds) and DDT mapping (blue squares). Background projection is obtained by performing the complete mapping of pixels, so it may be regarded as ground truth. Only the chessboard corners are shown. The three projection planes displayed in Fig. 10.1 (*b*): (*a*) plane 1; (*b*) plane 2; (*c*) plane 3; (*d*) the average projection error as a function of the difference between the image optical axis vector and the plane's normal vector.

orientation. The same is to say that DDT is accurate and robust to changes in the orientation of the projection plane. In the case of texture mapping the angular difference highly affects the projection error. Although a direct correlation cannot be established, the projection error seems to increase with the increase in the angular difference.

An image can be viewed as a function ($\mathbf{f}$) over a two dimensional domain, i.e., $\mathbf{f} : \mathbb{R}^2 \rightarrow \mathbb{R}$, where the inputs are the image row and column coordinates and the output is the color of the image. A triangulated mesh $\mathtt{T}$ over $\mathbb{R}^2$ has the objective of approximating the values of $\mathbf{f}$ by interpolating over the vertices of every triangle t of $\mathtt{T}$. The quality of the approximation, that is, the distance between the interpolation and of the triangle's vertices and the actual value of $\mathbf{f}$, is defined in particular by three factors, that are:

- the number of vertices in $\mathtt{T}$;
- the position of those vertices;
- the connectivity between those vertices, i.e., the shape of the triangles.

The goal of a data dependent triangulation is, on the one hand, to obtain the best approximation possible, and on the other to reduce the number of triangles and in turn the memory load. Consequently, the number of vertices should be kept as small as possible to speed up processing and reduce memory load. The two variables that should be tunned to achieve a good approximation are then the position of the vertices and the connections between them. Even if we decide to fix the number of triangles and vertices, the possible combinations of the connections between vertices are usually very large. Hence, an exhaustive search of all possible combinations is not possible. Also, no assumptions should be made on the optimal shape or size of the triangles. One might tend to assume long, thin triangles are not adequate but in fact that depends on the nature of the image [Rippa 1992]. If the image contains high gradient long feature such as poles or trees, such triangles could be well suited to represent these regions.

DDT algorithms can be divided into refinement, decimation, or modification approaches. In refinement approaches, the starting point for the algorithm is a very coarse triangular mesh that is then refined. The mesh is refined by inserting new vertexes. Since the number of possible positions where vertices can be inserted is very high, authors make use of heuristics to limit the number of options. The greedy refinement algorithm proposed in [Garland & Heckbert 1995] works by inserting vertices into a triangulated mesh. In every step a new vertex is inserted at the position of the largest distance between the approximation and the data provided in the image. In [Schätzl *et al.* 2001], the choice of which triangles to decimate is based of the high curvature of the data, and the positions where new vertexes are inserted are locations with high approximation distance to the data. These methods have the drawback of tending to a local optima. Decimation approaches are the opposite of refinement meshes. The algorithms start from a very fine mesh and try to remove vertices and collapse triangles as they iterate. In [Hoppe 1996] the initial triangulation is a full triangulation where each pixel is a vertex in the mesh. The algorithm then decimates the mesh by collapsing one of the edges of the mesh. The edge to collapse is the one that implicates less increase in the approximation error. Similar approaches were proposed in [Demaret *et al.* 2006] and [Sappa & García 2007]. Finally, modification strategies start from a random arbitrary mesh and try to improve it by performing modification operations. These modification operations usually are edge swaps and the number of vertices in the initial mesh remains the same. It is the case of the algorithms proposed in [Dyn *et al.* 1992] and [Schumaker 1993]. Both propose different criteria for the selection of which are the edges that should be swapped.

Although there are many approaches in the literature to the data dependent triangulation problem, most of them are focused on the fact that such a triangulated mesh is capable of producing very good data compression ratios with respect to the real image, while still maintaining low approximation errors. Real time performance of the algorithms has seldom been debated, with authors reporting processing times of over three seconds for 512×512 images. The exception was the study conducted in [Cervenansksy *et al.* 2010], where DDT was parallelized and executed on accelerated graphics cards, resulting in a significant speed up. Nonetheless, the fact is that most DDT approaches are slow to process, which is not adequate for the problem at hand, where DDT is just a small portion of the entire scene reconstruction algorithm. In this work, DDT must be quite fast, even if it means finding a less than optimum solution for the triangulation. Because of this we propose a very simple procedure that is based on the philosophy of data dependent triangulations. The procedure is very similar to the one presented in [García *et al.* 2000]. As discussed before, there are three alternatives for designing a data dependent triangulated mesh. The first is the number of vertices, which we want to keep as low as possible, the second is where those vertices are located, and the third concerns the connections between those vertices, i.e., the edges of the triangles. The connectivity between edges is usually obtained using iterative procedures and this is one of the factors responsible for the high processing times. The procedure should run in a single step, so edge connectivity will not be optimized.

The question that remains is then, what are the pixels in the image that should be considered vertices in the triangulation. Note that, in order to have a good texture mapping performance, faces should cover, as much as possible, constant color areas in the image. Given this, the algorithm we propose is quite simple: to attempt to place the edges of the triangulated mesh over regions in the image where edges have been detected. Hough lines have long been used for detecting line features in images [Svalbe 1989]. There have also been many proposals to extend the algorithm to obtain a description of line segments instead of lines [Kamat & Ganesan 1998] [Guerreiro & Aguiar 2011]. Hence, it is straight forward to obtain a list of starting and ending points of line segments that describe edges in an image.

A Delaunay triangulation for a set of points in a plane is a triangulation such that no point in the set of points is inside the circumcircle of any triangle defined in the triangulation, which is called the empty circle or Delaunay property. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid skinny triangles. Delaunay triangulation are widely used in computer graphics and other fields, and there are many available toolboxes that implement the algorithm. In this particular case, the toolbox from [Yvinec 2012] is used to provide the triangulation code. So it is possible, using a Delaunay triangulation, to compute a triangulated mesh from a set of points. These points are used as input by the algorithm and become vertices of the triangulated mesh.

Let a given image be described by M line segments with starting points $\mathtt{s}_m$ and endpoints $\mathtt{e}_m$, where each detected line segment is defined as $\overline{\mathtt{s}_m\mathtt{e}_m}$. Figure 10.5 (*a*) shows an image of a building, to be used as an example. The blue lines show the triangulated mesh defined for the purpose of a naive affine triangulation (as shown in Fig. 10.1 (*a*)). Figure 10.5 (*b*) shows the line segments obtained using Hough based algorithms: line segments are represented by the dashed red lines and starting and endpoints by the green circles. One first option would be to use the list of starting and endpoints to define the vertices of the triangulation. A function that would perform a Delaunay triangulation (**Delaunay**) could be written as:

$$\mathtt{t} = \textbf{Delaunay}\big(\{\mathtt{s}_0, \mathtt{e}_0, \mathtt{s}_1, \mathtt{e}_1 ..., \mathtt{s}_{M-1}, \mathtt{e}_{M-1}\}\big), \tag{10.7}$$

where $\mathtt{t}$ would be the resulting triangulated mesh. Figure 10.5 (*c*) shows the result of a Delaunay triangulation (using the vertices represented in Fig. 10.5 (*b*)): the mesh is represented by the blue lines, and the vertices are shown as green circles. The objective is to align the edges of the triangles with high gradient regions of the image. Since only the information about the vertices position was given to the Delaunay triangulation this does not occur. Note for example the region where the roof of the house changes to the blue sky. Here, there are several triangles whose faces cross the frontier from the roof to the sky. If a linear interpolation was to be made over these triangles, it would result in an inaccurate mapping as shown in Fig. 10.4. To solve this problem we propose to employ a constrained Delaunay triangulation mechanism. A constrained Delaunay triangulation is a generalization of the Delaunay triangulation that forces certain required line segments into the triangulation. They were proposed by [Chew 1987] for two dimensional spaces and later generalized to N dimensional spaces by [Shewchuk 2008]. Because a Delaunay triangulation is almost always unique, often a constrained Delaunay triangulation contains edges that do not satisfy the Delaunay condition. Thus a constrained Delaunay triangulation often is not a Delaunay triangulation itself, meaning it does not fulfill the empty circle property. This is not a problem for the application at hand since, as discussed before, there is no a priori knowledge about the preferable shape the triangles should have. A constrained Delaunay triangulation function (**cDelaunay**) requires two inputs, a list of vertices and a list of line segments that should exist in the triangulation. These imposed line segments are called constraints. We propose to build a constrained Delaunay triangulation using as input information given by the Hough lines detection algorithm:

$$\mathtt{t} = \textbf{cDelaunay}\big(\{\mathtt{s}_0, \mathtt{e}_0, \mathtt{s}_1, \mathtt{e}_1 ..., \mathtt{s}_{M-1}, \mathtt{e}_{M-1}\}, \{\overline{\mathtt{s}_0\mathtt{e}_0}, \overline{\mathtt{s}_1\mathtt{e}_1}, ..., \overline{\mathtt{s}_{M-1}\mathtt{e}_{M-1}}\}\big), \tag{10.8}$$

where the first argument of the **cDelaunay** function is the list of vertices and the second argument corresponds to the list of constraints. Figure 10.5 (*d*) shows the mesh obtained using a constrained Delaunay triangulation: the mesh is shown in blue, vertices as green circles, and constrained edges in the triangulation have a red line in the middle to signal their constrained status. If the same region of the image (the roof and the sky) that was analysed before is now observed, the advantages are clear.

In this case, there are no triangle faces crossing the frontier between the sky and the roof, triangle edges are aligned with the edges in the image. This leads to a more accurate texture mapping of the triangles: since triangles contain faces with very similar colors or patterns, a linear interpolation inside these triangles will not produce artifacts.

Figure 10.6 compares the projections obtained using: complete mapping (left column); affine texture mapping over a naive triangulation (middle column); and affine texture mapping over the proposed DDT triangulation (right column). The projection planes are the same as those shown in Fig.10.1 (*b*). It is possible to observe that affine texture mapping using naive triangulation (middle column) presents many differences with respect to the ground truth (left column), the DDT texture mapping (right column) obtained very similar projections to those given by the ground truth. Results

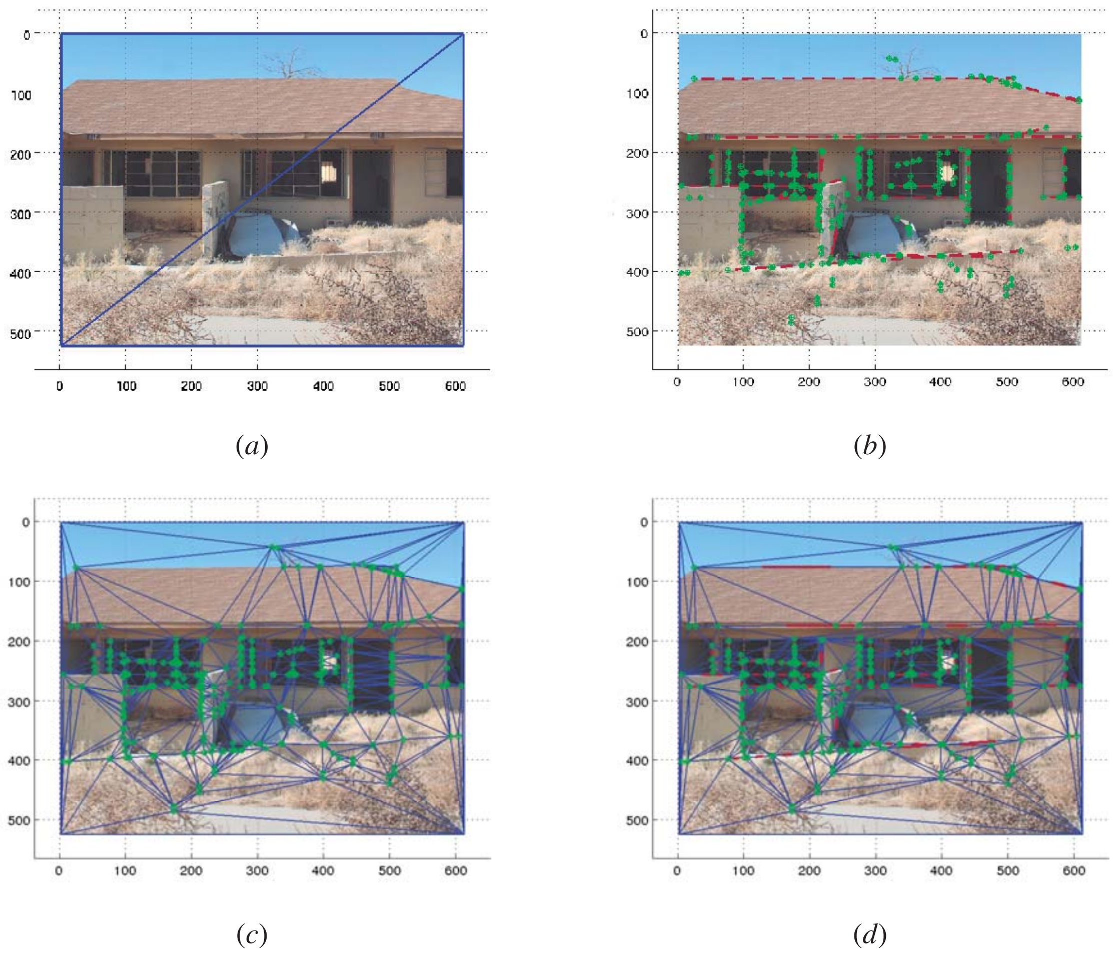


Figure 10.5: Data dependent triangulation: (*a*) naive triangulation; (*b*) line segments detected using Hough transform; (*c*) Delaunay triangulation using line segments vertices as input; (*d*) constrained Delaunay triangulation using vertices and line segments as input.

from the affine texture mapping are, as discussed before, less accurate for more oblique planes (third row).

One mention should also be made to the data compression effectiveness of DDT. In standard image representation, all the pixels contain color information. Assuming an RGB image with 8 bit depth information, the required memory for standard image representation is (M_{std}):

$$M_{std} = \underbrace{H \times W}_{number\,of\,pixels} \times \underbrace{3}_{RGB} \times \underbrace{8}_{type\,char}, \tag{10.9}$$

where H and W are the image's height and width respectively. On the other hand, the required memory for a DDT image representation (M_{ddt}) is:

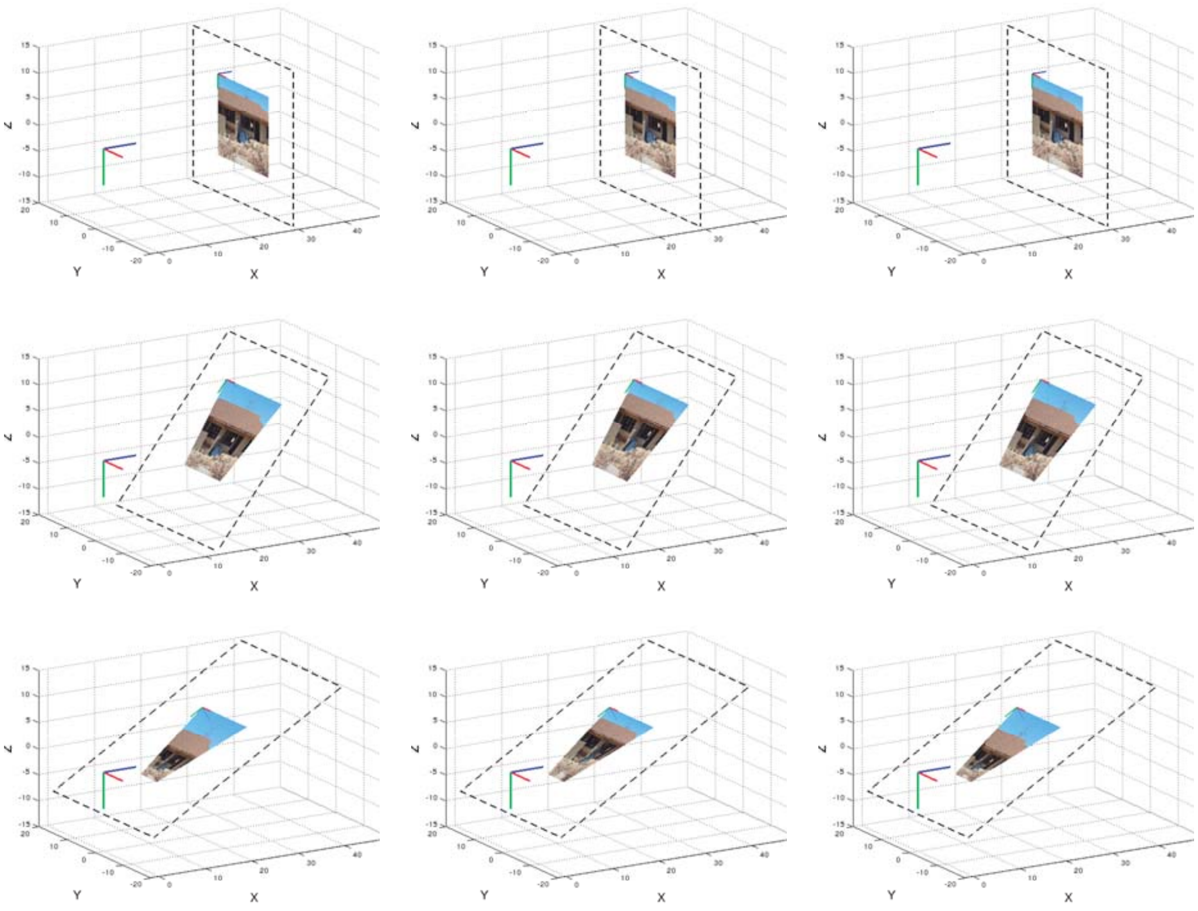


Figure 10.6: Projection using complete mapping (*left column*), texture mapping (*middle column*) and data dependent triangulation mapping (*right column*). The three projection planes displayed in Fig. 10.1 (*b*) are shown: planes 1, first row, plane 2, second row, and plane 3, third row.

$$M_{std} = \underbrace{V}_{number\,of\,vertices} \times \underbrace{3}_{RGB} \times \underbrace{8}_{type\,char}, \tag{10.10}$$

where V is the number of vertices in the triangulation. Using eqs. (10.9) and (10.10), the compression ratio of DDT (R_{ddt}) is given by:

$$R_{ddt} = \frac{M_{ddt}}{M_{std}} = \frac{V}{H \times W}, \tag{10.11}$$

which means that the compression of a DDT, depends on the ratio between the number of vertices in the triangulated mesh and the number of pixels. In the example shown in Fig. 10.5, the image has $H = 524$, $W = 611$, and the DDT mesh contains 652 vertices. The compression ratio is then about 0.2%, which is a very good value. It is also possible to obtain a finner mesh if the detection of line segments is tunned to be more sensitive.

As a conclusion, we can say that texture mapping is a standard technique for applying information from images on 3D models. However, affine texture mapping does pose problems when the projection plane is oblique with respect to the image plane. To solve this issue, we propose to use a DDT technique that is capable of triangulating an image in such a way that the linear interpolation performed by texture mapping in each of the triangles does not affect the accuracy of the projection.

10.3.2 Image Bounding Polygon

Section 10.2 discussed in detail the texture mapping technique. In particular, it showed how a triangle defined in the image can be projected to the 3D world. Using, this tool, the next step is to be able to split the image space into a set of triangles. To avoid problems with affine texture mapping onto oblique planes, a special technique called DDT was presented in section 10.3.1. DDT is capable of triangulating the image taking into account high gradient regions in the image and avoiding the creation of artifacts in the textured 3D surface.

But the problem, as presented in sections 10.2 and 10.3.1, was in fact simplified. It was assumed that the entire image viewed the surface to which the color had to be projected. In other words, it was assumed that the image viewed only the surface of the polygon. It was because of this simplified assumption that the examples in Figs. 10.2 and 10.6 show the entire image projected to the projection plane.

At first glance, one would think that the entire image given by a vehicle's onboard camera is potentially capable of showing a view of the primitive. The question is, if the camera is facing the primitive should the entire image be considered as a candidate to contain visual information of that primitive. Albeit one distracted answer would be yes, the entire image can be used, the fact is that in real applications this is not the fact. In some cases, the camera is mounted in positions so that only a portion of the image views the scene around the vehicle, while the other portion captures a part of the vehicle itself. Figure 10.7 shows a couple of examples. In Fig. 10.7 (a), an image from the stereo camera of the *AtlasCar* is shown. The lower part of the image views the front of the *AtlasCar* body. In Fig. 10.7 (*b*), an image front the *Talos* forward center camera also views portion of the vehicle, in this case a mechanical structure mounted in the front of the vehicle to accommodate Laser Range Finder (LRF) sensors.

One could argue that the cameras should be positioned so that they only view the outside scene, but this can be a difficult exercise especially in vehicles such as the *Talos*, where the large number of sensors makes it nearly impossible to find a combination of positions for all sensors so that none of them captures part of the vehicle. Partial occlusion of onboard sensors by the host vehicle is a real problem, and a solution must be devised to deal with it. The solution to this problem is to compute which portion of the sensor data is viewing the vehicle, and then to discard it.

Let us consider the case of a single camera onboard a vehicle travelling a given scenario. Three coordinate frames are defined: the world (**W**), vehicle (**V**) and camera (**C**) coordinate frames. In section 10.2, eq. (10.1) provided an overall view of the direct projection mechanism. Considering the mechanism in detail, we can say that pixels from an image are to be mapped onto the world coordinate frame, that is:

$$\mathtt{x} = \mathtt{projection}(\mathcal{X}), \tag{10.12}$$

where $\mathtt{x}$ is the position of a point / pixel, thus defined in the camera coordinate frame, and $\mathcal{X}$ is a 3D point / vertex defined in the world coordinate frame. Note that the 3D point is defined over $\mathbb{R}^3$ space, while the pixel is defined in $\mathbb{R}^2$ space (although its position in the image is actually in the $\mathbb{N}^2$ space, it is converted to $\mathbb{R}^2$ space). In detail, eq. (10.12) is composed of:

$$\mathtt{x} = \mathbf{K} \cdot {}^{\mathbf{C}}\mathbf{T}_{\mathbf{W}} \cdot \mathcal{X}, \tag{10.13}$$

where $\mathbf{K}$ is the intrinsic matrix of the camera and ${}^{\mathbf{C}}\mathbf{T}_{\mathbf{W}}$ is the transformation between the camera

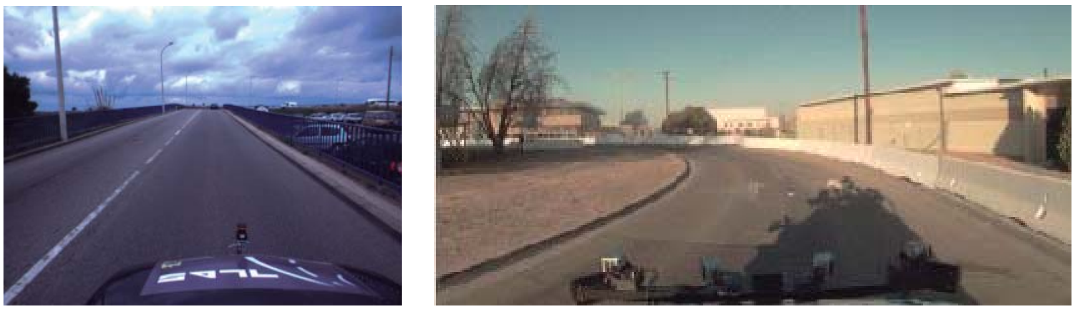

(*a*) (*b*)

Figure 10.7: Images from onboard cameras sometimes view portions of the host vehicles: (*a*) *AtlasCar* stereo camera; (*b*) *Talos* forward center camera.

coordinate system and the world coordinate system. Note that we use handwritten notation, e.g., $\mathcal{X}$, for the variables represented in $\mathbb{R}^3$ space, and machine type notation, e.g., $\mathtt{x}$, for variables defined in $\mathbb{R}^2$ spaces. The $^{\mathbf{C}}\mathbf{T}_{\mathbf{W}}$ transform can in turn be replaced by its corresponding lower order transforms:

$$\mathtt{x} = \mathbf{K} \cdot ^{\mathbf{C}}\mathbf{T}_{\mathbf{V}} \cdot ^{\mathbf{V}}\mathbf{T}_{\mathbf{W}} \cdot \mathcal{X}, \tag{10.14}$$

where $^{\mathbf{C}}\mathbf{T}_{\mathbf{V}}$ is the transformation from a camera's coordinate frame to the vehicle's coordinate frame, and $^{\mathbf{V}}\mathbf{T}_{\mathbf{W}}$ is the transformation from the vehicle to the world coordinate frames. Let us now consider a vehicle moving around a scenario. While the vehicle is moving, images acquired from the cameras change. As a conclusion, something in equation (10.14) must change also. Intrinsic camera values ($\mathbf{K}$ matrix) are fixed and, since the camera is rigidly fixed to the vehicle structure, also the $^{\mathbf{C}}\mathbf{T}_{\mathbf{V}}$ transform is fixed, regardless of the vehicles position. The only quantity in eq. (10.14) that varies with vehicle position (or mission time t, considering that the vehicle is moving) is the $^{\mathbf{V}}\mathbf{T}_{\mathbf{W}}$ transform. This makes perfect sense, as the vehicle moves around, so will the transform from the vehicle to the world coordinate frames be updated. We will notate this time dependency of the transform as $^{\mathbf{V}}\mathbf{T}^{t}_{\mathbf{W}}$, which turns eq. (10.14) into:

$$\mathtt{x} = \mathbf{K} \cdot ^{\mathbf{C}}\mathbf{T}_{\mathbf{V}} \cdot ^{\mathbf{V}}\mathbf{T}^{t}_{\mathbf{W}} \cdot \mathcal{X} \quad \forall t \in [0, \inf[, \tag{10.15}$$

where t is the mission time. Dependencies will henceforward be notated as right super indexes. Now consider a pixel that is a view of the host vehicle. It can be any of the pixels in Fig. 10.7 that capture a part of the vehicle. Applying the inverse projection (detailed in chapter 5) to eq. (10.15) would make it possible to retrieve the 3D point $\mathcal{X}$ that is viewed by pixel $\mathtt{x}$. In the case of these pixels that view the vehicle's body, the inverse projection would lead to an erroneous mapping because it assumes that the pixel is viewing part of the scenario outside the vehicle, when in fact it views part of the vehicle. In other words, $\mathcal{X}$ is in fact a different point in 3D contained by the vehicle body. This new 3D point is notated as $\hat{\mathcal{X}}$. For this point the expression of the inverse projection obtained from eq. (10.15) is not valid. However, if the point is defined in the vehicle reference system ($\hat{\mathcal{X}}$) one can state that:

$$\mathtt{x} = \mathbf{K} \cdot ^{\mathbf{C}}\mathbf{T}_{\mathbf{V}} \cdot \hat{\mathcal{X}}, \tag{10.16}$$

which means, since both $\mathbf{K}$ and $^{\mathbf{C}}\mathbf{T}_{\mathbf{V}}$ are constant over time, that the pixel will always view the same portion of the vehicle. This is an important conclusion because it proves that the pixels that view the vehicle are always the same. And this is true for all the images of the given camera. In other words, since the position of the invalid pixels is always the same in the image, a mask can be created offline to discard them in every image. The remaining pixels, i.e., those that are not discarded, can then be used in eq. (10.15), since the equation is valid for all other cases.

In order to discard the pixels that view the host vehicle, we use a manual calibration where a user defines a polygon in the image that contains the pixels that view the scene. As discussed before,

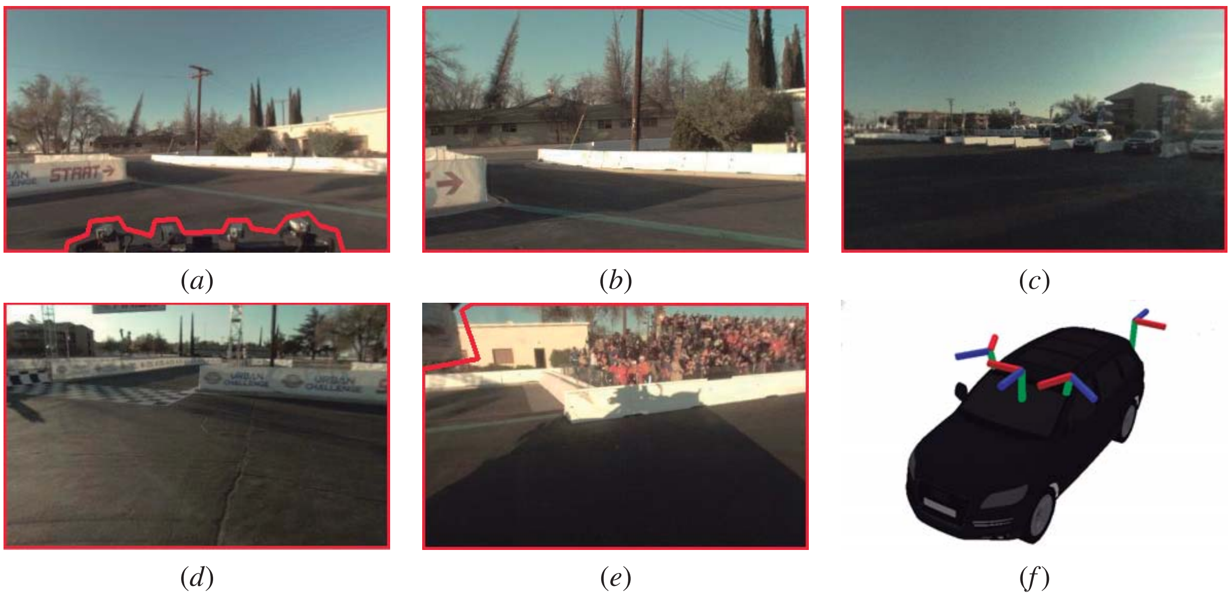


Figure 10.8: Image bounding polygons for the camera of the *Talos* viewing location *C*, sequence 1: (*a*) forward camera; (*b*) 6 millimeter lens forward camera; (*c*) rear camera; (*d*) forward left camera; (*e*) forward right camera; (*f*) the coordinate frames of the all the cameras.

this procedure is executed offline, and needs only to be made once for each camera. We refer to this polygon as the image bounding polygon.

Figure 10.8 shows the image bounding polygons for all the cameras onboard the *Talos* vehicle. Two of the cameras view portions of the vehicle. It is the case of cameras forward center (Fig. 10.8 (*a*)) and forward right (Fig. 10.8 (*e*)). In these cases the image bounding polygons exclude the regions where the cameras view portions of the vehicle. Note that this can be used as a general solution, meaning it can be employed also in the cases where the camera does not view the vehicle body. In these cases the image bounding polygon corresponds to the image canvas, as shown in Figs. 10.8 (*b*), (*c*) and (*d*). Finally, Fig. 10.8 (*e*) shows the local coordinate frames of all the cameras. Note that the forward center and forward center 6 millimeter cameras are very close and their coordinate frames overlap, which is why in the figure only four coordinate frames are shown for five cameras.

10.3.3 Projection Polygon

It was already discussed that approaches detailed in sections 10.2 and 10.3.1 are in fact simplified. Section 10.3.2 introduced this topic commenting on the fact that it is not possible to assume that the entire image views a given primitive. One of the situations that may occur, is that a camera views part of the vehicle body. This problem was addressed in section 10.3.2. The solution is to define, for each camera and in an offline procedure, the image bounding polygon. In the region of the image that is inside the image bounding polygon, we are sure that no vehicle body is viewed. Hence, in that region, only the scenario around the vehicle is viewed. The question that is addressed in this section is the following: even if only the region inside the image bounding polygon is considered, is it possible to ensure that all the pixels view a single geometric polygonal primitive. In other words, in the simplified examples of Figs. 10.2 and 10.6, only the support plane of primitives is employed. It is used directly to define the projection plane. However, the contours of the primitive where not taken into account, as they should be. This section shows how it is possible to address this problem. To take into account a primitive bounding polygon in order to compute the region of the image that actually contains visual information of that primitive. Note that an image does not necessarily always contains information about a given primitive. For example, consider that a geometric polygonal primitive was detected on the right side of the vehicle. If the image provided by the camera facing the left side of the vehicle is to be used, then it is sure not to contain any visual information of that primitive.

Let $\mathtt{r}$ notate the list of vertices of the image bounding polygon. These vertices are defined in image space. To compute the vertices in the world coordinate frame we use eq. (10.15), that is:

$$\mathtt{r} = \mathbf{K} \cdot {}^{\mathbf{C}}\,\mathbf{T}_{\mathbf{V}} \cdot {}^{\mathbf{V}}\mathbf{T}^{t}_{\mathbf{W}} \cdot \mathcal{R} \quad \forall t \in [0, \inf[, \tag{10.17}$$

where $\mathcal{R}$ is the list of vertices in the image bounding polygon, viewed from the world coordinate frame. In order to compute the 3D world coordinates of those vertices, we use the inverse projection, that is, eq. (10.17) becomes:

$$\mathcal{R} = \left[\mathbf{K} \cdot {}^{\mathbf{C}}\,\mathbf{T}_{\mathbf{V}} \cdot {}^{\mathbf{V}}\mathbf{T}^{t}_{\mathbf{W}}\right]^{-1} \cdot \mathtt{r} \quad \forall t \in [0, \inf[, \tag{10.18}$$

which is a system of equations without solution. Since a detailed discussion on this topic is presented in chapter 5, we will not focus on the complete explanation here. It is suffice to say that the system is complete when a plane is defined so that it is assumed that the 3D point ($\mathcal{R}$ in this case) lies on that plane. In other words, the following system of equations must be solved:

$$\begin{cases} \mathcal{R} = \left[\mathbf{K} \cdot {}^{\mathbf{C}}\,\mathbf{T}_{\mathbf{V}} \cdot {}^{\mathbf{V}}\mathbf{T}^{t}_{\mathbf{W}}\right]^{-1} \cdot \mathtt{r} \quad \forall t \in [0, \inf[\\ a \cdot \mathcal{R}_x + b \cdot \mathcal{R}_y + c \cdot \mathcal{R}_z + d = 0 \end{cases}, \tag{10.19}$$

where $\mathcal{R}_x$ notates the x component of 3D point $\mathcal{R}$, and a, b, c and d, the Hessian form coefficients of the projection plane. The objective is to introduce all the factors that affect the projection of the image of a camera. From the previous discussion and the observation of eq. (10.19), we can say that the factors that affect a projection are:

- The geometric polygonal primitive to which texture should be mapped, since it defines the coefficients a, b, c and d of the second equation.

- The camera, for three reasons: first, the intrinsic parameters (**K**) are unique to each camera, second, the image bounding polygon ($\mathtt{r}$) is also configured independently for each camera, and, third, because transform ${}^{\mathbf{C}}\mathbf{T}_{\mathbf{V}}$ also changes according to the cameras position.

- the mission time t, since it affects the ${}^{\mathbf{V}}\mathbf{T}_{\mathbf{W}}^{t}$ transform.

Suppose a given situation where a vehicle is travelling a scenario, and producing a geometric representation, that is, a list of geometric polygonal primitives. We will consider k, the index of the geometric polygonal primitives, $k \in [0, Nk[$, where Nk is the number of primitives. As the vehicle travels, images are acquired from different cameras. We will use index l to notate the camera index, $l \in [0, Nl[$, where Nl is the number of cameras onboard the vehicle. Also, mission time t is taken into account. Using the super index notation to refer the indexes of both the cameras and the primitives, eq. (10.19) becomes:

$$\begin{cases} \mathcal{R} = \left[\mathbf{K}^{l} \cdot {}^{\mathbf{C}}\mathbf{T}_{\mathbf{V}}^{l} \cdot {}^{\mathbf{V}}\mathbf{T}_{\mathbf{W}}^{t}\right]^{-1} \cdot \mathtt{r}^{l} \quad \forall t \in [0, \inf[\\ a^{k} \cdot \mathcal{R}_{x} + b^{k} \cdot \mathcal{R}_{y} + c^{k} \cdot \mathcal{R}_{z} + d^{k} = 0 \end{cases} . \tag{10.20}$$

Equation (10.20) shows that a given projection depends on three factors: the geometric primitive index k, the camera index l, and mission time t. We shall disregard the dependency of mission time for now, since it will be addressed in detail in chapter 11. At present, lets assume that we are considering only a moment in time, which we will call the time of projection t_p, hence, $t = t_p$.

For a given time of projection t_p, a camera l, and a primitive k, using eq. (10.20), it is possible to compute the projection of the image bounding polygon, represented in the world coordinate frame. We refer to this as $\mathcal{R}^{\{k,l,t=t_p\}}$, i.e., the list of 3D vertices that correspond to the tuple $\{k, l, t = t_p\}$.

Figure 10.9 (*a*) shows a birds eye view of a scene. In this case only a single primitive is detected, which corresponds to the ground plane. The bounding polygon of that primitive is shown in blue in Figs. 10.9 (*a*) and (*b*). The set of possible values of k is composed of a single index, e.g., $k \in \{0\}$. All five cameras are used in the projection. Hence, $l \in \{0, 1, 2, 3, 4\}$.

As a consequence, there will be five different image bounding polygons and the same number of the projections of these to the world coordinate frame. Figure 10.9 (*b*) shows all these polygons: $\mathcal{R}^{\{k=0,l=0,t=t_p\}}$, $\mathcal{R}^{\{k=0,l=1,t=t_p\}}$, $\mathcal{R}^{\{k=0,l=2,t=t_p\}}$, $\mathcal{R}^{\{k=0,l=3,t=t_p\}}$ and $\mathcal{R}^{\{k=0,l=4,t=t_p\}}$. Note these are the projections in world coordinate frame of the polygons defined in image coordinate frame. In other words, the polygons shown in Fig. 10.9 (*b*) correspond to the solving of eq. (10.20) for each of the image bounding polygons $\mathtt{r}^{l}$ shown in Figs. 10.8 (*a*), (*b*), (*c*), (*d*) and (*f*). For example, the image bounding polygon $\mathcal{R}^{\{k=0,l=0,t=t_p\}}$ ($l = 0$ is the index for forward center camera) shown in Fig. 10.8 (*a*) is easily identifiable in Fig. 10.9 (*b*) because of the contours on the bottom part of the image are also present in the 3D view of Fig. 10.9 (*b*).

Previous paragraphs have described how the image bounding polygons are projected to the world coordinate frame. The main objective, is to compute which is the subset of pixels in an image (given the set of all the pixels in the image) that are viewing the primitive. More specifically, the problem

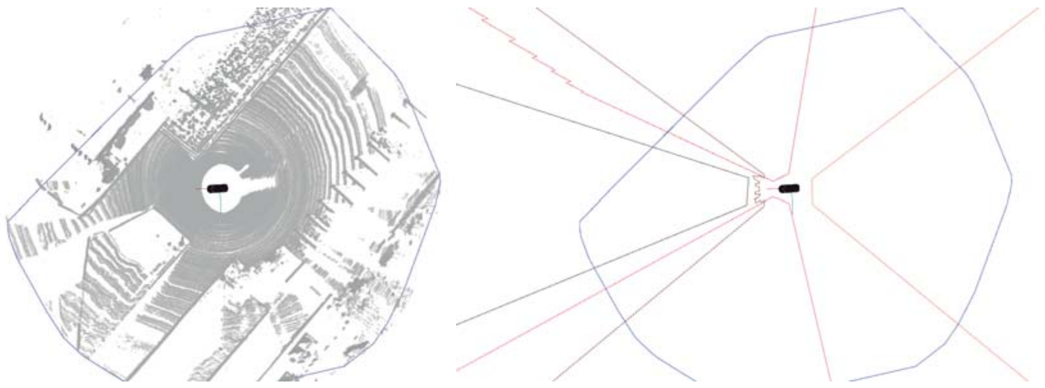


Figure 10.9: Projections of the image bounding polygons shown in Fig. 10.8 to the world coordinate frame, considering the ground plane primitive (shown in blue). Data from location *C*, sequence 1: (*a*) the scene viewed from a birds eye view; (*b*) the projections of the image bounding polygons.

is the following: for a given projection defined by the tuple $\{k = k_p, l = l_p, t = t_p\}$ where k_p, l_p and t_p are the projection primitive, projection camera and projection time respectively, which is the region in the image that contains pixels that view the primitive. We propose to obtain this region in the image by computing what we call projection polygon. This polygon, when viewed in image space, will delimit the pixels that are a view of the primitive k_p.

Chapter 8 provided a detailed description of the generation and refinement of the geometric polygonal primitives. More specifically in section 8.3.1, the representation of the geometric polygonal primitives is discussed. In that section, it is said that geometric polygonal primitives required a description of the support plane in Hessian form as well as a list of vertices that define the primitives bounding polygon. The coefficients of the primitive's support plane are directly used in eq. (10.20) so no further discussion is required on this matter. Regarding the representation of the primitive's bounding polygon, section 8.3.5 describes it in detail. In particular, the explanation from eqs. (8.2) to (8.7) shows that the primitives bounding polygon is defined over a local primitive coordinate system. Since this local primitive primitive coordinate system is defined so that its X and Y axes are contained by the primitive's support plane, when a vertex of the bounding polygon is viewed under the local coordinate frame its Z coordinate will always be equal to zero. In this way, because the Z coordinate is discarded, the vertices of the primitives bounding polygon are defined in $\mathbb{R}^2$, and the list of these vertices is defined as $\mathtt{P}$, or, in this case, $\mathtt{P}^k$.

Let **P** represent the local primitive's coordinate frame, and ${}^{\mathbf{W}}\mathbf{T}_{\mathbf{P}}$ the transform from the world coordinate frame to the primitive's local coordinate frame. Since this transform depends on the primitive in question, following the previous conventions we will notate it ${}^{\mathbf{W}}\mathbf{T}^{k}_{\mathbf{P}}$. In the notation introduced before it was defined that handwritten characters are used for variables defined in $\mathbb{R}^3$, while machine type characters are used for variables in $\mathbb{R}^2$. As an example, the image bounding polygon is represented as $\mathtt{r}$, when viewed from the camera reference system in the image space, and $\mathcal{R}$ if we refer to some projection of that point to the 3D world coordinate frame. Since non capital machine type characters are used for representing points (or lists of points) in image space, we use capital machine type characters to represent the point viewed in $\mathbb{R}^2$ space in the primitive's local coordinate frame. Therefore, the list of vertices of the image bounding polygon that is viewed from the primitives local coordinate frame is referred to as $\mathtt{R}$. This is the reason why the symbol for notating the list of vertices in the primitives support point is referred to as $\mathtt{P}^k$.

The projection polygon (represented by $\mathtt{H}$) is computed as the intersection ($\cap$) between the primitive's bounding polygon $\mathtt{P}^k$ and the projection of the images bounding polygon $\mathtt{R}^l$, viewed in the primitives local coordinate frame:

$$\mathtt{H}^{\{k,l\}} = \mathtt{P}^k \cap \mathtt{R}^l, \tag{10.21}$$

where $\mathtt{R}^l$ is obtained by:

$$\mathtt{R}^l = \mathbf{f}\Big(\left[{}^{\mathbf{W}}\mathbf{T}_{\mathbf{P}}\right]^{-1} \cdot \mathcal{R}^l\Big), \tag{10.22}$$

and $\mathbf{f}$ is a function that discards the Z coordinate of a 3D point, i.e., $\mathbf{f}: \mathbb{R}^3 \rightarrow \mathbb{R}^2$. Since the projection polygon $\mathtt{H}^{\{k,l\}}$ is defined in the local coordinate frame of primitive k and what is sought is the polygon in image space, the expressions in eqs. (10.17) and (10.22) are combined and result in:

$$\mathtt{h}^{\{k,l\}} = \mathbf{K} \cdot {}^{\mathbf{C}}\,\mathbf{T}_{\mathbf{V}} \cdot {}^{\mathbf{V}}\mathbf{T}^{t}_{\mathbf{W}} \cdot {}^{\mathbf{W}}\mathbf{T}_{\mathbf{P}} \cdot \mathbf{g}(\mathtt{H}^{\{k,l\}}), \tag{10.23}$$

where $\mathbf{g}$ is a function that adds a zero value coordinate to a 2D point, i.e., $\mathbf{g}: \mathbb{R}^2 \rightarrow \mathbb{R}^3$. It is also possible to compute the 3D coordinates viewed in world coordinate frame of $\mathtt{h}^{\{k,l\}}$, referred to as $\mathcal{H}^{\{k,l\}}$, using the following expression:

$$\mathcal{H}^{\{k,l\}} = {}^{\mathbf{W}}\mathbf{T}_{\mathbf{P}} \cdot \mathbf{g}(\mathtt{H}^{\{k,l\}}), \tag{10.24}$$

Note also, that from eq. (10.21) the projection polygon $\mathtt{H}^{\{k,l\}}$ is obtained from the intersection of the image bounding polygon and the primitive's bounding polygon. In the case that the image does not view portions of the vehicle body, the image bounding polygon corresponds to the image canvas. When this occurs, and if the image canvas contains the entire primitive's bounding polygon, the projection polygon is in fact equal to the primitive's bounding polygon. In other words, if $\mathtt{P}^k \subseteq \mathtt{R}^l$, then the following applies:

$$\mathtt{H}^{\{k,l\}} = \mathtt{P}^k \cap \mathtt{R}^l = \mathtt{P}^k, \quad when \quad \mathtt{P}^k \subseteq \mathtt{R}^l. \tag{10.25}$$

Figure 10.10 (*a*) shows a scenario where the *Talos* vehicle has detected a single primitive (marked in blue) corresponding to a wall in front of it. The scene corresponds to location *D*, sequence 1. Only the forward center camera is used in this example. From eq. (10.20), the image bounding polygon $\mathtt{r}$, shown in Fig. 10.8 (*a*), is projected to the world coordinate frame using the primitive's support plane. $\mathcal{R}$ is shown in Fig. 10.10 (*b*) in dark color. From eq. (10.21), the projection polygon $\mathtt{H}^{\{k,l\}}$ results in the intersection of the two polygons shown in Fig. 10.10 (*b*) (actually, the polygons shown in this Fig. are $\mathcal{P}^k$ and $\mathcal{R}^l$, but the intersection is done in 2D with polygons $\mathtt{P}^k$ and $\mathtt{R}^l$). In Fig. 10.10 (*c*) the projection polygon $\mathcal{H}^{\{k,l\}}$ is shown in dark color. Finally, Fig. 10.10 (*d*) shows an image with the image bounding polygon $\mathtt{r}^l$ (in red) and the projection polygon $\mathtt{h}^{\{k,l\}}$ (in blue).

The presented algorithm is able to define which of the pixels in an image are a view of the selected primitive. It is also possible to know whether an image views the primitive or not. When the intersection computed in eq. (10.21) is a null set, then it means that none of the image pixels view the primitive.

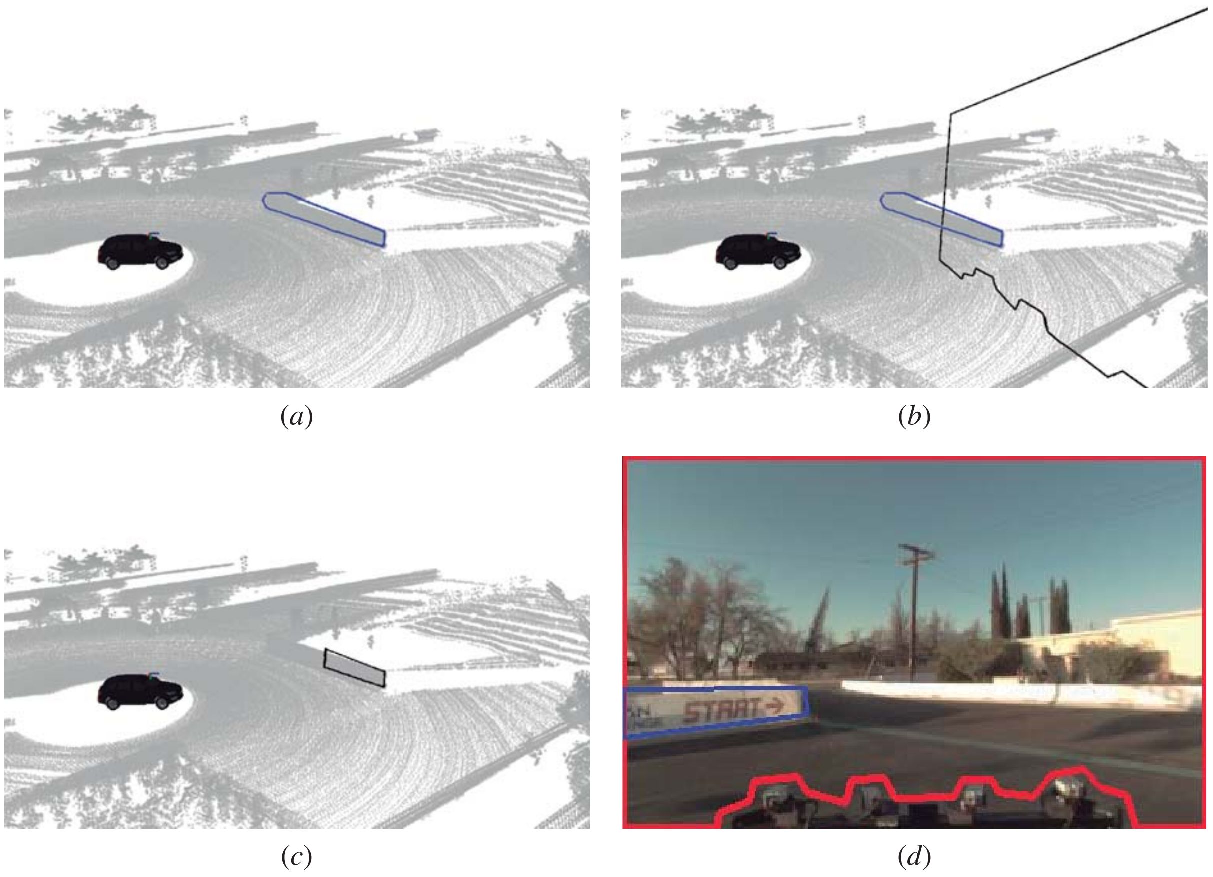


Figure 10.10: The algorithm for computing the projection polygon in image space: (*a*) location *D*, sequence 1, a single primitive detected (blue); (*b*) the image bounding polygon (dark color); (*c*) the projection polygon (dark color); (*d*) the image with the image bounding polygon (red) and the projection polygon (blue) .

This section has described how the projection polygon is computed. The solution is general, which means it is valid for any camera, mounted on any vehicle. All that is required is a description of the camera, of its position, and a few other variables in order to compute the projection polygon. The algorithm is also capable of identifying whether a camera views, partially or not, a given primitive. These tools will be used in the following chapters when the mechanism that executes texture mapping onto the polygonal primitives is presented.

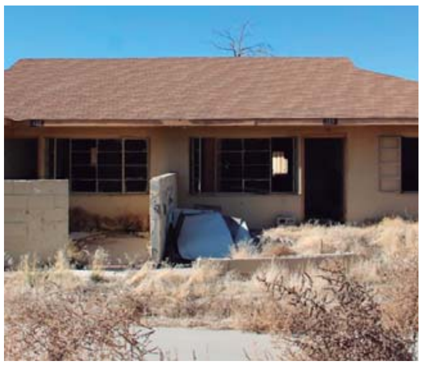
10.3.4 Mapping Images to Geometric Polygonal Primitives

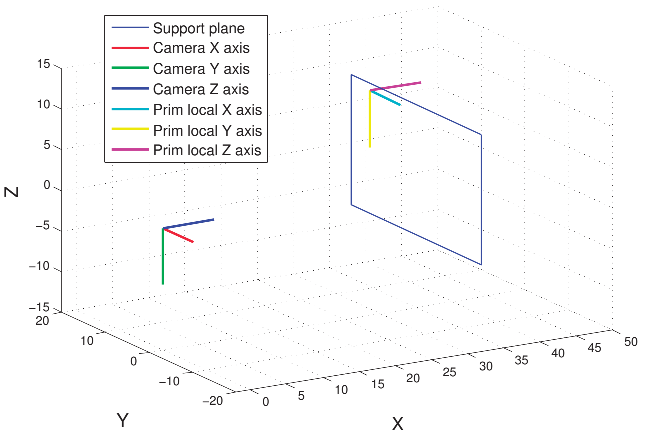
The previous sections have described several algorithms or parts of algorithms that are used in the mapping of images to polygonal primitives. The problems that those algorithms have solved are:

- How color is transferred from an image that is a discrete space to 3D geometry which is a continuous space, using interpolation over triangles. Texture mapping and interpolation techniques, section 10.2;
- How the problem of affine texture mapping can be solved if a DDT triangulation is performed over the image. DDT triangulations, section 10.3.1;
- How to handle the fact that images from onboard cameras may be partially occluded by the vehicle body. Image bounding polygons, section 10.3.2;
- How to compute which region of the image contains visual information from a primitive. Projection polygons, section 10.3.3;

This section focuses on how to merge all these algorithms in order to map images onto geometric polygonal primitives. To demonstrate the entire process, a scene with a single primitive and a single camera is used. The scene shown in Fig. 10.11 (*a*) is used. In this image, a house is shown. A visual analysis immediately concludes that there are several planes in the 3D structure that are captured in the image. For now, one plane will be used: the wall of the house, where the doors and windows are located. Figure 10.11 (*b*) shows a view of the 3D scene, including the camera's position and the location of the primitive's support plane.

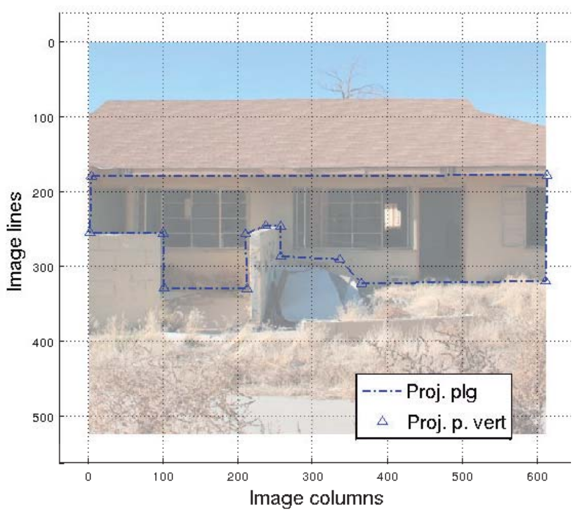
The primitive contains information not only about the support plane, but also regarding its bounding polygon ($\mathtt{H}^{\{k,l\}}$). The first step is to compute the projection polygon (eq. (10.21)) and project it to the image. Figure 10.12 (*a*) shows the projection polygon in image space ($\mathtt{h}^{\{k,l\}}$), while Fig. 10.12 (*b*) shows the same polygon represented in 3D ($\mathcal{H}^{\{k,l\}}$). Note that $\mathtt{h}^{\{k,l\}}$ is not convex. It is not possible to assume that $\mathtt{H}^{\{k,l\}}$ is convex, since $\mathtt{h}^{\{k,l\}}$ is obtained from the intersection of two polygons, $\mathtt{P}^{\{k\}}$ and $\mathtt{R}^{\{l\}}$. In order to ensure the convexity of $\mathtt{h}^{\{k,l\}}$, one would have to ensure the convexity of both $\mathtt{P}^{\{k\}}$ and $\mathtt{R}^{\{l\}}$. While it is possible to ensure the convexity of $\mathtt{P}^{\{k\}}$, if a convex hull operator is used to compute the primitive's bounding polygon (see section 8.3.5), in the case of

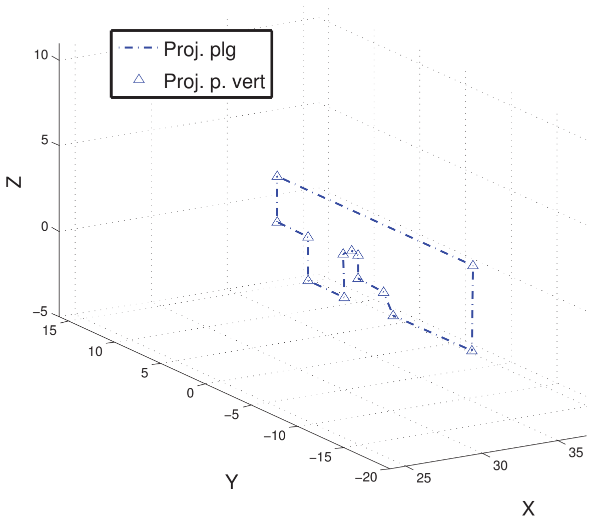




(*a*) (*b*)

Figure 10.11: Demonstration scene for mapping images to geometric polygonal primitives: (*a*) a view of the image of the scene; (*b*) the 3D scene with the location of the camera and the support plane of the primitive that corresponds to the wall of the house, where the windows and the door are.





(*a*) (*b*)

Figure 10.12: The projection polygon: (*a*) in the image space; (*b*) in 3D space.

$\mathrm{R}^{\{l\}}$ this is not possible, since the image bounding polygon often needs to have a non convex shape to delimit the portions of the image that view the vehicle body (see Fig. 10.8 (*a*), for an example). In conclusion, there is no assurance that the projection polygon is convex. This seems like a minor detail at the moment, but will be important further ahead, since a special mechanism must be proposed for handling the non convexity of the projection polygon.

The second step is to compute the line segments that describe regions of high gradient in the image. As described in section 10.3.1, this is vital to ensure accurate texture mapping. However, in this case, Hough lines are searched only in the area of the image delimited by the projection polygon. This is a more efficient operation since only a portion of the image is processed, instead of the entire image. Figure 10.13 (*a*) shows the line segments detected in the region delimited by the projection polygon.

The third step is to compute a Delaunay triangulated mesh. In section 10.3.1, it was suggested that the starting and end points of the detected line segments are used as input vertices, and the line segments are used as input constraints to a constrained Delaunay triangulation (see eq. (10.8)). Since it is important to keep the boundaries defined by the projection polygon, this information must also be taken into account in the constrained Delaunay function. Hence, we propose to extend the set of input vertices and constraints to the constrained Delaunay triangulation using the vertices and line segments of the projection polygon. Let $\mathtt{h}_i^{\{k,l\}}$ be the ith vertex of the projection polygon $\mathtt{h}^{\{k,l\}}$. Considering there are M line segments detected using Hough transform and N vertices in the projection polygon, the set of vertices (**SV**) to input to the triangulation is:

$$\mathbf{SV} = \{\mathtt{s}_0^l, \mathtt{e}_0^l, \mathtt{s}_1^l, \mathtt{e}_1^l ..., \mathtt{s}_{M-1}^l, \mathtt{e}_{M-1}^l, \mathtt{h}_0^{\{k,l\}}, \mathtt{h}_1^{\{k,l\}}, \mathtt{h}_{N-1}^{\{k,l\}}\}, \tag{10.26}$$

and the set of constraints (**SC**) is:

$$\mathbf{SC} = \left\{\overline{\mathtt{s}_0^l \mathtt{e}_0^l}, \overline{\mathtt{s}_1^l \mathtt{e}_1^l}, ..., \overline{\mathtt{s}_{M-1}^l \mathtt{e}_{M-1}^l}, \overline{\mathtt{h}_0^{\{k,l\}}, \mathtt{h}_1^{\{k,l\}}}, ..., \overline{\mathtt{h}_{N-2}^{\{k,l\}}, \mathtt{h}_{N-1}^{\{k,l\}}}, \overline{\mathtt{h}_{N-1}^{\{k,l\}}, \mathtt{h}_0^{\{k,l\}}}\right\}, \tag{10.27}$$

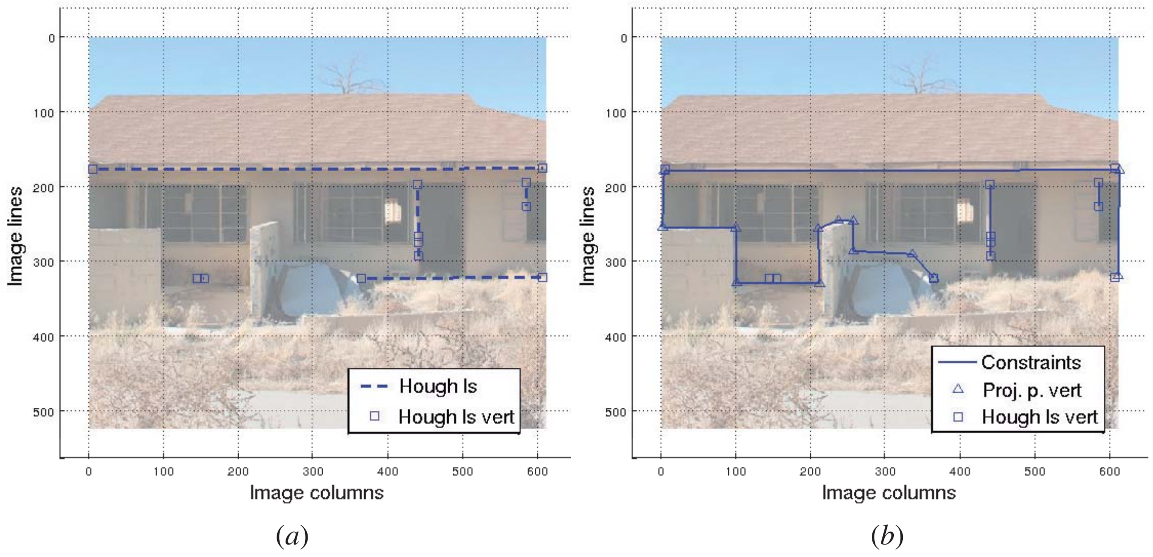
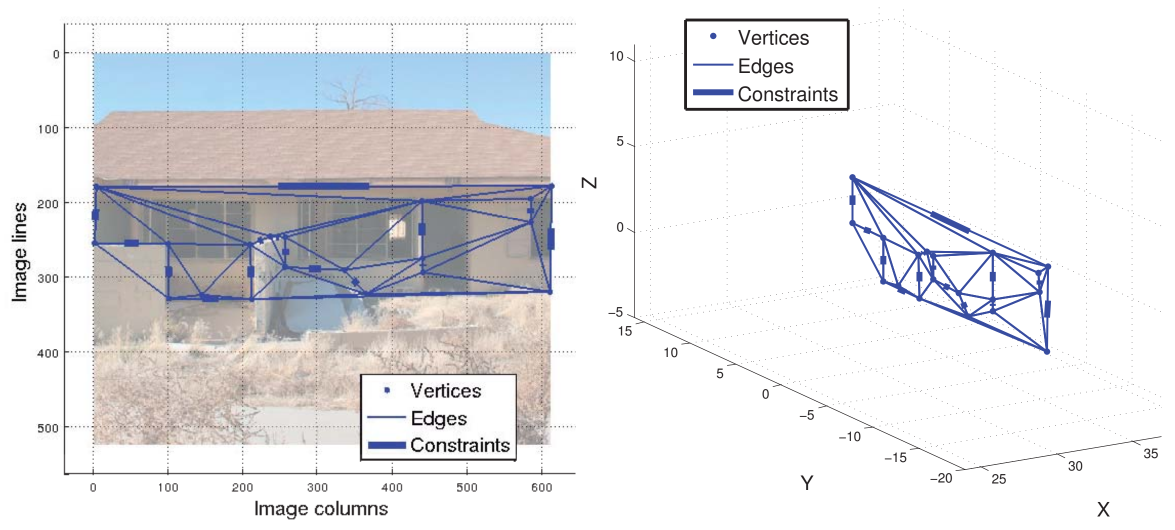


Figure 10.13: Setting up the constrained Delaunay triangulation: (*a*) line segments detected using Hough lines transform; (*b*) vertices and constraints used as input to the constrained Delaunay triangulation.



(*a*) (*b*)

Figure 10.14: Triangular mesh: (*a*) in image space; (*b*) projected to 3D space.

which results in a constrained Delaunay operation (**cDelaunay**) as follows:

$$\texttt{t} = \mathbf{cDelaunay}(\mathbf{SV}, \mathbf{SC}). \tag{10.28}$$

Figure 10.13 (*b*) shows all the vertices and line segments used as input to the constrained Delaunay triangulation. It is possible to see that some constraints are derived from the detected line segments (shown in Fig. 10.13 (*a*)), while others come from the projection polygon (shown in Fig. 10.12 (*a*)).

Figure 10.14 (*a*) shows the triangular mesh obtained using a constrained Delaunay triangulation. The mesh is shown in the image space, i.e., referred as $\texttt{t}$. As can be seen, some of the edges in the mesh are constrained, while others, which are generated automatically, are left unconstrained. Figure 10.14 (*b*) shows the projection of the triangular mesh to 3D world space ($\mathcal{T}$). Note that the mechanism of mapping a mesh $\texttt{t}$ in image space into a mesh $\mathcal{T}$ in 3D space is done here resorting to the inverse projection expression detailed in eq. (10.20). However, for reasons that will become clearer in the next sections, the algorithm that is actually employed is more complex. Details on this will be provided in the next sections. For now, it is sufficient to think of $\mathcal{T}$ as the mesh obtained by inverse projecting $\texttt{t}$.

The fourth step of the process is to iterate through all triangles in the mesh and map the texture contained by them to the 3D world. The mapping of a triangle's texture onto 3D was discussed in section 10.2, so there is no need to go in detail. Figure 10.15 (*a*) shows the triangular mesh with a texture overlay. Figure 10.15 (*b*) only shows the texture.

This section has presented the algorithm that was devised for mapping images onto geometric polygonal primitives. The process consists of four steps:

- Compute the projection polygon;
- Compute the line segments using Hough in the region delimited by the projection polygon;
- Run a constrained Delaunay triangulation using both the line segments and the projection polygon as input vertices and constraints;
- Map each triangle in image space to 3D and apply texture.

10.3.5 Handling the Convexity of Delaunay Meshes

Section 10.3.4 has described the process for mapping texture onto polygonal primitives. Results from this process are shown in Fig. 10.15. Although the results seem to execute an accurate mapping, a more careful observation notes that the shape of the textured area in Fig. 10.15 does not correspond to the shape of the projection polygon (which is shown in Fig. 10.12). For example, on the left bottom part of the textured region, a brick wall is textured. This brick wall is actually in front of the wall with the doors and windows. Fig. 10.16 (*a*) shows zoomed view of the triangulated mesh. Fig. 10.16 (*b*) shows the projection polygon overlayed onto the triangulated mesh. The brick wall that was referred is the one contained by triangle 11-12-13 (see indexes on Fig. 10.16 (*a*)). Although it is out of the projection polygon, it appears on the triangulated mesh. The reason is that Delaunay triangulations triangulate the entire convex hull space of the points given as input vertices. This is an intrinsic characteristic of the Delaunay method, it cannot be changed. The option is to detect which

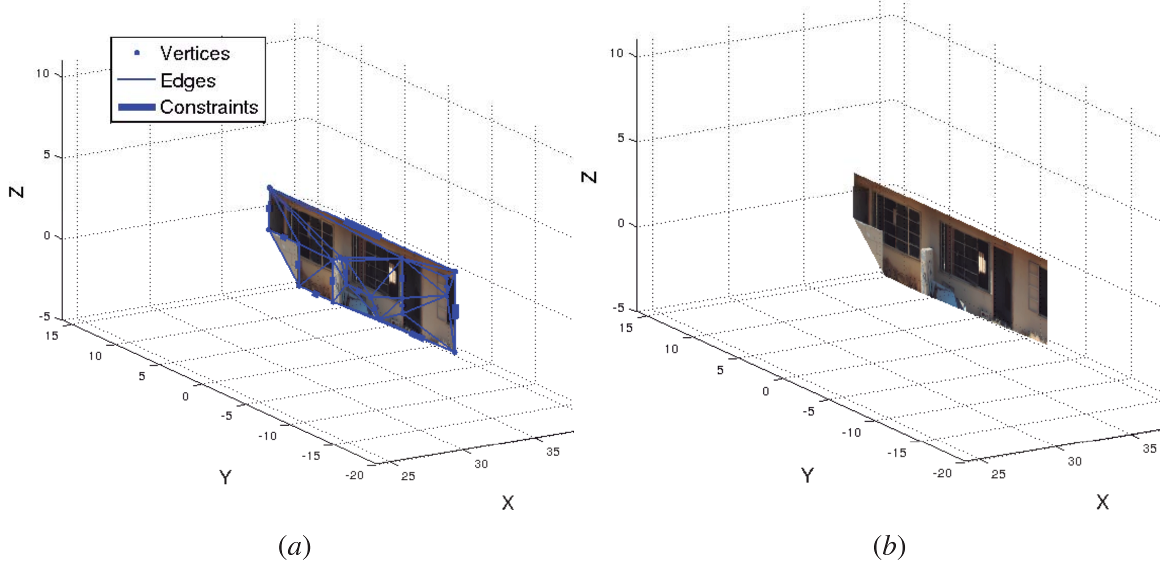
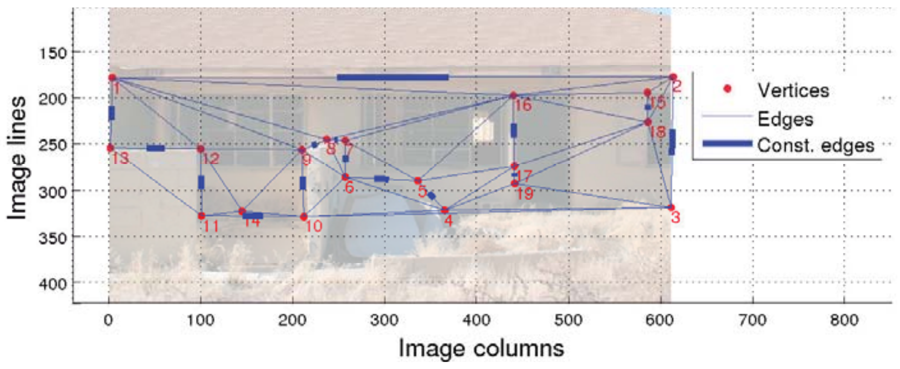


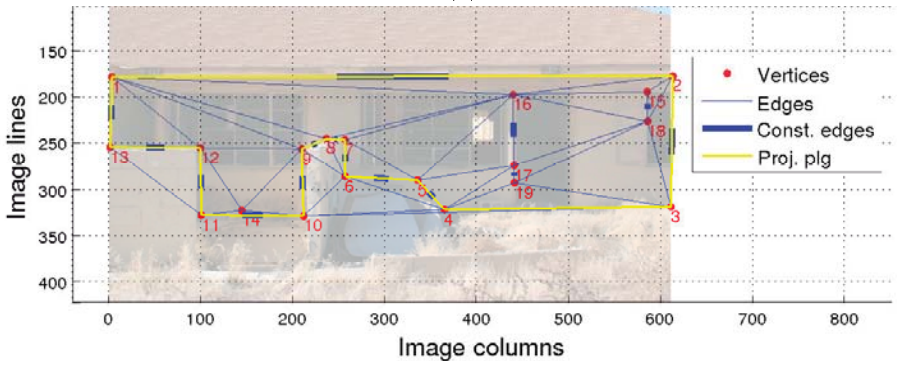
Figure 10.15: Triangular mesh in 3D space: (*a*) triangles and texture; (*b*) texture only.

triangles are inside the projection polygon and which are outside of it. This is done using a recursive algorithm which will be presented next.

A somewhat complex notation will be used. For this reason it is described in the following lines. Let a given triangulated mesh containing several triangles, be denoted as $\mathtt{t}_{\{v0,v1,v2\}}$, where $v0, v1, v2$ are the indices of the vertices of the triangle. It can also be denoted as $\mathtt{t}_i$, where i is the index of the triangle. Similarly, an edge is denoted as $\mathtt{e}_{\{v1,v2\}}$. Each triangle has two flags associated to it: flag $is\_visited$ and flag is_inside. The first assesses if the triangle was already visited or not, and the second indicates if the triangle is outside or inside the given polygon, in this case the projection polygon. Function **set**$(\mathtt{t}_{\{v0,v1,v2\}}), flag, value)$ writes the value $value$ to the flag $flag$ of triangle $\mathtt{t}_{\{v0,v1,v2\}}$. Function **get**$(\mathtt{t}_{\{v0,v1,v2\}}, flag)$ reads the current value of the flag $flag$ of triangle $\mathtt{t}_{\{v0,v1,v2\}}$. Function **edge**$(\mathtt{t}_{\{v0,v1,v2\}}, j)$ recovers the jth edge ($j \in \{0, 1, 2\}$) of triangle $\mathtt{t}_{\{v0,v1,v2\}}$. Edges also have an additional flag $is\_boundary$ to indicate whether or not they are part of the projection polygon (it is read using **get**(**edge**$(\mathtt{t}_{\{v0,v1,v2\}}, j), is\_boundary)$). Finally, function **neighbor**$(\mathtt{t}_{\{v0,v1,v2\}}, j)$ returns the triangle that is the neighbor of triangle $\mathtt{t}_{\{v0,v1,v2\}}$ over edge j. Just to give some examples of the application of the functions, from Fig. 10.16 (a) and (b), one



(a)



(b)

Figure 10.16: A detailed view of the triangulated mesh: (a) the index of each vertex; (b) the projection polygon overlayed onto the mesh.

can see that, **edge**$(\mathtt{t}_{\{4,19,3\}}, 0) = \mathtt{e}_{\{4,19\}}$, that **get**$\big($**edge**$(\mathtt{t}_{\{17,16,18\}}, 0), is\_boundary\big) = false$, that **get**$\big(\mathtt{e}_{\{13,12\}}, is\_boundary\big) = true$, or that **neighbor**$(\mathtt{t}_{\{10,6,4\}}, 1) = \mathtt{t}_{\{6,5,4\}}$.

The algorithm starts in a triangle that contains one of the outer segments of the mesh. That way it is possible to know the is_inside state of this initial triangle. Then, the algorithm propagates from the starting triangle to others. A propagation from one triangle $\mathtt{t}_{parent}$ to another triangle $\mathtt{t}_{child}$ is done over an edge $\mathtt{e}_{propagation}$. If $\mathtt{e}_{propagation}$ is a boundary edge, then the is_inside state of $\mathtt{t}_{child}$ should be the opposite of the state of $\mathtt{t}_{parent}$. The procedure is propagated to the entire mesh in order to attribute a value to the flag is_inside of all the triangles in the mesh. It is considered that the $is\_boundary$ flag is already defined for all the edges in the mesh. Note that, to make this possible,

Algorithm 10.1 Compute which triangles are inside the projection polygon

```
Input: A triangulated mesh $\mathtt{t}$ with $M$ triangles
Output: The triangulated mesh $\mathtt{t}$, where get$(\mathtt{t}_i, is\_visited) = true$, $\forall i \in \{0, 1, ..., M-1\}$
  for $i = 0 \to M-1$ do                                    ▷ Initialize the $is\_visited$ flag of all triangles
      set$(\mathtt{t}_i), is\_visited, false)$
  end for
  Compute a random edge on the outer limit of the triangulation, $\mathtt{e}_{\{v_{out1}, v_{out2}\}}$
  Get initial triangle, that contains the outer limit edge, $\mathtt{t}_{\{v_{out1}, v_{out2}, v_k\}}$
  if get$(\mathtt{e}_{\{v_{out1}, v_{out2}\}}, is\_boundary) == true$ then          ▷ Set the $is\_inside$ flag for initial triangle
      set$(\mathtt{t}_{\{v_{out1}, v_{out2}, v_k\}}), is\_inside, true)$
  else
      set$(\mathtt{t}_{\{v_{out1}, v_{out2}, v_k\}}), is\_inside, false)$
  end if
  Set initial triangle as visited, set$(\mathtt{t}_{\{v_{out1}, v_{out2}, v_k\}}), is\_visited, true)$
  while Q$\neg$ empty do                                          ▷ Propagate through the mesh
      Set parent triangle as the first on the queue list, $\mathtt{t}_{parent} \leftarrow \mathbf{Q}_0$
      Set parent triangle flag visited, set$\big(\mathtt{t}_{parent}, is\_visited, true\big)$
      Remove parent triangle from queue list Q, i.e., remove first element from Q
      for $j = 0 \to 2$ do                                    ▷ Cycle the edges of the parent triangle
          Compute neighbor triangle over edge $j$, $\mathtt{t}_{child} \leftarrow$ neighbor$(\mathtt{t}_{parent}, j)$
          if $\mathtt{t}_{child}$ exists then         ▷ If edge $j$ is an outer edge there is no neighbor triangle
              if get$\big(\mathtt{t}_{child}, is\_visited\big) == false$ then
                  if get(edge$(\mathtt{t}_{parent}, j), is\_boundary) == false$ then
                      set$\Big(\mathtt{t}_{child}, is\_inside,$ get$\big(\mathtt{t}_{parent}, is\_inside\big)\Big)$
                  else
                      set$\Big(\mathtt{t}_{child}, is\_inside, \neg$get$\big(\mathtt{t}_{parent}, is\_inside\big)\Big)$
                  end if
                  Add child triangle to queue list, $\mathbf{Q} \leftarrow \{\mathbf{Q}, \mathtt{t}_{child}\}$
              end if
          end if
      end for
  end while
```

it is required that edges in the projection polygon are also edges in the triangulated mesh. This way for each segment of the boundary polygon, there is a corresponding edge in the mesh, which is set to have the $is\_boundary$ flag to $true$. This is the reason why segments of the projection polygon are set as constraints to the constrained Delaunay triangulation (see eq. (10.26)), to ensure they are edges in the mesh and can be mapped to have the corresponding $is\_boundary$ flag set accordingly. The procedure is presented in Algorithm 10.1.

Using Algorithm 10.1 it is possible to compute, for each triangle, whether it is inside our outside the projection polygon. Figure 10.17 shows the computed state of each triangle for the example shown in Figs. 10.15.

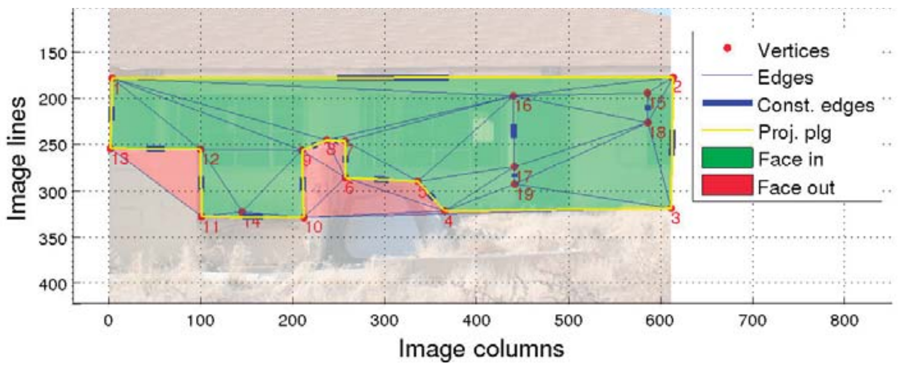
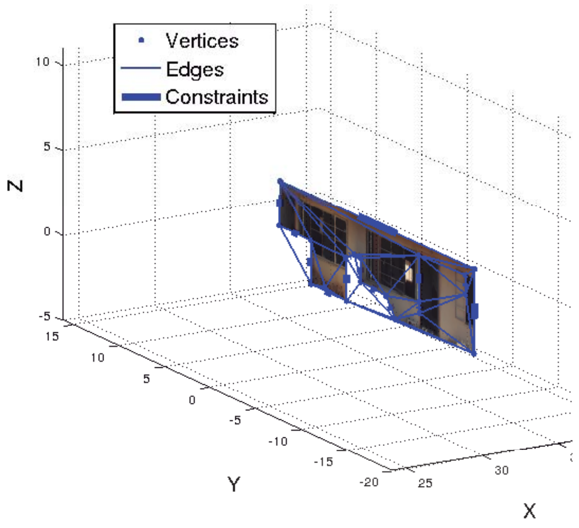
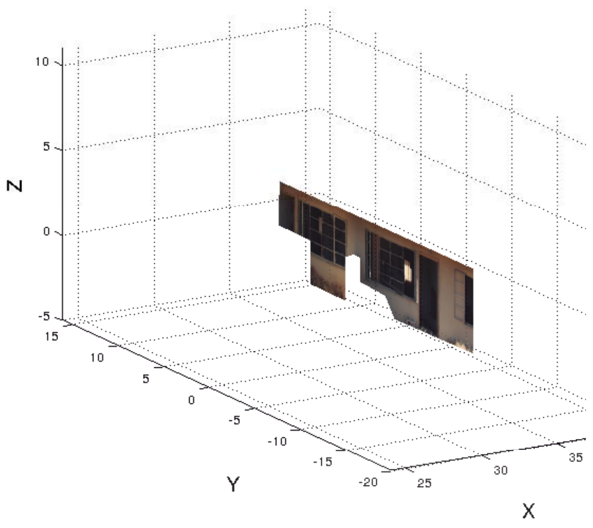


Figure 10.17: The inside or outside state of each triangle (or face) in the mesh.





(a) (b)

Figure 10.18: Texture mapping only triangles inside the projection polygon: (a) triangles and texture; (b) texture only. Figure 10.15 shows results when all triangles are mapped.

Texture mapping triangles is a procedure that iterates all triangles and maps each of them. If an additional test is added to this procedure, making so that only triangles that are inside the projection polygon are texture mapped, then it is possible to handle the convexity of Delaunay triangulated meshes. Results are shown in Fig. 10.18. By comparing these results with the primitive's texture mapping of all triangles (shown in Fig. 10.15), one can see that the problems with inaccurate mapping of areas not included in the projection polygon are solved with this technique.

10.4 Results

Previous sections have explained in detail the procedure that was devised to map texture from images to geometric polygonal primitives. This section will present some results regarding this methodology. It is quite difficult to devise a way of qualitatively comparing the results of texture mapping 3D models. Partial results from previous sections, show the proposed approach is capable of producing view pleasant textured polygonal primitives. But of course that other texture mapping techniques should work with the same accuracy. The question here is not whether or not the proposed approach is capable of producing better textured 3D models of the environment, but rather that it is capable of doing it much faster. To this factor contrives especially the fact that the generation of the 3D models themselves is much faster using the proposed geometric polygonal primitives, as shown in chapter 8. The other key factor is that the proposed approach is devised in a way so that it is capable of refining the texture when new visual information is received. This item will be detailed in chapter 11, but it is important to refer it now. In conclusion, we do not claim that the proposed approach for texture mapping polygonal primitives is intrinsically better than standard texture mapping techniques, but we do emphasize that it works just as good as any other techniques. More importantly, we claim that the proposed texture mapping approach can handle the texture mapping of geometric primitives and, as will be shown later, will be capable of improving texture with new visual data. To the best of our knowledge, this last capability is not reported in the literature. Under these considerations this section will present only qualitative results regarding the texture mapping of polygonal primitives, focusing more on the flexibility of the proposed approach. All it is required to texture map a given primitive is a description of the primitive and a description of the camera / image that provides the texture. Everything else is computed automatically. Multiple projections generated from several cameras / images can be computed over a great number of primitives. This way, it is possible to generate complex textured models. In the following lines, several examples are given that show the flexibility of the proposed approach.

10.4.1 Two Cameras Overlapping Projections

Figure 10.19 shows an example at location C, sequence 1, where a wall in front of the vehicle generates a primitive. The primitive is shown in blue on Fig. 10.19 (a), together with the scene around

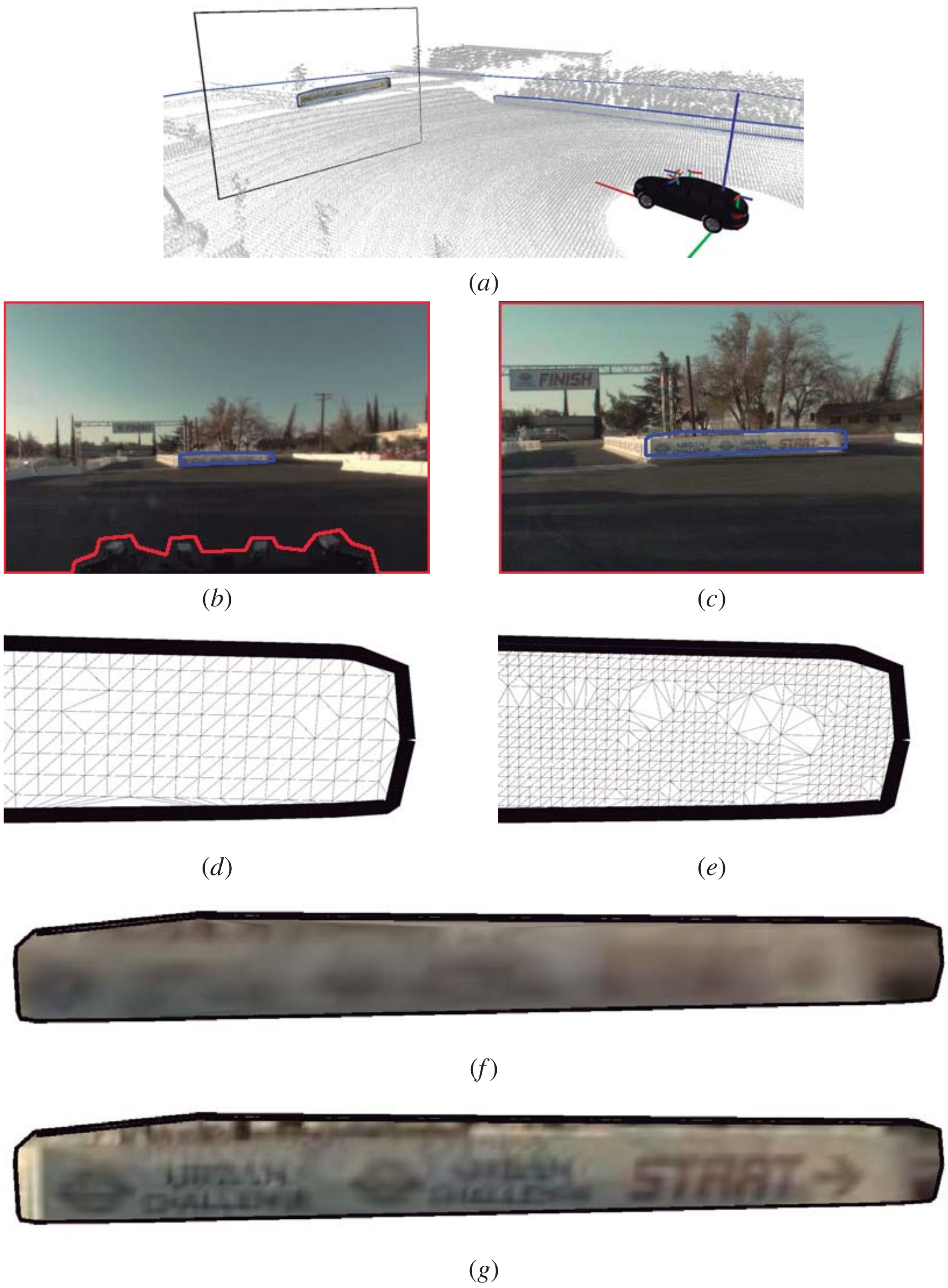
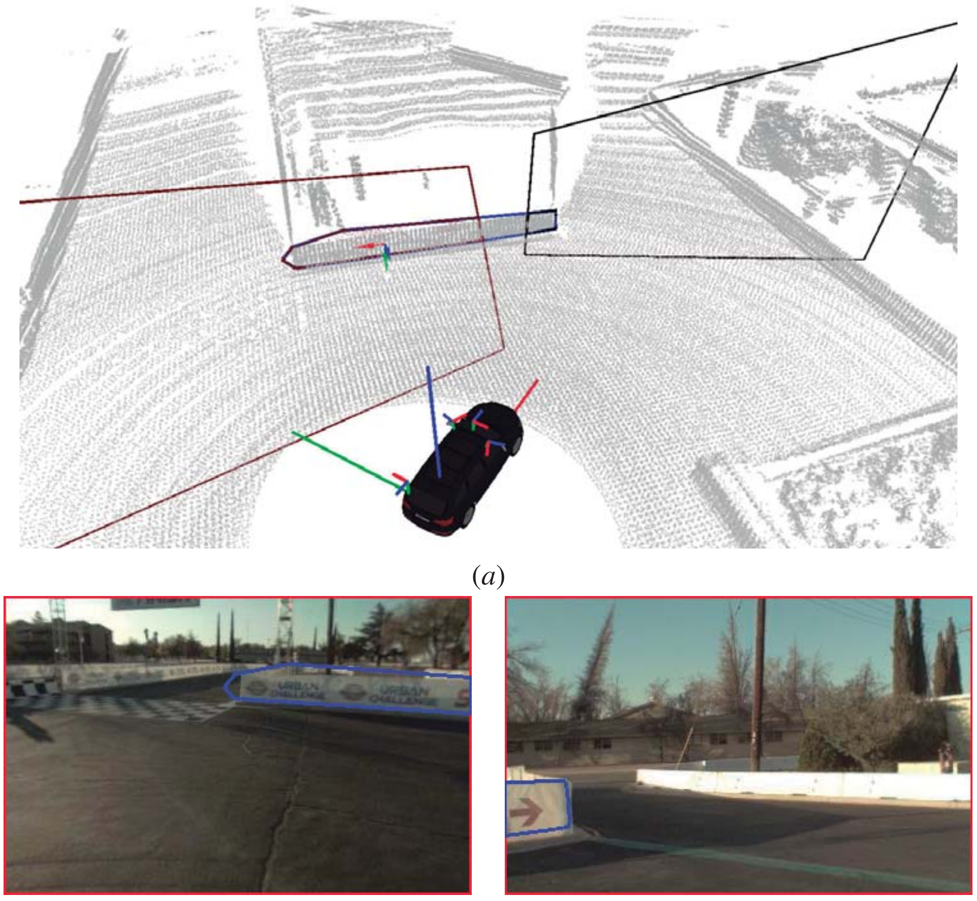


Figure 10.19: Texture mapping a primitive with images from different cameras: (*a*) the scene; (*b*) image from the forward center camera; (*c*) image from the forward center 6 millimeter camera; (*d*) right side detail of the triangulated mesh, forward center camera; (*e*) the same for forward 6 millimeter camera; (*f*) primitive with texture using forward center camera; (*g*) the same for 6 millimeter camera.

the vehicle, and several primitives in blue. The primitive in front of the vehicle is going to be texture mapped. The image bounding polygon can also be seen in black. As discussed before, the *Talos* vehicle has five onboard cameras. Two of them are facing the front of the vehicle. Hence, there are two alternative texture mappings for that primitive: using the forward center camera, or using the forward center 6 millimeter camera. The difference between these two cameras is that the latest has a larger focal length, and therefore it captures zoomed images of the environment. The forward center camera, on the other hand, has a smaller focal length but captures a wider view of the scene. The image from the forward center camera is shown in Fig. 10.19 (*b*), and the image from the forward center 6 millimeter camera is shown in Fig. 10.19 (*c*). In both Figs. the corresponding image bounding polygon is shown in red, and the projection polygon is shown in blue. It is observable that the 6 millimeter camera captures a more zoomed in image. The region contained by the projection polygon for the 6 millimeter camera is larger. Since both cameras have the same image resolution, the region also contains more pixels. As a consequence, it can provide a more refined mesh, with more vertices and triangles that describe changes in the image. Fig. 10.19 (*d*) shows a detail (on the right side of the primitive) of the triangulated mesh using the forward center camera, and the same detail is shown for the forward center 6 millimeter camera in Fig. 10.19 (*e*). As expected, in the case of Fig. 10.19 (*e*), there is a finner mesh, consequence of the larger number of pixels. The textured primitive is shown in Fig. 10.19 (*f*) (forward center camera) and Fig. 10.19 (*g*) (forward center 6 millimeter camera). The natural consequence of having finner meshes is that texture is also finner. This is clearly noticeable in a comparison of Figs. 10.19 (*f*) and (*g*).

10.4.2 Two Cameras Non Overlapping Projections

The previous case has shown how two cameras can be used to map a single primitive. For now, we do not consider how to fuse information from two or more cameras (this is only addressed in chapter 11). Hence, in the case depicted in Fig. 10.19, one would have to somehow make a decision on whether to use one camera or another. However, this is not always the case. Figure 10.20 shows a case where the same wall panel that is shown in Fig. 10.19 is used, but the vehicle is now at location *D* (of sequence 1). In other words, the vehicle is in a different position, which enables its forward center camera to view just a portion of the right side of the primitive, and its forward left camera to view another portion (on the left side) of the same primitive. Fig. 10.20 (*a*) shows the scene and the vehicle's position. In dark colors, the projections of the images projection polygons to 3D space are shown. From this its possible to see that both cameras view different portions of the primitive. This is also shown in the images from the cameras, in Figs. 10.20 (*b*) and (*c*). This situation opens the possibility of composite texture mapping. For instance, there are sometimes situations where two or more cameras can map different areas on the same primitive.


(*a*)

(*b*) (*c*)

Figure 10.20: A situation where two cameras map different portions of the same polygon: (*a*) the scene with the vehicle position; (*b*) image from the forward left camera; (*c*) image from the forward center camera.

10.4.3 Single Camera Multiple Primitives

Yet another case that may occur is that one single image can be used to texture map several primitives. In this example, the same demonstration scene that was used in section 10.3.4 is employed. Consider a single image (Fig. 10.21 (*a*)) that views a building. Three primitives are detected: the wall of the house, with the windows and the door (primitive 0), the roof of the house (primitive 1) and the brick wall in front of the house (primitive 2). The support planes of each primitive are depicted in Fig. 10.21 (*b*). Each primitive will generate a projection polygon, as depicted in Figs. 10.21 (*c*) (in image space) and (*d*) (in 3D space). Meshes are independently built using, for each primitive, the detected Hough line segments (Fig. 10.21 (*e*)) and information from the projection polygon. The entire set of inputs to the constrained Delaunay triangulations is shown in Fig. 10.21 (*f*). The computed triangulated meshes are shown in Fig. 10.22 (*a*) in the image space, and in Fig. 10.22 (*b*),

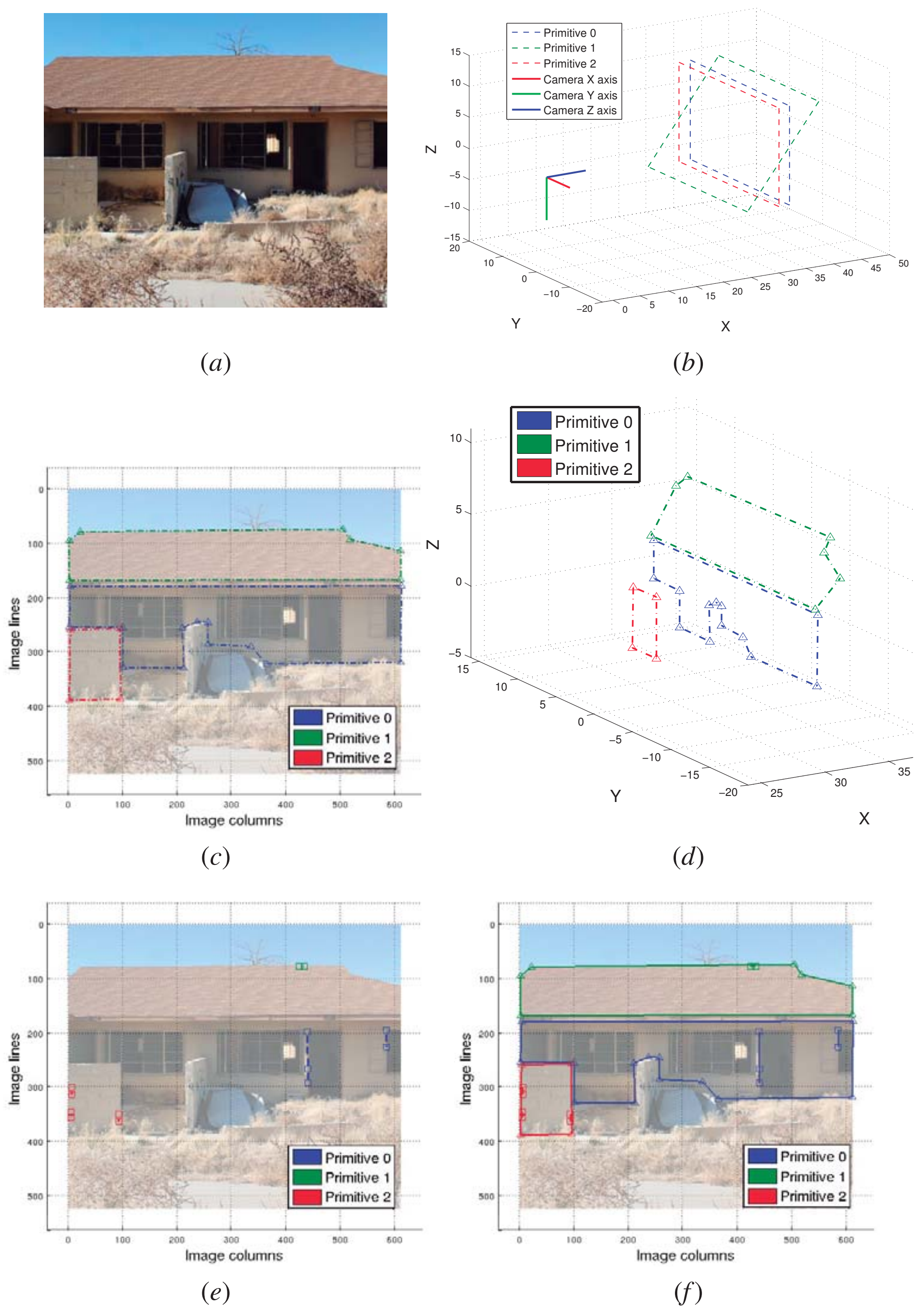
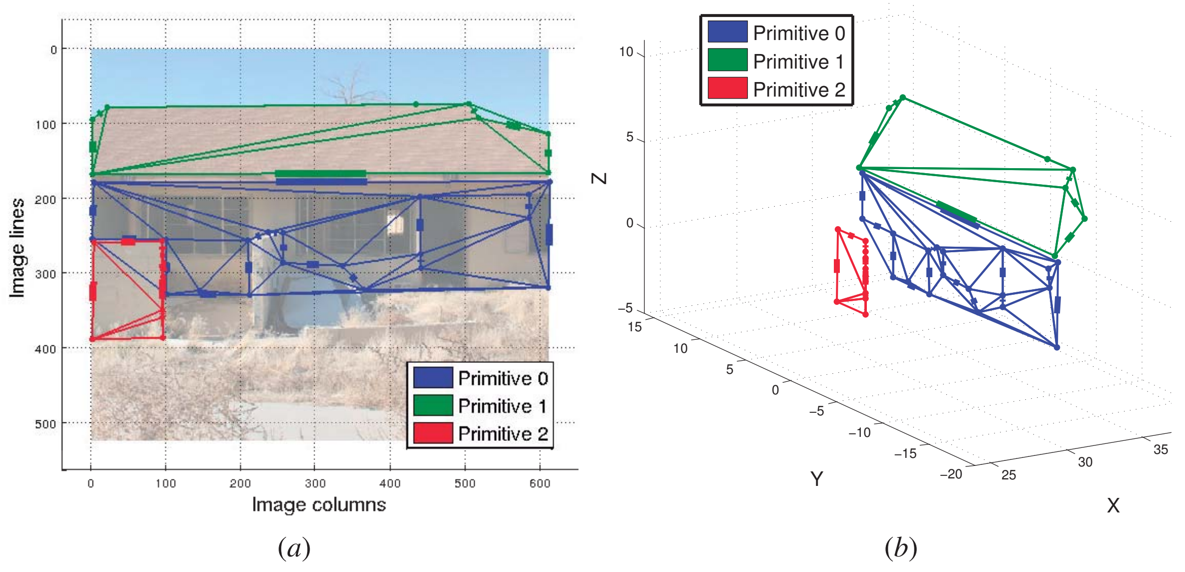


Figure 10.21: A case where a single image texture maps several primitives: (*a*) the image scene; (*b*) the support planes of each primitive; (*c*) the projection polygons in image space; (*d*) the projection polygons in 3D space; (*e*) the detected Hough line segments; (*f*) the input vertices and constraints to the Delaunay triangulation.



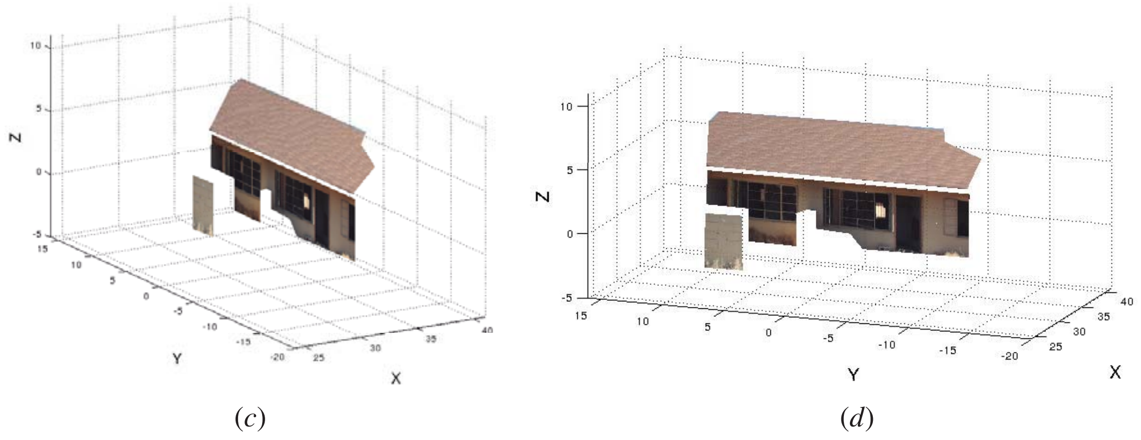
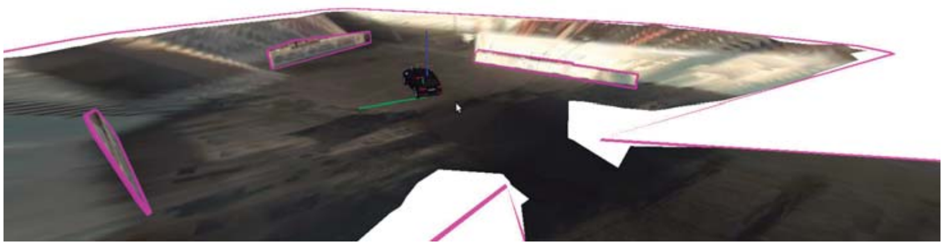


Figure 10.22: Triangulated meshes and textured primitives: (*a*) meshes in image space; (*b*) meshes in 3D space; (*c*) and (*d*) texture mapped primitives.

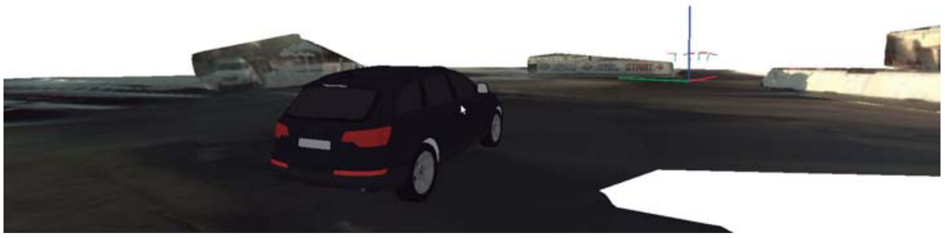
projected to the corresponding primitive's support plane. Finally, Figs. 10.22 (*c*) and (*d*) show the obtained textured scene.

10.4.4 Qualitative Analysis

Figure 10.22 shows a texture mapped complex scene representation containing several primitives. Note that also the ground plane primitive is shown in this case. The mechanism for texture mapping the ground primitive is exactly the same as for any other primitive. From this point of view, the proposed approach can be seen as a full generalization of the technique of Inverse Perspective Mapping, which was documented in detail in chapter 5.



(*a*)



(*b*)

Figure 10.23: Two different views of a textured complex scene representation, sequence 1.

10.5 Conclusions

This chapter has addressed the problem of texture mapping geometric polygonal primitives. Results show that it is possible to texture map complex scenes. Scene representations may contain several primitives and can make use of multiple cameras to provide the texture. The proposed approach is capable of dealing with several problems associated with the projection of textures onto to 3D models. Previous sections have described in detail the algorithms proposed to handle these problems, and the results presented in this section show that it is possible to map texture accurately. It is also important to highlight the flexibility of the proposed approach, since it can deal with combinations of multiple primitives and multiple projection cameras, in order to build complex 3D textured scene representations.

Chapter 11

Photometric Scene Refinement

This chapter proposes a framework for refining the photometric representation of the scene that is computed as described in chapter 10. In some cases, after the texture from an image is applied to a geometric polygonal primitive, another image is received that views the same primitive. If this second image contains higher quality texture, it should somehow replace the previous texture. In this chapter, the proposed mechanism for handling these cases is presented.

In section 11.1, the problem is introduced. Related work is presented in section 11.2. The proposed approach is described in section 11.3 and, finally, results and conclusions are given in sections 11.4 and 11.5.

11.1 Introduction

Chapter 10 has described in detail the proposed algorithms for mapping images to geometric polygonal primitives. As posed in that chapter, the problem could be stated as follows: given a description of a camera / image and a description of a polygonal primitive, how to map the texture in the image to the primitive. The texture mapping of one image to a camera was generically referred to as a projection. In the results presented in that section, several possibilities for the combinations of mapping were shown. Here is a small list:

- Projection of a single camera to a single primitive;
- Projections of multiple cameras to non overlapping regions of a single primitive;
- Projections of regions of a single camera to multiple primitives;
- Projections of multiple cameras to multiple primitives.

Despite the exhaustive list of examples that were shown, one was deliberately skipped. How to handle cases where one or more images map to overlapping regions of the same primitive. More than an example, it is rather a family of cases that require a higher level of generalization from the algorithm.

One of the advantages of the proposed scene representation is that it is capable of removing duplicate data. For example in the case of the geometric polygonal primitives, the expansion mechanism was able to discard data of the same object (see chapter 9 for details). As a consequence, the algorithm provides a deterministic representation of the environment. In other words, it makes a deterministic decision (even if based of probabilistic methods) about the shape, size and other properties of the scene. For example, a given geometric description of a wall in the scene places the wall in a specific position, without error margins or areas of probability. More than a limitation from the algorithm, this is more of a philosophic decision on how to approach the problem of scene reconstruction from massive multi sensor data. In our opinion, there are many advantages to this. We list here some of them. We will use the word *processing* to refer the computations done for obtaining a scene representation and the word *post-processing* to refer to any other processing that could be done on top of the reconstructed scene, i.e., that would use the reconstructed scene as input.

- Processing speed and complexity: given the amount of data used as input for the computation of a scene representation, it would be very difficult to generate a more complex probabilistic representation;
- Post-processing speed: if other subsequent processes are going to process the reconstructed scene, they will do it faster with a deterministic representation;
- Post-processing simplification: the level of complexity inherent to massive multi sensor data is lowered after reconstruction;
- Visualization: it would be quite difficult and counter intuitive to visualize a probabilistic representation of the scene.

Hence, the option is to have a scene representation that contains a single value for each property tuple it represents (a tuple could be a position in space, its color and temperature, for example). In 3D, for example, if two range measurements of the same object say that the object is at different locations, the objective of the scene representation algorithm is to end up with a description of the object that is unique. In the example, the object would have to be described as being positioned in one of the locations the two measurements described or, for example, in an average location. Still, the object is described in a single, unique location. When this philosophy is applied to photometric properties, the functionality should be exactly the same. An object (the same location of the object) must be described with a single color.

The problem is what to do when there are two or more color measurements of the same region of the same object. Under the current framework, the same question can be reshaped to: how to handle multiple camera projections that map onto overlapping areas of the same primitive. The following sections will present the algorithms proposed to solve this problem.

11.2 Related Work

Section 10.3.4 has described in detail the process of mapping images, or rather the portion of the images that are bounded by the projection polygon, onto the polygonal primitives. The process consists of computing a triangulated mesh contained inside the projection polygon. This is done, for each projection, in the image space. This local image triangulated mesh is referred to as $\mathtt{t}$. In all chapter 10, especially in the part where results are presented (section 10.4), only a single triangulated mesh is shown per primitive. Actually, to be more precise, some results in that section do present several meshes on the same primitive, but those meshes never overlap. That being said, when triangulated meshes are shown in 3D, what is actually shown are the projections to the 3D world coordinate frame of the local image mesh $\mathtt{t}$. In other words, what is shown are the triangulated meshes $\mathcal{T}$. This is the case in Fig. 10.14 (*b*), Figs. 10.19, (*d*) (*e*), or in Fig. 10.22 (*b*).

To illustrate the methods that will be discussed, the example mentioned in section 10.4.2 (shown in Fig. 10.19) is used. The triangulated meshes $\mathcal{T}$ for each of the four projections in that example are shown in Figs. 11.1 (*a*), (*b*), (*c*) and (*d*). Figures 11.2 (*a*), (*b*), (*c*) and (*d*) show the textures obtained using each of the local projections. From the figures it is possible to conclude that these meshes do overlap between each other, and that some projections provide considerably better quality textures than others.

In theory there are several alternatives to perform the fusion of the overlapped data. If we consider that there is a way of assessing the quality of each projection, then it would be possible to rank all projections in terms of the supposed quality it has to be texture mapped. Future sections will address this issue with more detail. For now let us just consider that there is a function $\mathbf{q}(\mathbf{C}^{\{k,l,t\}})$ that can output a normalized score value for each of the projections, that translates the quality the projection has for texture mapping. Furthermore, let us consider that in addition to the measurement of the quality of each projection in general, the quality function can evaluate and score each of the triangles in the triangulated mesh of each projection. Let $\mathtt{t}_i^{\{k,l,t\}}$ denote the *ith* triangle in the local mesh of projection $\{k, l, t\}$. The quality function that evaluates each triangle is notated as $\mathbf{q}(\mathtt{t}_i^{\{k,l,t\}})$. If these functions are available, then some other alternative fusing strategies could be devised. Below, some possible strategies for achieving the texture mapping of a primitive with multiple projections are listed:

- Texture map all local textures to 3D space and use alpha channel to average them;
- Compute a new triangulated mesh (in the 3D space) using all the vertices and constraints of local meshes as input;

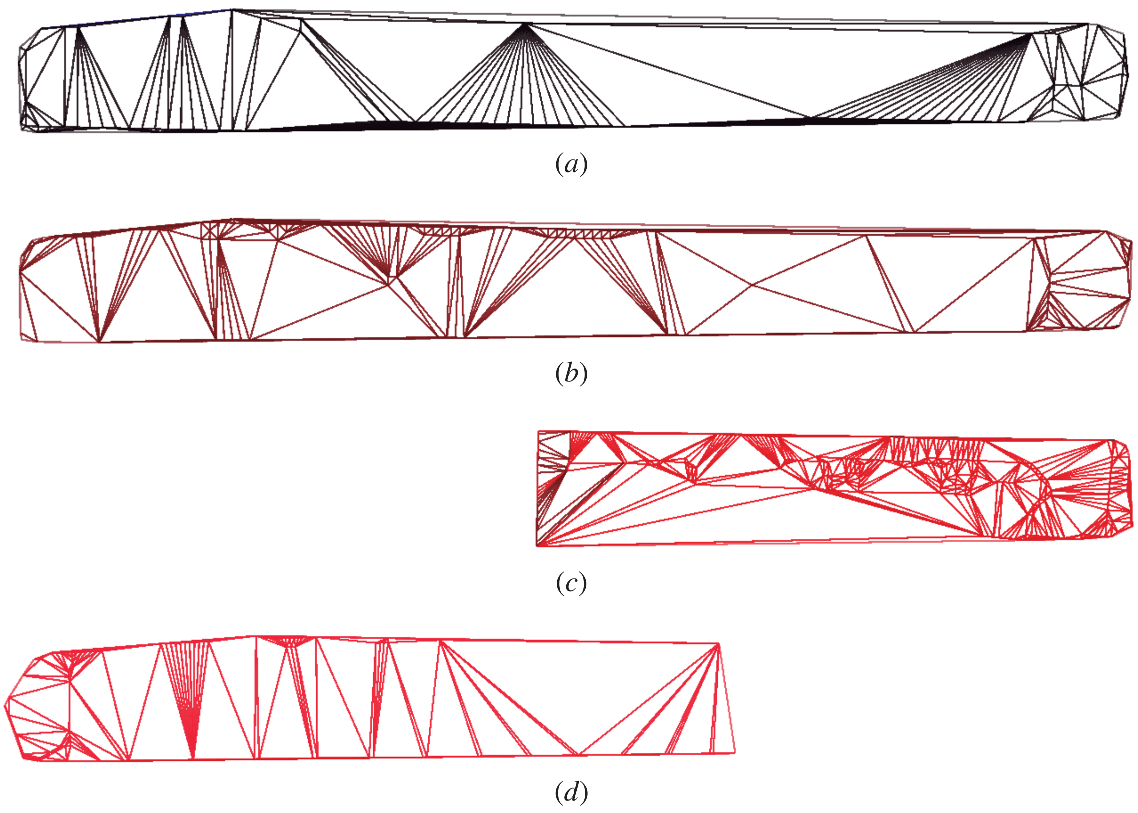
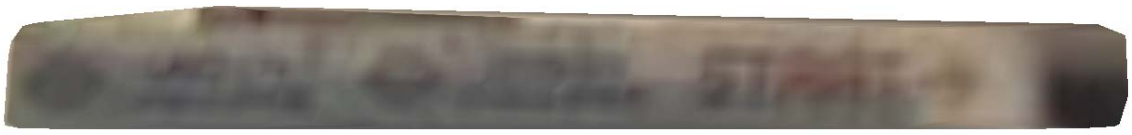


Figure 11.1: Local image meshes from example shown in Fig. 11.8: (*a*), (*b*), (*c*), (*d*), local triangulated meshes from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively.

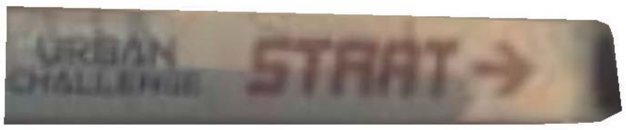
- Rank all projections using the projection quality function and texture map only from the best scoring projection;
- Rank all triangles in all projections using the triangle quality function. When two triangles overlap, map only the best scoring one.

The first option listed is the most straightforward one. It consists of averaging the textures provided by each local mesh. This could be achieved by setting the alpha channel of all local meshes so that they average out. The primitive would have several layers, each with a given triangulated mesh, as shown in Fig. 11.3 (*a*). Figure 11.3 (*b*) shows the texture obtained using an alpha map strategy for the fusion. The results are not good. Several artifacts arise from the *blind* averaging of all the textures. Furthermore, there are also other disadvantages with this approach. Since all local meshes are kept, the memory required to store local meshes from all projections would become quite large. Another problem is that, with the increase of projections, the computational load to project all the meshes to

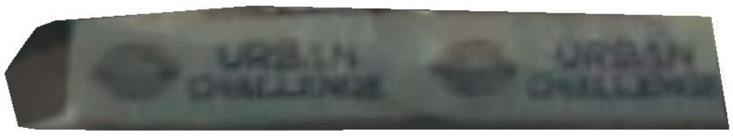
(*a*)



(*b*)



(*c*)

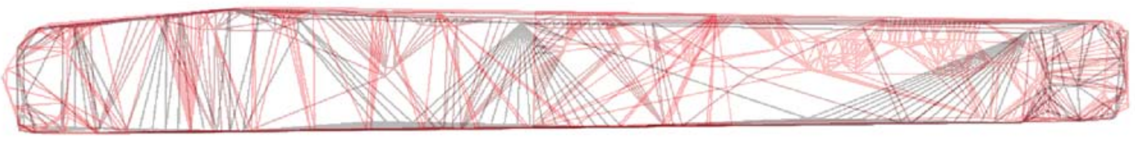


(*d*)

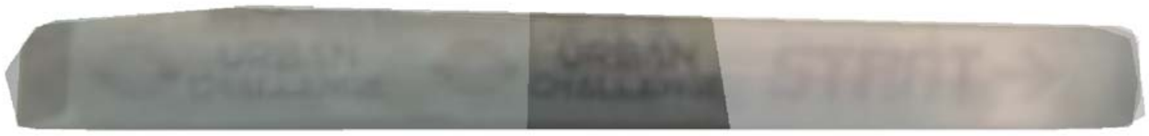
Figure 11.2: Textures obtained from local image meshes of the shown in Fig. 11.8: (*a*), (*b*), (*c*), (*d*), textures from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively.

the 3D space would increase as well. For all these reasons, it does not seem to make sense to keep all the local projection triangulated meshes and mix them at the time of visualization. Anyhow it does not seem to be a good solution just to use all the meshes at the same time. From the observation of Fig. 11.2 it is possible to conclude that some textures have a much better quality than others.

The second listed option is to compute a new triangulated mesh, using as inputs all the vertices and constraints of the local meshes. This new mesh would have to be computed on the primitive's local coordinate frame. Figure 11.4 illustrates this strategy. Consider four projections of the wall panel primitive depicted in Figs. 11.1 and 11.2. For each projection, the global triangulated mesh is recomputed by adding the vertices and constraints of the current projection. Figures 11.4 (*a*), (*c*), (*e*) and (*g*) show all the vertices used as input to the global mesh with one, two, three and the four projections, respectively. One problem with this approach is that there is always some error associated with the projection. That means that two projections will never texture map the objects precisely at the same position. Because of this, the vertices of a second projection never overlap the first, even if they are viewing exactly the same portion of the same object. Due to precision errors, the mapping



(*a*)



(*b*)

Figure 11.3: Textures obtained from local image meshes of the shown in Fig. 11.8: (*a*), (*b*), (*c*), (*d*), textures from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively; (*e*) the overlay of all textures using an alpha channel fusion strategy.

is always slightly deviated. As a consequence, the textures generated from all the vertices appear to be flickered. Figures 11.4 (*b*), (*d*), (*f*) and (*h*) show the textures generated for each of the meshes computed from the set of vertices shown in Figs. 11.4 (*a*), (*c*), (*e*) and (*g*), respectively. Figure 11.4 (*i*) shows a detail of Fig. 11.4 (*h*). Here, the flickering phenomena is clearly visible. The word *start* written on the right side of the panel is actually deteriorated from the fact that all the vertices are kept. In addition to the poor quality textures, there are also other disadvantages associated with this strategy. Since all the local meshes' vertices are added, the global mesh will quickly grow in size and require large amounts of memory space.

The problem with the second listed option seems to be that the vertices from different projections that in fact were views of the same location of the same object are projected to slightly different positions. From this idea a more complex derivation of the previous option could be devised: to test for each vertex whether or not it is close enough to a vertex already mapped from a previous projection. In other words, the procedure consists of testing if vertices are very close (Euclidean distance) to others, and, if so, to handle their insertion onto the global mesh. Handling insertion here means that if two vertices are mapped very close to each other, only one can be present in the global mesh. The choice of which vertex should be kept can be done using the projection quality functions discussed above. The procedure can be viewed as a conditional insertion of all the vertices from local image meshes to the global mesh. Figure 11.5 shows an example of such a procedure. In the shown example, the vertices from the more recent meshes, signaled with higher temperature colors in Fig. 11.4, will always have better quality than the vertices from the old meshes.

In Fig. 11.5 (*a*), only two projections are added to the global mesh. Then in Figs 11.5 (*b*) and (*c*), two other projections are added and new vertices appear in the global mesh. At the same time, vertices from older projections tend to disappear. However, in the final global mesh there are still

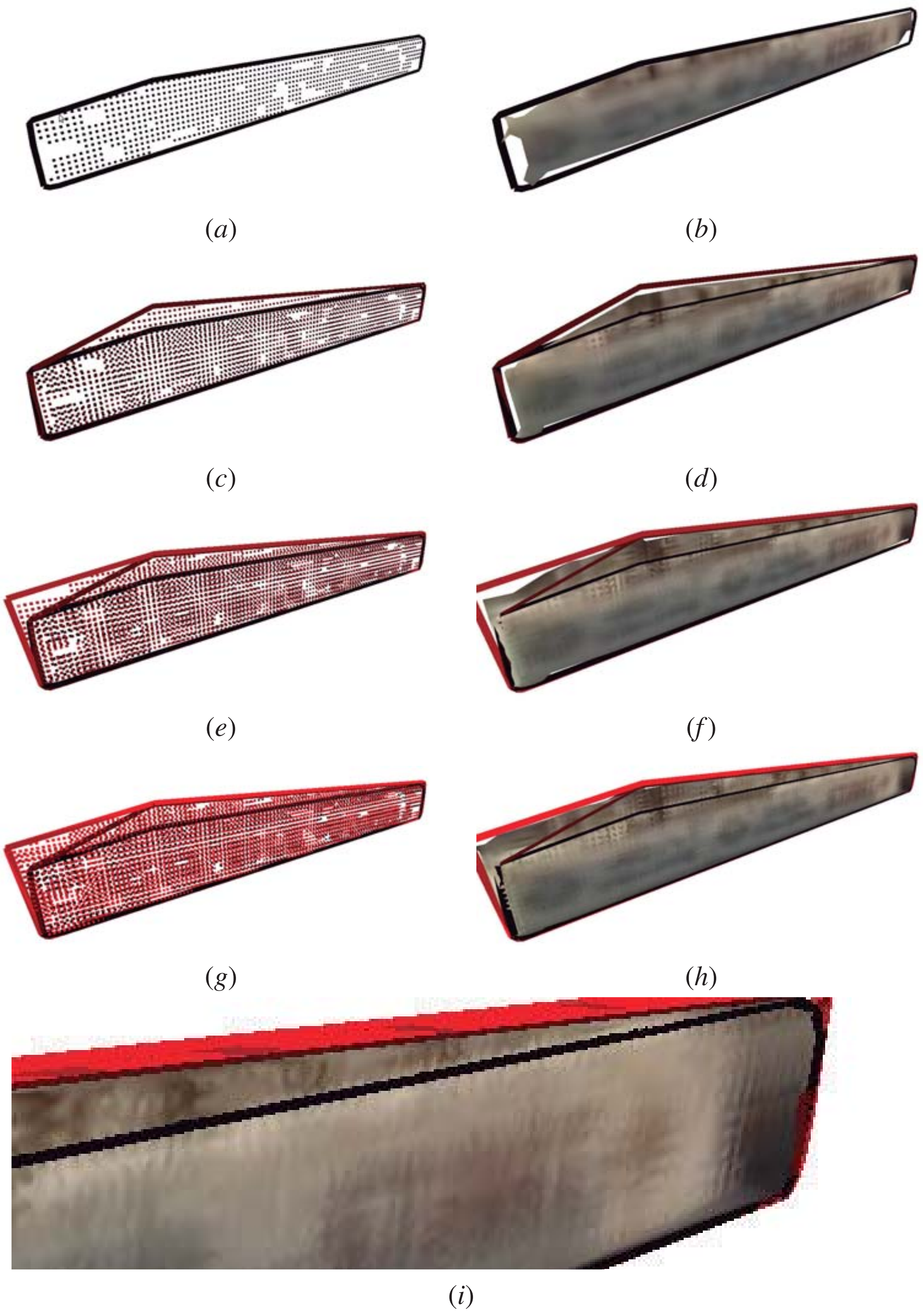


Figure 11.4: Adding all vertices in the local image meshes to produce a global mesh. Projections are colored with a black to orange colormap: (*a*), (*c*), (*e*) and (*g*) all the vertices used as input to the global mesh with one, two, three and four projections, respectively; (*b*), (*d*), (*f*) and (*h*) textures generated for each of the meshes computed from the set of vertices from (*a*), (*c*), (*e*) and (*g*), respectively. (*i*) a detail of (*h*).

vertices from multiple projections. As a result, the texture, shown in Fig. 11.5 (*i*), shows a very poor quality. This approach is also incapable of executing a reasonable fusion of the visual information from multiple projections.

The third option concerned the ranking, according to the quality function, of all the available projections. Then, only the best ranked projection is texture mapped. Such approaches do not contemplate the fact that the best projection may not be able to map an entire primitive. For example, from an observation of all the local meshes in Fig. 11.2, one could argue that the best quality mesh is in fact the one provided by projection $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ (shown in Fig. 11.2 (*c*)). The problem is that if

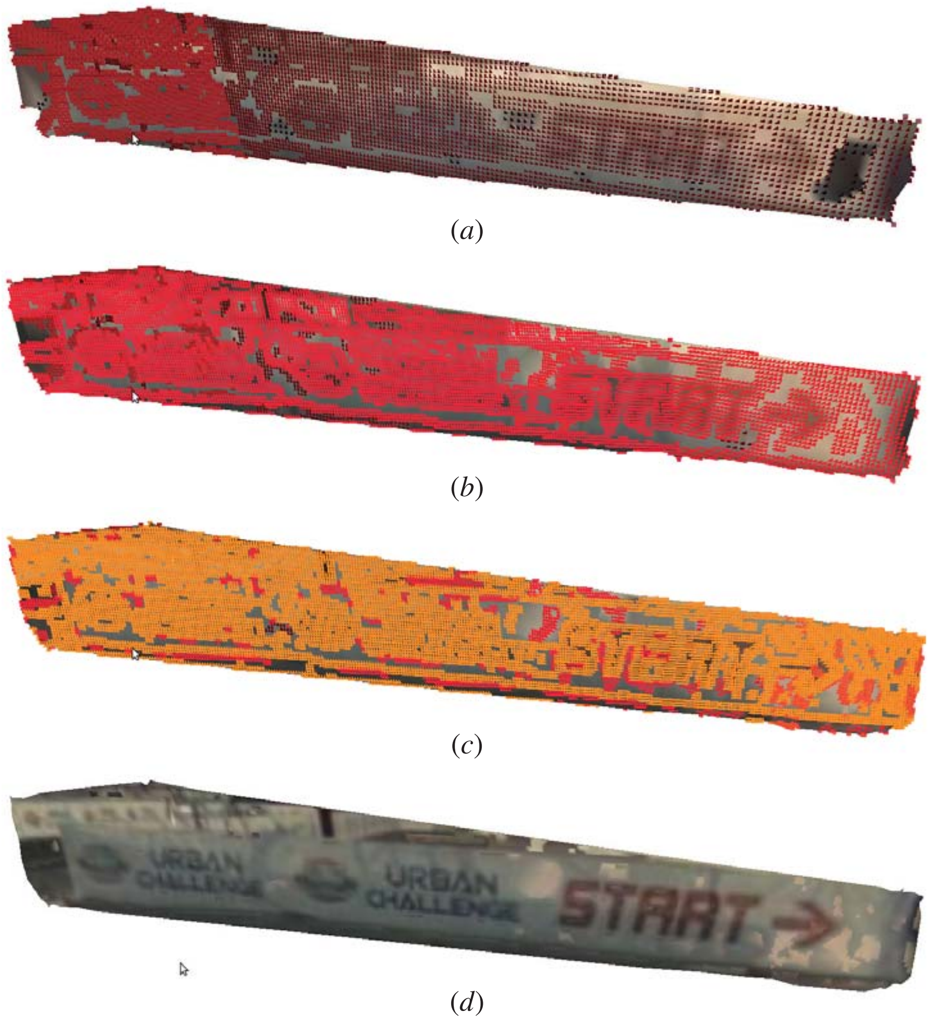


Figure 11.5: Conditional adding of all the vertices in the local image meshes to produce a global mesh. Projections are colored with a black to orange colormap: (*a*), (*b*) and (*c*), all the vertices used as input to the global mesh with two, three and four projections, respectively; (*d*) the texture mapped from the final stage of the global mesh.

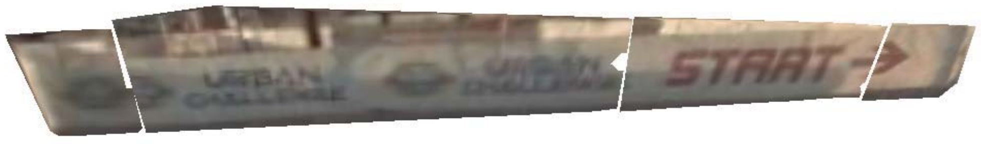


Figure 11.6: Mapping using triangle quality functions.

this mesh is used for texture mapping the left side of the primitive is left unmapped.

To avoid that problem, the fourth option listed concerned the ranking of triangles instead of the complete projections. All triangles from all the local meshes are inserted onto the global mesh. Whenever two triangles intersect, the one with the highest quality is kept and the other is discarded. However, there is a problem: erased triangles often overlap other triangles in just a portion of their area. When deleted, these triangles leave empty spaces where no texture is provided. This phenomena is visible at the regions where triangles from one projection connect to triangles from another projection. The resulting texture using this method is shown in Fig. 11.6. The untextured gaps at the frontiers between projections make the overall quality of mapping insufficient.

In conclusion, none of the strategies seems to provide reasonable textures. Each have their own problems. In the following section, we present the proposed approach for solving this problem. It consists in a slightly more complex derivation of the fourth option, but, as will be shown, is capable of generating higher quality textures.

11.3 Proposed Approach

This section addresses the problem of how to represent multiple projections, in particular what is the proposed framework to deal with multiple projections over time. One issue that was deliberately left out of the discussions and results of chapter 10 was the mission time. In section 10.3.3, the deductions that end up in eq. (10.23) proposes an expression for the computation of the projection polygon, where right side super indexes indicated a dependence over variables k, the primitive index and l, the camera index. Under the assumption proposed in that chapter, that only a snapshot in time was used for each scene reconstruction, e.g., mission time $t = t_p$, then the projection polygon would only be dependent on the camera and the primitive. This is why the projection polygon was notated as $\mathtt{h}^{\{k,l\}}$ in eq. (10.23). However, if we wish to generalize entirely and do not make the assumption that $t = t_p$, but rather that $t \in [0, \infty[$, then an observation of eq (10.23) shows that the variable mission time affects the transform from the vehicle to the world coordinate frames, ${}^{\mathbf{V}}\mathbf{T}^{t}_{\mathbf{W}}$. This makes perfect sense if one considers that the vehicle is moving around in a scenario. As the vehicle moves around (and variable t increases), the transform ${}^{\mathbf{V}}\mathbf{T}^{t}_{\mathbf{W}}$ is constantly updated, to reflect the fact that the vehicle, and the onboard cameras, are in a different position in the world. As a consequence, the projection polygon must be notated not as $\mathrm{h}^{\{k,l\}}$ but instead as $\mathrm{h}^{\{k,l,t\}}$. This means that, a single camera $l = l_p$ can generate multiple projections over a single primitive $k = k_p$, as long as there is a difference in variable t.

This insight is of particular importance to the framework that is going to be proposed. There is one unique projection to each different set of the tuple $\{k, l, t\}$. The framework we propose is interesting

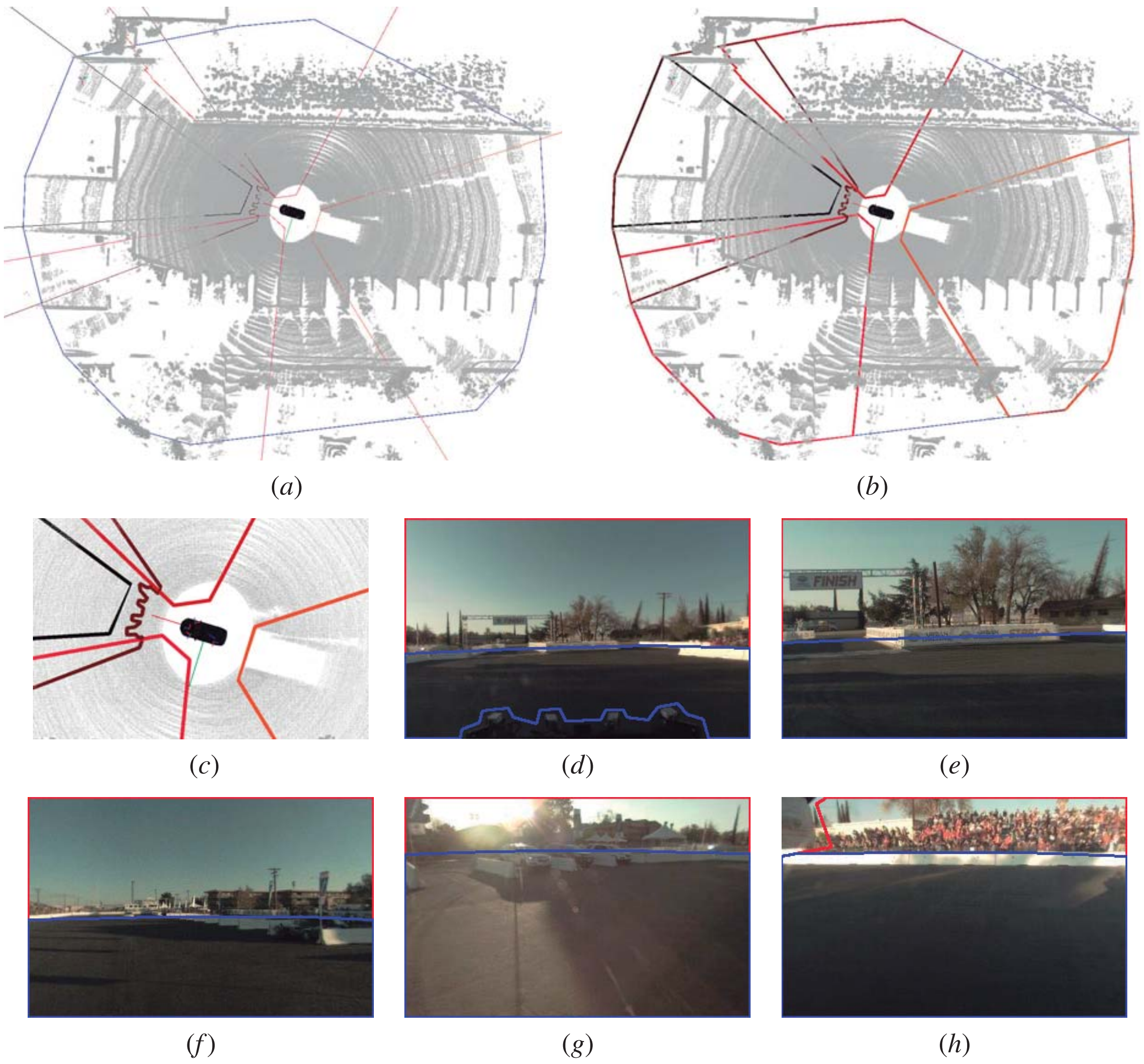


Figure 11.7: An example of overlapping projection polygons in multiple projections of the same mission time: (*a*) a 3D scene with the primitive $k = 0$ (in blue) and the image bounding polygons (black to orange colormap); (*b*) the same scene with the projection polygons represented; (*c*) a detailed view of (*b*); (*d*), (*e*), (*f*), (*g*), (*h*), images from projections $\mathbf{C}^{\{k=0,l=0,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=1,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=2,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=3,t=t_p\}}$ and $\mathbf{C}^{\{k=0,l=4,t=t_p\}}$, respectively.

since it generalizes a projection using such a tuple. In doing so, all cases are considered, those that were presented in chapter 10 and others where there is overlap between two or more projections to the same primitive. Note that, under the proposed framework, there is no actual difference between two overlapping projections from the same camera at different times, or two overlapping projections from different cameras at the same time or even at different times. All cases are treated in the same way.

Two examples are provided to show how projections are referenced. In the first example, shown in Fig. 11.7, a polygon representing the ground plane ($k = 0$) is to be texture mapped using information from all five cameras onboard the *Talos* $l \in \{0, 1, 2, 3, 4\}$. In this case, a single moment in time is considered, thus $t = t_p$. Projections are notated by the symbol $\mathbf{C}^{\{k,l,t\}}$. Figure 11.7 (*a*) shows the scene around the robot, bounding polygon of the polygonal primitive $\mathcal{P}^{\{k=0\}}$ shown in blue. The image bounding polygons $\mathcal{R}^{\{k=0,l=\{0,1,2,3,4\},t=t_p\}}$ (projected to 3D space) are shown in colors black to orange. A black to orange colormap is used to color the projections as they are computed. Hence, a black signaled primitive was computed first, and the orange was computed more recently. The highest the color temperature, the more recent the primitive. Figure 11.7 (*b*) shows the projection polygons $\mathcal{H}^{\{k=0,l=\{0,1,2,3,4\},t=t_p\}}$ (in 3D space), using the same colormap. Figure 11.7 (*c*) shows a detail of Fig. 11.7 (*b*). In both figures, overlapping of the projection polygons is clearly visible. There are five projections, each corresponding to one of the cameras in the *Talos*. Actually, there are five projections because each of the images from the cameras does have some portion of information about the primitive in question. This remark is just to say that sometimes cameras do not generate a projection. It is the case when the projection polygon does not exist, which occurs if there is no intersection between the image bounding polygon and the primitive's bounding polygon (see eq. (10.21)). Figures 11.7 (*d*), (*e*), (*f*), (*g*), (*h*) show the images for the front center ($l = 0$), front center 6 millimeter ($l = 1$), rear center ($l = 2$), front left ($l = 3$) and front right ($l = 4$) cameras respectively. In this example it is shown that overlaps in the projection polygons of several cameras may occur, even for a fixed mission time. In this case the ground primitive's texture would have to be computed using all five projections, that is, using information from projections $\mathbf{C}^{\{k=0,l=0,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=1,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=2,t=t_p\}}$, $\mathbf{C}^{\{k=0,l=3,t=t_p\}}$ and $\mathbf{C}^{\{k=0,l=4,t=t_p\}}$.

The second example is shown in Fig. 11.8. In this case, the vehicle travels from location *C* to location *D*, sequence 1 of the Massachusetts Institute of Technology (MIT) data set. Images are feed to the algorithm at three locations: location *C*, location D and an intermediate location between those two. Figure 11.8 (*b*) shows the vehicle at each location. Only images from cameras front center ($l = 0$) and front left ($l = 3$) are given to the algorithm. In this example, the primitive (index $k = 4$) corresponds to the wall panel in front of the vehicle, represented in blue, on the left side of Figs. 11.8 (*a*) and (*b*). Also, it is considered that the vehicle was at location *C* at mission time t_0, at the intermediate location at $t = t_1$, and at location *D* at time $t = t_2$. At location *C* (right side vehicle in Fig. 11.8 (*b*)) both the front center and front left cameras are tested to see if they can

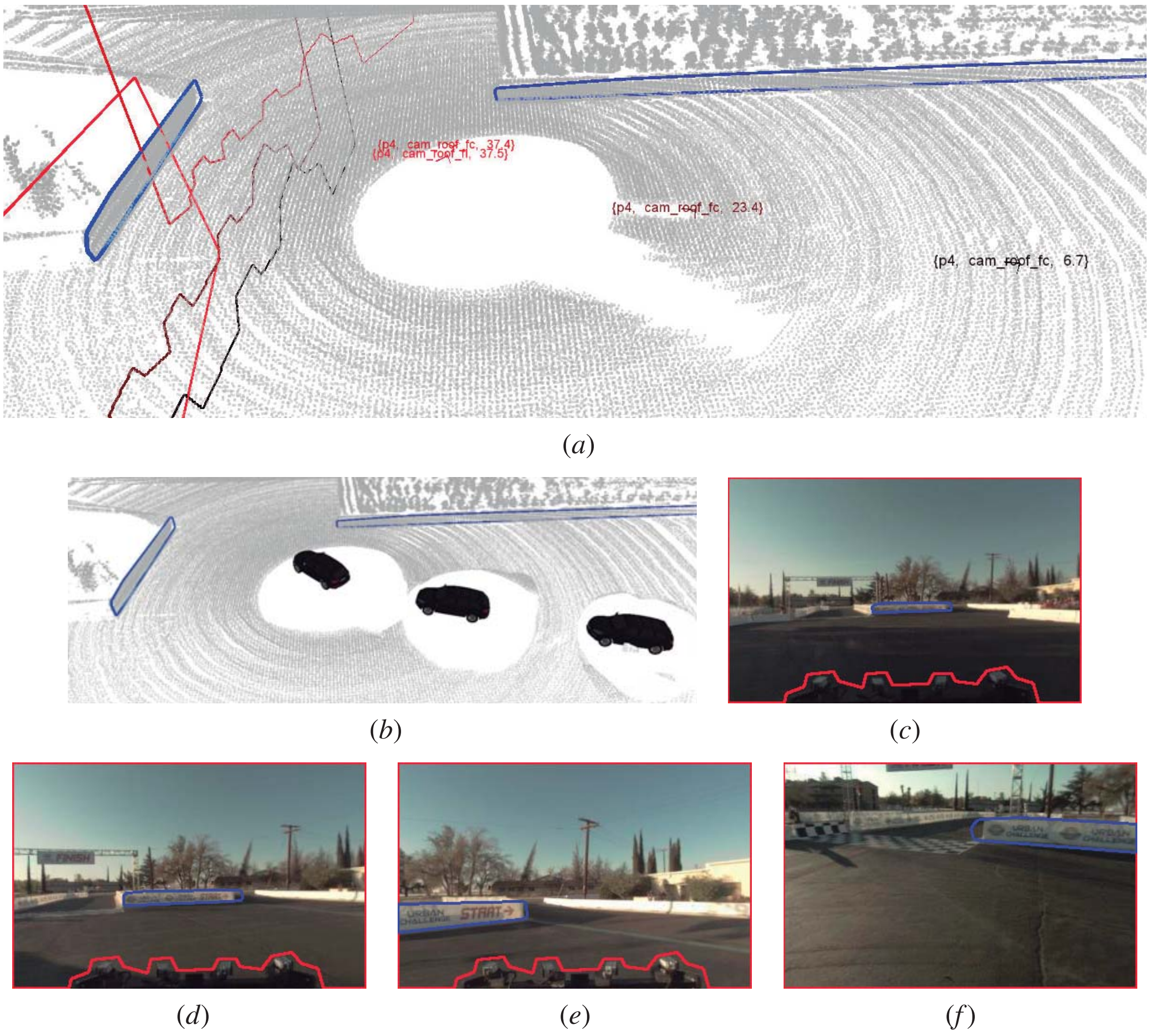


Figure 11.8: An example of overlapping projection polygons in multiple projections at different mission times t_0, t_1 and t_2: (*a*) a 3D scene with the primitive $k = 4$ (in blue) and projection polygons (black to orange colormap); (*b*) the vehicles position at times t_0, t_1 and t_2, respectively from right to left in the figure: (*c*), (*d*), (*e*), (*f*), images from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively.

generate a projection. However, since that, at location *C*, the vehicle is front facing the wall primitive, only the front center camera generates a projection, projection $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$. The front left camera does not view the primitive in question at that location. At the intermediate location (vehicle in the middle of Fig. 11.8 (*b*)), the same occurs and a new projection is added, projection $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$. At location *D*, the vehicle has turned right which makes so that both cameras view a portion of the primitive. Projections $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$ are added. Figure 11.8 (*a*) shows the projection tuples at their corresponding position. The same black to orange colormap as that of the previous example is used. Image bounding polygons for each projection are also shown with the same colormap. Figures 11.8 (c), (d), (e) and (f) show the images for projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively. Also in this example there is a significant overlap between the several projection polygons.

Using a tuple based on the identification of the primitive, the camera, and the mission time it is possible to have a description of a unique projection. The term projection is understood as an image captured from a camera that can be used to map some texture to the polygon. There are several properties associated to a projection. By properties we mean variables that are used to describe the projection or that are computed to execute the projection and stored in the projection data structure. The previous three properties are obvious, but there are some others. All of them are listed in the following lines:

- A primitive k to where the image should be projected to, defining the plane coefficients a, b, c and d and the primitive's bounding polygon $\texttt{p}$, $\mathcal{P}$ or $\texttt{P}$;
- A camera l that provided the image;
- A mission time t associated with the time in which the image was captured;
- The image captured by the camera at time t;
- An image bounding polygon $\texttt{r}$, $\mathcal{R}$ or $\texttt{R}$;
- A projection polygon $\texttt{h}$, $\mathcal{H}$ or $\texttt{H}$;
- A triangulated mesh $\texttt{t}$, $\mathcal{T}$ or $\texttt{T}$.

Note that right super indices are removed from some variables to simplify the notation, since it is obvious in this case that variables depend on the projection and thus, of the $\{k, l, t\}$ tuple. The notation for symbols that was introduced before is kept: machine type small symbols, e.g., $\texttt{r}$, for $\mathbb{R}^2$ data in image space, handwritten symbols, e.g., $\mathcal{R}$, for $\mathbb{R}^3$ data represented in 3D world coordinate frames, and machine type capital symbols, e.g., $\texttt{R}$, for $\mathbb{R}^2$ data in the primitive's local coordinate frame.

Although the itemized list signals all possibilities of representation for some variables, e.g., $\texttt{r}$, $\mathcal{R}$ or $\texttt{R}$, this does not mean all this representations are actually stored. In fact, typically, only one representation is stored in memory and the others are computed when necessary (see section 10.3.3 for details on this topic).

11.3.1 Mapping Multiple Projections

Several possible approaches to the problem of how to handle multiple projections in the texture mapping of a primitive were described. Some approaches concerned the addition, conditional or not, of the vertices in the local meshes to compute a global mesh. These approaches are those which give less interesting textures. Several reasons have been exposed when these approaches were analysed. However, there is one which was not addressed and that is the most important. Local meshes are computed using constrained Delaunay triangulations. These triangulations take as input a set of vertices and a set of constraints. These are generated by a Hough line detection algorithm, that selects the ideal constraints so that triangulated mesh contains triangles with smooth color transitions. Hence, local meshes contain several triangles. The configuration of the mesh, that is, the triangles and the edges, is defined so that the texture mapping operation is accurate. Different mesh configurations do not ensure the same accuracy. Chapter 10 provides details on this subject. That being said, the atomic unit for projection should be the triangles, since that Data Dependent Triangulation (DDT) ensure that they map texture accurately. Vertices based approaches cannot guarantee that triangles are kept. Thus, they also do not ensure that the local mesh configurations are passed to the global mesh. In our opinion, this is the main reason why vertex based approaches show such poor texture mapping performance.

The strategies with the best potential for obtaining accurate mappings seem to be the strategies where triangles from the local meshes are added to the global mesh. The biggest problem in these cases is that erased triangles often leave untextured regions between the borders where the primitives overlap.

In this section, we present an algorithm that is capable of texture mapping primitives from multiple projections. The philosophy behind the method is that, in order to ensure the most accurate texture possible, local meshes' triangles should be kept with the same configuration. In principle, this can ensure an accurate texture.

Let $\mathtt{M}$ be the global triangulated mesh. Only a single global mesh exists per primitive. Like local image meshes, the global mesh is also defined in $\mathbb{R}^2$. However, global meshes are defined in the corresponding local primitive's coordinate frame. This global mesh can also be viewed in 3D world coordinates ($\mathcal{M}$), or even projected to some image space ($\mathtt{m}$). Note that it is not the same to refer a local triangulated mesh represented in 3D ($\mathcal{T}$) or the global mesh represented in the same space ($\mathcal{M}$). While $\mathcal{T}$ notates the transform of a local image mesh ($\mathtt{t}$) to 3D space, $\mathcal{M}$ notates the global mesh. For simplification purposes, let the right super index j indicate a projection with an unique tuple. A local triangulated mesh from projection $j = \{k, l, t\}$ is notated as $\mathtt{t}^j$. Local triangulated meshes contain T^j number of triangles. Individual triangles are notated as $\mathtt{t}_i^j, \forall i \in \{0, 1, ..., T^{\{k,l,t\}}\}$, when indicating the ith triangle of the local mesh j, or notated as $\mathtt{t}^j_{\{\mathtt{v}_1,\mathtt{v}_2,\mathtt{v}_3\}}$, in the case the vertices $\mathtt{v}_1$, $\mathtt{v}_2$ and $\mathtt{v}_3$ are specified. Likewise, triangles in the global mesh are notated as $\mathtt{M}_n, \forall n \in \{0, 1, ..., N\}$, where N is the number of triangles in the global mesh. When the vertices of the triangles are specified, then the notation $\mathtt{M}_{\{\mathtt{v}_1,\mathtt{v}_2,\mathtt{v}_3\}}$ is used.

The algorithm works by incrementally building the global mesh. The term incremental is used since the global mesh is updated every time a new projection is used to refine it. The update of the global mesh with a new projection is executed by iterating over all triangles in the local projection mesh. Each triangle may or may not update or refine the mesh. For each triangle $\mathtt{t}_i^j$, a set of operations are executed. First, all the operations are listed. Later on, each will be described in detail.

- Map $\mathtt{t}_i^j$ (in image space) to the primitive's local coordinate frame ($\mathtt{T}_i^j$). This new triangle is referred to as the candidate triangle;
- Assess if there is any overlap between the candidate triangle ($\mathtt{T}_i^j$) and any of the existing triangles in the global mesh $\mathtt{M}_n, \forall n \in \{0, 1, ..., N\}$. Compute a list of overlapping global mesh triangles;
- Assess the benefit of inserting the candidate triangle to the mesh;
- If there is benefit, execute the insertion of the candidate triangle in the global mesh.

The first step is to compute what is called the candidate triangle. A local triangle is mapped to the local primitive's coordinate frame. In other words, each triangle $\mathtt{t}^j_{\{\mathtt{v}_1,\mathtt{v}_2,\mathtt{v}_3\}}$ is mapped to $\mathtt{T}^j_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}$. Section 10.3.3 has described in detail how direct and inverse projection expressions are used to transform points defined in the image coordinate space to points defined in the primitive's local coordinate frame, e.g., from $\mathtt{t}$ to $\mathtt{T}$. Hence, no further details are required on this topic. Let **map** be a function that transforms points in the image coordinate frame to points in the primitive's local coordinate frame. The mapping of a triangle from one space to the other is done by mapping the vertices of the triangle:

$$\mathtt{V}_o = \mathbf{map}(\mathtt{v}_o), \quad \forall o \in \{1, 2, 3\}. \tag{11.1}$$

Using eq. (11.1) it is possible to map triangle $\mathtt{t}^j_{\{\mathtt{v}_1,\mathtt{v}_2,\mathtt{v}_3\}}$ to triangle $\mathtt{T}^j_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}$ and thus generate the candidate triangle.

The second step is to check whether the candidate triangle overlaps with any of already existing triangles on the global mesh. Let $\mathbf{intr}(A, B)$ be a function that tests intersection between triangles A and B. The test can be written as:

$$do\_intersect = \mathbf{intr}(\mathtt{T}^j_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}, \mathtt{M}_n), \forall n \in \{0, 1, ..., N\}, \tag{11.2}$$

where N corresponds to the total number of triangles in the global mesh $\mathtt{M}$. There are several approaches to triangle triangle intersection tests, that provide fast and efficient algorithms [Moller 1997] [Chang & Kim 2009] [Sappa & García 2000].

In the particular application at hand the objective is to detect if there is overlap between the triangles. Here, there is a distinction between overlap and intersection. In fact, what must be assessed is whether or not an insertion of the candidate triangle onto the global mesh will affect the configuration of the latest. To affect the existing configuration means to change the configuration of the already existing triangles. This is referred to as the overlap test. This concept is not the same as a pure intersection test, since that there are some cases where the triangles do intersect but the mesh configuration is not altered. The overlap test is based on a set of rules that analyse the output of the intersection test. In our case, the intersection implementation in [Fogel *et al.* 2012] is used. Details are provided in [Devillers & Guigue 2002]. This is a sophisticated implementation since it returns not only whether or not the triangles intersect but also what are the geometric types computed from the intersection. A triangle to triangle intersection can result in an empty set, whenever there is no intersection, in a point, a line segment, or a polygon.

Figure 11.9 shows several examples. Consider the already existing global primitive mesh $\mathtt{M}$ to be composed by a single triangle (depicted in blue in Fig 11.9). The candidate triangles are shown in red. In each case, the geometries returned by the intersection function are as follows: an empty set (*d*), points (*a*) and (*e*), line segments (*b*) and (*f*), and polygons (*c*). If we consider the cases where the insertion of the candidate triangle (in red) does not change the configuration of the already existing global mesh (in this case, the initial global mesh is composed of a single triangle, in blue), we can say that in case (*a*), (*b*) and (*c*) the mesh would be altered, and that, in cases (*d*), (*e*) and (*f*) the mesh would remain unaltered. This concept will be more easily understood when the triangle insertion mechanism is presented. The overlap test is based on a set of rules that analyse the return of the intersection function (**intr**), between candidate triangle $\mathtt{T}_i^j$ and global mesh triangle $\mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}$. It returns *yes* if the triangles overlap or *no* otherwise. Note that the algorithm does not consider the possibility of degenerate triangles. These cases are handled previously. The algorithm is detailed in eqs. (11.3), (11.4) and (11.5):

$$\begin{cases} no, \quad if\left(\mathbf{intr}\left(\mathtt{T}_i^j, \mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}\right) = empty\ set\right) \\ yes, \quad if\left(\mathbf{intr}\left(\mathtt{T}_i^j, \mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}\right) = list\,of\,polygons\right) \\ Go\,to\,(11.4), \quad if\left(\mathbf{intr}\left(\mathtt{T}_i^j, \mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}\right) = points\,\mathbf{X} : \mathbf{X} = \{\mathtt{X}_0, \mathtt{X}_1, ..., \mathtt{X}_N\}\right) \\ Go\,to\,(11.5), \quad if\left(\mathbf{intr}\left(\mathtt{T}_i^j, \mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}\right) = line\,seg.\,\mathbf{L} : \mathbf{L} = \{\overline{\mathtt{S}_0\mathtt{E}_0}, ..., \overline{\mathtt{S}_N\mathtt{E}_N}\}\right) \end{cases} \tag{11.3}$$

$$\begin{cases} no, \quad if\left(\exists\,\mathtt{V}_g : \mathtt{V}_g = \mathtt{X}_o\right), \forall \mathtt{V}_g \in \{\mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3\}, \forall o \in \{0, 1, ..., N\} \\ yes, \quad otherwise \end{cases} \tag{11.4}$$

$$\begin{cases} no, \; if\left(\exists\,\mathtt{V}_g : \mathtt{V}_g = \mathtt{S}_o, \wedge \exists\,\mathtt{V}_h : \mathtt{V}_h = \mathtt{E}_o\right), \forall\,\mathtt{V}_g, \mathtt{V}_h \in \{\mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3\}, \forall o \in \{0, ..., N\} \\ yes, \; otherwise \end{cases} \tag{11.5}$$

The third step is to assess the benefit of inserting the candidate triangle in the global mesh. Benefit is understood as the improvement of the quality or extension of the area of the primitive that is texture

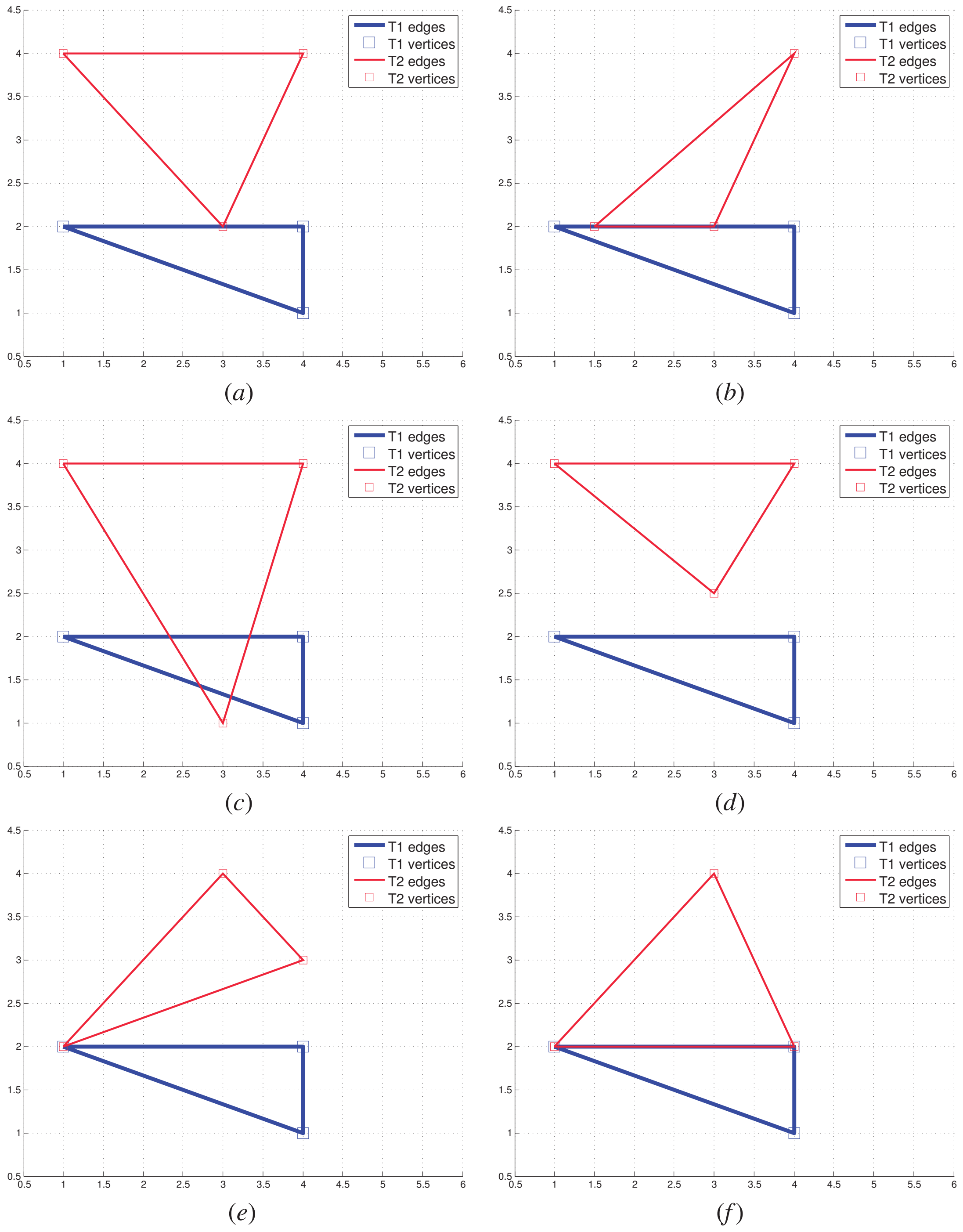


Figure 11.9: The triangles overlap test: (*a*) intersection returns points, overlap true; (*b*) intersection returns line segments, overlap true; (*c*) intersection returns polygons, overlap true; (*d*) intersection returns empty, overlap false; (*e*) intersection returns points, overlap false; (*f*) intersection returns line segments, overlap false;

mapped. When no overlap was detected between the candidate triangle and any of the triangles in the global mesh there is no knowledge about the quality of the mesh at that location. If there is no overlap, then the existing mesh will remain unaffected, in the way that existing triangles will remain unchanged. In this case, the addition of the candidate triangle onto the global mesh is considered beneficial. Although there is no information of the influence this operation will have the quality of the mesh, it does extend the area of the mesh and thus, the area of the primitive that is texture mapped. As a result, the rule is to consider the insertion of the candidate triangle as beneficial, whenever no overlap is detected.

When the candidate triangle overlaps some of the triangles in the existing global mesh, then, the benefit is evaluated using the quality functions. The quality functions were introduced in section 11.2. Here, we will use the notation $\mathbf{q}(\mathtt{T}_i^j)$ to denote the quality of the *ith* candidate triangle of the *jth* projection, and $\mathbf{q}(\mathtt{M}_n)$ to denote the quality of the *nth* global mesh triangle. The specific details of the implementation and configuration of these quality functions will only be addressed in section 11.3.3. For now, it is just necessary to assume that they are designed so that, for every triangle, they return a normalized score. When $\mathbf{q}(\mathtt{T}_i^j) = 1$, the quality is the best possible, and when the $\mathbf{q}(\mathtt{T}_i^j) = 0$, the quality is considered the worst possible. When the quality of the candidate triangle is larger than the quality of all the triangles it overlaps, then it is considered that the mesh will increase its quality and the insertion is considered beneficial.

Let $\mathbf{L} = \{L_0, L_1, ..., L_N\}$ be the list of indices of the triangles from the global mesh $\mathtt{M}$ that are overlapped by the candidate triangle. The overall algorithm used to to assess if the insertion of the candidate triangle $\mathtt{T}_i^j$ is beneficial is shown in eq. (11.6):

$$\begin{cases} \textit{beneficial}, & if\big(\mathbf{L} = \textit{empty set}\big) \\ \textit{otherwise} & \begin{cases} \textit{beneficial}, & if\big(\mathbf{q}(\mathtt{T}_i^j) > \alpha \cdot \mathbf{q}(\mathtt{M}_{L_g})\big),\ \forall g \in \{0, ..., N\} \\ \textit{not beneficial}, & \textit{otherwise} \end{cases} \end{cases} \tag{11.6}$$

where N is the number of existing triangles in the global mesh, and $\alpha \geq 1$ is a cost parameter. As will be shown in the following lines, the insertion of the candidate triangle implies some computational load, especially because the global mesh needs to be restructured. Since there are computational costs involved in restructuring the mesh, the cost parameter is used to control how much better the quality of candidate triangle must be to any other triangle it overlaps, in order for the insertion to be considered beneficial.

Finally, the fourth step concerns the insertion of the candidate triangle. As observed in the previous lines, the insertion of the candidate triangle is only operated if there is no overlap between the candidate triangle and the existing global mesh, or, when there is overlap, if the quality of the candidate triangle is larger than that of the triangles it overlaps.

The global primitive mesh is built as a constrained Delaunay triangulation. Hence, a description of the mesh contains a set of vertices, edges and constraints. The implementation from [Yvinec 2012] is used, since it allows an incremental meshing procedure. In other words, the toolbox contains a description of the current mesh and allows several operations that change the configuration of the mesh. Hence, it is possible to add and remove vertices, as well as add and remove constraints. Note that the unconstrained edges are automatically computed by the algorithm. In fact, this is what makes it a meshing algorithm.

The key objective of an insertion is that the candidate triangle's configuration is preserved on the updated mesh. However, if the candidate triangle does not ovelap any of the existing mesh triangles, then the configuration of the existing mesh should also be preserved. On the other hand, if there are some triangles that overlap the candidate triangle, that means (since an insertion operation has been called) that the candidate triangle's configuration should be preserved but that there is no need to preserve the configuration of the overlapping triangles.

In order to comply with those objectives, several change operations must be done on the existing mesh. We combine these into a set of sequential operations over the mesh that guarantee that the objectives of the insertion are achieved. Figure 11.10 is be used to demonstrate the necessity of the proposed set of operations, by comparing the proposed set of operations with other possibilities. Figure 11.10 (*a*) shows a situation where an existing mesh should be altered with the insertion of a candidate triangle (in red). It is clear that there is overlap between the triangles, so the objective will be to execute a set of operations on the mesh that guarantee that the candidate triangles shape is preserved. In this case, since the triangles overlap, the shape of the existing mesh triangle cannot be preserved. We assume that the existing mesh was built from the insertion of a triangle from a different projection. This implies that the triangle has all three edges constrained (blue squares, in Fig. 11.10 (*a*)). For now, it is not trivial to grasp why this is so. Think of the existing mesh triangles as triangles that resulted from previous insertion operations. Once the complete insertion process is explained, that conclusion will be clear.

Let **insert**$(\mathtt{V},\mathtt{M})$ be a function that inserts vertex $\mathtt{V}$ into mesh $\mathtt{M}$. At first sight, one might think that a simple insertion of all the vertices of the candidate triangle $\mathtt{T}^{j}_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}$ into the global mesh $\mathtt{M}$ would be sufficient:

$$\mathtt{M}^{*} = \textbf{insert}\big(\mathtt{V},\mathtt{M}\big),\ \forall\,\mathtt{V} \in \{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}, \tag{11.7}$$

where $\mathtt{M}^{*}$ is the updated mesh. Figure 11.10 (*b*) shows the updated mesh after the insertion of the three vertices, indices 4, 5 and 6 (see vertices indices in the Fig. 11.10). The updated mesh does not preserve the configuration of the candidate triangle. In other words, there is no triangle with vertices 4,5,6 in the updated mesh. The expression that asserts if the configuration of the candidate triangle is preserved can be stated as follows:

$$\mathtt{T}^{j}_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}\ \textit{is preserved},\ \textit{if}\Big(\exists\,\mathtt{M}^{*}_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}} \in \mathtt{M}^{*} : \mathtt{V}_d = \mathtt{V}_a \wedge \mathtt{V}_e = \mathtt{V}_b \wedge \mathtt{V}_f = \mathtt{V}_c\Big). \tag{11.8}$$

One of the reasons why the simple insertion of the vertices does not work is that the existing mesh had some constrained edges. After the mesh is updated, these constraints continue to exist (see squares on edges 1-2, 2-3, and 1-3 in Fig. 11.10 (*b*)). The configuration of the candidate triangle is not kept because no constraints over the edges of that triangle are set. Hence, the second alternative is to execute an additional operation on top of the insertion of vertices $\mathtt{V}_a$, $\mathtt{V}_b$ and $\mathtt{V}_c$. Let **add_constraint**($\mathtt{e}$, $\mathtt{M}$) be a function that adds a constraint on edge $\mathtt{e}$. The operation can be expressed as follows:

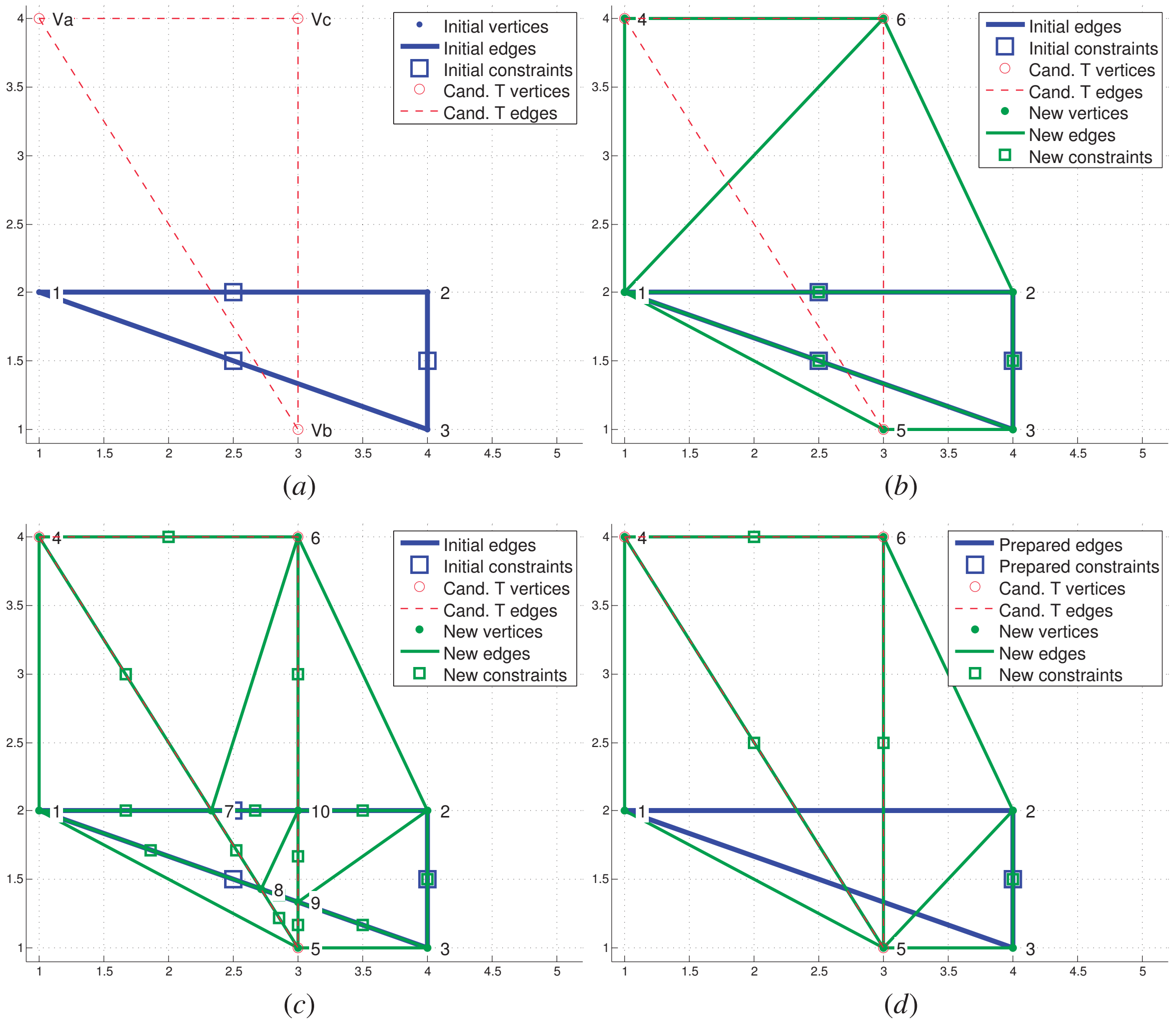


Figure 11.10: The insertion operation: (*a*) candidate triangle and initial mesh; (*b*) insertion of candidate triangle's vertices; (*c*) insertion of the candidate triangle's vertices and constraints; (*d*) preparation of the mesh followed by the insertion of the candidate triangle's vertices and constraints.

$$\mathtt{M}^* = \textbf{add\_constraint}\big(\mathtt{e},\mathtt{M}\big),\ \forall\,\mathtt{e} \in \{\mathtt{V}_a\text{-}\mathtt{V}_b, \mathtt{V}_b\text{-}\mathtt{V}_c, \mathtt{V}_a\text{-}\mathtt{V}_c\}. \tag{11.9}$$

Figure 11.10 (*c*) shows the updated mesh after this procedure is executed. Also in this case the configuration of the candidate triangle is not preserved. The reason is that there are conflicting constraints inserted in the mesh. For example, initially, the global mesh had a constraint over edge 1-2 (see indices in Fig. 11.10). At the same time the constraint $\mathtt{V}_a$-$\mathtt{V}_b$ is inserted into the mesh. Since these two constraints intersect, a new vertex is created at the intersection point (vertex 7). Since a vertex is created at the intersection of the two initial constrained edges, four new edges are created (edges 4-7, 7-8, 1-7 and 7-10). All of these edges are constrained. From Fig. 11.10 (*c*), one can see that the overal result of this approach is that neither the candidate triangle nor the existing mesh is preserved. The reason is that contradictory (intersecting) constraints are inserted in the mesh.

The proposed approach is to execute a preparation of the mesh before the new vertices and constraints are inserted. The problem in the previous alternative was that there were intersecting constraints. Since the key objective is to preserve the configuration of the candidate triangle, it is feasible to assume that the constrained edges of the candidate triangle should impose over the constraints of the existing mesh. The solution is to compute the intersection between constrained edges and to remove the constraints from the global mesh, before inserting the vertices and constraints of the candidate triangle. This is referred to as the mesh preparation stage. After this stage is complete, the global primitive mesh is referred to as prepared global primitive mesh. Let $\mathbf{E} = \{\mathtt{e}_0, \mathtt{e}_1, ..., \mathtt{e}_N,\}$ be the list of the global mesh constrained edges that intersect any of the candidate triangle's edges, and **remove_constraint**$(\mathtt{e},\mathtt{M})$ a function that removes the constraint from edge $\mathtt{e}$ in the mesh $\mathtt{M}$. The prepared mesh $\mathtt{M}'$ is obtained as follows:

$$\mathtt{M}' = \textbf{remove\_constraint}\big(\mathtt{e},\mathtt{M}\big),\ \forall\,\mathtt{e} \in \mathbf{E}, \tag{11.10}$$

and the second stage of the proposed approach is to execute the operations described in eqs. (11.7) and (11.9) on the prepared mesh.

Fig. 11.10 (*d*) shows the results of this approach. The mesh preparation stage detected the following intersections (indices in Figs. 11.10 (*a*) and (*d*)): $\mathtt{V}_a$-$\mathtt{V}_b$ intersects with $\mathtt{V}_1$-$\mathtt{V}_2$, $\mathtt{V}_a$-$\mathtt{V}_b$ intersects with $\mathtt{V}_1$-$\mathtt{V}_3$, $\mathtt{V}_b$-$\mathtt{V}_c$ intersects with $\mathtt{V}_1$-$\mathtt{V}_2$ and $\mathtt{V}_b$-$\mathtt{V}_c$ intersects with $\mathtt{V}_1$-$\mathtt{V}_3$. As a result, the constraints of edges $\mathtt{V}_1$-$\mathtt{V}_2$ and $\mathtt{V}_2$-$\mathtt{V}_3$ are removed. Note that in Fig. 11.10 (*d*), the prepared mesh (not the initial global mesh) is shown in blue, and those constraints no longer appear. More important, the candidate triangle's configuration is preserved (triangle 4-5-6). In this particular case, the initial configuration of the mesh is lost, since there was overlap between the candidate triangle and the initial mesh triangle.

The proposed procedure for the insertion of triangles is capable of preserving the configuration of candidate triangles. In the following sections, several examples are given that show the flexibility of the proposed approach in more complex meshes.

11.3.2 Sequential Updating of Complex Meshes

Section 11.3.1 described the proposed mechanism for the insertion of candidate triangles in an existing mesh. It was also explained that candidate triangles are only inserted into the mesh if they do not overlap any triangle with a higher quality. This section will show some more complex examples of the overall mesh refinement algorithm in action.

Suppose an initial mesh $\mathtt{M}$, created by previous projections, that contains seven vertices and six triangles. It is referred to as mesh at iteration $j = 0$. Now suppose there are three new projections available to map to the initial mesh $\mathbf{C}^{j=1}$, $\mathbf{C}^{j=2}$, $\mathbf{C}^{j=3}$. Here, the super indexes j could be mapped to any $\{k, l, t\}$ projection tuple, provided each tuple is unique, i.e., these are different projections. Each projection contains a single triangle to map to the global mesh. Triangles $\mathtt{T}^1_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}$, $\mathtt{T}^2_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}$ and $\mathtt{T}^3_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}$, correspond to projections $\mathbf{C}^{j=1}$, $\mathbf{C}^{j=2}$, $\mathbf{C}^{j=3}$, respectively. Initially, candidate triangle $\mathtt{T}^1_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}$ is mapped to the initial global mesh $\mathtt{M}$, which results in a new mesh $\mathtt{M}^*$. Then, the second candidate triangle $\mathtt{T}^2_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}$ is inserted into the new mesh $\mathtt{M}^*$, resulting in the updated mesh $\mathtt{M}^{**}$. The process is repeated for the remaining candidate triangles.

In this example the quality of each triangle, which will be referred to as $\mathbf{q}$, is:

$$\mathbf{q}\big(\mathtt{M}_n\big) < \mathbf{q}\big(\mathtt{T}^1_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}\big) < \mathbf{q}\big(\mathtt{T}^2_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}\big) < \mathbf{q}\big(\mathtt{T}^3_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}\big) \, \forall \mathtt{M}_n \in \mathtt{M}, \tag{11.11}$$

and the mesh update cost parameter is $\alpha = 1$, which means that there is no cost associated to the updating of the mesh (see eq. (11.6)). In other words, all three candidate triangles are selected beneficial for insertion since that, even if they overlap, they are sure to have better quality than any other already existing triangles. The initial mesh is shown in Fig. 11.11 (*a*), along with the three candidate triangles.

Figure 11.11 (*b*) shows the mesh after the insertion of the first candidate triangle, i.e., $\mathtt{M}^*$. Since there is no overlap (according to the definition of overlap in section 11.3.1), the candidate triangle is added to the mesh ($\mathtt{M}^*_{\{4,5,8\}}$), and edges $\mathtt{M}^*_{\{4\text{-}5\}}$, $\mathtt{M}^*_{\{5\text{-}8\}}$, and $\mathtt{M}^*_{\{4\text{-}8\}}$ are constrained. Also, since there was no overlap detected, the initial configuration of the mesh is preserved.

The second insertion is shown in Fig. 11.11 (*c*). In this case, there is overlap between candidate triangle $\mathtt{T}^2_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}$ and triangle $\mathtt{M}^*_{\{2,5,7\}}$ (seen in Fig. 11.11 (*b*)). From eq. (11.11), it is possible to conclude that:

$$\mathbf{q}\big(\mathtt{M}_{\{2,5,7\}}\big) < \mathbf{q}\big(\mathtt{T}^2_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}\big), \tag{11.12}$$

and from the observation of Figs. 11.11 (*a*) and (*b*), it is also clear that:

$$\mathtt{M}_{\{2,5,7\}} = \mathtt{M}^*_{\{2,5,7\}}, \tag{11.13}$$

and thus, that:

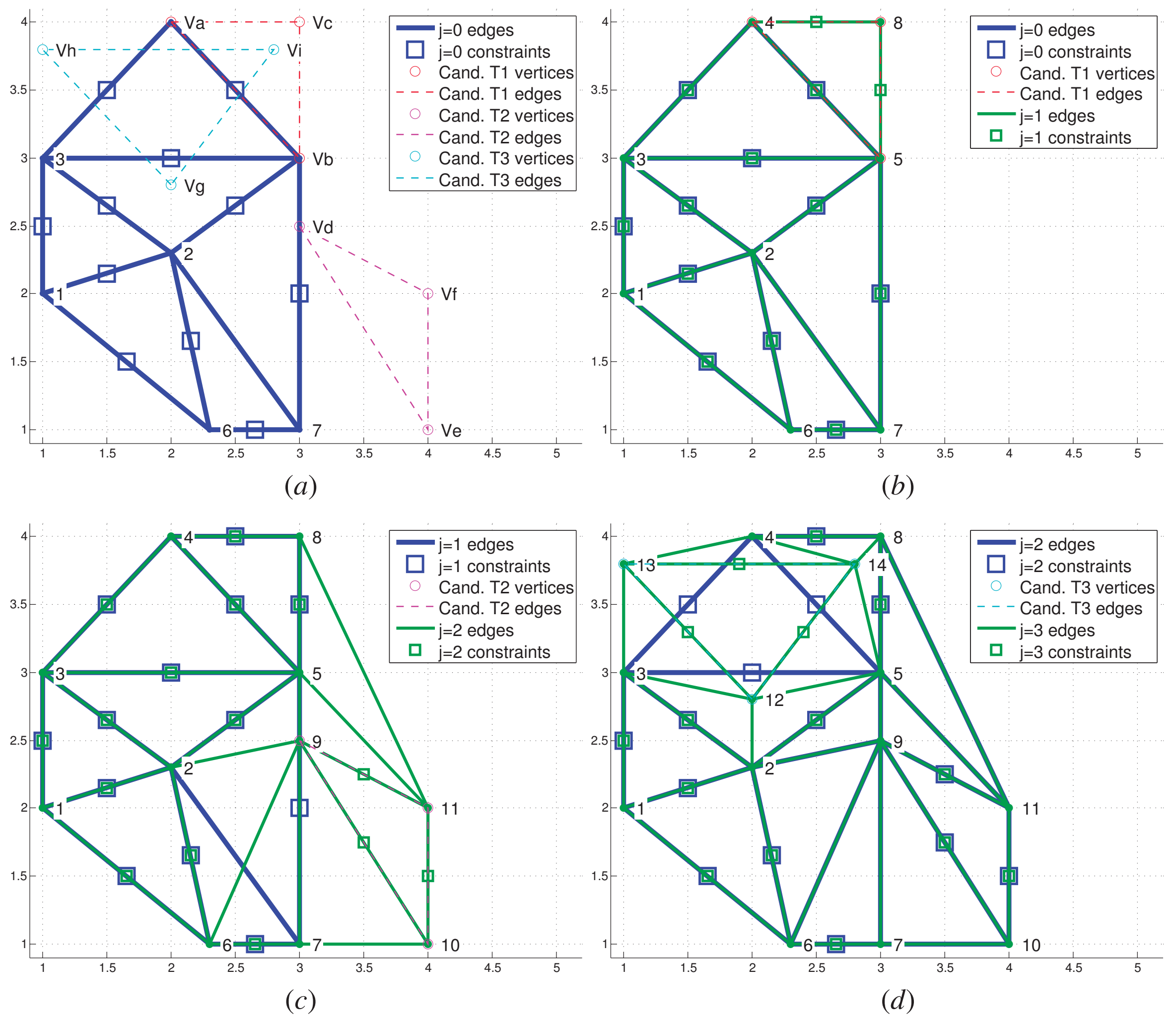


Figure 11.11: The insertion operation, example 1: (*a*) candidate triangles and initial mesh $\mathtt{M}$; (*b*) first insertion, mesh $\mathtt{M}^{*}$; (*c*) second insertion, mesh $\mathtt{M}^{**}$; (*d*) third insertion, mesh $\mathtt{M}^{***}$.

$$\mathbf{q}\left(\mathtt{M}_{\{2,5,7\}}\right) = \mathbf{q}\left(\mathtt{M}^{*}_{\{2,5,7\}}\right), \tag{11.14}$$

which results in the following:

$$\mathbf{q}\left(\mathtt{M}^{*}_{\{2,5,7\}}\right) < \mathbf{q}\left(\mathtt{T}^{2}_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}\right). \tag{11.15}$$

Since the quality of the candidate triangle is higher than the quality of the overlapping triangle, $\mathtt{T}^{2}_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}$ should be inserted to the mesh. In terms of intersections, an intersection between edges $\mathtt{V}_d$-$\mathtt{V}_e$ and edge $\mathtt{M}^{*}_{\{5\text{-}7\}}$ (seen in Fig. 11.11 (*b*)), is detected. As a result, the constraint from edge

$\mathtt{M}^{*}_{\{5\text{-}7\}}$ is removed. The insertion results in a new triangle $\mathtt{M}^{**}_{\{9,10,11\}}$. Note also that the overlapping triangle $\mathtt{M}^{*}_{\{2,5,7\}}$ was not preserved, i.e., it does not exist in the new mesh $\mathtt{M}^{**}$.

Finally, the third insertion detects that triangle $\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}$ overlaps triangles $\mathtt{M}^{**}_{\{3,4,5\}}$, $\mathtt{M}^{**}_{\{4,5,8\}}$ and $\mathtt{M}^{**}_{\{2,3,5\}}$. With regards to triangles $\mathtt{M}^{**}_{\{3,4,5\}}$ and $\mathtt{M}^{**}_{\{2,3,5\}}$, since they existed in the initial mesh, a similar deduction to that presented in eqs. (11.12) to (11.14) may be applied, which leads to:

$$\mathbf{q}\big(\mathtt{M}^{**}_{\{3,4,5\}}\big) < \mathbf{q}\big(\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}\big), \tag{11.16}$$

and to:

$$\mathbf{q}\big(\mathtt{M}^{**}_{\{2,3,5\}}\big) < \mathbf{q}\big(\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}\big). \tag{11.17}$$

With regards to triangle $\mathtt{M}^{**}_{\{4,5,8\}}$, from the observation of Fig 11.11 it is possible to conclude that:

$$\mathtt{M}^{**}_{\{4,5,8\}} = \mathtt{T}^{1}_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}, \tag{11.18}$$

and thus, that:

$$\mathbf{q}\big(\mathtt{M}^{**}_{\{4,5,8\}}\big) = \mathbf{q}\big(\mathtt{T}^{1}_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}\big), \tag{11.19}$$

which, based on eq. (11.11), results in:

$$\mathbf{q}\big(\mathtt{M}^{**}_{\{4,5,8\}}\big) < \mathbf{q}\big(\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}\big). \tag{11.20}$$

Hence, the candidate triangle $\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}$ has a larger quality than all the triangles it overlaps. This is why this candidate triangle is also marked for insertion. Edges $\mathtt{M}^{**}_{\{3\text{-}4\}}$, $\mathtt{M}^{**}_{\{4\text{-}5\}}$ and $\mathtt{M}^{**}_{\{3\text{-}5\}}$ intersect the edges of $\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}$ which is why their constraints are removed (actually, in this case they disappear after the candidate triangle is inserted).

The insertion of candidate triangles sometimes creates not only the candidate triangle itself, but also some additional triangles on the mesh. For example, in Fig. 11.11 (*c*), the insertion of $\mathtt{T}^{2}_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}$ causes the removal of triangle $\mathtt{M}^{*}_{\{2,5,7\}}$, and the addition of a triangle $\mathtt{M}^{**}_{\{9,10,11\}}$, which is a copy of the candidate triangle. However, there are also other triangles that did not existed previously on the mesh $\mathtt{M}^{*}$ but occur in mesh $\mathtt{M}^{**}$. It is the cases of triangles $\mathtt{M}^{**}_{\{2,5,9\}}$, $\mathtt{M}^{**}_{\{5,9,11\}}$, $\mathtt{M}^{**}_{\{5,8,11\}}$, $\mathtt{M}^{**}_{\{2,6,9\}}$, $\mathtt{M}^{**}_{\{2,6,9\}}$, $\mathtt{M}^{**}_{\{6,7,9\}}$ and $\mathtt{M}^{**}_{\{7,9,10\}}$.

In section, 11.2, several alternatives to the implementation of projection quality functions are discussed. These functions provide the quality of a triangle, using as input the projection that generated it. In other words, the input for computing the quality associated with each triangle is based on the $\{k, l, t\}$, primitive, camera, time, tuple. The problem is that meshes have two types of triangles. Some have a parent projection, for example triangle $\mathtt{M}^{**}_{\{9,10,11\}} = \mathtt{T}^{2}_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}$ has as parent projection $\mathbf{C}^{j=2}$. But there are some others that are created automatically and hence have no projection associated to them. It is the case, for example, of triangle $\mathtt{M}^{**}_{\{7,9,10\}}$. We refer to this type of triangles as orphan triangles, meaning they have no parent projection. The problem is that, since the quality functions require as input the $\{k, l, t\}$ of the projection, they cannot be computed for orphan triangles.

Figure 11.12 shows the same sequence of insertion as in Fig. 11.11. The status of the triangles with respect to their parent projection is shown. Initially, it is assumed that all the triangles in the mesh correspond to projection 0, i.e., to $\mathbf{C}^{j=0}$. This is why in Fig. 11.12 (*a*) all triangles are marked in blue. After the first insertion, Fig. 11.12 (*b*), since there was no overlap, the existing triangles

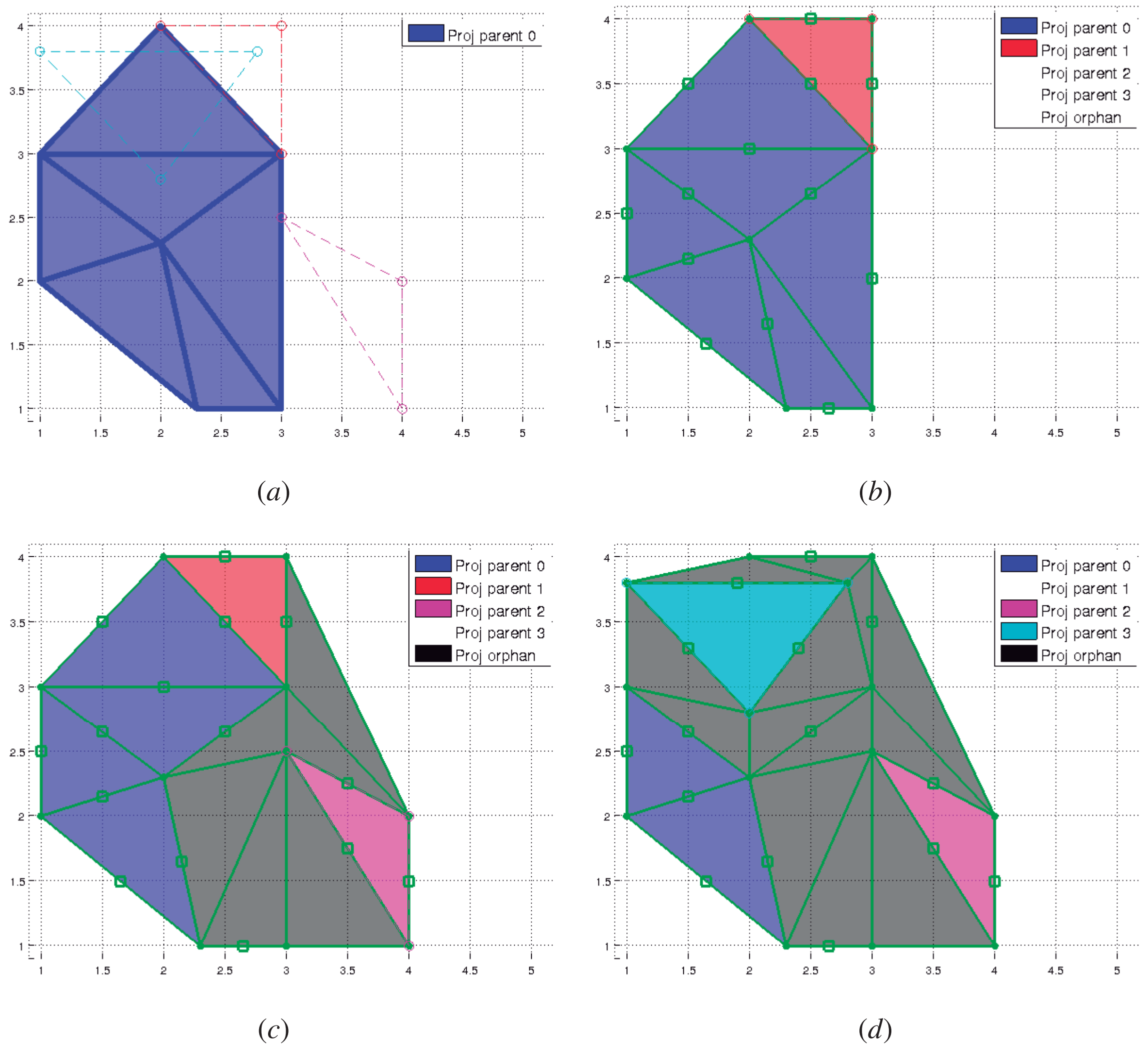
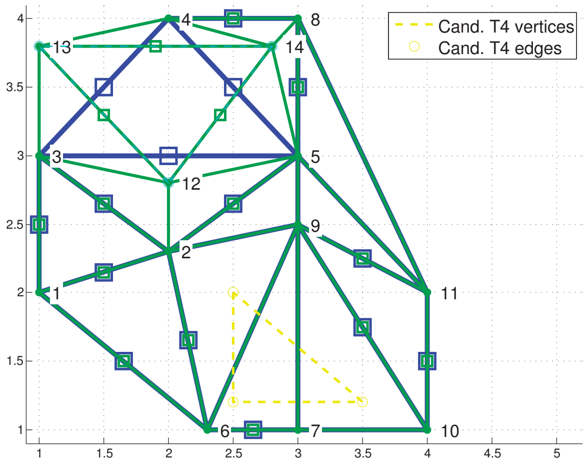


Figure 11.12: The projection parent status of each triangle (same example as in Fig. 11.11): (*a*) candidate triangles and initial mesh $\mathtt{M}$; (*b*) first insertion, mesh $\mathtt{M}^{*}$; (*c*) second insertion, mesh $\mathtt{M}^{**}$; (*d*) third insertion, mesh $\mathtt{M}^{***}$.

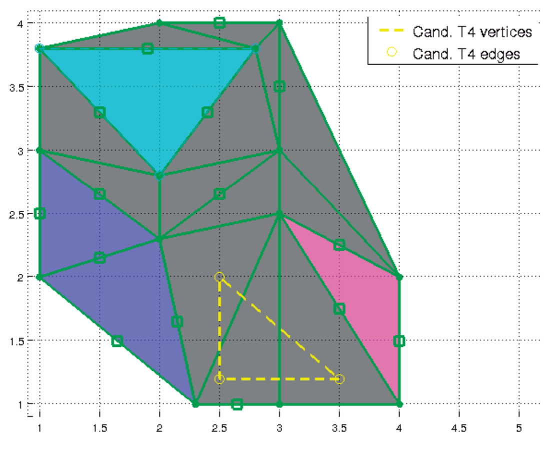
are not altered, which is why they continue to belong to $\mathbf{C}^{j=0}$. The inserted triangles are marked to belong to projection $\mathbf{C}^{j=1}$. The second insertion, Fig. 11.12 (*c*), destroys some triangles from projection $\mathbf{C}^{j=0}$ and creates some new ones. All these triangles are left orphan of projection. The third insertion removes not only some triangles from projection $\mathbf{C}^{j=0}$, but also destroys the triangle from projection $\mathbf{C}^{j=1}$.

As seen, the sequential insertion of triangles can cause some of the triangles in the mesh to be left without a projection parent. Suppose that now that fourth insertion is to be made over mesh $\mathtt{M}^{***}$, and that the fourth candidate triangle $\mathtt{T}^4$ is positioned so that it overlaps triangles $\mathtt{M}^{***}_{\{2,6,9\}}$, $\mathtt{M}^{***}_{\{6,7,9\}}$ and $\mathtt{M}^{***}_{\{7,9,10\}}$. This is shown in Fig. 11.13. Since those overlapping triangles are projection orphans, it is not possible to compute their quality with respect to a projection. Unlike triangles with parent projections, these triangles do not derive directly from a triangle computed in the image space of some projection. In other words, they do not derive from the DDT triangulation executed over an image of some projection. Because of this, there is no guarantee that these orphan triangles are compliant with edges in the projection images.

For this reason, we propose that orphan triangles are set to have the quality -1. Since the quality function that is computed over triangles with parent projections returns values in the interval $[0, 1]$, the effect this has on the algorithm is that orphan triangles are there to fill the gaps left unmapped by real projections, but are considered to have such poor quality that any triangle from any projection will have a higher quality. This mechanism achieves a very interesting effect of filling the holes left by the projections. In Fig. 11.6 these gaps are visible.



(*a*)



(*b*)

Figure 11.13: Insertion over orphan triangles (same example as in Fig. 11.11): (*a*) candidate triangles and initial mesh $\mathtt{M}^{***}$; (*b*) the same as in (*a*) but with the color of each triangle associated to the corresponding parent projection.

In conclusion, the proposed approach for the update of a global primitive mesh consists of a set of procedures that are capable of updating the mesh whenever new, better quality triangles are available for insertion, but at the same time the mechanism is capable of filling the gaps left empty using orphan triangles. Several textured primitives using this algorithm will be shown in section 11.4, where it will be shown that this algorithm generates view pleasant textures, which did not occur in the approaches mentioned in section 11.2.

11.3.3 Projection Quality Functions

Previous sections have described the proposed mechanism used to refine a primitive's global mesh. The mechanism consists of inserting candidate triangles on the existing mesh. A candidate triangle should be inserted into the mesh if its projection quality is higher than the quality of all the triangles it overlaps. In the previous sections, the projection quality of triangles was an abstract concept. In this section, it will be discussed how the projection quality of a triangles can be computed.

The first task is to define the concept of the projection quality of a triangle. One could say that a triangle is more adequate for projection, i.e., has a better projection quality, if the textured primitive that results from mapping the portion of the image contained by the triangle has in turn a good quality. However, the decision on whether a candidate triangle has good quality or not must be made prior to the mapping of that triangle into the global primitive mesh. In other words, the quality of the triangle must be estimated. The information available to each candidate triangle is related to the projection of its parent. A projection as the notation $\mathbf{C}^{\{k,l,t\}}$, where k is the index of the primitive to which the projection's triangles are to be mapped, l is the index of the camera that provided the image, and t is the mission time at which the image was acquired by the camera. Hence, the quality function $\mathbf{q}$ of a triangle $\mathtt{T}^{\{k,l,t\}}$, that belongs to projection $\mathbf{C}^{\{k,l,t\}}$ can be written as:

$$\mathbf{q}\big(\mathtt{T}^{\{k,l,t\}}\big) = \mathbf{f}\big(k, l, t\big). \tag{11.21}$$

The interesting point is that under the proposed mechanism for the management of the update of global primitive meshes is very flexible. Several criteria may be proposed, according to the requirements of the mission or the desired characteristics of the representation.

This work is done under the assumption that the environment around the vehicle is geometrically static. This means that the geometric shape of the objects does not change over time, nor there are moving obstacles in the environment (or, if there are, they can be filtered a priori, see chapter 8 for a discussion on this topic). However, a geometrically static environment may not be photometrically static. Immediately, the iconic pictures of the Times Square, New York City, come to mind (Fig. 11.14). The electronic advertising panels, although in the same position and with the same geometry, change their photometric appearance periodically. In such a case, one could design a quality function that accounted only for the time variable of the projection, giving preference (higher quality) to

Figure 11.14: Times Square: example of a photometrically non static environment.

projection images that are acquired more recently.

$$\mathbf{q}\big(\mathtt{T}^{\{k,l,t\}}\big) = \frac{\Delta t_{max} - \mathbf{min}\big(t_{current} - t, \Delta t_{max}\big)}{\Delta t_{max}}, \tag{11.22}$$

where t is the time at which the image from projection $\mathbf{C}^{\{k,l,t\}}$ was acquired, $t_{current}$ is the current time, i.e., the time in which the projection is actually being mapped, **min** is a function that return minimum or the two arguments, and Δt_{max} the maximum time window from which all the projections have an equally bad quality value. In other words, Δt_{max} is a saturation value for the maximum time difference allowed to the difference $t_{current} - t$.

Another situation that may occur is that one camera has a better quality than another. Suppose that the vehicle contains only two cameras, camera $l = 0$, and camera $l = 1$, and that for some reason camera $l = 0$ always provides much better quality images when compared to those of camera $l = 1$. One reason could be for example that camera $l = 1$ is not well focused, or that its images have a great deal of flickering. If this is known, a preference can be established before hand, and the quality mapping function can be devised to reflect this:

$$\mathbf{q}\big(\mathtt{T}^{\{k,l,t\}}\big) = \begin{cases} 1, \, if(l = 0) \\ 0, \, if(l = 1) \end{cases} \tag{11.23}$$

The quality function can also be computed from an analysis of the image used for the projection. In Fig. 11.15 (*a*) a given projection contains an image that was taken when the camera was facing a road. The sun is hidden behind the trees, which is why the image does not present shadows or saturated colors. Figure 11.15 (*b*) shows an image from a second projection. Although the camera is the same in both projections, since the vehicle moved a bit to the front, the image from the second projection is directly facing the sun. As a consequence, the image shown in Fig. 11.12 (*b*) contains

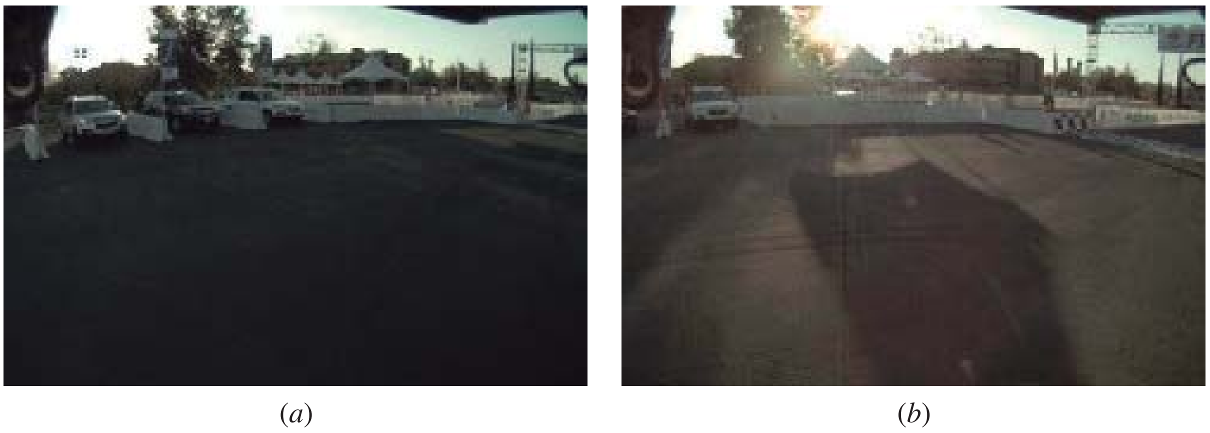


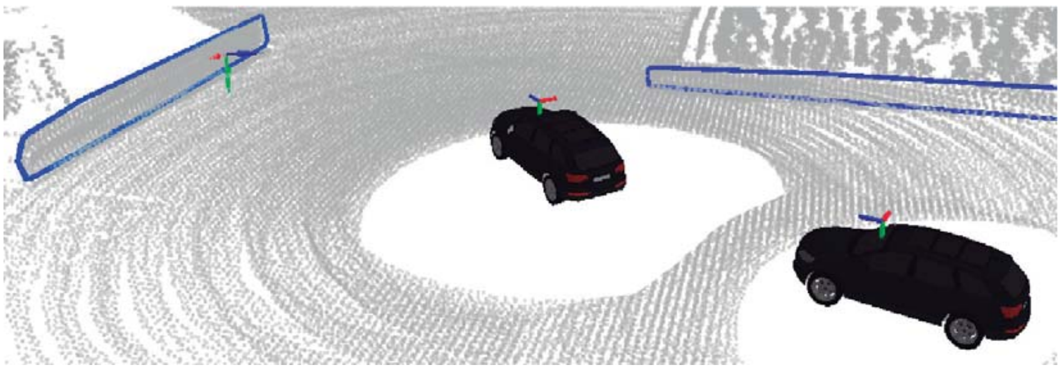
Figure 11.15: Example of images from two different projections but from the same camera: (*a*) image with good quality; (*b*) image with bad quality.

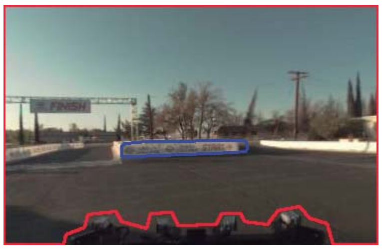
several shadows and saturated regions. The same camera can provide images with very different qualities, in different projections. In this case, the projection quality function could reflect the outcome of an image processing analysis made on the image of projection.

$$\mathbf{q}\big(\mathtt{T}^{\{k,l,t\}}\big) = \textbf{processing}\big(\mathbf{I}^{\{k,l,t\}}\big), \tag{11.24}$$

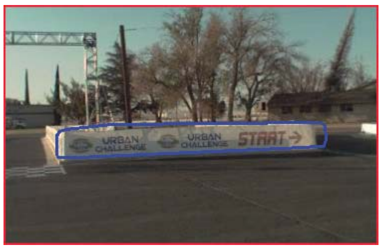
where $\mathbf{I}^{\{k,l,t\}}$ is the image of projection $\{k, l, t\}$, and **processing** is an image processing procedure that would provide a measure of how many shadows there are in the image, or of how saturated its colors are.

Although interesting in theory, the previous approaches could be difficult to implement in practice. The most straight forward solution, and perhaps the most logical, is to assume that a projection has a higher quality whenever it is capable of providing an image of the primitive with a higher resolution. Note that this is not directly related to the size of the triangles computed in image space. Since a DDT triangulation is used, large triangles can exist because they represent smooth color surfaces. But the key is that, if an image from a projection contains a large number of pixels that are a view of the primitive, then the image should have large quality. On the other hand, if the portion of the image that views the primitive has a small number of pixels, the projection quality of the image should not be so good. Figure 11.16 (*a*) shows a scene with four projections over the primitive in front of the vehicle ($k = 4$). The vehicle is traveling from the right to the left of the scene. At time instant t_1 the vehicle is at the position on the right, and captured images with its front center camera, projection $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, and from its front 6 millimeter camera $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$. At instant t_2 the vehicle is on the position on the left, and again captured images using its front center camera, projection $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$, and from its front 6 millimeter camera $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$. The images from projections $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$ are shown in Figs. 11.16 (*b*),

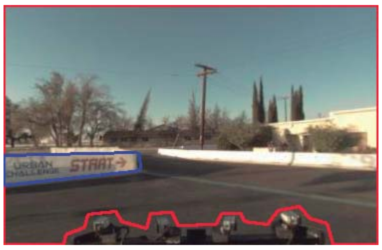

(*a*)



(*b*)



(*c*)



(*d*)

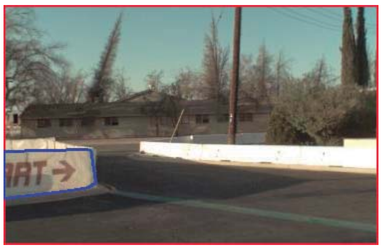

(*e*)

Figure 11.16: Quality functions for multiple projections at different mission times t_1 and t_2: (*a*) a 3D scene with the primitive $k = 4$ (in blue) and the vehicle's position at times t_1 and t_2, respectively from right to left in the figure: (*c*), (*d*), (*e*), (*f*), images from projections $\mathbf{C}^{\{k=4,l=0,t=t_0\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_2\}}$, respectively.

(*c*), (*d*) and (*e*), respectively.

In a first example, let us focus only on the projections generated by the front center camera ($l = 0$). From the analysis of Figs. 11.16 (*b*) and (*d*) it is possible to conclude that the projection $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ seems to have a higher quality when compared to $\mathbf{C}^{\{k=4,l=0,t=t_1\}}$. The reason is that there is a higher resolution view of polygonal primitive. This is caused by the fact that the image of projection $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$ was acquired at time t_2, and that the camera was closer to the primitive $k = 4$ at time t_2, than at time t_1. We propose to evaluate the quality of the projection (and thus of the triangles to which it is parent) by measuring the distance between the position of the camera at the time of projection and the local primitive's coordinate frame. In Fig. 11.16 (*a*), the local coordinate frames of each cameras at times t_1 and t_2 are shown as the small (red-green-blue) reference system on the roof of the vehicle. The local primitive coordinate frame is shown also as a red-green-blue reference system located near the primitive. In section 10.3.3 the different coordinate frames involved in the reconstruction algorithm were presented and discussed. In sum, the there is a coordinate frame for each camera, a coordinate frame for the vehicle, the world, and a local coordinate frame for each primitive. The origin of the coordinate frame of camera l, i.e., a point ${}^{\mathbf{C}^l}p = [0,0,0])$ viewed in the camera l local coordinate frame, can be viewed from perspective of the coordinate frame of primitive k, at time t (${}^{\mathbf{P}^k}p$) is given by:

$$ {}^{\mathbf{P}^k}p = {}^{\mathbf{C}^l}\mathbf{T}_{\mathbf{V}} \cdot {}^{\mathbf{V}}\mathbf{T}^t_{\mathbf{W}} \cdot {}^{\mathbf{W}}\mathbf{T}_{\mathbf{P}} \cdot {}^{\mathbf{C}^l}p, \tag{11.25} $$

and the distance between both coordinate frames is given by the difference between the origin of the primitives local coordinate frame (${}^{\mathbf{P}^k}o$) and ${}^{\mathbf{P}^k}p$. Since the coordinates of (${}^{\mathbf{P}^k}o$), when viewed at the local coordinate system of polygon $\mathbf{P}^k$ are located at the origin, then ${}^{\mathbf{P}^k}o = [0,0,0]$, and the distance (D) between those points is obtained by:

$$ D = \mathbf{dist}\Big({}^{\mathbf{P}^k}p, {}^{\mathbf{P}^k}o\Big) = \|{}^{\mathbf{P}^k}pi\|, \tag{11.26} $$

where **dist** is a function that computes the Euclidean distance between two points. One solution could be to define the quality of the projection based on this distance:

$$ \mathbf{q}\big(\mathrm{T}^{\{k,l,t\}}\big) = 1 - \frac{\mathbf{min}\big(D^{\{k,l,t\}}, D_{max}\big)}{D_{max}}, \tag{11.27} $$

where D_{max} is the maximum distance allowed, a saturation value from which all distances have equal zero quality, and $D^{\{k,l,t\}}$ is obtained for each projection using eq. (11.26).

The quality function presented in (11.26) works for images that have similar focal distance. Since $D^{\{k=4,l=0,t=t_2\}} < D^{\{k=4,l=0,t_1\}}$, the function would return a higher quality for the projection showed in Fig. 11.16 (*d*), when compared to the projection shown in Fig. 11.16 (*b*), which makes sense.

However, if other cameras are involved, especially other cameras which have different focal distances, the solution presented in eq. (11.27) would not work. For example, consider only two projections: $\mathbf{C}^{\{k=4,l=0,t=t_2\}}$, shown in Fig. 11.16 (*d*) and $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$, Fig. 11.16 (*c*). A visual analysis would lead to the conclusion that $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$ has higher quality. However, since that $D^{\{k=4,l=0,t=t_2\}} < D^{\{k=4,l=1,t_1\}}$, the criteria proposed in eq. (11.27) would give the opposite result. This is because of the fact that camera $l = 1$ has a higher focal distance than camera $l = 0$. Thus, we propose to extend eq. (11.27) using also the focal distance of each camera:

$$ \mathbf{q}\big(\mathrm{T}^{\{k,l,t\}}\big) = \bigg(1 - \frac{\mathbf{min}\big(\frac{f^l}{f_{max}} D^{\{k,l,t\}}, D_{max}\big)}{D_{max}}\bigg), \tag{11.28} $$

where f^l is the focal distance of camera l, and f_{max} is the largest focal distance amongst all cameras used for representing the environment.

This section presented several possible solutions for the computation of projection quality functions. The flexibility of the proposed approach was shown by the vast range of possibilities that may be used for the quality functions, and how this choice may influence the obtained representation. In the particular case of the MIT data set, the *Talos* vehicle contains five cameras, and the results that will be presented in section 11.4 use the projection quality functions proposed in (11.28).

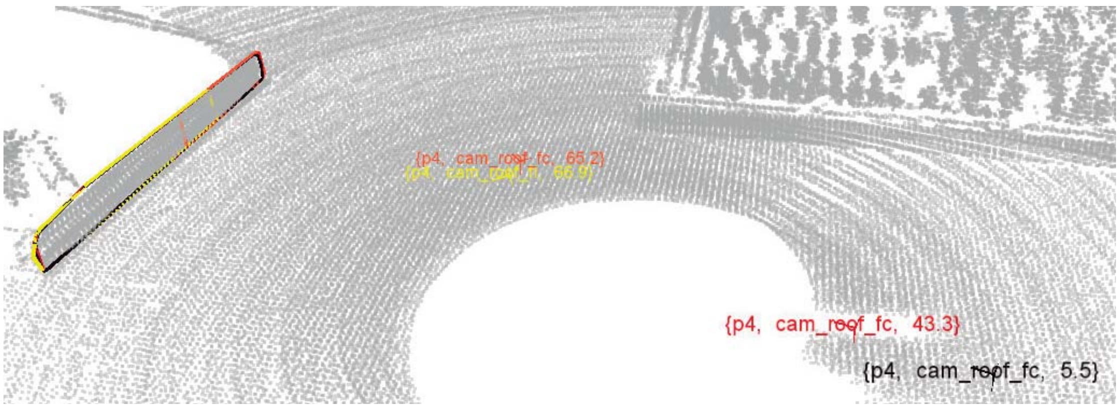
11.4 Results

In this section several results will be shown that prove the concept of the refinements of photometric primitives. Several examples are given, to show the flexibility of the proposed algorithm. As already discussed in section 10.4, there is no ground truth available in the MIT data set. To the best of our knowledge, excluding simulation setups, there are no data sets available that contain geometry as well as texture ground truth. At least, no data sets that are comparable to the MIT data set, both in the amount of data and the variety of sensors. Because of this, it is not possible to present quantitative results. Hence, in the examples that follow, several qualitative results are presented.

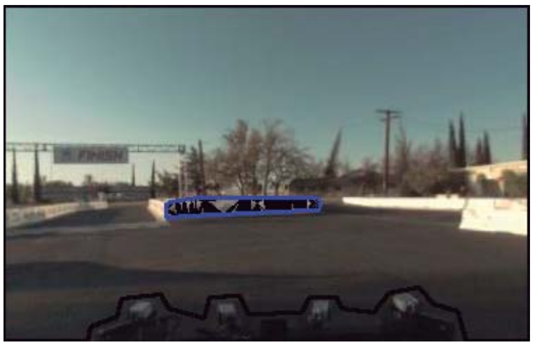
11.4.1 Texture Improvement Over Multiple Projections

The first result is collected from sequence 1 of the MIT data set. The vehicle is approaching a wall panel, which has the word *start* written on it. It is shown in Fig. 11.17. A single primitive, the one representing the wall, is considered in this representation ($k = 4$). Only the cameras front center ($l = 1$) and front left ($l = 3$) are used for generating projections. Projections are generated at three different times, t_1, t_2 and t_3. At time t_1, the front left camera does not see the primitive. Hence, only the projection containing the front center camera image is generated. This is projection $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$, shown in black in Fig. 11.17 (*a*). At time t_2 the vehicle as moved forward a little bit. The front left camera still does not view the primitive so only projection $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$ is generated. It is shown in red in Fig. 11.17 (*a*). At time t_3 the vehicle has moved forward and turned slightly to the right. As a consequence, the front center camera views only a right side portion of the primitive, projection $\mathbf{C}^{\{k=4,l=1,t=t_3\}}$. It is shown in orange in Fig. 11.17 (*a*). Since the vehicle has turned to the right, the front left camera now views a left side portion of the primitive, generating projection $\mathbf{C}^{\{k=4,l=3,t=t_3\}}$. It is shown in yellow in Fig. 11.17 (*a*).

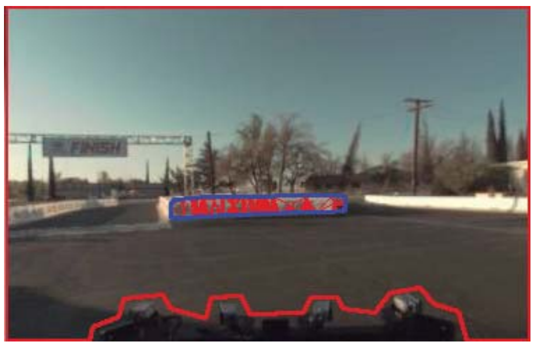
Images from projections $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$, $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$, $\mathbf{C}^{\{k=4,l=1,t=t_3\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_3\}}$ are shown in Figs. 11.17 (*b*), (*c*), (*d*) and (*e*), respectively. For each projection, a local mesh is computed in image space using DDT triangulations. Local triangulated meshes are shown in Figs. 11.17 (*b*), (*c*), (*d*) and (*e*). Note that, for a better visualization of the mechanism, triangulated meshes are purposely configured to generate large triangles. This observation is valid throughout this entire section, whenever triangulated meshes are visualized.



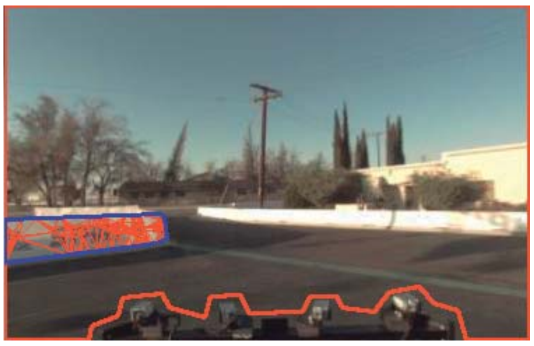
(*a*)



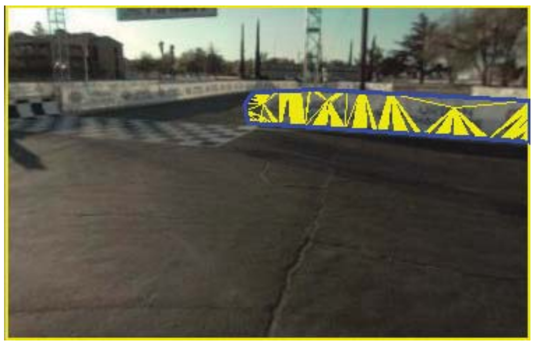
(*b*)



(*c*)



(*d*)



(*e*)

Figure 11.17: Computing a representation for a vertical wall: (*a*) a 3D view; (*b*) camera front center projection at time t_1, black color; (*c*) camera front center projection at time t_2, red color; (*d*) camera front center projection at time t_3, orange color; (*e*) camera front left projection at time t_3, yellow color;

Figure 11.18 shows the evolution of the global triangulated mesh over time. At time t_1, only projection $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$ is available to compute the global mesh. Hence, the global mesh is composed only of triangles with parent projection $\mathbf{C}^{\{k=4,l=front\,center,t=t_1\}}$ and eventually of some orphan triangles (11.18 (*a*)). At time t_2, a new projection $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$ becomes available. The global mesh is then updated using this new information (11.18 (*b*)). Note that since the image of projection $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$ is acquired when the vehicle is closer to the wall (by comparison with $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$), the quality of the triangles of mesh $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$ is better. As a consequence, the global mesh now contains a majority of triangles from $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$. Since two projections are added, let us consider two separate stages at time t_3. The first stage regards the insertion of projection $\mathbf{C}^{\{k=4,l=1,t=t_3\}}$. Since that the image from this projection was acquired closer to the wall primitive, most of the local triangles in $\mathbf{C}^{\{k=4,l=1,t=t_3\}}$ are mapped to the global mesh. However, since only a right side portion of the primitive is seen, the left side of the primitive retains triangles from previous projections (Fig. 11.18 (*c*)). Note that orphan triangles (in blue) are generated to fill the gaps between the triangles with parent projections. The second stage of time t_3 concerns the insertion of projection $\mathbf{C}^{\{k=4,l=3,t=t_4\}}$, which views only the left portion of the wall primitive. Again, since the front left camera ($l = 3$) is very close to the primitive at time $t = t_3$, most of the local mesh triangles are inserted onto the global mesh (Fig. 11.18 (*d*)). Again orphan triangles fill the gaps between projections.

The resulting textured primitive is show in Fig. 11.19. At time t_1, the texture was provided by

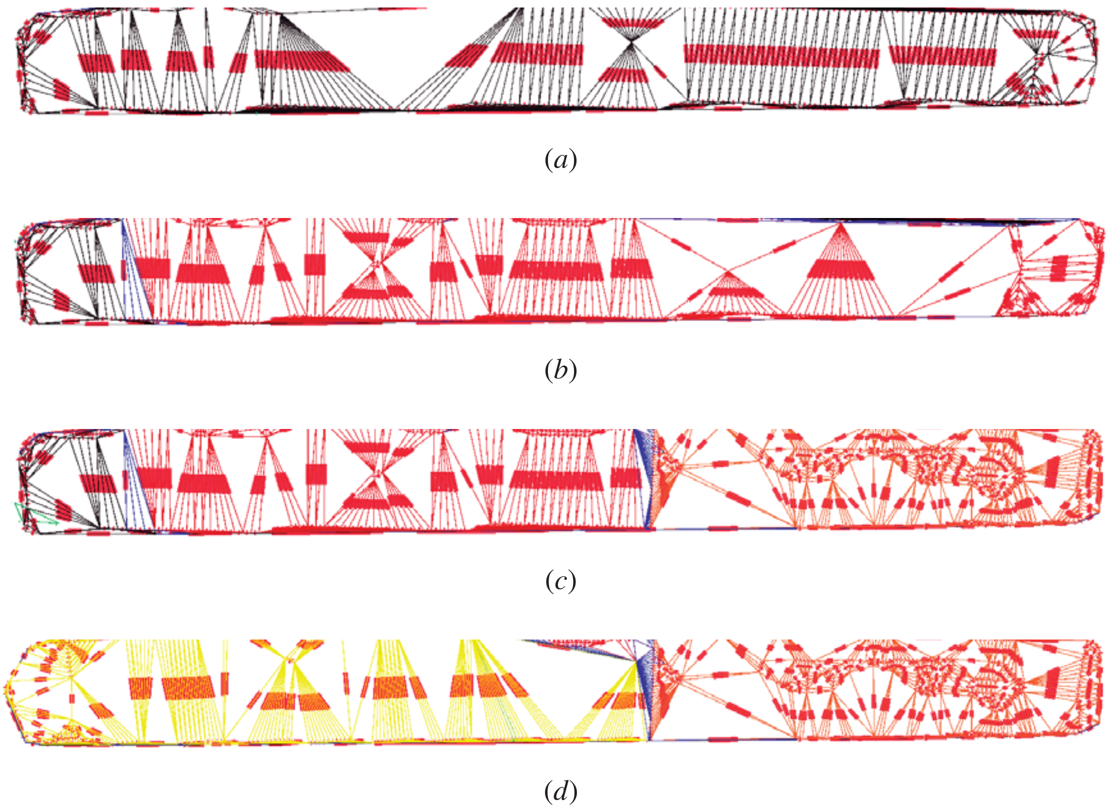


Figure 11.18: The evolution of the global primitive's mesh. (*a*) time $t = t_1$; (*b*) time $t = t_2$; (*c*) time $t = t_3$, insertion of front center camera; (*d*) time $t = t_3$, insertion of front left camera.

an image captured with the camera very far away. As a result, the texture is blurred and with small resolution (Fig. 11.19 (a)). At time t_3, a projection is added where the resolution increases slightly, which is why the texture presented in Fig. 11.19 (b) is less blurred. At time t_3, first stage the right side of the primitive is mapped with a much higher resolution (Fig. 11.19 (c)). Finally, at time $= t_3$, second stage the left side of the primitive is also mapped with high resolution texture. In section 11.2, Fig. 11.6 showed a similar approach to the proposed one. However, in that case, the visual quality of the computed textures where deteriorated by the gaps on the connections of different projections. The orphan triangle mechanism provides a solution for this problem and contrives to the generation of higher quality textures.

The quality functions provide an efficient mechanism that drives the global mesh towards good quality textures, by selecting which triangles from local meshes should be mapped to the global mesh. Suppose for example that the scene presented in Fig. 11.17 had occurred in inverse order. Instead of approaching to the wall primitive, the vehicle would actually be driving backwards moving away from the primitive. In this case, the textured primitive generated at the first instant, when the vehicle was closer to the primitive would be similar to the texture shown in Fig. 11.19 (d), with a global

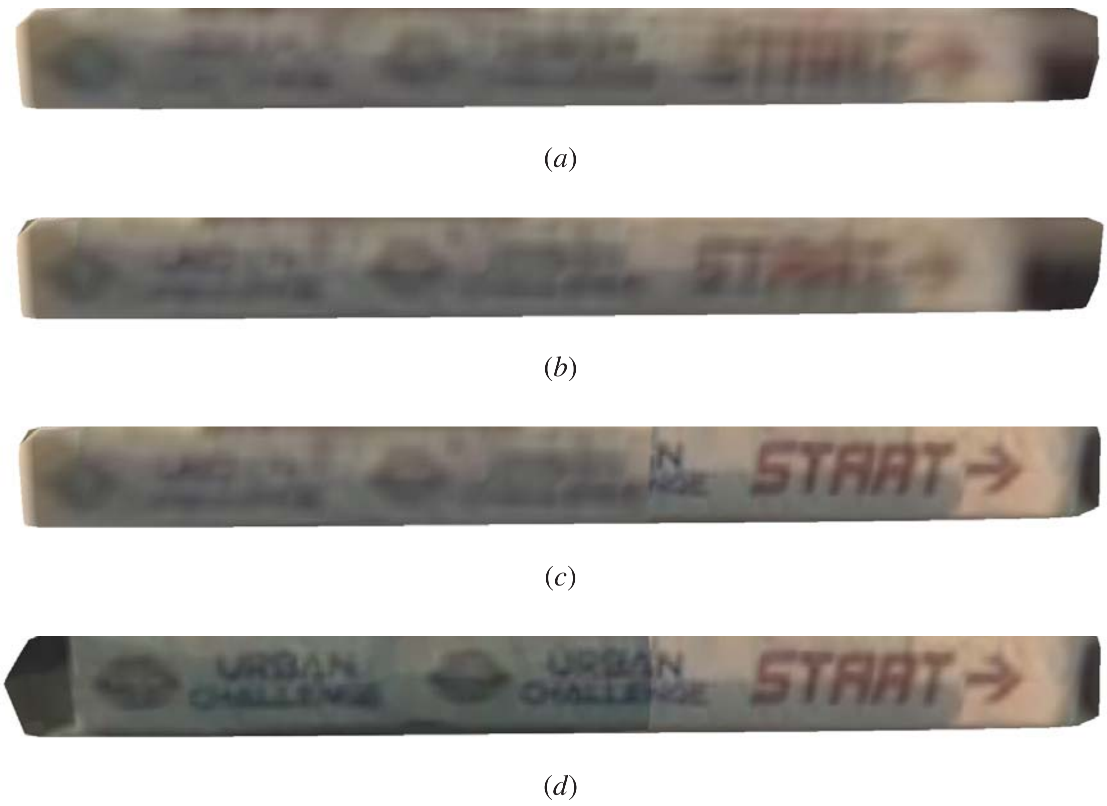


Figure 11.19: The evolution of the global primitive's texture. (a) time $t = t_1$; (b) time $t = t_2$;; (c) time $t = t_3$, insertion of front center camera; (d) time $t = t_3$, insertion of front left camera.

mesh similar to that of Fig. 11.18 (*d*). In subsequent projections, the images would be collected with the camera further away from the primitive, and the local mesh triangles would have less quality. As a consequence, the initial texture would be maintained since no better quality triangles would be provided to refine the global mesh.

11.4.2 Evolution of the Global Primitive Mesh

In the next example, we consider a similar scene to the one presented in Fig. 11.17 (*a*). Throughout the three time instants $t = t_1$, $t = t_2$ and $t = t_3$, the vehicle is moving forward. From t_1 to t_2 the vehicle drives straight, and from t_2 to t_3 the vehicle turns slightly to the right. In this case the primitive that represents the ground plane is used for texture mapping ($k = 0$). As a consequence, there is always a portion of the images from the projections that view the ground. In other words, at all instants any of the cameras view a portion of the ground, since they are pointed downwards. We will consider three different cases, each generating a unique scene representation:

- In the first case only the front center camera ($l = 1$) is used for projection. Hence there will be three projections: $\mathbf{C}^{\{k=0,l=1,t=t_1\}}$, $\mathbf{C}^{\{k=0,l=1,t=t_2\}}$ and $\mathbf{C}^{\{k=0,l=1,t=t_3\}}$.
- In the second case only the rear center camera ($l = 4$) is used for projection. Hence there will be three projections: $\mathbf{C}^{\{k=0,l=4,t=t_1\}}$, $\mathbf{C}^{\{k=0,l=4,t=t_2\}}$ and $\mathbf{C}^{\{k=0,l=4,t=t_3\}}$.
- In the third case only the front left camera ($l = 3$) is used for projection. Hence there will be three projections: $\mathbf{C}^{\{k=0,l=3,t=t_1\}}$, $\mathbf{C}^{\{k=0,l=3,t=t_2\}}$ and $\mathbf{C}^{\{k=0,l=3,t=t_3\}}$.

The final global primitive meshes (those obtained after inserting projections at times t_1,t_2 and t_3) for each case are displayed in Fig. 11.20 (*a*), (*b*) and (*c*), for cases one, two and three, respectively.

Fig. 11.20 (*a*) shows the distribution of triangles according to the parent projection. In this case, the images are provided by the front center camera. This camera, as its name suggests, is facing the front of the vehicle. As the vehicle moves forward, the ground in front of the vehicle that has been previously mapped by other projections is now visible in images at a closer range. The effect this has on the distribution of triangles according to their parent projection is that more recent projections tend to override older projections. This is why in Fig. 11.20 (*a*) the red color (projection at t_2) overrides the black color (projection at t_1), and the yellow color (projection at t_3) overrides any of the previous two.

The second case is shown in 11.20 (*b*). Here, since the camera is facing the rear side of the vehicle, the opposite phenomena occurs: since the vehicle is moving away from the ground behind it, older projections were taken at closer distances. As a consequence, the red color (projection at t_2) overrides the yellow color (projection at t_3), and the black color (projection at t_1) overrides any of the previous.

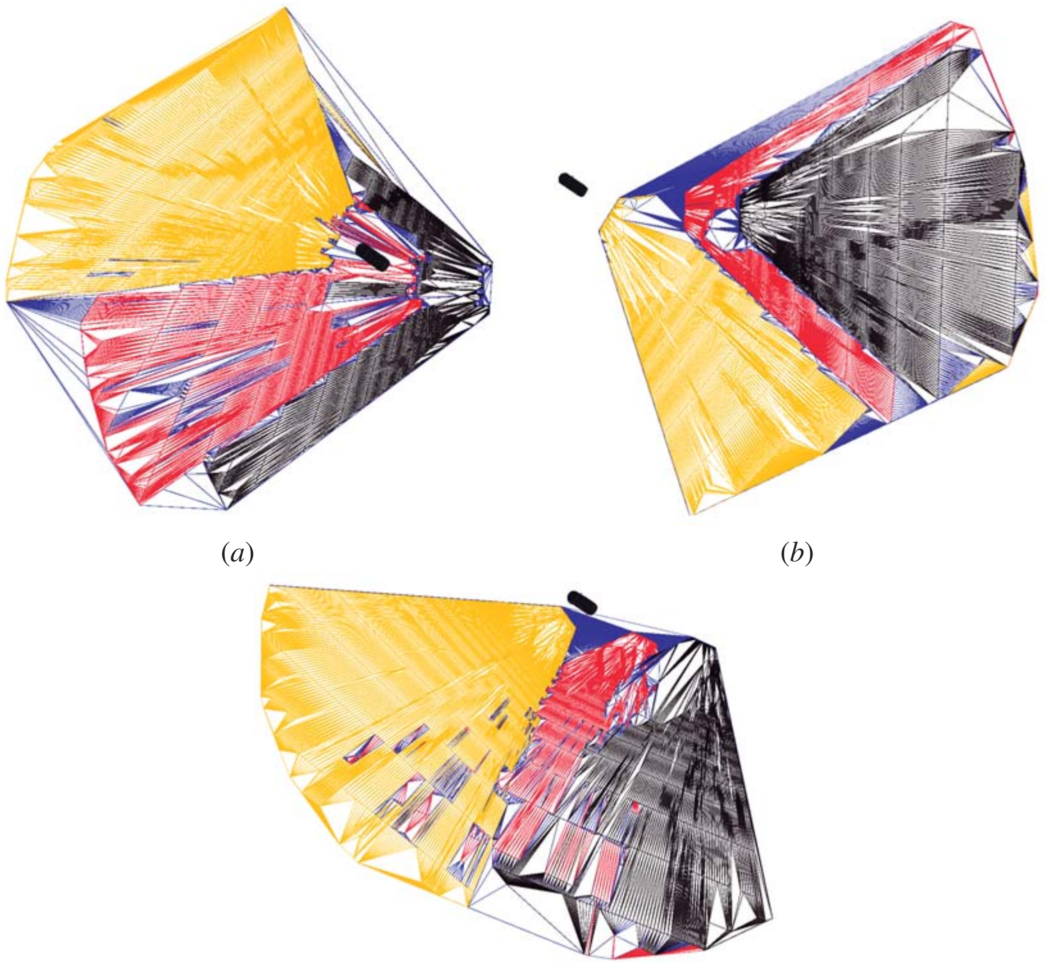

(*a*) (*b*) (*c*)

Figure 11.20: Global primitive meshes for three different cases of mapping of a single camera to the ground plane primitive: (*a*) front center camera; (*b*) rear center camera; (*c*) front left camera; Colors denote the projection instant: t_1 black, t_2 red and t_3 yellow.

Figure 11.20 (*c*) shows the third case. Here, since the camera is facing the left side of the vehicle, a hybrid phenomena takes place. For each projection, there is always a portion of the triangles, i.e., those that map the ground directly in front of the camera at that instant, that have a higher projection quality than any others.

Figure 11.21 shows the images from all projections displayed in Fig. 11.20, as well as the local triangulated meshes. Figure 11.22 shows the percentage of triangles each projection contains in the global primitive mesh, as a function of the mission time.

Figure 11.22 (*a*) shows the results for the front center camera. At time t_1, only triangles from the first projection (black) and orphan triangles (blue) exist. Then, at time t_2, the triangles from the second projection (red) are added to the global mesh. As a consequence, the percentage of triangles

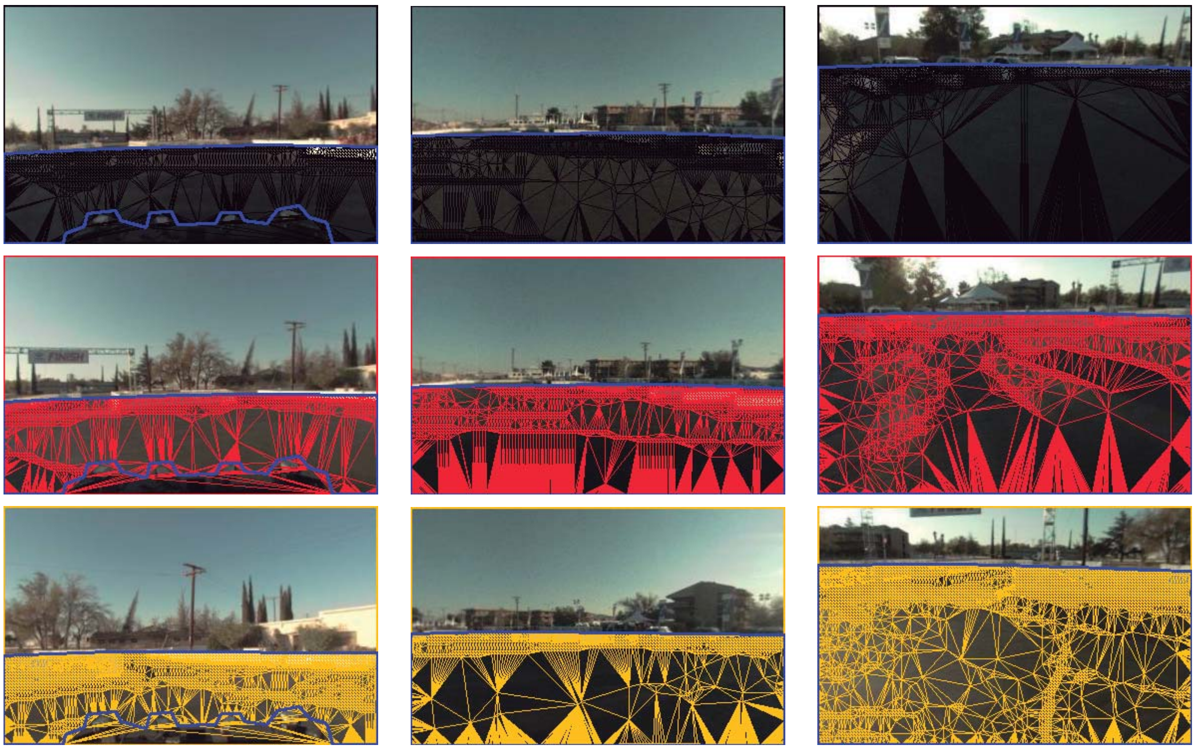


Figure 11.21: Images and local triangulated meshes for all projections shown in Fig. 11.20: Left column, front center camera (Fig. 11.20 (*a*)); Middle column, rear center camera (Fig. 11.20 (*b*)); Right column, front left camera (Fig. 11.20 (*c*)); First row: time t_1; Second row: time t_2; Third row: time t_3;

from the first projection (black) decreases dramatically. At time t_3, the third projection again takes the major slice of percentage and the previous two projections decrease. This results were expected behaviour from the analysis of Fig. 11.20 (*a*). In front facing cameras (when the vehicle is moving to the front), more recent projections tend to contribute with a larger portion of the total triangles in the global mesh.

In the case of rear facing cameras (again, when the vehicle is moving towards the front), less recent cameras will tend to contrive the majority of the triangles in the global mesh. This is observable in Fig. 11.22 (*b*), where the first projection (black) is, at all times, the one with the largest percentage of triangles. Whenever a new projection is added (second projection in red at time t_2 or third projection in yellow at time t_3) it always gets a smaller portion than any of the previous projections.

In the case of the front left facing cameras, Fig. 11.22 (*c*), there is a hybrid phenomenon as discussed before. The percentage of projections tends to be the same for all projections, which is why the second projection (red) when first mapped at time t_2 achieves approximately the same percentage of triangles as the first projection (black). They continue to have similar percentages also at time t_3. At time t_3, the third projection (yellow) obtained a higher value of percentage because the vehicle as turned slightly to the right and the left camera faced an area of the ground that was not previously mapped by any of the previous projections.

The previous example has shown that the proposed quality functions are capable of handling multiple projections and accurately decide which are the best quality projections to map to the global mesh. However, that example was simplified since only one camera was considered to provide projections in each case. Given that the *Talos* vehicle contains five cameras, how do they map over the ground primitive. In this example, the five cameras onboard the *Talos* are considered. The ground polygon is used for texture mapping. The same sequence from the previous example is used: the vehicle is moving forward and three time instants are used to generate projections. Each time instant t_1, t_2 and t_3 generates five projections, one for each camera. Figure 11.23 (a) shows the state of the

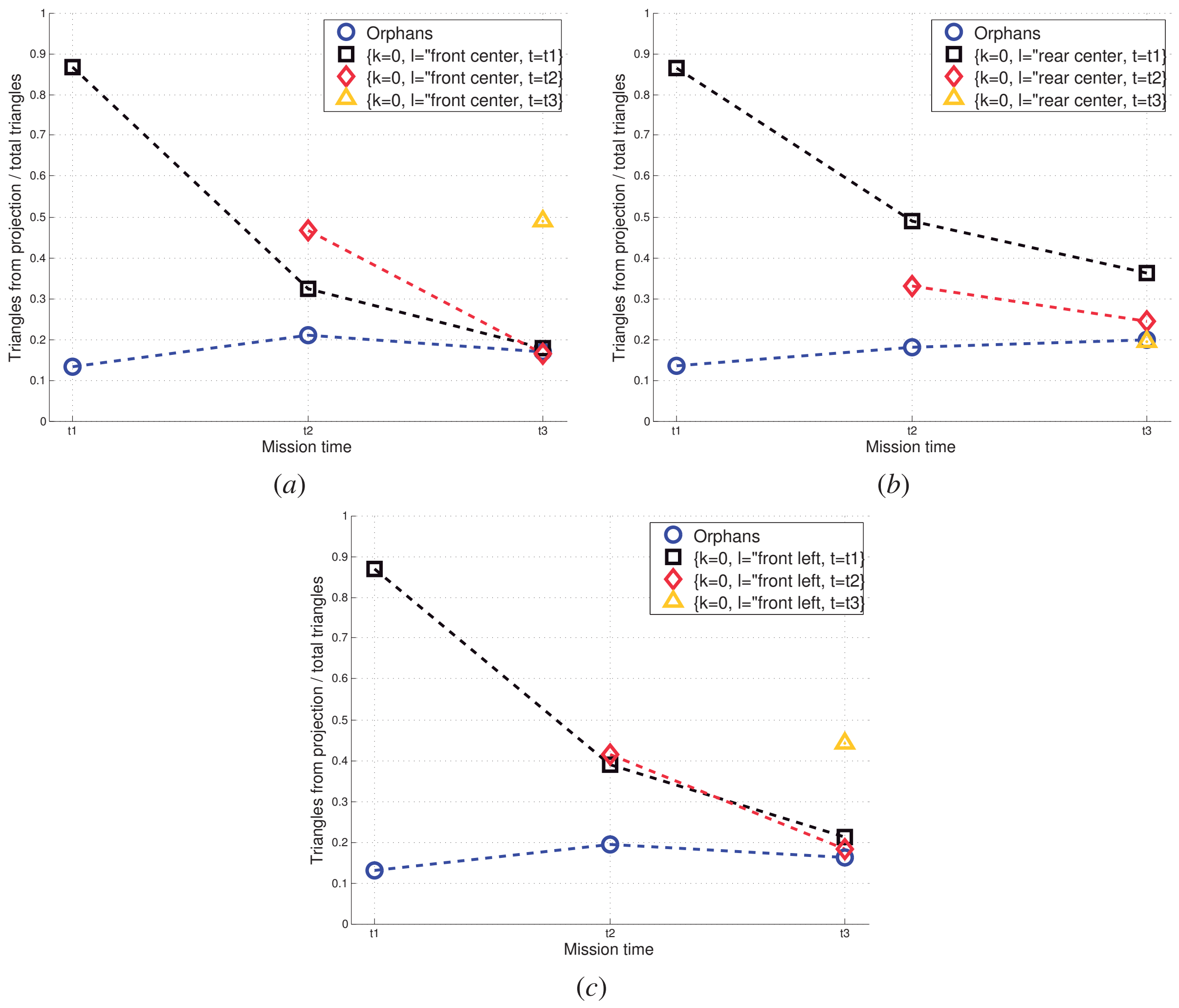


Figure 11.22: Number of triangles from each projection over the total number of triangles in the global mesh as a function of mission time, for the examples shown in Fig. 11.20: (a) front center camera; (b) rear center camera; (c) front left camera;

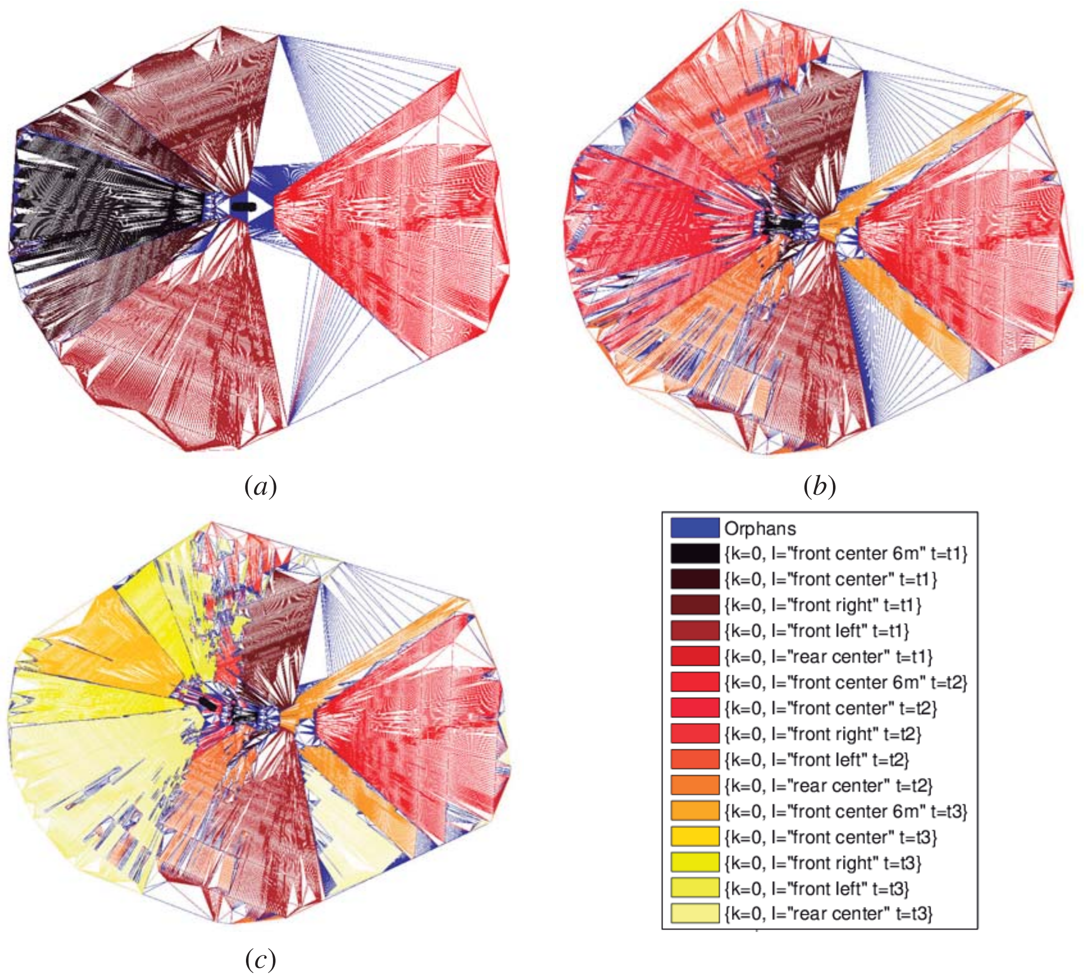

(a)
(b)
(c)
Orphans
{k=0, l="front center 6m" t=t1}
{k=0, l="front center" t=t1}
{k=0, l="front right" t=t1}
{k=0, l="front left" t=t1}
{k=0, l="rear center" t=t1}
{k=0, l="front center 6m" t=t2}
{k=0, l="front center" t=t2}
{k=0, l="front right" t=t2}
{k=0, l="front left" t=t2}
{k=0, l="rear center" t=t2}
{k=0, l="front center 6m" t=t3}
{k=0, l="front center" t=t3}
{k=0, l="front right" t=t3}
{k=0, l="front left" t=t3}
{k=0, l="rear center" t=t3}

Figure 11.23: Distribution of triangles according to projection. Rear center camera only: (*a*) a 3D view; (*b*) projection at time t_1; (*c*) projection at time t_2; (*d*) projection at time t_3

global mesh after time t_1. Five projections are contained in the mesh. At time t_2, the global mesh incorporates many of the projections that are computed at this time (Fig. 11.23 (*b*)). The same occurs at time t_3 (Fig. 11.23 (*c*)). The resulting mesh is an intricate mosaic of triangles coming from several projections.

Figure 11.24 shows the fifteen images used to compute these representations. An interesting observation is that at time t_1, the area of projection from the rear center camera was not connected to the areas of projection of the other cameras. Note that the red triangles in Fig. 11.23 (*a*) are not connected to any triangle with a parent projection, only to orphan triangles. This unmapped region is understood if one thinks of the ground that is bellow the vehicle. There is no coverage from the cameras of this region, which is why only orphan triangles (blue) occupy it. At time t_2, the vehicle has moved in front, and the ground that was bellow it at time t_1 is now visible from the rear center camera. Hence, the areas mapped by the rear cameras connect to the areas mapped by the other

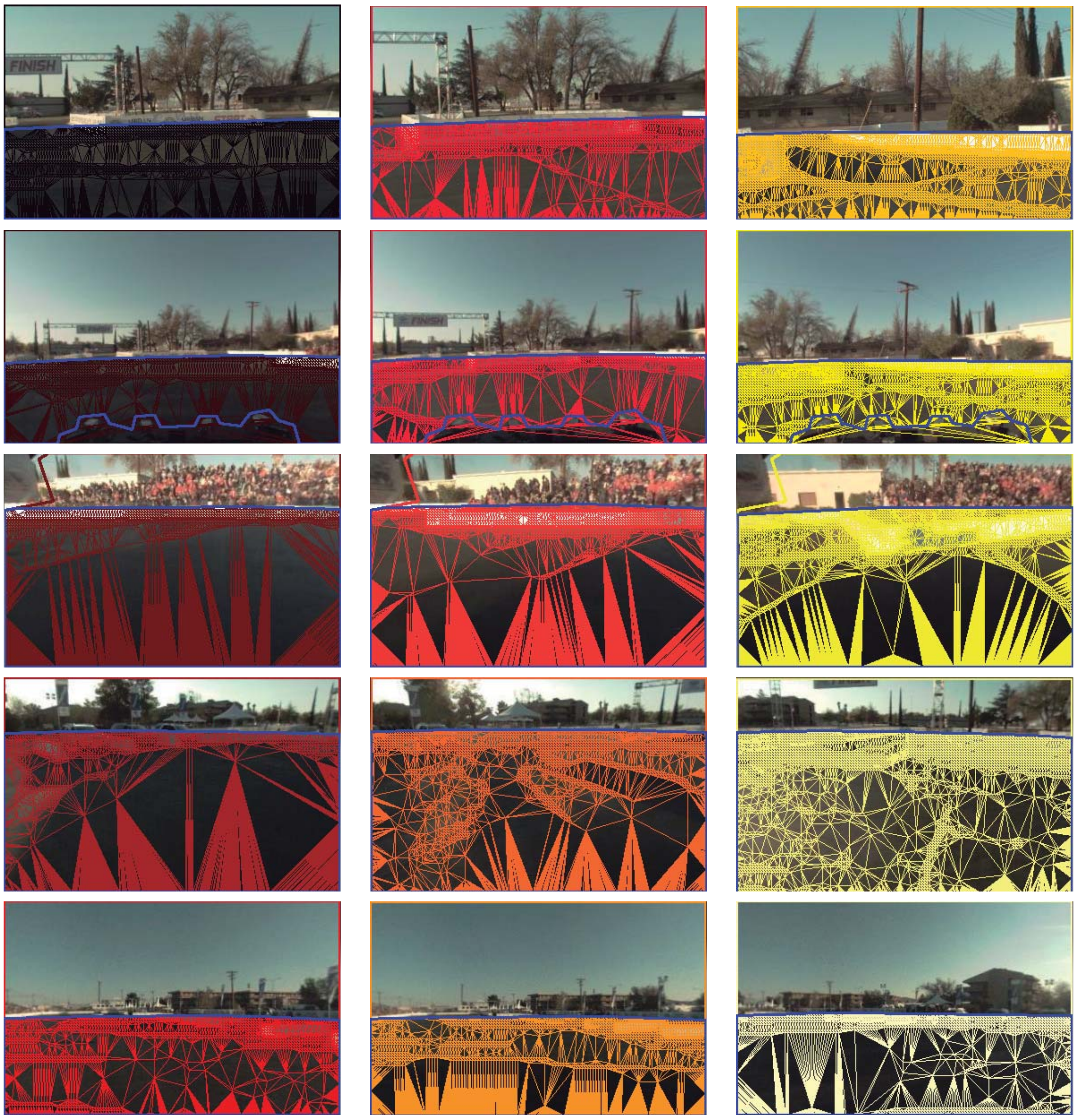


Figure 11.24: Images and local triangulated meshes for all projections shown in Fig. 11.23: First row front center teleobjective camera; Second row front center camera; Third row, front right camera; Fourth row, front left camera; Fifth row, rear center camera; Left column: time t_1, (Fig. 11.20 (a)); Middle column: time t_2, (Fig. 11.20 (b)); Right column: time t_3, (Fig. 11.20 (c));

cameras, as seen in Fig. 11.23 (b). At time t_3, since the vehicle has turned to the left, the rear camera now views a different portion of the ground that had not been captured by any other camera. Note, in Fig. 11.23 (c), how the triangles of the rear camera (the brightest yellow at the bottom right side) map a region that was not seen before and was previously covered only by orphan (blue) triangles.

As the vehicle moves and turns around, more and more of the ground that was not viewed before is covered by the cameras. This is a clear example of why integrating several projections over time is advantageous. A composite photometric description of the environment can be obtained that was impossible to compute without the capability of integrating multiple projections. Although it might be more obvious in this case, this conclusion is valid not only for the ground polygonal primitive but, in potential, to any other primitive.

Figure 11.25 shows the percentage of triangles of each projection as a function of the mission time. The amount of projections increases at each time instant. At time t_3, the amount of different projections (fifteen) makes so that the percentage obtained by each is relatively small (around 10%). In the previous sections, the algorithm for the insertion of candidate triangles was explained. The idea is to insert candidate triangles whenever they have a better quality that the existing triangles they overlap. As each time instant, only newly acquired projections are used to update the mesh. Hence, triangles from previous iterations, if removed, will not be retested for insertion. The algorithm is designed this way so that it may run fast. Consider the present example. If at all times, all projections (new and previous) were considered, at time t_3, the global mesh would have to cycle fifteen local meshes. This represents many tenths of thousands of triangles, and the number would continue to increase. In the end, the algorithm would not be able to comply with real time or near real time demands. Because of this, the triangles of a given projection are tested for insertion a single time. That means that if they are removed they will never again be reinserted. This can be observed in Fig. 11.25 by the fact that none of the projections actually increases the number of triangles it contains. Either the percentage stays the same, or it decreases.

Another important observation that should be made about the results displayed on Fig. 11.25 is related to the orphan triangles. From the observation of the graph it may seem that the number of

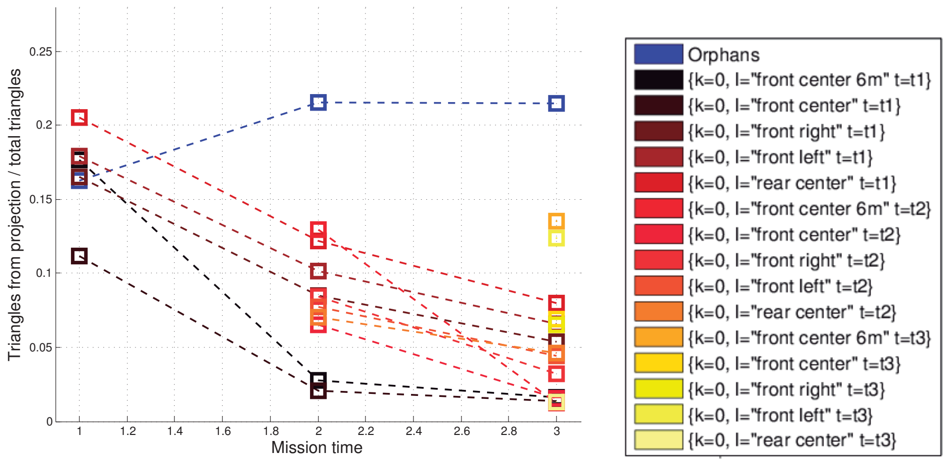


Figure 11.25: Number of triangles from each projection over the total number of triangles in the global mesh as a function of mission time, for the examples shown in Fig. 11.23.

orphan triangles it does not decrease. One could ask what would be the quality of the texture if a high percentage of orphan triangles is present in the mesh. In fact, this phenomenon could lead to a loss in the overall quality of the primitive's mesh, because, as discussed before and unlike triangles with parent projections, orphan triangles do not give guarantee of having an accurate mapping of texture. However, this is not the case. Although the percentage of orphan triangles increases from t_1 to t_2, it remains stable from t_2 to t_3. In the results shown in Figs. 11.22 (a) and (b) the percentage actually decreases. Consider also that, in Fig. 11.25, the maximum observable percentage of orphan triangles is of 23%. This means that there are 77% of triangles that have parent projections. Although this global value is distributed by the large number of projections, the fact is that the greatest portion of the meshes triangles are triangles with a parent projection and, thus, with guaranteed texture mapping quality.

11.4.3 Full Scene Geometric and Photometric Reconstruction

Previous examples have focused on describing particular characteristics of the proposed scene reconstruction algorithm. In the next example an entire scenario reconstruction is shown. The scenario is composed of the entire sequence 1, of the MIT data set. To perform photometric reconstruction and refinement, all five cameras onboard the *Talos* vehicle are used. A view of the scenario by means of an accumulated point cloud is shown in Fig. 11.26 (a). Four views of the scenario are fixed, so that they record the evolution of the representation over time. We refer to these as virtual cameras:

- Virtual camera 1, shown in Fig. 11.26 (b), views the scene as shown red in Fig. 11.26 (a);
- Virtual camera 2, shown in Fig. 11.26 (c), views the scene as shown green in Fig. 11.26 (a);
- Virtual camera 3, shown in Fig. 11.26 (d), views the scene as shown blue in Fig. 11.26 (a);
- Virtual camera 4, shown in Fig. 11.26 (e), views the scene as shown yellow in Fig. 11.26 (a).

The vehicle travels the entire scenario in three minutes, starting from the right (near the green virtual camera) to the left (just above the blue virtual camera), in Fig. 11.26 (a). At periodic intervals, all detected polygonal primitives are tested for mapping with all five cameras. At times $t_1 = 30$, $t_2 = 60$, $t_3 = 90$, $t_4 = 120$, $t_5 = 150$ and $t_6 = 180$ seconds the images from the virtual cameras record the status of the scenario representation at the corresponding time. Figures 11.27, 11.28, 11.29, 11.30, 11.31 and 11.32 show the representation viewed by each of the virtual cameras at times t_1, t_2, t_3, t_4, t_5 and t_6, respectively.

Geometric primitives are represented in the environment by the blue-green polygons. A blue to green colormap is used to color the primitives according to their index, the more recently detected the primitive, the more green it is. Photometry is represented by the texture mapped onto the primitives. Note that at each of the time instants new projections will update the global meshes of the detected

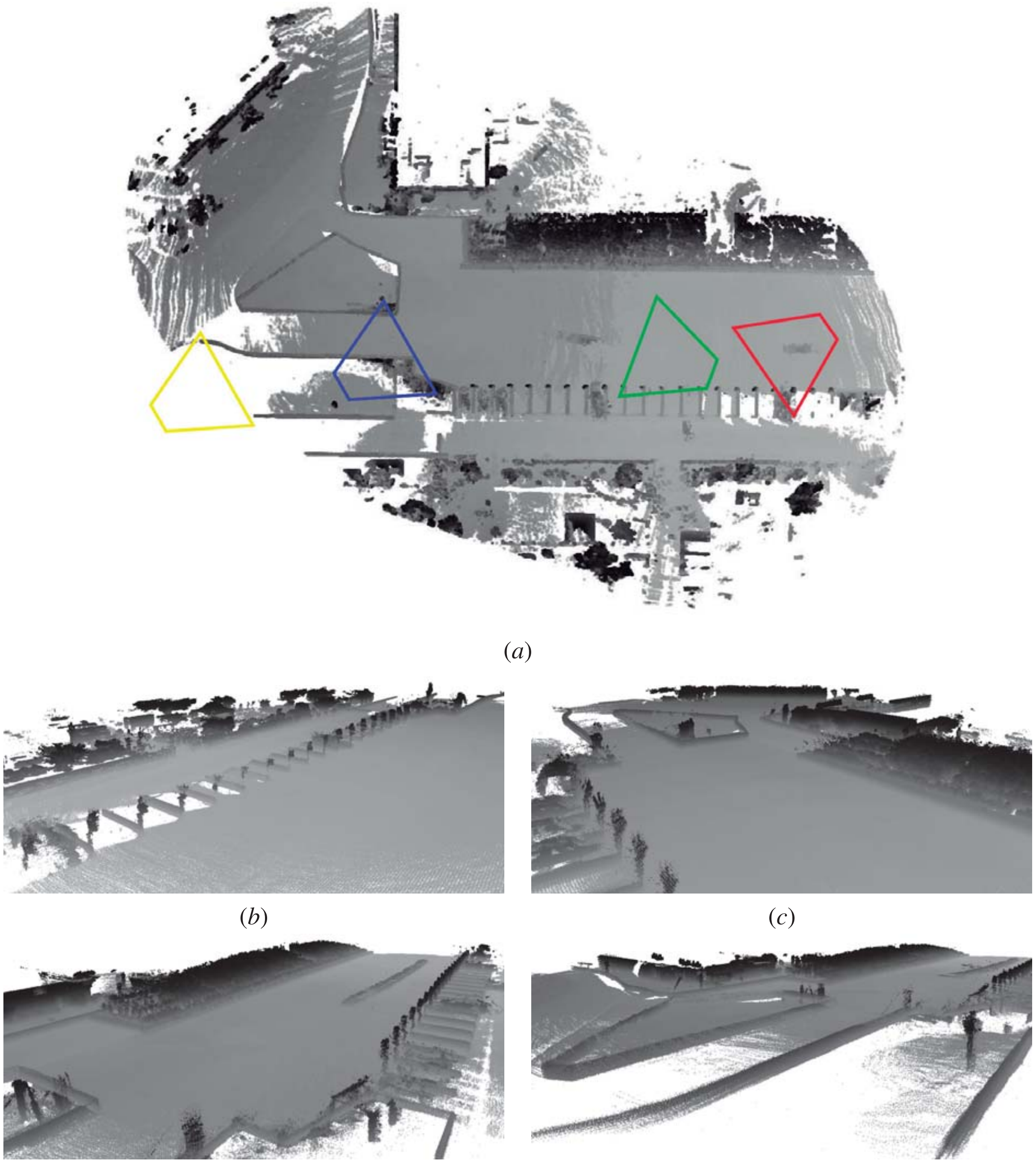


Figure 11.26: Full reconstruction of the sequence 1, MIT data set: (*a*) a top view of the entire scenario; (*b*) virtual camera 1, red in (*a*); (*c*) virtual camera 2, green in (*a*); (*d*) virtual camera 3, blue in (*a*); (*e*) virtual camera 4, yellow in (*a*);

polygonal primitives. Hence the scenario representation will evolve photometrically over time. Furthermore, as discussed in chapter 8, also the geometric representation will evolve over time, since geometric polygonal primitives are also refined. For a better visualization of the representation, the primitive that represents the ground plane will not be textured.

At time t_1 (Fig. 11.27) only two geometric primitives are detected. Since one of them is the ground plane, only the other is texture mapped. At time t_2 (Fig. 11.28), more geometric primitives are detected near to the vehicle. These new primitives are texture mapped, and the previous one is refined using new projections. The process continues until, in Fig. 11.32 the final scene representation is obtained. During the course of the evolution of the representation, several photometric refinements are observable.

For example at time t_4 (Fig. 11.30 (c)) there is an unmapped primitive (light green polygon without texture). The reason was that the primitive is to the rear left of the vehicles position (at t_4). At that time, there was laser data from that primitive, which was why the corresponding geometric polygonal primitive was generated, but the cameras did not view that particular primitive. At time t_5 (Fig. 11.31), the vehicle is in a different position, from where the cameras are already able to view that primitive. As a result, that same primitive appears mapped in Fig. 11.31.

Another clear example occurs at times t_5 and t_6. At time t_5, Fig. 11.31 (d), one of the detected geometric primitives is only partially mapped. Again, at that time, the cameras only viewed a portion of that primitive. However at time t_6 that same primitive is now completely texture mapped.

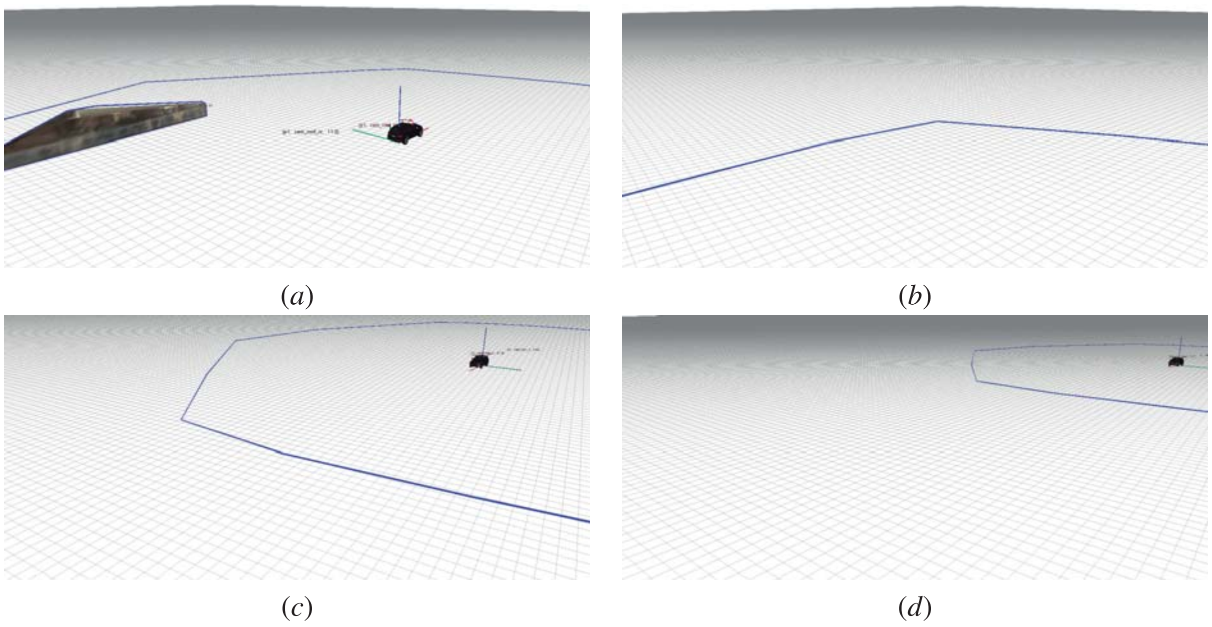


Figure 11.27: Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_1 = 30$ seconds: virtual cameras 1, 2, 3 and 4, (a), (b), (c) and (d), respectively.

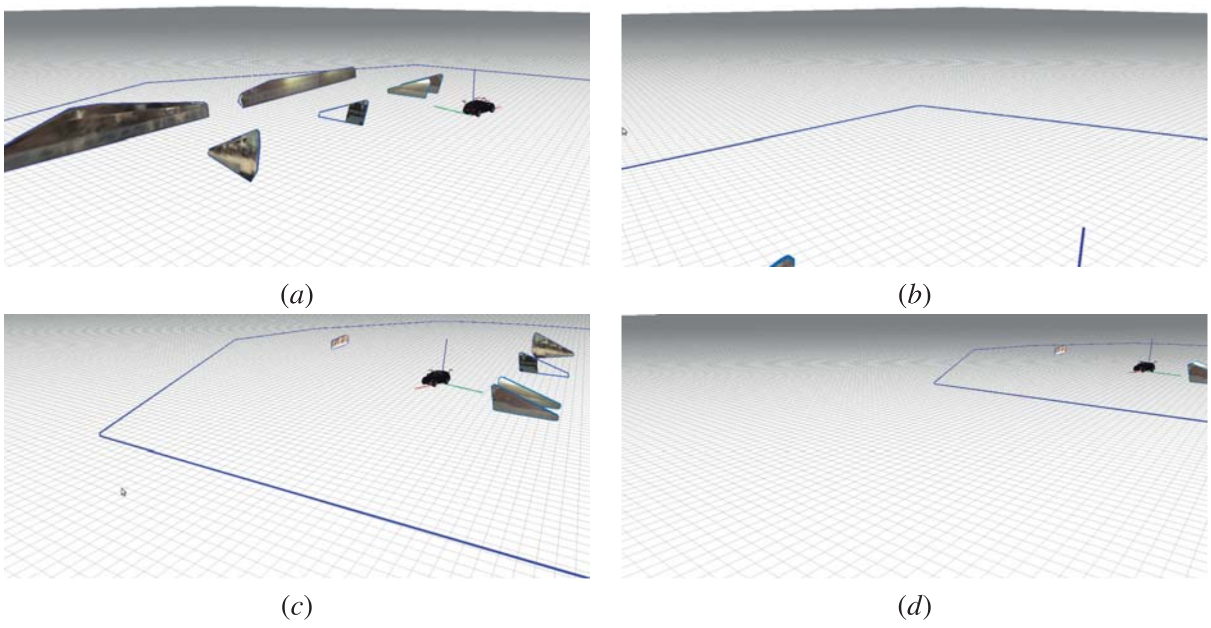


Figure 11.28: Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_2 = 60$ seconds: virtual cameras 1, 2, 3 and 4, (a), (b), (c) and (d), respectively.

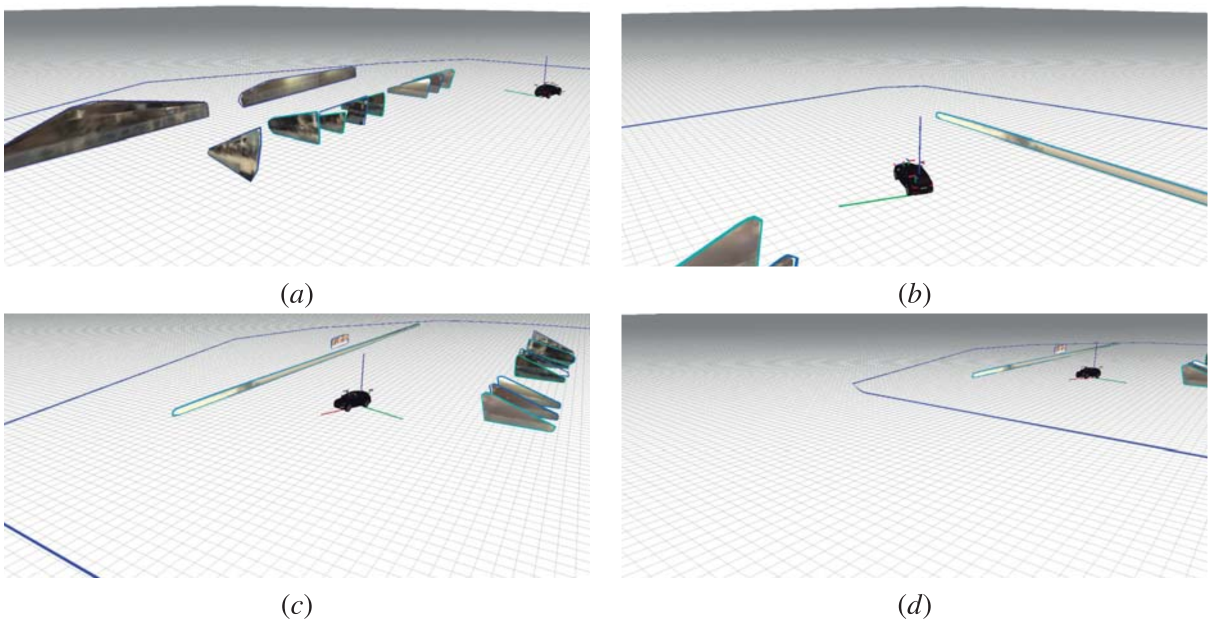
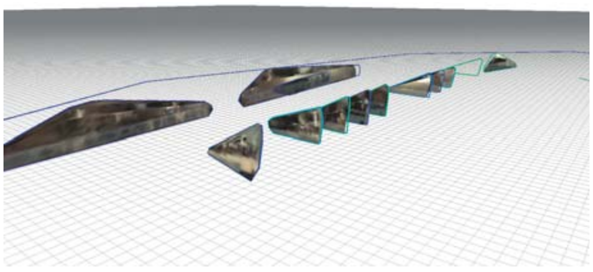
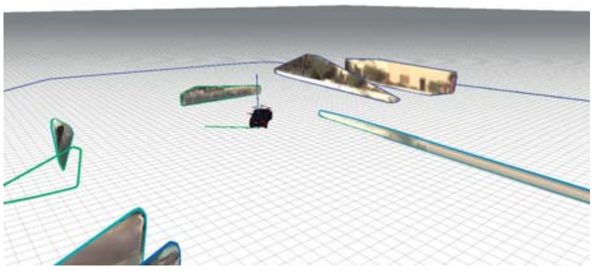


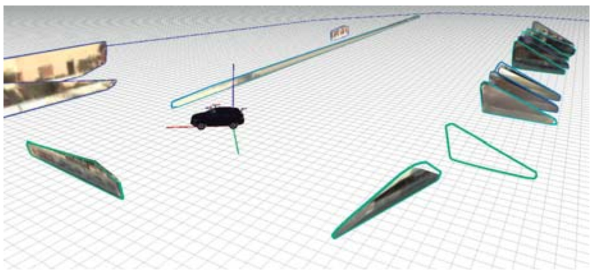
Figure 11.29: Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_3 = 90$ seconds: virtual cameras 1, 2, 3 and 4, (a), (b), (c) and (d), respectively.



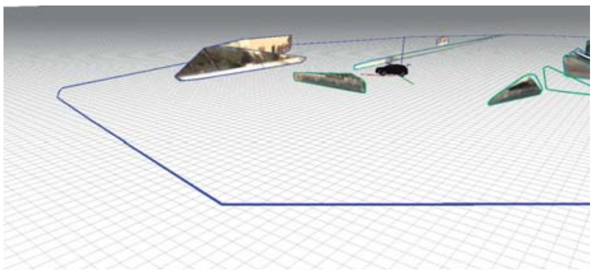
(*a*)



(*b*)

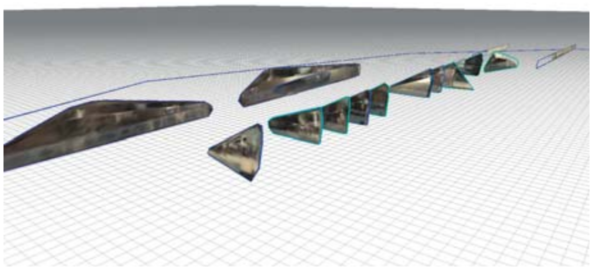


(*c*)

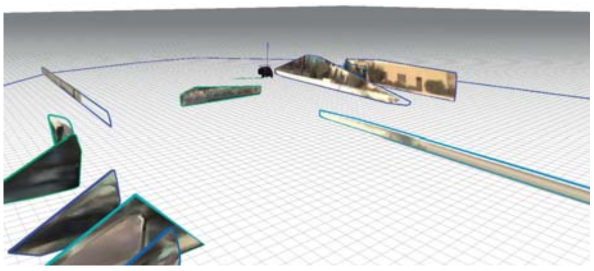


(*d*)

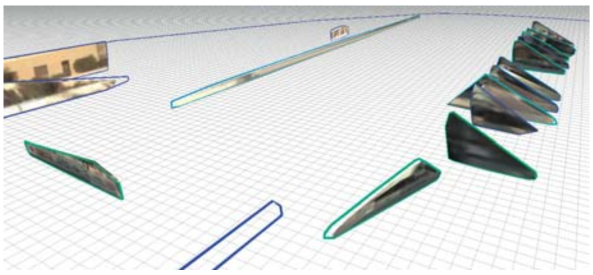
Figure 11.30: Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_4 = 120$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively.



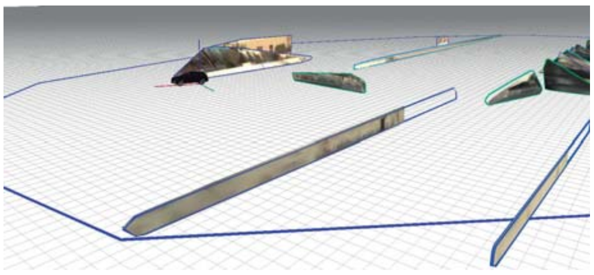
(*a*)



(*b*)



(*c*)



(*d*)

Figure 11.31: Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_5 = 150$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively.

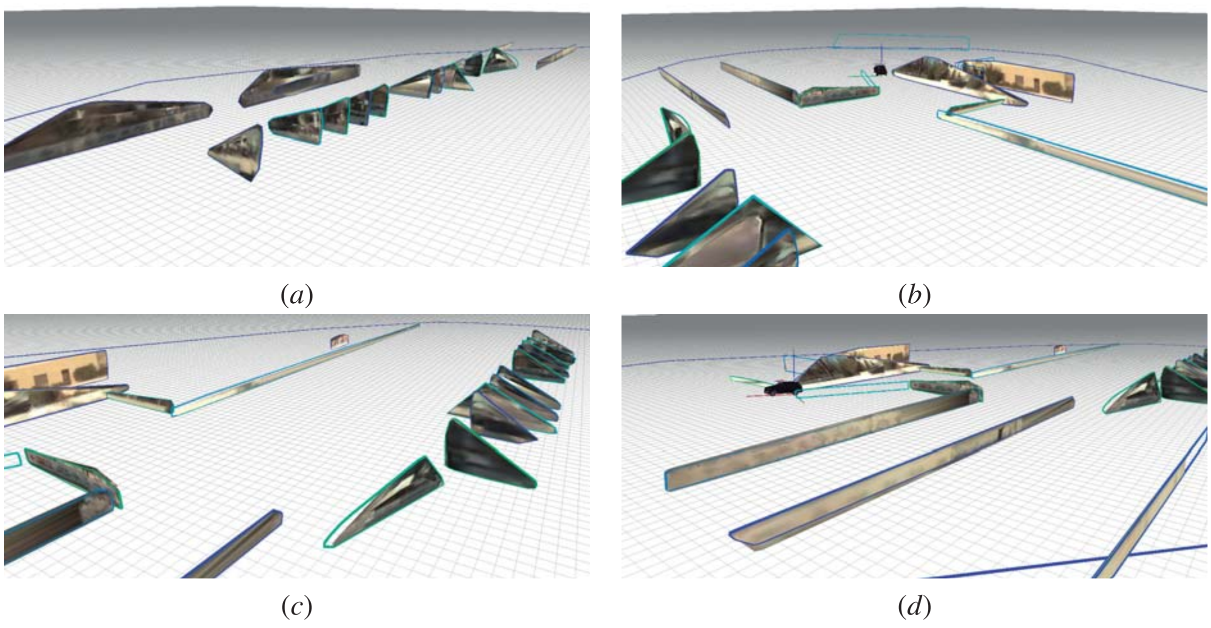


Figure 11.32: Reconstruction of sequence 1, MIT data set. Scene representation at time $t = t_6 = 180$ seconds: virtual cameras 1, 2, 3 and 4, (*a*), (*b*), (*c*) and (*d*), respectively.

11.5 Conclusions

As discussed in the previous sections, the refinement of texture is done by means of a constrained Delaunay triangulation. This triangulated mesh is defined in the local primitive's coordinate frame and, because of this, is a 2D triangulation. The global primitive mesh, as it is called, is built by inserting triangles from local meshes computed in the image space of each projection. From multiple local meshes, the global mesh is computed using quality functions to assert whether triangles should be inserted. The proposed algorithm is capable of refining a mesh from multiple projections, maintaining a good visual quality of the textures. Perhaps most important, the mechanism fill the gaps in the mesh where there are to triangles with parent projections with orphan triangles. Using this mechanism, the holes that could exist between textures of different projections are replaced by orphan triangles where texture is interpolated, resulting in a better overall quality of the texture.

Quite often, different projections are generated at different times. Let us consider an example of a vehicle travelling through a scenario. As it moves, its cameras capture different images. Each image generates a projection. One very interesting feature of the algorithm is that not only it can select the best texture from a set of multiple projections, but it is also capable of providing the best texture at a given time. In other words, for a given primitive, there is not a single texture that represents it. The texture of a primitive may evolve if new projections are added to the global mesh. We say may evolve, since that, depending of the configuration of the projection quality functions, a primitive's texture is refined only if better texture is provided by more recent projections.

This chapter presented results that show that the proposed algorithm is capable of dealing with multiple cameras, multiple projections, and multiple primitives, and come up with a visually appealing scenario representation that is refined over time. At the moment, the photometric generation and refinement of geometric primitives is not computable in real time. Hopefully, future implementations that undergo code optimizations, and are supported by better hardware platform, will achieve real time performance.

Chapter 12

Conclusions

The work presented in this document has addressed the general problem of how technology can be applied to motor vehicles to enable them to be aware of the surrounding environment. This automatic awareness will increase the safety, thus significantly contributing to solving the problem of road accidents. There were four main objectives for the proposed work, which were tackled and developed as discussed next:

- The development of perception based algorithms that process onboard sensor data and generate information about the road scene. This objective was acomplished with the development of:

  – A flood fill based algorithm for detecting the area bounded by the road delimiting lane markers;

  – A lane marker detection algorithm based on comparing simple statistics collected from line candidates;

  – A road maintenance area navigation algorithm, based on the color segmentation of orange pins and a convex hull operation in polar space for determining the location of the *base* of the pins.

- The development and integration of basic software functionalities in the autonomous driving robotic prototypes, that allows them to participate in autonomous driving competitions. This objective was carried out by:

  – The implementation of the flood fill road detection algorithm in the *Atlas2000*;

  – The implementation of the statistical analysis based algorithm for lane marker detection on the *AtlasMV*;

  – The implementation of the multi-camera, multi-modal Inverse Perspective Mapping (IPM) algorithm in the active perception Pan and Tilt Unit (PTU) onboard the *AtlasMV*;

– The developement and implementation of the Laboratory of Automation and Robotics Toolkit (LARtk) software architecture in the *Atlas2000* and *AtlasMV* robotic prototypes;

– The development of a novel path planning algorithm that uses data from the IPM image and a Laser Range Finder (LRF). It was implemented in the *AtlasMV* robot.

- The testing and assessement of the effectiveness of the proposed algorithms on a full scale vehicle was acomplished by:

  – The testing of the proposed IPM approaches in the *AtlasCar* and the comparisson of the results with the classical approach;

  – The testing and implementation of both the LARtk and the Robot Operating System (ROS) software architectures in the *AtlasCar*.

- The development of alternative data representations that may cope with multiple sensors and that improve the effectiveness of subsequent processing algorithms was achieved by the development of:

  – A multi-camera, multi-modal approach to IPM.

  – Several algorithms for computing the color correction of images.

  – An integrated geometric and photometric scene representation that is computed much faster than alternative techniques.

  – Mechanisms for refining both the geometric and the photometric representations over time.

The main focus of this thesis was on the development of algorithms and techniques for enhanced representation of onboard sensor data. Finding alternative sensor data representations is very useful, since it may improve the effectiveness of algorithms used in the perception of road-like scenarios. Additionally, these intermediate scene representations may be advantageous since thay are devised to cope with the large amounts of data, thus enabling real-time computation. One example is the Velodyne laser [Velodyne 2012], which is now considered a standard in autonmous vehicles, but that generates 1.3 million range data points per second. It was shown that classical approaches (such as 3D triangulations) are unable to process such large amounts of data in real-time. The polygonal primitives scene representation approach has proved to be six times faster than alternative methodologies. Nonetheless, it was not possible to achieve real-time computation of geometric polygonal primitives. It should be possible to improve parts of the algorithm to address this issue. Another advantage of intermediate representations is that they can cope with the asynchrony of the data in multi sensorial setups. This is particularlly important, since that autonomous vehicles are equipped with a vast number of sensors of varied nature. Additional advantages related to the usage of an intermediate data representation framework are discussed in section 12.2.

Throughout the course of the work, the testing and validation of algorithms in real robotic platforms were done in multiple occasions. The *Atlas2000* and the *AtlasMV* robotic prototypes were used as test beds for many of the proposed algorithms. This work has also contributed in the development of these prototypes. These robots have participated in the Autonomous Driving Competition (ADC) of the Portuguese Robotics Open (PRO) and have won the competition a total of six times. Furthermore, this work has contributed to the development of the *AtlasCar*, the first full scale autonomous vehicle developed in Portugal. These developments were discussed in chapter 3.

Several software architectures were used for programing the robots. Both Carnegie Mellon Robot Navigation Toolkit (CARMEN) and ROS software architectures were implemented on the robots or on applications used to test the algorithms. Additionally, this work contributed to the development of LARtk, an extension of CARMEN that features shared memory functionalities for easing the transfer of large messages. Nowadays, ROS seems to be the standard for robot programming, due to the very large community supporting it. However, the usage of CARMEN and the development of LARtk have been very important to have a better insight of ROS. In fact, the underlying architectures are very similar, relying on separate modules and inter process communications. Chapter 4 addressed these topics.

In chapter 5, a novel multi-camera, multi-modal IPM technique was presented. The technique makes use of a LRF sensor to produce higher quality IPM images when compared to classical IPM approaches. Artifacts caused by other vehicles or pedestrians, which appear in a classical IPM image, are filtered in the proposed approach.

Chapter 6 focused on the issue of color correction of images for quality mosaicking. Three different algorithms were proposed. It was shown that they are very effective and, in most cases, better than the state of the art.

In chapters 7 through 11, a novel scene representation algorithm was proposed that relies more on 3D data processing. Chapters 8 and 9 presented 3D processing algorithms designed to compute a geometric scene representation. It was shown that a polygonal primitive based approach can be very efficient and is computed in less time than other triangulation approaches.

Chapters 10 and 11 proposed algorithms for texture mapping the geometric representation, thus extending the representation with the photometric dimension. There are no techniques proposed in literature similar to the proposed approach, in particular in what concerns the capability to cope with changing the photometric representation as new data is received.

In sum, during the course of the work, the following specific contributions were made:

- A multi-camera multi-modal IPM methodology that uses data from a LRF to enhance the IPM resulting image. Results have shown that the images produced by this approach are more precise than images computed using the classical IPM approaches;

- Three alternative color correction algorithms for obtaining quality mosaics. Results have shown that these techniques are more efficient and robust than the state of the art;

- A methodology based on geometric polygonal primitives, that computes an accurate geometric scene representation in approximately one sixth of the time of the fastest 3D triangulation approach tested. Additionally, a method was proposed to refine the polygonal primitive based representation over time;

- A methodology for texture mapping the geometric polygonal primitive representation using Data Dependent Triangulation (DDT). Additionally, a methodology for refining the model's texture over time was proposed;

- The integration of some of the algorithms described in previous lines (as well as others) into three robotic prototypes. Testing and evaluation of the algorithms in real platforms used in several competitions.

12.1 Publications

The following publications derived from the work described in this thesis:

1. Miguel Oliveira and Vitor Santos. Multi-camera active perception system with variable image perspective for mobile robot navigation. In Portuguese Robotics Open 2008, International Conference on Autonomous Robot Systems and Competitions, April 2008. Best conference paper award. Selected for publication in Robótica (ISSN 0874-9019), published in number 76, September 2009.

2. Miguel Oliveira, Procopio Stein, Jorge Almeida, and Vitor Santos. Modular scalable architecture for the navigation of the atlas autonomous robots. In Portuguese Robotics Open 2009, International Conference on Autonomous Robot Systems and Competitions, April 2009.

3. Vitor Santos, Jorge Almeida, Emanuel Avila, David Gameiro, Miguel Oliveira, Ricardo Pascoal, Remi Sabino, and Procopio Stein. Atlascar - technologies for a computer assisted driving system on board a common automobile. In 13th International IEEE Conference on Intelligent Transportation Systems, pages 1421 - 1427, September 2010.

4. Miguel Oliveira and Vitor Santos. Autonomous driving competition: Perception approaches used in the atlas project. In Portuguese Robotics Open 2011, International Conference on Autonomous Robot Systems and Competitions, April 2011.

5. Miguel Oliveira, Angel D. Sappa, and Vitor Santos. Unsupervised local color transfer for coarsely registered images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 201-208, June 2011.

6. Miguel Oliveira, Vitor Santos, and Angel D. Sappa. Short term path planning using a multiple hypothesis evaluation approach for an autonomous driving competition. In IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles, October 2012.

7. Miguel Oliveira, Angel D. Sappa, and Vitor Santos. Color correction using 3d gaussian mixture models. In ICIAR (1), pages 97-106, June 2012.

8. Miguel Oliveira, Angel D. Sappa, and Vitor Santos. Color correction for onboard multi-camera systems using 3d gaussian mixture models. In Intelligent Vehicles Symposium, pages 299-303, 2012.

9. Miguel Almeida, Paulo Dias, Miguel Oliveira, and Vitor Santos. 3d-2d laser range finder calibration using a conic based geometry shape. In ICIAR (1), pages 312-319, 2012.

Additionally, there are some previous publications that have inspired parts of the work:

1. Miguel Oliveira, Rui Cancela, Miguel Neta, and Vitor Santos. Atlas: Robô com visão orientado para provas de condução autónoma. In Portuguese Robotics Open 2005, International Conference on Autonomous Robot Systems and Competitions, April 2005. Selected for publication in Robótica (ISSN 0874-9019), published in number 62, January 2006.

2. Miguel Oliveira and Vitor Santos. A vision-based solution for the navigation of the atlas autonomous robots. In Portuguese Robotics Open 2007, International Conference on Autonomous Robot Systems and Competitions, April 2007. Selected for publication in Robótica (ISSN 0874- 9019), published in number 69, October 2007.

3. Miguel Oliveira and Vitor Santos. Combining view-based object recognition with template matching for the identification and tracking of fully dynamic targets. In Portuguese Robotics Open 2007, International Conference on Autonomous Robot Systems and Competitions, April 2007. Selected for publication in Robótica (ISSN 0874-9019), published in number 73, October 2008.

4. Miguel Oliveira and Vitor Santos. Real-time extraction of road border lines using simple statistical descriptors. In IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles, September 2008.

5. Miguel Oliveira and Vitor Santos. Automatic detection of cars in real roads using haar features. In Controlo, Portuguese Conference on Automatic Control, July 2008.

12.2 Future Work

Future work may include improvements on many of the algorithms described before. However, this section will focus on the advantages inherent to a intermediate sensor representation framework, in particular, to the one proposed in chapters 8, 9, 10 and 11.

The question that is addressed is whether or not it is preferable to process sensor data directly with the traditional algorithms (pattern recognition algorithms for road, traffic sign or pedestrian detection) or instead to have an algorithm that collects raw sensor data and generates an intermediate representation of that data. Rather than analysing the raw sensor data, traditional pattern recognition algorithms are alternatively used to process the intermediate representation of the data. The advantages brought by the second option are many. They are discussed in the following lines.

First of all, an intermediate scene representation algorithm is specifically devised for coping with multi-sensor setups. Thus, unlike traditional algorithms, it specializes in handling common problems such as the asynchrony and the large size of the data received from sensors. It was shown in chapter 7 that current autonomous vehicles generate very large amounts of data. Traditional algorithms have difficulties in processing that much data in real-time. Furthermore, very often the data collected by the sensors is repeated, as in the case of a vehicle stopped in a static environment. In a traditional setup, recognition algorithms would repeatedly process the data. An intermediate representation framework can cope with this issue by keeping the representation unchanged whenever the sensor data is repeated.

The proposed approach for an intermediate sensor representation comprises the computation of a 3D model, along with the corresponding texture. There is a very important advantage associated with this. The efficiency of traditional image based pattern recognition algorithms is very sensitive to changes in the perspective from which objects are observed. A 3D textured intermediate representation may synthesize images of the reconstructed scene from any desired point of view. By selecting the adequate point of view, synthesized images can be free of perspective distortions. If these synthesized images are given to the recognition algorithms, it is expected that their efficiency significantly improves. Note that this is the underlying idea of the IPM methodology: to compute a perspective distortion free bird's eye view of the road surface, and then to process that image. A 3D texture representation is capable of synthesizing images from any surface, not just the road.

Since the representation is computed from the input of several sensors, subsequent processing algorithms would actually be searching for patterns in a representation that is a composition of multiple sensor inputs. Suppose for example that there are two cameras facing the front of the host vehicle. Another vehicle in front is located so that the images from each camera capture only half of the vehicle. Unless designed in very complex ways, traditional recognition algorithms (in this case used for detecting a vehicle) are not suited to process images with only half of a vehicle. An intermediate representation can fuse the texture from both cameras and synthesize an image where the vehicle is completely visible, thus enabling the detection of the vehicle. In this way, recognition algorithms may be kept simple and straightforward, since that it is the representation algorithm that does the job of synthesizing an adequate image for processing, by selecting an adequate point of view and fusing information from several sensors.

The final advantage provided by an intermediate scene representation can be viewed as more futuristic concept. Driving a vehicle is actually a very complex task. It is vital for the driver (or the autonomous system) to have a high level of *understanding* of the road scenario. More than that, the task of driving requires also the ability to anticipate the actions of other agents on the road. In other words, driving comprises not only a comprehensive analysis of the road scenario by the driver / vehicle, but in addition an estimation of the intentions of the other entities in the proximity of the vehicle. An intermediate scene representation may be very valuable to ascertain this second component. From an intermediate sensor representation, it is also possible to synthesize the estimated view of other agents. This is a first step in making an autonomous system grasping not only the position and velocity of pedestrians and other vehicles, but also being capable of formulating an estimate about what they view and, consequently, about the intentions of those subjects.

References

[Abadpour & Kasaei 2004] A. Abadpour and S. Kasaei. *A fast and efficient fuzzy color transfer method*. In Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, pages 491–494, December 2004.

[Abadpour & Kasaei 2007] Arash Abadpour and Shohreh Kasaei. *An efficient PCA-based color transfer method*. Journal of Visual Communication and Image Representation, vol. 18, no. 1, pages 15–34, February 2007.

[Aichholzer *et al.* 1995] Oswin Aichholzer, Franz Aurenhammer, David Alberts and Bernd Gartner. *A Novel Type of Skeleton for Polygons*. j-jucs, vol. 1, no. 12, pages 752–761, December 1995.

[Alliez *et al.* 2012] Pierre Alliez, Sylvain Pion and Ankit Gupta. *Principal Component Analysis*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Almeida & Santos 2010] J. Almeida and V. Santos. *Laser-based Tracking of Mutually Occluding Dynamic Objects*. In Portuguese Conference in Automatic Control. 2010, September 2010.

[Aly 2008] M. Aly. *Real time detection of lane markers in urban streets*. In IEEE Intelligent Vehicles Symposium, pages 7 –12, June 2008.

[Amato & Preparata 1993] Nancy M. Amato and Franco P. Preparata. *An NC parallel 3D convex hull algorithm*. In Proceedings of the ninth annual symposium on Computational geometry, SCG '93, pages 289–297, 1993.

[Andrew 1979] A M Andrew. *Another Efficient Algorithm for Convex Hulls in Two Dimensions*. Information Processing Letters, vol. 9, no. 5, pages 216–219, 1979.

[Applanix 2012] Applanix. *Applanix POS LV homepage*. http://www.applanix.com/products/land/pos-lv.html, May 2012.

[Arya & Mount 1993] S. Arya and D.M. Mount. *Algorithms for fast vector quantization*. In Data Compression Conference, 1993. DCC '93., pages 381 –390, 1993.

[Arya *et al.* 1998] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman and Angela Y. Wu. *An optimal algorithm for approximate nearest neighbor searching fixed dimensions*. J. ACM, vol. 45, no. 6, pages 891–923, November 1998.

[Asbach *et al.* 2012] M. Asbach, D. Abrosimov, G. Bradski, O. Bornet, J. Bouguet, A. Bovyrin, D. Burenkov, T. Cherepanova, V. Cherepennikov, B. Chudinovich, D. Cowperthwaite, B. Davies, D. Eaton, V. Eruhimov, I. Grachev, V. Khudyakov, A. Kuranov, V. Kuriakin, R. Lienhart, P. Lantz, S. Molinov, V. Mosyagin, A. Nefian, S. Oblomov, V. Pisarevsky, A. Pleskov, A. Sobolev and V. Zaguskin. *Open source computer vision library, (Opencv)*. http://opencv.willowgarage.com/wiki/, May 2012.

[Banachowicz 2012] Konrad Banachowicz. *LMS1xx laser drivers*. http://ros.org/wiki/LMS1xx, May 2012.

[Barber *et al.* 1996] C. Bradford Barber, David P. Dobkin and Hannu Huhdanpaa. *The Quickhull algorithm for convex hulls*. ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE, vol. 22, no. 4, pages 469–483, 1996.

[Barrera *et al.* 2012a] Fernando Barrera, Felipe Lumbreras and Angel D. Sappa. *Multimodal Stereo Vision System: 3D Data Extraction and Algorithm Evaluation*. IEEE Journal of Selected Topics in Signal Processing, 2012.

[Barrera *et al.* 2012b] Fernando Barrera, Felipe Lumbreras and Angel D. Sappa. *Multispectral Piecewise Planar Stereo using Manhattan-World Assumption*. Pattern Recognition Letters, 2012.

[Baten *et al.* 1998] Stefan Baten, Michael Lützeler, Ernst D. Dickmanns, Robert Mandelbaum and Peter J. Burt. *Techniques for Autonomous, Off-Road Navigation*. IEEE Intelligent Systems, vol. 13, no. 6, pages 57–65, 1998.

[Ben-Ezra *et al.* 2005] M. Ben-Ezra, A. Zomet and S.K. Nayar. *Video super-resolution using controlled subpixel detector shifts*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 6, pages 977–987, June 2005.

[Berkmann & Caelli 1994] J. Berkmann and T. Caelli. *Computation of surface geometry and segmentation using covariance techniques*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 16, no. 11, pages 1114 –1116, November 1994.

[Bernardini *et al.* 1999] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva and G. Taubin. *The ball-pivoting algorithm for surface reconstruction*. Visualization and Computer Graphics, IEEE Transactions on, vol. 5, no. 4, pages 349 –359, October 1999.

[Bertozzi & Broggi 1998] M. Bertozzi and A. Broggi. *GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection*. IEEE Transactions on Image Processing, vol. 7, no. 1, pages 62 –81, January 1998.

[Bertozzi & Broggi 1999] Massimo Bertozzi and Alberto Broggi. *Tools for code optimization and system evaluation of the image processing system PAPRICA-3*. Journal of Systems Architecture, vol. 45, no. 6-7, pages 519–542, 1999.

[Bertozzi *et al.* 1997] M. Bertozzi, A. Broggi and A. Fascioli. *Real-time obstacle detection on a massively parallel linear architecture*. In International Conference on Algorithms and Architectures for Parallel Processing, pages 535 –542, December 1997.

[Bertozzi *et al.* 2011] Massimo Bertozzi, Luca Bombini, Alberto Broggi, Michele Buzzoni, Elena Cardarelli, Stefano Cattani, Pietro Cerri, Alessandro Coati, Stefano Debattisti, Andrea Falzoni, Rean Isabella Fedriga, Mirko Felisa, Luca Gatti, Alessandro Giacomazzo, Paolo Grisleri, Maria Chiara Laghi, Luca Mazzei, Paolo Medici, Matteo Panciroli, Pier Paolo Porta, Paolo Zani and Pietro Versari. *VIAC: An out of ordinary experiment*. In Intelligent Vehicles Symposium, pages 175–180, 2011.

[Birk *et al.* 2009] A. Birk, N. Vaskevicius, K. Pathak, S. Schwertfeger, J. Poppinga and H. Buelow. *3-D perception and modeling*. Robotics Automation Magazine, IEEE, vol. 16, no. 4, pages 53 –60, December 2009.

[Blanco *et al.* 2009] José-Luis Blanco, Francisco-Angel Moreno and Javier González. *A Collection of Outdoor Robotic Datasets with centimeter-accuracy Ground Truth*. Autonomous Robots, vol. 27, no. 4, pages 327–351, November 2009.

[Blanco *et al.* 2012] José-Luis Blanco, Javier González-Jiménez and Juan-Antonio Fernández-Madrigal. *An alternative to the Mahalanobis distance for determining optimal correspondences in data association*. IEEE Transactions on Robotics (T-RO), vol. 28, no. 4, August 2012.

[Blythe 2006] David Blythe. *The Direct3D 10 system*. ACM Trans. Graph., vol. 25, no. 3, pages 724–734, July 2006.

[Bowman & Mihelich 2012] James Bowman and Patrick Mihelich. *Camera calibration*. `http://www.ros.org/wiki/camera_calibration`, May 2012.

[Branden 1986] Bart Branden. *The Surveyor's Area Formula*. The College Mathematics Journal, vol. 17, no. 4, pages 326–337, August 1986.

[Broggi *et al.* 2000] Alberto Broggi, Massimo Bertozzi and Alessandra Fascioli. *Architectural Issues on Vision-Based Automatic Vehicle Guidance: The Experience of the ARGO Project*. Real-Time Imaging, vol. 6, no. 4, pages 313–324, 2000.

[Broggi *et al.* 2006] Alberto Broggi, Stefano Cattani, Pier Paolo Porta and Paolo Zani. *A Laserscanner-Vision Fusion System Implemented on the TerraMax Autonomous Vehicle*. In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pages 111 – 116, October 2006.

[Broggi *et al.* 2008] A. Broggi, P. Cerri, S. Ghidoni, P. Grisleri and Ho Gi Jung. *Localization and analysis of critical areas in urban scenarios*. In IEEE Intelligent Vehicles Symposium, pages 1074 –1079, June 2008.

[Broggi *et al.* 2010] Alberto Broggi, Andrea Cappalunga, Claudio Caraffi, Stefano Cattani, Stefano Ghidoni, Paolo Grisleri, Pier Paolo Porta, Matteo Posterli and Paolo Zani. *TerraMax Vision at the Urban Challenge 2007*. IEEE Transactions on Intelligent Transportation Systems, vol. 11, no. 1, pages 194–205, 2010.

[Brown & Lowe 2003] M. Brown and D. G. Lowe. *Recognising panoramas*. In Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2, pages 1218–1225, October 2003.

[Brown & Lowe 2007] Matthew Brown and David G. Lowe. *Automatic panoramic image stitching using invariant features*. International Journal of Computer Vision, vol. 74, pages 59–73, August 2007.

[Bykat 1978] A Bykat. *Convex hull of a finite set of points in two dimensions*. Information Processing Letters, vol. 7, pages 296–298, 1978.

[Cacciola 2012] Fernando Cacciola. *2D Straight Skeleton and Polygon Offsetting*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Cerri & Grisleri 2005] P. Cerri and P. Grisleri. *Free Space Detection on Highways using Time Correlation between Stabilized Sub-pixel precision IPM Images*. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 2223 – 2228, April 2005.

[Cervenansksy *et al.* 2010] Michal Cervenansksy, Zsolt Toth, Juraj Starinsky, Andrej Ferko and Milos Sramek. *Parallel GPU-based data-dependent triangulations*. Computers and Graphics, vol. 34, no. 2, pages 125 – 135, 2010.

[Chang & Kim 2009] Jung-Woo Chang and Myung-Soo Kim. *Efficient triangle–triangle intersection test for OBB-based collision detection*. Computers & Graphics, vol. 33, no. 3, pages 235–240, June 2009.

[Chen & Bhanu 2009] Hui Chen and B. Bhanu. *Efficient Recognition of Highly Similar 3D Objects in Range Images*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 31, no. 1, pages 172 –179, January 2009.

[Chen & Lai 2011] Yi-Ling Chen and Shang-Hong Lai. *An Orientation Inference Framework for Surface Reconstruction From Unorganized Point Clouds*. Image Processing, IEEE Transactions on, vol. 20, no. 3, pages 762 –775, March 2011.

[Chen *et al.* 2009] Yi-Liang Chen, Venkataraman Sundareswaran, Craig Anderson, Alberto Broggi, Paolo Grisleri, Pier Paolo Porta, Paolo Zani and John Beck. *TerraMax: Team Oshkosh Urban Robot*. In The DARPA Urban Challenge, pages 595–622, 2009.

[Chew 1987] L. P. Chew. *Constrained Delaunay triangulations*. In Proceedings of the third annual symposium on Computational geometry, SCG '87, pages 215–222, 1987.

[Cignoni *et al.* 1998] P. Cignoni, C. Rocchini and R. Scopigno. *Metro: Measuring Error on Simplified Surfaces*. Computer Graphics Forum, vol. 17, pages 167 –174, October 1998.

[CNR 2005] Visual Computing Lab ISTI CNR. *MeshLab*, 2005. http://meshlab.sourceforge.net/.

[Comaniciu & Meer 2002] Dorin Comaniciu and Peter Meer. *Mean shift: a robust approach toward feature space analysis*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pages 603–619, May 2002.

[Coulombeau & Laurgeau 2002] P. Coulombeau and C. Laurgeau. *Vehicle yaw, pitch, roll and 3D lane shape recovery by vision*. In Proceedings of the IEEE Intelligent Vehicles Symposium, pages 619–625, Versailles, France, June 2002.

[Coulter & Mueller 1994] R. Craig Coulter and George G. Mueller. *A Reconfiguration Study for the NavLab II Mobile Robot*, January 1994.

[Cousins *et al.* 2010a] S. Cousins, B. Gerkey, K. Conley and W. Garage. *Sharing Software with ROS [ROS Topics]*. Robotics Automation Magazine, IEEE, vol. 17, no. 2, pages 12 –14, June 2010.

[Cousins *et al.* 2010b] S. Cousins, B. Gerkey, K. Conley and W. Garage. *Welcome to ROS Topics [ROS Topics]*. Robotics Automation Magazine, IEEE, vol. 17, no. 1, pages 12 –14, June 2010.

[Darms *et al.* 2009] Michael Darms, Paul Rybski, Christopher R. Baker and Christopher Urmson. *Obstacle Detection and Tracking for the Urban Challenge*. IEEE Transactions on Intelligent Transportation Systems, vol. 10, no. 3, pages 475–485, September 2009.

[de Medeiros Brito *et al.* 2008] A. de Medeiros Brito, A.D. Doria Neto, J. Dantas de Melo and L.M. Garcia Goncalves. *An Adaptive Learning Approach for 3-D Surface Reconstruction From Point Clouds*. Neural Networks, IEEE Transactions on, vol. 19, no. 6, pages 1130 –1140, June 2008.

[Debevec *et al.* 1998] Paul Debevec, Yizhou Yu and George Boshokov. *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*. Rapport technique, 1998.

[DeMarco *et al.* 2011] K. DeMarco, M.E. West and T.R. Collins. *An implementation of ROS on the Yellowfin autonomous underwater vehicle (AUV)*. In OCEANS 2011, pages 1 –7, September 2011.

[Demaret *et al.* 2006] Laurent Demaret, Nira Dyn and Armin Iske. *Image compression by linear splines over adaptive triangulations*. Signal Process., vol. 86, no. 7, pages 1604–1616, July 2006.

[Derenick 2012] Jason Derenick. *Sick toolbox*. `http://www.ros.org/wiki/sicktoolbox`, May 2012.

[Devillers & Guigue 2002] Olivier Devillers and Philippe Guigue. *Faster Triangle-Triangle Intersection Tests*. Rapport technique RR-4488, INRIA, June 2002.

[Dey *et al.* 2005] T.K. Dey, G. Li and J. Sun. *Normal estimation for point clouds: a comparison study for a Voronoi based method*. In Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings, pages 39 – 46, June 2005.

[Dickmanns & Mysliwetz 1992] Ernst D. Dickmanns and Birger D. Mysliwetz. *Recursive 3-D Road and Relative Ego-State Recognition*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 14, no. 2, pages 199–213, 1992.

[Dickmanns 2004] Ernst D. Dickmanns. *Dynamic Vision-Based Intelligence*. AI Magazine, vol. 25, no. 2, pages 10–30, 2004.

[Dickmanns 2012] Ernst D. Dickmanns. *Detailed Visual Recognition of Road Scenes for Guiding Autonomous Vehicles*. In Advances in Real-Time Systems, pages 225–244, 2012.

[Diosi *et al.* 2011] A. Diosi, S. Segvic, A. Remazeilles and F. Chaumette. *Experimental Evaluation of Autonomous Driving Based on Visual Memory and Image Based Visual Servoing*. IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 3, pages 870–883, 2011.

[Dolgov *et al.* 2010] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo and James Diebel. *Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments*. I. J. Robotic Res., vol. 29, no. 5, pages 485–501, 2010.

[Dornaika *et al.* 2011] F. Dornaika, J. Álvarez, A. Sappa and A. López. *A New Framework for Stereo Sensor Pose Through Road Segmentation and Registration*. IEEE Trans. on Intelligent Transportation Systems, vol. in press, 2011.

[Duncan & Ayache 2000] J.S. Duncan and N. Ayache. *Medical image analysis: progress over two decades and the challenges ahead*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 1, pages 85–106, January 2000.

[Dyn *et al.* 1992] Nira Dyn, David Levin and Samuel Rippa. *Boundary correction for piecewise linear interpolation defined over data-dependent triangulations*. J. Comput. Appl. Math., vol. 39, no. 2, pages 179–192, March 1992.

[Eberly & Shoemake 2004] David H. Eberly and Ken Shoemake. *Chapter 10 - Quaternions*. In Game Physics, pages 507 – 544. Morgan Kaufmann, San Francisco, 2004.

[Ehlgen *et al.* 2008] T. Ehlgen, T. Pajdla and D. Ammon. *Eliminating Blind Spots for Assisted Driving*. IEEE Trans. on Intelligent Transportation Systems, vol. 9, no. 4, pages 657–665, 2008.

[European Commission, Directorate-General for Energy and Transports 2001] European Commission, Directorate-General for Energy and Transports. *Halving the number of road accident victims in the EU by 2010: A shared responsability*. http://ec.europa.eu/transport/road_safety/observatory/doc/memo_rsap_en.pdf, September 2001.

[European Road Safety Observatory 2011] European Road Safety Observatory. *Annual Statistical Report*. http://ec.europa.eu/transport/road_safety/pdf/statistics/dacota/dacota-3.5-asr-2011.pdf, March 2011.

[Evans & Kim 1998] O. D. Evans and Y. Kim. *Efficient Implementation of Image Warping on a Multimedia Processor*. Real-Time Imaging, vol. 4, no. 6, pages 417 – 428, 1998.

[Fang *et al.* 2009] H. Fang, M. Yang, R. Yang and C. Wang. *Ground-Texture-Based Localization for Intelligent Vehicles*. IEEE Trans. on Intelligent Transportation Systems, vol. 10, no. 3, pages 463–468, 2009.

[Fecker *et al.* 2008] U. Fecker, M. Barkowsky and A. Kaup. *Histogram-based prefiltering for luminance and chrominance compensation of multiview video*. IEEE Transactions on Circuits and Systems for Video Technology, vol. 18, no. 9, pages 1258–1267, September 2008.

[Ferguson *et al.* 2008a] D. Ferguson, M. Howard and M. Likhachev. *Motion planning in urban environments: Part II*. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 1070 –1076, September 2008.

[Ferguson *et al.* 2008b] D. Ferguson, T.M. Howard and M. Likhachev. *Motion planning in urban environments: Part I.* In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 1063 –1069, September 2008.

[Ferguson *et al.* 2008c] Dave Ferguson, Thomas M. Howard and Maxim Likhachev. *Motion planning in urban environments: Part II.* In IROS, pages 1070–1076, 2008.

[Fischler & Bolles 1981] M. A. Fischler and R. C. Bolles. *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography.* In ACM, June 1981.

[Fogel *et al.* 2012] Efi Fogel, Ron Wein, Baruch Zukerman and Dan Halperin. *2D Regularized Boolean Set-Operations*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Foote *et al.* 2012] Tully Foote, Eitan Marder-Eppstein and Wim Meeussen. *ROS Transforms*. `http://www.ros.org/wiki/tf`, May 2012.

[García *et al.* 2000] Miguel Angel García, Boris Xavier Vintimilla and Angel Domingo Sappa. *Approximation and Processing of Intensity Images with Dicontinuity-Preserving Adaptive Triangular Meshes*. In Proceedings of the 6th European Conference on Computer Vision-Part I, ECCV ’00, pages 844–855, 2000.

[Garland & Heckbert 1995] Michael Garland and Paul S. Heckbert. *Fast Polygonal Approximation of Terrains and Height Fields*. Rapport technique CMU-CS-95-181, September 1995.

[Geiger *et al.* 2012] Andreas Geiger, Philip Lenz and Raquel Urtasun. *Are we ready for Autonomous Driving?* In Computer Vision and Pattern Recognition (CVPR), June 2012.

[Gerkey *et al.* 2003] Brian P. Gerkey, Richard T. Vaughan and Andrew Howard. *The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems*. In In Proceedings of the 11th International Conference on Advanced Robotics, pages 317–323, 2003.

[Gerkey *et al.* 2012] Brian P. Gerkey, Jeremy Leibs and Blaise Gassend. *Hokuyo laser drivers*. `http://ros.org/wiki/hokuyo_node`, May 2012.

[Giezeman & Wesselink 2012a] Geert-Jan Giezeman and Wieger Wesselink. *2D Polygons*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Giezeman & Wesselink 2012b] Geert-Jan Giezeman and Wieger Wesselink. *2D Polygons*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Graham 1972] R L Graham. *An efficient algorithm for determining the convex hull of a finite planar set*. Information Processing Letters, vol. 1, no. 4, pages 132–133, 1972.

[Gregor *et al.* 2002] Rudolf Gregor, Michael Lützeler, Martin Pellkofer, Karl-Heinz Siedersberger and Ernst D. Dickmanns. *EMS-Vision: a perceptual system for autonomous vehicles*. IEEE Transactions on Intelligent Transportation Systems, vol. 3, no. 1, pages 48–59, 2002.

[Guerreiro & Aguiar 2011] R.F.C. Guerreiro and P.M.Q. Aguiar. *Incremental local Hough Transform for line segment extraction*. In Image Processing (ICIP), 2011 18th IEEE International Conference on, pages 2841 –2844, September 2011.

[Guo *et al.* 2009] Chunzhao Guo, S. Mita and D. McAllester. *Stereovision-based road boundary detection for intelligent vehicles in challenging scenarios*. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1723 –1728, October 2009.

[Hartenberg & Denavit 1955] R. S. Hartenberg and J. Denavit. *A kinematic notation for lower pair mechanisms based on matrices*. Journal of Applied Mechanics, vol. 77, pages 215–221, 1955.

[Hartenberg & Denavit 1964] R. S. Hartenberg and J. Denavit. Kinematic synthesis of linkages. McGraw-Hill, New York, 1964.

[Hasler & SÃijsstrunk 2004] David Hasler and Sabine SÃijsstrunk. *Mapping colour in image stitching applications*. Journal of Visual Communication and Image Representation, vol. 15, no. 12, pages 65–90, June 2004.

[He *et al.* 2008] Lifeng He, Yuyan Chao and K. Suzuki. *A Run-Based Two-Scan Labeling Algorithm*. Image Processing, IEEE Transactions on, vol. 17, no. 5, pages 749 –756, May 2008.

[Hermann *et al.* 2011] Simon Hermann, Sandino Morales and Reinhard Klette. *Half-Resolution Semi-Global Stereo Match*. In Proceedings of the 2011 IEEE Intelligent Vehicles Symposium (IEEE IV '11), pages 201–206, July 2011.

[Hershberger & Faust 2012] Dave Hershberger and Josh Faust. *RVIZ, a 3d visualization environment for robots using ROS*. http://ros.org/wiki/rviz, May 2012.

[Hert & Schirra 2012] Susan Hert and Stefan Schirra. *2D Convex Hulls and Extreme Points*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Hoppe 1996] Hugues Hoppe. *Progressive meshes*. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, SIGGRAPH '96, pages 99–108, 1996.

[Howard & Roy 2003] Andrew Howard and Nicholas Roy. *The Robotics Data Set Repository (Radish)*, 2003.

[Hsu *et al.* 2005] Chin-Hao Hsu, Zhih-Wei Chen and Cheng-Chin Chiang. *Region-based color correction of images*. In Proceedings of the third International Conference on Information Technology and Applications, pages 710–715, July 2005.

[Huang *et al.* 2010] A.S. Huang, E. Olson and D.C. Moore. *LCM: Lightweight Communications and Marshalling*. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pages 4057 –4062, October 2010.

[Huang *et al.* 2011] Albert S. Huang, Matthew Antone, Edwin Olson, Luke Fletcher, David Moore, Seth Teller and John Leonard. *A High-rate, Heterogeneous Data Set from the DARPA Urban Challenge*. International Journal of Robotics Research, vol. 29, no. 13, pages 1595–1601, 2011.

[Huang 2012] Albert Huang. *Camunits*. `http://code.google.com/p/camunits/`, May 2012.

[Instituto Nacional de Estatística 2011] Instituto Nacional de Estatística. *Estatísticas dos Trasportes 2010.* , February 2011.

[Jarvis 1973] R. A. Jarvis. *On the Identification of the Convex Hull of a Finite Set of Points in the Plane*. Information Processing Letters, vol. 2, no. 1, pages 18–21, March 1973.

[Javier Civera & Montiel 2009] Juan A. Magallon Javier Civera Andrew J. Davison and J. M. M. Montiel. International Journal of Computer Vision, vol. 81, no. 2, pages 128–137, February 2009.

[Jia & Tang 2003] Jiaya Jia and Chi K. Tang. *Image registration with global and local luminance alignment*. In Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 1, pages 156–163, Washington, DC, USA, October 2003.

[Jia & Tang 2005] Jiaya Jia and Chi K. Tang. *Tensor voting for image correction by global and local intensity alignment*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 1, pages 36–50, January 2005.

[Jovanovic & Lorentz 2011] R. Jovanovic and R. Lorentz. *Compression of volumetric data using 3D Delaunay triangulation*. In Modeling, Simulation and Applied Optimization (ICMSAO), 2011 4th International Conference on, pages 1 –5, April 2011.

[Kai & Da 2012] Tran Kai and Frank Da. *2D Alpha Shapes*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Kai *et al.* 2012] Tran Kai, Frank Da and Mariette Yvinec. *3D Alpha Shapes*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Kallay 1984] Michael Kallay. *The Complexity of Incremental Convex Hull Algorithms in R^d*. Information Processing Letters, vol. 19, no. 4, page 197, 1984.

[Kamat & Ganesan 1998] V. Kamat and S. Ganesan. *A robust Hough transform technique for description of multiple line segments in an image*. In Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on, volume 1, pages 216 –220 vol.1, October 1998.

[Kamberov & Kamberova 2007] G. Kamberov and G. Kamberova. *Geometric Integrability and Consistency of 3D Point Clouds*. In Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, pages 1 –6, October 2007.

[Kanade *et al.* 1986] Takeo Kanade, Charles E. Thorpe, Steven A. Shafer and Martial Hebert. *Carnegie Mellon Navlab Vision System*. In Louis O. Hertzberger and Frans C. A. Groen, editeurs, Intelligent Autonomous Systems, An International Conference, Amsterdam, The Netherlands, 8-11 December 1986, pages 681–693. North-Holland, 1986.

[Kastrinaki *et al.* 2003] V. Kastrinaki, M. Zervakis and K. Kalaitzakis. *A survey of video processing techniques for traffic applications*. Image and Vision Computing, vol. 21, pages 359–381, 2003.

[Kazhdan *et al.* 2006] Michael Kazhdan, Matthew Bolitho and Hugues Hoppe. *Poisson surface reconstruction*. In SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing, pages 61–70. Eurographics Association, 2006.

[Keyes 2011] Bradley Keyes. *A brief History of Driverless Vehicles*. `http://driverlessworld.com/2011/06/a-brief-history-of-driverless-vehicles/`, June 2011.

[Khosla 2002] D. Khosla. *Accurate estimation of forward path geometry using two-clothoid road model*. In Intelligent Vehicle Symposium, 2002. IEEE, volume 1, pages 154 – 159 vol.1, june 2002.

[Kim & Pollefeys 2008] Seon Joo Kim and Marc Pollefeys. *Robust radiometric calibration and vignetting correction*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 4, pages 562–576, April 2008.

[Klasing *et al.* 2009] K. Klasing, D. Althoff, D. Wollherr and M. Buss. *Comparison of surface normal estimation methods for range sensing applications*. In Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, pages 3206 –3211, May 2009.

[Koenig & Howard 2012] Nate Koenig and Andrew Howard. *Gazebo*. `http://www.ros.org/wiki/gazebo`, May 2012.

[Krein & Smulian 1940] M. Krein and Smulian. *On regularly convex sets in the space conjugate to a Banach space*. Annals of Mathematics, Second series, vol. 41, pages 556–583, 1940.

[Kumpakeaw & Dillmann 2007] Saman Kumpakeaw and Rüdiger Dillmann. *Semantic Road Maps for Autonomous Vehicles*. In AMS, pages 205–211, 2007.

[Labayrade & Aubert 2003] R. Labayrade and D. Aubert. *A Single Framework for Vehicle Roll, Pitch, Yaw Estimation and Obstacles Detection by Stereovision*. In Proceedings of the IEEE Intelligent Vehicles Symposium, pages 31–36, Columbus, OH, USA, June 2003.

[Labayrade *et al.* 2005] R. Labayrade, C. Royere, D. Gruyer, and D. Aubert. *Cooperative Fusion for Multi-Obstacles Detection With Use of Stereovision and Laser Scanner*. Image Processing, IEEE Transactions on, vol. 19, pages 117 –140, 2005.

[Lalonde *et al.* 2005] J.F. Lalonde, R. Unnikrishnan, N. Vandapel and M. Hebert. *Scale selection for classification of point-sampled 3D surfaces*. In 3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on, pages 285 – 292, June 2005.

[Laumond 1998] J. P. Laumond. *Advanced holonomic and non-holonomic planning*. Robot Motion Planning and Control, Springer, 1998.

[Lehner *et al.* 2007] Burkhard Lehner, Georg Umlauf and Bernd Hamann. *Survey of techniques for data-dependent triangulations*. In H. Hagen, M. Hering-Bertram and C. Garth, editeurs, GI Lecture Notes in Informatics, Visualization of Large and Unstructured Data Sets, pages 178–187, 2007.

[Lempitsky & Ivanov 2007] V. Lempitsky and D. Ivanov. *Seamless mosaicing of image-based texture maps*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–6, June 2007.

[Levin *et al.* 2003] Anat Levin, Assaf Zomet, Shmuel Peleg and Yair Weiss. *Seamless image stitching in the gradient domain*. In Proceedings of the Eighth European Conference on Computer Vision, pages 377–389, May 2003.

[Li & Hai 2011] S. Li and Y. Hai. *Easy Calibration of a Blind-Spot-Free Fisheye Camera System Using a Scene of a Parking Space*. IEEE Trans. on Intelligent Transportation Systems, vol. 12, no. 1, pages 232–242, 2011.

[Li *et al.* 2004] Yin Li, Heung-Yeung Shum, Chi-Keung Tang and R. Szeliski. *Stereo reconstruction from multiperspective panoramas*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 1, pages 45–62, January 2004.

[Li *et al.* 2008] Yanfang Li, Yaming Wang, Wenqing Huang and Zuoli Zhang. *Automatic image stitching using SIFT*. In Proceedings of the International Conference on Audio, Language and Image Processing, pages 568 –571, July 2008.

[Likhachev & Ferguson 2009] Maxim Likhachev and Dave Ferguson. *Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles*. I. J. Robotic Res., vol. 28, no. 8, pages 933–945, 2009.

[Lina María Paz & Neira 2007] J. D. Tardós Lina María Paz José Guivant and J. Neira, June 2007.

[Lina María Paz & Neira 2008] J. D. Tardós Lina María Paz and J. Neira. IEEE Transactions on Robotics, vol. 24, no. 5, pages 1107–1120, October 2008.

[Litvinov & Schechner 2005a] A. Litvinov and Y.Y. Schechner. *Addressing radiometric nonidealities: a unified framework*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 52–59, June 2005.

[Litvinov & Schechner 2005b] Anatoly Litvinov and Yoav Y. Schechner. *Radiometric framework for image mosaicking*. Journal of the Optical Society of America, vol. 22, no. 5, pages 839–848, May 2005.

[Luo *et al.* 2009] LinBo Luo, InSung Koh, SangYoon Park, ReSen Ahn and JongWha Chong. *A software-hardware cooperative implementation of bird's-eye view system for camera-on-vehicle*. In IEEE International Conference on Network Infrastructure and Digital Content, pages 963 –967, November 2009.

[Ma *et al.* 2007] Guanglin Ma, Su-Birm Park, S. Miiller-Schneiders, A. Ioffe and A. Kummert. *Vision-based pedestrian detection -reliable pedestrian candidate detection by combining IPM and a 1D profile*. In IEEE Intelligent Transportation Systems Conference, pages 137 –142, October 2007.

[Mallot *et al.* 1991] H. Mallot, H. Bülthoff, J. Little and S. Bohrer. *Inverse perspective mapping simplifies optical flow computation and obstacle detection*. Biological Cybernetics, vol. 64, no. 3, pages 177–185, 1991.

[Mann & Mann 2001] S. Mann and R. Mann. *Quantigraphic imaging: estimating the camera response and exposures from differently exposed images*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, volume 1, pages 842–849, June 2001.

[Marton *et al.* 2009] Zoltan Csaba Marton, Radu Bogdan Rusu and Michael Beetz. *On Fast Surface Reconstruction Methods for Large and Noisy Datasets*. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), May 2009.

[Maslennikova & Vezhnevets 2007] A. Maslennikova and V. Vezhnevets. *Interactive local color transfer between images*. GraphiCon, June 2007.

[Mathibela *et al.* 2012] B Mathibela, Michael A Osborne, I Posner and P.M. Newman. *Can priors be trusted? Learning to anticipate roadworks*, September 2012.

[Mathworks 2012] Mathworks. *Fitting an Orthogonal Regression Using Principal Components Analysis*. http://www.mathworks.com/products/statistics/demos.html?file=/products/demos/shipping/stats/orthoregdemo.html, May 2012.

[McCall & Trivedi 2005] J.C. McCall and M.M. Trivedi. *Performance evaluation of a vision based lane tracker designed for driver assistance systems*. In Intelligent Vehicles Symposium, 2005. Proceedings. IEEE, pages 153 – 158, june 2005.

[McCall *et al.* 2004] J.C. McCall, O. Achler and M.M. Trivedi. *Design of an instrumented vehicle test bed for developing a human centered driver support system*. In IEEE Intelligent Vehicles Symposium, pages 483 – 488, June 2004.

[Meeussen *et al.* 2012] Wim Meeussen, John Hsu and Rosen Diankov. *Unified Robot Description Format*. http://www.ros.org/wiki/urdf, May 2012.

[Meeussen 2012] Wim Meeussen. *Robot State Pulisher*. http://www.ros.org/wiki/robot_state_publisher, May 2012.

[Micusik & Kosecka 2009] B. Micusik and J. Kosecka. *Piecewise planar city 3D modeling from street view panoramic sequences*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2906–2912, June 2009.

[Mihelich *et al.* 2012a] Patrick Mihelich, Kurt Konolige and Jeremy Leibs. *Image processing*. http://www.ros.org/wiki/image_proc, May 2012.

[Mihelich *et al.* 2012b] Patrick Mihelich, Kurt Konolige and Jeremy Leibs. *Stereo image processing*. http://www.ros.org/wiki/stereo_image_proc, May 2012.

[Miller *et al.* 2009] Isaac Miller, Mark Campbell, Dan Huttenlocher, Aaron Nathan, Frank-Robert Kline, Pete Moran, Noah Zych, Brian Schimpf, Sergei Lupashin, Ephrahim Garcia, Jason Catlin, Mike Kurdziel and Hikaru Fujishima. *Team Cornell's Skynet: Robust Perception and Planning in an Urban Environment*. In Martin Buehler, Karl Iagnemma and Sanjiv Singh, editeurs, The DARPA Urban Challenge, volume 56, chapitre 7, pages 257–304. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Mitra *et al.* 2004] N. J. Mitra, A. Nguyen and L. Guibas. *Estimating Surface Normals in Noisy Point Cloud Data*. In special issue of International Journal of Computational Geometry and Applications, volume 14, pages 261–276, 2004.

[Mitsunaga & Nayar 1999] T. Mitsunaga and S.K. Nayar. *Radiometric self calibration*. In Proceedings the IEEE Conference on Computer Vision and Pattern Recognition, volume 1, page 380, September 1999.

[Moller 1997] Tomas Moller. *A Fast Triangle-Triangle Intersection Test*. Journal of Graphics Tools, vol. 2, pages 25–30, 1997.

[Montemerlo *et al.* 2003a] M. Montemerlo, N. Roy and S. Thrun. *Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) Toolkit*. In Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, volume 3, pages 2436–2441, October 2003.

[Montemerlo *et al.* 2003b] Michael Montemerlo, Nicholas Roy and Sebastian Thrun. *Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit*. In In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS, pages 2436–2441, 2003.

[Montemerlo *et al.* 2006] Michael Montemerlo, Sebastian Thrun, Hendrik Dahlkamp, David Stavens and Sven Strohband. *Winning the DARPA Grand Challenge with an AI Robot*. In AAAI, pages 982–987, 2006.

[Montemerlo *et al.* 2008] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt and S. Thrun. *Junior: The Stanford Entry in the Urban Challenge*. Journal of Field Robotics, 2008.

[Muad *et al.* 2004] A.M. Muad, A. Hussain, S.A. Samad, M.M. Mustaffa and B.Y. Majlis. *Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system*. In IEEE Region 10 Conference TENCON, volume A, pages 207 – 210 Vol. 1, November 2004.

[Muja & Lowe 2009] Marius Muja and David G. Lowe. *Fast approximate nearest neighbors with automatic algorithm configuration*. In In VISAPP International Conference on Computer Vision Theory and Applications, pages 331–340, 2009.

[Nayak & Chaudhuri 2006] Arvind Nayak and Subhasis Chaudhuri. *Automatic illumination correction for scene enhancement and object tracking*. Image and Vision Computing, vol. 24, no. 9, pages 949–959, September 2006.

[Neira & Trinkle 2009] J. Neira and J. Trinkle. Autonomous Robots, vol. 26, pages 99–101, April 2009.

[Newman 2003] P. M. Newman. *MOOS - A Mission Oriented Operating Suite*. Rapport technique OE2003-07, MIT Dept. of Ocean Engineering, 2003.

[Nieto *et al.* 2007] M. Nieto, L. Salgado, F. Jaureguizar and J. Cabrera. *Stabilization of Inverse Perspective Mapping Images based on Robust Vanishing Point Estimation*. In IEEE Intelligent Vehicles Symposium, pages 315 –320, June 2007.

[Nuchter *et al.* 2007] A. Nuchter, K. Lingemann and J. Hertzberg. *Cached k-d tree search for ICP algorithms*. In 3-D Digital Imaging and Modeling, 2007. 3DIM '07. Sixth International Conference on, pages 419 –426, August 2007.

[Oliveira & Santos 2007] Miguel Oliveira and Vitor Santos. *A vision-based solution for the navigation of the Atlas autonomous robots*. In Portuguese Robotics Open 2007, International Conference on Autonomous Robot Systems and Competitions, April 2007. Selected for publication in Robótica (ISSN 0874-9019), published in number 69, October 2007.

[Oliveira & Santos 2008a] M. Oliveira and V. Santos. *Real Time Extraction of Road Border Lines using Simple Statistical Descriptors*. In In 2nd Workshop on Planning, Perception and Navigation for Intelligent Vehicles, 2008 Intelligent Robots Systems, IEEE International Conference on, September 2008.

[Oliveira & Santos 2008b] Miguel Oliveira and Vitor Santos. *Multi-Camera Active Perception System with Variable Image Perspective for Mobile Robot Navigation*. In Portuguese Robotics Open 2008, International Conference on Autonomous Robot Systems and Competitions, April 2008. Best conference paper award. Selected for publication in Robótica (ISSN 0874-9019), published in number 76, September 2009.

[Oliveira & Santos 2008c] Miguel Oliveira and Vitor Santos. *Real-time Extraction of Road Border Lines using Simple Statistical Descriptors*. In IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles, September 2008.

[Oliveira & Santos 2011] Miguel Oliveira and Vitor Santos. *Autonomous Driving Competition: Perception Approaches used in the ATLAS Project*. In Portuguese Robotics Open 2011, International Conference on Autonomous Robot Systems and Competitions, April 2011.

[Oliveira *et al.* 2005] Miguel Oliveira, Rui Cancela, Miguel Neta and Vitor Santos. *ATLAS: Robô com visão orientado para provas de condução autónoma*, April 2005. Selected for publication in Robótica (ISSN 0874-9019), published in number 62, January 2006.

[Oliveira *et al.* 2011] Miguel Oliveira, Angel D. Sappa and Vitor Santos. *Unsupervised local color transfer for coarsely registered images*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 201–208, June 2011.

[Oliveira *et al.* 2012] Miguel Oliveira, Vitor Santos and Angel D. Sappa. *Short term path planning using a multiple hypothesis evaluation approach for an autonomous driving competition*. In IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles, October 2012.

[Oniga & Nedevschi 2010] F. Oniga and S. Nedevschi. *Processing Dense Stereo Data Using Elevation Maps: Road Surface, Traffic Isle, and Obstacle Detection*. Vehicular Technology, IEEE Transactions on, vol. 59, no. 3, pages 1172 –1182, March 2010.

[Opengl *et al.* 2005] Opengl, D. Shreiner, M. Woo, J. Neider and T. Davis. OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition). Addison-Wesley Professional, August 2005.

[O'Quin & Tossell 2012] Jack O'Quin and Ken Tossell. *Camera1394*. http://www.ros.org/wiki/camera1394, May 2012.

[Patel *et al.* 2005] Kayur Patel, Walter Macklem, Sebastian Thrun and Michael Montemerlo. *Active Sensing for High-Speed Offroad Driving*. In ICRA, pages 3162–3168, 2005.

[Piniés & Tardós 2008] Pedro Piniés and J. D. Tardós. IEEE Transactions on Robotics, vol. 24, no. no. 5, pages 1094–1106, October 2008.

[Pitie *et al.* 2005] Francois Pitie, Anil C. Kokaram and Rozenn Dahyot. *N-dimensional probablility density function transfer and its application to colour transfer*. Proceedings of the Eleventh IEEE International Conference on Computer Vision, vol. 2, pages 1434–1439, October 2005.

[Pitie *et al.* 2007] Francois Pitie, Anil C. Kokaram and Rozenn Dahyot. *Automated colour grading using colour distribution transfer*. Computer Vision and Image Understanding, vol. 107, no. 1-2, pages 123–137, July 2007. Special issue on color image processing.

[Pomerleau 1995] D. Pomerleau. *RALPH: rapidly adapting lateral position handler*. In Intelligent Vehicles '95 Symposium., Proceedings of the, pages 506 –511, September 1995.

[PRO 2001] PRO. *First Portuguese Robotics Open*. http://www.robotica.dei.uminho.pt/robotica2001/, April 2001.

[PRO 2002] PRO. *Second Portuguese Robotics Open*. http://robotica.ua.pt/robotica2002/index1.htm, April 2002.

[PRO 2003] PRO. *Third Portuguese Robotics Open*. http://robotica2003.ist.utl.pt/, April 2003.

[PRO 2004] PRO. *Fourth Portuguese Robotics Open*. , April 2004.

[PRO 2005] PRO. *Fifth Portuguese Robotics Open*. http://robotics.dem.uc.pt/web/, April 2005.

[PRO 2006] PRO. *Sixth Portuguese Robotics Open*. http://www.robotica2006.dei.uminho.pt/robotica2006/, April 2006.

[PRO 2007] PRO. *Seventh Portuguese Robotics Open*. http://www.ccvalg.pt/robotica2007/, April 2007.

[PRO 2008] PRO. *Eight Portuguese Robotics Open*. http://robotica.ua.pt/robotica2008/, April 2008.

[PRO 2009] PRO. *Ninth Portuguese Robotics Open*. http://www.est.ipcb.pt/robotica2009/, April 2009.

[PRO 2010] PRO. *Tenth Portuguese Robotics Open*. http://robotica2010.ipleiria.pt/robotica2010/, April 2010.

[PRO 2011] PRO. *Eleventh Portuguese Robotics Open*. http://robotica2011.ist.utl.pt/, April 2011.

[PRO 2012] PRO. *Twelfth Portuguese Robotics Open*. http://www.robotica2012.org/, April 2012.

[Qian *et al.* 2010] XiaoYan Qian, BangFeng Wang and Lei Han. *An efficient fuzzy clustering-based color transfer method*. In Proceedings of the Seventh International Conference of Fuzzy Systems and Knowledge Discovery, pages 520–523, August 2010.

[Reinhard *et al.* 2001] Erik Reinhard, Michael Ashikhmin, Bruce Gooch and Peter Shirley. *Color transfer between Images*. IEEE Computer Graphics and Applications, vol. 21, pages 34–41, October 2001.

[Rippa 1992] Shmuel Rippa. *Long and Thin Triangles Can be Good for Linear Interpolation*. SIAM Journal on Numerical Analysis, vol. 29, no. 1, pages pp. 257–270, 1992.

[Rivadeneyra *et al.* 2009] C. Rivadeneyra, I. Miller, J.R. Schoenberg and M. Campbell. *Probabilistic estimation of Multi-Level terrain maps*. In Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, pages 1643 –1648, May 2009.

[Rocha 2011] Tiago Rocha. Piloto automático para controlo e manobras de navegação do atlascar. Master's thesis, University of Aveiro, 2011.

[Rusu & Cousins 2011] Radu Bogdan Rusu and Steve Cousins. *3D is here: Point Cloud Library (PCL)*. In IEEE International Conference on Robotics and Automation (ICRA), May 2011.

[Rusu 2009] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische UniversitÃć Munchen, Germany, October 2009. Advisor: Univ.-Prof. Michael Beetz (TUM) Ph.D.; Committee: Univ.-Prof. Dr. Nassir Navab (TUM), Univ.-Prof. Michael Beetz (TUM) Ph.D., Prof. Kurt Konolige (Stanford) Ph.D., Prof. Gary Bradski (Stanford) Ph.D.; summa cum laude.

[Sajadi *et al.* 2010] Behzad Sajadi, Maxim Lazarov and Aditi Majumder. *ADICT: Accurate Direct and Inverse Color Transformation.* In European Conference on Computer Vision, pages 72–86, September 2010.

[Samet & Tamminen 1988] H. Samet and M. Tamminen. *Efficient component labeling of images of arbitrary dimension represented by linear bintrees*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 10, no. 4, pages 579 –586, July 1988.

[Santos *et al.* 2010] V. Santos, J. Almeida, E. Avila, D. Gameiro, M. Oliveira, R. Pascoal, R. Sabino and P. Stein. *ATLASCAR - technologies for a computer assisted driving system on board a common automobile*. In 13th International IEEE Conference on Intelligent Transportation Systems, pages 1421 –1427, September 2010.

[Sappa & García 2000] Angel Domingo Sappa and Miguel Angel García. *Incremental Multiview Integration of Range Images*. In ICPR, pages 1546–1549, 2000.

[Sappa & García 2007] Angel Domingo Sappa and Miguel Angel García. *Coarse-to-fine approximation of range images with bounded error adaptive triangular meshes*. J. Electronic Imaging, vol. 16, no. 2, page 23010, 2007.

[Sappa *et al.* 2008] A. Sappa, F. Dornaika, D. Ponsa, D. Gerónimo and A. López. *An efficient approach to on-board stereo vision system pose estimation*. IEEE Trans. on Intelligent Transportation Systems, vol. 9, no. 3, pages 476–490, 2008.

[Schätzl *et al.* 2001] René Schätzl, Hans Hagen, James C. Barnes, Bernd Hamann and Kenneth I. Joy. *Data-Dependent Triangulation in the Plane with Adaptive Knot Placement*. In Geometric Modelling, pages 309–321, 2001.

[Schauwecker *et al.* 2011] Konstantin Schauwecker, Sandino Morales, Simon Hermann and Reinhard Klette. *A Comparative Study of Stereo-Matching Algorithms for Road-Modeling in the Presence of Windscreen Wipers*. In Proceedings of the 2011 IEEE Intelligent Vehicles Symposium (IEEE IV '11), pages 7–12, June 2011.

[Schneider & Rolf 1993] B. Schneider and M. Rolf. *Convex bodies: The Brunn-Minkowski theory*. Encyclopedia of mathematics and its applications, vol. 41, pages 126–196, 1993.

[Schumaker 1993] Larry L. Schumaker. *Computing optimal triangulations using simulated annealing*. Computer Aided Geometric Design, vol. 10, no. 3-4, pages 329–345, 1993.

[Segal *et al.* 1992] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran and Paul Haeberli. *Fast shadows and lighting effects using texture mapping*. SIGGRAPH Comput. Graph., vol. 26, no. 2, pages 249–252, July 1992.

[Shewchuk 2008] Jonathan Richard Shewchuk. *General-Dimensional Constrained Delaunay and Constrained Regular Triangulations I: Combinatorial Properties*. Discrete Comput. Geom., vol. 39, no. 1, pages 580–637, March 2008.

[Siddiqui & Bouman 2008] H. Siddiqui and C.A. Bouman. *Hierarchical color correction for camera cell phone images*. IEEE Transactions on Image Processing, vol. 17, no. 11, pages 2138 – 2155, November 2008.

[Silpa-Anan & Hartley 2008] C. Silpa-Anan and R. Hartley. *Optimised KD-trees for fast image descriptor matching*. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pages 1 –8, June 2008.

[Simmons & Apfelbaum 1998] R. Simmons and D. Apfelbaum. *A task description language for robot control*. In Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on, volume 3, pages 1931 –1937 vol.3, October 1998.

[Simpson *et al.* 2011] R. Simpson, J. Cullip and J. Revell. *The Cheddar George data set*, January 2011.

[Smith *et al.* 2009] M. Smith, I. Baldwin, W. Churchill, R. Paul and P. Newman. *The new college vision and laser data set*. The International Journal of Robotics Research, vol. 28, no. 5, pages 595–599, May 2009.

[Soille 2006] P. Soille. *Morphological image compositing*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 5, pages 673–683, May 2006.

[Sotelo *et al.* 2004] Miguel Angel Sotelo, Francisco Javier Rodriguez, Luis Magdalena, Luis Miguel Bergasa and Luciano Boquete. *A Color Vision-Based Lane Tracking System for Autonomous Driving on Unmarked Roads*. Auton. Robots, vol. 16, no. 1, pages 95–116, January 2004.

[Specht & Devy 2004] A. Specht and M. Devy. *Surface segmentation using a modified ball-pivoting algorithm*. In Image Processing, 2004. ICIP '04. 2004 International Conference on, volume 3, pages 1931 – 1934 Vol. 3, October 2004.

[Stavens *et al.* 2007] David Stavens, Gabriel Hoffmann and Sebastian Thrun. *Online Speed Adaptation Using Supervised Learning for High-Speed, Off-Road Autonomous Driving*. In IJCAI, pages 2218–2224, 2007.

[Svalbe 1989] I.D. Svalbe. *Natural representations for straight lines and the Hough transform on discrete arrays*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 11, no. 9, pages 941 –950, September 1989.

[Tai *et al.* 2005] Yu W. Tai, Jiaya Jia and Chi K. Tang. *Local color transfer via probabilistic segmentation by expectation-maximization*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 747–754, June 2005.

[Tai *et al.* 2007] Yu Wing Tai, Jiaya Jia and Chi-Keung Tang. *Soft color segmentation and its applications*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 9, pages 1520 –1537, September 2007.

[Tan *et al.* 2006] S. Tan, J. Dale, A. Anderson and A. Johnston. *Inverse perspective mapping and optic flow: A calibration method and a quantitative analysis*. Image Vision Computing, vol. 24, no. 2, pages 153–165, 2006.

[Thomanek & Dickmanns 1995] Frank Thomanek and Ernst D. Dickmanns. *Autonomous Road Vehicle Guidance in Normal Traffic*. In ACCV, pages 499–507, 1995.

[Thorpe *et al.* 1988] Chuck Thorpe, Martial Hebert, Takeo Kanade and Steven Shafer. *Vision and navigation for the Carnegie-Mellon Navlab*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, no. 3, pages 362 – 373, May 1988.

[Thorpe *et al.* 1991] Chuck Thorpe, Martial Hebert, Takeo Kanade and Steven Shafer. *Toward Autonomous Driving: The CMU Navlab. Part I: Perception*. IEEE Expert, vol. 6, no. 4, pages 31 – 42, August 1991.

[Thrun *et al.* 2006a] S. Thrun, M. Montemerlo and A. Aron. Probabilistic terrain analysis for high-speed desert driving, pages 21–28. University of Pennsylvania, Philadelphia, PA, 2006.

[Thrun *et al.* 2006b] Sebastian Thrun, Michael Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia M. Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary R. Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara V. Nefian and Pamela Mahoney. *Stanley: The robot that won the DARPA Grand Challenge*. J. Field Robotics, vol. 23, no. 9, pages 661–692, 2006.

[Tian *et al.* 2002] Gui Yun Tian, Duke Gledhill, Dave Taylor and David Clarke. *Colour correction for panoramic imaging*. In Proceedings of the International Conference on Information Visualisation, pages 483–488, November 2002.

[Tsin *et al.* 2001] Y. Tsin, V. Ramesh and T. Kanade. *Statistical calibration of CCD imaging process*. In Proceedings of the Eighth IEEE International Conference on Computer Vision, volume 1, pages 480–487 vol.1, 2001.

[Tuohy *et al.* 2010] S. Tuohy, D. O'Cualain, E. Jones and M. Glavin. *Distance determination for an automobile environment using Inverse Perspective Mapping in OpenCV*. In Signals and Systems Conference, pages 100–105, June 2010.

[United Nations & World Health Organization 2011] United Nations & World Health Organization. *Global Plan for the Decade of Action for Road Safety 2011-2020*. http://www.who.int/roadsafety/decade_of_action/plan/en/index.html, May 2011.

[United Nations 2010] United Nations. *Resolution 64/255*. http://www.who.int/violence_injury_prevention/publications/road_traffic/UN_GA_resolution-54-255-en.pdf, March 2010.

[Urmson & Whittaker 2008] Christopher Urmson and William (Red) L. Whittaker. *Self-Driving Cars and the Urban Challenge*. IEEE Intelligent Systems, vol. 23, no. 2, pages 66–68, April 2008.

[Urmson *et al.* 2004] Christopher Urmson, Joshua Anhalt, Michael Clark, Tugrul Galatali, Juan Pablo Gonzalez, Jay Gowdy, Alexander Gutierrez, Sam Harbaugh, Matthew Johnson-Roberson, Hiroki Kato, Phillip L Koon, Kevin Peterson, Bryon K Smith, Spencer Spiker, Erick Tryzelaar and William (Red) L. Whittaker. *High Speed Navigation of Unrehearsed Terrain: Red Team Technology for Grand Challenge 2004*. Rapport technique CMU-RI-TR-04-37, Robotics Institute, Pittsburgh, PA, June 2004.

[Urmson *et al.* 2006] Christopher Urmson, Joshua Anhalt, Daniel Bartz, Michael Clark, Tugrul Galatali, Alexander Gutierrez, Sam Harbaugh, Joshua Johnston, Hiroki Kato, Phillip L Koon, William Messner, Nick Miller, Aaron Mosher, Kevin Peterson, Charlie Ragusa, David Ray, Bryon K Smith, Jarrod M Snider, Spencer Spiker, Joshua C Struble, Jason Ziglar and William (Red) L. Whittaker. *A Robust Approach to High-Speed Navigation for Unrehearsed Desert Terrain*. Journal of Field Robotics, vol. 23, no. 8, pages 467–508, August 2006.

[Urmson *et al.* 2007] Christopher Urmson, Joshua Anhalt, J. Andrew (Drew) Bagnell, Christopher R. Baker , Robert E Bittner, John M Dolan, David Duggins, David Ferguson, Tugrul Galatali, Hartmut Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Alonzo Kelly, David Kohanbash, Maxim Likhachev, Nick Miller, Kevin Peterson, Ragunathan Rajkumar, Paul Rybski, Bryan Salesky, Sebastian Scherer, Young-Woo Seo, Reid Simmons, Sanjiv Singh, Jarrod M Snider, Anthony (Tony) Stentz, William (Red) L. Whittaker and Jason Ziglar.

Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge. Rapport technique CMU-RI-TR-, Robotics Institute, http://archive.darpa.mil/grandchallenge/, April 2007.

[Urmson *et al.* 2008] Christopher Urmson, Joshua Anhalt, Hong Bae, J. Andrew (Drew) Bagnell, Christopher R. Baker , Robert E Bittner, Thomas Brown, M. N. Clark, Michael Darms, Daniel Demitrish, John M Dolan, David Duggins, David Ferguson, Tugrul Galatali, Christopher M Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Sascha Kolski, Maxim Likhachev, Bakhtiar Litkouhi, Alonzo Kelly, Matthew McNaughton, Nick Miller, Jim Nickolaou, Kevin Peterson, Brian Pilnick, Ragunathan Rajkumar, Paul Rybski, Varsha Sadekar, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod M Snider, Joshua C Struble, Anthony (Tony) Stentz, Michael Taylor , William (Red) L. Whittaker, Ziv Wolkowicki, Wende Zhang and Jason Ziglar. *Autonomous driving in urban environments: Boss and the Urban Challenge*. Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I, vol. 25, no. 8, pages 425–466, June 2008.

[Urmson *et al.* 2009] Christopher Urmson, Christopher R. Baker , John M Dolan, Paul Rybski, Bryan Salesky, William (Red) L. Whittaker, David Ferguson and Michael Darms. *Autonomous Driving in Traffic: Boss and the Urban Challenge*. AI Magazine, vol. 30, no. 2, pages 17–29, June 2009.

[Vacek *et al.* 2007] Stefan Vacek, Tobias Gindele, Johann Marius Zöllner and Rüdiger Dillmann. *Using case-based reasoning for autonomous vehicle guidance*. In IROS, pages 4271–4276, 2007.

[Velodyne 2012] Velodyne. *Velodyne HDL64 homepage*. `http://velodynelidar.com/lidar/hdlproducts/hdl64e.aspx`, May 2012.

[Vincent 2007] L. Vincent. *Taking online maps down to street level*. Computer, vol. 40, no. 12, pages 118–120, December 2007.

[Wang *et al.* 2004] Chieh-Chih Wang, David Duggins, Jay Gowdy, John Kozar, Robert MacLachlan, Christoph Mertz, Arne Suppe and Charles Thorpe. *Navlab SLAMMOT Datasets*. www.cs.cmu.edu/˜bobwang/datasets.html, May 2004. Carnegie Mellon University.

[Wang *et al.* 2007] Sen Wang, Yang Wang, Miao Jin, X.D. Gu and D. Samaras. *Conformal Geometry and Its Applications on 3D Shape Matching, Recognition, and Stitching*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 29, no. 7, pages 1209 –1220, July 2007.

[Wang *et al.* 2011] Miao Wang, Tinosch Ganjineh and Raúl Rojas. *Action annotated trajectory generation for autonomous maneuvers on structured road networks*. In ICARA, pages 67–72, 2011.

[Weiss *et al.* 2007] T. Weiss, B. Schiele and K. Dietmayer. *Robust Driving Path Detection in Urban and Highway Scenarios Using a Laser Scanner and Online Occupancy Grids*. In Intelligent Vehicles Symposium, 2007 IEEE, pages 184 –189, June 2007.

[Weisstein 2012a] Eric W. Weisstein. *Plane*. `http://mathworld.wolfram.com/Plane.html`, May 2012.

[Weisstein 2012b] Eric W. Weisstein. *Point-Plane Distance*. `http://mathworld.wolfram.com/Point-PlaneDistance.html`, May 2012.

[Wikipedia 2004] Wikipedia. *Darpa Grand Challenge — Wikipedia, The Free Encyclopedia*, 2004. [Online; accessed 22-July-2004].

[World Health Organization 2004] World Health Organization. *World report on road traffic injury prevention: summary*. `http://www.who.int/violence_injury_prevention/publications/road_traffic/world_report/summary_en_rev.pdf`, May 2004.

[Xiang *et al.* 2009] Yao Xiang, Beiji Zou and Hong Li. *Selective color transfer with multi-source images*. Pattern Recognition Letters, vol. 30, no. 7, pages 682–689, February 2009.

[Xiao & Ma 2006] Xuezhong Xiao and Lizhuang Ma. *Color transfer in correlated color space*. In Proceedings of the ACM International Conference on Virtual Reality Continuum and its Applications, pages 305–309, June 2006.

[Xu & Mulligan 2010] Wei Xu and J. Mulligan. *Performance evaluation of color correction approaches for automatic multi-view image and video stitching*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 263–270, June 2010.

[Yin & Cooperstock 2004] Jianfeng Yin and Jeremy R. Cooperstock. *Color correction methods with applications to digital projection environments*. Journal of WSCG, vol. 12, pages 1–3, February 2004.

[Yin *et al.* 2010] Chen Yin, Dang Gang, Cheng Zhi-quan, Li Hong-hua, Li Jun and Jin Shi-yao. *An algorithm of CUDA-based Poisson surface reconstruction*. In Audio Language and Image Processing (ICALIP), 2010 International Conference on, pages 203 –207, November 2010.

[Yu *et al.* 2004] Wonpil Yu, Yunkoo Chung and Jung Soh. *Vignetting distortion correction method for high quality digital imaging*. In Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on, volume 3, pages 666 – 669 Vol.3, August 2004.

[Yvinec 2012] Mariette Yvinec. *2D Triangulations*. In CGAL User and Reference Manual. CGAL Editorial Board, 4.0 édition, 2012.

[Zaman *et al.* 2011] S. Zaman, W. Slany and G. Steinbauer. *ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues*. In Electronics, Communications and Photonics Conference (SIECPC), 2011, pages 1–5, April 2011.

[Zhang & Georganas 2004] Maojun Zhang and Nicolas D. Georganas. *Fast color correction using principal regions mapping in different color spaces*. Real-Time Imaging, vol. 10, no. 1, pages 23–30, February 2004.

[Zhang & Wandell 1997] X. Zhang and B. A. Wandell. *A spatial extension of CIELAB for digital color-image reproduction*. Journal of the Society for Information Display, vol. 5, no. 1, pages 61–63, March 1997.

[Zheng *et al.* 2008] Yuanjie Zheng, Jingyi Yu, Sing Bing Kang, S. Lin and C. Kambhamettu. *Single-image vignetting correction using radial gradient symmetry*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8, June 2008.

[Zheng *et al.* 2009] Yuanjie Zheng, S. Lin, C. Kambhamettu, Jingyi Yu and Sing Bing Kang. *Single-image vignetting correction*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 12, pages 2243–2256, December 2009.

[Zhou *et al.* 2011] Kun Zhou, Minmin Gong, Xin Huang and Baining Guo. *Data-Parallel Octrees for Surface Reconstruction*. Visualization and Computer Graphics, IEEE Transactions on, vol. 17, no. 5, pages 669 –681, May 2011.

[Zitová & Flusser 2003] Barbara Zitová and Jan Flusser. *Image registration methods: a survey*. Image and Vision Computing, vol. 21, no. 11, pages 977–1000, August 2003.

Appendix A

Images from sequence 1, Massachusetts Institute of Technology (MIT) dataset. Sequence 1 is composed of locations *A* through *E*.

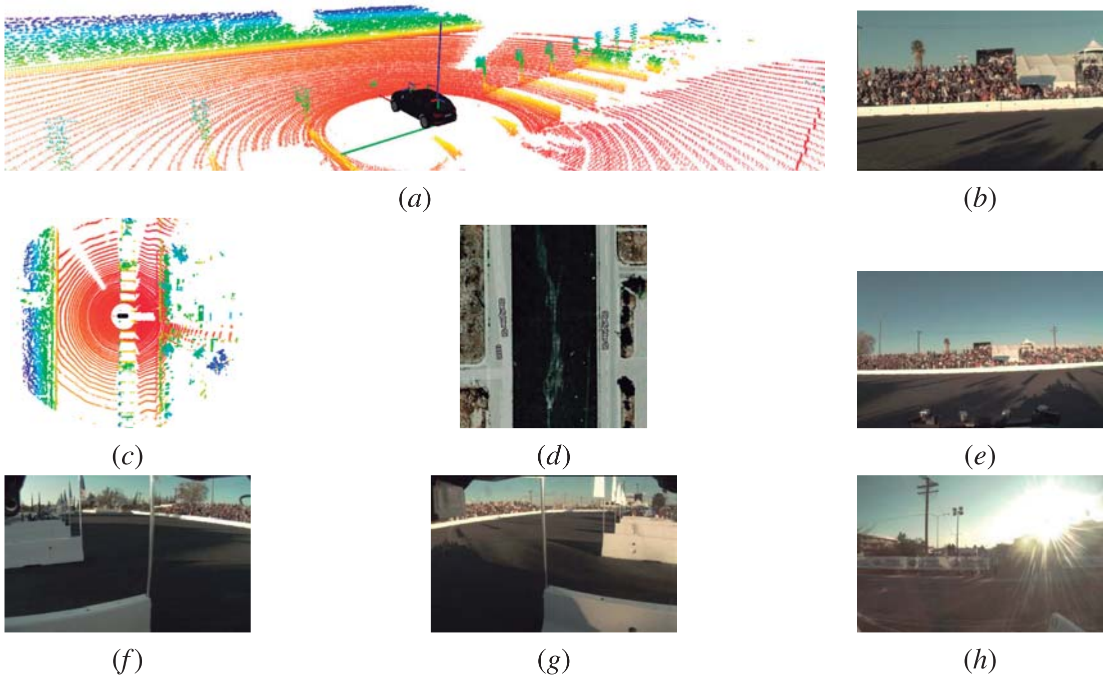


Figure A.1: Location *A* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

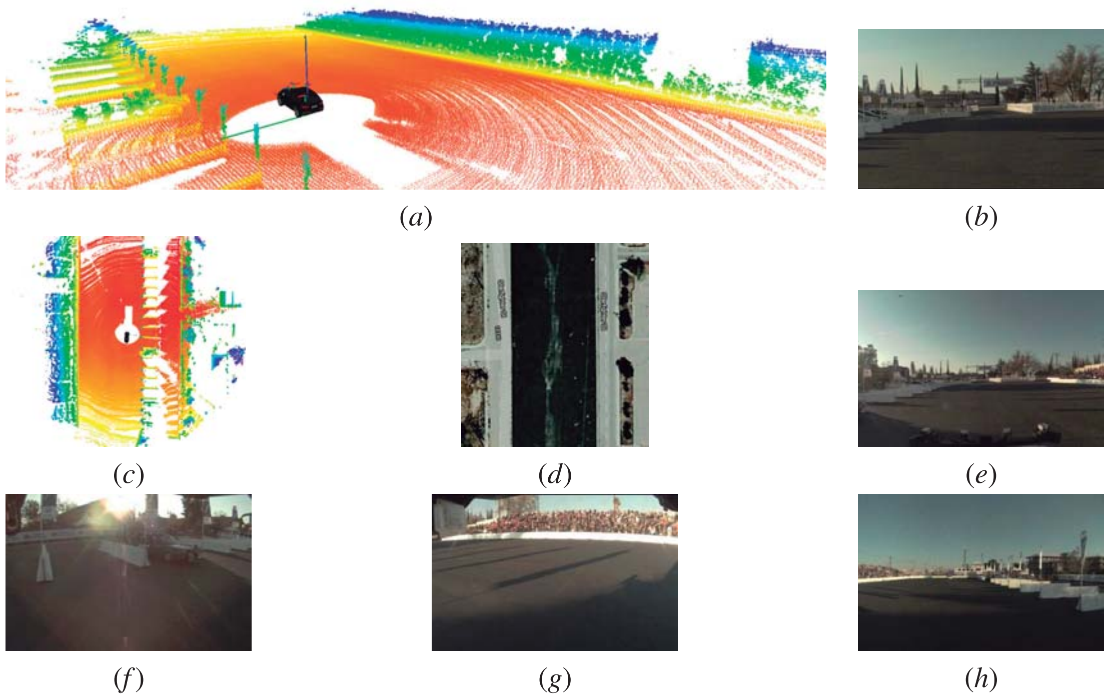


Figure A.2: Location *B* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

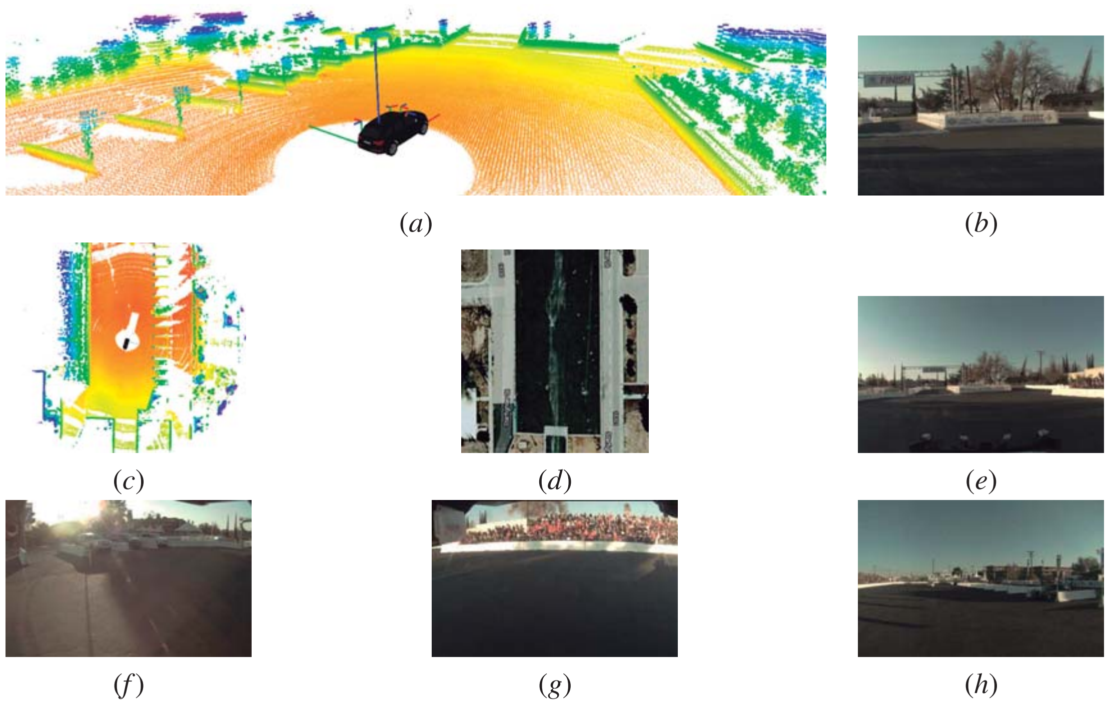


Figure A.3: Location *C* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

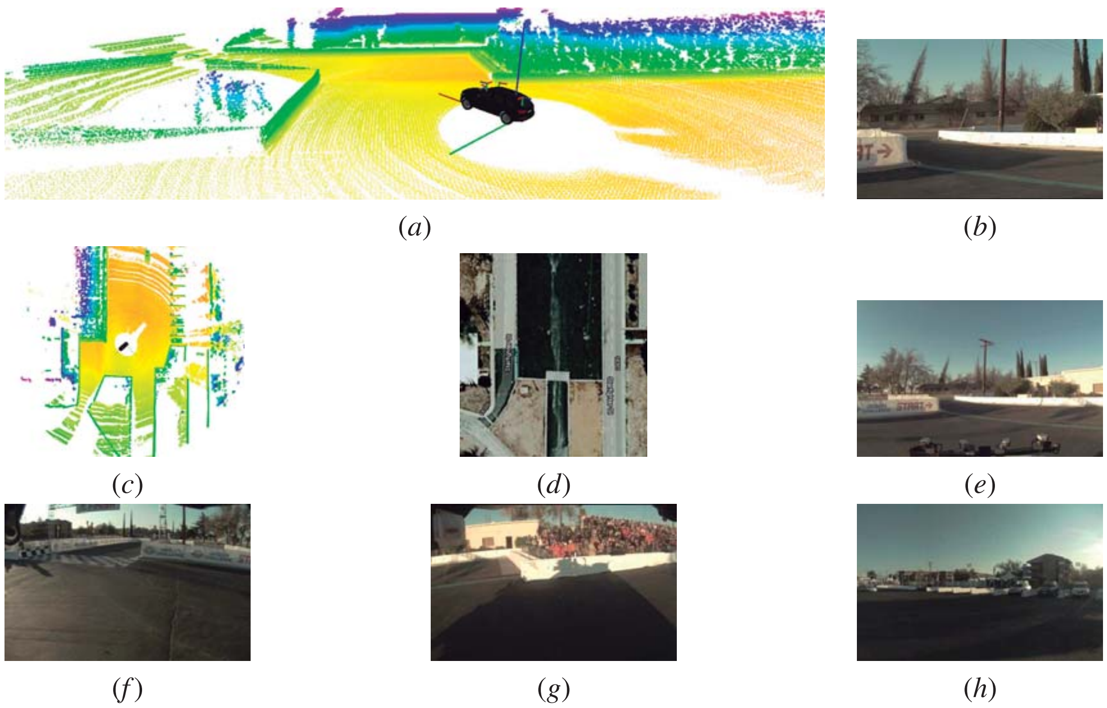


Figure A.4: Location *D* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

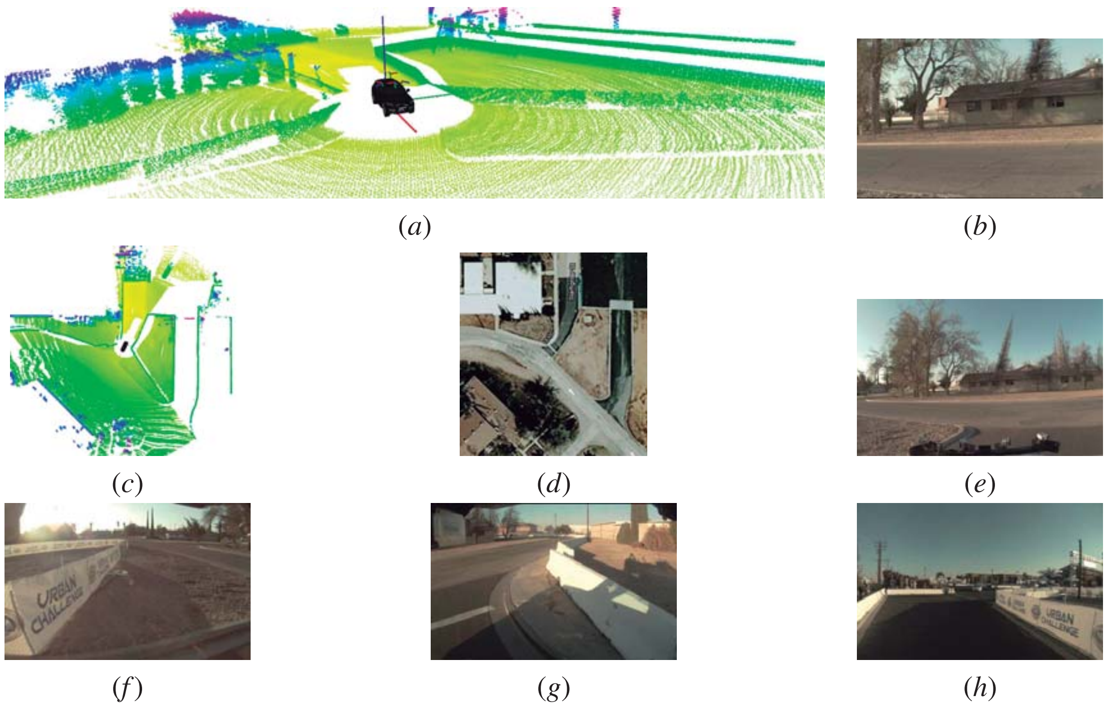


Figure A.5: Location *E* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

Appendix B

Images from sequence 2, MIT dataset. Sequence 2 is composed of locations *A* through *I*.

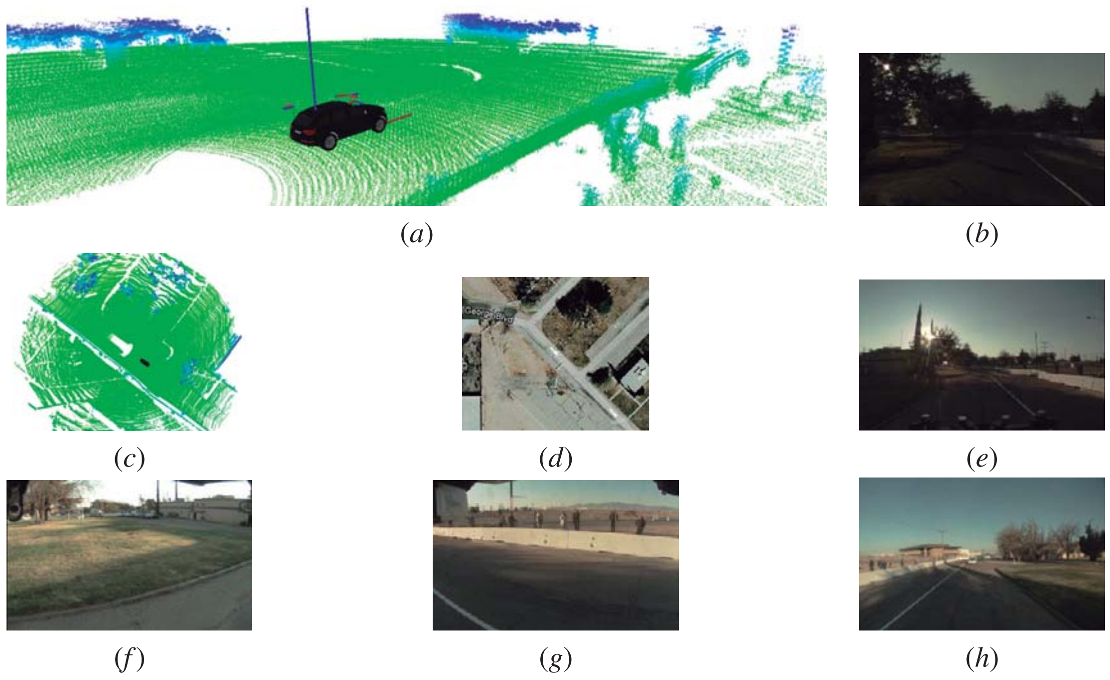


Figure B.1: Location *A* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

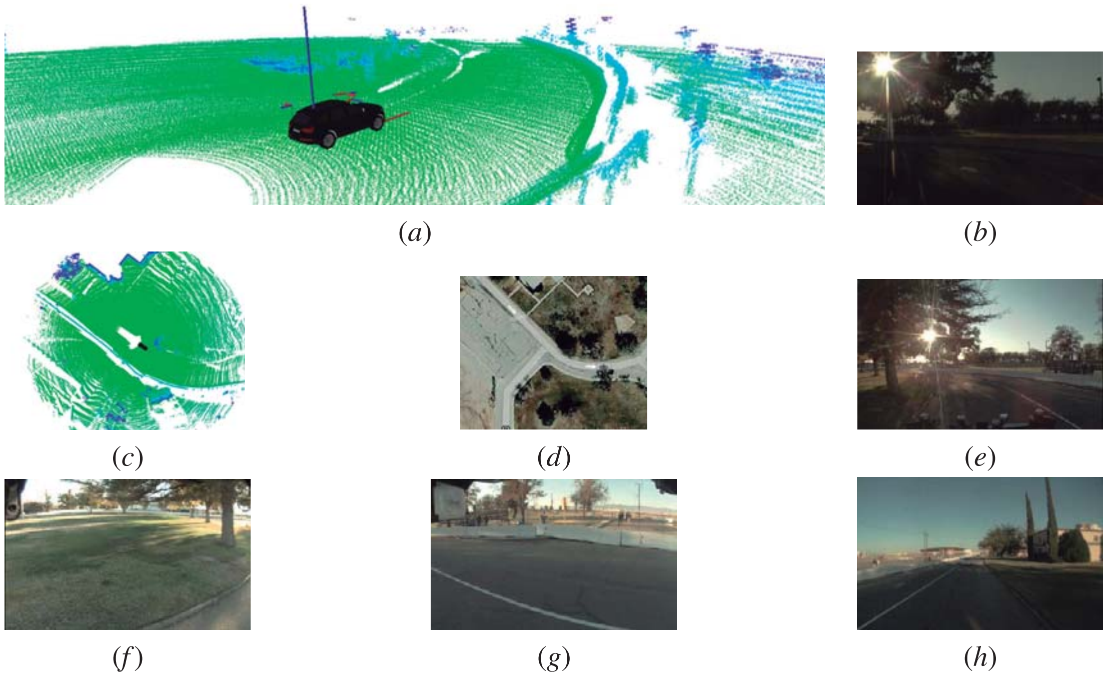


Figure B.2: Location *B* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

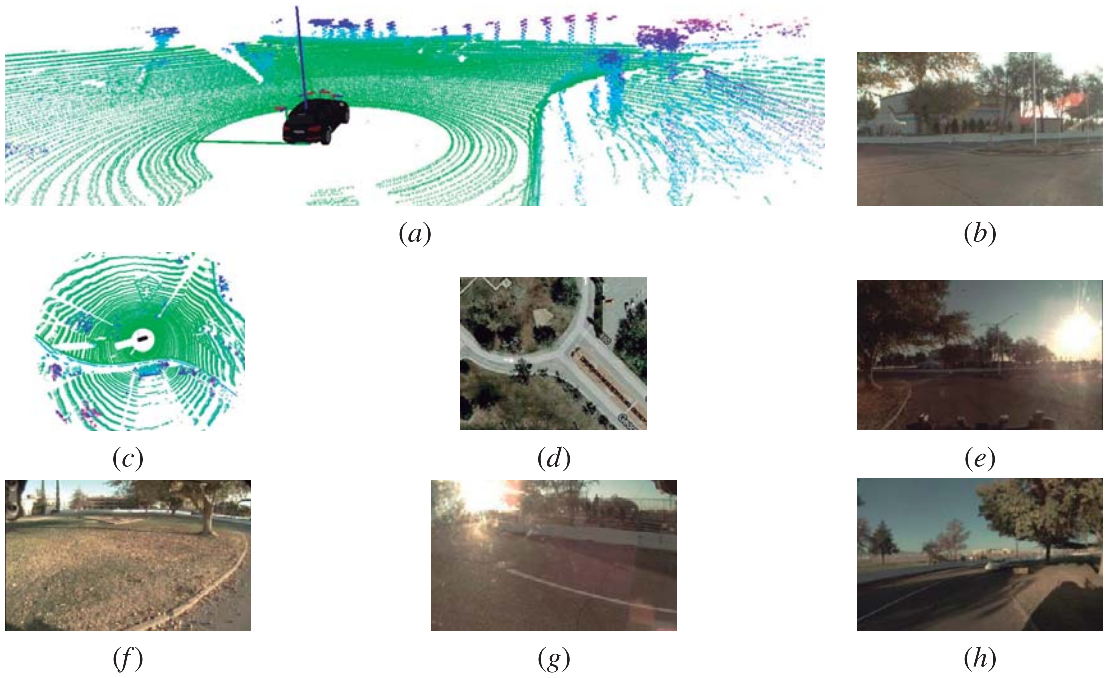


Figure B.3: Location *C* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

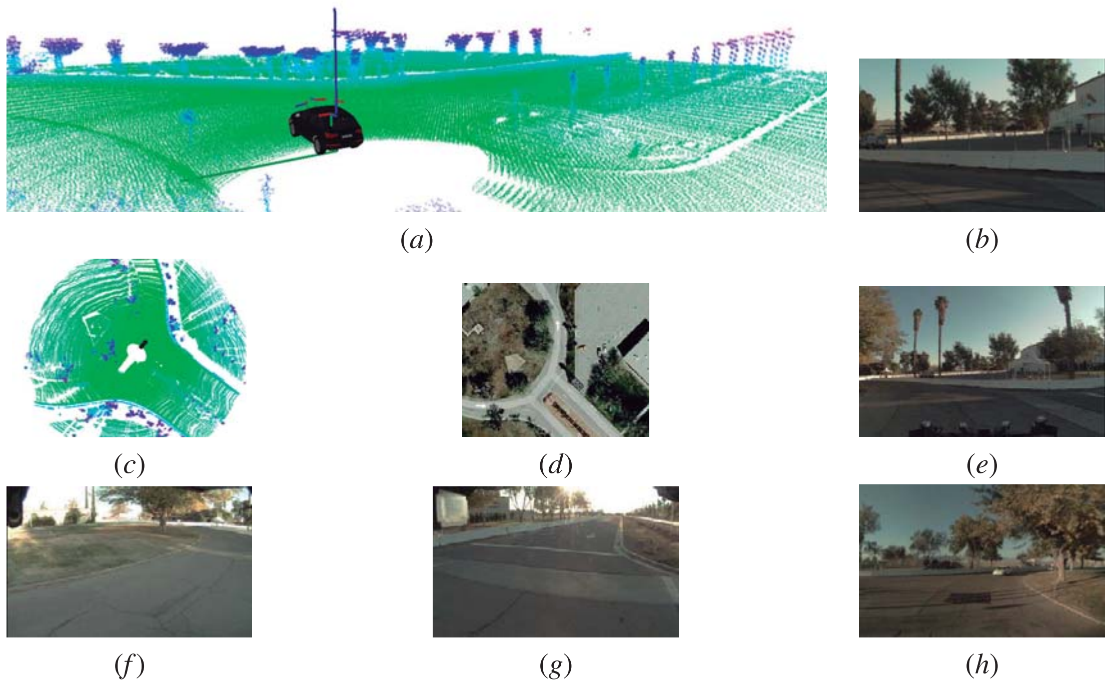


Figure B.4: Location *D* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

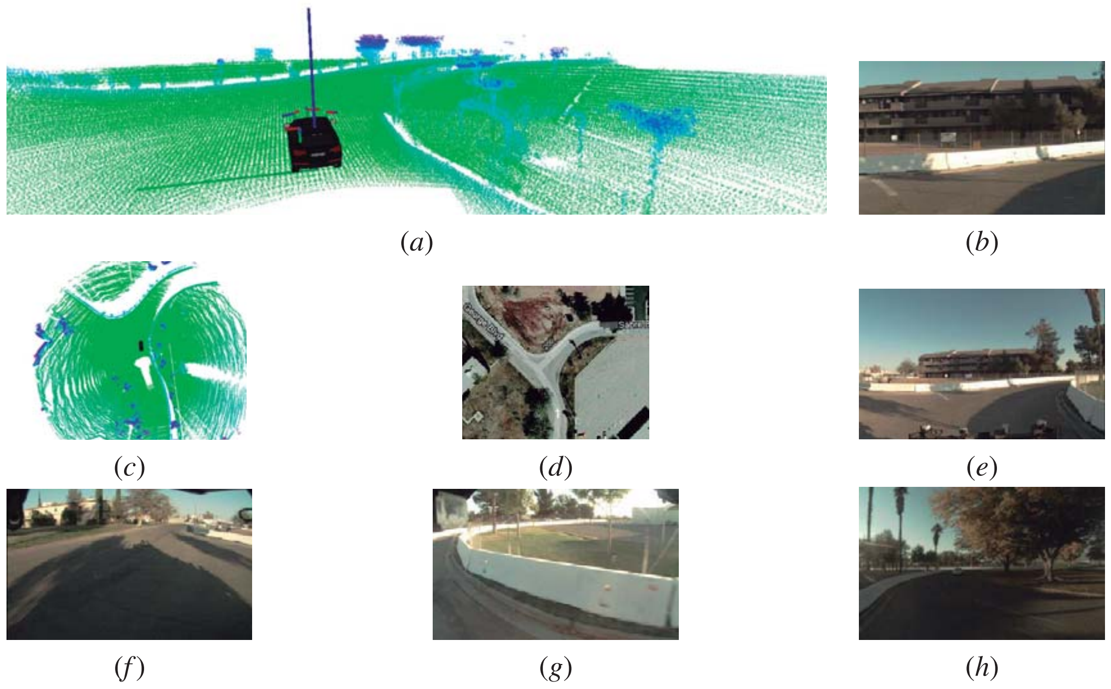


Figure B.5: Location *E* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

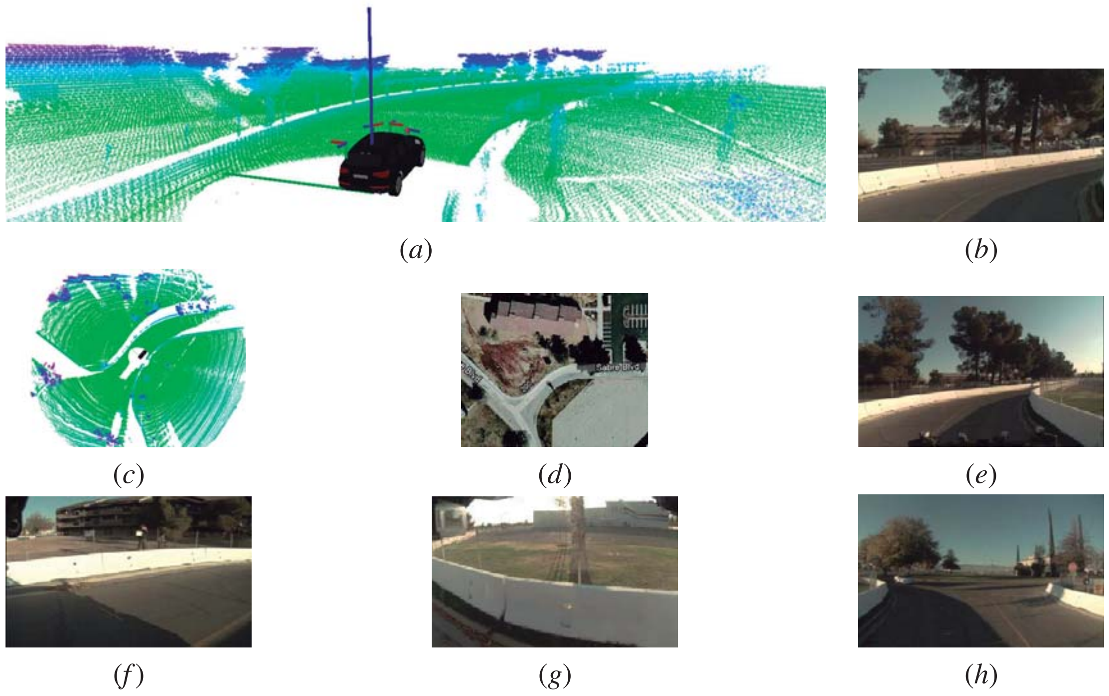


Figure B.6: Location *F* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

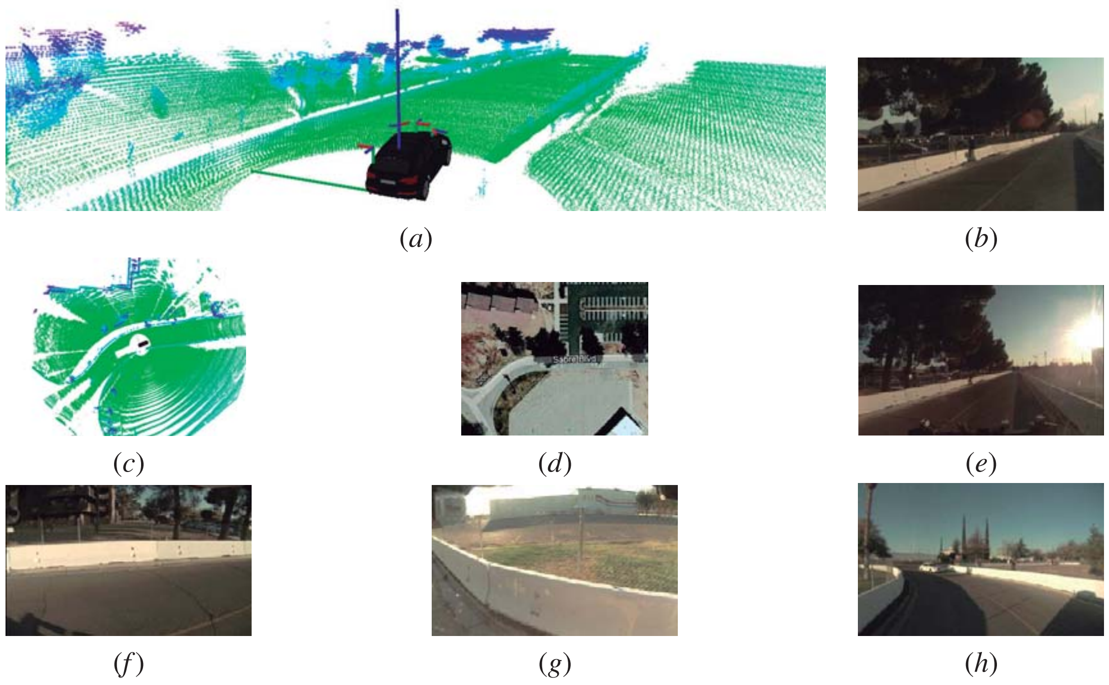


Figure B.7: Location *G* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

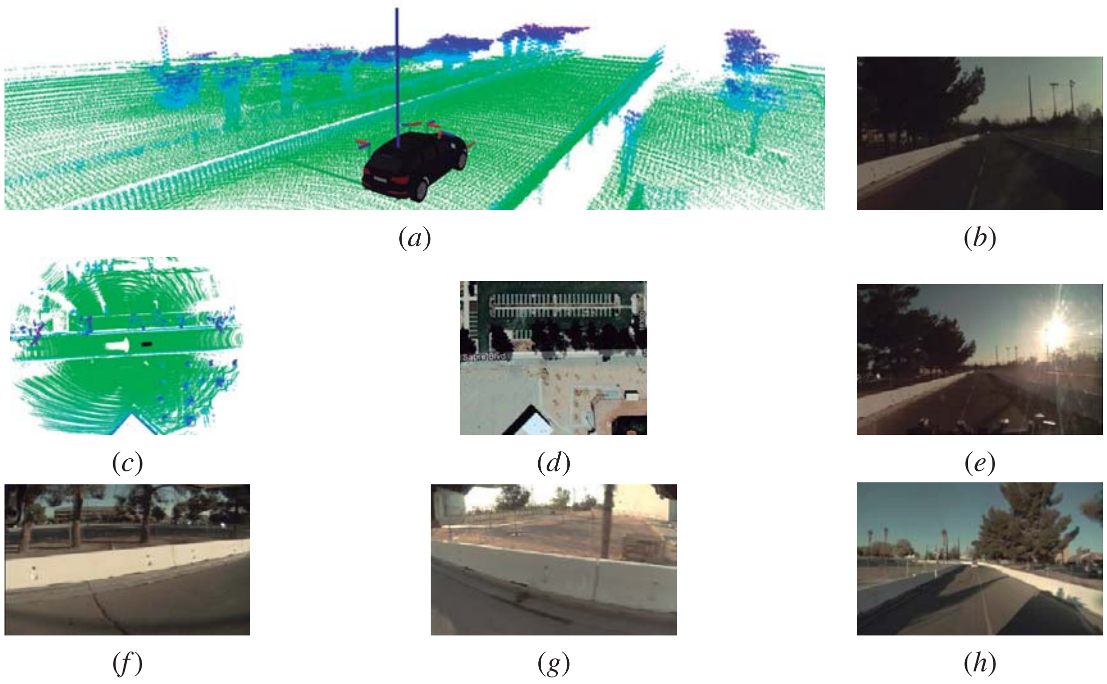


Figure B.8: Location *H* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

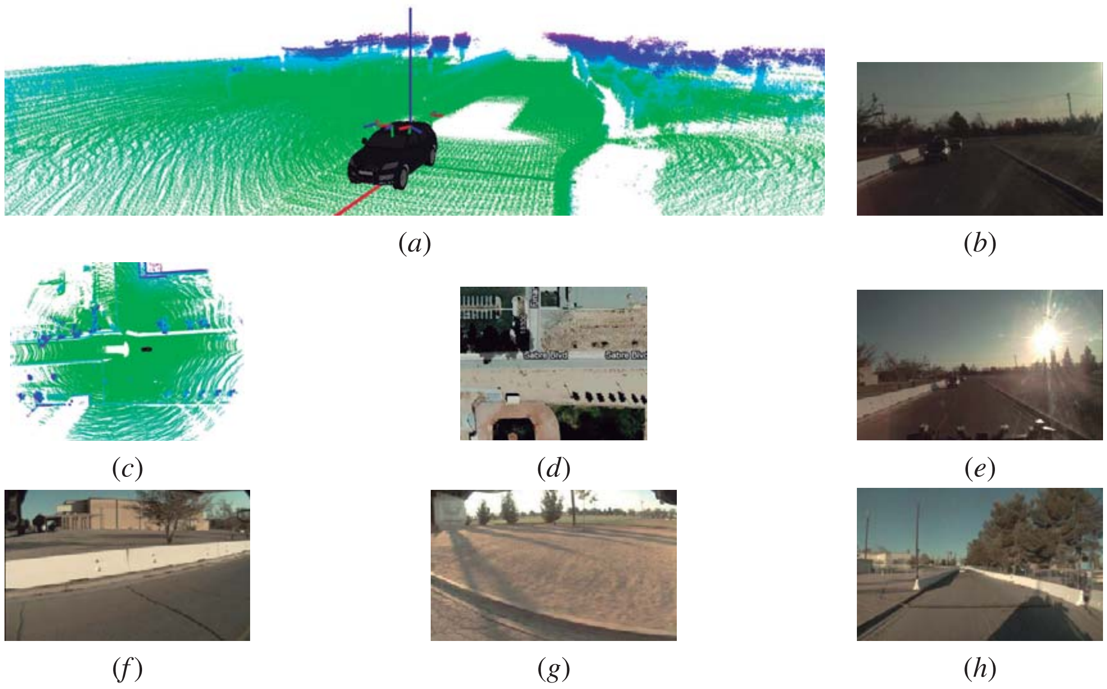


Figure B.9: Location *I* of the sequence. Isometric (*a*) and bird's eye (*c*) view of the 3D data; front 6mm (*b*), front (*e*), rear (*h*), left (*f*) and right (*g*) camera images; (*e*) satellite view of the location.

Appendix C

Results from the extraction of geometric surface reconstruction in sequence 1 of the MIT data set for approaches Ball Pivoting Algorithm (BPA), Greedy triangulation (GT), Poisson Surface Reconstruction (POIS) and Geometric Polygonal Primitives (GPP) parameters set 2;

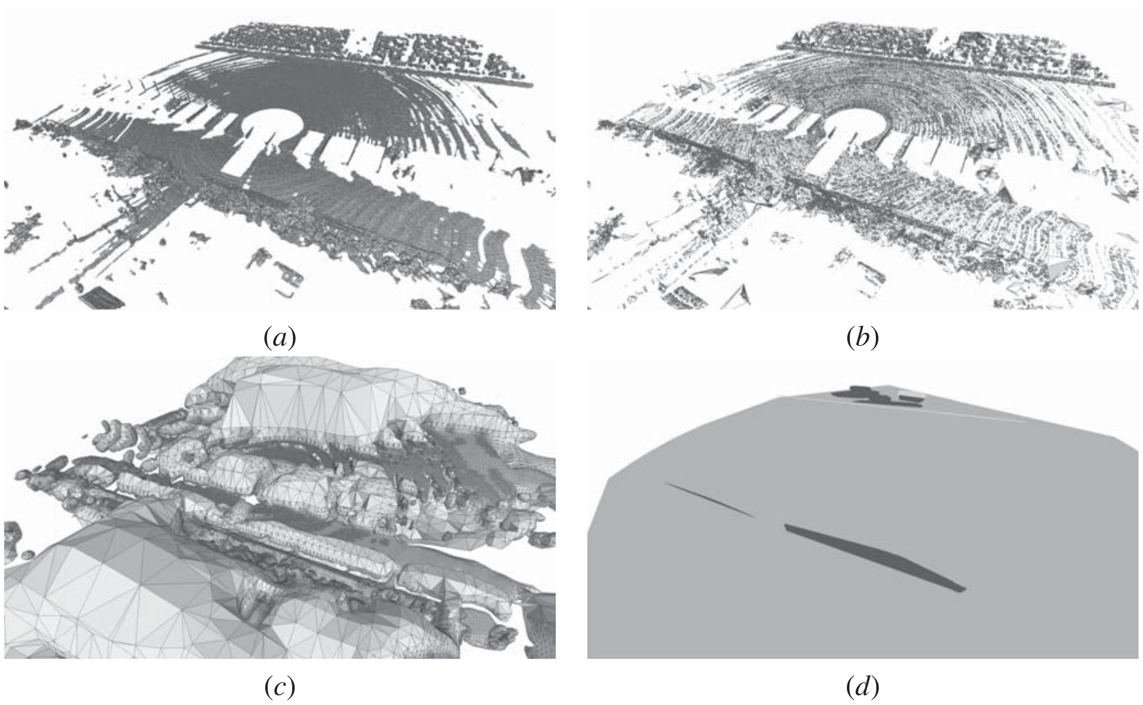


Figure C.1: Qualitative comparison of several surface reconstruction methodologies in location *A* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

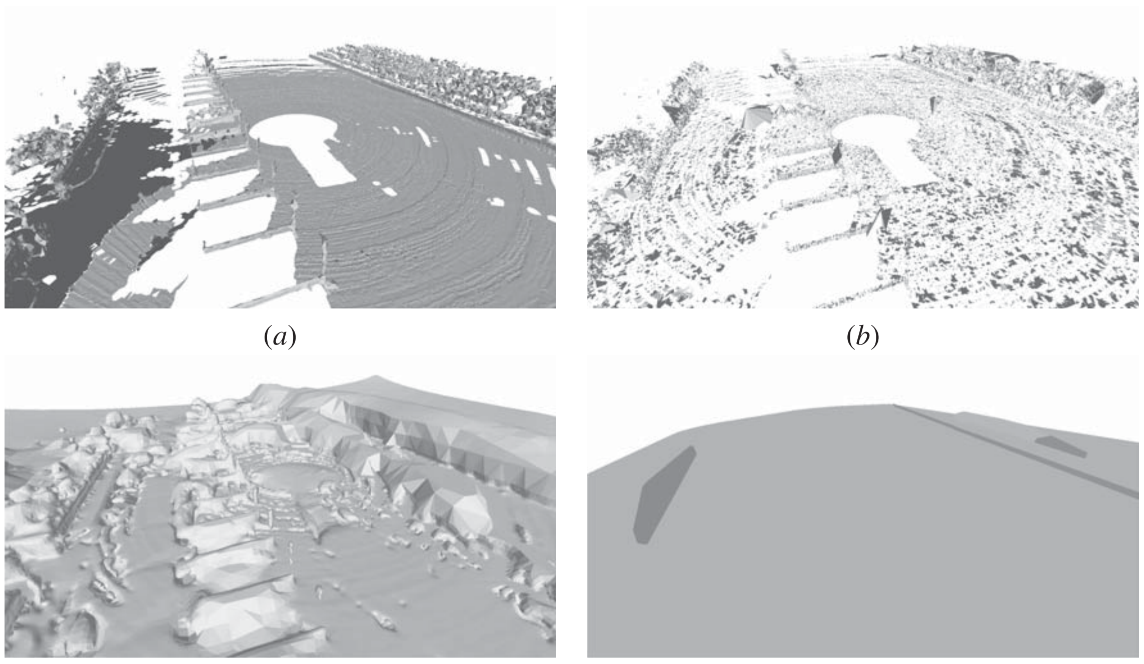

(*a*) (*b*) (*c*) (*d*)

Figure C.2: Qualitative comparison of several surface reconstruction methodologies in location *B* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

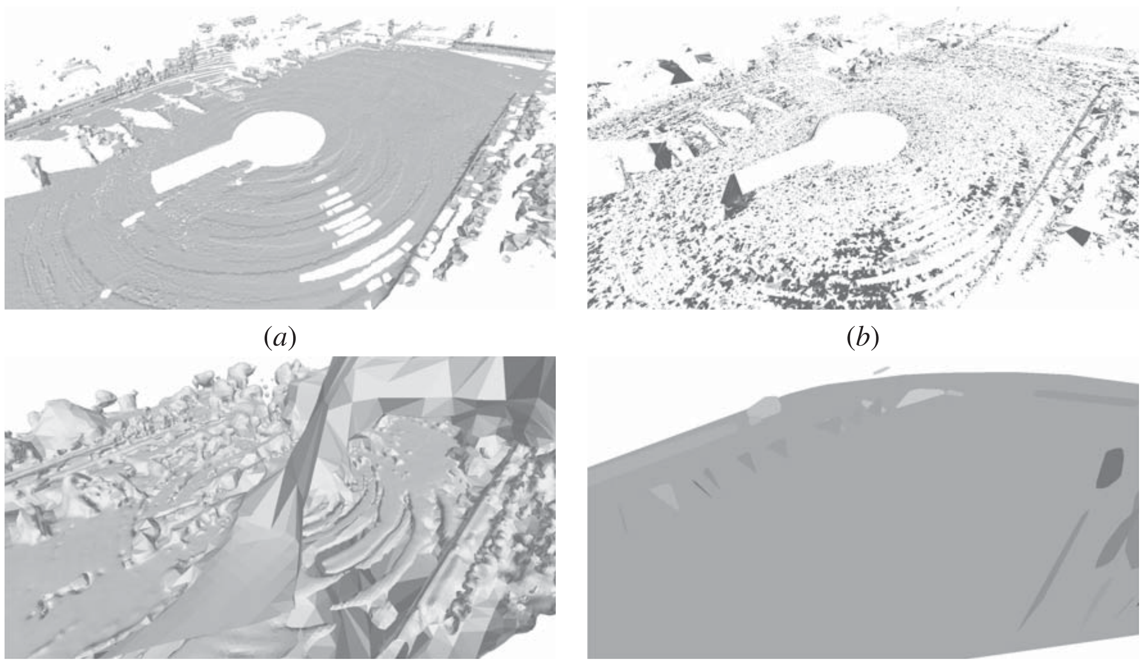

(*a*) (*b*) (*c*) (*d*)

Figure C.3: Qualitative comparison of several surface reconstruction methodologies in location *C* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

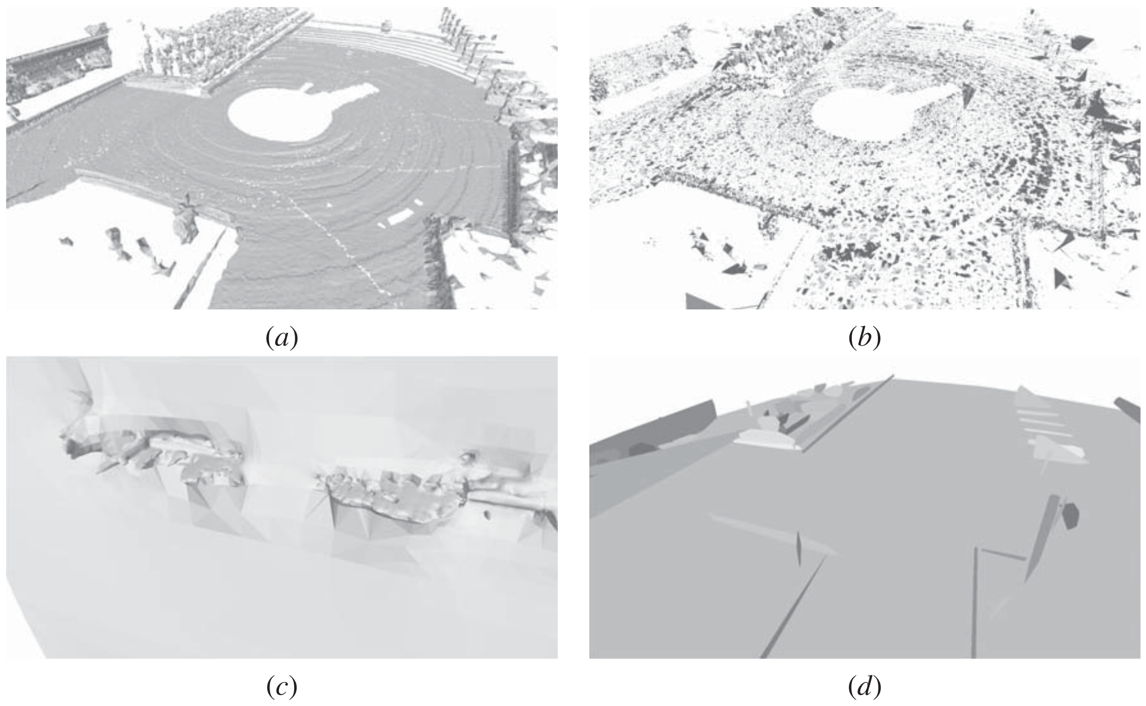


Figure C.4: Qualitative comparison of several surface reconstruction methodologies in location *D* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

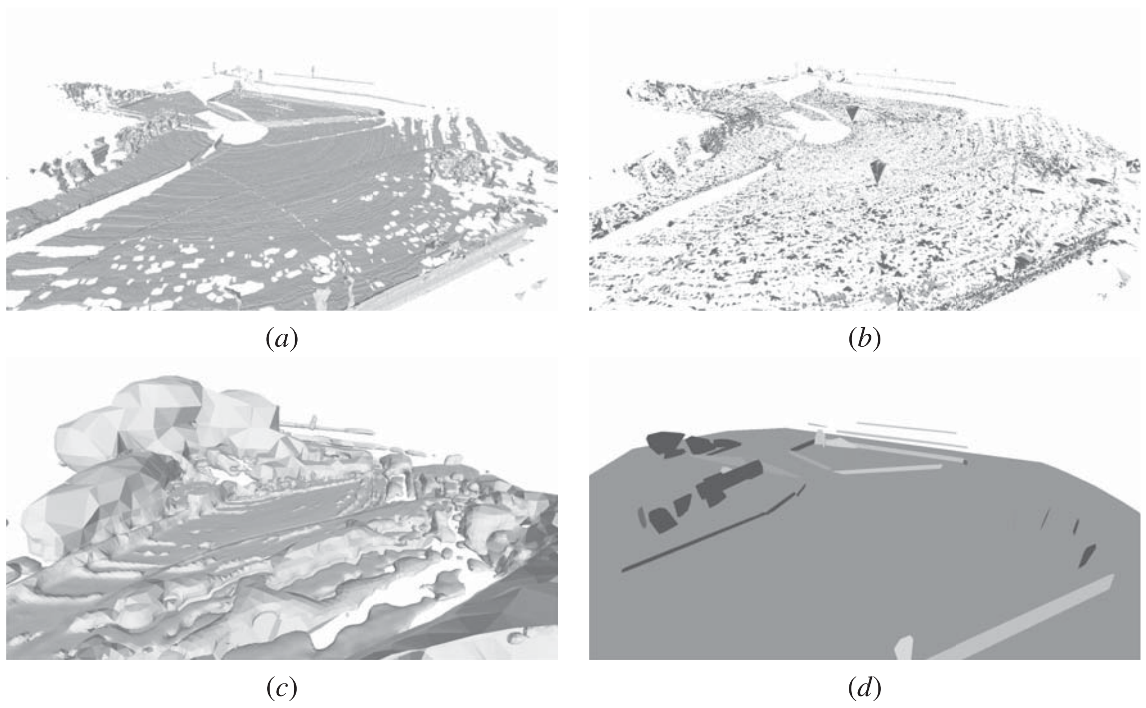
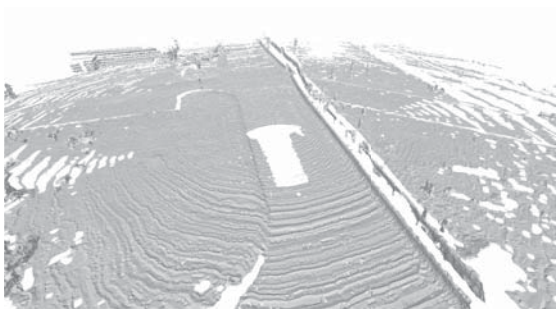


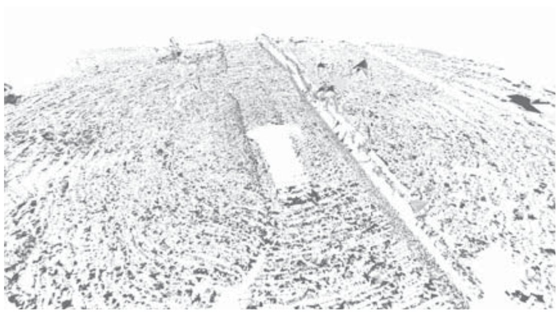
Figure C.5: Qualitative comparison of several surface reconstruction methodologies in location *E* of MIT sequence 1: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

Appendix D

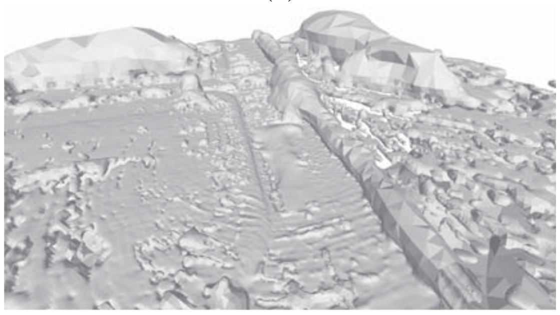
Results from the extraction of geometric surface reconstruction in sequence 2 of the MIT data set for approaches BPA, GT, POIS and GPP parameters set 2;



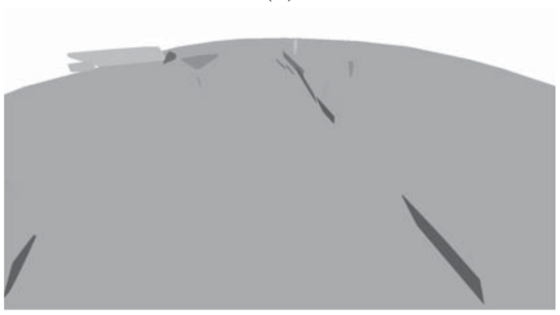
(*a*)



(*b*)



(*c*)



(*d*)

Figure D.1: Qualitative comparison of several surface reconstruction methodologies in location *A* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

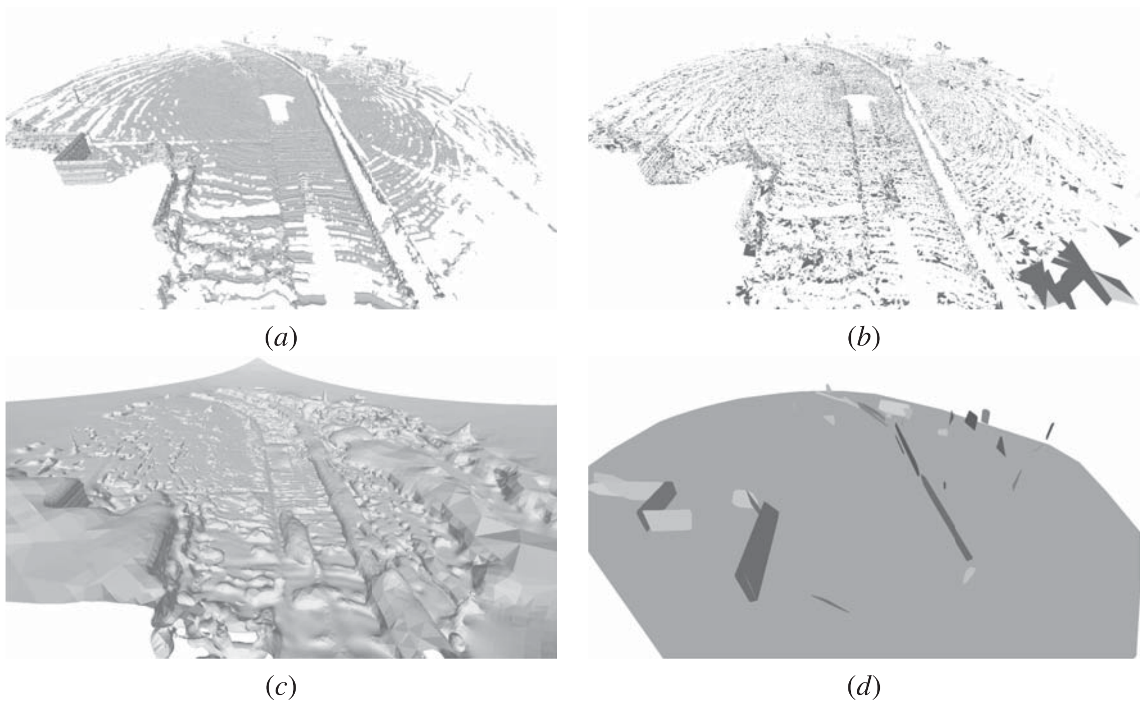


Figure D.2: Qualitative comparison of several surface reconstruction methodologies in location *B* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

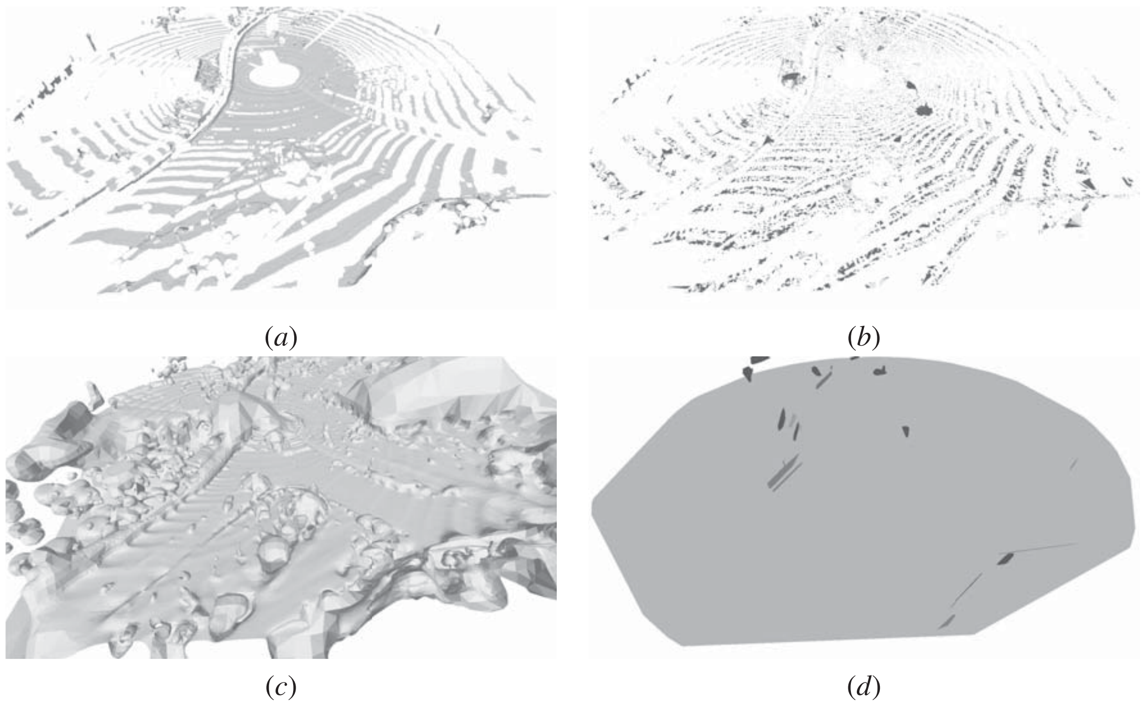


Figure D.3: Qualitative comparison of several surface reconstruction methodologies in location *C* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

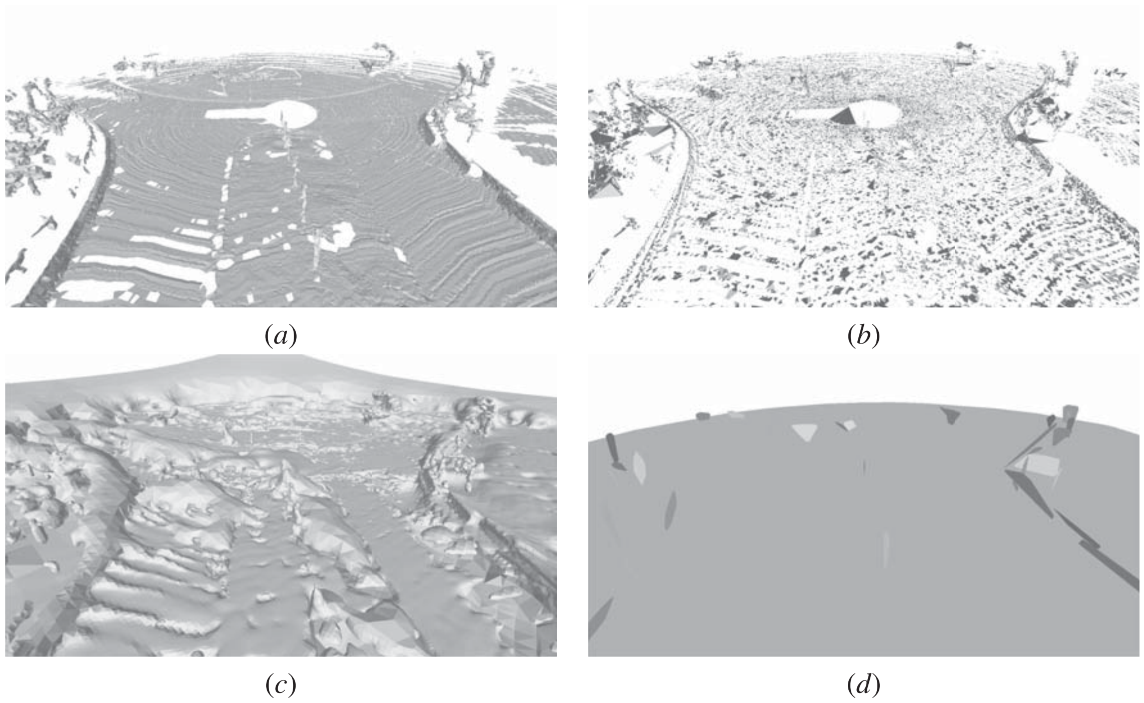


Figure D.4: Qualitative comparison of several surface reconstruction methodologies in location *D* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

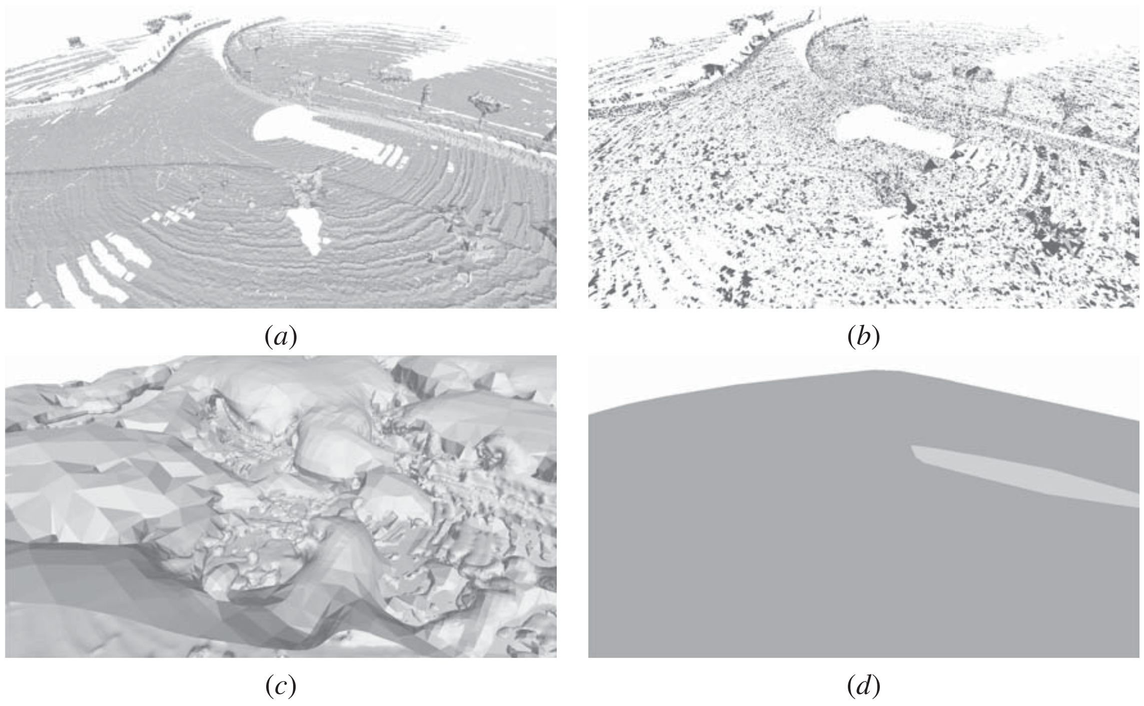


Figure D.5: Qualitative comparison of several surface reconstruction methodologies in location *E* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

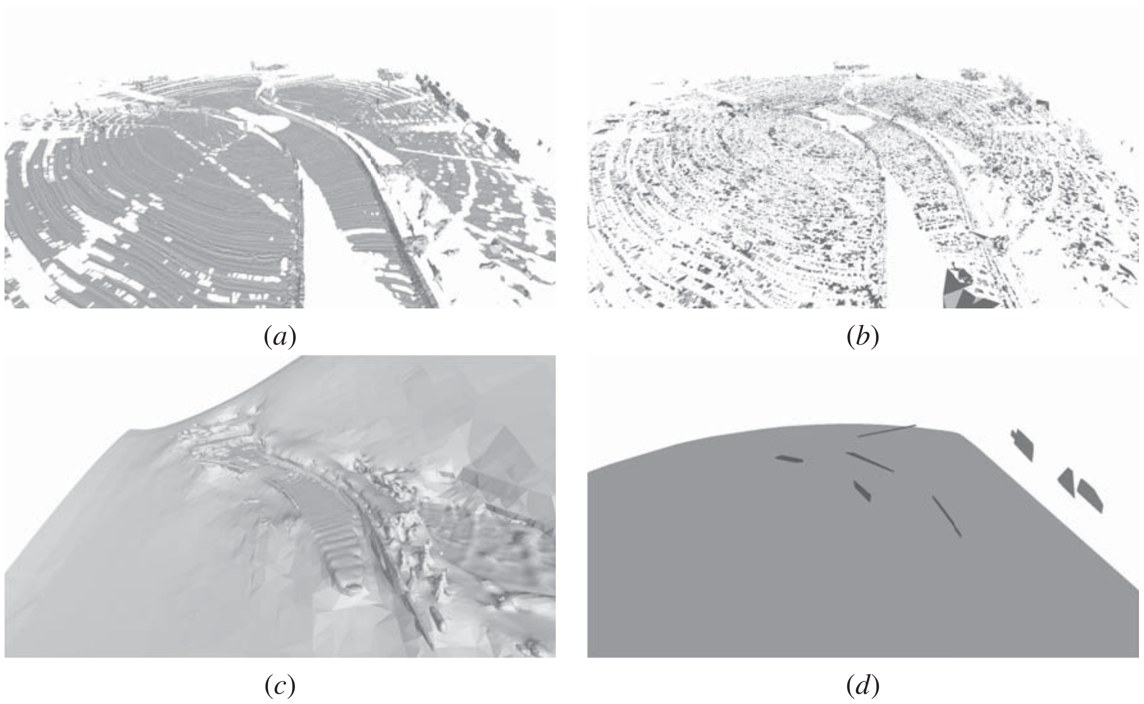


Figure D.6: Qualitative comparison of several surface reconstruction methodologies in location *F* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

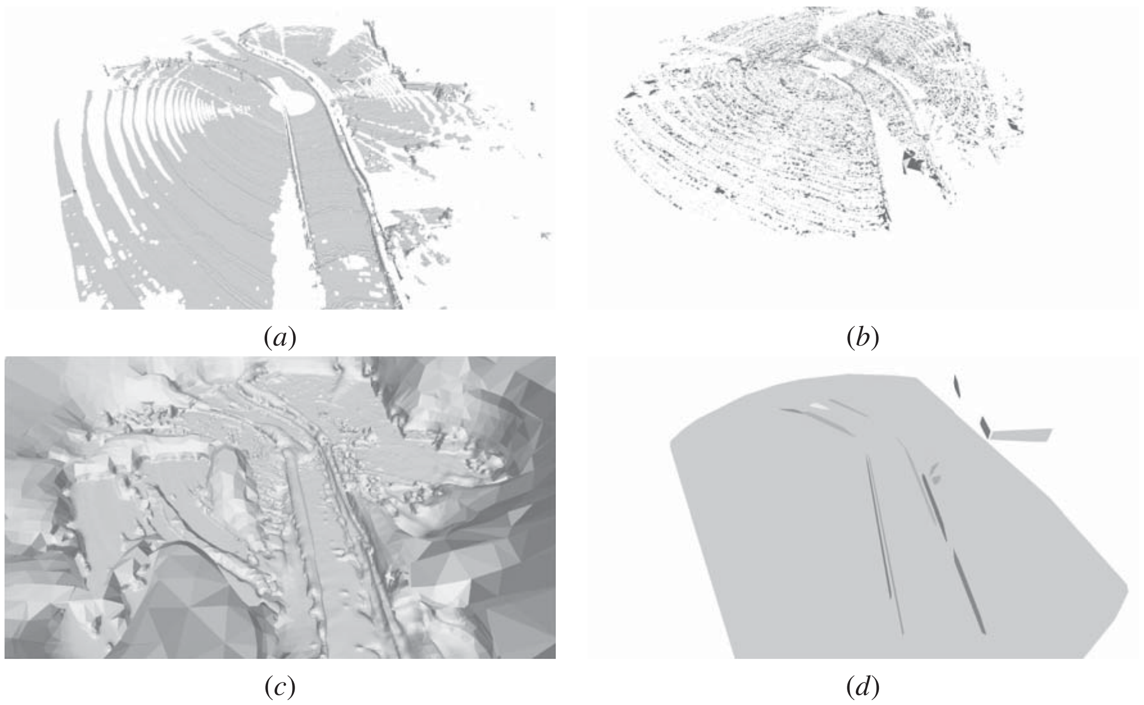


Figure D.7: Qualitative comparison of several surface reconstruction methodologies in location *G* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

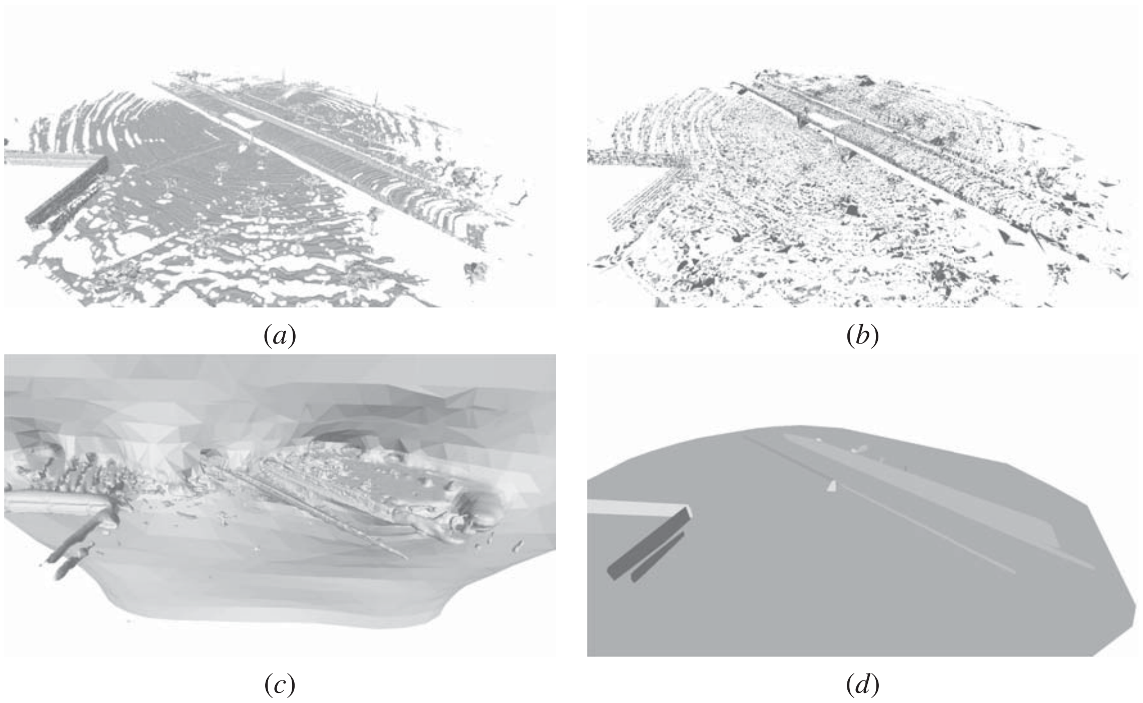


Figure D.8: Qualitative comparison of several surface reconstruction methodologies in location *H* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;

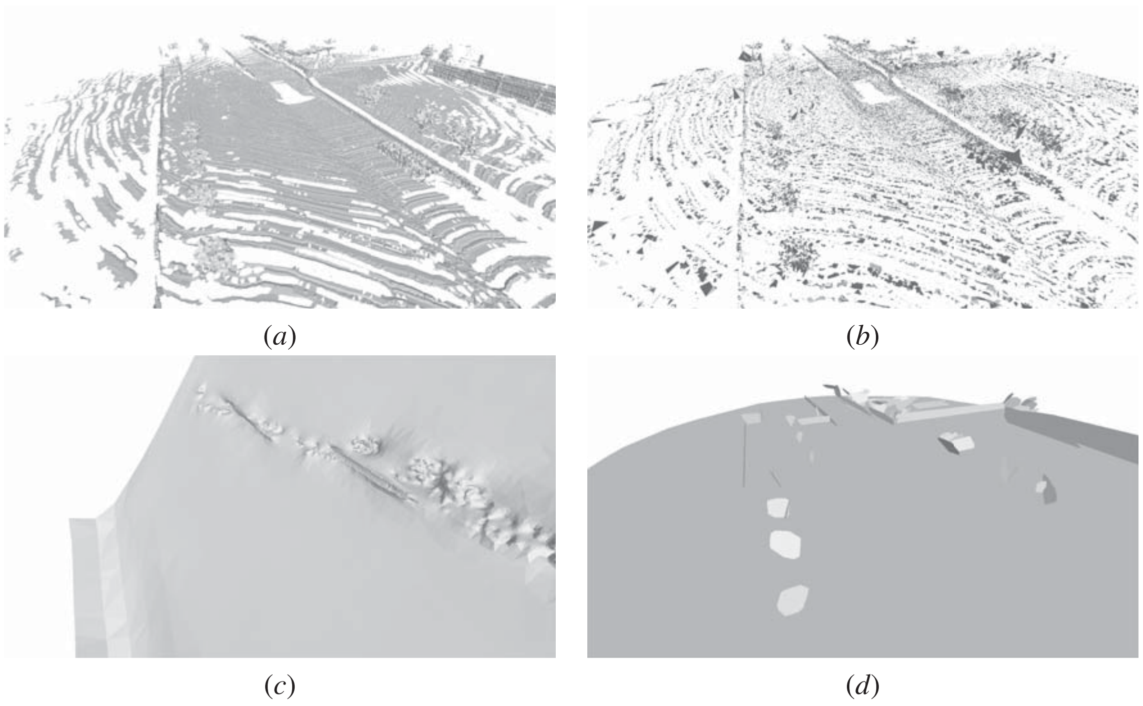


Figure D.9: Qualitative comparison of several surface reconstruction methodologies in location *I* of MIT sequence 2: (*a*) BPA; (*b*) GT; (*c*) POIS; (*d*) GPP parameters set 2;