Dissertation zur Erlangung des Doktorgrades der Fakultät für Angewandte Wissenschaften der Albert-Ludwigs-Universität Freiburg im Breisgau

Three-dimensional Perception for Mobile Robots

Rudolph Triebel

Mai 2007

Betreuer: Prof. Dr. W. Burgard

Dekan der Fakultät für Angewandte Wissenschaften:
Prof. Dr. Bernhard Nebel

1. Gutachter: Prof. Dr. Wolfram Burgard, Universität Freiburg
2. Gutachter: Prof. Dr. Matthias Teschner, Universität Freiburg

Datum der mündlichen Prüfung: 14. September 2007

Zusammenfassung

Die vorliegende Arbeit befasst sich mit dem Problem der dreidimensionalen Wahrnehmung von mobilen Robotern, wobei die Wahrnehmung aus drei wichtigen Aufgabenbereichen besteht, nämlich der Datenaufnahme, der konsistenten und effizienten Datenrepräsentation, sowie dem Hinzufügen von semantischer Information durch Klassifikation der vorhandenen Objekte. Für all diese drei Aufgabenbereiche werden neuartige Herangehensweisen bereitgestellt.

Zur *Datenaufnahme* stellen wir ein Robotersystem vor, welches aus einem Manipulator mit vier Freiheitsgraden besteht, auf dem ein Laser-Entfernungs-Sensor befestigt ist. Dieses System ist in der Lage, ein Objekt, das innerhalb der Reichweite des Manipulators ist, adaptiv in 3D zu erfassen. Die Wege, die der Laser-Sensor dabei abfährt, werden selbstständig geplant, indem der erwartete Informationsgewinn entlang möglicher kollisionsfreier Wege maximiert wird. Auf diese Weise wird die Anzahl der Datenerfassungen minimiert und die Sensorwege werden dem Objekt angepasst.

Für den zweiten Aufgabenbereich werden Methoden zur *kompakten und konsistenten Repräsentation* von dreidimensionalen Daten vorgeschlagen. In einem ersten Schritt entwickeln wir einen probabilistischen Algorithmus zur Erkennung von planaren Strukturen in dreidimensoinalen Entfernungsdaten, welcher auf einem hierarchischen Bayesschen Netz basiert und das Prinzip der Erwartungswertmaximierung (*engl:* Expectation Maximization, *kurz:* EM) anwendet um die Lage der Ebenen zu schätzen. Die dabei erreichten Verbesserungen gegenüber bisherigen Ansätzen beruhen auf dem Einbeziehen von *Hauptrichtungen* in den Prozess der Ebenenschätzung. Dabei werden Ebenen gleichzeitig so zu Teilmengen zusammengefasst, dass diese entweder parallel zueinander sind oder sich in einem bestimmten Winkel schneiden. Dadurch können kleinere Fehler beim Übereinstimmen korrigiert werden. Desweiteren wird die Verwendung von Texturinformation zur Verbesserung der Ebenenberechnung eingeführt. Im zweiten Schritt dieses Aufgabenbereiches formulieren wir eine neue probabilistische Datenstruktur, die besonders zur Bewegungsplanung und zur Navigation in Umgebungen mit mehreren Oberflächen auf verschiedenen Höhenniveaus geeignet ist. Diese Datenrepräsentation bezeichnen wir als Mehrstufige Oberflächenkarte (*engl:* Multi-level Surface Map, *kurz:* MLS map). Sie stellt eine Erweiterung der sogenannten Erhebungskarten (*engl.:* elevation maps) dar. Der Vorteil von mehrstufigen Oberflächenkarten gegenüber den bisherigen Erhebungskarten ist die Fähigkeit, überhängende Objekte wie zum Beispiel Bäume, sowie mehrere übereinander liegende Ebenen wie beispielsweise bei Brücken, darzustellen. Auf diese Weise ist der Roboter in der Lage Wege zu planen, die unter einer Brücke hindurch und gleichzeitig über sie hinweg verlaufen. Schließlich, im dritten und letzten Schritt, schlagen wir einen Algorithmus vor, der global konsistente Kartendarstellungen aus lokalen 3D Ansichten berechnet, indem die Roboterpositionen in sechs Freiheitsgraden mit Hilefe von Einschränkungen optimiert werden (*engl:* constraint optimization). Hierbei können die lokalen Ansichten sowohl in Form von MLS-Karten, als auch von anderen 3D Darstellungen wie beispielsweise Punktwolken vorliegen. Die Verbesserung, die durch die vorgestellte Registrierungsmethode erreicht wird, beruht auf der Hinzunahme von globalen Einschränkungen an die Roboterpositionen, zusätzlich zu den lokalen Einschränkungen, die durch paarweises Zusammenfügen von zeitlich aufeinander folgenden lokalen Ansichten (*engl.:* scan matching) entwickelt werden. Eine solche globale Einschränkung wird hergeleitet aus Merkmalen in Form von Geradenstücken, die in verschiedenen lokalen Ansichten vorkommen und einer gemeinsamen Ebene zugeordnet sind. Die Lage dieser Ebenen wird iterativ und zugleich mit der Berechnung der Roboterpositionen geschätzt. Das Ergebnis ist, dass der Registrierungsalgorithmus mit weniger benötigten lokalen Ansichten zurechtkommt, die sich gegenseitig zu geringeren Anteilen überlappen können. Außerdem wird kein Schleifenschließen (*engl:* loop closing) benötigt, um den angesammelten Positionierungsfehler zu korrigieren.

Im dritten und letzten Aufgabenbereich stellen wir einen neuartigen Ansatz zur *Klassifikation* von dreidimensionalen Objekten in eine vordefinierte Menge von Klassen bereit. Der Ansatz wird im Kontext des überwachten Lernens formuliert, bei dem zunächst eine Anzahl von Parametern im sogenannten *Lernschritt* berechnet wird. Diese werden später im *Inferenzschritt* verwendet, um die Objektklassifikation durchzuführen. Die Grundidee des Klassifizierungsalgorithmus entspricht dem der *kollektiven Klassifikation*, bei der die Entscheidung über die Klassifizierung eines jeden Datenpunktes sowohl von der lokalen Evidenz abhängt, dass dieser Datenpunkt zu einer bestimmten Klasse gehört, als auch von der Klassifizierung der benachbarten Datenpunkte. Dabei werden die statistischen Abhängigkeiten der Klassifizierungen benachbarter Datenpunkte modelliert. Zur Beschreibung der A-Posteriori-Verteilung der Klassenzuweisungen für eine gegebene Menge von Merkmalsvektoren benutzen wir Assoziative Markov-Netze (*engl.:* Associative Markov Networks, *kurz:* AMNs), die bereits zuvor erfolgreich eingesetzt wurden, um 3D Objekte in Punktwolken zu klassifizieren. Wir erweitern den AMN-Ansatz in zwei Aspekten. Einerseits entwickeln wir eine Technik zur adaptiven Reduktion des Trainingsdatensatzes mit Hilfe von *kd*-Bäumen, die in geeigneter Form beschnitten werden. Dadurch wird der Lernschritt beschleunigt, ohne die Qualität der Klassifizierung zu mindern. Zum anderen kombinieren wir den AMN-Klassifikator mit einer instanzbasierten Nächste-Nachbarn-Methode. Die Idee hierbei ist, die Merkmale so zu transformieren, dass sie einfacher durch Hyperebenen trennbar sind, was eine der Bedingungen für die Benutzung von AMNs darstellt. Dadurch wird die Klassifikation verbessert im Vergleich zum Standard AMN-Ansatz.

Schließlich stellen wir zwei Anwendungen für den vorgeschlagenen Ansatz der 3D Wahrnehmung vor. Die erste Anwendung ist die Erstellung von sehr großen, konsistenten MLS-Karten von Außenbereichen mit Hilfe eines Roboterautos. Das Auto ist ausgestattet mit verschiedenen Sensoren zum Lokalisieren und Kartenerstellen, wie z.B. einem GPS Empfänger, einem Trägheitssensor, einem optischen Gyroskop und einem 3D Laser-Entfernungs-Sensor. Wir zeigen, dass trotz des sehr genauen Lokalisierungsmoduls des Systems, basierend auf einem Informationsfilter, der hier vorgestellte globale Registrierungsalgorithmus notwendig ist, um die Positionen des Fahrzeugs zu korrigieren. Die zweite Anwendung beschäftigt sich mit dem Problem der Okklusionen in 3D Entfernungsdaten. Hierzu wenden wir zunächst unsere Objekterkennungsmethode an, um die vorhandenen Objekte zu klassifizieren. Objekte, die wegen Okklusionen vom Sensor nur teilweise erfasst wurden, werden trotzdem richtig klassifiziert, solange ihr sichtbarer Anteil großgenug ist. Dann werden alle Instanzen eines Objekttyps zusammengefügt zu einem Prototypen. Dieser Prototyp enthält dann die Forminformation von allen Klasseninstanzen und stellt daher eine genauere Beschreibung der einzelnen Objekte dar. Insbesondere kann der verdeckte Teil eines Objektes vervollständigt werden, indem das Objekt duch den Prototypen ersetzt wird. Wir zeigen das am Beispiel eines verdeckten Fensters, bei dem die Okklusion während des Prozesses der Datenaufnahme durch einen Baum hervorgerufen wurde.

Abstract

This thesis addresses the problem of three-dimensional perception for mobile robots, where perception comprises three major tasks, namely data acquisition, consistent and efficient representation, and addition of semantic information by classification of the encountered objects. For all three of these subtasks we provide novel approaches.

For the *data acquisition* we present a robotic system consisting of a 4 DOF manipulator that carries a laser range finder. This system is able to adaptively scan an object that is positioned within the range of the manipulator. The paths that are followed with the scanner are planned automatically by maximizing the expected information gain along possible collision-free scan paths. This way, the number of scans is minimized and the paths are adapted to the object.

For the second task we propose methods for a *compact and consistent representation* of three-dimensional data. In a first step, we develop a probabilistic algorithm that detects planar structures in 3D range scans, which is based on a hierarchical Bayes net and uses expectation maximization (EM) to estimate the positions of the planes. The improvements over existing approaches are achieved by incorporating *main directions* into the plane estimation process. This way, planes are simultaneously clustered so that they are either parallel or intersecting in a certain angle and smaller alignment errors can be corrected. Furthermore, the use of texture information is introduced as an improvement of the plane estimation. In the second step, we formulate a new probabilistic representation that is particularly suited for motion planning and navigation in environments with several surfaces at different height levels. This representation is called multi-level surface maps (MLS maps) and constitutes an extension of the framework of elevation maps. The advantage of MLS maps over standard elevation maps is the ability to represent overhanging objects such as trees and multiple surface levels as in the case of bridges. This way the robot is able to plan paths which pass under a bridge and cross over it at the same time. Finally, in the third and last step, we propose an algorithm that computes globally consistent map representations from local 3D views by using a constraint optimization of the robot poses in 6 degrees of freedom. Here, the local views can be given as MLS maps or other 3D data representations such as point clouds. The improvement achieved by the presented registration method is based on the introduction of global constraints in addition to the local constraints that are imposed by pairwise scan matching of consecutive views. A global constraint is derived from line features in different scans that correspond to a common plane. The position of these planes is estimated iteratively during the computation of the robot poses. As a result, the registration algorithm can deal with less local views that overlap each other in smaller portions. Also, no loop closing is required to correct for accumulated positioning errors.

In the third and last task we provide a novel approach to *classify* three-dimensional objects into a predefined set of classes. The approach is formulated in the context of supervised learning, where a set of parameters is first computed in the *learning step* and later used to perform the object classification in the *inference step*. The main idea of the classification algorithm is taken from the framework of *collective classification*, where the labeling of each data point is decided upon the local evidence that this point belongs to a certain class as well as the labeling of the neighboring data points. This provides a method to model the statistical dependence of class labels of nearby data points. To describe the posterior distribution of the class labels for a given set of feature vectors, we use associative Markov networks (AMNs), which have been successfully used before to classify 3D objects in point cloud data. We extend the AMN approach in two ways. First, we provide a technique for adaptively reducing the training data set, which is based on *kd*-trees, that are pruned appropriately. This results in a faster learning step without reducing the quality of the classification results. Second, we combine the AMN classifier with an instance-based nearest-neighbor method. The idea here is to transform the features so that they are more easily separable by hyperplanes, which is one requirement for the use of AMNs. As a result, the classification improves over the standard AMN approach.

Finally, we present two applications of the proposed 3D perception approach. The first application is the computation of a large-scale consistent MLS map of outdoor environments by use of a robotic car. The car is equipped with several sensors for localization and mapping, such as a GPS receiver, an inertial measurement unit, an optical gyroscope, and a 3D laser range finder. We show that, although the system includes a very accurate localization module based on an information filter, the global registration algorithm is needed for the correction of the vehicle poses. The second application addresses the problem of occlusions in 3D range scan data. Here, we first apply our object recognition technique to classify several objects in a range scan. Objects that are only partially visible due to occlusions in the input data, are still classified correctly, as long as their visible part is large enough. Then, all instances of one object class are matched to one object prototype using scan matching. The resulting prototype contains the shape information from all class instances and therefore provides a more accurate description of the objects. In particular, the occluded part of an object instance can be completed by replacing the occluded object with the prototype. As an example, we show the completion of an occluded window, where the occlusion was caused by a tree during the data acquisition process.

Danksagung

Ich möchte mich hiermit bei den Menschen bedanken, die mich während meiner Arbeit an der Universität Freiburg und danach beim Beenden der Doktorarbeit begleitet und unterstützt haben. Insbesondere meinem Betreuer, Herrn Prof. Dr. Wolfram Burgard, danke ich für die vielen hilfreichen Hinweise und Ideen, die mich während meiner Arbeit vorangebracht haben. Mit seiner Geduld und seinem Interesse an immer neuen, interessanten Fragestellungen hat er zu einem großen Teil an dieser Doktorarbeit beigetragen. Ebenso bedanke ich mich bei Prof. Dr. Roland Siegwart für sein Verständnis und seine Geduld, vor allem während der "heißen" Phase des Aufschreibens der Arbeit.

Bedanken möchte ich mich auch bei meinen Kollegen und Freunden an der Universität Freiburg und der ETH Zürich, insbesondere bei Henrik Andreasson, Kai Arras, Giorgio Grisetti, Slawomir Grzonka, Kristian Kersting, Óscar Martínez Mozos, Patrick Pfaff, Christian Plagemann, Axel Rottmann und Cyrill Stachniss. Bei Kristian Kersting bedanke ich mich besonders für die interessanten Gespräche, die mir nicht nur in fachlicher Hinsicht sehr viele Einsichten gebracht haben. Außerdem bedanke ich mich bei den Diplomanden, mit denen ich zusammenarbeiten durfte: vielen Dank an Barbara Frank, Rainer Kümmerle, Jörg Meyer, Reimund Renner, Richard Schmidt und Kai Wurm. Die Zusammenarbeit mit ihnen hat mir in vielerlei Hinsicht geholfen bei der Erstellung der Doktorarbeit. Des weiteren bedanke ich mich bei Frank Dellaert und Dieter Fox für hilfreiche Tips für meine Arbeit. Fürs Korrekturlesen bedanke ich mich bei Giorgio Grisetti, Christian Plagemann und Shrihari Vasudevan. Danke auch an Ralf Kästner, der mich fachlich und moralisch unterstützt hat, diese Arbeit fertigzustellen. Schließlich bedanke ich mich bei meiner Mutter für ihre vielen hilfreichen Ratschläge, ihre materielle und ideelle Unterstützung und ihr Verständnis. Danke!

Erklärung

Contents

1 Introduction **1**
1.1 The Individual Steps of 3D Perception 3
1.2 Structure of the Thesis 4
1.3 Contributions of this Thesis 5
1.4 Related Work 7
1.4.1 3D Data Acquisition with Laser Range Finders 7
1.4.2 Plane Extraction 8
1.4.3 3D Data Representations 8
1.4.4 Global Registration and SLAM 9
1.4.5 3D Object Recognition and Scene Analysis 11
1.4.6 3D Outdoor Mapping with Cars 12
1.4.7 Occlusion Handling in 3D Range Scans 12

I Basics **15**

2 Mathematical Foundations and Notation **17**
2.1 Introduction 17
2.2 Basics from Linear Algebra 17
2.2.1 Notation 17
2.2.2 Matrix Properties 18
2.3 Dot product, Outer Product, Cross Product 18
2.4 Points, Lines and Planes 20
2.4.1 Distances 21
2.5 Transforms in 3D 22
2.6 Model fitting 23
2.6.1 Fitting Planes into Point Sets 23
2.6.2 Fitting Planes into Line Sets 25

3 Probabilistic Reasoning with Graphical Models **29**
3.1 Basics from Probability Theory 29
3.1.1 Notions and Notations 29

3.2 Graphical Representations . . . 30
3.3 Directed Graphs . . . 31
3.3.1 Bayesian Networks . . . 32
3.4 Undirected Graphs . . . 32
3.4.1 Markov Networks . . . 33
3.4.2 Conditional Markov Random Fields . . . 33

II 3D Map Building 35

4 Representations of 3D Range Data 37
4.1 Introduction . . . 37
4.2 Point Clouds . . . 38
4.2.1 Computation of Point Clouds from Laser Range Data . . . 38
4.3 Tree Structures . . . 41
4.3.1 3D-trees . . . 41
4.3.2 OBB-trees . . . 43
4.4 3D Occupancy Grids . . . 43
4.4.1 Mathematical Formulation . . . 43
4.4.2 An Application Example:
Using 3D Occupancy Grids for Autonomous Exploration . . . 45
4.5 3D Triangulations . . . 49
4.5.1 Local Triangulations . . . 49
4.5.2 Contouring Algorithms . . . 51
4.5.3 Alpha-shapes . . . 52
4.5.4 Comparison of 3D Triangulations . . . 53
4.6 Summary of Standard 3D Data Representations . . . 54

5 Planar Approximations 57
5.1 Introduction . . . 57
5.2 Plane Extraction: The State of the Art . . . 58
5.2.1 Region Growing . . . 58
5.2.2 Hough Transform . . . 60
5.2.3 Edge-based Approach . . . 61
5.3 A Probabilistic Approach . . . 64
5.4 First Extension: Introduction of Main Directions . . . 67
5.4.1 Estimating the Model Complexity . . . 68
5.4.2 Implementation Details . . . 69
5.5 Second Extension: Use of Color Information . . . 75
5.5.1 Experimental Results . . . 76
5.6 Conclusions . . . 80

6 Map Representations for Localization and Planning **81**
6.1 Introduction 81
6.2 Requirements for Localization and Planning 82
6.2.1 Localization 82
6.2.2 Planning 82
6.3 Elevation Maps 83
6.3.1 Main Concepts 83
6.3.2 Historical Background of Elevation Maps 85
6.4 Extended Elevation Maps 85
6.4.1 Measurement Update 85
6.4.2 Cell Analysis 86
6.5 Multi Level Surface Maps 88
6.5.1 Map Representation 88
6.5.2 Map Creation 88
6.5.3 Map Update 90
6.5.4 Feature extraction 91
6.5.5 Map Annotation 92
6.5.6 Visualization 92
6.6 Conclusions 94

7 Globally Consistent Maps **95**
7.1 Introduction 95
7.2 Map Registration 96
7.2.1 Registering Local Maps: The ICP Algorithm 96
7.2.2 Variants of the ICP Algorithm 99
7.2.3 Registration of MLS maps using ICP 100
7.3 Closing the Loop using Local Constraints 101
7.3.1 Network-based Pose Optimization 102
7.3.2 Local Constraints between Robot Poses 102
7.3.3 The Optimization Algorithm 103
7.3.4 Experiments 104
7.4 Closing the Loop using Local and Global Constraints 107
7.4.1 Extension of the Pose Network 108
7.4.2 Local and Global Constraints in a Common Error Function 108
7.4.3 Details of the New Optimization Algorithm 111
7.4.4 Experimental Results 113
7.5 Conclusions 115

III 3D Object Recognition **117**

8 Object Classification by Supervised Learning **119**
8.1 Introduction 119
8.2 Supervised Learning 120

8.2.1 Formulation of the Followed Approach . . . 121
8.3 The Naïve Bayes Classifier . . . 121
8.3.1 Discussion . . . 122
8.4 The Nearest-Neighbor Classifier . . . 122
8.4.1 Discussion . . . 123
8.5 AdaBoost . . . 123
8.5.1 Multiple Classes . . . 124
8.5.2 Discussion . . . 125
8.6 Conclusion . . . 125

9 Collective Classification 127
9.1 Introduction . . . 127
9.2 Associative Markov Networks . . . 128
9.2.1 Learning and Inference with AMNs . . . 129
9.3 Reduction of the Training Data . . . 130
9.3.1 Adaptive Reduction . . . 131
9.3.2 Parameterless Reduction . . . 132
9.3.3 Tests on a Real Data Set . . . 134
9.3.4 Feature Extraction . . . 134
9.3.5 Building the Markov Network . . . 134
9.3.6 Evaluation . . . 135
9.4 Instance-based Extension . . . 135
9.4.1 The Transformed Feature Vector . . . 135
9.4.2 *M* Nearest Neighbors . . . 136
9.4.3 Implementation Details . . . 137
9.4.4 Experimental Results . . . 137
9.5 Conclusions and Future Work . . . 140

IV Applications 143

10 3D Mapping with a Robotic Car 145
10.1 Introduction . . . 145
10.2 Description of the System . . . 146
10.2.1 The Localization Module . . . 146
10.2.2 The 3D Mapping Module . . . 148
10.3 Experimental Results . . . 149
10.4 Conclusions and Future Work . . . 152

11 Occlusion Handling 153
11.1 Introduction . . . 153
11.2 Description of the Algorithm . . . 154
11.2.1 Clustering . . . 154
11.2.2 Entity Matching . . . 155

11.2.3 Mesh Generation and Remapping 155
11.3 Experimental Results 157
11.4 Conclusions and Future Work 158

12 Discussion 161
12.1 Conclusions 161
12.1.1 Main Concept Followed in this Thesis 161
12.2 Future Work 163
12.2.1 Efficient and Consistent 3D Mapping 163
12.2.2 3D Object Recognition and Scene Analysis 163

List of Figures

1.1 The individual steps of 3D perception. 3

2.1 Normal vector 20
2.2 Distance between a point and a line 21
2.3 Plane fitting 25

3.1 Graphical model of conditional independency 31

4.1 Point cloud example 38
4.2 4 DOF manipulator 39
4.3 Local reference frame of the laser. 40
4.4 A possible exploration sub-path 47
4.5 3D occupancy grid 48
4.6 Local triangulation of a point cloud 50
4.7 3D Triangulations 53

5.1 Edge detection in range scans 62
5.2 Bayesian networks for hierarchical EM 65
5.3 Example of plane extraction with hierarchical EM 69
5.4 Hierarchical EM result on real data 72
5.5 Summary of results for EM plane extraction 73
5.6 Hierarchical EM result on simulated data 74
5.7 Plane extraction from simulated data 75
5.8 Colored point cloud data of a hallway 77
5.9 Result of the color plane extraction 78
5.10 Real world example for color-based plane extraction 79
5.11 Simulated scene used for color plane extraction 80

6.1 Comparison of four different 3D map representation 84
6.2 Measurement variance of the height values 86
6.3 Point cloud classification of an outdoor scan 87
6.4 The structure of an MLS map 89
6.5 Updating an MLS map 90

6.6 Visualization of MLS maps 93

7.1 Example of ICP registration 100
7.2 Closing a loop of robot positions 102
7.3 The robot Herbert 104
7.4 An MLS map of the campus 106
7.5 An MLS map of the train station 107
7.6 Extended robot pose network 108
7.7 Loop closing on a simulated data set 113
7.8 Loop closing on a real data set 114
7.9 Statistical analysis of the registration 115

9.1 Processing time and classification results 131
9.2 The AMN after data reduction 132
9.3 Training data and result of AMN classification 133
9.4 More results of AMN classification 133
9.5 Feature transformation on a synthetic data set 136
9.6 Results of AMN classification on an occupancy grid map 138
9.7 Results of AMN classification for indoor scene 139
9.8 Classification rates for indoor scenes 140
9.9 Statistical analysis of the iAMN classifier 140

10.1 The 3D scanning outdoor robot S T 146
10.2 Localization of the Smart on EPFL campus 147
10.3 A single MLS map 148
10.4 MLS Map of the EPFL campus in Lausanne 150
10.5 MLS map of a military test site 151

11.1 Initial steps of the occlusion handling algorithm 156
11.2 Clustering and entity matching step 157
11.3 Mesh generation 158
11.4 Result of occlusion handling 159

List of Tables

4.1 3D triangulations 53
4.2 3D data representations 54

5.1 Results of plane extraction from real data 72

7.1 Comparison of registration algorithms 115

9.1 AMN classification results 141

List of Algorithms

1 Nearest neighbor search with *kD*-trees 42
2 Finding planes with region growing 59
3 Iterative closest point 97
4 Robot pose optimization 111
5 AdaBoost 124

There are things known and there are things unknown, and in between are the doors of perception

Aldous Huxley (1894-1963)

1

Introduction

During the last two decades, the use of mobile robots has gained substantially increasing attraction. Nowadays, mobile robots are used more and more in real world applications. Examples include the automated loading and unloading of goods in harbors, inspection of pipelines that are dangerous or difficult to reach, or cleaning tasks in large-scale environments such as airports. In order to perform these tasks reliably, a robot first needs to be able to sense its environment accurately. To this end, many different kinds of sensors are available, such as ultrasound sensors, CCD cameras, laser range finders, GPS receivers, inertial measurement units, or odometers. The data that is acquired from these sensors needs to be processed by the robot to perform appropriate actions. In principle, there are two possible ways to choose these actions. The first one is a direct response to the input data, where in the simplest case a table is stored that maps a certain data input to a corresponding action. This is called the *reactive* model. The second method is to maintain a representation of the environment and to decide upon an action by using the current sensor input and the knowledge taken from the stored representation. This is called the *model-based* approach. In general, robots that use a model of the environment to decide on their actions are more reliable and effective than reactive robots. For example, a cleaning robot that is provided with a map of the area to be cleaned can plan the most effective path so that the entire area is cleaned equally well, whereas a reactive cleaning robot that simply turns whenever it hits an obstacle might not reach some areas to be cleaned in a long time. However, when using models of the environment, three major questions arise. First: how can we store the highest amount of needed information with the lowest amount of memory? Second: how can we deal with erroneous measurements from the sensors when using the sensor input to create environment models? And finally: what type of information should be stored in an environment model?

The first question deals with the problem of a compact representation of the environment that still contains the information needed for specific tasks such as navigation. Most existing robot systems use two-dimensional map representations for this purpose. These 2D maps have the advantage that they require a comparably small amount of memory while still providing a good approximation of the real world, especially in indoor environments. However, in many cases they fail to reflect the three-dimensional world accurately enough to perform important tasks such as obstacle avoidance. Therefore, in this work, we will address the problem of three-dimensional representations.

The second question addresses the problem of sensor noise and sensor imperfections such as spurious measurements. These errors can lead to imprecise mappings of the environment and – even worse – they can accumulate during the time of the mapping process, which often makes the resulting environment map useless. In this thesis, we will address this problem particularly for the case of 3D maps by providing a probabilistic mapping technique than can deal with sensor noise and yields globally consistent maps.

The last question tries to clarify what kind of information is needed for a given task of a robot. Many different approaches have been proposed for particular applications and there is so far no technique that seems to be appropriate in all cases. Most of the existing mapping approaches can be divided into one of two major categories: metric maps and topological maps. A metric map tries to represent the geometry of the environment as accurate as possible, whereas a topological map only contains semantic information such as the type of places in the world and how they are connected to each other. In this work, we will mainly deal with metric maps, because they are more useful for navigation tasks that require a high precision. However, we will also consider to enrich these metric maps with semantic information. This information will consist in annotations that are added to the map representation and express the type of objects that are encountered in the map. With such an annotated map, the robot can better reason about its environment and the interaction between robots and humans becomes easier. For example, a robot that operates in a home environment may detect different kinds of objects in different rooms and use this information to conclude where the kitchen, the bathroom or the living room is. A human user might then be able to communicate with the robot on a higher level by giving it commands like 'go to the kitchen', instead of having to show the robot where the kitchen is. In this work, we will not consider the problem of how this high-level reasoning about the environment can be done. Instead, we focus on the task of annotating the map with class labels such as 'chair', 'table', etc. We will formulate this in a supervised learning framework, i.e. all objects that might potentially be recognized need to be shown to the robot beforehand. By extracting features from the 3D input data and using an algorithm that maps features to class labels, the robot is enabled to *learn* the shape of the objects.

For the scope of this work, we subsume all these three mentioned questions under the notion of *perception*. By perception we mean a mapping from sensor inputs to a map representation that is *efficient*, *consistent* and incorporates *semantic information*. For the implementation of the perception task, we will split it into three steps: data acquisition, efficient (and consistent) representation and scene analysis, where the latter is a more general notion for the task of adding semantic information. These three steps will be described more detailed in the following.

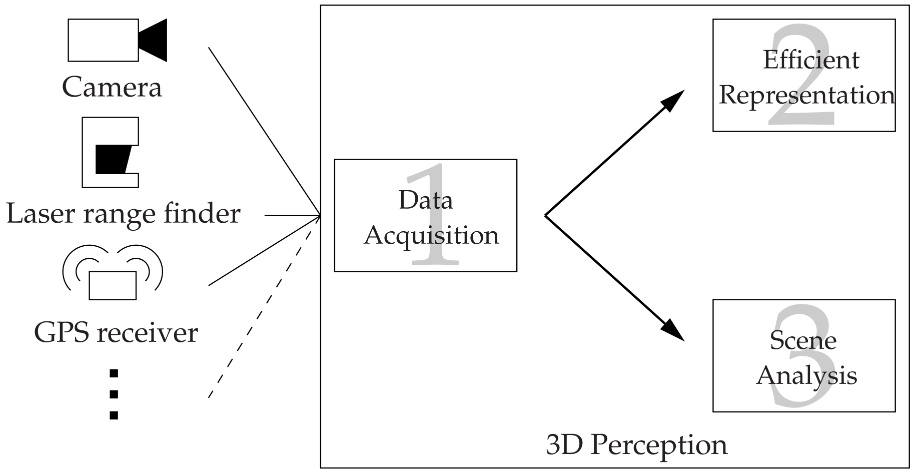


Figure 1.1: *The individual steps of 3D perception. The sensor that is mainly used in the 3D data acquisition step is the laser range finder. However, we will also consider camera data and position sensors such as GPS, IMU etc.*

1.1 The Individual Steps of 3D Perception

As mentioned above, this thesis presents a contribution to accomplishing the perception task for mobile robots where the sensor input consists mainly of three-dimensional (3D) laser range measurements. In some cases we will also consider texture information obtained from a CCD camera, and for outdoor applications we will use positioning sensors such as a GPS receiver or an inertial measurement unit (IMU). These positioning sensors will be needed to acquire consistent 3D range data from a moving vehicle. All these sensor inputs are used for the first out of three steps in the perception process, which is the *data acquisition* process. The other two steps are *efficient representation* and *scene analysis*. Figure 1.1 shows a graphical description of all three steps. In the following, we will further describe the efficient representation and the scene analysis, because they are particularly addressed in this thesis.

Efficient Representation

In this step, the goal is to extract the information that is needed for the robot to accomplish certain tasks and to reduce the amount of data acquired in the first step. The kind of information that is extracted here depends strongly on the goal tasks of the robot. In our case of a mobile robot, we will consider the navigation task, which includes path planning and localization. The particular way this navigation task is accomplished is out of the scope of this thesis. Instead, we will consider it as a design goal for an efficient data representation in the second step of 3D perception. Two major aspects will be relevant in this second step: *consistency* and *efficiency*. Here, consistency means that the reduced data set obtained from the input data should be as close as possible to the real environment in which the data was acquired. As an example, this includes the reduction of alignment errors between individual local views of the environment as well as a reliable estimation of the sensor noise. By efficiency we mean that the acquired 3D input data should be reduced as much as possible, while still retaining the necessary information. The efficiency aspect is very important, because usually the amount of raw data acquired with 3D range sensors grows very large. In some sense there is a trade-off between efficiency and consistency, because a very strong reduction of the input data tends to be a poor estimate of the real environment and therefore gets inconsistent.

Scene Analysis

The third step in the 3D perception task adds semantic knowledge to the acquired data. This part distinguishes the 3D perception from the mere mapping task, because it introduces a *cognitive* aspect. It also brings the entire approach closer to what we usually associate with human perception, because it utilizes previously acquired knowledge to better understand and interpret the sensor input. As we will see, this knowledge can be acquired from 3D training data where the scene analysis step has been performed by a human beforehand. This is called *supervised learning*. There are two major reasons why a robot needs a good scene analysis included in the perception task. First, it enables the robot to further reduce the amount of data required to represent the environment, because it provides a higher level of abstraction. Second, it facilitates the robot's interaction with humans, and thus enlarges the range of applicability of robotic systems. As an example, consider an application where a robot explores an unknown environment and provides a human user with an annotated 3D map. In this case, the human does not have to do the map annotation himself and he can communicate with the robot on a higher level, e.g. by giving it commands like "Show me the nearest room".

In Fig. 1.1, we depict the scene analysis as a step that directly follows the data acquisition and not the map building step, as the step ordering would suggest. This reflects the fact that the scene analysis, as it is presented in this thesis, is somehow independent of the mapping process. The advantage of this is that no information needed for the scene analysis can get lost because we operate on the raw and uncompressed data. Of course, one could think of a consecutive ordering of the steps. However, it is unclear whether it is better to do the scene analysis on the reduced map representation or to create a map from the annotated raw data. In the first case we might not be able to extract some important features from the reduced data for the scene analysis, in the second case the raw data might be too large to perform the scene analysis efficiently. For the scope of this thesis, we will focus on the formulation and analysis of efficient algorithms for both steps and leave the problem of an appropriate combination as future work.

1.2 Structure of the Thesis

This thesis is organized as follows: In the remainder of this chapter we present the major contributions made in this thesis and discuss the related work. Then, the thesis is divided into four parts. Part I consists of chapters 2 and 3 and introduces the basic mathematical concepts that are used later in the thesis. In particular, these are 3D geometry formulations (chapter 2) and probabilistic reasoning techniques (chapter 3).

Part II is dedicated to the first and second step in the perception task. In chapter 4 we discuss and compare the state-of-the-art methods to represent 3D data. We also give an example of a 3D scanning device that consists of a manipulator with 4 degrees of freedom and is able to autonomously acquire 3D point cloud data from its environment. Chapter 5 gives an overview of existing approaches to efficiently represent 3D data by approximating it with planar structures. We also present a new method based on a hierarchical expectation maximization (EM) algorithm to find planes in 3D point clouds. In chapter 6 we focus on data structures that are particularly useful for the planning and localization task. Here, we develop a new approach that solves the trade-off between efficiency and consistency mentioned above. In chapter 7 we further address the consistency issue for the case that several local maps have to be joined into one global map. In this context, we suggest an improved global map registration algorithm that uses knowledge about the environment for a more robust alignment of the local maps.

In Part III we are concerned with the scene analysis task. Chapter 8 summarizes the major approaches that have been presented in this context. Although these approaches have been applied successfully in the past, for example AdaBoost for face detection in camera images, they lack the ability to model the statistical dependence of data points that are close to each other. Recent methods that can deal with this statistical dependence are known as collective classification algorithms and are the subject of chapter 9. Here, we will present two novel extensions to an approach based on associative Markov networks (AMNs), which lead to a faster performance of the learning step and a more robust classification result.

Finally, in part IV, we present two example applications of the techniques developed throughout the thesis. In chapter 10 we describe a robotic system that consists of a car equipped with positioning sensors, 3D laser range finders and projective and omnidirectional cameras. The goal of the system is to autonomously drive through an off-road or urban environment while creating consistent 3D maps. We describe in detail, how the mapping task is performed and show that it is possible to do this in real-time while the vehicle moves. In chapter 11 we present an application of the scene analysis algorithm developed in chapter 9. We will show how the problem of recovering the shape of objects that are partially occluded by other objects can be addressed. This is done using a robust object classification and a clustering algorithm that uses the shape of similar objects in the same scene to learn object prototypes. Chapter 12 concludes the thesis and discusses future work.

1.3 Contributions of this Thesis

The work presented in this thesis is based on several publications in major conferences and journals. In the following, we group these by subjects and name the corresponding chapters in the thesis.

- Assembly of a 3D scanning system with path planning and exploration capabilities (see Chapter 4, especially Section 4.4.2)

  R. Triebel, B. Frank, J. Meyer, and W. Burgard. First steps towards a robotic system for flexible volumetric mapping of indoor environments. In M.I. Ribeiro and J. Santos-Victor, editors, *5th IFAC/Euron Symposium on Intelligent Autonomous Vehicles*, 2004.

- Plane extraction from 3D range scans (see Chapter 5)

  R. Triebel, W. Burgard, and F. Dellaert. Using hierarchical EM to extract planes from 3d range scans. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

  H. Andreasson, R. Triebel, and W. Burgard. Improving plane extraction from 3d data by fusing laser data and vision. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2005.

- A compact and efficient 3D map representation for outdoor environments (see Chapter 6)

  R. Triebel, P. Pfaff, and W. Burgard. Multi level surface maps for outdoor terrain mapping and loop closing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

  P. Pfaff, R. Triebel, and W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *International Journal of Robotics Research*, 26(2), 2007.

- Globally consistent 3D map registration and SLAM (see Chapter 7)

  R. Triebel and W. Burgard. Improving simultaneous mapping and localization in 3d using global constraints. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2005.

  P. Lamon, S. Kolski, R. Triebel, R. Siegwart, and W. Burgard. The SmartTer for ELROB2006 – a vehicle for fully autonomous navigation and mapping in outdoor environments. Technical report, Ecole Polytechnique Fédérale de Lausanne and Albert-Ludwigs-Universität Freiburg, 2006.

- 3D object recognition from laser range data (see Chapter 9)

  R. Triebel, K. Kersting, and W. Burgard. Robust 3d scan point classification using associative markov networks. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2006.

  R. Triebel, R. Schmidt, Ó. Martínez Mozos, and W. Burgard. Instance-based AMN classification for improved object recognition in 2d and 3d laser range data. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2007.

  Ó Martínez Mozos, R. Triebel, P. Jensfelt, A. Rottmann, and W. Burgard. Supervised semantic labeling of places using information extracted from sensor data. *Robotics and Autonomous Systems*, 55(5), 2007.

- Application I: 3D mapping with a robotic car (see Chapter 10)

  P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, W. Burgard, and R. Siegwart. Towards mapping of cities. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

  P. Lamon, C. Stachniss, R. Triebel, P. Pfaff, C. Plagemann, G. Grisetti, S. Kolski, W. Burgard, and R. Siegwart. Mapping with an autonomous car. In *Workshop on Safe Navigation in Open and Dynamic Environments,International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2006.

- Application II: Resolving occlusions in 3D range scans (see Chapter 11)

  R. Triebel and W. Burgard. Recovering the shape of objects in 3d point clouds with partial occlusions. In *Proc. of Field and Service Robotics*, 2007. (to appear).

1.4 Related Work

In the following, we name the most relevant publications that are related to the context of this thesis. We will order them roughly by the topics of the major contributions that have been mentioned in the previous section. Some of the references might be cited under two or more of these topics, because they are relevant in different contexts.

1.4.1 3D Data Acquisition with Laser Range Finders

In the past, many different authors have presented systems that are able to acquire three dimensional range data by using laser range finders. These systems usually rely on scanning devices that acquire range measurements in a plane that intersects the environment. By moving such a 2D scanner in a predefined way, 3D point clouds are obtained. For example, Allen et al. [2001] use a laser range scanner to obtain realistic 3D models of buildings. Pervölz et al. [2004] use a 2D laser scanner that pitches up and down to scan indoor environments. Früh and Zakhor [2004] present a system to map cities in 3D using two laser scanners mounted on a truck, one vertically and one horizontally. Thrun et al. [2000] also use two laser range scanners. The first is oriented horizontally to the front and the second points towards the ceiling. By registering the horizontal scans the system generates accurate three-dimensional models. An application of this was presented by Thrun et al. [2003a] to obtain 3D models of underground mines. The same setup with two scanners has been used by Hähnel et al. [2003] in indoor and outdoor applications, as well as a robotic system that is equipped with a pan/tilt unit that carries a laser scanner. Montemerlo and Thrun [2004] also use a pan/tilt unit, but the scanner is mounted vertically and sweeps from left to right and back to scan the environment. A different approach presented by Kohlhepp et al. [2003] uses a laser scanner mounted on a mobile robot, where the scanner points forward and rotates around the front axis. In a similar way, a rotating 2D laser setup is chosen by Wulf et al. [2004], where the scanner rotates around the vertical axis and is mounted either vertically or pointing upwards.

In addition to these approaches which are mainly used in the context of mobile robotics, several authors have studied the problem of creating high-resolution models of scientifically interesting objects. For example, in the Michelangelo project presented by Levoy et al. [2000], historic statues were scanned with a high-resolution 3D scanning system. Lyons et al. [2000] present a system to obtain 3D models of fossils with a scanner resolution of $100\mu m$. In these approaches the major focus lies on the question of data acquisition and registration of the individual scans. Color information is typically mapped onto the resulting models after creating the three-dimensional structures.

All these approaches for 3D data acquisition have in common that the number of degrees of freedom in which the scanner is moved is at most two, and that the scanning process is passive. In chapter 4, we will present a new method to actively acquire 3D range data using a manipulator with four degrees of freedom. By active scanning we mean that the motion of the scanner is planned according to the surface structure of the objects to be scanned. This way, we obtain a more efficient scanning process.

1.4.2 Plane Extraction

The problem of extracting planes from three dimensional range data has been addressed by many authors in the last two decades. We will give an overview of the existing approaches and a more detailed comparison later in chapter 5. At this point, we only name the most recent and most relevant contributions. For example, Hähnel et al. [2003] use a region growing technique to detect planes in 3D laser range scans. Kohlhepp et al. [2003] apply a modified split-and-merge technique based on point features such as curvature to obtain planar surface patches. Qiang et al. [2003] apply a combination of region-growing and planar patch merging. A region-growing technique is also applied by Weingarten and Siegwart [2005], however not directly on the data points, but on planar patches which are detected using RANSAC on points in regular 3D grid cells.

Recently, some authors have used a new framework that addresses the plane extraction problem in a probabilistic way. The idea here is to model the uncertainty of the range measurements and to find a set of planes that maximizes the likelihood of the observed data using expectation maximization (EM). For example, Liu et al. [2001] use EM to find planes from 3D range scans. Thrun et al. [2003b] propose an online version of the EM based plane extraction that runs in real-time. Most recently, Lakaemper and Latecki [2006] have presented an extended version of the EM based plane extraction by inserting a split-and-merge step that operates on planar patches rather than on infinite planes.

The novel approach which we will present in chapter 5 also applies the concept of probabilistic modeling and uses EM. However, it extends the EM algorithm by using a hierarchical Bayes net and estimates also *main directions* in addition to the position of the most likely planes. In a further extension, we also use color information at each data point and incorporate an extended metric based on color information into the EM framework.

1.4.3 3D Data Representations

In the area of efficient representations for three dimensional range data, several different approaches have been used in the literature, which we will briefly summarize here. In chapter 4 we will provide a more detailed overview. One of the most popular 3D data representations are point clouds or triangle meshes. They have been used amongst others by Allen et al. [2001], Levoy et al. [2000], Pervölz et al. [2004], and Thrun et al. [2003b]. These representations are highly accurate and can easily be textured, but their disadvantage lies in the huge memory requirement, which grows linearly in the number of scans taken. An alternative is to use three-dimensional grids, as presented by Moravec [1996] or tree-based representations such as octrees (see Samet [1989]) or *kd*-trees (see Bentley [1975]). As we will see, these approaches are more efficient than point clouds with respect to memory requirements, but they are still too memory intensive and – in the case of the tree structures – can not be updated efficiently. Therefore, several researchers have considered *elevation maps* as an attractive alternative to avoid the complexity of full 3D maps. The key idea underlying elevation maps is to store the height information of the terrain in a two-dimensional grid. For example, Bares et al. [1989] as well as Hebert et al. [1989] use elevation maps to represent the environment of a legged robot. They extract points with high surface curvatures and match these features to align maps constructed from consecutive range scans. Parra et al. [1999] represent the ground floor by elevation maps and use stereo vision to detect and track objects on the floor. Singh and Kelly [1996] extract elevation maps from laser range data and use these maps for navigating an all-terrain vehicle. Ye and Borenstein [1994] propose an algorithm to acquire elevation maps with a moving vehicle carrying a tilted laser range scanner. They propose special filtering algorithms to eliminate measurement errors or noise resulting from the scanner and the motions of the vehicle. Lacroix et al. [2002] extract elevation maps from stereo images. Hygounenc et al. [2004] construct elevation maps with an autonomous blimp using 3D stereo vision. They propose an algorithm to track landmarks and to match local elevation maps using these landmarks. Olson [2000] describes a probabilistic localization algorithm for a planetary rover that uses elevation maps for terrain modeling.

The major disadvantage of elevation maps is the fact that at each *xy*-position always the highest *z*-value is stored. This causes problems in the presence of overhanging objects such as trees or when vertical gaps need to represented, for example in the case of bridges. The area under a bridge is represented as a huge obstacle which makes path planning impossible. This problem has been addressed by Pfaff and Burgard [2005]. They use *extended elevation maps*, which identify map cells that contain vertical gaps and thus enable path planning under bridges. In chapter 6, we will extend this approach even further by introducing *multi-level surface maps*. These maps can represent several height values at each *xy*-position and provide the possibility of planning a path passing under bridges and crossing over them at the same time. In a way, this is comparable to an approach by Wellington et al. [2005], where each 2D grid cell stores a vertical *voxel column*. However, in contrast to that work, MLS maps are not restricted to a particular ordering of the types of terrain in a voxel column. Furthermore the height values are not discretized, which leads to a more accurate representation.

1.4.4 Global Registration and SLAM

Whenever 3D laser range scans at several different positions are given, the problem arises how to combine these local views into a consistent global map. This is called *registration*. The most common technique to do 3D scan registration between two local views is the iterative closest point (ICP) algorithm by Besl and McKay [1992]. A similar approach presented by Chen and Medioni [1991] minimizes the point-to-plane distance between the views instead of the point-to-point distance. Another extension of ICP is presented by Granger and Pennec [2002]. They formulate the problem as a general maximum-likelihood estimation that also incorporates the sensor noise variance and use expectation maximization (EM) for the registration. They show that in the case of Gaussian noise the algorithm is equal to ICP. In addition, several authors have considered to register all local views of a 3D object at once. For example, Pulli [1999] present a multi-view registration algorithm based on constraints that are derived from pairwise view matches. Huber and Hebert [2003] build up a *model graph* of single views and perform the global registration on a spanning tree of this graph. In a similar way, global registration of 3D range scans is done by Stamos and Leordeanu [2003], where each scan is aligned to a common *anchor scan* and

Dijkstra's algorithm is used to find a transformation path. Nüchter et al. [2005] align a set of 3D indoor range scans by applying pairwise standard ICP and use a region growing technique to register overlapping neighbor scans. Furthermore, there has been some work on incorporating color information during the registration, such as the color ICP presented by Johnson and Kang [1997] or the extension using illuminance and gradient information proposed by Weik [1997]. Both of these approaches estimate the registration from camera images only. A combination of laser and vision information is used in the work by Yoshida and Saito [2002] who use texture information around each scan point to select corresponding points for the registration of two scans.

Closely related to the global registration of 3D scans is the problem of simultaneous mapping and localization (SLAM). It has been studied intensively in the past, where the individual approaches can be classified along multiple dimensions which include important aspects like the type of the representation of the environment and the question of how the posterior about the robot's pose and the map is represented. Extended Kalman Filter methods belong to the most popular approaches and different variants of this technique have been proposed by Dissanayake et al. [2001], Guivant et al. [2000], Leonard and Feder [1999], Moutarlier and Chatila [1989], and Thrun et al. [2004]. The key idea of these techniques is to simultaneously estimate the poses of landmarks using an Extended Kalman Filter or a variant of the Extended Information Filter. An alternative approach is to compute the most likely map based on a graph of spatial relations. Approaches that use such spatial relations have been proposed by Frese [2004], Gutmann and Konolige [1999], Konolige [2004], and Lu and Milios [1997]. The advantage of such methods is that they do not require predefined landmarks. Rather they can cope with arbitrary representations by considering so-called constraints between the poses where observations of the environment were perceived. However, most of these approaches rely on the assumption that the environment can be represented by a two-dimensional structure. In contrast, Davison et al. [2004] presented an approach to *vision-based* SLAM with a single camera moving freely through the environment. This approach uses an extended Kalman Filter to simultaneously update the pose of the camera and the 3D feature points extracted from the camera images.

The global 3D scan registration method which we will present in chapter 7 differs in two major aspects from the approaches mentioned here. First, it applies the graph-based global pose estimation scheme known from the 2D case to the 3D registration problem. Thus, it does not require any edge elimination method of the model graph, such as spanning tree, Dijkstra or region growing and hence does not disregard any information. Second, it distinguishes between local and global constraints between the particular robot poses. A local constraint is imposed by two consecutive poses with overlapping range scans and a global constraint is introduced whenever two or more local scans contain features (in our case lines), that belong to a common model (in our case planes). We show that by using these global constraints the algorithm can cope with fewer scans that even have a smaller overlap. Also, no loop closing is required to obtain a globally consistent representation.

1.4.5 3D Object Recognition and Scene Analysis

As mentioned in Section 1.1, the second major step in the perception process is the acquisition of semantic knowledge about the environment. This can be performed in several ways, for example by classifying the objects that are encountered in the environment or by annotating the acquired data with more high-level information, e.g. by labeling rooms and hallways. For the latter application, there has been an approach presented by Buschka and Saffiotti [2002], in which rooms and hallways are detected in 2D occupancy maps by using image processing algorithms. The aim of the work is to obtain topological maps of the environment. In a more recent work by Martínez Mozos and Burgard [2006], the same problem has been addressed using a supervised learning technique, which is more flexible and requires less human input at the design level. However, these approaches operate on 2D occupancy maps and not on 3D point clouds, which is the aim of this thesis. Instead, we will focus on the 3D object recognition task, which can be considered as an important first step towards a more elaborate scene analysis. In the context of oject recognition, the approaches that have been presented previously mainly differ in the selection of the features to be extracted and in the learning strategies. For example, *spin images* have been introduced as a type of rotation invariant features by Johnson [1997]. Also, de Alarcón et al. [2002] use spin images and an indexing schema based on artificial neural networks to retrieve 3D objects from a database. Ruiz-Correa et al. [2003b] apply spin images to recognize deformable shapes. Frome et al. [2004] extend spin images to *point descriptors* and apply a voting technique to recognize objects in range data. Vandapel et al. [2004] extract saliency features based on the eigenvalues of local covariance matrices and apply EM to learn a Gaussian Mixture Model classifier. Another object description technique is called *shape distributions* and has been applied by Osada et al. [2001]. A shape distribution is defined by a histogram of distances between randomly sampled points on the surface of a 3D object. Furthermore, several other types of 3D features have been proposed, including local tensors (see Mian et al. [2004]), shape maps (see Wu et al. [2004]), and multi-scale features (see Li and Guskov [2005]).

A framework for 3D object detection that is based on detecting object *parts* has been presented by Huber et al. [2004]. The idea of this is to learn a set of classes of object parts and a mapping between part classes and object classes. A similar idea of detecting object components has been presented by Ruiz-Correa et al. [2003a], who introduced symbolic surface signatures to encode the geometrical relationships between the parts. Another approach to object recognition proposed by Boykov and Huttenlocher [1999] is based on Markov random fields (MRFs). They classify objects in camera images by incorporating statistical relationships between nearby object parts. Wellington et al. [2005] also use MRFs to do environment classification for agricultural applications. They compute the elevation of the cell depending on the classification of the cell and its neighbors. A relational approach using MRFs has been proposed by Limketkai et al. [2005]. The idea here is to exploit the spatial relationship between nearby objects, which in this case consist of 2D line segments. A similar idea is the basis of the work presented by Anguelov et al. [2005], which uses a specialized instance of relational MRFs called associative Markov networks (AMNs). AMNs provide an efficient implementation of the learning and the inference step.

The new approach for 3D object recognition which we will present in chapter 9 is mostly related to this last mentioned work by Anguelov et al. [2005]. It also uses AMNs in a supervised learning approach and introduces two major extensions to the classification algorithm. First, it adaptively selects data points from the training data set and uses these points as representatives for neighboring points. This way, the training data set is reduced in size to an abstraction of the original range scan. As a result, our approach requires less complex constraints and, in turn, yields a faster training phase without decreasing classification rates. Second, we combine the AMN approach with techniques from instance-based classification to address the problem that AMNs are restricted to linear separable feature spaces. By previously transforming the feature space, this restriction is abolished and we obtain improved classification results.

1.4.6 3D Outdoor Mapping with Cars

In the context of autonomous cars, a series of successful systems have been developed amongst others by Cremean et al. [2006], Thrun et al. [2006], and Urmson [2005] for the DARPA Grand Challenge [2004], which was a desert race for autonomous vehicles along an approximatively 130 mile course. As a result of this challenge, there exist autonomous cars that reliably avoid obstacle and navigate at comparably high speeds. The focus of the Grand Challenge was to finish the race as quickly as possible whereas certain issues like building consistent large-scale maps of the environment have been neglected since they where not needed for the race. In contrast, the novel approach towards mapping large areas, which will be presented in chapter 10 has a different aim compared to the vehicles participating to the Grand Challenge. We will apply the techniques described in the preceding chapters to obtain globally consisted large-scale maps. Nevertheless, our Smart car also benefits from different techniques used within the Grand Challenge. We apply an approach to follow a given trajectory similar to the one of the winning vehicle Stanley presented by Thrun et al. [2006].

1.4.7 Occlusion Handling in 3D Range Scans

Occlusions occur whenever an object is partially or totally hidden by another object and thus can not be perceived entirely. This may occur with most kinds of sensors such as cameras and 3D range scanners. In the literature, mostly algorithms to detect and handle occlusions have been presented that operate on camera images. A good overview and analysis of such approaches is given by Elmqvist and Tsigas [2007]. An example of an occlusion handling algorithm for two-frame stereo was presented by Sun et al. [2005]. They define a symmetric visibility constraint on both images and estimate the disparity and the occlusion iteratively in two separate steps. Another way to resolve occlusions is by using multiple cameras. For example, Fleuret et al. [2005] detect and resolve occlusions using such a multi-view approach.

To the best of our knowledge, no author has yet addressed the problem of occlusions in 3D range scans, where the work of Rodgers et al. [2005] might be seen as an exception.

They describe a method to detect different poses of partially occluded puppets and humans in 3D range scans. However, the primary goal in their work is not to resolve the occlusions, but to detect the particular poses. In chapter 11, we will present a novel technique to resolve occlusions in 3D range scans by matching all objects that correspond to the same object class to one prototype. This way, the occluded part of a particular object can be filled with the corresponding part of another object of the same type. We will show this in an example application where windows of a building are occluded by a tree and where the fact that all windows have the same shape is used to resolve this occlusion.

Part I

Basics

Do not worry about your difficulties in Mathematics. I can assure you mine are still greater.

Albert Einstein (1879 - 1955)

2 Mathematical Foundations and Notation

2.1 Introduction

In this chapter, we present the mathematical background which is used in the following chapters. Of course, it is not possible to include all mathematical formulations that are needed in this work in detail. Therefore, we focus only on the most often used formulations and refer to the literature (for example Zwillinger, 1996) where a more detailed description is needed.

2.2 Basics from Linear Algebra

2.2.1 Notation

Throughout this work, we will stick to the following notations:

- a d-dimensional *vector* will be notated with a bold letter, e.g. $\mathbf{x} = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$. If not otherwise noted, all vectors are considered as column vectors.

- a *matrix* with m rows and n columns will be denoted with a capital letter, for example

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}. \tag{2.1}$$

  The transpose of a matrix A will be denoted as A^T.

- all types of *sets* will be denoted with calligraphic capital letters. For example, a set containing N vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ will be denoted as

$$\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \tag{2.2}$$

- angles will be denoted with small Greek letters, i.e. $\alpha, \beta, \ldots$

2.2.2 Matrix Properties

Here, we summarize some important properties of matrices:

- A matrix A is called *symmetric* iff $A^T = A$
- A matrix A is called *diagonal* iff $a_{ij} = 0$ for $i \neq j$. The *identity* matrix is a special diagonal matrix and has only ones on the diagonal, i.e. $a_{ij} = 1$ for $i = j$. The identity matrix is denoted as I.
- A matrix A is called *invertible* iff $\det A \neq 0$. Otherwise, A is called *singular*. If A is invertible then A^{-1} is defined so that $A^{-1}A = I$.
- A matrix A is called *orthogonal* iff $A^T A = I$.
- A matrix A is called *square* iff the number of rows equals the number of columns, i.e. $m = n$.
- A matrix A is called *positive definite* iff it is square and $\mathbf{x}^T A \mathbf{x} > 0 \; \forall \mathbf{x} \in \mathbb{R}^n$. Accordingly, a matrix can be *positive semidefinite* (if $\mathbf{x}^T A \mathbf{x} \geq 0$), *negative definite* and *negative semidefinite*.

2.3 Dot product, Outer Product, Cross Product

For the sake of completeness we give the following definitions. They can be found in any introductory mathematical textbook (for example in Fischer [1995]).

Definition 2.1. The *dot product* or *inner product* $(\cdot)$ between two vectors $\mathbf{x} = (x_1, \ldots, x_d)^T$ and $\mathbf{y} = (y_1, \ldots, y_d)^T$ is defined as

$$\mathbf{x} \cdot \mathbf{y} := \sum_{i=1}^{d} x_i y_i = \mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x} \tag{2.3}$$

From the definition it follows that the dot product is symmetric, i.e. $\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}$. The dot product can be used to define a *norm* $||.||$ of a vector $\mathbf{x}$ in the way that $||\mathbf{x}|| := \sqrt{\mathbf{x} \cdot \mathbf{x}}$. Furthermore, the definition of an *angle* can be expressed by means of the dot product:

Definition 2.2. The *angle* α between two vectors $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^d$ is defined as

$$\alpha := \arccos\left(\frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||||\mathbf{y}||}\right) \tag{2.4}$$

From this definition, we can formulate a geometric interpretation of the dot product, because it follows that

$$\begin{aligned} \mathbf{x} \cdot \mathbf{y} &= \|\mathbf{x}\|\|\mathbf{y}\| \cos(\alpha) \\ &= \|\mathbf{x}\|\|\mathbf{y}^{\perp}\| \operatorname{sign}(\cos(\alpha)) \end{aligned} \tag{2.5}$$

where $\mathbf{y}^{\perp}$ is the orthogonal projection of $\mathbf{y}$ onto $\mathbf{x}$. In the special case that $\mathbf{x}$ has unit length, we can say that the absolute value of the dot product is the length of the orthogonal projection of $\mathbf{y}$ onto $\mathbf{x}$.

Another useful operation on vectors is the *outer product*, which is given by the following

Definition 2.3. The *outer product* between two vectors $\mathbf{x} = (x_1, \dots, x_d)^T$ and $\mathbf{y} = (y_1, \dots, y_d)^T$ is defined as

$$\mathbf{x} \circ \mathbf{y} := \begin{pmatrix} x_1y_1 & x_1y_2 & \dots & x_1y_d \\ x_2y_1 & x_2y_2 & \dots & x_2y_d \\ \vdots & \vdots & \ddots & \vdots \\ x_dy_1 & x_dy_2 & \dots & x_dy_d \end{pmatrix} = \mathbf{x}\mathbf{y}^T \tag{2.6}$$

In contrast to the dot product, the outer product is not symmetric. A possible application of the outer product can be given by the following

Example 2.1. *The covariance matrix C of a set of d-dimensional data points $\mathbf{p}_1, \dots, \mathbf{p}_N \in \mathbb{R}^d$ is defined as*

$$C := \frac{1}{N} \sum_{i=1}^{N} (\mathbf{p}_i - \mu) \circ (\mathbf{p}_i - \mu) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{p}_i - \mu)(\mathbf{p}_i - \mu)^T \tag{2.7}$$

where $\mu = N^{-1} \sum_{i=1}^{N} \mathbf{p}_i$ is the mean of the data points. From this definition we can directly derive the two main properties of covariance matrices, namely:

- *symmetry:* $((\mathbf{p_i} - \mu)(\mathbf{p}_i - \mu)^T)^T = ((\mathbf{p}_i - \mu)^T)^T(\mathbf{p}_i - \mu)^T = (\mathbf{p}_i - \mu)(\mathbf{p}_i - \mu)^T$
- *positive definiteness:* $\mathbf{x}^T((\mathbf{p_i} - \mu)(\mathbf{p}_i - \mu)^T)\mathbf{x} = (\mathbf{x} \cdot (\mathbf{p}_i - \mu))(\mathbf{x} \cdot (\mathbf{p}_i - \mu)) = (\mathbf{x} \cdot (\mathbf{p}_i - \mu))^2 > 0$

For details about symmetry and positive definiteness of matrices see Hartley and Zisserman [2000].

For the special case of vectors in 3 dimensions, we provide the following

Definition 2.4. The *cross product* between two vectors $\mathbf{x} = (x_1, x_2, x_3)^T$ and $\mathbf{y} = (y_1, y_2, y_3)^T$ is defined as

$$\mathbf{x} \times \mathbf{y} := \begin{pmatrix} x_2y_3 - y_2x_3 \\ x_3y_1 - y_3x_1 \\ x_1y_2 - y_1x_2 \end{pmatrix} \tag{2.8}$$

One property of the cross product that follows from the definition is $\mathbf{x} \times \mathbf{y} = -\mathbf{y} \times \mathbf{x}$. Geometrically, the cross product is the vector that is orthogonal to both $\mathbf{x}$ and $\mathbf{y}$, provided that $\mathbf{x}$ and $\mathbf{y}$ are linear independent, i.e. $\nexists \lambda \in \mathbb{R} : \mathbf{x} = \lambda \mathbf{y}$. Otherwise, the cross product is the zero vector.

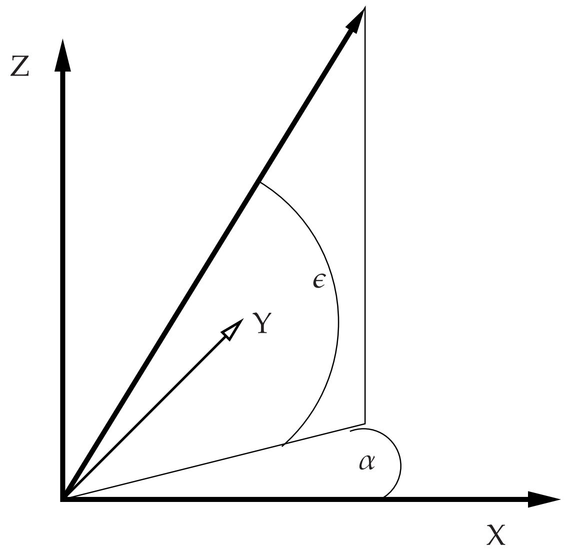


Figure 2.1: *The normal vector of a plane can be described by the* azimuth *angle α and the* elevation *angle ϵ.*

2.4 Points, Lines and Planes

In most of the formulations throughout this work, we will deal with the basic geometrical objects in the Euclidean 3D space. Therefore, we define

- a *point* as a 3D vector $\mathbf{p} \in \mathbb{R}^3$
- a *line* l as the pair of vectors $(\mathbf{p}, \mathbf{v})$ where $\mathbf{p}, \mathbf{v} \in \mathbb{R}^3$ and $\|\mathbf{v}\| = 1$. The set of all points on the line l is then defined as $\{\mathbf{x} \in \mathbb{R}^3 \mid \exists s \in \mathbb{R} : \mathbf{x} = \mathbf{p} + s\mathbf{v}\}$. This means, a line is defined by its *origin* $\mathbf{p}$ and its *direction* $\mathbf{v}$.
- a *plane* p as the pair $(\mathbf{n}, r)$ where $\mathbf{n} \in \mathbb{R}^3$, $r \in \mathbb{R}$ and $\|\mathbf{n}\| = 1$. The set of all points on the plane p is then defined as $\{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{n} \cdot \mathbf{x} = r\}$. Thus, a plane is defined by its *normal vector* $\mathbf{n}$ and its distance to the origin r.

A point in 3D has 3 degrees of freedom, which is obvious, because it is defined by the 3 components of its corresponding vector. A line has 4 degrees of freedom, which is more difficult to see. We can say that the set of all lines passing through a point $\mathbf{p}$ which is closest to the origin on the particular line, is identically defined by the three coordinates of $\mathbf{p}$. The remaining degree of freedom is the angle of rotation inside the plane $(\|\mathbf{p}\|^{-1}\mathbf{p}, \|\mathbf{p}\|)$ around the axis defined by $\mathbf{p}$. Finally, a plane has also 3 degrees of freedom, namely one for the distance r to the origin and two for the unit normal vector $\mathbf{n}$. The latter holds because $\mathbf{n}$ is identically defined by its *azimuth* α and *elevation* ϵ where

$$\begin{aligned} \alpha &= \arctan2(n_y, n_x) \\ \epsilon &= \arctan2(n_z, \sqrt{n_x^2 + n_y^2}) \end{aligned} \tag{2.9}$$

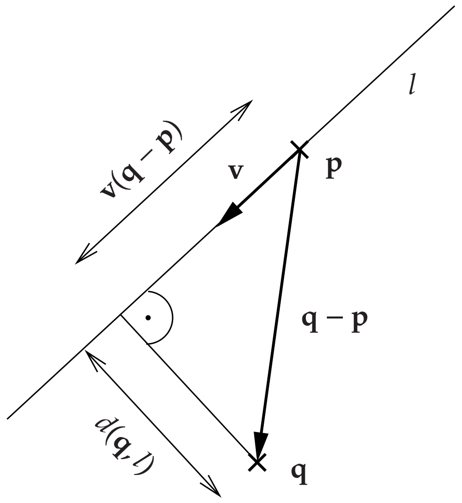


Figure 2.2: *The distance $d(\mathbf{q}, l)$ between a point $\mathbf{q}$ and a line l can be computed using a rectangular triangle. The length of the hypotenuse of this triangle equals the distance $\|\mathbf{q}-\mathbf{p}\|$ between $\mathbf{p}$ and $\mathbf{q}$. Using Equation (2.5) it can be shown that the projection of the vector $(\mathbf{q}-\mathbf{p})$ onto $\mathbf{v}$ has length $\mathbf{v}(\mathbf{q}-\mathbf{p})$. Thus, with the Pythagorean theorem we can derive Equation (2.11).*

and $\mathbf{n} = (n_x, n_y, n_z)^T$. Here, we used the arctan2 function which is defined as

$$\operatorname{arctan2}(a, b) := \begin{cases} \arctan(\frac{a}{b}) & \text{if } b > 0 \\ \operatorname{sign}(a)\frac{\pi}{2} & \text{if } b = 0 \\ \arctan(\frac{a}{b}) + \operatorname{sign}(a)\pi & \text{if } b < 0. \end{cases}$$

Figure 2.1 gives a graphical explanation of the azimuth and the elevation angles.

2.4.1 Distances

In many applications, the (Euclidean) distances between geometric objects such as points, lines and planes are needed. Therefore, we will provide the formulae to compute these distances here.

Definition 2.5. The *distance* $d(.,.)$ between two geometric objects is a binary function from the set of geometric objects into the set of non-negative real numbers $\mathbb{R}_0^+$. In particular, it is defined between

- a point $\mathbf{q}$ and a point $\mathbf{p}$ as

$$d(\mathbf{q}, \mathbf{p}) := \|\mathbf{q}-\mathbf{p}\| = \sqrt{(\mathbf{q}-\mathbf{p}) \cdot (\mathbf{q}-\mathbf{p})} \tag{2.10}$$

- a point $\mathbf{q}$ and a line $l = (\mathbf{p}, \mathbf{v})$ where $\|\mathbf{v}\| = 1$ as

$$d(\mathbf{q}, l) := \sqrt{\|\mathbf{q}-\mathbf{p}\|^2 - \|\mathbf{v} \cdot (\mathbf{q}-\mathbf{p})\|^2} \tag{2.11}$$

  Fig. 2.2 gives an explanation of how this formula is derived.

- a point $\mathbf{q}$ and a plane $p = (\mathbf{n}, r)$ where $||\mathbf{n}|| = 1$ as

$$d(\mathbf{q}, p) := |\mathbf{q} \cdot \mathbf{n} + r| \tag{2.12}$$

- a line $l_1 = (\mathbf{p}_1, \mathbf{v}_1)$ to a line $l_2 = (\mathbf{p}_2, \mathbf{v}_2)$ as

$$d(l_1, l_2) := \begin{cases} d(\mathbf{p}_1, l_2) = d(\mathbf{p}_2, l_1) & \text{if } \mathbf{v}_1 \times \mathbf{v}_2 = \mathbf{0} \\ \dfrac{|(\mathbf{p_2} - \mathbf{p}_1) \cdot (\mathbf{v}_1 \times \mathbf{v}_2)|}{||\mathbf{v}_1 \times \mathbf{v}_2||} & \text{else.} \end{cases} \tag{2.13}$$

- a line $l = (\mathbf{p}, \mathbf{v})$ to a plane $p = (\mathbf{n}, r)$ only if it is parallel to p, i.e. only if $\mathbf{v} \cdot \mathbf{n} = 0$. Then the distance between l and p is

$$d(l, p) := |\mathbf{p} \cdot \mathbf{n} + r| \tag{2.14}$$

- a plane $p_1 = (\mathbf{n}_1, r_1)$ and a plane $p_2 = (\mathbf{n}_2, r_2)$ only if the planes are parallel, i.e. if $\mathbf{n}_1 \cdot \mathbf{n}_2 = \pm 1$. Then the distance is defined as

$$d(p_1, p_2) := |r_1 - r_2| \tag{2.15}$$

2.5 Transforms in 3D

One of the most important operations on 3D objects described in the previous section are *transforms*. A transform is a mapping from $\mathbb{R}^3$ to $\mathbb{R}^3$. Several transforms exist, e.g. projective transforms, affine transforms etc. A good summary is given by Hartley and Zisserman [2000]. For our purpose we focus on affine transforms, which are given by the following

Definition 2.6. An *affine transform* T in 3D is a mapping from $\mathbb{R}^3$ to $\mathbb{R}^3$ so that

$$T(\mathbf{x}) := R\mathbf{x} + \mathbf{t} \tag{2.16}$$

where R is a rotation matrix and $\mathbf{t}$ is a translation vector.

A rotation matrix is an orthogonal matrix whose determinant is 1. Each rotation matrix R can be decomposed into three elementary rotations $R_x(\varphi)$, $R_y(\vartheta)$, and $R_z(\psi)$ as follows

$$\begin{aligned} R &= R_z(\psi) R_y(\vartheta) R_x(\varphi) \quad \text{where} \\ R_x(\varphi) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} \\ R_y(\vartheta) &= \begin{pmatrix} \cos\vartheta & 0 & \sin\vartheta \\ 0 & 1 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta \end{pmatrix} \\ R_z(\psi) &= \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{2.17}$$

The three angles φ, ϑ and ψ are called the *Euler angles*. There are 4 different representations for a rotation in 3D: Euler angles, a rotation matrix, quaternions and the angle-axis representation. For this work, we will generally stick to the matrix notation and use one of the others where it is more convenient.

2.6 Model fitting

In this section, we will describe methods of how to find a geometrical model that approximates best a given 3D data set. In general, any kind of geometrical model is possible here (e.g. boxes, spheres etc.), but we will focus only on planes. The concepts derived here will be used later in chapter 5 when we present new approaches to find planar models for 3D data sets. First, we will derive the mathematical formulation for the problem of fitting a plane into a point set, then we consider the case of a given set of lines where a best approximating plane should be found.

2.6.1 Fitting Planes into Point Sets

Suppose we are given a set of N points $\mathbf{p_1}, \ldots, \mathbf{p}_N$. The task of fitting a model – in this case a plane – into the data is defined as finding model parameters, so that the mean squared distance of the points to the model is minimized. In our case, this means that we seek a normal vector $\mathbf{n}$ and a distance value d so that the sum of squared distances of all points to the plane defined by $(\mathbf{n}, d)$ is minimal. The minimization problem can be written as

$$\text{minimize } g(\mathbf{n}, d) := \frac{1}{N}\sum_{i=1}^{N}(\mathbf{p}_i \cdot \mathbf{n} + d)^2 \quad \text{such that } \|\mathbf{n}\| = 1 \tag{2.18}$$

One necessary condition for a minimum is that the partial derivative with respect to d vanishes. This means

$$\begin{aligned} \frac{\partial g}{\partial d} &= \frac{1}{N}\sum_{i=1}^{N} 2(\mathbf{p}_i \cdot \mathbf{n} + d) \\ &= \frac{2}{N}\sum_{i=1}^{N} \mathbf{p}_i^T \mathbf{n} + 2d \overset{!}{=} 0 \end{aligned} \tag{2.19}$$

Solving for d in equation (2.19) yields

$$\begin{aligned} d &= -\frac{1}{N}\left(\sum_{i=1}^{N} \mathbf{p}_i^T\right)\mathbf{n} \\ &= -\mu^T \mathbf{n} \end{aligned} \tag{2.20}$$

where μ is the mean of the data points $\mathbf{p}_i$ (see example 2.1). Now we can plug this into our minimization function from equation (2.18) and obtain

$$\text{minimize } g(\mathbf{n}) := \frac{1}{N}\sum_{i=1}^{N}(\mathbf{p}_i^T\mathbf{n} - \mu^T\mathbf{n})^2 \quad \text{such that } ||\mathbf{n}|| = 1 \tag{2.21}$$

Next, we perform some rearrangements of the distance function g

$$\begin{aligned} g(\mathbf{n}) &= \frac{1}{N}\sum_{i=1}^{N}((\mathbf{p}_i^T - \mu^T)\mathbf{n})^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}((\mathbf{p}_i - \mu)^T\mathbf{n})((\mathbf{p}_i - \mu)^T\mathbf{n}) \\ &= \frac{1}{N}\sum_{i=1}^{N}(\mathbf{n}^T(\mathbf{p}_i - \mu)(\mathbf{p}_i - \mu)^T\mathbf{n}) \\ &= \mathbf{n}^T\left(\frac{1}{N}\sum_{i=1}^{N}(\mathbf{p}_i - \mu)(\mathbf{p}_i - \mu)^T\right)\mathbf{n} \\ &= \mathbf{n}^T C\mathbf{n} \end{aligned} \tag{2.22}$$

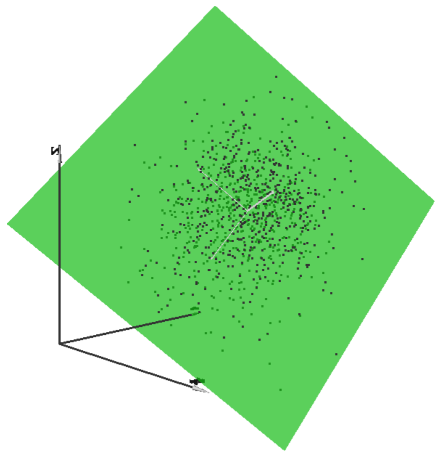
where C is the covariance matrix of the data points $\mathbf{p}_i$ defined in equation (2.7). For the minimization from equation (2.21) we need to incorporate the constraint that the normal vector $\mathbf{n}$ has unit length. This can be done by minimizing $g(\frac{\mathbf{n}}{||\mathbf{n}||})$ instead of $g(\mathbf{n})$.

$$\begin{aligned} g\left(\frac{\mathbf{n}}{||\mathbf{n}||}\right) &= \frac{1}{||\mathbf{n}||}\mathbf{n}^T C\frac{1}{||\mathbf{n}||}\mathbf{n} \\ &= \frac{1}{||\mathbf{n}||^2}\mathbf{n}^T C\mathbf{n} \\ &= \frac{\mathbf{n}^T C\mathbf{n}}{\mathbf{n}^T\mathbf{n}} \end{aligned} \tag{2.23}$$

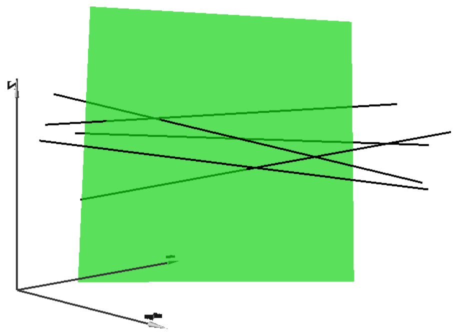
This last term is called a *Rayleigh quotient* and it is minimized by the eigenvector of C that corresponds to the smallest eigenvalue. As we have seen in example 2.1, the covariance matrix is symmetric and positive-definite, which means that its eigenvalues are all real and positive. In other words, C can be factorized as $C = UDU^T$ where U is an orthogonal matrix and D is a diagonal matrix containing the (positive) eigenvalues of C. The solution to our minimization problem is given by the column vector of U that corresponds to the smallest entry in D.

To summarize, we can say that the plane that fits best into a set of data points $\mathbf{p}_1, \ldots, \mathbf{p}_N$ in the sense of minimizing the mean squared distance must:

- contain the mean μ of the data points, because from equation (2.19) it follows that $\mu^T\mathbf{n} + d = 0$.



a. *The plane that minimizes the mean squared distance to all points contains the mean of the point cloud and is perpendicular to the eigenvector of the point cloud's covariance matrix corresponding to the smallest eigenvalue*



b. *The plane that best fits into a given set of lines has a normal vector that is mostly orthogonal to the direction vectors of all lines. Furthermore, it contains the point that minimizes the mean squared distance to all lines.*

Figure 2.3: *Fitting a plane into a set of points (a) and a set of lines (b). Note that the planes are rendered with transparency to better visualize points and lines behind the planes.*

- have a normal vector $\mathbf{n}$ that is equal to the eigenvalue of the covariance matrix C corresponding to the smallest eigenvalue.

As an example, consider figure 2.3a. The plane that fits best into the point set is drawn as a green square. The three eigenvectors of the point sets' covariance matrix are depicted as white arrows originating at the center of mass (mean) of the point set. The one that corresponds to the smallest eigen value defines the normal vector of the plane.

2.6.2 Fitting Planes into Line Sets

In some situations it may be the case that instead of a point set we are given a set of lines in 3D and we want to find a plane that best fits into these lines. The problem arising here is that a formulation based on the mean squared error like in the previous section is not appropriate, because the distance of a line l to a plane l is only defined if l and p are parallel, and we want to find a fitting plane also in the case that not all lines are parallel. Therefore, we proceed as follows. First, we seek a normal vector for the plane that is ”as orthogonal as possible” to the direction vectors of all lines. Then, we find a point with a minimal mean squared distance to all lines. This point should be contained in the plane.

Normal vector computation For a given set of N lines $l_1, \ldots, l_N$ with $l_i = (\mathbf{p}_i, \mathbf{v}_i)$ we want to find a plane normal vector $\mathbf{n}$ that is mostly orthogonal to all direction vectors

$\mathbf{v}_i$. Here, we assume that $N > 1$ and not all lines are collinear (i.e. identical), because otherwise there is no unique plane. Thus, our goal is to find $\mathbf{n}$ such that

$$\mathbf{v}_i \cdot \mathbf{n} = 0 \quad \forall i = 1, \ldots, N \tag{2.24}$$

If we stack all direction vectors $\mathbf{v}_i$ as row vectors into a matrix M, then we can reformulate equation (2.24) as

$$M\mathbf{n} = \begin{pmatrix} v_1^x & v_1^y & v_1^z \\ v_2^x & v_2^y & v_2^z \\ \vdots & \vdots & \vdots \\ v_N^x & v_N^y & v_N^z \end{pmatrix} \mathbf{n} \overset{!}{=} \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{2.25}$$

In general, this problem can not be solved if M has full rank. However, we can compute a solution that minimizes the residual error $r = ||M\mathbf{n}||$. This is done by first computing the *singular value decomposition* (SVD) of M, which yields two orthogonal matrices U and V and a diagonal matrix D such that $M = UDV^T$. The dimensions of these matrices are $N \times 3$ for U and 3×3 for D and V. Now we obtain the following minimization problem

$$\text{minimize } ||UDV^T\mathbf{n}|| \quad \text{such that } ||\mathbf{n}|| = 1 \tag{2.26}$$

Exploiting the fact that a multiplication with an orthogonal matrix does not change the norm of a vector we can reformulate equation (2.26) as follows

$$\text{minimize } ||D\mathbf{y}|| \quad \text{such that } ||\mathbf{y}|| = 1 \tag{2.27}$$

where $\mathbf{y} = V^T\mathbf{n}$. Assuming that the (positive) diagonal entries of D are ordered decreasingly, the unit length vector that minimizes (2.27) is given by $\mathbf{y} = (0, 0, 1)^T$. This means that the normal vector minimizing (2.26) is given by

$$\mathbf{n} = V \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.28}$$

In other words, the normal vector we seek for is the column vector of V that corresponds to the smallest singular value of M. This is equivalent to the eigenvector of M^TM that corresponds to the smallest eigenvalue.

Point computation As mentioned above, we seek a point $\mathbf{q}$ that minimizes the mean squared distance to all lines. Using equation (2.11) this means

$$\begin{aligned} g(\mathbf{q}) &:= \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{q} - \mathbf{p}_i||^2 - ||\mathbf{v}_i \cdot (\mathbf{q} - \mathbf{p}_i)||^2 \\ &= \frac{1}{N}\sum_{i=1}^{N} (\mathbf{q} - \mathbf{p}_i)^T(\mathbf{q} - \mathbf{p}_i) - (\mathbf{q} - \mathbf{p}_i)^T\mathbf{v}_i\mathbf{v}_i^T(\mathbf{q} - \mathbf{p}_i) \\ &= \frac{1}{N}\sum_{i=1}^{N} (\mathbf{q}^T\mathbf{q} - 2\mathbf{q}^T\mathbf{p}_i + \mathbf{p}_i^T\mathbf{p}_i) - \frac{1}{N}\sum_{i=1}^{N} (\mathbf{q} - \mathbf{p}_i)^T\mathbf{v}_i\mathbf{v}_i^T(\mathbf{q} - \mathbf{p}_i) \end{aligned} \tag{2.29}$$

By deriving this with respect to $\mathbf{q}$ and setting the result to 0 we obtain

$$\frac{\partial g}{\partial \mathbf{q}} = \frac{1}{N}\sum_{i=1}^{N}(2\mathbf{q} - 2\mathbf{p}_i) - \frac{1}{N}\sum_{i=1}^{N} 2\mathbf{v}\mathbf{v}_i^T(\mathbf{q} - \mathbf{p}_i) \overset{!}{=} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.30}$$

Now we rearrange equation (2.30) as follows

$$\begin{aligned}
\frac{2}{N}\sum_{i=1}^{N}(\mathbf{q} - \mathbf{p}_i) &= \frac{2}{N}\sum_{i=1}^{N} \mathbf{v}\mathbf{v}_i^T(\mathbf{q} - \mathbf{p}_i) \\
\sum_{i=1}^{N}\mathbf{q} - \sum_{i=1}^{N}\mathbf{p}_i &= \sum_{i=1}^{N}\mathbf{v}_i\mathbf{v}_i^T\mathbf{q} - \sum_{i=1}^{N}\mathbf{v}_i\mathbf{v}_i^T\mathbf{p}_i \\
\sum_{i=1}^{N}\mathbf{q} - \sum_{i=1}^{N}\mathbf{v}_i\mathbf{v}_i^T\mathbf{q} &= \sum_{i=1}^{N}\mathbf{p}_i - \sum_{i=1}^{N}\mathbf{v}_i\mathbf{v}_i^T\mathbf{p}_i \\
\left(\sum_{i=1}^{N}(I - \mathbf{v}_i\mathbf{v}_i^T)\right)\mathbf{q} &= \sum_{i=1}^{N}(I - \mathbf{v}_i\mathbf{v}_i^T)\mathbf{p}_i \\
\mathbf{q} &= \left(\sum_{i=1}^{N}(I - \mathbf{v}_i\mathbf{v}_i^T)\right)^{-1}\sum_{i=1}^{N}(I - \mathbf{v}_i\mathbf{v}_i^T)\mathbf{p}_i \qquad (2.31)
\end{aligned}$$

Roughly speaking, we can say that $\mathbf{q}$ is the weighted average of the points $\mathbf{p}_i$ where the weights are defined by matrices $M_i = I - \mathbf{v}_i\mathbf{v}_i^T$:

$$\mathbf{q} = \left(\sum_{i=1}^{N} M_i\right)^{-1}\sum_{i=1}^{N} M_i\mathbf{p}_i \tag{2.32}$$

Not to be absolutely certain is, I think, one of the essential things in rationality.

Bertrand Russell (1872 - 1970)

3

Probabilistic Reasoning with Graphical Models

3.1 Basics from Probability Theory

3.1.1 Notions and Notations

- A sample space $\mathcal{S}$ is a set of outcomes of an experiment.
- A *random variable* is a function that assigns a real number to a subset of $\mathcal{S}$. Random variables will be denoted with a capital letter $A, B, \ldots$
- A value of a random variable will be denoted with a small letter $a, b, \ldots$
- For a given finite set of random variables $\mathcal{A} = \{A_1, A_2, \ldots\}$ where A_i takes the value a_i and $i = 1, 2, \ldots$ we say that the set of values $\mathbf{a} := (a_1, a_2, \ldots)$ is a *configuration*.
- A probability distribution $p(.)$ assigns a positive real number to each configuration of a set of random variables $\mathcal{A}$ so that $\int_S p(\mathcal{A}) = 1$.
- The conditional probability $p(A \mid B)$ is defined as $\frac{p(A,B)}{p(B)}$.

Using these notations we can formulate the definition of conditional independency. This will be an important notion in the remainder of this chapter and later in this work.

Definition 3.1. Let $\mathcal{U} = \{A, B, \ldots\}$ be a finite set of discrete random variables and p be a probability distribution over the variables in $\mathcal{U}$. Then two arbitrary subsets $\mathcal{X} \subset \mathcal{U}$ and $\mathcal{Y} \subset \mathcal{U}$ are said to be *conditionally independent* given a third subset $\mathcal{Z} \subset \mathcal{U}$ if

$$p(\mathbf{y}, \mathbf{z}) > 0 \Rightarrow p(\mathbf{x} \mid \mathbf{y}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z}) \quad \forall \mathbf{x}, \mathbf{y}, \mathbf{z} \tag{3.1}$$

where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are configurations of $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ respectively. The conditional independence of $\mathcal{X}$ and $\mathcal{Y}$ given $\mathcal{Z}$ under the distribution p will be denoted as $I(\mathcal{X}, \mathcal{Y}, \mathcal{Z})_p$.

To illustrate this definition, we will give a short

Example 3.1. *Consider a mobile robot traveling around in an indoor environment. Assume the robot is equipped with two different sensors of different reliability for detecting obstacles. We can model this situation using the three different binary random variables O, S_1 and S_2 with possible boolean values o, s_1 and s_2.*

$$p(s_1 \mid o, s_2) = p(s_1 \mid o) \quad \forall s_1, s_2, o \in \{0, 1\} \tag{3.2}$$

The conditional independence relationship I has four important properties:

- Symmetry: $I(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) \iff I(\mathcal{Y}, \mathcal{X}, \mathcal{Z})$
- Decomposition : $I(\mathcal{X}, \mathcal{Y}, \mathcal{Z} \cup \mathcal{W}) \Longrightarrow I(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) \wedge I(\mathcal{X}, \mathcal{Y}, \mathcal{W})$
- Weak union: $I(\mathcal{X}, \mathcal{Y}, \mathcal{Z} \cup \mathcal{W}) \Longrightarrow I(\mathcal{X}, \mathcal{Y} \cup \mathcal{W}, \mathcal{Z})$
- Contraction: $I(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) \wedge I(\mathcal{X}, \mathcal{Y} \cup \mathcal{Z}, \mathcal{W}) \Longrightarrow I(\mathcal{X}, \mathcal{Y}, \mathcal{Z} \cup \mathcal{W})$

3.2 Graphical Representations

When dealing with probability distributions of many random variables $X_1, \ldots, X_N$, the question arises of how to represent this distribution. The simplest but most inefficient way is by using a *probability table*. In such a table, the probability $p(x_1, \ldots, x_N)$ for each single joint event event is stored. In the simplest case, where all variables are binary, this requires a table of size 2^N, which is intractable when the number of variables grows large. A way to overcome this problem is to exploit the conditional independencies of some of the random variables.

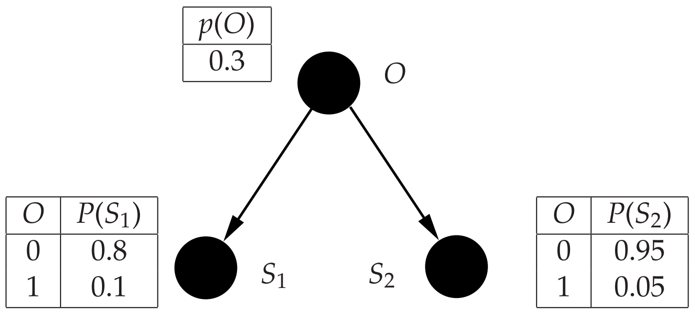
Example 3.2. *Let us again consider the obstacle detecting robot from example 3.1. Assume that sensor* 1 *detects an obstacle with probability* 0.8 *and sensor* 2 *with probability* 0.95*. Both sensors sometimes give false positive measurements, where s_1 does this with probability* 0.1 *and sensor* 2 *with probability* 0.05*. The probability of an obstacle is assumed to be* 0.3*. Thus we have:*

$$\begin{array}{lll} p(s_1 \mid o) = 0.8 & p(s_2 \mid o) = 0.95 & p(o) = 0.3 \\ p(s_1 \mid \neg o) = 0.1 & p(s_2 \mid \neg o) = 0.05 & \end{array}$$

With these values we can compute the joint distribution $p(o, s_1, s_2)$ as follows

$$\begin{aligned} p(s_1, s_2, o) &= p(s_1 \mid s_2, o) p(s_2, o) \\ &= p(s_1 \mid o) p(s_2 \mid o) p(o) \end{aligned} \tag{3.3}$$

Using this formula we can compute all 8 *probabilities and store them in a probability table:*



$p(O)$
0.3

O	$P(S_1)$
0	0.8
1	0.1

O	$P(S_2)$
0	0.95
1	0.05

Figure 3.1: *Conditional independency graph for example 3.2*

O	S_1	S_2	$p(o, s_1, s_2)$
0	0	0	0.5985
0	0	1	0.0315
0	1	0	0.0665
0	1	1	0.0035
1	0	0	0.003
1	0	1	0.057
1	1	0	0.012
1	1	1	0.228

The example shows that the representation as a probability table is equivalent to the one using only the conditional probabilities from equation (3.3), but it requires a lot more memory. Of course this holds only if there are variables that are conditionally independent, because only then we can do the decomposition in equation (3.3). Thus, instead of storing the entire probability table, we only need to store the *local* tables for each random variable. This is illustrated in Figure 3.1.

In the following, we will briefly summarize the concepts of directed and undirected graphs and how they can be used to represent probability distributions. The notations are taken from Pearl [1988] and we also refer there for any more detailed analysis.

3.3 Directed Graphs

Definition 3.2. A *directed acyclic graph* (DAG) D is defined as a pair $D = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges with

$$(v_1, v_2) \in \mathcal{E} \Rightarrow (v_2, v_1) \notin \mathcal{E} \;\; \forall v_1, v_2 \in \mathcal{V}$$

For an edge $e = (v_1, v_2)$, we say that it is *outgoing* for node v_1 and *ingoing* for the node v_2. Furthermore we define a *path* between nodes v_1 and v_2 as a set of nodes $w_1, \ldots, w_n$ where $w_1 = v_1$, $w_n = v_2$ and for w_i, w_{i+1} there is either an edge (w_i, w_{i+1}) or (w_{i+1}, w_i).

Definition 3.3 (D-Separation)**.** Suppose $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ are subsets of the nodes $\mathcal{V}$ of a DAG D. Then we say that $\mathcal{Z}$ *d-separates* $\mathcal{X}$ from $\mathcal{Y}$ if along every path between a node in $\mathcal{X}$ and a node in $\mathcal{Y}$ there is a node w so that one of the following conditions holds:

1. w has ingoing edges and neither w nor any of its descendants is in $\mathcal{Z}$.
2. w has no ingoing edges and is in $\mathcal{Z}$

We will denote the fact that $\mathcal{Z}$ d-separates $\mathcal{X}$ from $\mathcal{Y}$ as $\langle \mathcal{X} \mid \mathcal{Y} \mid \mathcal{Z} \rangle_d$.

3.3.1 Bayesian Networks

Definition 3.4. A DAG D is an *I-map* of a dependency model p if for all triples of disjoint sets of vertices $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ it holds:

$$\langle \mathcal{X} \mid \mathcal{Y} \mid \mathcal{Z} \rangle_d \Rightarrow I(\mathcal{X}, \mathcal{Y}, \mathcal{Z})_p \tag{3.4}$$

Furthermore, an I-map is *minimal* if by deleting any edge from D condition (3.4) is violated.

Definition 3.5. Let $\mathcal{U}$ be a set of random variables and P be a probability distribution on $\mathcal{U}$. Then a DAG D is called a *Bayesian network* iff D is a minimal I-map of p.

Example 3.3. *Using the three random variables from Example 3.2 as nodes we can draw a directed graph D like the one in Figure 3.1. We can clearly see that the second condition from definition 3.3 is valid for the node O, which means that O d-separates S_1 and S_2. As we have seen already, for the corresponding random variables O, S_1 and S_2 it holds $I(S_1, O, S_2)_p$. This means, D is an I-map of p. In addition, it is a minimal I-map, because if we remove one of the arrows, we would have an isolated node in the graph.*

3.4 Undirected Graphs

Similar to the definitions in the previous section, we provide the following

Definition 3.6. An *undirected graph* G is a pair $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, where, in contrast to a DAG, the edge relation is symmetric, i.e.

$$(v_i, v_j) \in \mathcal{E} \Longleftrightarrow (v_j, v_i) \in \mathcal{E}.$$

A *clique* C of a graph G is a subset of the nodes $\mathcal{V}$ so that

$$(v_i, v_j) \in \mathcal{E} \quad \forall v_i, v_j \in C$$

Definition 3.7 (U-Separation)**.** Suppose $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ are subsets of the nodes $\mathcal{V}$ of a DAG D. Then we say that $\mathcal{Z}$ *d-separates* $\mathcal{X}$ from $\mathcal{Y}$ if along every path between a node in $\mathcal{X}$ and a node in $\mathcal{Y}$ there is a node w so that one of the following condition holds:

1. w has ingoing edges and neither w nor any of its descendants is in $\mathcal{Z}$.
2. w has no ingoing edges and is in $\mathcal{Z}$

3.4.1 Markov Networks

Definition 3.8. An undirected graph G is an *I-map* of a dependency model M if for all triples of disjoint sets of vertices $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ it holds:

$$\langle \mathcal{X} \mid \mathcal{Y} \mid \mathcal{Z} \rangle_d \Rightarrow I(\mathcal{X}, \mathcal{Y}, \mathcal{Z})_M \tag{3.5}$$

Furthermore, an I-map is *minimal* if by deleting any edge from D condition (3.5) is violated.

Definition 3.9. Let $\mathcal{U}$ be a set of random variables and P be a probability distribution on $\mathcal{U}$. Then an undirected graph G is called a *Markov network* iff G is a minimal I-map of P.

Theorem 1 (Hammersley and Clifford)**.** *Let $G = (\mathcal{V}, \mathcal{E})$ be an arbitrary undirected graph where each node $V \in \mathcal{V}$ corresponds to a discrete random variable X. Assume we are given a* potential function ϕ *that assigns a positive value to any given subset of random variables. If we denote the set of all cliques of G with C and the subset of random variables that correspond to a clique c with $\mathcal{X}_c$, then the probability function P formed as*

$$P(\mathcal{X}) = \frac{\prod_{c \in C} \phi(\mathcal{X}_c)}{\sum_{\mathcal{X}'} \prod_{c \in C} \phi(\mathcal{X}'_c)} \tag{3.6}$$

is a Markov network corresponding to G. In other words, G is a minimal I-map of P.

For the proof of this theorem we refer to Clifford [1990] and Besag [1974].

3.4.2 Conditional Markov Random Fields

The notion of conditional Markov random fields was first introduced by Lafferty et al. [2001]. The idea is the following: when using probabilistic modeling to classify objects – represented by feature vectors $\mathbf{x}$ – into a finite set of classes $y_1, \ldots, y_N$, we are mainly interested in the *conditional* probability $p(y|\mathbf{x})$. As we will see in chapter 9, this conditional probability can be used to do *inference* in a classification task. This means, after learning the parameters of the distribution $p(y \mid \mathbf{x})$ we can infer a label to a new given test example $\hat{\mathbf{x}}$ by finding the class label y that maximizes $p(y \mid \hat{\mathbf{x}})$. In analogy to the Markov Networks presented above, we can express the conditional probability as

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{\prod_{c \in C} \phi(\mathbf{x}_c, \mathbf{y}_c)}{\sum_{\mathbf{y}'} \prod_{c \in C} \phi(\mathbf{x}'_c, \mathbf{y}_c)}. \tag{3.7}$$

Here, we denote the set of all class labels with $\mathbf{y}$, and the features and labels of a given clique c with $\mathbf{x}_c$ and $\mathbf{y}_c$.

At this point, we will not discuss the mathematical details of conditional MRFs any further and refer to the literature instead (for example Wallach [2002], Dietterich [2002], but also Lafferty et al. [2001]). We only note that the potential function is now defined on the features *and* the labels and that it is usually used to model the compatibility of a given labeling to the features. Later in chapter 9, we will see how this is done in practice.

Part II

3D Map Building

You don't understand anything until you learn it more than one way.

Marvin Minsky (* 1927)

4

Representations of 3D Range Data

4.1 Introduction

This chapter gives an overview of possible approaches to represent 3D range data. We will discuss advantages and drawbacks of the different data structures presented here. These considerations will be important for developing new 3D data representations in the following chapters. The chapter is organized as follows: First we present the concept of *point clouds*, which is the most common way to store 3D data. In fact, all other data structures mentioned here will be constructed from point clouds, because they store the data as it comes from the 3D sensor, independent on the sensor (stereo camera, 3D laser etc.). In section 4.3 we present tree structures as a more efficient way of storing 3D data, especially for applications in which a search needs to be performed. We will focus on *kD*-trees and OBB-trees, because they will be used later in this work. Section 4.4 introduces 3D *occupancy grids*, which store the information of an occupied or unoccupied region by using binary random variables in a grid structure. This gives a powerful representation and is useful for computing the *expected information gain* for exploration applications. This will be shown in an example. Furthermore, we will address data structures that provide a polygonal representation of 3D data. In particular, we will show in section 4.5 how point clouds can be enriched by connecting points to triangles. This is especially useful for visualization purposes. We will compare three different triangulation strategies. Finally, section 4.6 concludes the chapter with a comparison and a discussion on the different approaches to represent 3D data.

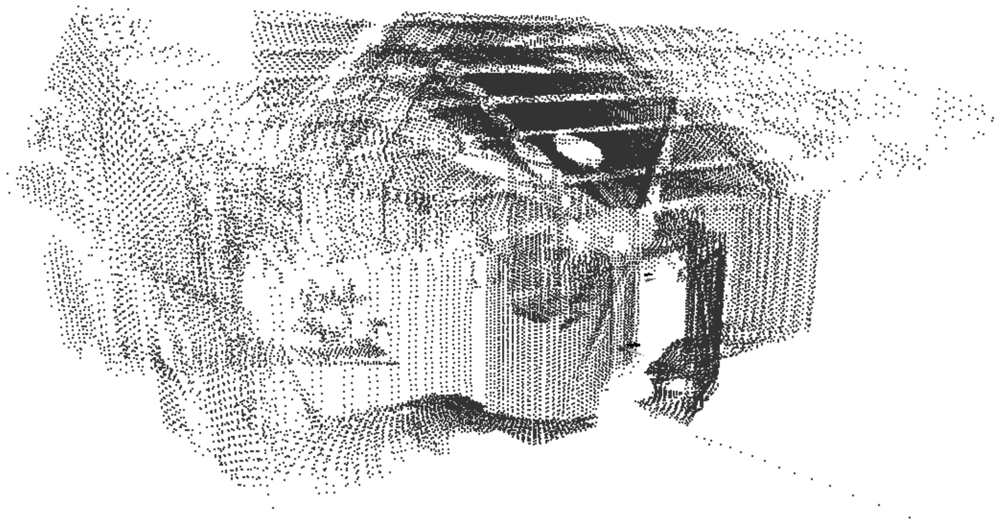


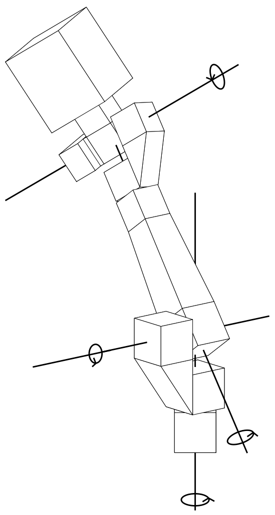
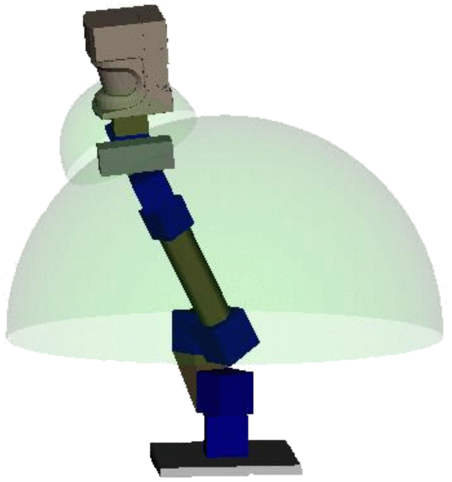
Figure 4.1: *Point cloud representation of a barn. The data consists of* 45013 *points and was acquired using two rotating laser range finders.*

4.2 Point Clouds

A point cloud C is the simplest way of representing 3D range data. It consists of a set of N 3D vectors $\mathbf{p}_1, \ldots, \mathbf{p}_N$ that correspond to the coordinates of each point in C. Figure 4.2 shows an example of a point cloud. The data set was acquired using two laser range finders rotating about the vertical z-axis. The representation as a point cloud is independent of the sensor used for data acquisition. For example, when using a stereo camera rig a point cloud can be obtained by finding corresponding pixels in both camera images and re-projecting these pixels into the 3D space. The advantage of point clouds obtained from stereo rigs is that the color information of the points is provided directly by the camera images. However, usual point clouds obtained from stereo cameras are less dense than point clouds obtained with laser range finders. In the following, we will show how point clouds can be computed from laser range data.

4.2.1 Computation of Point Clouds from Laser Range Data

Depending on the sensor used for acquiring 3D data, the formulation for computing point clouds differs. As an example, we will show in the following the computation of point coordinates from range data that are acquired by a 2D laser range scanner mounted on a 4 DOF manipulator (see Figure 4.2c). The four joints of the manipulator are rotational and will be denoted j_1 through j_4 (from bottom to top). Whereas the joints j_1 and j_3 rotate about the vertical Z-axis from $-\pi$ to π, j_2 and j_4 rotate about the horizontal X-axes from

a. *Photograph of the manipulator.* **b.** *Simplified model showing the rotation axes.* **c.** *The visualized configuration space consisting of a big half sphere $S^{\downarrow}$ and a small half sphere $S^{\uparrow}$.*

Figure 4.2: 4 *DOF Manipulator with a laser range finder on top. All* 4 *joints are rotational and consist of two blue cubes respectively (see Fig (a)). Fig. (b) shows the* 4 *different rotation axes of the manipulator. The configuration space can be visualized as two half spheres (see Fig (c)).*

$-\frac{\pi}{2}$ to $\frac{\pi}{2}$. Thus, the configuration space C of the manipulator is given as:

$$C = \left\{ \mathbf{j} \in \mathbb{R}^4 \,\middle|\, \begin{array}{c} -\pi \leq j_1, j_3 \leq \pi, \\ -\frac{\pi}{2} \leq j_2, j_4 \leq \frac{\pi}{2} \end{array} \right\} \tag{4.1}$$

Due to the length of the connections between the joints, the individual configurations in C geometrically correspond to points on small half spheres $S^{\uparrow}$ whose centers lie on the surface of a bigger half sphere $S^{\downarrow}$ (see Figure 4.2).

If we assign a local coordinate frame to each of the joints and one to the laser we can define 5 different transformations $T_1, \ldots, T_5$ so that $T_i(\mathbf{x})$ returns the coordinates of the point $\mathbf{x}$ in the coordinate frame of joint $i - 1$. Here we assume that T_1 converts to coordinates in the global reference frame. For a given point $\mathbf{p}_l$ in the local reference frame of the laser, we can then compute its global coordinates $\mathbf{p}_g$ as

$$\mathbf{p}_g = T_1(T_2(T_3(T_4(T_5(\mathbf{p}_l))))) \tag{4.2}$$

Each of the transformations $T_1, \ldots, T_4$ depends on the position of the joints $j_1, \ldots, j_4$ respectively while T_5 consists of a fixed rotation and translation between the laser reference

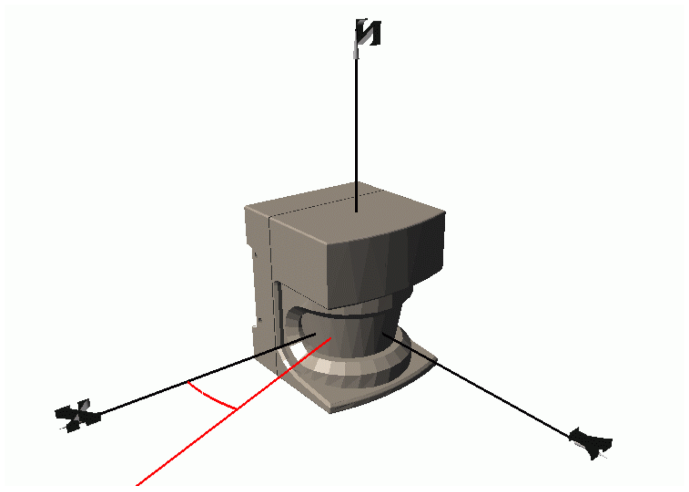


Figure 4.3: *Local reference frame of the laser. The origin is defined as the center of the rotating mirror inside the device. The azimuth angles α of the laser beams range from $0°$ to $180°$. As an example, a laser beam at $\alpha = 20°$ is depicted.*

frame and the reference frame that corresponds to j_4. Thus, we have

$$\begin{aligned} T_1(\mathbf{x}) &:= R_z(j_1)\mathbf{x} + \mathbf{t}_1 \\ T_2(\mathbf{x}) &:= R_x(j_2)\mathbf{x} + \mathbf{t}_2 \\ T_3(\mathbf{x}) &:= R_z(j_3)\mathbf{x} + \mathbf{t}_3 \\ T_4(\mathbf{x}) &:= R_x(j_4)\mathbf{x} + \mathbf{t}_4 \\ T_5(\mathbf{x}) &:= R\mathbf{x} + \mathbf{t}_5 \end{aligned} \tag{4.3}$$

Here, we used the notation introduced in equation (2.17) for the rotations. The point coordinates $\mathbf{p}_l$ in the local reference frame of the laser are computed as

$$\mathbf{p}_l = \begin{pmatrix} d \cdot \cos(\alpha) \\ d \cdot \sin(\alpha) \\ 0 \end{pmatrix} \tag{4.4}$$

where d is the distance measured by the laser and α is the corresponding beam angle (see Fig. 4.3 for an explanation). In the case of the manipulator shown in Fig. 4.2, we obtain the overall transform $T^\star$ as

$$\begin{aligned} T^\star(\mathbf{p}_l) &= R^\star \mathbf{p}_l + \mathbf{t}^\star \quad \text{where} \\ R^\star &= \begin{pmatrix} \cos j_1 \cos j_3 - \sin j_1 \cos j_2 \sin j_3 & -\cos j_1 \sin j_3 \cos j_4 - \sin j_1 \cos j_2 \cos j_3 \cos j_4 + \sin j_1 \sin j_2 \sin j_4 & -\cos j_1 \sin j_3 \sin j_4 - \sin j_1 \cos j_2 \cos j_3 \sin j_4 - \sin j_1 \sin j_2 \cos j_4 \\ \sin j_1 \cos j_3 + \cos j_1 \cos j_2 \sin j_3 & -\sin j_1 \sin j_3 \cos j_4 + \cos j_1 \cos j_2 \cos j_3 \cos j_4 - \cos j_1 \sin j_2 \sin j_4 & -\sin j_1 \sin j_3 \sin j_4 + \cos j_1 \cos j_2 \cos j_3 \sin j_4 + \cos j_1 \sin j_2 \cos j_4 \\ -\sin j_2 \sin j_3 & -\sin j_2 \cos j_3 \cos j_4 - \cos j_2 \sin j_4 & -\sin j_2 \cos j_3 \sin j_4 + \cos j_2 \cos j_4 \end{pmatrix} \\ \mathbf{t}^\star &= -R^\star \begin{pmatrix} c_1 \\ c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} -\sin(j_1)\cdot\sin(j_2)\cdot c_3 + c_1 \\ \cos(j_1)\cdot sin(j_2)\cdot c_3 + c_1 \\ \cos(j_2)\cdot c_3 + c_2 - c_3 \end{pmatrix} \end{aligned}$$

where we use the three constants $c_1 = 0.045m$, $c_2 = 0.73m$ and $c_3 = 0.47m$, which were taken from the specifications of the manipulator.

4.3 Tree Structures

One problem when dealing with point clouds is the fact that special operations such as searching for the nearest neighbor to a given query point can not be performed efficiently. Especially when the point clouds are big ($> 1,000,000$ points), a linear search in the list of all points is very inefficient.

4.3.1 3D-trees

A general data structure that allows efficient search algorithms is the *kD-tree* (see Bentley [1975]). Here, k stands for the number of dimensions in which the input data points are given, in our case $k = 3$. As one can imagine, a kD-tree stores the input data in a tree-like structure. Each node in this tree is either a leaf node without successors or has exactly two successor nodes or *sub-trees*, which we call the *left* and the *right* sub-tree. The data are stored only in the leaf nodes, where each leaf node can contain many data points. A parameter, usually called the *bucket size*, determines the maximum number of data points in a leaf node. If the bucket size is exceeded for a given leaf node, this node needs to be split into two sub-trees, and therefore turns into an inner node. The way this splitting is done is handled by the *splitting rule*, another parameter of the kD-tree.

Many different splitting rules are possible, but they all separate the data using axis-aligned hyperplanes (in our case planes) into two subsets. Depending on the position of the hyperplanes, the resulting subsets are more or less balanced in size, which makes the resulting tree more or less balanced. Of course, searching in a balanced tree is usually more efficient because balanced trees tend to be flatter and therefore the search paths are shorter. One simple splitting rule is the *median rule*, where the hyperplanes are determined by finding the median of the coordinates in the data at one given dimension and using this median as the shift of the hyperplane from the origin. The dimension of this median may be fixed or varying over the levels of the tree. For example, when applying this rule to a 3D-tree, the plane at the root node will be orthogonal to the X-axis and its distance from the origin will be the median of all X-values in the data. The hyperplanes at the next level are then orthogonal to the Y-axis and their position is the median of all Y-values in the subsets. This way the splitting goes on until all leaf nodes contain at most as many data points as given by the bucket size.

Efficient searching in $3D$-trees Several operations such as n-nearest neighbor search or range search can be performed efficiently using kD-trees. As an example, we only explain the computation of the nearest neighbor $\mathbf{p}$ from a point cloud C to a given query point $\mathbf{q}$. We assume that a Minkowski metric $d(\mathbf{p}, \mathbf{q})$ is defined on the data points as follows:

$$d(\mathbf{p}, \mathbf{q}) := \sqrt[r]{\sum_{i=1}^{k} (p_i - q_i)^r}; \quad \mathbf{p} = (p_1, \ldots, p_k), \quad \mathbf{q} = (q_1, \ldots, q_k), \quad r \in \mathbb{R}. \tag{4.5}$$

Algorithm 1 *searchNN*$(\tau, \mathbf{q}, (\mathbf{p}, d))$ – Nearest neighbor search using a *kD*-tree τ

```
Definitions:
      $\tau_l$ : left subtree of $\tau$
      $\tau_r$ : right subtree of $\tau$
      $H_\tau$ : hyperplane at subtree $\tau$
 1: if $isLeaf(\tau)$ then
 2:    $(\mathbf{p}, d) \leftarrow searchBucket(\tau, \mathbf{q}, (\mathbf{p}, d))$
 3:    return $(\mathbf{p}, d)$
 4: end if
 5: if $isLeftOf(\mathbf{q}, H_\tau)$ then
 6:    $(\mathbf{p}, d) \leftarrow searchNN(\tau_l, \mathbf{q}, (\mathbf{p}, d))$
 7:    if $intersects(H_\tau, sphere(\mathbf{q}, d))$ then
 8:       $(\mathbf{p}, d) \leftarrow searchNN(\tau_r, \mathbf{q}, (\mathbf{p}, d))$
 9:    end if
10: else
11:    $(\mathbf{p}, d) \leftarrow searchNN(\tau_r, \mathbf{q}, (\mathbf{p}, d))$
12:    if $intersects(H_\tau, sphere(\mathbf{q}, d))$ then
13:       $(\mathbf{p}, d) \leftarrow searchNN(\tau_l, \mathbf{q}, (\mathbf{p}, d))$
14:    end if
15: end if
16: return $(\mathbf{p}, d)$
```

Assuming that C is represented as a *kD*-tree τ, we can now formulate the algorithm to determine the nearest neighbor with respect to this metric to the query point $\mathbf{q}$. It goes back to Friedman et al. [1977] and is shown in Algorithm 1. The first call to the recursive algorithm is made using an arbitrary point $\mathbf{p}$ and a distance $d = \infty$. First, the algorithm searches for the leaf node where the query point $\mathbf{q}$ falls into. Then it determines the closest point $\mathbf{p}$ in the corresponding bucket along with its distance d to $\mathbf{q}$. Finally, the hyperplanes of all subtrees on the way to the leaf node are tested on intersection with the Sphere $S = (\mathbf{p}, r)$. If one such hyperplane intersects S, then the corresponding other subtree may contain a point $\mathbf{p\prime}$ that is closer to $\mathbf{q}$ than $\mathbf{p}$. At the end, all potential subtrees are searched and the result is the closest point $\mathbf{p}$, along with its distance d to $\mathbf{q}$.

Disadvantages of *kD*-trees One major drawback of *kD*-trees is that they are static. This means that it is not possible to insert new data points into the tree without the need to recompute the whole tree. Of course, one can think of inserting a new data point $\mathbf{p}$ by extending the leaf node n into which $\mathbf{p}$ falls. However, this strategy may result in arbitrarily unbalanced trees, because the insertion of $\mathbf{p}$ can affect the application of the splitting rule in all nodes from n up to the root node.

4.3.2 OBB-trees

The key operation that has to be carried out to avoid collisions of a mobile robot with obstacles is the test whether there is an intersection of the robot's shape with obstacles stored in the map. A fast and robust method that uses 3D polygons to represent the obstacles and the robot has been proposed by Gottschalk [2000]. In this approach the collision check is performed using *Oriented Bounded Boxes* and by organizing these in a tree-structure (*OBB-tree*). The tree is built from top to bottom for a given set of 3D polygons. Each inner node of the tree consists of a 3D oriented bounding box for a subset of the polygons. The bounding boxes are oriented along the principal directions of the polygon vertices. This way we obtain a tight fit of the bounding boxes to the polygons.

The main idea is that the overlap test for two oriented bounding boxes can be performed efficiently by projecting both boxes onto a line and checking the resulting line segments for overlap. As Gottschalk [2000] shows, only 15 different line directions need to be tested, namely the 6 principal directions of both boxes and the 9 mutual cross products of these.

For a collision check between the robot and a set of obstacle polygons, an OBB-tree is built both for the robot and for the obstacles. The collision check begins by testing the root node boxes for overlap and then proceeds in both trees until a level is reached where the boxes do not overlap or until a leaf node is encountered. In the first case, the algorithm returns 'no collision'. In the second case the corresponding 3D polygons need to be checked for intersection.

4.4 3D Occupancy Grids

The major drawback when using point clouds for representing 3D data is the fact that each point only represents the end point of the line of sight from which the scene was observed. This means that the 3D coordinates of such a point represent the first occurrence of solid matter while traveling along a linear beam. Depending on the sensor used for acquiring the data, this beam starts at different positions in space and ends at different types of material. The problem with this kind of data representation is that only the information about *occupied* space is stored, while the fact that all space along this beam is *free* can not be represented by a point cloud. This causes problems in cases where knowledge about occupied and free space is needed, for example when a collision between moving objects needs to be detected.

A data structure that takes also the free space into account is the *occupancy grid* [Elfes, 1989, Moravec, 1988]. We will briefly present the main ideas in the next section. A more detailed analysis can be found in the book of Thrun et al. [2005].

4.4.1 Mathematical Formulation

The main idea of the occupancy grid is to subdivide the space into a finite set of *grid cells*. Each of these cells reflects the fact that the space at the particular position is either occupied or free. Mathematically, this is expressed in terms of a binary random variable

m_i where i is the index of the cell in the map m. The probability that the cell m_i is occupied is then written as $p(m_i = 1)$ or simply $p(m_i)$. Now, the algorithm to build an occupancy grid map is stated as a time-dependent method where at each time step t the information z_t that a robot acquires from its environment (also called the *measurements*) is incorporated while the robot travels around this environment. We assume that we are given the exact robot position x_t at each time step, which is why this concept is called *mapping with known poses*. Mathematically, the goal in building an occupancy map is defined as finding the probability distribution

$$p(m \mid z_{1:t}, x_{1:t}) \tag{4.6}$$

where $z_{1:t}$ denotes all measurements that the robot received at the time frames $1, \ldots, t$ and $x_{1:t}$ are the respective robot positions. For the computation of (4.6) we assume that the particular grid cells are statistically independent. This is a strong assumption that is violated in most cases, but it will serve us as a convenient approximation. We refer again to Thrun et al. [2005] for further considerations on this topic. Thus, equation (4.6) turns into

$$\begin{aligned} p(m \mid z_{1:t}, x_{1:t}) &= \prod_i p(m_i \mid z_{1:t}, x_{1:t}) \\ &= \prod_i \frac{p(z_t \mid m_i, z_{1:t-1}, x_{1:t}) p(m_i \mid z_{1:t-1}, x_{1:t})}{p(z_t \mid z_{1:t-1}, x_{1:t})} \\ &= \prod_i \frac{p(z_t \mid m_i, x_{1:t}) p(m_i \mid z_{1:t-1}, x_{1:t})}{p(z_t \mid z_{1:t-1}, x_{1:t})} \end{aligned} \tag{4.7}$$

Here, we applied Bayes' rule and exploited the independence of the current measurement z_t from the previous ones $z_{1:t-1}$ given the map m_i. If we again apply Bayes' rule for the measurement model $p(z_t \mid m_i, x_{1:t})$ we obtain

$$\frac{p(m_i \mid z_t, x_{1:t}) p(z_t \mid x_{1:t}) p(m_i \mid z_{1:t-1}, x)}{p(m_i \mid x_{1:t}) p(z_t \mid x_{1:t})} \tag{4.8}$$

Now, exploiting the fact that all m_i are binary variables, we can compute the *log odds ratio* instead of $p(m_i \mid z_{1:t}, x_{1:t})$, which is defined by

$$l_{i,t} := \log \frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})} \tag{4.9}$$

The log odds ratio is more convenient and computationally more stable for probabilities close to 1 or 0. By deriving a similar expression as in Equation (4.8) for the opposite event $p(\neg m_i \mid z_{1:t}, x_{1:t}) = 1 - p(m_i \mid z_{1:t}, x_{1:t})$ and plugging both into equation (4.9), we obtain

$$\begin{aligned} l_{i,t} &= \log \frac{p(m_i \mid z_{1:t-1}, x_t)}{1 - p(m_i \mid z_{1:t-1}, x_t)} + \log \frac{p(m_i \mid z_t, x_t)}{1 - p(m_i \mid z_t, x_t)} - \log \frac{p(m_i)}{1 - p(m_i)} \\ &= l_{i,t-1} + \lambda(m_i, z_t, x_t) - l_0 \end{aligned} \tag{4.10}$$

Here, we define the function λ as the log odds ratio of the *inverse sensor model* $p(m_i | z_t, x_t)$ at time t and l_0 as the log odds ratio of the map occupancy prior $p(m_i)$. For the computation of the inverse sensor model there exist different strategies.

4.4.2 An Application Example: Using 3D Occupancy Grids for Autonomous Exploration

In this section, we want to show the representational power of 3D occupancy grids in a concrete application example. We will see that the information stored in an occupancy grid can be used to get an idea of what is known about the environment and what not. This will be expressed mathematically by means of the *negative entropy*. Using this, we can estimate the amount of new information we will acquire when observing a given scene from a different view point. This value is usually called the *information gain*. This means, we can formulate an algorithm that *explores* an unknown environment by generating a set of paths along which a robot observes a given scene. These paths can then be evaluated with respect to the expected information gain by storing the acquired information in an occupancy grid. When choosing the path which has the maximal expected information gain, we obtain a greedy strategy for the problem of autonomous exploration in an unknown environment.

The System

Consider the robotic manipulator with 4 degrees of freedom that was shown in figure 4.2. At each position of the four joints, the laser can acquire a fixed set of measurements. We will denote a joint position at a given time step t with $\mathbf{x}_t \in C$ where C is the configuration space defined in equation (4.1). The measurements taken at a position $\mathbf{x}_t$ consist of N laser beams and will be denoted as $\mathbf{z}_t \in \mathbb{R}^N$.

Due to the configuration of the manipulator, it can reach a 3D area of about $1m$ range. Objects that are farther away, can also be scanned by the system, but then it is more difficult to find different view points onto the object. In particular, it is not possible to scan objects from their back side. Therefore, we restrict our attention to a cubical area B with $1m$ edge length around the manipulator, which will be our *region of interest*. The information that is acquired during the exploration process is represented in a 3D occupancy grid inside B. In the remainder, B is also denoted the *grid box*. Now, we define the local exploration task as follows: For our given 3d rectangular region B we search for a set of sensor paths along which the acquired sensor information is maximized while the overall path cost is minimized.

The Expected Information Gain

As already mentioned, we will use the negative entropy to measure the amount of information acquired by the sensor. In the case of the binary random variable m_i where i is the index of the grid cell, the entropy H is defined as

$$H(m_i) := -p(m_i)\log_2 p(m_i) - (1 - p(m_i))\log_2(1 - p(m_i)) \qquad (4.11)$$

The entropy measures the amount of chaos or uncertainty about m_i. It reaches the maximum value 1 when $p(m_i) = 0.5$ and the minimum value 0 for $p(m_i) = 0$ or $p(m_i) = 1$. The negative entropy can be used to measure the information about m_i. In cases where we are certain that m_i is occupied ($m_i = 1$) or free ($m_i = 0$), the negative entropy $-H(m_i)$ is maximal. After updating the occupancy probability of m_i for a given new measurement $\mathbf{z}_t$ and a manipulator position $\mathbf{x}_t$ according to equation (4.10), we obtain a new negative entropy $-H(m_i \mid \mathbf{z}_t, \mathbf{x}_t)$. The difference of this new amount of information and the old one is then defined as the information gain I:

$$I(m_i \mid \mathbf{z}_t, \mathbf{x}_t) := H(m_i) - H(m_i \mid \mathbf{z}_t, \mathbf{x}_t) \tag{4.12}$$

Based on the information gain and an appropriate path cost function f, we can evaluate a possible path $\mathcal{P}$ of robot positions $(\mathbf{x}_1, \ldots, \mathbf{x}_T)$ by calculating the weighted difference between the information gain and $f(\mathcal{P})$ (see also Stachniss and Burgard [2003]). A typical problem in this context is that the information gain cannot be calculated in advance as one does not know which measurement will be received along $\mathcal{P}$. The usual solution is to compute the *expected information gain* by integrating over all possible measurements. In our case, however, this is infeasible, since the number of possible measurements grows exponentially with the number of time steps or with the length of $\mathcal{P}$. Therefore, we approximate the expected information gain by considering the most likely measurement $\bar{z}$ for each beam. This measurement is determined by traversing the grid in B along each beam until a cell with probability higher than a given constant, which is set to 0.5 in our example, is reached. The expected information gain EI of a particular path $\mathcal{P}$ is then computed as:

$$EI(\mathcal{P}) = \sum_{t=1}^{T} \sum_{i=0}^{N} \sum_{m \in \mathcal{B}(\bar{z}_{t,i})} I(m \mid \bar{z}_{t,i}, \mathbf{x}_t), \tag{4.13}$$

where T is length of the path $\mathcal{P}$, N is the number of beams of each scan, and $\mathcal{B}(\bar{z}_{t,i})$ is the set of all cells intercepted by the beam with length $\bar{z}_{t,i}$.

The Utility of a Path

For a good path evaluation we need a path cost function f that penalizes dangerous paths. A good way to achieve this is by defining f as the inverse distance to the next object. We approximate this value by creating a set of k sample points $\{\mathbf{s}_0, \ldots, \mathbf{s}_k\}$ inside the volume of the manipulator and determining the sample with minimum distance to the grid box B. Thus, we have

$$f(\mathcal{P}) = \underset{i=1,\ldots,k}{\operatorname{argmin}} \{d(\mathbf{s}_i, B)^{-1}\}. \tag{4.14}$$

The distance $d(\mathbf{s}_i, B)$ from a sample $\mathbf{s}_i$ to the box B can be efficiently calculated using a *generalized voronoi diagram* (GVD) of B, which is described in detail by Lin [1993]. A GVD assigns to each face, each edge and each vertex of B a *voronoi region* which is defined by the set of all points to which the corresponding face, edge or vertex is closer than all others. Given a GVD for B, we only need to check into which voronoi region $\mathbf{s}_i$ falls. Then, $d(\mathbf{s}_i, B)$

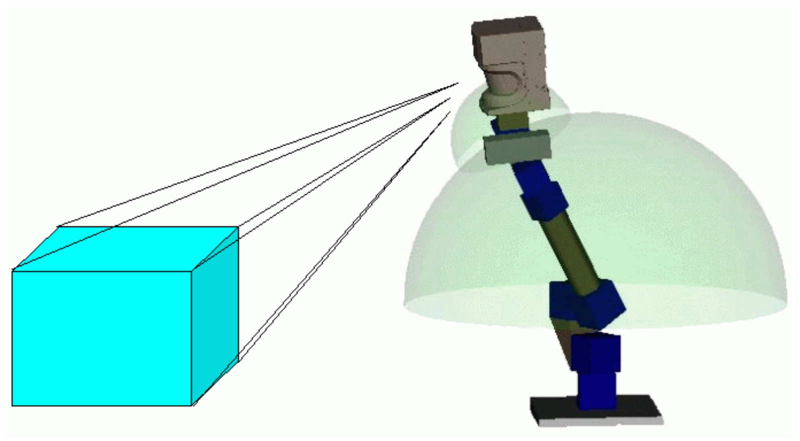


Figure 4.4: *Creation of a possible exploration path. For each edge of the box B we determine a plane that is tangent to the upper half sphere $S^{\uparrow}$. The points where these planes touch $S^{\uparrow}$ are then connected to a sub-path.*

is given as the distance to the box feature (face, edge or vertex) that corresponds to the found voronoi region.

Using equations (4.12) and (4.14), we define the best exploration path $\mathcal{P}^*$ as the one that maximizes the utility:

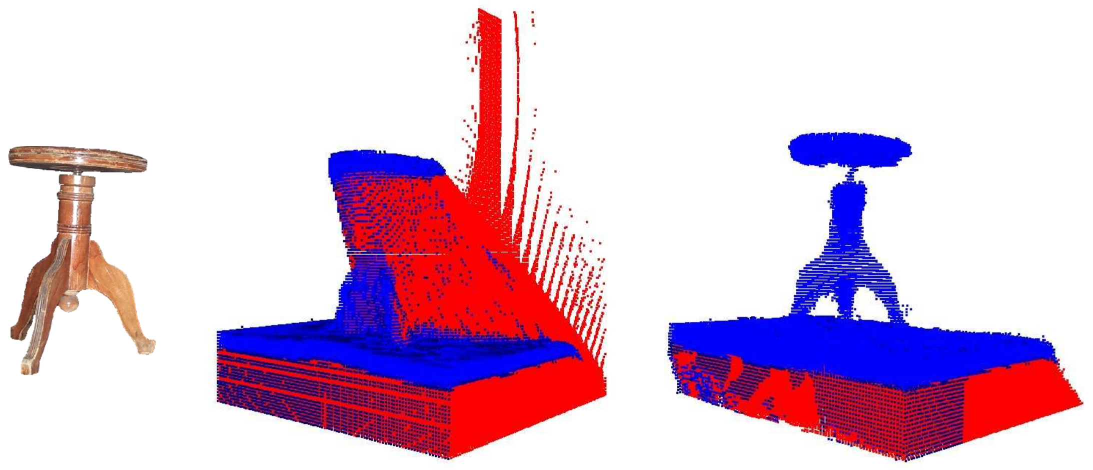
$$\mathcal{P}^* = \operatorname*{argmax}_{\mathcal{P}\in\mathfrak{P}} \{EI(P) - \lambda f(P)\} \tag{4.15}$$

where λ is a fixed weighing factor and $\mathfrak{P}$ is the set of all possible paths.

The Exploration Algorithm

The major problem in evaluating equation (4.15) is that $\mathfrak{P}$ cannot be determined efficiently. Therefore, we consider only a small subset $\mathfrak{S}$ of $\mathfrak{P}$, which contains all paths that are reachable by the manipulator and at the same time include good view points. These paths should provide us a good coverage of the object(s) inside B, which is accomplished when the laser sweeps across the edges $\{e_1, \dots, e_{12}\}$ of B. The start and end points of such paths can be determined by finding manipulator positions at which one of the edges e_i is inside the laser plane. Geometrically, this corresponds to points at which a plane passing through a given e_i is tangent to a given small sphere $S^{\uparrow}$. Thus, the set $\mathfrak{S}$ of good view paths is constructed as follows (see also Figure 4.4):

First, we create a set of points on the surface of the big half sphere $S^{\downarrow}$ so that no point is nearer to B than the maximum distance of the laser to the rotation axis of joint j_3. This way we ensure that the laser does not collide with B. For each of these points we determine all points on the corresponding upper half sphere $S^{\uparrow}$ that lie on a plane tangent to $S^{\uparrow}$ and passing through an e_i as described. We obtain a set of *vantage points* $\mathbf{v}_j$ together with the corresponding edges of B. Those vantage points, for which the edges are parallel, are



a. *Object* **b.** *Grid after one scan* **c.** *Grid after several scans*

Figure 4.5: *Example of a 3D occupancy grid after scanning the 3D object shown in Fig. (a). After the first scan the grid looks like depicted in Fig. (b) and after several scans the grid results like in Fig. (c). Blue grid cells represent a high occupancy probability, red cells are unknown areas.*

then connected to *sub-paths*. These sub-paths correspond to different sweeping motions of the scanner along the faces of B.

Given this set $\mathfrak{S}$ of sub-paths we proceed as follows: Out of a set of h sub-paths that are nearest to the current arm position we select the one with maximum utility. The value h will be denoted as the *exploration horizon*. If the obtained utility is lower than a given *minimum utility* u_{min}, the sub-path is omitted and the next h sub-paths are considered. If a sub-path is found, it is executed and the newly gathered information is incorporated into the occupancy grid. The algorithm terminates if there is no sub-path left.

Implementation

We have tested this exploration strategy with a real 3d object, a piano stool (see figure 4.5a). The stool was placed on top of a table in front of the manipulator. The cell size of the 3D occupancy grid was $1cm$. After the first scan we obtained the grid shown in figure 4.5b. We can see that the front part of the object has been observed, but there are still many occluded areas. Note also that there are unknown areas where the laser could not reach due to its restricted angular resolution. For the exploration, we chose an exploration horizon h of 10 paths. Initially, there were 424 sub-paths, out of which only 16 were executed (the maximum would have been 42). The occupancy grid that resulted from the exploration is shown in figure 4.5c. We can see that all grid cells above the table are either occupied (drawn in blue) or free (not drawn). Areas below the table were not reachable by the scanner and therefore remain unobserved (red cells).

4.5 3D Triangulations

So far we have seen data structures for 3D data that are point-based. Dealing with points has the advantage of being easily representable. Also, point representations are "closer" to the measurement representation of a 3D sensor, because a sensor usually provides single (range) measurements and can not determine the geometrical structure of the environment. However, in many applications this geometrical structure is needed. For example, if a texture obtained from camera images is to be mapped onto the 3D data, then the mapping can be substantially improved by exploiting planar structures like walls and floors in the 3D data. This way a resolution independent mapping can be achieved by assigning texture coordinates to the vertices of a planar polygon that was found in the data. In a point-based setting like point clouds or occupancy grids, each point will have an assigned color value, but the information about texture between the data points can not be represented. Furthermore, a polygon-based 3D model is usually more compact than a point-based model.

In the literature, many different approaches to find planar structures from point-based 3D data can be found. We will not give an overview of all these techniques here, but instead show some simple approaches to obtain triangulated meshes from point-based data. In chapter 5 we will present a new technique to detect planar structures in point clouds using a probabilistic approach. References to other techniques will be given there.

In the following, we will describe and compare three different approaches to triangulate point clouds. The main focus in these techniques is to find a good visualization of the data. This means that no polygons with more than three vertices are provided by these algorithms, because triangles are sufficient, and in fact better suited for 3D visualization engines like OpenGL based graphics processors.

4.5.1 Local Triangulations

The first idea to find a triangular mesh from a point cloud is by connecting neighboring points to triangles. However, care has to be taken here, because in a general setting it is not clear which points have to be connected to triangles. Therefore, we consider first a special case in which knowledge about the data acquisition process can be used to do a triangulation. Then, we formulate a possible solution for finding triangles in arbitrary point clouds.

Exploiting 2D manifolds

In many cases, we are in a situation in which the point set that has to be triangulated is acquired as a two dimensional projection of the 3D space. For example, when the data is acquired with a 3D laser scanning device such as the one shown in figure 4.2, the resulting data consists of several scan lines, one for each new position of the laser. Each of these scan lines consists of a vector of laser range readings, from which the 3D points are computed. Therefore, the 3D point cloud is stored in a 2D manifold, namely an array of points with a line index i for each scan line and a column index j for each laser beam.

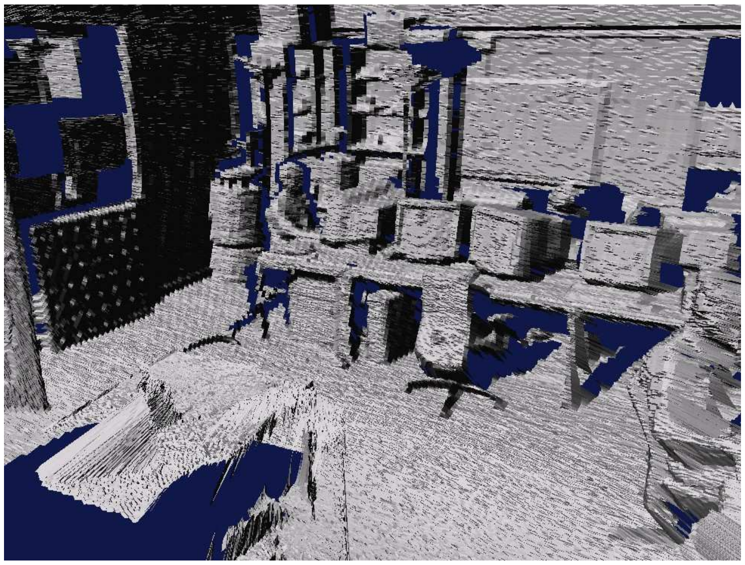


Figure 4.6: *Local triangulation of a point cloud that was computed from an indoor scan. For the triangulation, the fact that all 3D points lie on a 2D manifold was used. The 3D scan consists of scan lines and each scan line of* 180 *range readings. Thus, each point can be parameterized by the scan line index and the beam index.*

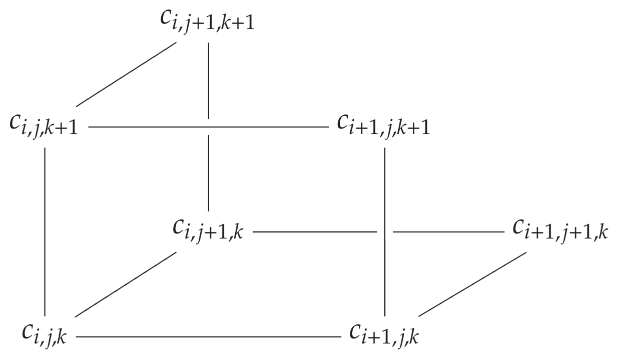
If we denote a data point that is computed from the laser range reading of the i-th scan line and the j-th laser beam, we can triangulate the data using the following scheme

$$\begin{tikzcd}[arrows=-]
 & {} & {} & {} & \\
{} \arrow[r, dashed] & \mathbf{p}_{i-1,j-1} \arrow[r] \arrow[u, dashed] \arrow[d] & \mathbf{p}_{i-1,j} \arrow[r] \arrow[u, dashed] \arrow[d] & \mathbf{p}_{i-1,j+1} \arrow[r, dashed] \arrow[u, dashed] \arrow[d] & {} \\
{} \arrow[r, dashed] \arrow[ru, dashed] & \mathbf{p}_{i,j-1} \arrow[r] \arrow[ru] \arrow[d] & \mathbf{p}_{i,j} \arrow[r] \arrow[ru] \arrow[d] & \mathbf{p}_{i,j+1} \arrow[r, dashed] \arrow[ru, dashed] \arrow[d] & {} \\
{} \arrow[r, dashed] \arrow[ru, dashed] & \mathbf{p}_{i+1,j-1} \arrow[r] \arrow[ru] \arrow[d, dashed] & \mathbf{p}_{i+1,j} \arrow[r] \arrow[ru] \arrow[d, dashed] & \mathbf{p}_{i+1,j+1} \arrow[r, dashed] \arrow[ru, dashed] \arrow[d, dashed] & {} \\
 & {} & {} & {} &
\end{tikzcd}$$

As an example of this triangulation method, consider the indoor scene shown in figure 4.6. The triangulation improves the visualization compared to a point cloud because the triangular faces can be rendered with shading methods. Note that the vertices of the triangles correspond to the raw sensor data, therefore the triangulation appears noisy.

Local Triangulations of Arbitrary Point Clouds

In the case that the points are not given as a 2D manifold, there is no ordering of the points and it is more difficult to determine which points should be connected to triangles. One simple way to find an ordering is by means of a 3D grid structure. For example, one could use a 3D histogram to estimate the density of the point cloud. Then, we can connect neighboring cells to triangles whenever the neighbor cells reflect a sufficiently high data density. Several possible ways to connect neighboring cells exist. One is by considering the three squares adjacent to the current cell $c_{i,j,k}$ as in the following scheme



Then, by subdividing these squares into triangles, we obtain a triangulation. An example of this simple technique can be seen in Fig. 4.7a, where we used the point cloud data of the piano stool shown in Fig. 4.5. As we can see, this triangulation gives a rather bad result. The main reason for this is that the object to be modeled is not represented as a polyhedron, where the surface consists of triangles that are adjacent so that at each edge there are exactly two triangles. An approach that avoids this is presented in the following.

4.5.2 Contouring Algorithms

A major problem when using local triangulations is that the resulting model is not a polyhedron, i.e. it does not consist of a closed 2D surface. This is because at a given position in the point cloud it is impossible to say where the inside of the object is and where the outside. A way to address this problem is by using *contouring algorithms*. These operate on a grid similar to the occupancy grids presented in section 4.4. The difference here is that each grid cell stores the *signed distance* to the surface of the 3D object that is to be modeled. The contouring algorithm then finds a surface that intersects the cells where the distance crosses zero. One popular contouring algorithm is the *marching cubes* algorithm presented by Lorensen and Cline [1987], which is implemented in the Visualization Toolkit (see Schroeder et al. [1997]). Another contouring algorithm is based on the level set method by Sethian [1999].

The Normalized Density Function

The problem that arises when applying contouring algorithms to the mesh generation of point clouds is that the signed distance function is not available. This is, as mentioned already, because the inside and the outside of the object can not be distinguished. Instead of the signed distance we could use the unsigned, i.e. the Euclidean distance of each cell in the grid to the closest point in the point cloud. However, then we can not use the zero crossings for the contouring, because the unsigned distance function actually never crosses zero. It reaches zero only when a data point falls exactly inside a cell of the grid, which is very unlikely. In addition, the Euclidean distance function is not robust against noise in the point cloud data and against the density of the point cloud. Therefore, we use a different function, namely the *normalized density function*.

The normalized density function first computes for each cell in the 3D grid around the point cloud the number of data points that are closer than a given distance. Then, these numbers are divided by the maximum number of neighbors that have been encountered in the grid. Thus, the values in the 3D grid range between 0 and 1. The advantage of the normalized density function is that the value of 0 is definitely reached at cells that are farther away than the chosen neighbor distance. Thus, we can use the zero level as a contouring threshold. In practice however, it is recommendable to use a threshold slightly higher than zero to obtain a contour that is closer to the point cloud data. Also, the normalized distance function has the positive effect of smoothing the input data by considering a vicinity of data points instead of the closest point. The grade of smoothing can be adjusted by different choices of the neighbor distance.

The Mesh Generation Algorithm

The overall algorithm to generate meshes from point clouds, which we will use in this work for comparison with other meshing techniques, can be described as follows: First, we compute the oriented bounding box (OBB, see section 4.3.2) for the given point cloud. This minimizes the amount of memory required for the 3D grid. Then, we divide the OBB into equally sized voxels or grid cells and compute the normalized density function for each voxel. Finally, we apply a contouring algorithm such as marching cubes with a contouring threshold close to zero. As an example, we applied this algorithm to the point cloud data of the piano stool shown in Fig. 4.5. The result is shown in Fig. 4.7b. As we can see, the algorithm produces a closed surface, but it also introduces some artifacts from the underlying grid structure. A method that addresses this problem will be presented in the next section.

4.5.3 Alpha-shapes

One drawback of the contouring algorithms is that they require the evaluation of some function at discrete points in 3D space. This has the effect that the 3D contours never look perfectly smooth, because the steps from the grid structure will always be visible. For a really smooth surface, the grid resolution must be very high, but this requires a huge amount of memory and slows the contouring process down dramatically. Therefore, Bernardini and Bajaj [1997] developed a triangulation method that uses the points in the point cloud as vertices. It is based on the Delaunay triangulation of the points. For a given value α the alpha-shape is defined by all simplices in the Delaunay triangulation

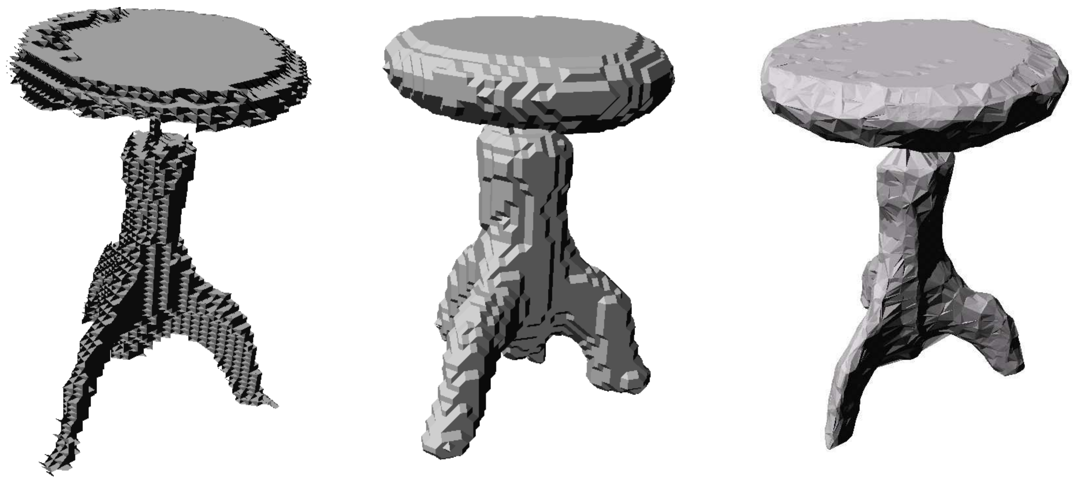


Figure 4.7: *Different types of triangulations of the piano stool shown in Fig. 4.5a. See text for a detailed description of the different meshing algorithms.*

Algorithm	Computation Time	Drawbacks
Local triangulation	++	no valid polyhedrons, artifacts
Contouring	−	artifacts due to grid structure
Alpha shapes	+	sensitive to noise in the data

Table 4.1: *Comparison of the three presented 3D triangulation algorithms.*

that have an empty circumsphere with a squared radius that is equal or smaller than α. Here, a circumsphere is said to be empty if it does not contain any data points in its interior, i.e. excluding the boundary. In general alpha-shapes are not connected, but the α-value can be chosen so that the shape only consists of one connected component. This is often a good choice, but it can also lead to wrong triangulations in object regions that are sampled only sparsely. In such cases the α-value should be chosen appropriately. The optimal choice of α, however, is difficult to find, which is a major drawback of the triangulation method using alpha-shapes.

4.5.4 Comparison of 3D Triangulations

By looking at the different triangulation results in Figure 4.7 and by analyzing the algorithms we can say the following:

- Local triangulation is fast and easy to implement, but gives poor results. It introduces artifacts due to the underlying grid structure and does not result in correct polyhedrons.

- Contouring algorithms provide a closed surface of the object and they are relatively robust against noise in the data. However, they are comparably slow in computation time and also introduce artifacts.
- Alpha-shapes are faster to compute than contoured meshes and do not require the computation of a 3D grid. However, they directly connect the input data points and are therefore sensitive to noise in the data.

To summarize, we show the comparison of all three algorithms in Table 4.1. From this, we conclude that both the contouring algorithm and the alpha-shapes can be used to generate 3D mesh representations from point cloud data, where the alpha-shapes are preferred when the noise in the data is low. In contrast, a contoured mesh should be chosen when it is required to obtain a valid polyhedron with a closed surface.

4.6 Summary of Standard 3D Data Representations

In this section, we summarize the comparison of the standard 3D data presentations presented in this chapter. In particular, the approaches have the following properties:

- Point clouds represent directly the raw 3D data. Their drawback is that their memory requirement grows linear with the number of data points even if always the same 3D region is scanned. Also, the search for a particular data point is not efficient and the shape of 3D objects can not be modeled.
- Tree structures implement an efficient search for data points. However, they also grow linearly with the number of stored points and provide no good shape representation for 3D objects.
- Occupancy grids represent the environment more accurately because they also model the free space. Their memory requirement does not increase with the number of data points, but with the volume of the scanned environment. Thus, for small environments which are scanned very often they are very efficient.
- Triangular meshes require a memory size that is either linear in the number of data points or proportional to the volume of the data, depending of the triangulation algorithm. They are particularly useful for representing the shape of 3D objects.

Data structure	**Memory requirement**	**Search time**	**3D shape representation**
Point cloud	$O(n)$	$O(n)$	−
Tree structures	$O(n)$	$O(\log n)$	−
Occupancy grid	$O(w \cdot d \cdot h)$	$O(1)$	+
Triangular mesh	$O(w \cdot d \cdot h)$ or $O(n)$	$O(n)$	++

Table 4.2: *Comparison of standard 3D data representations. Here, n denotes the number of data points and w, d and h denote the width, height and depth of the data respectively.*

Table 4.2 summarizes this comparison. For the context of mobile robotics we conclude that occupancy grids and triangular meshes are well suited for representing 3D data, but they both have drawbacks that limit their applicability. Especially for the task of localization and path planning, they are not useful, as we will see later in chapter 6. Therefore, new data representations have to be found, which will be the subject of the next two chapters.

If I have a thousand ideas and only one turns out to be good, I am satisfied.

Alfred Nobel (1833 - 1896)

5 Planar Approximations

5.1 Introduction

As we have seen in the previous chapter, it is often cumbersome and inconvenient to operate with 3D data represented as point clouds directly. In many applications, especially in real-time scenarios, a more compact and abstract representation is required. Humans build such compact models intuitively by visually inter- and extrapolating the perceived set of 3D points. In essence, this is comparable to the matching of the point cloud with known 3D patterns, such as planes, spheres, boxes, etc. In the computer vision community this problem is known as *range image segmentation* and is defined as finding three dimensional primitives (planes, boxes, cylinders, etc) that best explain the point cloud data. In the literature, a whole body of work can be found on the topic of range image segmentation. A good overview is given by Hoover et al. [1996], later work was done by Pulli [1997].

For the scope of this work, we will focus on a specialized version of the range image segmentation problem, namely the detection of planar structures in point clouds. Thus, instead of matching arbitrary 3D shapes into the points, we want to fit planes and planar polygons into the data. For that purpose, the plane fitting techniques developed in section 2.6 will serve us as the basic operations. The novel plane extraction technique developed in this work is based on a probabilistic framework where the planes are considered as random variables underlying a Gaussian distribution. It uses the Expectation Maximization (EM) algorithm applied to a hierarchical Bayesian Network which represents the joint probability of the point coordinates and the positions of the planes. The contribution of the presented method is the introduction of *main directions* which consist of normal vectors perpendicular to one or more of the planes to estimate. In addition to the data points, these main directions constrain the position of the planes in the sense that planes that are nearly parallel, i.e. which belong to the same main direction, are very likely to be perfectly parallel. This construct proves especially useful in indoor and other man-made environments (e.g. cities like Manhattan), where most of the planes can be clustered into three orthogonal main directions. As we will see, the new approach yields better plane detection results than the standard probabilistic approach by Liu et al. [2001], which does not incorporate main directions.

This chapter is organized as follows: In section 5.2 we will describe the principles of state-of-the-art techniques to identify planes in point clouds. In particular, these approaches can be subdivided into *region-based* and *edge-based* approaches, as has been done by Fan et al. [1987] for the general range image segmentation problem. Region-based approaches start with a planar region defined by a point that is selected from the point cloud and try to enlarge this region by adding adjacent points until some planarity condition is violated. A basic example will be given in section 5.2.1. In contrast, edge-based approaches try to find the boundaries of the planar regions, defined as edges and fit planes into these edges. A concrete solution using this approach is presented in section 5.2.3. In addition, we show in section 5.2.2 the idea of using a three dimensional Hough transform to find planes in point clouds. A rather new approach by Liu et al. [2001] uses a probabilistic model, yielding better results, as we will see in section 5.3. In section 5.4, we will present a first extension of this approach by introducing main directions as mentioned above. It will be shown in experiments that this approach yields better results. Then, in section 5.5 we will further extend the approach by adding color information into the plane computation. Again, we show in experiments that the plane detection results are substantially improved.

5.2 Plane Extraction: The State of the Art

In this section, we will briefly summarize the most common standard techniques for the problem of plane extraction in range scan images. These techniques are *region growing*, *Hough transform* and plane extraction based on *edge detection*.

5.2.1 Region Growing

A standard way of detecting planes in 3D point clouds is based on a *region growing* algorithm that operates on the points together with the normal vectors computed for each data point. Algorithm 2 shows the particular steps of this method. In line 1, a normal vector $\mathbf{n}$ is computed for each data point $\mathbf{z}$. This can be done in many different ways (see Pulli and Pietikäinen [1993] for an overview). The most common approach is to perform a Principle Component Analysis (PCA) on the vicinity $\mathcal{V}_z$ of $\mathbf{z}$. The first two eigen vectors of this point cloud $\mathcal{V}_z$ determine the plane that best fits the points of $\mathcal{V}_z$, i.e. the neighbors of $\mathbf{z}$. The normal vector of this plane, which is equal to the third eigen vector, is then defined as the vector $\mathbf{n}$.

Once the normal vectors are computed, the algorithm proceeds as follows: As long as not all points have been considered, it draws randomly a point $\mathbf{z}$ and the corresponding

Algorithm 2 *findPlanesRegGrow*($\mathcal{P}$) – Detect planar regions in 3D point cloud $\mathcal{P}$ with region growing

```
1:  $\mathcal{N} \leftarrow$ N ($\mathcal{P}$)
2:  $\mathfrak{R} \leftarrow$ E S O R ()
3:  while A P C ($\mathcal{P}$) do
4:    $(\mathbf{z}, \mathbf{n}) \leftarrow$ R ($\mathcal{P}, \mathcal{N}$)
5:    $\mathcal{R} \leftarrow$ E R ()
6:    $Q \leftarrow$ Q $(\mathbf{z}, \mathbf{n})$
7:    while E (Q) do
8:      $(\mathbf{z}', \mathbf{n}') \leftarrow$ Q ($Q$)
9:      if B $(\mathbf{n}, \mathbf{n}') < \theta$ then
10:       $\mathcal{R} \leftarrow \mathcal{R} \cup \mathbf{z}'$
11:       $S \leftarrow$ N $(\mathbf{z})$
12:       for $(\mathbf{z}_s, \mathbf{n}_s) \in S$ do
13:         if $\mathbf{z}_s \notin \mathcal{R}$ then
14:           $Q \leftarrow$ Q $(Q, \mathbf{z})$
15:         end if
16:       end for
17:     end if
18:   end while
19:   $\mathfrak{R} \leftarrow \mathfrak{R} \cup \mathcal{R}$
20: end while
```

normal vector $\mathbf{n}$ from the set of points $\mathcal{P}$. This single point $\mathbf{z}$ initializes a new planar region $\mathcal{R}$. Then, in line 6, a queue of neighboring points is created, where neighbors are points that are not farther away than a given threshold. The idea is that all neighbors of $\mathbf{z}$ together with all neighbors' neighbors should be added to the queue, if they haven't been added to the region $\mathcal{R}$ already. This queue defines a set of candidates for addition to the planar region $\mathcal{R}$. All candidates that have a normal vector with a small angular deviation from the initial normal vector $\mathbf{n}$ are added to $\mathcal{R}$. This way it is guaranteed that the normal vectors of all points in $\mathcal{R}$ are approximately collinear. Furthermore, $\mathcal{R}$ is guaranteed to define a contiguous set of points, because all elements of $\mathcal{R}$ are neighbors of a smaller subset of $\mathcal{R}$.

Main Drawbacks of Region Growing

The advantage of the region growing approach is that it is relatively easy to implement and that it is comparably fast in execution. However, some principle problems exist, which we will describe in the following.

Normal vector computation: The algorithm depends on an accurate computation of the normal vectors. This is not possible in general, e.g. in the case of isolated points that have no or only a few neighbors. For these points, no normal vector can be computed.

Also, it is difficult to find a radius for the neighborhood, on which the normals are computed. Using a big radius smoothes the data too much and results in large areas at the borders of a planar structure that should not be identified with the planar structure. In addition, the time to compute the normal vectors grows dramatically when using a big neighborhood radius, because the *number* of neighbors usually gets very large (in the worst case of a densely sampled cube it is a cubic function of the radius). On the other side, a small radius results in very noisy normal vectors, that can not be classified accurately. One could think of an adaptive way to define the neighborhood radius, e.g. depending on the data density. But it is difficult to define a functional relationship between the data density and the optimal neighborhood radius.

Normal vector clustering: The approach presented in Algorithm 2 clusters the point data into regions of similar normal vectors. One problem with this approach is that the normals of all possible candidate points are compared to the normal vector of the – randomly chosen – first data point in a region. This may result in sub-optimally detected planar regions. For example, in the case of a single planar surface it may happen that the first point that is drawn is close to the border, where the local normal vectors usually deviate from the global plane normal. This results in a region where all normals are close to the local estimate of the plane normal and not to the global plane normal.

Behavior at edges: Another problem with the standard region growing approach is that points at sharp edges in the data can not be handled correctly. This is due to the fact that the local normal vectors at edges where two planes meet, smoothly interpolate between the global normal vectors of both planes. In other words, edges are *rounded*. This results in edge regions that are detected as new planar regions with a normal vector that interpolates between the two plane normals. This means that the obtained planes are a smoothed approximation of the point data, which is unacceptable.

5.2.2 Hough Transform

Several authors have proposed to perform a Hough transform in three dimensions for detecting planes (see for example Iocchi et al. [2000], Okada et al. [2001], Sabe et al. [2004]). This is done as follows: First, a different representation for planes is used instead of the standard notation. In particular, if $\mathbf{n}$ is the normal vector and d the distance of a plane p to the origin, then the plane is usually defined by

$$p_{\mathbf{n},d} = \{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{n} \cdot \mathbf{x} = d\}. \tag{5.1}$$

This has the disadvantage that a plane is represented by four values, namely three for the normal vector and one for the distance. But, as the normal vector has always unit length, the number of degrees of freedom of a plane is only three (see section 2.4 for details). Therefore, a different notation is used instead

$$p_{\mathbf{n},d} = \{(x_1, x_2, x_3) \in \mathbb{R}^3 \mid x_1 \cos\alpha \cos\epsilon + x_2 \sin\alpha \cos\epsilon + x_3 \sin\epsilon = d\}, \tag{5.2}$$

where α and ϵ denote the azimuth and the elevation of the normal vector respectively. Thus, a plane is defined by the values α, ϵ and d. Given this parametric representation of a plane in 3D, the Hough transform proceeds as follows: All 3D input data points are mapped into a 3D array, which covers the whole range of all possible values of α, ϵ and d. This array is called the *accumulator* and initially contains zeros in all cells. The mapping is done so that one point from the input data corresponds to a set of cells in the accumulator. The cells are determined using Equation (5.2), i.e. a point $\mathbf{x} = (x_1, x_2, x_3)$ is mapped to the set of cells (α, ϵ, d), for which the equation holds. The values stored in all these cells are incremented for each given point $\mathbf{x}$. The motivation behind this is that each cell in the accumulator corresponds to a plane in parametric representation. By incrementing the values in the cells, all planes that possibly pass trough the input data point $\mathbf{x}$ are marked. A plane that passes through many points is thus found by determining an accumulator cell with a high value. The 3D index of that cell determines the parameters of the plane, while the stored value corresponds to the number of points which are close to the plane.

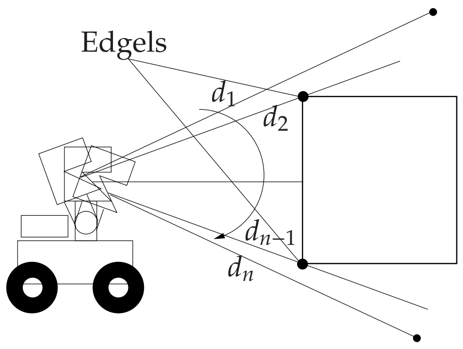
Discussion Using the 3D Hough transform to detect planes can be seen as a clustering of planes in 3D parameter space using a histogram. Therefore, we are faced with the typical problems of histogram based clustering, namely the selection of the histogram's bin size and the occurrence of local maxima in the histogram. The bin size, which determines the resolution of the accumulator, is a delicate parameter: if it is too big, the resulting plane parameters are inaccurate, while a too small bin size results in planes that fit only a small number of points and therefore do not generalize. Another problem is that in the vicinity of a local maximum in the accumulator, usually other local maxima occur. These other maxima often correspond to the same plane in reality, but due to noise in the data and the discretization of the parameter space, many different but similar planes may result.

5.2.3 Edge-based Approach

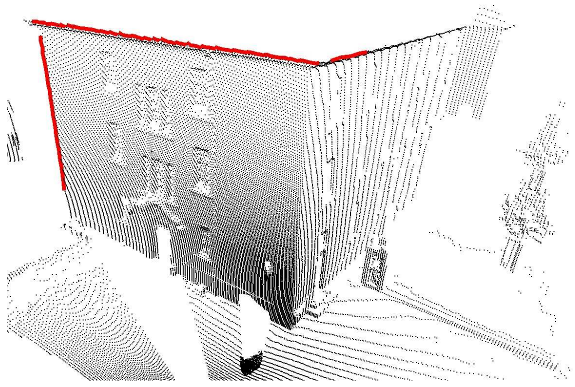
There are several techniques for range image segmentation that are based on the detection of *region borders*. In our case the region borders consist of 3D line segments, because we are interested in finding planes, or better planar polygons. The borders of these polygons are defined by edges. In the following, we will give an example of a method to detect edges in 3D range scans and, based on these, to find planes that fit into the edges. The example is taken from the work Triebel and Burgard [2005], which is one of the contributions mentioned in Section 1.3.

Edge Detection For a given point cloud, we can perform the following steps to extract edges.

1. We detect a set of edge points (*edgels*). These are defined as those scan points of a vertical scan line for which one vertically neighboring point is far away from the view point compared to the edgel itself (see Fig. 5.1a). A vertical neighbor in this context is defined as the scan point from the previous scan line with the same beam index. In section 4.5.1, we mentioned already this definition of neighbor points.



a. *Detection of edgels. An edgel is defined as a scan point in a vertical scan line that has a distance d_i from the view point which is smaller than the difference of a neighbor's distance and a given threshold τ, i.e.: $d_i < d_{i-1} - \tau$ or $d_i < d_{i+1} - \tau$*



b. *Edge features extracted from a single 3D scan of a building. For the plane detection, we are only interested in long edges. In this case, all edges that are longer than 3m are selected. Note that the edgels at the building corner have not been detected because the difference in range distance was below the threshold τ.*

Figure 5.1: *Edge detection in range scans. Figure (a) shows the definition of edge points (edgels), in figure (b) we see an example of the edge detection in a real 3D scan. The two long edges are sufficient to determine a plane for the front wall.*

2. We calculate the tangent vectors $\mathbf{t}_j$ for each edgel $\mathbf{e}_j$. Here, $\mathbf{t}_j$ is defined by the first *principle direction* for a set of edgels in the vicinity of $\mathbf{e}_j$. This principle direction is computed as the eigenvector of the covariance matrix that corresponds to the largest eigenvalue (see section 2.6.1 for a detailed description).

3. Next, we cluster the edgels twice. The first clustering is done with respect to the tangent directions. We use a *spherical histogram* to find tangent vectors that point into similar directions. The obtained clusters are then clustered with respect to the positions of the edgels in space. This is done using a region growing technique.

4. All edgels in each cluster are connected to a poly-line. To this end, the edgels are ordered according to their position on the 3D line. This means, we first project all edgels onto the line that fits best into the cluster and then sort them along the line direction.

Fig. 5.1a shows an example of a set of edge features extracted from a real 3D scan. In this figure, only poly-lines with a minimum length of 3*m* are shown. Some edges, such as the corner of the building or the edge between the wall and the floor, have not been detected. This is because the difference in range distance between two vertical neighbor points is not large enough. However, for the application in this example, it is not necessary to detect these edges, as there are enough other "sharp" edges that can be used to find the planes.

Fitting Planes into the Edges

Once the edge features have been extracted from the point cloud, we search for planes that fit best into the edges. This is again a situation in which subsets (of edges) have to be found which in some sense are best grouped together (to planes). However, it is different from the classical clustering problem, because one edge can lie in several planes. This means that standard clustering methods like region growing, k-means-clustering, etc, cannot be applied. Therefore, we instead use a variant of the RANSAC algorithm by Fischler and Bolles [1981].

1. We start by randomly sampling a pair (e_1, e_2) of non-collinear edges from the set of all edges. The sampling is done with a probability proportional to the sum of the length of both edges. This way we obtain with a higher likelihood planes that correspond to large planar areas in the scan.

2. We find the plane p_m whose normal vector is orthogonal to both edges of the pair. Here we have to consider two different cases, namely that e_1 and e_2 are parallel, but not collinear, or that e_1 and e_2 are not parallel:

   - In the first case, an orthogonal vector to e_1 and e_2 is not uniquely defined. Therefore, we define P_m in this case as the plane that minimizes the squared distance to all edgels from e_1 and e_2. This is computed by the plane fitting method from section 2.6.1.
   - In the second case, the normal vector of P_m is defined by the cross product of the main directions of e_1 and e_2 (see section 2.3).

3. We measure the quality of P_m by summing up the length of all edges in the scan that lie entirely inside a given corridor around P_m. The set of these edges is called the *support* of P_m, while the sum of the edge lengths is denoted as the *support value*.

4. We apply a hill climbing strategy to obtain more general planes by fitting a new plane P'_m into all edges from the support of plane P_m. This is done using the planes-to-lines fitting algorithm presented in section 2.6.2, with the difference that the point $\mathbf{q}$ which determines the position of the plane is computed as the mean of all edgels in the support. This way we ensure that the resulting plane is close to the actual edge segments and not to the lines defined by the edges. The hill climbing is done by iteratively fitting a plane into the current support and computing a new support for this new plane as long as the support value increases.

Later, in chapter 7, we will use this plane extraction algorithm in a situation where a global 3D map representation and a set of optimal robot positions is computed from a set of local 3D scans. This is usually referred to as *simultaneous localization and mapping* (SLAM). We will see that the planes found by the algorithm can be used to introduce global constraints between the robot poses, making the SLAM algorithm more robust. The results of the plane extraction based on edges will be shown there.

Discussion The plane extraction algorithm presented here works well in situations where edgels can be extracted reliably, as it is the case for the presented example. Of course, the edge detection that was used here is rather simple and there are much better methods available. However, it still remains a problem to find planes where no or not enough edges have been found. In addition, the algorithm requires many parameters to be adjusted for a good plane detection. In fact, this is a drawback of most non-probabilistic approaches (recall that region growing, for example, needs a threshold for the angles between the normal vectors and one for determining the neighborhood). The problem with these parameters is often that they are hard to determine intuitively and must be tuned experimentally. In the following, we will present an algorithm based on a probabilistic framework that uses only one parameter, which depends on the sensor and not on the data set.

5.3 A Probabilistic Approach

Recently, a new method to detect planes from 3D range data has been proposed by Liu et al. [2001]. This approach will be described in the remainder of this section, because it forms the basis of the extensions presented in the following sections. The novelty of Liu's approach is that the planes to be detected are considered as random variables, as well as the N data points, which will be denoted as the *measurements* $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$. The idea behind this is the following: Suppose we are given a plane θ with its normal vector $\mathbf{n}$ and its distance d to the origin. Then there will be a certain probability that in the vicinity of this plane there is a 3D point measurement $\mathbf{z}$. This probability is called the *measurement model* and will be denoted as $p(\mathbf{z} \mid \theta)$. Its value is assumed to be proportional to the Euclidean distance $d(\cdot, \cdot)$ between $\mathbf{z}$ and θ. If we assume Gaussian measurement noise with variance ρ, we can formulate the measurement model as

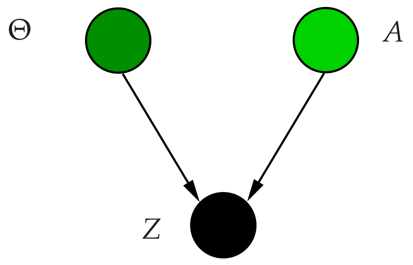
$$p(\mathbf{z}_n \mid \theta_m) \quad = \quad \frac{1}{\sqrt{2\pi\rho^2}} e^{-\frac{1}{2}\frac{d(\mathbf{z}_n,\theta_m)^2}{\rho^2}}. \tag{5.3}$$

Here the index n runs over all N measurements and the index m over all M planes. For now, we assume that the number M of planes is known, later we will drop this assumption.

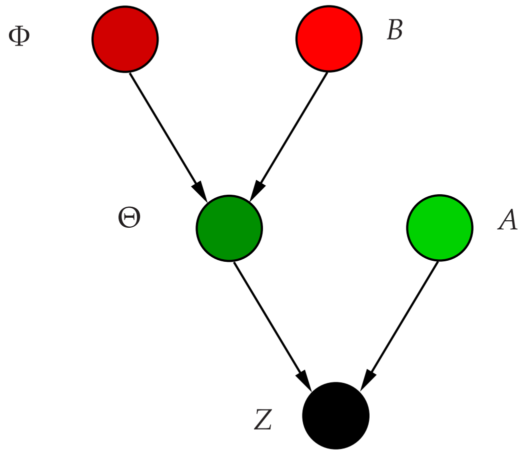
In equation (5.3) we assumed that the measurement $\mathbf{z}_n$ corresponds to the plane θ_m. However, in general we are given a set of M planes $\theta_1, \ldots, \theta_M$ and we do not know to which of these planes $\mathbf{z}_n$ corresponds. Therefore, we introduce a set of *correspondence variables* $A_n = \{\alpha_{n1}, \ldots, \alpha_{nM}\}$. Each of these variables is binary and reflects the fact that measurement $\mathbf{z}_n$ corresponds to plane θ_m. Each measurement can correspond to only one plane, therefore the sum of all variables in A_n is always 1.

Now, if we assume that all M correspondences are equally likely, we can formulate the probability of a measurement $\mathbf{z}_n$ and the correspondences A_n given the set of all planes $\Theta = \{\theta_1, \ldots, \theta_M\}$ as

$$p(\mathbf{z}_n, A_n \mid \Theta) = \frac{1}{M\sqrt{2\pi\rho^2}} e^{-\frac{1}{2}\sum_{m=1}^{M} \alpha_{nm} \frac{d(\mathbf{z}_n,\theta_m)^2}{\rho^2}}. \tag{5.4}$$



a. *Bayesian network used in plane extraction using standard EM.*



b. *Extended Bayesian network for plane extraction using hierarchical EM.*

Figure 5.2: *Bayesian networks used for plane extraction. Figure (a) shows the net used in the standard approach. In figure (b) we see the extended network for the hierarchical approach.*

Note that in this equation only one term in the sum is non-zero, namely the one for which the correspondence between $\mathbf{z}_n$ and a plane θ_m holds. Under the assumption that all measurements are statistically independent, the overall data likelihood can be expressed as

$$p(Z, A \mid \theta) = \prod_{n=1}^{N} \frac{1}{M\sqrt{2\pi\rho^2}} e^{-\frac{1}{2}\sum_{m=1}^{M} \alpha_{nm} \frac{d(\mathbf{z}_n, \theta_m)^2}{\rho^2}}. \tag{5.5}$$

The goal now is to find a set of planes θ that maximizes the data likelihood given in equation (5.5). The problem here is that the correspondences A are not known and can not be observed like the measurements. This means that they are *hidden* variables in a Bayesian network like the one shown in figure 5.2a. A standard method to solve such a problem is the *expectation maximization* (EM) algorithm described by Dempster et al. [1977]. In our example this means that, instead of maximizing (5.5) directly we maximize the *expected log-likelihood* where the expectation is computed over all possible correspondences between points and planes, i.e. over all possible assignments of A. Thus, we have

$$LL := E_A[\log p(Z, A \mid \Theta)]. \tag{5.6}$$

Note that by taking the logarithm we alter the function to be maximized, but the value Θ for which this maximum is reached stays the same, because the logarithm increases monotonically. Later we will see why the computation in log-space is more convenient.

The problem arising here is that equation (5.6) can not be maximized directly, because the computation of the expected value requires a fixed assignment to the plane variables Θ. Therefore, the maximization is done iteratively. We start with an initial guess for the plane variables $\Theta^{[0]}$ and compute at each iteration i a new set of planes $\Theta^{[i+1]}$, i.e.

$$\Theta^{[i+1]} = \operatorname*{argmax}_{\Theta} E_A[\log p(Z, A \mid \Theta) \mid \Theta^{[i]}]. \tag{5.7}$$

In this equation, the notation $E_A[.|\Theta^{[i]}]$ is used to express the fact that for the computation of the expected value over all possible assignments of A the planes $\Theta^{[i]}$ are given.

For the computation of the expression given in (5.7), we first note that the logarithm can be expressed in a more compact form. By taking the logarithm of equation (5.5) and dropping a constant term which is irrelevant for the optimization, we obtain

$$\begin{aligned}\log p(Z, A \mid \Theta) &= \log \prod_{n=1}^{N} \frac{1}{M\sqrt{2\pi\rho^2}} e^{-\frac{1}{2}\sum_{m=1}^{M} \alpha_{nm} \frac{d(\mathbf{z}_n,\theta_m)^2}{\rho^2}} \\ &= -\sum_{n=1}^{N} \log(M\sqrt{2\pi\rho^2}) - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} \alpha_{nm} \frac{d(\mathbf{z}_n,\theta_m)^2}{\rho^2} \\ &\propto -\frac{1}{\rho^2}\sum_{n=1}^{N}\sum_{m=1}^{M} \alpha_{nm} d(\mathbf{z}_n,\theta_m)^2.\end{aligned} \tag{5.8}$$

Plugging this into equation (5.7) yields

$$\begin{aligned}\Theta^{[i+1]} &= \operatorname*{argmax}_{\Theta} E_A\left[-\frac{1}{\rho^2}\sum_{n=1}^{N}\sum_{m=1}^{M} \alpha_{nm} d(\mathbf{z}_n,\theta_m)^2 \mid \Theta^{[i]}\right] \\ &= \operatorname*{argmax}_{\Theta} \left\{-\frac{1}{\rho^2}\sum_{n=1}^{N}\sum_{m=1}^{M} E_A[\alpha_{nm} \mid \mathbf{z}_n, \Theta^{[i]}] d(\mathbf{z}_n,\theta_m)^2\right\}.\end{aligned} \tag{5.9}$$

Here we used the fact that the expectation operator $E[\cdot]$ is linear and can therefore be moved into the sum.

It remains to compute the expected value in equation (5.9). Using the fact that the correspondence variables are binary and applying Bayes' rule we can write

$$\begin{aligned}E_A[\alpha_{nm} \mid \mathbf{z}_n, \Theta] &= p(\alpha_{nm} \mid \mathbf{z}_n, \Theta) \\ &= \frac{p(\mathbf{z}_n \mid \alpha_{nm}, \Theta) p(\alpha_{nm} \mid \Theta)}{\sum_{j=1}^{M} p(\mathbf{z}_n \mid \alpha_{nj}, \Theta) p(\alpha_{nj} \mid \Theta)} \\ &= \frac{\exp\left\{-\frac{1}{2}\left(\frac{d(\mathbf{z}_n,\theta_m)}{\rho}\right)^2\right\}}{\sum_{j=1}^{M} \exp\left\{-\frac{1}{2}\left(\frac{d(\mathbf{z}_n,\theta_j)}{\rho}\right)^2\right\}}.\end{aligned} \tag{5.10}$$

In the last step we used the independence of α_{nm} and Θ and the assumption that the α_{nm} are distributed uniformally. Therefore, the term $p(\alpha_{nm} \mid \Theta) = p(\alpha_{nm}) = p(\alpha_{nj})$ cancels out.

To summarize we can formulate the algorithm as follows:

1. Initialize the plane parameters $\Theta^{[0]}$
2. Compute the expectations $E_A[\alpha_{nm} \mid \mathbf{z}_n, \Theta^{[i]}]$ for all n and all m according to equation (5.10)
3. Find new plane parameters $\Theta^{[i+1]}$ that maximize equation (5.7). This can be done in closed-form, as proposed by Liu et al. [2001] or by an iterative optimization technique such as gradient-descent.
4. If the increase of the expected log-likelihood between step i and $i + 1$ is below a given threshold, stop. Otherwise return to 2.

The convergence proof of the EM algorithm by Dempster et al. [1977] guarantees that the algorithm always converges to a local maximum. However, as in other iterative optimization techniques, EM does not generally find a global maximum of the likelihood function. Therefore, a good initial guess improves the result of the estimation.

5.4 First Extension: Introduction of Main Directions

In addition to the derivation from the previous section, we will assume that we are also given a set Φ of K main directions $\phi_k \in \mathbb{R}^3, k = 1, \ldots, K$ of these planes. Similarly, we introduce correspondence variables β_{mk} to indicate that a plane m belongs to main direction k. Again, we collect the correspondence variables in the sets $A = \{\alpha_{nm}\}$ and $B = \{\beta_{nk}\}$. Our new goal is to maximize the joint posterior $p(Z, \Theta, \Phi, A, B)$. Exploiting the independence between these variables this term can be rewritten as (see Anguelov et al. [2002])

$$\begin{aligned} p(A, B, \Phi, \Theta, Z) &\propto p(Z \mid A, \Theta) p(\Theta \mid B, \Phi) \\ &\propto p(Z, A \mid \Theta) p(\Theta, B \mid \Phi). \end{aligned} \tag{5.11}$$

This is similar to the likelihood function defined in equation (5.5) with the extension of the likelihood model for the planes given the main directions. For the relationship between a plane θ_m and a main direction φ_k we use a second distance function d_2 defined as

$$d_2(\theta_m, \varphi_k) \quad = \quad \sqrt{1 - (\mathbf{n}_m \cdot \varphi_k)^2}. \tag{5.12}$$

This corresponds to the sine of the angle between the normal vector $\mathbf{n}_m$ and the main direction φ_k. Again we assume that the planes belonging to a main direction are normally distributed with variance σ. Given we know that plane θ_m belongs to main direction φ_k we can calculate its likelihood as

$$p(\theta_m \mid \varphi_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\left(\frac{d_2(\theta_m, \varphi_k)}{\sigma}\right)^2\right\}. \tag{5.13}$$

In analogy to the derivation above we obtain

$$p(\Theta, B \mid \Phi) \propto \exp\left\{-\frac{1}{2}\sum_{m=1}^{M}\sum_{k=1}^{K}\beta_{mk}\left(\frac{d_2(\theta_m, \varphi_k,)}{\sigma}\right)^2\right\}. \tag{5.14}$$

This leads to the following expression for the joint posterior

$$p(A, B, \Phi, \Theta, Z) \propto \exp\left\{-\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_{nm}\left(\frac{d_1(\mathbf{z}_n, \theta_m)}{\rho}\right)^2 - \frac{1}{2}\sum_{m=1}^{M}\sum_{k=1}^{K}\beta_{mk}\left(\frac{d_2(\theta_m, \varphi_k)}{\sigma}\right)^2\right\}. \tag{5.15}$$

Our goal is to determine the model (Θ^*, Φ^*) that maximizes the likelihood of the data Z. The update rule for each iteration of the EM algorithm then becomes

$$(\Theta^{[i+1]}, \Phi^{[i+1]}) = \underset{(\Theta,\Phi)}{\operatorname{argmax}}\, E_{AB}\left[\log p(A, B, \Phi, \Theta, Z) \mid \Theta^{[i]}, \Phi^{[i]}\right] \tag{5.16}$$

Inserting the expression (5.15) for the posterior into this equation and exploiting linearity of the expectation we obtain

$$\begin{aligned}(\Theta^{[i+1]}, \Phi^{[i+1]}) = \underset{(\Theta,\Phi)}{\operatorname{argmax}}\Bigg\{ &-\frac{1}{\rho^2}\sum_{n=1}^{N}\sum_{m=1}^{M} E_A[\alpha_{nm} \mid \Theta^{[i]}](\mathbf{z}_n \cdot \mathbf{n}_m - d_m)^2 \\ &- \frac{1}{\sigma^2}\sum_{m=1}^{M}\sum_{k=1}^{K} E_B[\beta_{mk} \mid \Phi^{[i]}](1 - (\mathbf{n}_m \cdot \varphi_k)^2)\Bigg\}.\end{aligned} \tag{5.17}$$

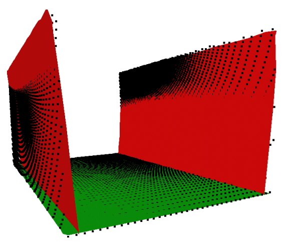
The expectations $E[\alpha_{nm} \mid \Theta]$ are computed as in equation (5.10). Similarly, the $E[\beta_{mk} \mid \Phi]$ are obtained as

$$E[\beta_{mk} \mid \Phi] = \frac{\exp\left\{-\frac{1}{2}\left(\frac{d_2(\theta_m, \varphi_k)}{\sigma}\right)^2\right\}}{\sum_{j=1}^{K}\exp\left\{-\frac{1}{2}\left(\frac{d_2(\theta_m, \varphi_j)}{\sigma}\right)^2\right\}}. \tag{5.18}$$

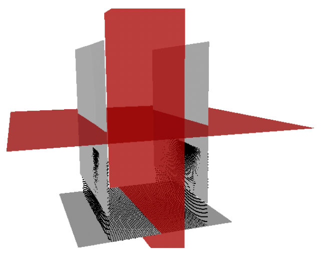
The extended version of our EM based plane extraction now additionally involves the computation of the expectations in equation (5.18) and the maximization of (5.17). In contrast to the maximization step derived in the previous section, there is no closed-form solution for the our extended version. Therefore, we need to use an iterative optimization technique. For the example applications presented in the following, we used the Fletcher-Reeves conjugate gradient algorithm to maximize the log-likelihood function.

5.4.1 Estimating the Model Complexity

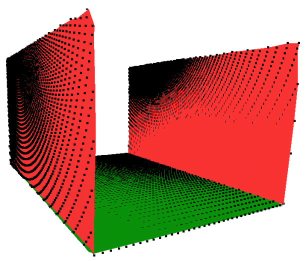
So far, we assumed that the number of planes M and the number of main directions K were given in advance. In practice, however, this is generally not the case. Instead, we



a. *A simulated data set and its planar approximation obtained with the standard EM-based clustering approach. The angle between the normals of the two planes belonging to the parallel walls is* 17.2 *degrees.*



b. *Planes obtained by the initialization process for the data set shown in Figure 5.3a. In this case our algorithm was initialized with five planes and five main-directions from which several were coplanar.*



c. *Model obtained with hierarchical EM. Compared to the model in Figure (a), the angle between the normals corresponding to the two parallel walls is* 1.9 *degrees.*

Figure 5.3: *Example data set and the results obtained by standard EM (a), initialization of planes and main directions (b) and hierarchical EM (c).*

need to estimate M and K – we will call this the *model complexity* – during the estimation process. One standard way to achieve this is by applying the *Bayesian Information Criterion* (see Schwarz [1978] for details), which is calculated as

$$BIC = -2L + (3M + 2K)\ln(N). \tag{5.19}$$

In this equation, L is the log likelihood of the data given the current model where the factor -2 stems from the *BIC* formula. The term $3M + 2K$ corresponds to the number of free parameters (3 for each plane and 2 for each main direction). The goal is to find a model which has a minimal *BIC* value. As can be seen from Equation (5.19), a high model complexity results in a large *BIC* value and hence less complex models are preferred.

To minimize the *BIC* we constantly monitor its value. High *BIC*-values, for example, result from redundancy in the model. For example, it is possible that after convergence of the EM-algorithm two planes are equal. This can happen if the two planes are initialized too close to each other or if the data only supports a smaller number of planes. Such a case of redundancy can be detected applying the leave-one-out rule: after convergence of the EM we calculate the *BIC* for all possible models, in which one plane is left out. If there is a model that has a smaller *BIC* than the current one, then the plane, which has been left out to obtain this particular model, must be redundant and can safely be removed. The same strategy is applied to the main directions.

5.4.2 Implementation Details

Initialization

Since the EM-algorithm can get stuck in local optima of the log-likelihood function, it needs to be initialized appropriately to converge to the global optimum, just like other optimization techniques, e.g., gradient-descent. In our case, the initialization is performed by sampling randomly from all scan points. During this sampling process each point is drawn with a probability proportional to the minimum distance to any of the already existing planes. Thus, when no planes are given each point is equally likely to be drawn. However in later initialization steps, the points that are badly explained by the current model, are more likely to be drawn. To initialize a plane for a point drawn earlier we fit a plane to the points in its local neighborhood. A typical result of the plane initialization for the data depicted in Figure 5.3a is shown in Figure 5.3b. Here, the algorithm was initialized with five planes and five main directions. Since the planes four and five and the main-directions three to five are coplanar to existing planes and main-directions, only three planes and two main-directions are visible.

Weighting Factors for Planes

When clustering the planes into main directions, each plane has a contribution to the resulting main direction according to its normal vector. This contribution is independent on the number of points that were used to calculate the plane in the plane clustering process. The problem that arises here is that planes which are obtained from less data points (or with lower *support*) have the same influence as planes with a high support. This may result in wrong main directions if, for example, a plane resulting from spurious measurements, is clustered together with a wall.

Additionally, the number of planes typically is very small compared to the number of data points. Thus, the influence of the main directions decreases the more data points are given. In practical experiments it turned out to be very useful to introduce weighting factors w_m for the planes which are dependent on the support of a given plane θ_m

$$w_m = \sum_{n=1}^{N} E[\alpha_{nm} \mid \Theta]. \tag{5.20}$$

In the EM we then use a modified distance function

$$d_2'(\theta_m, \varphi_k) = \sqrt{w_m} d_2(\theta_m, \varphi_k). \tag{5.21}$$

Sketch of the Algorithm

Our algorithm proceeds as follows:

1. Start with a fixed number of M_0 planes.
2. Initialize a main direction for each plane by taking the normal vector of that plane. This means the initial number of main directions K_0 equals M_0.
3. Apply EM until convergence
4. Drop one main direction as long as the *BIC* of the reduced model increases. This results in a new number of main directions $K_{i+1} \leq K_i$

5. Drop one plane as long as the *BIC* of the reduced model increases. This results in a new number of planes $M_{i+1} \leq M_i$

6. Select a new plane from the initialization queue and take its normal vector as a new main direction. Adding these to the model increases the complexity:

$$K_{i+1} := K_{i+1} + 1 \text{ and } M_{i+1} := M_{i+1} + 1$$

7. If no such plane can be found, stop. Otherwise go back to 3).

Note that it is not possible that K_i exceeds M_i in any time step i.

Post-Processing

So far, the goal of our algorithm was to find planes and main directions. In practice, however, we want to represent the environment as a set of *polygons*, because they indicate the faces of the objects in the environment. In general, a plane that is found in the data set contains more than one polygon. A typical situation is a wall that is "interrupted" by a doorway. In our implementation, we choose the following approach to extract polygons from planes:

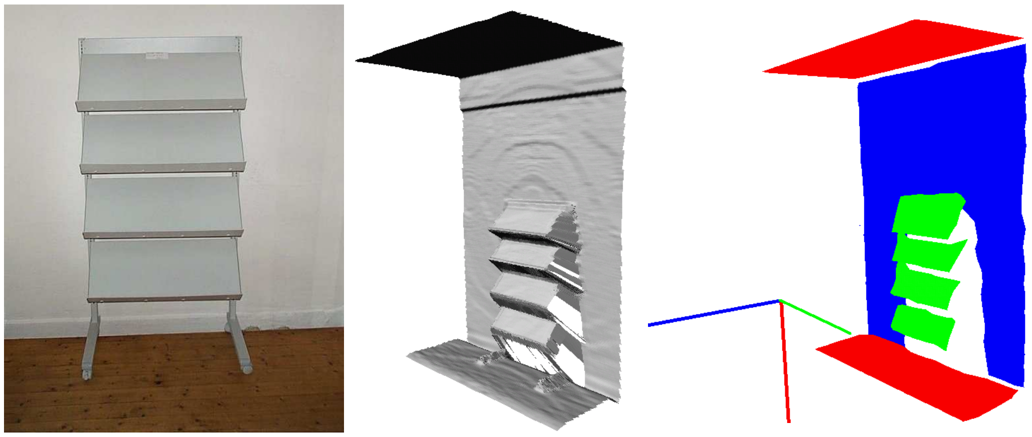
- Determine all scan points that are close to the given plane (in our case: less than $0.1m$).
- Project all these points onto the plane.
- Perform a region growing on the points, where for each point all neighbors in a certain distance ϵ are added to the region. This is done efficiently using a two-dimensional kD-tree (that is, a 2D-tree).
- Create a two-dimensional α-shape (see section 4.5.3) from each region, where $\alpha = \epsilon$.

Experimental Results

The approach described above has been implemented and evaluated on real and simulated 3D data. Figure 5.3c contains the resulting planes obtained for the data set shown in Figure 5.3a. The hierarchical EM is able to exploit the constraints introduced by the main directions of the planes and corrects the planes for the two parallel walls. In this example the angular error between the plane normals decreases from 17.2 degrees to 1.9 degrees.

Experimental Evaluation on Real Data

The real data experiment was carried out with our mobile robot Zora shown in the left image of Figure 5.4. Zora is a B21R platform equipped with a 4DOF AMTEC manipulator which carries a SICK PLS range scanner. This setup allows our robot to flexibly scan complex scenes. The second image of Figure 5.4 shows a picture of a scene scanned with our robot. The third image of this picture depicts a typical data set obtained for this



a. *Photograph of a newspaper rack that was scanned by the system. This scene contains 7 planes, namely the floor, the wall, the ceiling (not shown here) and the shelves of the rack. The shelves are all parallel.*

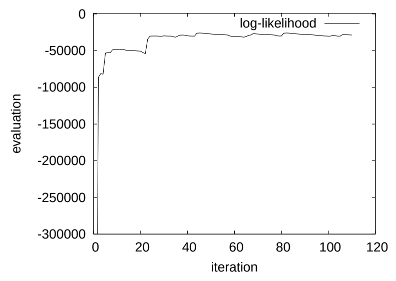
b. *Triangular mesh of the scanned scene. The mesh has been generated from the original point cloud with the local triangulation technique described in section 4.5.1.*

c. *Plane extraction result of the hierarchical EM approach. All 7 planes have been detected correctly, as well as the 3 main directions (here depicted in different colors). Note that the green main direction vector is coplanar with the other two vectors.*

Figure 5.4: *Example of a real world scene and the resulting planes and main directions obtained with the hierarchical EM approach.*

Table 5.1: *Results for the simulated data set. The values are angular deviations from the ground truth in degrees.*

Plane	plain EM			hierarchical EM		
	1	2	3	1	2	3
θ_1	0.220	0.234	0.227	0.155	0.208	0.224
θ_2	0.382	0.415	0.408	0.381	0.415	0.423
θ_3	0.479	0.512	0.492	0.289	0.067	0.190
θ_4	5.879	4.297	2.128	3.023	2.033	1.101
θ_5	0.105	0.070	0.083	1.524	0.318	0.121
θ_6	2.070	0.453	0.321	1.720	0.365	0.181
θ_7	0.672	0.940	0.889	0.244	0.608	0.624
θ_8	1.606	1.915	1.780	0.650	0.883	0.895



a. *Evolution of the log-likelihood during the estimation process.*

Plane	plain EM			hierarchical EM		
	x	y	z	x	y	z
1	92.05	138.33	48.40	91.91	146.02	56.09
2	92.95	146.09	56.26	92.65	146.39	56.52
3	92.27	140.71	50.80	92.01	146.06	56.13
4	90.88	146.56	56.58	91.18	147.32	57.35

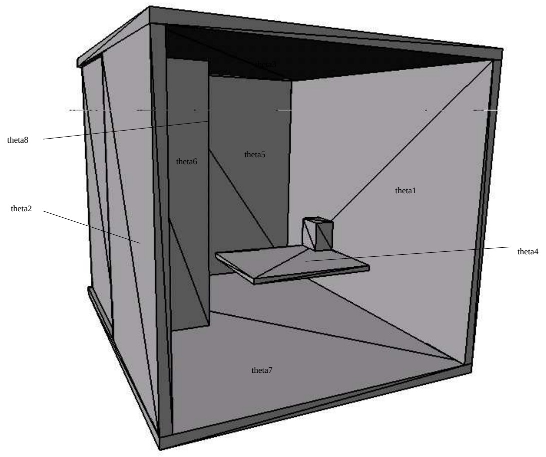
b. *Angles in degrees of the normals for the four planes in the newspaper rack in x, y and z direction.*

Figure 5.5: *Results of the simulated data set*

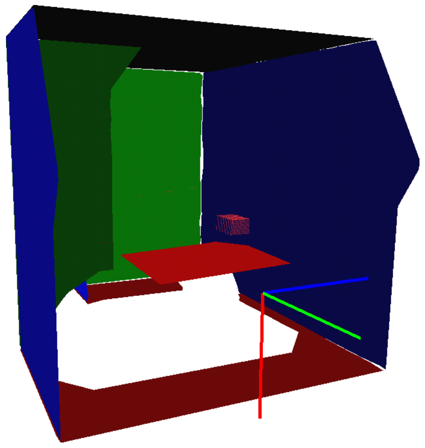
scene. The scan shown there consists of 21.479 points. To enhance visibility the data was smoothed and neighboring scan points were connected by triangles. The rightmost image of Figure 5.4 shows the result obtained with our hierarchical EM. The colors/grey-levels of the individual planes correspond to that of their main directions, which are also visualized. The final model consists of 7 planes and 3 main directions. The planes for the floor and the ceiling are only slightly corrected by the hierarchical EM. Whereas the plain EM approach yields a deviation of 2 degrees for the ceiling and the floor, our algorithm generated planes with an angular distance of 1.7 degrees between the two planes. The most interesting part are the green/light grey planes for the newspaper rack. Note that their main direction is neither orthogonal to the main direction for the ceiling and the floor nor to that of the wall. Table 5.5b lists the individual angles of the normals for the four planes found for the rack in x, y, and z direction obtained with the plain EM approach and with our hierarchical EM. As can be seen from the numbers, our approach reduces the maximum deviation between the individual angles of the planes from 8 degrees to 2 degrees. Figure 5.5a plots the evolution of the log-likelihood during the hierarchical EM. Note that the log-likelihood does not always increase because of the introduction and removal of model components.

Quantitative Evaluation

Additionally, we performed several simulation experiments to evaluate the quality of the resulting models compared to the ground truth. Figure 5.6a shows a simulated scene used for these experiments. This scene represents a room with five corner walls, the floor, and the ceiling. Additionally, it contains a horizontal plane with a box on top. The walls are parallel to the coordinate axes, so that their normal vectors are the standard basis vectors $(1,0,0)^T$, $(0,1,0)^T$, and $(0,0,1)^T$. In total, there were eight visible planes. These planes are enumerated from θ_1 to θ_8. To evaluate the capabilities of our algorithm we performed three different experiments in which we varied the position of the small box on the small horizontal plane. In the first case the box was placed in the right rear corner of the horizontal plane. In the second experiment the box was placed halfway between



a. *Simulated 3d scene used to evaluate the quality of the resulting maps. On top of the horizontal plane is a small cube that introduces errors in the plane extraction.*



b. *Typical result obtained for the situation, in which the box was placed at the right rear corner of the small horizontal plane (see Figure 5.6a)*

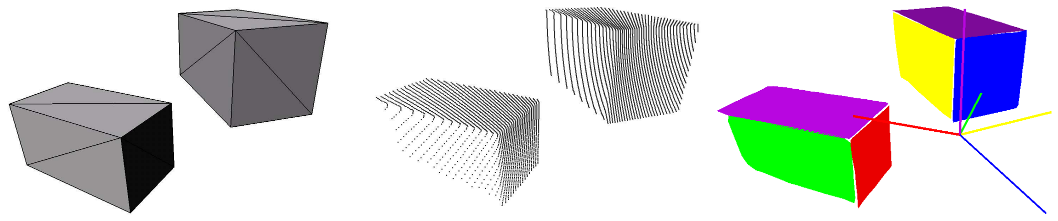
Figure 5.6: *The simulated data set used in the experiments and the polygons resulting from the hierarchical EM approach.*

the center of the horizontal plane and its right rear corner. In the third situation the box was located in the center of the horizontal plane. A typical model obtained with our hierarchical EM applied to the third situation is depicted in Figure 5.6b.

The performance of the plain EM algorithm and our algorithm on these simulated data is given in table 5.1. Each column contains for all three experiments the deviation in degrees for each of the eight normal vectors from its respective ground truth. Especially the table plane θ_4 is corrected by the hierarchical EM. However, other planes like θ_8 are corrected using the knowledge of the main directions. Note that the error in some planes, e.g., the walls θ_1 and θ_2 increases slightly. This is because the plane θ_8, which has the same main direction, has a higher deviation and therefore slightly increases the error of the planes with the same main direction. In the final model obtained with our algorithm all three planes are almost parallel which indicates that the error is introduced by the constraints imposed by the corresponding main direction. The same holds for the planes θ_5 and θ_6.

Models with More Than Three Main Directions

In indoor scenes with mostly orthogonal or perpendicular planar structures such as offices, we only encounter three main directions. To illustrate that our algorithm can deal with more than just three main directions we performed an experiment with the simulated box world shown in Figure 5.7a. The 3d data used as input to our algorithm is depicted in Figure 5.7b. Applied to this data set our algorithm found six planes and five main directions (see Figure 5.7c). In this particular example, the hierarchical EM only slightly corrects the two top planes of the two boxes. All other planes were identical to



a. *Example of a situation with many main directions. The top and bottom faces of both boxes are parallel, all other box faces define new main directions (in total, there are 5).*

b. *Simulated data set obtained by virtually scanning the two boxes from figure a.*

c. *Result of the plane extraction using hierarchical EM. The algorithm found all 6 visible planes and all 5 main directions.*

Figure 5.7: *Simulated scene with more than 3 main directions (fig.a), simulated 3D-scan (b), and resulting planes and main directions (c).*

the planes obtained by the non-hierarchical EM, since there was exactly one plane for each main direction.

5.5 Second Extension: Use of Color Information

In this section, we present a further refinement of the plane extraction algorithm using hierarchical EM. We assume that in addition to the point cloud data we are given a color vector $\mathbf{c}_n$ assigned to each data point $\mathbf{z}_n$. Furthermore, we extend our plane model by a color vector $\bar{\mathbf{c}}_m$. This means that each plane θ_m is represented as a 3-tuple $(\mathbf{n}_m, d_m, \bar{\mathbf{c}}_m)$ consisting of the normal vector $\mathbf{n}_m$, the distance d_m to the origin and the color $\bar{\mathbf{c}}_m$. All three parameters will be estimated during the maximum likelihood estimation process, together with the main directions ϕ_k. Here, ϕ_k is represented as a unit vector in $\mathbb{R}^3$.

In order to calculate the probability that a measurement corresponds to a plane, we first need to define an appropriate distance measure. In our application, where color information is added to planes and scan points, this distance measure consists of a *geometrical* part d_1 and a *visual* part d_2:

$$\begin{aligned} d_1(\mathbf{z}_n, \theta_m) &= \mathbf{s}_n \cdot \mathbf{n}_m - d_m && (5.22)\\ d_2(\mathbf{z}_n, \theta_m) &= \|\mathbf{c}_n - \bar{\mathbf{c}}_m\|. && (5.23) \end{aligned}$$

If we assume that both the geometrical and the visual assignments between scan points and planes underlie a Gaussian error with variances σ_1 and σ_2 respectively, we can write

$$p(\mathbf{z}_n \mid \theta_m) = \eta \exp\left\{-\frac{1}{2}\left(\frac{d_1(\mathbf{z}_n, \theta_m)^2}{\sigma_1^2} + \frac{d_2(\mathbf{z}_n, \theta_m)^2}{\sigma_2^2}\right)\right\}. \quad (5.24)$$

Here, $\eta = (2\pi\sigma_1\sigma_2)^{-1}$ is the normalization factor that stems from the two independent Gaussian distributions. As we have seen in our maximum likelihood estimation process described below this constant has no influence in the maximization step, so that we can neglect it in the following.

If we incorporate the correspondence variables α_{mn}, assuming that they are uniformly distributed, we obtain

$$p(Z, A \mid \Theta) \propto \exp\left\{-\frac{1}{2}\sum_{n}\sum_{m}\alpha_{nm}\left(\frac{d_1(\mathbf{z}_n, \theta_m)^2}{\sigma_2^2} + \frac{d_2(\mathbf{z}_n, \theta_m)^2}{\sigma_2^2}\right)\right\}. \tag{5.25}$$

With this modification, equation (5.17) becomes

$$\begin{aligned}(\Theta^{[i+1]}, \Phi^{[i+1]}) = \operatorname*{argmax}_{(\Theta,\Phi)}\Bigg\{ &-\sum_{nm} E[\alpha_{nm} \mid \Theta^{[i]}]\left(\frac{d_1(\mathbf{z}_n, \theta_m)^2}{\sigma_1^2} + \frac{d_2(\mathbf{z}_n, \theta_m)^2}{\sigma_2^2}\right) \\ &-\sum_{mk} E[\beta_{mk} \mid \Phi^{[i]}]\frac{d_3(\theta_m, \phi_k)^2}{\sigma_3^2}\Bigg\}.\end{aligned} \tag{5.26}$$

Here we used the fact that the expectation operator is linear and that it needs to be computed only over all possible values for the hidden variables α_{nm} and β_{mk}.

5.5.1 Experimental Results

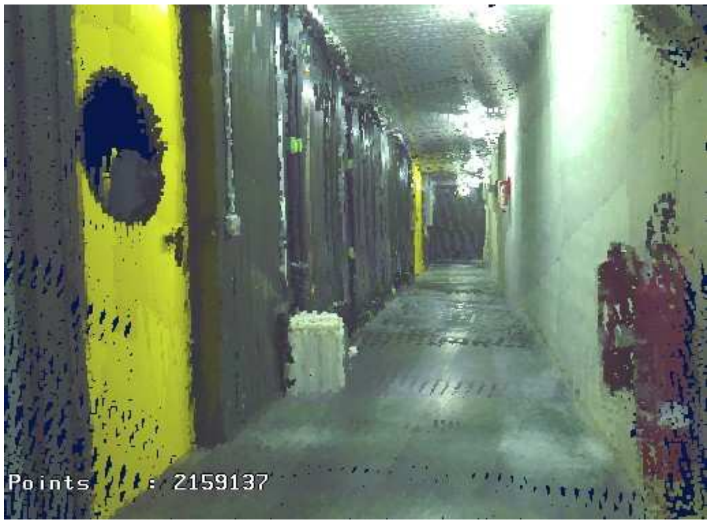
The algorithm described above has been implemented and intensively tested using data collected with our mobile robot Zora (see Figure 5.8a). Zora is a B21r robot equipped with a 4DOF AMTEC manipulator which carries a SICK LMS range scanner. In addition it carries a camera so that we can acquire color information for the individual three-dimensional range data. To carry out the experiments described below, we relied on an accurate calibration obtained with the algorithm presented in Andreasson et al. [2005]. The goal of the experiments described below is to demonstrate that our algorithm can reliably extract planar models from colored range data. We also illustrate that the combination of range and color information yields more accurate results than can be obtained with previous approaches relying solely on range data.

Real World Experiment

The first experiment described in this section has been carried out in a 20m long and 2m wide corridor of building 78 at the University of Freiburg. This corridor contains four doors, from which one is blue and the other three are yellow. The data set consists of 72 scans, which were taken at eight different positions each approximately 2m apart. The need for taking 9 scans at each position is due to the limited opening angle of the camera. To register the individual scans we applied the iterative closest point algorithm (ICP) presented by Besl and McKay [1992]. After the alignment we only stored those 3D points for which color information was available. We furthermore sub-sampled the model to reduce its complexity.

Figure 5.8b shows a three-dimensional visualization of the corresponding data. In this figure the view-point of the camera lies inside the corridor. Two virtual views from

a. *Robot in a corridor*



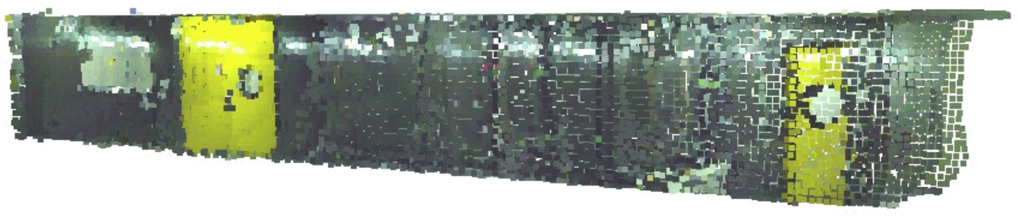
b. *Data recorded with a mobile robot in a corridor environment at the CS Department in Freiburg. The whole data set, which consists of 2,159,137 points has been registered using the ICP algorithm.*

Figure 5.8: *Robot in corridor (a) and 3D data recorded in a hallway (b).*

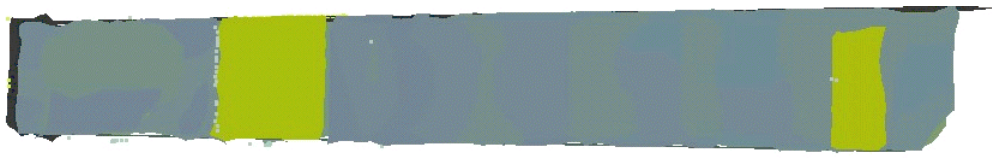
the outside are depicted in Figure 5.9. Note that the resulting data poses high challenges for color-based plane extraction. Many surfaces in this corridor are rather shiny, so that many data points have wrong colors. For example, there were several reflections of doors and lights in data points belonging to the floor, the ceiling, and the walls.

Figure 5.9 depicts the model obtained with our color-based plane extraction algorithm. Shown are two side-views corresponding to the two views shown in Figure 5.9. As can be seen from the figure the four doors, which have an indentation of 6, 14, 23, and 25cm from the four walls, have correctly been extracted from the range data. Note that this model shows yellow planes close to the blue door. These planes have been added due to the reflections of the yellow door on the other side of this corridor. Furthermore, our current algorithm to extract polygons from the data cannot handle holes in surfaces so that the round windows in the doors are not visible in the resulting model.

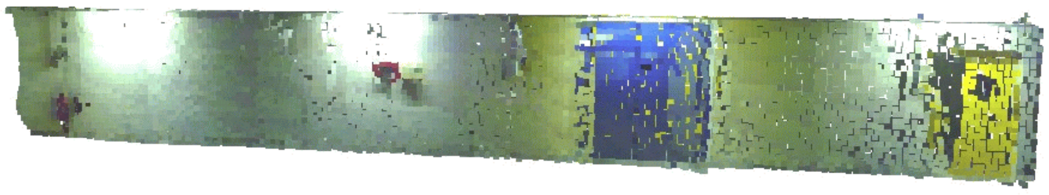
We additionally carried out a series of experiments with a plane clustering algorithm that does not utilize the color information. Thereby we tried to identify the optimal value for the geometric variance parameter σ_1 so that EM learns a model with individual planes for all four doors. In all our experiments we could not find such a value. Whereas too small values for σ_1 result in a huge number of planes, whereas larger values for σ_1 yield models with appropriately many planes but typically with one or even more doors missing.



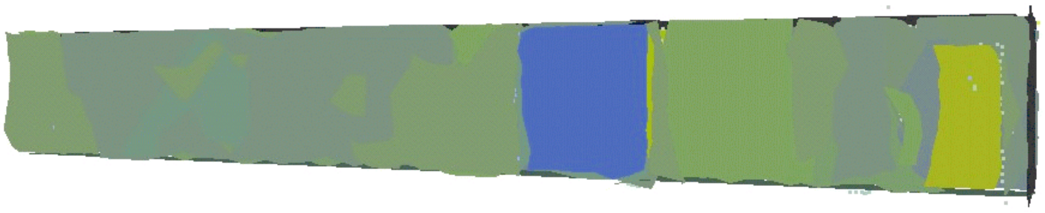
a. *Point model, front side*



b. *Plane model, front side*



c. *Point model, back side*



d. *Plane model, back side*

Figure 5.9: *Front and back-side view of the corridor data shown in Figure 5.8b.*

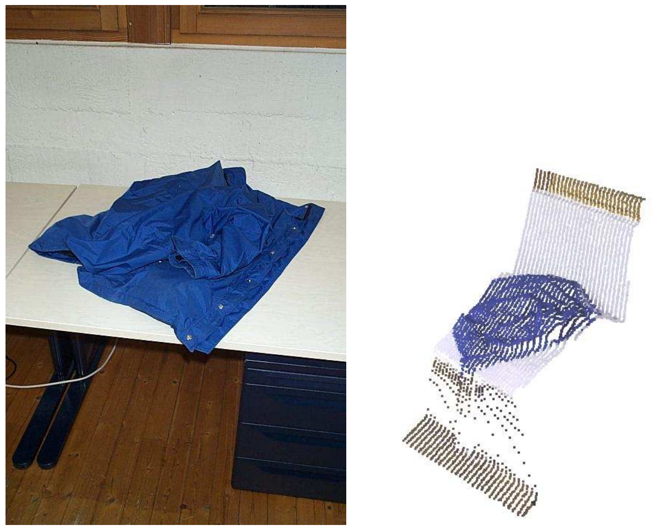


Figure 5.10: *A blue jacket lying on a desk. Photograph of the scene (left) and 3D scan with color information added. The example is used to evaluate the color-based plane extraction. Comparing the plane extracted for the table top, we obtain an improved estimate over the non-color based plane extraction (see text).*

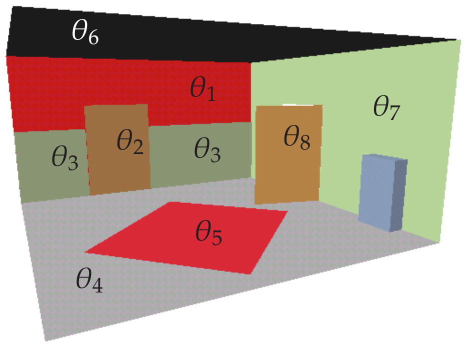
Quantitative Comparison to Non-Color-Based Clustering

The second experiment is designed to illustrate that our color-based plane clustering approach yields more accurate results than an approach relying on range data only. In this particular experiment we placed a blue jacket on top of a desk and generated one scan with Zora. The corresponding situation in combination with the acquired scan is depicted in Figure 5.10. Additionally we scanned the table without the jacket from which we determined an accurate estimate of the parameters of the table top. The errors in the height of the plane reduces from 5.9cm for the non-color-based approach to 1.3cm. Simultaneously our algorithm reduces the angular deviation from 4.4 degrees to 3.1 degrees.

Simulation Experiments

To evaluate the performance of our algorithm with respect to a known ground truth, we created an artificial 3d environment which allows us to simulate colored three-dimensional range scans. The scene used for the experiment described here is depicted in Figure 5.11a. It represents a room with several planar structures of different colors, sizes, and orientations. The planar structures used for the quantitative evaluation are also labeled in this figure.

Table 5.11b shows the angular deviations of the plane normals from the ground truth for eight different planes. Note that the planes θ_2, θ_3, and θ_5 can not be found without color information. For all other planes the estimates obtained with our color-based approach were closer to the ground truth. Without the color information the



a. *Artificial environment used for simulation experiments.*

Plane	non-color based	color based
θ_1	1.97	0.25
θ_2	—	0.54
θ_3	—	0.49
θ_4	0.13	0.11
θ_5	—	0.12
θ_6	0.15	0.08
θ_7	0.28	0.25
θ_8	0.22	0.06

b. *Angular deviations from the ground truth in degrees*

Figure 5.11: *Simulation results for texture based plane extraction.*

EM cannot distinguish between data points belonging to different objects. For example, the surface of the carpet lies slightly above the floor. The standard approach cannot distinguish between the two point sets and usually clusters them into a single plane. Our color-based algorithm, however, generates two different planes with more accurate parameters. Whereas the height of the plane for the floor deviates from the ground truth only by 2mm, the error in the height of the plane corresponding to the carpet is zero. The non-color-based version, in contrast, generates only one plane with a distance of 5mm to both objects. Here the height difference was determined as the distance of the plane to the center of the rectangle corresponding to the floor.

5.6 Conclusions

In this chapter, we discussed several different techniques to find planar structures in three dimensional range data. We showed that standard techniques such as region growing do not reflect the statistical nature of the data and we introduced a new probabilistic approach based on a hierarchical Bayes net. We further improved this approach by using texture information obtained from camera images. As a result, we obtained a compact 3D data representation that is particularly useful for environments with many parallel planes. Thus, it can be used not only indoors but also outdoors in urban environments where the walls of buildings are mostly parallel.

In preparing for battle I have always found that plans are useless, but planning is indispensable.

Dwight D. Eisenhower (1890 - 1969)

6

Efficient Map Representations for Localization and Planning

6.1 Introduction

In chapter 4 we analyzed the most common approaches to represent three-dimensional data. In particular, we discussed point clouds, 3D tree structures, 3D occupancy grids and 3D triangulations. As we have seen, all these data structures have advantages and drawbacks with respect to the required memory, the time required to search for a given data point, and the possibility to visualize the underlying data so that it appears as realistic as possible. However, none of these data structures was designed for one special purpose, but rather for many kinds of applications. In this chapter, we will instead present a new kind of 3D data representation, which we will call multi-level surface map, that is particularly useful for the tasks of localization and planning for a mobile robot. These two tasks are crucial for any mobile robot and therefore it is very important to find an efficient data structure that is especially suited for these tasks. In addition to this design goal, we will show that the new data structure is also efficient with respect to the before mentioned aspects of memory requirement, search efficiency and the ability to visualize the 3D data.

The chapter is organized as follows. In Section 6.2 we will briefly summarize the most important requirements for the localization and planning tasks. These will be the primary design goals for the new data structure. Then we present two different state-of-the-art approaches to accomplish these tasks, where the first one, presented in Section 6.3, is referred to as elevation maps and is the most common one. The second one is an extension of the elevation maps and is presented in Section 6.4. Then, in Section 6.5, we introduce the new concept of multi-level surface (MLS) maps, which can also be seen as an extension to the framework of elevation maps, with the additional feature that multiple height levels can be represented at any given xy position in the 2.5D grid representation. Finally, Section 6.6 concludes the chapter.

6.2 Requirements for Localization and Planning

As mentioned already, special capabilities are required for the tasks of localization and planning. In this section, we will name three major aspects for each task. Later we will see how these issues are addressed in the different 3D data representations.

6.2.1 Localization

The following aspects are particularly relevant when designing a map representation that is used to do localization of a mobile robot:

- **Accuracy:** For a good localization the map must be as accurate as possible because the observations taken at each robot position in the real world must be compared to the map. Accuracy also includes that special features of the environment such as street lamps, trees, etc. should be mapped as similar as possible to their appearance in the real world, because these features are very helpful in resolving ambiguities and, therefore, to improve the localization.
- **Compactness:** Mobile robot localization is a task that needs to run in real time, because the robot needs to know its position at each time frame while it is traveling through the environment. A map representation that requires a huge amount of memory, like point clouds, is therefore not appropriate, because it would take too much time to search the map for the current position.
- **Level of Abstraction:** The biggest problem of mobile robot localization is that at the time when the robot localizes itself in the map, it (nearly) never observes the environment from the same point of view as when it has created the map. This means that it will never find the exact same view of the environment in the set of views that were used to create the map. Therefore, a good map representation needs to abstract from the raw sensor input consisting of single measurements to higher level representations like triangles or polygons.

6.2.2 Planning

In order to perform the planning task efficiently, a number of specialized properties must be provided by the map representation. These depend strongly on the planning algorithm that is used. However, most planning algorithms such as A^*, D^* [Koenig and Likhachev, 2002, Stentz, 1995], Field D^* [Ferguson and Stentz, 2005], or probabilistic roadmaps [Kavraki et al., 1996] require at least a memory efficient representation of the state space and a fast way to perform a lookup of neighboring states in the state space. Thus, the requirements with respect to the planning task can be described as follows:

- **Discretization:** Most planning algorithms discretize the state space of the robot. For two-dimensional environments, this is easily accomplished by using occupancy grids. However, as we have seen in section 4.4, 3D occupancy grids have huge memory requirements and can not be used in large-scale environments, especially outdoors. Therefore, a discretization strategy that uses less memory is needed.

- **Neighborhood computation:** One of the most frequent operations that are performed in state-based planning algorithms such as A^* is the computation of all neighboring states for a given state. This can be most efficiently done when using grid-based map representations. Again, the 3D occupancy grid would be the best choice here, but because of its memory complexity it is not useful.

- **Cost computation:** When planning a path through the environment, it is necessary to evaluate the cost of this path. These cost usually depend on the length of the path and on the 'danger' that is caused by traveling over different kinds of terrain. Therefore, a local path cost value must be stored at each position in the map. The computation of this cost value must be efficient and it should reflect the level of danger as accurate as possible.

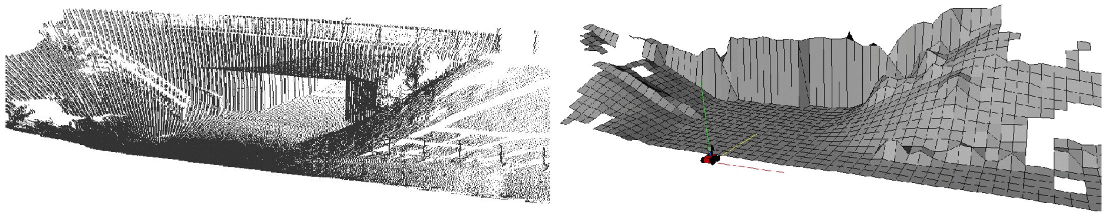
In the past, there has been mainly one approach that fulfills most of the presented requirements. This approach uses so called *elevation maps*, sometimes also referred to as *digital elevation maps*. In the following, we will describe this data structure, as it constitutes the basis of our new 3D map representation.

6.3 Elevation Maps

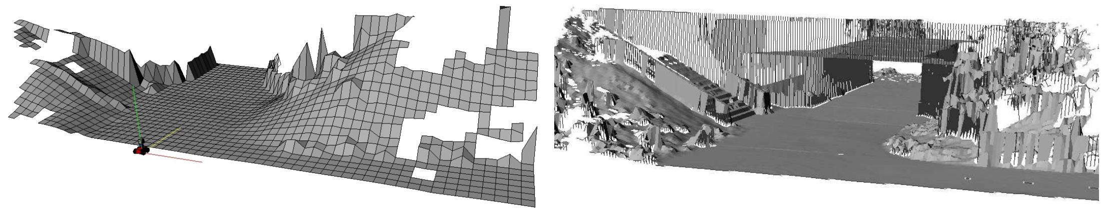
This section presents the main concepts underlying the framework of elevation maps and presents a list of previous works in which elevation maps have been applied successfully.

6.3.1 Main Concepts

The key idea underlying elevation maps is to store the height information of the environment in a $2\frac{1}{2}$-dimensional grid structure. This means that an elevation map consists of a two-dimensional grid that discretizes the ground plane on which the robot moves into equally spaced intervals. Then, at each grid cell a height value is stored that corresponds to the *elevation* of the terrain at that particular position. In addition to the height value, other information can be stored in each grid cell. Examples include texture information, geometric information such as normal vectors, and different measures of planarity of the environment. A very common approach is to additionally represent the uncertainty of the height measurements. This can be achieved by storing a mean value μ and a variance σ for the height at each grid cell. This way, the reliability of the 3D sensor and the number of measurements per cell can be incorporated into the 3D map representation. A standard way to compute μ and σ from the sensor data is by means of the update rule in the Kalman filter. We will discuss this in detail in Sec. 6.4.



a. *Point cloud* **b.** *Elevation map*



c. *Extended elevation map* **d.** *Multi level surface map*

Figure 6.1: *Scan (point cloud) of a bridge (fig. a), standard elevation map computed from this data set (fig. b), extended elevation map which correctly represents the underpass under the bridge (fig. c), and multi level surface map that correctly represents the height of the vertical objects (fig. d).*

Two conceptually different methods of constructing elevation maps from 3D sensor data such as point clouds or range images are possible. We will call these the *forward insertion* and the *backward interpolation*. The forward insertion algorithm can be applied either to range image data, where each pixel represents a depth value at a given discrete horizontal and vertical angle, or to arbitrary point clouds where this information is not available. It computes for each 3D data point the discrete xy-position of the corresponding grid cell in the map and assigns the z-value of the data point to the grid cell. Alternatively, the Kalman filter update rule can be applied to account for the uncertainty in the z-value. The backward interpolation algorithm proceeds in the other direction. It iterates over all discrete cells in the elevation map and finds the corresponding height value by interpolating in the range image. It can not be applied for arbitrary point clouds, but has the advantage that the resolution of the elevation map can be chosen arbitrarily. A detailed description of this algorithm has been described by Hebert et al. [1989].

An example of an elevation map is shown in Figure 6.1b. It has been computed from the point cloud data shown in Figure 6.1a. As we can see, the elevation map represents the environment in a more natural and more general way, because each cell reflects a planar patch of the surface. In addition, it is possible to do path planning for a mobile robot using the map cells as possible discrete states of the robot. However, the particular data set shown in the example also illustrates the main drawback of standard elevation maps, namely the fact that overhanging objects and underpasses can not be represented correctly. As a result, the robot is unable to plan a path that passes under the bridge as shown in the example, because the bridge appears as a wall in the map. This problem can be solved by using *extended elevation maps*, which will be described in detail in Section 6.4.

6.3.2 Historical Background of Elevation Maps

In the past, elevation maps have been used by many different authors for different applications. For example, Bares et al. [1989] as well as Hebert et al. [1989] use elevation maps to represent the environment of a legged robot. They extract points with high surface curvatures and match these features to align maps constructed from consecutive range scans. Parra et al. [1999] represent the ground floor by elevation maps and use stereo vision to detect and track objects on the floor. Singh and Kelly [1996] extract elevation maps from laser range data and use these maps for navigating an all-terrain vehicle. Ye and Borenstein [1994] propose an algorithm to acquire elevation maps with a moving vehicle carrying a tilted laser range scanner. They propose special filtering algorithms to eliminate measurement errors or noise resulting from the scanner and the motions of the vehicle. Lacroix et al. [2002] extract elevation maps from stereo images. Hygounenc et al. [2004] construct elevation maps with an autonomous blimp using 3d stereo vision. They propose an algorithm to track landmarks and to match local elevation maps using these landmarks. Olson [2000] describes a probabilistic localization algorithm for a planetary rover that uses elevation maps for terrain modeling. However, all these approaches can not cope with the problem of overhanging objects and underpasses, as mentioned in the previous section. Therefore, we introduce extended elevation maps, that have been proposed by Pfaff and Burgard [2005], in the next section.

6.4 Extended Elevation Maps

The way in which extended elevation maps address the problem of overhanging objects and bridges is to identify cells in the map where such problems occur. Then, a more detailed analysis of the measurements in those cells is performed and a more plausible height value is stored in the cell instead of the highest elevation. We will describe this in Section 6.4.2 in more detail. Before, we show how the Kalman update rule is applied to compute the elevations in the map while maintaining an estimate of the measurement uncertainty.

6.4.1 Measurement Update

Our goal is to compute a height value at each grid cell that can be updated by incorporating new height measurements and at the same time accounts for the uncertainty in the measurement. A standard way to do this is by means of the update rule known from Kalman filters. Assume we have a height measurement z_t for a given map cell at time t, along with a variance σ_t reflecting the uncertainty. Assume further that in the map cell we already incorporated the height values and variances in time steps 1 through $t-1$. The resulting mean and variance will be denoted $\mu_{1:t-1}$ and $\sigma_{1:t-1}$ respectively. Then, we compute the new height value $\mu_{1:t}$ and its variance $\sigma_{1:t}$ according to the following

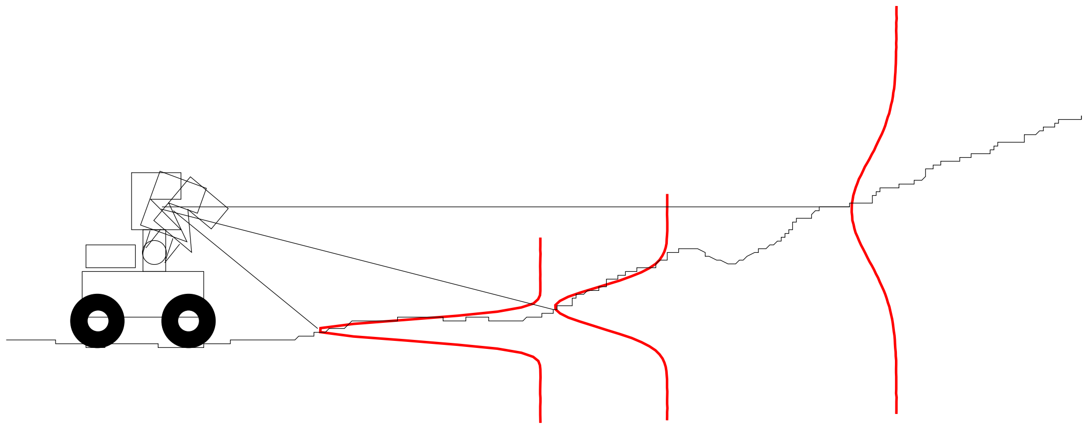


Figure 6.2: *Variance of the height measurements depending on the distance of the beam. The measurement uncertainty is assumed to underly an isotropic Gaussian distribution, which means that the uncertainty in height is equal to the one in range.*

equations (see Maybeck [1990]):

$$\mu_{1:t} = \frac{\sigma_t^2 \mu_{1:t-1} + \sigma_{1:t-1}^2 z_t}{\sigma_{1:t-1}^2 + \sigma_t^2} \tag{6.1}$$

$$\sigma_{1:t}^2 = \frac{\sigma_{1:t-1}^2 \sigma_t^2}{\sigma_{1:t-1}^2 + \sigma_t^2} \tag{6.2}$$

The measurement noise, which is represented by the variance σ_t, is given by the *sensor model*. In our case, where the sensor is a 3D laser range finder, we apply a simplified sensor model by assuming isotropic Gaussian noise that depends linearly on the measured distance. This means that the noise variance increases with the measured range and that the height variance σ_t is equal to this noise variance. Figure 6.2 illustrates this idea. Although this approach is an approximation, there was never evidence from practical experiments that it causes any noticeable errors.

6.4.2 Cell Analysis

As mentioned above, extended elevation maps perform an analysis of the individual cells to allow a more accurate representation of bridges and overhanging objects. This analysis is done by classifying the cells into four possible classes:

- **horizontal and traversable cells:** These represent regions that have been scanned from above and that are planar enough for the robot to pass. Planarity can be measured by fitting planes in the vicinity of the cell and thresholding the local height variance.

- **horizontal and non-traversable cells:** These are horizontal cells where the planarity criterion is not met. The idea of these cells is to assign a high cost in the path planning

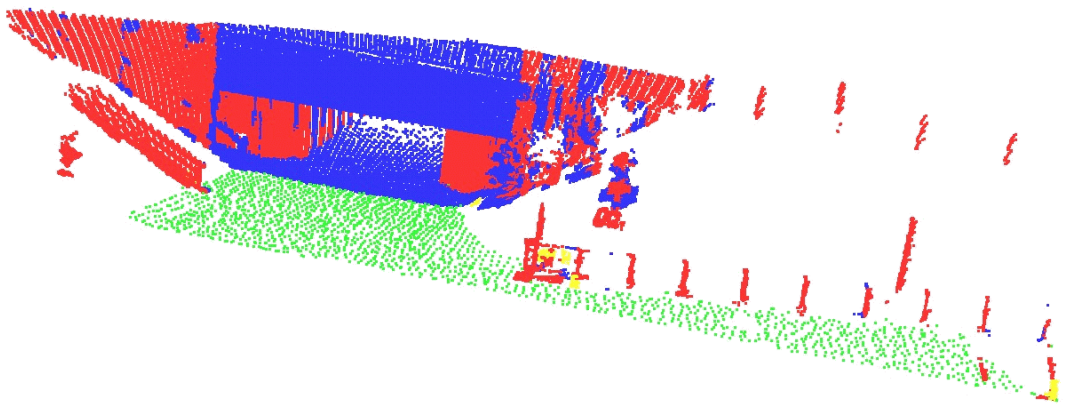


Figure 6.3: *Labeling of the data points depicted in Figure 6.1a according to their classification. The four different classes are indicated by different colors. green is traversable, yellow is non-traversable, red is vertical and blue are gap cells.*

procedure to regions that are dangerous or impossible for the robot to traverse.

- **vertical cells:** These are identified by computing vertical intervals from all measurements that correspond to a cell. If there is only one such interval or if the distance between two consecutive intervals is so small that the robot can not pass the cell, it is identified as a vertical object (e.g. a wall).

- **gap cells:** These cells account for all cases where multiple intervals have been found in the height values of the measurement data and where the vertical intervals are far enough apart from each other. For gap cells, the minimum traversable elevation is used as height value.

An example of this classification is shown in Figure 6.3. The input data was the same point cloud as shown in Figure 6.1. The visualization shows for each classified grid cell all 3D data points that correspond to the cell. The four classes are distinguished with colors: traversable cells are green, non-traversable cells yellow, vertical cells appear in red and gap cells are blue. The extended elevation map computed from this data set is shown in Figure 6.1c, in which we only plot the height values for the lowest interval in each cell. As a result, the area under the bridge now appears as a traversable surface. This allows the robot to plan a safe path through the underpass. However, as we can see from the figure, the extended elevation map can only provide means to compute a path *either* under the bridge *or* crossing over it, but not both. This is because there is still only one height value stored at each grid cell. Therefore, we present in the next section a new data structure that addresses this problem. We will call this a *multi level surface map*.

6.5 Multi Level Surface Maps

As we have seen in the motivating example shown in Figure 6.1, multi level surface maps, or MLS maps for short, aim at representing environments in which the robot can traverse the same xy position at different height levels, e.g. bridges. The way this is done in MLS maps is by storing multiple height values along with their variances at each grid cell. In addition, the main concepts of extended elevation maps are also implemented in MLS maps. In particular, the computation of vertical intervals and the Kalman update rule for maintaining the measurement variance are applied in analogy to extended elevation maps. The details are described in Section 6.5.2. Before, in Section 6.5.1, we show how the 3D data is represented in MLS maps. Section 6.5.3 describes how MLS maps are updated. In Section 6.5.4 we deal with the extraction of features from MLS maps and Section 6.5.5 shows how MLS maps can be annotated. Finally, in Section 6.5.6 we present a method to visualize MLS maps.

For the remainder of this section, we assume we are given a set of N 3D scan points $C = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ with $\mathbf{p}_i \in \mathbb{R}^3$, and a set of variances $\{\sigma_1^2, \ldots, \sigma_N^2\}$. As before, the variance σ_i^2 expresses the isotropic measurement uncertainty that grows with the measured distance. Furthermore, we define a measurement z as a pair $(\mathbf{p}, \sigma^2)$ of a 3D point and a variance.

6.5.1 Map Representation

A multi level surface map (MLS map) consists of a 2D grid of variable size where each cell c_{ij}, $i, j \in \mathbb{Z}$ in the grid stores a list of *surface patches* $P_{ij}^1, \ldots, P_{ij}^K$. A surface patch in this context is represented as the mean μ_{ij}^k and variance σ_{ij}^k of the measured heights at the position of the cell c_{ij} in the map. Each surface patch in a cell reflects the possibility of traversing the 3D environment at the height given by the mean μ_{ij}^k, while the uncertainty of this height is represented by the variance σ_{ij}^k.

In addition to the mean and variance of a surface patch, we also store a *depth value* d for each patch. This depth value reflects the fact that a surface patch can be on top of a vertical object like a building, bridge or ramp. In these cases, the depth is defined by the difference of the height h_{ij}^k of the surface patch and the height ${h'}_{ij}^k$ of the lowest measurement that is considered to belong to the vertical object. For flat objects like the floor, the depth is 0. Figure 6.4 depicts some examples of the map cells in an MLS map.

6.5.2 Map Creation

An MLS map can be generated in two different ways: either from a set of 3D measurements, i.e. a point cloud with variances, or by joining two other MLS maps into one. Both ways are equivalent, i.e. if map m_1 is created from point cloud C_1 and map m_2 from cloud C_2, then the map m_3 that results from joining m_1 and m_2 is identical to the map generated by the joined point cloud $C_3 = C_1 \cup C_2$. For a given point cloud C with variances $\sigma_1, \ldots, \sigma_n$ the MLS map is created as follows:

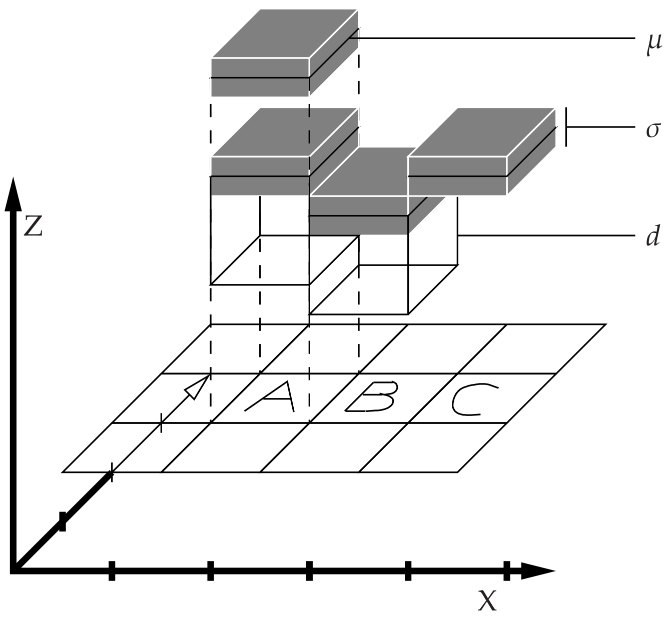


Figure 6.4: *Example of different cells in an MLS Map. Cells can have many surface patches (cell A), represented by the mean and the variance of the measured height. Each surface patch can have a depth, like the patch in cell B. Flat objects are represented by patches with depth* 0*, as shown by the patch in cell C.*

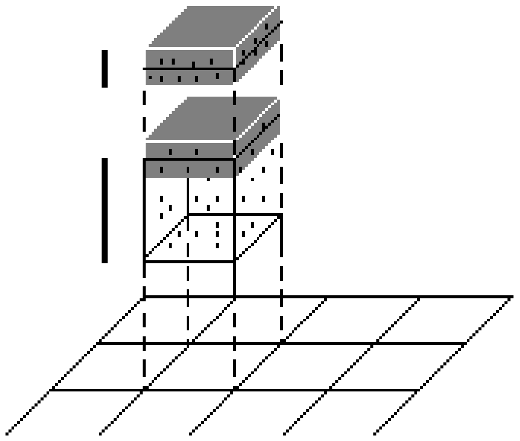
- Each map cell with index (i, j) collects all points $\mathbf{p} = (x, y, z)$, s.t. $si \leq x \leq s(i + 1)$ and $sj \leq y \leq s(j + 1)$ where s denotes the size (edge length) of a map cell.

- In each cell, we calculate a set of *height intervals* from the height values of the stored points. As long as two consecutive height values are closer than a given *gap size* γ, they belong to the same interval. This means that two intervals are at least γ meters away from each other. The gap size should be chosen so that a robot that navigates through the map can still pass the gap, i.e., it should be higher than the robot height. In our implementation we choose $1.0m$.

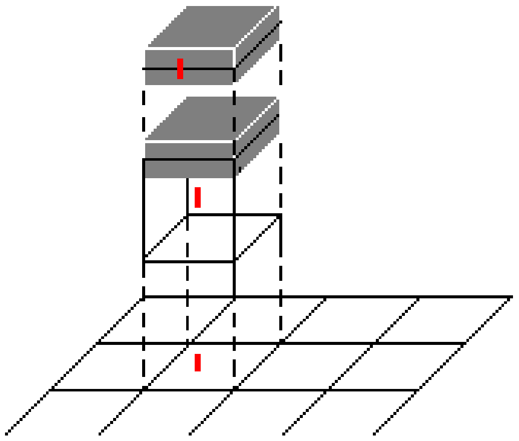
- The intervals are classified as a *horizontal* or a *vertical* structure. These structures are distinguished according to the height of the interval. If it exceeds a *thickness value* $\tau = 10$cm, it is considered as vertical, otherwise it is horizontal.

- For each interval classified as vertical, we store the mean and variance of the highest measurement in the interval. The intuition behind this is that for traversability only the highest measurement is relevant. Additionally, we store the length of the interval, which is identified with the depth d mentioned above. This value is used when matching two MLS maps together.

- For each horizontal object in a cell, we compute a mean μ and a variance σ from all measurements in the interval. This is done by applying the Kalman update rule to



a. *For a given xy-cell, all 3D data points are collected that fall into this cell. Then a set of height intervals is computed for the cell. The intervals are then classified into 'horizontal' and 'vertical'.*



b. *A new measurement (here depicted as a red line) can be either part of a horizontal object, e.g. a surface, a vertical object, or it can correspond to a new object that has not been mapped.*

Figure 6.5: *Creating and updating an MLS map*

all measurements. The depth d of a horizontal object is set to 0.

Figure 6.5a demonstrates the map creation from point clouds. In this case, two vertical intervals have been determined, where the upper one is classified as 'horizontal' and the lower one as 'vertical'. As we can see, the horizontal surface patch that was created has no depth value.

After computing the means, variances and depths of the surface patches, we delete the point cloud data. All further calculations are performed only on the map data. This substantially reduces the memory required for an MLS map compared to point clouds and at the same time achieves a highly accurate representation.

6.5.3 Map Update

Whenever a new measurement $z = (\mathbf{p}, \sigma)$ is inserted into an MLS map, we first need to know whether the measurement belongs to an object that is already represented in the map, or if it corresponds to a new object. To this end, we first determine the cell c_{ij} in which the measured point falls. Then we find the surface patch $P_{ij}^k = (\mu_{ij}^k, \sigma_{ij}^k)$ in c_{ij} whose mean μ_{ij}^k is closest to the height of z. If this patch is close enough, we update it with the new measurement z, again using the Kalman update rule. In our implementation, we define a surface patch to be close to a measurement z if the height value of z is within 3σ of the patch.

If z is far from the nearest patch, it is still possible that it corresponds to a vertical object. This can be found out by checking whether z is inside the occupancy of one of the surface patches in the cell. In this case, the measurement z is simply disregarded, because a vertical object with an assigned patch already exists. Otherwise, z is introduced as a new surface patch into the cell.

As an illustration, figure 6.5b shows the three different possibilities for a new measurement z. In the figure, the measurement is represented as a vertical line, where the line length represents the variance in height. In the example case, the upper measurement will be identified with the horizontal surface patch and the patch height and variance will be updated accordingly. The new measurement shown in the middle is assigned to a vertical object, but not to its surface. Therefore it is simply disregarded. The lowest new measurement, however is regarded as a new object and will cause the creation of a new surface patch.

6.5.4 Feature extraction

In chapter 7 we will see that it is often required and convenient to reduce the information that is stored in a 3D map by extracting features from the map. This is for example the case when several partial maps are created from the same environment and have to be matched to each other to form a global map. In these situations, it is more efficient to perform the computations on a small set of features extracted from the maps instead of using the entire map representation. The features we will use for this purpose consist of 3D points that are computed from surface patches P_{ij}^k in the MLS map. The x and y-coordinates of the features are defined by the indices i and j of the patch, multiplied with the cell size. The coordinate z of the feature is defined by the mean height μ_{ij}^k of the patch. Thus, the feature set is a point cloud where the x and y coordinates are distributed on a regular grid.

The reduction of the feature set is performed by sampling from the feature point cloud. This can be done using a fixed number of samples or adapted to the size of the map. Also, one could think of a weighted sampling where the weight is defined by the inverse measurement variance that corresponds to each feature. This would guarantee that at regions in the map where the measurements are more reliable, a feature is sampled with a higher probability. For the purpose of matching local MLS maps together, which we will describe in the next chapter, it is sufficient to use a simple fixed-size uniform sampling strategy.

A further improvement of the feature extraction can be achieved by classifying the surface patches of the MLS map before computing the feature points. The idea behind this is the same as for the labeling technique used for the cells of extended elevation maps. The difference here is that no gaps may occur, because we label surface patches and not the entire cell. The other three classes, namely 'horizontal and traversable', 'horizontal and not traversable' and 'vertical' in contrast are still meaningful here and may be used for tasks like path planning or local map matching. The way in which surface patches are classified is somewhat easier than the labeling of cells in extended elevation maps, because patches with a non-zero depth value are already considered as vertical patches. The distinction of the horizontal patches into traversable an non-traversable is again performed by fitting planes in the vicinity of the patches and thresholding the height variance. As a result of the surface patch labeling, we obtain three different point clouds from the feature extraction process, one for each class of surface patches.

6.5.5 Map Annotation

In many situations, it is convenient to store additional semantic information together with the map representation. This can be any kind of information. For example, when classifying the surface patches as described above, we can add an index to each surface patch that is identified with one of the classes ’vertical’, ’horizontal and traversable’ and ’horizontal and not traversable’. For this reason, each surface patch stores an integer value a in addition to μ, σ and d. This value will be referred to as the *annotation* and can be interpreted in several different ways. Here, we name only three possible applications of map annotation, which seem particularly useful:

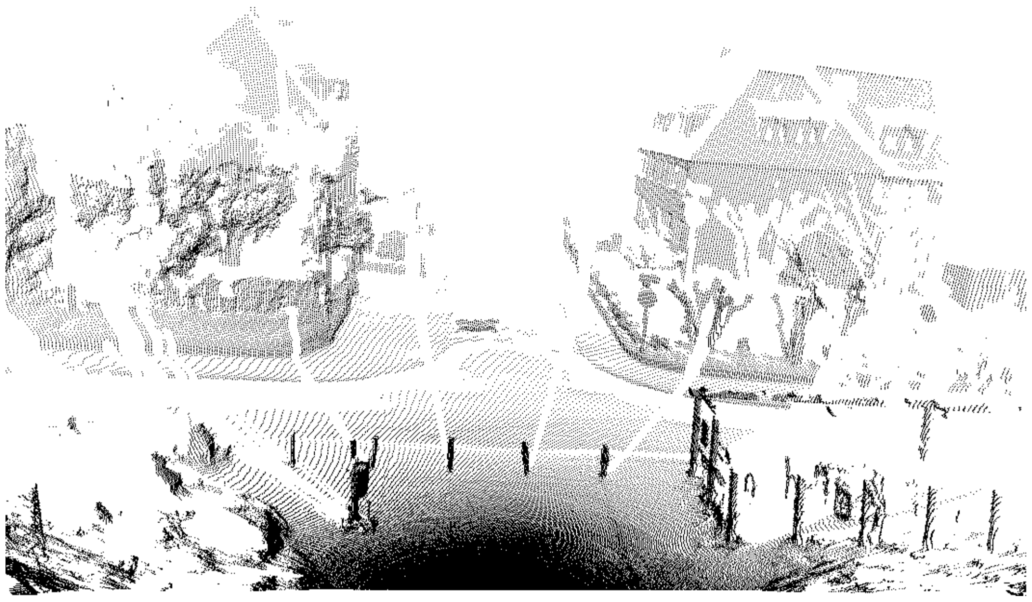
- **traversability:** the integer value a is interpreted as an index of several different levels of traversability.
- **object classification:** a classification algorithm detects the class of the object of which the surface patch is part of. Examples of possible classification algorithms will be presented in chapter 9.
- **texture:** the value a is interpreted as an index into a color lookup table. This lookup table is computed from the texture information available for the given surface patch.

For the MLS maps that are shown in the remainder of this thesis we will use the traversability scheme to do the map annotation. In the simplest case of two traversability levels this can be implemented by distinguishing between horizontal and vertical surface patches.

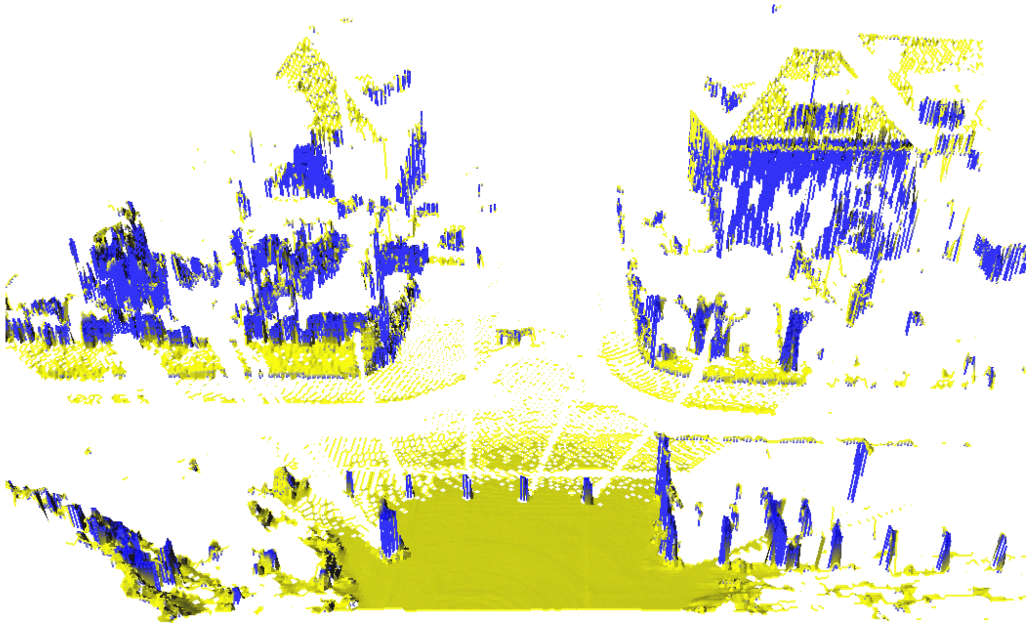
6.5.6 Visualization

As mentioned in Section 4.5, it is often important to find a 3D representation that is based on polygons rather than on points. A polygon-based representation improves not only the visual aspect of the underlying data, it also facilitates the task of texture mapping onto the 3D structure whenever such texture information is available. Therefore, we are interested in a method to triangulate the 3D data stored by an MLS map. One possible way to do this is to assign to each surface patch a square that is parallel to the xy ground plane and has a side length equal to the MLS map’s cell size. This has the advantage that each surface patch can be visualized independently and that it is comparably easy to implement. However, it gives a rather unrealistic result, because all polygons (i.e. squares) are parallel and the height difference of the surface patches gets visible in the form of a step field.

Therefore, we apply a more elaborate technique to visualize MLS maps. The idea here is to use the 3D data points that correspond to the surface patches as *corners* in the mesh representation. These corners are then connected to quadrangles, triangles or lines, depending on the number of neighboring surface patches that exist for a given patch. A



a. *Point Cloud*



b. *MLS map*

Figure 6.6: *Visualization of MLS maps. Horizontal surface patches are shown in yellow and vertical patches in blue. To reflect the vertical structures, all vertical patches are represented as line segments.*

neighboring patch is here defined as one that is part of an adjacent cell and close enough to the given surface patch. Again, we use a threshold of 3 times the standard deviation in height to define which patches of an adjacent map cell are close (see Section 6.5.3). The overall algorithm to compute a surface mesh from an MLS map can be sketched as follows:

- Compute a 3D data point from each surface patch in the map. The obtained 3D points are the same as the feature points described in Section 6.5.4.
- Maintain a lookup table $\mathcal{L}$ of data points. In this table, each 3D data point is represented at most once and an index to it is stored. The index is needed to define lines, triangles and quadrangles.
- Maintain a point cloud C_I of isolated points.
- For all cells c_{ij} in the map:
  - For all surface patches P_{ij}^k:
    - ∗ Determine the neighboring patches of P_{ij}^k from the cells $c_{i+1,j}$ $c_{i,j+1}$ and $c_{i+1,j+1}$.
    - ∗ If there are neighbors:
      - · Insert the neighbor points into the lookup table $\mathcal{L}$ if not yet done.
      - · Connect the corresponding 3D points to a line, a triangle or a quadrangle, depending on the number of neighbors. This is done by retrieving the point indices from $\mathcal{L}$.
    - ∗ Otherwise, add the 3d point of P_{ij}^k to C_I.

As a result, we obtain a set of lines, triangles and quadrangles, as well as a point cloud C_I of isolated points. In addition, all vertical surface patches are visualized as vertical line segments that are as long as the depth of the vertical patch. An example of a surface mesh that was computed from an MLS map is shown in Figure 6.6. As can be seen, in areas where the data is less dense, the map is represented by triangles, lines and isolated points. Note that each vertex in the 3D mesh corresponds to one surface patch in the map.

6.6 Conclusions

In this chapter, we presented different approaches to store 3D data that are particularly useful for localization and planning in mobile robotics. We analyzed elevation maps, extended elevation maps and multi-level surface maps, where the latter approach was introduced as a new framework. The major advantage of MLS maps is their ability to represent multiple height values and vertical objects such as buildings or trees. This leads to more realistic maps while the memory requirements are still comparably low. In the experiments shown in the next chapter, we will see that MLS maps are capable to represent difficult environments such as bridges or underpasses and that they enable the robot to plan a path for traversing the same xy position at different height levels.

7

Globally Consistent Maps

7.1 Introduction

In the last chapter, we saw how we can efficiently store 3D data from point clouds while still maintaining accurate maps of the environment. However, these maps are still *local* in the sense that they are computed from range data that was collected at one fixed robot position[1]. This means for example, that occlusions appearing in the local map can not be resolved, because the scene was only perceived from one point of view. Therefore, we need to incorporate the data acquired from other positions into the map. To achieve this, we need to find the correct 3D rigid body transformation between the particular robot positions at which the local maps where acquired. For any pair of such robot positions this can be done by finding data points in both maps that correspond to the same object in reality and by subsequently computing a 3D rotation and translation from these correspondences. This process is called *map registration* and is described in section 7.2. The result of the map registration is a rotation and a translation that minimize the sum of the distances between corresponding data points. This is in general an over-constrained problem, which means that it is impossible to find a transform with a residual error of zero. The problem arising here is that these residual errors accumulate when the map registration is applied to a chain of subsequently acquired local maps. Especially, this becomes apparent when the last and the first map of the chain are taken at similar places. In this case, the accumulated registration error must be corrected globally. This is usually called *loop closing*. In section 7.3, we describe an algorithm for the loop closing problem which is based only on the results of the local map registration between nearby robot poses. We will denote these as *local constraints*. In section 7.4 we show that this

[1]We note that for the context of this chapter, a map may be any possible 3D data representation, such as point clouds, elevation maps, MLS maps etc.

loop closing can be improved by incorporating global knowledge about the environment. This is expressed in terms of *global constraints* between several different robot positions. The result of this will be a globally consistent map of the environment, which means that instances of objects that have been mapped in different local maps appear only as one object in the global map.

7.2 Map Registration

As mentioned already, in map registration we compute a 3D rigid body transformation between two local maps. If the particular local maps are totally disjoint from each other, i.e. there is no *overlap* between any pair of maps, then we can not identify certain objects from one map inside another one. In this case, we have no further information of how to merge the maps other than the odometry of the robot. However, if there is some overlap between pairs of maps, then we have an additional source of information, namely the fact that objects from the real world that have been mapped in the overlap area must appear in both maps. This enables us to determine a set of corresponding parts of the maps from which a 3D transformation can be computed. This will be described in the following.

7.2.1 Registering Local Maps: The ICP Algorithm

Let us consider the case of two local maps that have some overlap, i.e. some parts of one map can be found in the other one. We will assume that both maps are given as a *data* point cloud $\mathcal{D}$ and a *model* point cloud $\mathcal{M}$. This is the most general form of representing 3D data, because for any underlying map representation, such as MLS maps, we can think of a feature extraction algorithm that computes point features from both maps (see section 6.5.4). In the simplest case, these point features may be simply the center points of the surface patches of the MLS map.

The standard algorithm to compute a 3D transformation between $\mathcal{D}$ and $\mathcal{M}$ is the ICP algorithm, which stands for Iterative Closest Points. Sometimes it is also referred to as the Iterative Corresponding Points algorithm, which reflects the nature of the algorithm more precisely. The ICP algorithm was first stated by Besl and McKay 1992 and has been extended and applied in many areas by various researchers. In algorithm 3 we see the basic computational steps of ICP. In essence, they consist of iteratively determining points in $\mathcal{M}$ closest to points in $\mathcal{D}$, computing a 3D transform $T = (R, \mathbf{t})$ and applying T to the data points $\mathcal{D}$. In detail, the steps are as follows.

Finding the Correspondences In this step, a set of point pairs is returned which are considered as corresponding points. In the simplest case, this step computes for each point $\mathbf{p}$ in the current point cloud $\mathcal{P}_k$ its closest point $\mathbf{q}$ in $\mathcal{M}$. At the beginning, $\mathcal{P}_0$ is initialized with the data points $\mathcal{D}$. Later, we will see other possible computations for the corresponding point pairs. In terms of running time, the correspondence computation is the most demanding step in the ICP algorithm. The naïve implementation needs N comparisons with M points where N is the size of $\mathcal{D}$ and M the size of $\mathcal{M}$. This can be reduced

Algorithm 3 Iterative Closest Point – find a translation $\mathbf{t}$ and a rotation R between two point clouds $\mathcal{D}$ and $\mathcal{M}$

```
Definitions:
        $k$ : iteration counter
        $r_{-1}, r_0, \ldots$ : residual at each step
        $\tau$ : residual threshold
        $\mathcal{P}_0, \mathcal{P}_1, \ldots$ : local point clouds
        $\mathcal{Y}_0, \mathcal{Y}_1, \ldots$ : subsets of $\mathcal{M}$
 1: $\mathcal{P}_0 \leftarrow \mathcal{D}$
 2: $k \leftarrow 0$
 3: $r_{-1} \leftarrow \infty$
 4: $\mathcal{Y}_k \leftarrow$ C P $(\mathcal{P}_k, \mathcal{M})$
 5: $(R_k, \mathbf{t}_k, r_k) \leftarrow$ T $(\mathcal{D}, \mathcal{Y}_k)$
 6: $\mathcal{P}_{k+1} \leftarrow$ T $(R_k, \mathbf{t}_k, \mathcal{D})$
 7: if $r_{k-1} - r_k < \tau$ then
 8:     return $(R_k, \mathbf{t}_k)$
 9: else
10:     $k \leftarrow k + 1$
11:     goto 4
12: end if
```

when using *kD*-trees [Bentley, 1975; see also section 4.3.1]. The computation time is then in $O(N \log M)$, but in cases with large point sets the corresponding points computation is still slow. Therefore, other techniques such as sub-sampling can be applied. We will return to this later.

Computation of the 3D Transformation Assume we have two sets of corresponding points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and $\mathcal{Y} = \{\mathbf{x}\mathbf{y}_1, \ldots, \mathbf{y}_N\}$ as described in the previous paragraph. Our goal now is to find a rotation matrix R and a translation vector $\mathbf{t}$ that minimizes the mean squared distance between points in $\mathcal{X}$ and its corresponding transformed points in $\mathcal{Y}$. Thus, we minimize

$$e(R, \mathbf{t}) := \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - (R\mathbf{x}_i + \mathbf{t})\|^2 \tag{7.1}$$

Several possible ways to perform this minimization exist. The first one was presented by Horn [1987] and used quaternions to represent the 3D rotation R. Later, there was an approach by Umeyama [1991] which uses the singular value decomposition (SVD) to compute the rotation. This approach is more elegant and easier to implement and therefore we will shortly sketch it here.

First, we switch to a matrix representation of equation (7.1). This is done by introducing $3 \times n$ matrices $X = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ and $Y = [\mathbf{y}_1, \ldots, \mathbf{y}_N]$ in which each data point is represented as a column. In addition, we define an N-dimensional vector $\mathbf{h}$ as $\mathbf{h} = (1, 1, \ldots, 1)^T$.

Then we can reformulate equation (7.1) as

$$e(R, \mathbf{t}) := \frac{1}{N} \|Y - RX - \mathbf{t}\mathbf{h}^T\|^2 \tag{7.2}$$

Next, we introduce a normalization matrix $K = I - (1/N)\mathbf{h}\mathbf{h}^T$ and substitute X by $XK + (1/N)X\mathbf{h}\mathbf{h}^T$ and Y by $YK + (1/N)Y\mathbf{h}\mathbf{h}^T$. This yields

$$e(R, \mathbf{t}) = \frac{1}{N} \|YK - RXK\|^2 + \|\mathbf{t}'\|^2 \tag{7.3}$$

where

$$\mathbf{t}' = -\frac{1}{N} Y\mathbf{h} + \frac{1}{N} RX\mathbf{h} + \mathbf{t} \tag{7.4}$$

This means, that for the minimization, $\mathbf{t}'$ must be $\mathbf{0}$, i.e.

$$\begin{aligned} \mathbf{t} &= \frac{1}{N} Y\mathbf{h} - \frac{1}{N} RX\mathbf{h} \\ &= \boldsymbol{\mu}_Y - R\boldsymbol{\mu}_X \end{aligned} \tag{7.5}$$

where $\boldsymbol{\mu}_X$ and $\boldsymbol{\mu}_Y$ are the means of the point clouds $\mathcal{X}$ and $\mathcal{Y}$ respectively. For the computation of the optimal rotation matrix R we first define the cross covariance matrix Σ_{XY} as

$$\Sigma_{XY} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y}_i - \boldsymbol{\mu}_Y)(\mathbf{x}_i - \boldsymbol{\mu}_X)^T \tag{7.6}$$

and the singular value decomposition of Σ_{XY} as UDV^T. Using the lemma from Umeyama [1991], R is determined uniquely if the rank of Σ_{XY} is at least 2. In this case, R is computed as

$$R = USV^T \tag{7.7}$$

where

$$S = \begin{cases} I & \text{if } \det(\Sigma_{XY}) > 0 \\ & \text{or } \operatorname{rank}(\Sigma_{XY}) = 2 \wedge \det(U)\det(V) = 1 \\ \operatorname{diag}(1, 1, \ldots, 1, -1) & \text{if } \operatorname{rank}(\Sigma_{XY}) = 2 \wedge \det(U)\det(V) = -1 \\ & \text{or } \det \Sigma_{XY} < 0. \end{cases} \tag{7.8}$$

The distinctions in equation (7.8) are useful in cases where the cross covariance matrix is rank deficient. An example of such a case is when both point clouds are distributed on a plane. In this case, equation (7.7) yields a unique rotation even though the rotation around the plane normal is not unique.

To summarize, the computation of the rotation and translation between point clouds $\mathcal{X}$ and $\mathcal{Y}$ is done as follows

1. Compute the means $\boldsymbol{\mu}_X$ and $\boldsymbol{\mu}_Y$
2. Compute the cross covariance Σ_{XY} according to equation (7.6)
3. Compute the SVD of Σ_{XY}
4. Compute R according to equations (7.7) and (7.8)
5. Compute $\mathbf{t}$ according to equation (7.5)

In Besl and McKay [1992], a formal proof is given for the convergence of the ICP algorithm to a local optimum. Furthermore, it is mentioned that a good initial estimate of the rotation and translation to be computed increases the probability that the ICP algorithm converges to the global optimum. In our application, such an initial estimate can be obtained from the robot's odometry measurements that are taken while the robot travels from the position where the first local map is acquired to the position of the second local map.

7.2.2 Variants of the ICP Algorithm

In the literature, a series of different variants of the ICP algorithm can be found. A good overview was given by Rusinkiewicz and Levoy [2001], where the authors compared the existing approaches with respect to the following six aspects:

- **Point set selection:** Some algorithms use the whole point sets from the data and the model points, others apply different kinds of sampling strategies to reduce the complexity of the input data.
- **Point matching strategy:** This refers to different ways of defining a pair of corresponding points. Examples range from simply taking the point that is closest to each data point to projecting each data point (from $\mathcal{D}$) into the mesh of the model point set $\mathcal{M}$.
- **Correspondence pair weighting:** Some approaches additionally define weights for each corresponding point pair. These weights can for example be dependent on the point-to-point distance or on the compatibility of the normal vectors (defined by the dot product of the normals)
- **Correspondence rejection:** Similar to the weighting of correspondence pairs, this is an attempt to classify into good and bad correspondences. For example, pairs of points that are far away from each other might be rejected to obtain a more accurate estimation of the 3s transform. Also, a certain percentage of point pairs can be rejected. This is also referred to as the *trimmed* ICP algorithm [Pulli, 1999].
- **Error metric assignment:** Different error metrics may be used instead of the one defined in equation (7.1). For example, the color information at each data point may be included in the metric. Also, a point-to-plane metric has been proposed where

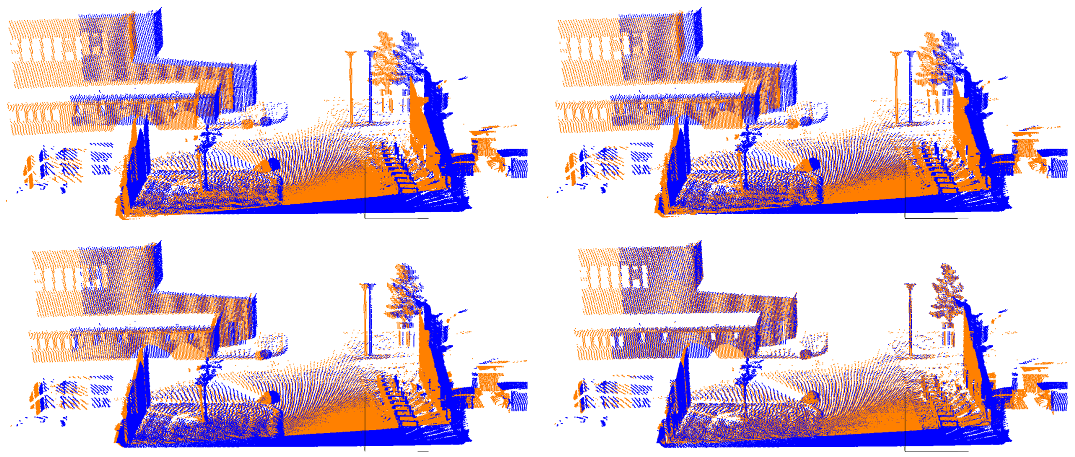


Figure 7.1: *Different stages of the ICP registration process. Shown is the initial displacement (top left) and the matching result after 2, 5, and 15 iterations.*

the distance of a data point from $\mathcal{D}$ to the plane containing the corresponding point in $\mathcal{M}$ and oriented perpendicular to its normal is computed [Chen and Medioni, 1991].

- **Minimization approach:** Apart from the standard minimization presented in the previous section, the new 3D transform can be found by extrapolating the last 3 transforms at each iteration either linearly or quadratically. This accelerates the convergence [Besl and McKay, 1992].

For the scope of this work, we will focus only on a combination of two ICP variants, namely the data reduction by sub-sampling and the rejection of bad point correspondences. These two strategies have shown to be most effective with respect to run time reduction and reducing the risk of running into local optima of the error metric. For a more detailed discussion we again refer to Rusinkiewicz and Levoy [2001].

As an example, consider the two point clouds shown in figure 7.1. They were obtained by extracting features from two different laser range scans (for details see Pfaff et al. [2007a]). In the example, the ICP algorithm takes 15 iterations to converge to the correct mapping. In particular, consider the street lamp in the background to see how the alignment error is reduced.

7.2.3 Registration of MLS maps using ICP

In the special case that we are given two local MLS maps as presented in section 6.5, we need to adapt the ICP based registration process to the MLS map structure. This means that we need to extract point data from both MLS maps that can be used to define the correspondences between the maps. In the case of horizontal surface patches this can be done by simply taking the mean height of the patch and the xy-coordinates of the corresponding cell. For vertical cells we can sample uniformly from the vertical interval defined by the patch. The number of samples should be dependent on the height of the interval. In our experiments, we sampled four points per meter.

Furthermore, we modify the error equation (7.1) slightly by splitting it up into three different sums of disjoint sets of correspondences. We use the classification of surface patches into 'vertical', 'horizontal and traversable' and 'horizontal and not traversable' which was presented in section 6.5.4. To formulate this mathematically, let us denote the points that were sampled from vertical objects in the first map with $\mathbf{u}_{i_c}$ and their corresponding sample points in the other map with $\mathbf{u}_{j_c}$ where $(i_1, j_1), \dots, (i_{C_1}, j_{C_1})$ are the C_1 pairs of corresponding indices. Similarly, we define $\mathbf{v}_{i_c}$ and $\mathbf{v}'_{j_c}$ as the corresponding points from traversable patches and $\mathbf{w}_{i_c}$ and $\mathbf{w}'_{j_c}$ as points from non-traversable patches. Then, the modified error equation is

$$e(R, \mathbf{t}) = \underbrace{\sum_{c=1}^{C_1} d(\mathbf{u}_{i_c}, \mathbf{u}'_{j_c})}_{\text{vertical cells}} + \underbrace{\sum_{c=1}^{C_2} d(\mathbf{v}_{i_c}, \mathbf{v}'_{j_c})}_{\text{traversable}} + \underbrace{\sum_{c=1}^{C_3} d(\mathbf{w}_{i_c}, \mathbf{w}'_{j_c})}_{\text{non-traversable}}. \tag{7.9}$$

Note that this modified error equation does not change the computation of the optimal transform. We still minimize the error $e(R, \mathbf{t})$ as a whole rather than minimizing each sum of distances separately. Equation (7.9) only says that the selection of correspondences is done class-wise instead of simply taking the nearest neighbor of each data point. This reduces the amount of wrong correspondences, which are one major cause for poor results of the ICP algorithm.

7.3 Closing the Loop using Local Constraints

As mentioned already, the map registration process in general ends up in a small residual error $e(R, \mathbf{t})$, because corresponding data points between two consecutive maps can in general not be identified with the same 3D location in the real environment. In fact, assigning nearby data points from different maps to each other as corresponding points, is an approximation. The problem with this approximation is that it decreases the certainty of the global transform between the first and the last local map in a chain of maps. The longer such a chain of local maps grows, the higher is the uncertainty of the computed robot position for the last local map. This means that when the robot returns to the position where it acquired the first map in the chain, the overall scan matching error may be so large that the resulting global map is inconsistent. Thus, the first and the last map do not "fit" together, although they have been acquired (nearly) at the same positions in space. The knowledge that the first and the last position are close to each other, is not incorporated in the registration process when it is based only on local scan matching. Olson et al. [2006] present an approach to cope this problem by defining a *network* of robot poses and minimizing a global error function. The main ideas are similar to the ones by Lu and Milios [1997] and will be presented in the next section.

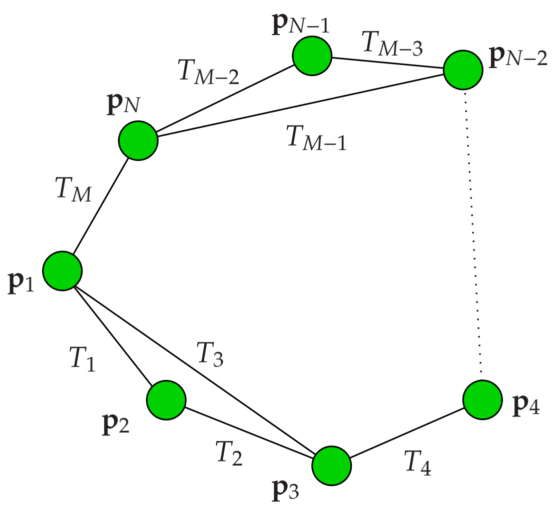


Figure 7.2: *Loop closing in a network of robot poses. Each node corresponds to a robot pose, i.e. position and orientation in 3D space. An edge between two poses represents the local rigid-body transform between the local maps at the poses. These transforms impose local constraints on the robot poses. Note that the local constraints are not necessarily defined on consecutive poses.*

7.3.1 Network-based Pose Optimization

Suppose the robot recorded local 3D maps at N different poses $\mathbf{p}_1, \ldots, \mathbf{p}_N$. A pose is defined as a 6-tuple $(x, y, z, \varphi, \vartheta, \psi)$ of location (x, y, z) and orientation $(\varphi, \vartheta, \psi)$. Whenever two local maps that have been acquired at poses $\mathbf{p}_i$ and $\mathbf{p}_j$ are matched to each other using the map registration technique described above, a *constraint* is imposed on the poses $\mathbf{p}_i$ and $\mathbf{p}_j$. If we represent all these constraints as undirected edges in a graph, where the nodes are defined by the robot poses, we obtain a *network* of robot poses. Figure 7.2 shows an example of such a network. It can be used to minimize the global registration error. An overview of techniques to achieve this has been presented by Olson et al. [2006]. There, the authors show that the LU decomposition and a new modified stochastic gradient descent (SGD) approach yield the best results. However, the experiments performed in that work were only done on two-dimensional input data. In fact, it turns out that the SGD approach can not be applied for the case of 3D data as presented by Olson et al. [2006]. This is because the assumption of a linear angular subspace made there does not apply for the 3D case. A 3D version of the SGD approach addressing this problem is currently under investigation.

7.3.2 Local Constraints between Robot Poses

As described above, a local constraint between two robot poses $\mathbf{p}_i$ and $\mathbf{p}_j$ is an edge between $\mathbf{p}_i$ and $\mathbf{p}_j$ in the network of poses. Mathematically, a local constraint corresponds to the rigid-body transform T_{ij} between data points in the local reference frame at $\mathbf{p}_i$ and the corresponding points in the local reference frame at $\mathbf{p}_j$. This transform directly depends on the rotation R_{ij} and the translation $\mathbf{t}_{ij}$ computed by the map matching process between the maps at $\mathbf{p}_i$ and $\mathbf{p}_j$. We can see the pair $(R_{ij}, \mathbf{t}_{ij})$ as the transform between the *global* coordinates of the points at pose $\mathbf{p}_i$ and the global coordinates of the points at $\mathbf{p}_j$, because the ICP algorithm takes as input an initial guess of the transform, which can also be given by the initial estimates of $\mathbf{p}_i$ and $\mathbf{p}_j$. This means, a local constraint T_{ij} is defined by

$$T_{ij}(\mathbf{f}) \quad := \quad P_i^{-1}(R_{ij}P_j(\mathbf{f}) + \mathbf{t}_{ij}), \qquad \mathbf{f} \in \mathbb{R}^3 \tag{7.10}$$

Here we use the notation $P_j(\mathbf{f})$ for the function that transforms a point $\mathbf{f}$ from the local reference frame at the robot position $\mathbf{p}_j$ to the global reference frame. Accordingly, P_i^{-1} transforms a feature in global coordinates into the local reference frame at position $\mathbf{p}_i$.

7.3.3 The Optimization Algorithm

Using the above notation we can now formulate the pose optimization problem. For a given set of initial robot poses $\mathbf{p}_1, \ldots, \mathbf{p}_N$ the local map registration returns a set of local constraints T_{ij} for $i, j = 1, \ldots, N, i \neq j$ according to equation (7.10). These constraints are then encoded in a *goal vector* $\mathbf{g}$ of length $6M$ where M is the number of constraints T_{ij}.

$$\mathbf{g} := (\ldots, x_{ij}, y_{ij}, z_{ij}, \varphi_{ij}, \vartheta_{ij}, \psi_{ij}, \ldots) \tag{7.11}$$

In other words, $\mathbf{g}$ contains all local constraints expressed as 3D translation and rotation. Next we define the *constraint function* $f : \mathbb{R}^{6N} \to \mathbb{R}^{6M}$ that maps robot poses to local constraints.

$$f(x_1, \ldots, \psi_1, \ldots, x_N, \ldots, \psi_N) := \begin{pmatrix} \vdots \\ \alpha(P_i^{-1}P_j) \\ \vdots \end{pmatrix} \tag{7.12}$$

Here we introduced the function α which converts a pose transform P into its 6 parameters $(x, y, z, \varphi, \vartheta, \psi)$. Using the constraint function f, we can formulate the loop closing problem as the minimization of the squared difference between $f(\mathbf{p})$ and the goal vector $\mathbf{g}$

$$\mathbf{p}^* = \underset{\mathbf{p}}{\operatorname{argmin}} \, \|f(\mathbf{p}) - \mathbf{g}\|^2 \tag{7.13}$$

Here, $\mathbf{p}$ denotes the vector of all robot poses $\mathbf{p}_1, \ldots, \mathbf{p}_N$ and $\mathbf{p}^*$ are the optimal poses to be found. The constraint function f is non-linear due to the rotations. We therefore linearize it according to $f(\mathbf{p}) \approx F|_{\mathbf{p}} + J|_{\mathbf{p}}\Delta\mathbf{p}$ where $J|_{\mathbf{p}}$ is the Jacobian of the constraint function f at $\mathbf{p}$. At each iteration of the pose optimization $F|_{\mathbf{p}}$ and $J|_{\mathbf{p}}$ are recomputed for the new poses and we will write simply F and J for better readability. Using this, we can reformulate the global pose optimization as

$$\mathbf{p}^* \approx \underset{\mathbf{d}}{\operatorname{argmin}} \, \| J\mathbf{d} - \mathbf{r} \|^2 \tag{7.14}$$

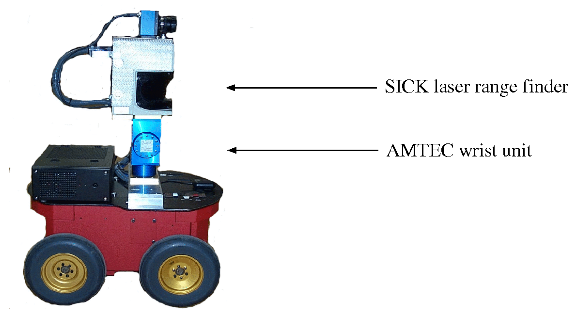


Figure 7.3: *The robot Herbert used for the 3D data acquisition in outdoor terrains. It is equipped with an AMTEC pan/tilt unit that carries a SICK LMS 291 laser range finder. In addition, it has a CCD camera which has not been used in our experiments.*

Here we substituted $\Delta\mathbf{p}$ by $\mathbf{d}$ and $(F - \mathbf{g})$ by $\mathbf{r}$.

By deriving with respect to $\mathbf{d}$ and setting to zero we obtain

$$J^T J\mathbf{d} = J^T\mathbf{r} \tag{7.15}$$

This is a standard linear algebra problem which can be solved by LU decomposition or by multiplication with the pseudo-inverse of $J^T J$. The overall algorithm can then be described by iterating the following steps:

- Compute F and J from the poses $\mathbf{p}$.
- Minimize (7.14) according to Equation (7.15).
- Compute new poses: $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{d}$.

This is repeated as long as the residual $\mathbf{r}$ exceeds a threshold or a maximum number of iterations is reached.

7.3.4 Experiments

As an application example of the loop closing algorithm derived above, we choose the construction of a consistent 3D map of an outdoor environment, which in our case is the campus of the computer science department at the university of Freiburg. The 3D data was acquired using a SICK LMS 291 laser range finder mounted on an AMTEC wrist module PW70. The robot platform we choose is a Pioneer II AT robot named Herbert. Figure 7.3 shows a photograph of the assembled system.

To acquire the data, we steered the robot over the university campus and recorded 3D point cloud data at different robot positions. From each of these point clouds we computed a local MLS map as described in section 6.5. As long as the robot did not return to a previously visited area, no loop closing could be performed and the registration of the MLS maps was done as described in section 7.2.3. In cases where a loop could be closed, we applied the algorithm described in the previous section. The decision of whether to close a loop was taken manually. In fact, for a better evaluation, the last robot position in each experiment was chosen to be exactly the first position, which was marked before on the ground. The aims of the experiments were the following:

- Show the high data reduction by using MLS maps
- Show the possibility of representing multiple levels
- Demonstrate the loop-closing technique in large loops

In total, two experiments were performed. In the first one, the robot acquired 77 scans consisting of $20,207,000$ data points. The area scanned by the robot spans approximately 195 by 146 meters. During the data acquisition, the robot traversed a loop with a length of $312m$. Figure 7.4 shows two views of the resulting MLS map with a cell size of $10cm \times 10cm$. The yellow/light grey surface patches are classified as traversable. It requires $57.96MB$ to store the computed map, where 24% of $2,847,300$ cells are occupied. As can be seen from the figures, the loop is closed properly and the resulting map is consistent. The final pose deviation between the first and the last robot pose was only a few centimeters.

In this first experiment, the robot was always positioned at the ground level. Therefore, to demonstrate the power of MLS maps in representing multiple levels, we performed a second experiment, in which the robot traversed an underpass of a bridge and crossed this same bridge later. Then, it drove down again to the ground level via a different path, traversed the underpass a second time and returned to its start position. In total, the collected data set consists of 172 individual scans with a total of $45,139,000$ data points. In contrast to the first experiment, this loop was too large to be closed, because the accumulated registration error was too big to find accurate correspondences between overlapping local maps. Therefore we proceeded as follows: Considering that the local scan matches between consecutive robot poses is highly accurate, we matched local maps corresponding to 5 consecutive poses into new and bigger local maps. This means that maps $m_1, \ldots, m_5$ are matched into the new map $\bar{m}_1$, maps $m_6, \ldots, m_{10}$ into $\bar{m}_2$ and so on. This way we obtain a set of $\frac{N}{5}$ new local maps, which reduces the pose optimization problem. A further improvement can be achieved by increasing the overlap between consecutive local maps $\bar{m}_i$ and $\bar{m}_{i+1}$. This can be done by adding the last partial map from $\bar{m}_i$ to $\bar{m}_{i+1}$. For example, the local map m_5 is then also a part of the new local map $\bar{m}_2$. In this way, the scan matching error between the joined local maps $\bar{m}_i$ and $\bar{m}_j$ can be reduced and the global pose optimization is more likely to converge to a global optimum.

The MLS map resulting from the second experiment is shown in Figure 7.5. Again, the cell size was chosen to $10cm \times 10cm$ and the yellow/light grey surface patches depict the traversable areas. The area scanned by the robot spans approximately 299 by 147 meters and the length of the loop traversed by the robot is $560m$. The memory requirement of the map is $73.33MB$, where 20% of $4,395,300$ cells are occupied. In addition to the fact that the map is consistent and highly efficient in memory, we can see from the figure the two different levels at which the robot was positioned during the data acquisition.

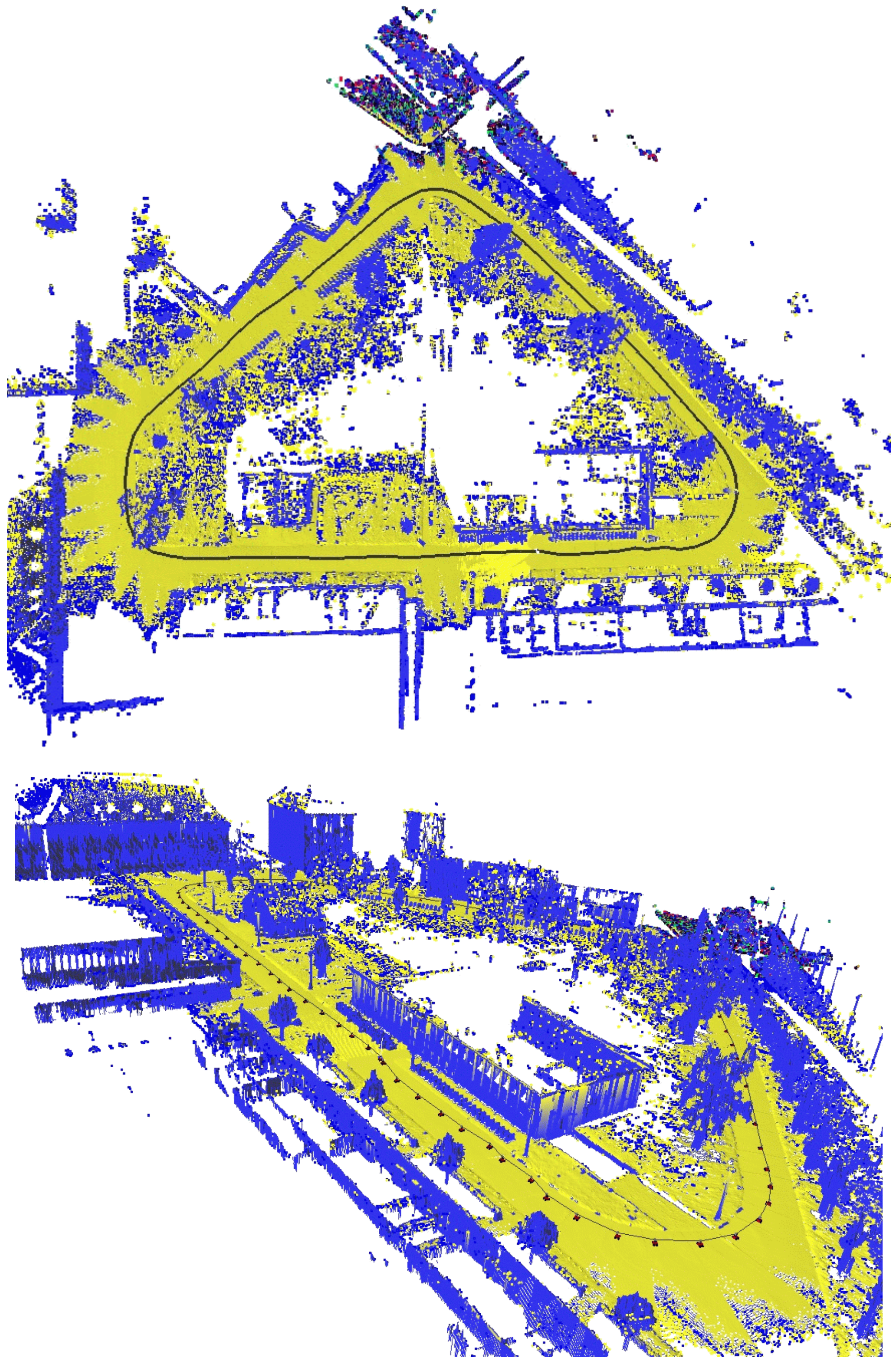


Figure 7.4: *Two views of the resulting MLS map of the first experiment with a cell size of 10cm x 10cm. The area scanned by the robot spans approximately* 195 *by* 146 *meters. During the data acquisition, where the robot collected* 77 *scans consisting of* 20, 207, 000 *data points, the robot traversed a loop with a length of* 312*m.*

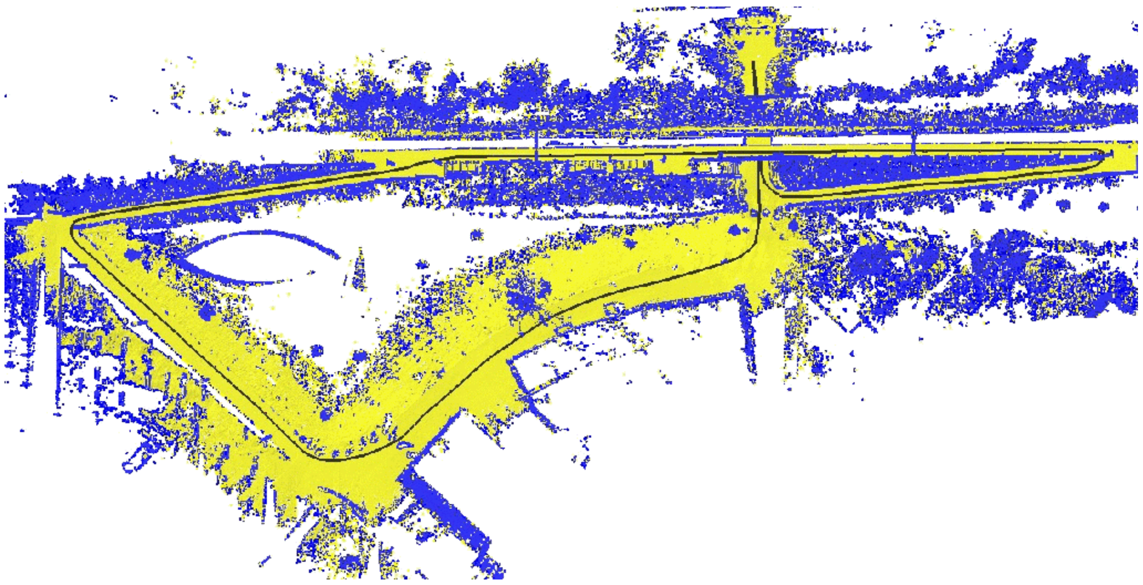


Figure 7.5: *Resulting MLS map of the second experiment with a cell size of 10cm x 10cm. The area scanned by the robot spans approximately 299 by 147 meters. During the data acquisition, where the robot collected 172 scans consisting of 45,139,000 data points, the robot traversed a loop with a length of 560m.*

7.4 Closing the Loop using Local and Global Constraints

As we have seen in the previous section, in cases of large loops it may be difficult or even impossible to compute a consistent map from the input data acquired by the robot. The reason for this is that the map registration can be performed only on maps acquired at consecutive robot poses because for other pairs of maps there is not enough overlap. This means that the estimate of each robot position is only based on the (poor) estimate of the previous position and the result of the scan matching. One way to cope with this problem is to use other sources of information about the positions of the robot. This can be done by exploiting the structure of the scene that is to be mapped by the robot. The idea behind this is motivated by the fact that many man-made environments such as buildings often contain features that can be observed in many views and at similar places. For example, the windows belonging to one and the same level of a building are typically at the same height on all sides of the building. Accordingly, the windows introduce additional constraints on range scans even if the scans have only a small or even no overlap. The key idea of the approach proposed in this section is to extract such *global constraints* from three-dimensional range scans and to improve the optimization process based on these global constraints.

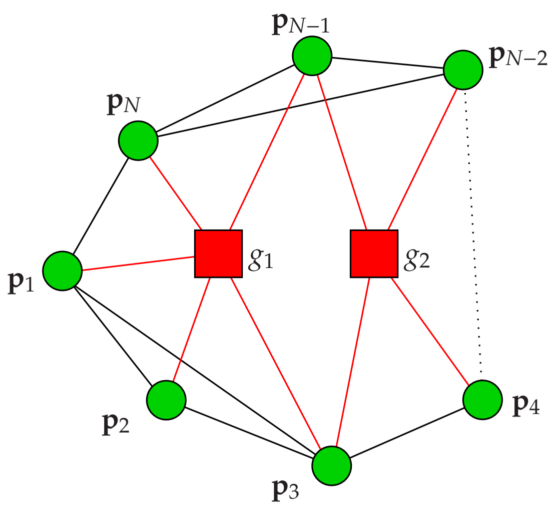


Figure 7.6: *Extended robot pose network. In addition to the local constraints represented as undirected edges between the nodes in the graph, we introduce global constraints represented as hyper-edges. In this example, two global constraints are imposed, one between the poses* $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_{N-1}$, *and* $\mathbf{p}_N$, *and one between the poses* $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_{N-2}$, *and* $\mathbf{p}_{N-1}$.

7.4.1 Extension of the Pose Network

As before, we assume we are given N different local maps recorded at the robot positions $\mathbf{p}_1, \ldots, \mathbf{p}_N$. To represent the estimation problem of these robot poses graphically, we introduced in section 7.3.1 the pose network with nodes for each robot pose and edges for each local constraint imposed on two robot poses. In this section, we extend the pose network by introducing *hyper-edges*. A hyper-edge connects several different nodes in the graph and is identified with a global constraint that is imposed on the corresponding robot poses. Figure 7.6 shows an example of such an extended robot pose network. In the example, two global constraints are added to the graph, one connecting five poses and one with four poses.

7.4.2 Local and Global Constraints in a Common Error Function

In analogy to the case where we are given only local constraints (see section 7.3.3), our goal for the case of local and global constraints is to minimize an appropriate error function similar to the one from equation (7.13). However, as we will see later, for the type of global constraints we use here, it is not possible to define a constraint function f as in equation (7.12). We therefore express the pose optimization problem in a more general form by defining an overall error function $f_e : \mathbb{R}^{6N} \to \mathbb{R}$ that is to be minimized

$$f_e(\mathbf{p}_1, \ldots, \mathbf{p}_N) := \gamma l_e(\mathbf{p}_1, \ldots, \mathbf{p}_N) + (1-\gamma) g_e(\mathbf{p}_1, \ldots, \mathbf{p}_N) \tag{7.16}$$

Thus, the overall error f_e is split up into an error function l_e that is assigned to all local constraints and an error function g_e corresponding to the global constraints. To weigh between the two types of errors we introduced the factor $\gamma \in \mathbb{R}$. In the following, we will describe the error functions l_e and g_e in detail.

The Error Function Based on Local Constraints

In section 7.3.3 we defined the error that stems from the local constraints as the squared distance between the 6D transform parameters $(x, y, z, \varphi, \vartheta, \psi)$ resulting from the ICP matcher and the 6D parameters derived from the pose transforms P_i. This means that the objective was to get as close as possible to the local transforms computed by the ICP algorithm. The problem with this is that the uncertainty of these local transforms varies over the transform parameters. The ICP algorithm, as it is described in section 7.2, does not provide this different uncertainties, so we have to assume them to be equal[2]. The result of this is that there may be a set of transform parameters that are similar to the ones obtained with the global pose optimizer, but yield a better fitting of the individual local maps. Thus, the actual objective of the pose optimization should be this fitting of the local maps instead of trying to reach the error prone local transform parameters.

For this reason we define the error function l_e as the sum of the Euclidean distances between corresponding points in the local maps. The local map at the robot position $\mathbf{p}_i$ in this context is represented by a set of 3D features $\mathcal{F}_i = \{\mathbf{f}^i_1, \ldots, \mathbf{f}^i_{s(i)}\}$ where $s(i)$ is the *size* of the map with index i. As mentioned above, these features can be interpreted differently depending on the 3D map representation used. In the case of point clouds, they correspond to the 3D coordinates of the data points. Using this, we define the error that results from the local constraint between two robot positions $\mathbf{p}_i$ and $\mathbf{p}_j$ as

$$l(\mathbf{p}_i, \mathbf{p}_j) \quad := \quad \sum_{k=1}^{C} \|P_i(\mathbf{f}^i_{c_1(k)}) - P_j(\mathbf{f}^j_{c_2(k)})\|^2 \tag{7.17}$$

where $(c_1(1), c_2(1)), \ldots, (c_1(C), c_2(C))$ is the set of C correspondences between points from the local maps at $\mathbf{p}_i$ and $\mathbf{p}_j$ that have been computed by the ICP algorithm after convergence. In the equation, we use again the notation $P_i(\mathbf{f}^i)$ for the transform of the feature $\mathbf{f}^i$ from local coordinates at $\mathbf{p}_i$ to global coordinates (see equation (7.10)). If we assume we are given L local constraints $l_1, \ldots, l_L$ as defined in equation (7.17), then the error function l_e can be expressed as

$$l_e(\mathbf{p}_1, \ldots, \mathbf{p}_N) = \sum_{i=1}^{L} l_i(\mathbf{p}_{\nu_1(i)}, \mathbf{p}_{\nu_2(i)}). \tag{7.18}$$

In this equation, we introduce the index functions ν_1 and ν_2 that map each local constraint index $i = 1, \ldots, L$ to the index of the first and second robot pose $\nu_1(i)$ and $\nu_2(i)$ between which the constraint is imposed.

[2]There exist approaches to estimate these uncertainties during the ICP computation. However, they usually yield an approximation and are not feasible due to their high computational time requirements.

The Error Function Based on Global Constraints

In general, a global constraint between different robot poses can be defined in many possible ways. For example, if a set of 3D landmarks with known poses is given, then each of these landmarks constitutes a global constraint on all robot poses from which a partial view onto the landmark was taken. The error function that corresponds to a global constraint may then defined by the squared Euclidean distance between one of the landmarks and the view of the landmark in global coordinates as seen from the corresponding robot pose.

In this context, we will not assume the existence of known landmarks. We rather define the global constraints based on the object(s) seen from the different views. This stems from the observation that many real world objects, such as buildings are highly self-similar. For example, if a building is seen from two sides, it is very likely that specific features that are extracted from both views (e.g., windows, the edge between walls and the roof etc.) are on the same absolute height in both views. In general, many different types of features are possible, where high-level features such as windows, doors etc. constrain the algorithm to be applicable for specific objects like buildings. We therefore rely on low-level features, in particular 3D edges.

Before we discuss how to extract the edges, we will first describe how we actually use them to generate global constraints. Assume we are given a set of edge features $\mathcal{E}_i = \{e^i_1, \ldots, e^i_M\}$ for the map at $\mathbf{p}_i$. We define a global constraint as a plane p that has a sufficient *support* by edges detected in different maps. Here the support supp(p) of a plane p is defined by all edges e that lie entirely inside a given corridor around p. For convenience, we will denote the set of all edges $e \in \mathcal{E}_i$ that are in the support of p as $\mathcal{S}^p_i$. Now, we calculate the error resulting from the global constraint g between the K poses $\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_K}$ as

$$g(\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_K}) := \sum_{k=1}^{K} \sum_{e \in \mathcal{S}^p_{i_k}} d(P_{i_k}(e), p) \tag{7.19}$$

Here, d defines the squared distance of the transformed edge e to the plane p. To compute it, we can not use equation (2.14) in general, because $P_{i_k}(e)$ is usually not parallel to p. Therefore, we compute $d()$ as the sum of squared Euclidean distances between the vertices of $P_{i_k}(e)$ and p.

Using (7.19) we can express the error function g_e for a given set of G global constraints $g_1, \ldots, g_G$ where the constraint g_i is imposed on K_i different robot poses as

$$g_e(\mathbf{p}_1, \ldots, \mathbf{p}_N) = \sum_{i=1}^{G} g_i(\mathbf{p}_{\mu_1(i)}, \ldots, \mathbf{p}_{\mu_{K_i}(i)}) \tag{7.20}$$

Again, we use index functions $\mu_1(i), \mu_2(i), \ldots$ that map from constraint indices $i = 1, \ldots, G$ to pose indices $n = 1, \ldots, N$.

Plugging (7.18) and (7.20) into (7.16), we obtain for the overall error function f_e:

Algorithm 4 *poseOpt*($\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathcal{F}_1, \ldots, \mathcal{F}_N, l_1, \ldots, l_L$) – Robot pose optimization using local and global constraints

Inputs:
$\mathbf{p}_1, \ldots, \mathbf{p}_N$: initial estimate of N robot poses
$\mathcal{F}_1, \ldots, \mathcal{F}_N$: 3D features of N local maps
$l_1, \ldots, l_L$: local constraints between pairs of robot poses

Outputs:
$\mathbf{p}_1, \ldots, \mathbf{p}_N$: optimized robot poses

```
1: $iterations \leftarrow 0$
2: for $i = 1, \ldots, L$ do
3:    $(v_1, v_2) \leftarrow$ P I $(l_i)$
4:    $C_i \leftarrow$ C $(\mathbf{p}_{v_1}, \mathbf{p}_{v_2}, \mathcal{F}_{v_1}, \mathcal{F}_{v_2})$
5: end for
6: $g_1, \ldots, g_G \leftarrow$ G C $(\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$
7: $(\mathbf{p}_1, \ldots, \mathbf{p}_N) \leftarrow$ P $(\mathbf{p}_1, \ldots, \mathbf{p}_N, \mathcal{F}_1, \ldots, \mathcal{F}_N, C_1, \ldots, C_L, g_1, \ldots, g_G)$
8: if C $(\mathbf{p}_1, \ldots, \mathbf{p}_N) < \theta$ then
9:    return $(\mathbf{p}_1, \ldots, \mathbf{N})$
10: else if $iterations >$ MAX_ITERATIONS then
11:    return FAILURE
12: else
13:    $iterations \leftarrow iterations + 1$
14:    goto 1
15: end if
```

$$f_e(\mathbf{p}_1, \ldots, \mathbf{p}_N) = \gamma \sum_{i=1}^{L} l_i(\mathbf{p}_{v_1(i)}, \mathbf{p}_{v_2(i)}) + (1 - \gamma) \sum_{i=1}^{G} g_i(\mathbf{p}_{\mu_1(i)}, \ldots, \mathbf{p}_{\mu_{K_i}(i)}) \tag{7.21}$$

This is the error equation that is to be minimized for the global pose estimation. A number of different standard techniques are available to achieve this. In our implementation, we used the Fletcher-Reeves gradient descent algorithm.

It should be noted that the global minimum for the error function f_e is not unique. This is because both local and global constraints are only defined with respect to the relative displacements between the robot poses and the global minimum of f_e is invariant with respect to affine transforms of the poses. In practice, this problem can be solved by fixing one of the robot poses at its initial value. Then the other poses are optimized relative to this fixed pose.

7.4.3 Details of the New Optimization Algorithm

When formulating equation (7.17) for the local constraints, we remarked that the correspondences used in the error computation are obtained by applying the ICP algorithm until convergence. As we have seen above, this requires a good estimate of the initial robot poses for the ICP to converge to a global optimum. However, in general the initial estimate of the robot poses is not good enough which results in a sub-optimal set of correspondences obtained by the ICP algorithm. The same observation holds for the estimation of the global constraints: A global constraint is defined by a plane that is computed from edges found in the local maps. The global coordinates of these edges depend on the estimated robot poses, which means that a bad pose estimate yields misplaced planes and thus incorrect global constraints. This means that the definition of both the local and the global constraints depend on a good initial estimate of the robot poses. However, in experiments it turned out that even a poor initial pose estimate usually leads to a sub-optimal, though acceptable estimation of correspondences and planes which could be used to improve the robot poses in the optimization step. In other words, we have a circular dependency between constraints and robot poses. Therefore, the idea is to estimate both constraints and poses iteratively. Note that such an approach is applied in many other iterative approximation methods such as Expectation Maximization (EM), where a set of hidden random variables, that are dependent on the parameters, is estimated and fed back to the optimization of the parameters. In contrast to the maximum likelihood estimation process applied in an EM framework, our algorithm is not guaranteed to converge. However, in our experiments we found that by introducing the global constraints the iterative optimization gets more stable with respect to convergence. Algorithm 4 summarizes the individual steps of the overall pose optimization. The particular sub-routines are described as follows:

- P I : For given local constraint, i.e. an edge in the pose network, this function returns the indices of the robot poses that are connected by the edge.

- C : This function applies the ICP algorithm on the two given local maps and uses the two given robot poses as initial estimates. After convergence, the final set of correspondences is returned as pairs of indices (see the explanation of equation (7.17)).

- G C : Here, we estimate the planes for the global constraints from the current robot poses. For this, we use the edge sets $\mathcal{E}_1, \ldots, \mathcal{E}_N$ computed for each local map $\mathcal{F}_1, \ldots, \mathcal{F}_N$. The details of the edge detection and the computation of the planes are described in section 5.2.3.

- P : Using the current estimate of the robot poses, correspondences and planes this function computes a new set of robot poses by minimizing (7.21) as described above.

- C : This computes the Euclidean distance between the previous set of robot poses and the newly obtained poses. If this distance is lower than a threshold θ, the algorithm terminates.

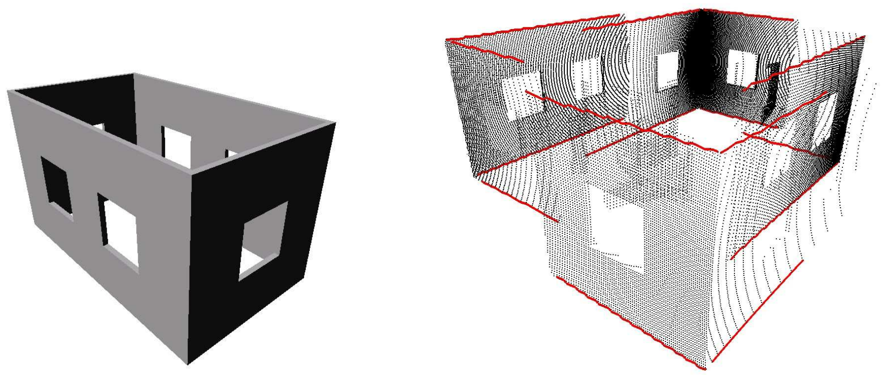


Figure 7.7: *Simulated scene used to verify the performance of the algorithm. left: 3D view of the scene; right:* 4 *generated partial views shown in an explosion view drawing.*

7.4.4 Experimental Results

To evaluate our algorithm we implemented it and tested it on real data as well as in simulation runs. The goal of the experiments reported here is to illustrate that the incorporation of global constraints increases the accuracy of the resulting models.

Real World Experiment

In order to analyze the performance of our registration algorithm in practice, we tested it with a data set taken from a real world scene. The data set consisted of 6 different three-dimensional scans taken from a building that is about 70m long, 20m wide and 11m high from the ground to the roof edge.

Fig. 7.8 shows the result of the view registration using global constraints. As can be seen, 5 different planes were detected by our algorithm. These planes were used as global constraints in the network of robot poses. The figure also depicts close-up-views of several parts of the model. Shown are the results obtained with our approach and with local constraints only. As can be seen from the figure, the edges detected in the partial views have been aligned more accurately to the planes using our approach. Especially at the roof plane we can see that the views are all at the same level. To quantify the improvement we measured the variance σ_α of the absolute angular deviation of the edges from the corresponding plane tangent. For the global-constraint based registration we obtained $\sigma_\alpha = 0.75$. In contrast, the local constraints only produce a value of $\sigma_\alpha = 2.56$.

Quantitative Results

To more systematically evaluate the quality of the models obtained with our algorithm and in comparison with other approaches we performed a series of experiments using the simulated scene shown in Figure 7.7. The object is 3m wide, 5m long and 3m high. In the particular experiment reported here we generated 4 different scans. These scans

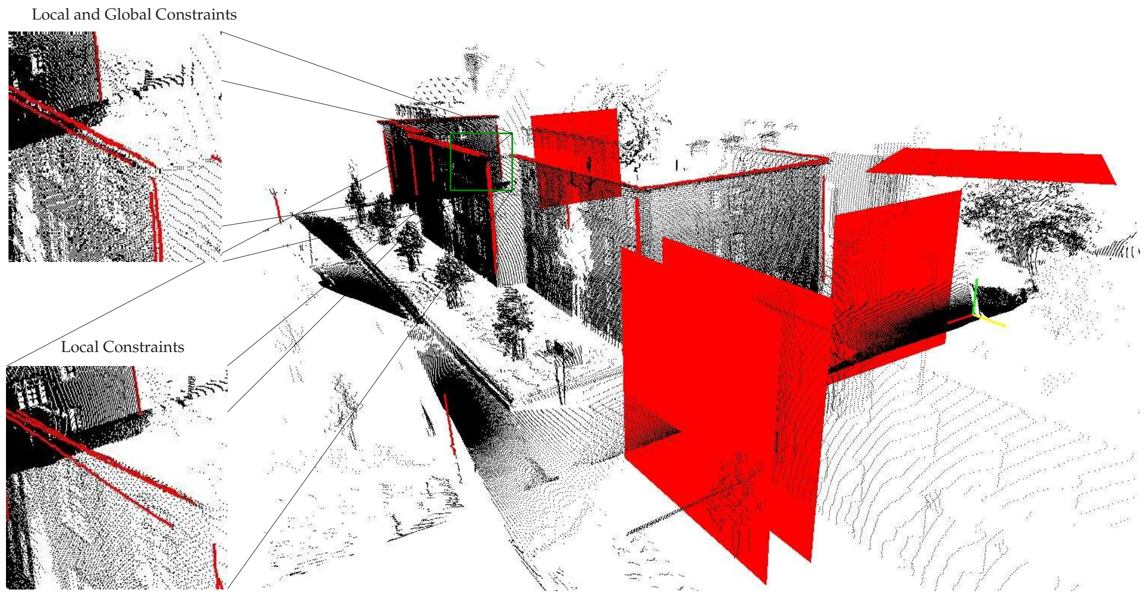


Figure 7.8: *Registered data set from an outdoor scene. Shown are the range data and the edges used to extract the planes as global structures.*

are shown in Figure 7.7 in an exploded view drawing. For the initial poses we added Gaussian noise to the poses known from the ground truth. The noise added to the angles had a variance that was different from the noise variance added to the positions. We ran two different kinds of experiments where the noise variances were (0.0005|0.05) and (0.0008|0.08) respectively. For both noise levels we started the optimization with 10 different initial sets of poses. We evaluated both the registration method using only local constraints and the one using local and global constraints.

The results for the low variance case are summarized in Table 7.1. Shown are the average deviations in angle μ_ϵ and position μ_x from the ground truth. As can be seen, the incorporation of global constraints improves the registration process. Especially the angular deviation is smaller when using global constraints. This is because the global planar constraints correct smaller errors that arise from the local scan matcher. These errors are mainly encountered in the rotations.

In the case of a high noise variance, the algorithm that uses only local constraints always diverged. This is because the local scan matching could not find enough correspondences and therefore diverged. However, in some of these cases convergence could be achieved by adding the global constraints.

In a further experiment we demonstrate the performance of our approach in situations where few overlap is given between the individual partial views. Again, we used the scene shown in fig. 7.7. We ran the registration process using different numbers of partial views, ranging from 6 down to 2, where the overlap decreased with the number of views. Fig. 7.9 plots the rotational deviations from the ground truth. As can be seen, our algorithm performs significantly better ($\alpha = 0.05$) in situations with fewer overlap

Table 7.1: *Comparison of performance for different registration algorithms with respect to average angular deviation μ_ϵ (in radians) and positional deviation μ_x (in meters)*

Method	μ_ϵ	μ_x
local constr.	0.0175	0.1595
global + local constr.	0.0071	0.1109

between the individual views.

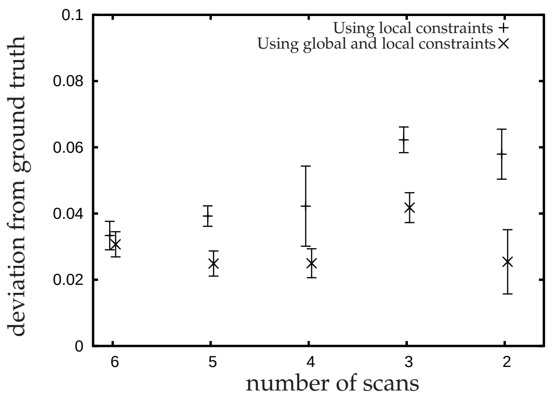


Figure 7.9: *Statistical analysis of the registration process with respect to the angular rotation from the ground truth*

7.5 Conclusions

In this chapter, we showed how consistent 3D environment maps can be constructed from a set of local maps that were acquired at different robot positions. In particular, we presented an efficient variant of the Iterative Closest Point (ICP) algorithm and a way to adapt this algorithm to the problem of registering MLS maps, which have been presented in chapter 6. In addition, we investigated the problem of finding an optimal set of robot poses where a local map representation was given for each robot pose and during the trajectory of the robot areas are visited more than once, which is usually called loop-closing. As a major contribution of this work, we extended the standard framework of the robot pose network by introducing global constraints based on the observation that many man-made structures have features that can be observed in different views (e.g. the windows of a building). In experiments we showed that by exploiting this additional knowledge the loop-closing can be performed even in cases where the local maps have low overlap. In addition, the global constraints are helpful in map registration, even if the robot trajectory is not a loop.

Part III

3D Object Recognition

Try to learn something about everything and everything about something.

Thomas Huxley (1825 - 1895)

8 Object Classification by Supervised Learning

8.1 Introduction

So far, we have been dealing with the first major part of the robot perception task, which involves the accurate, compact and consistent representation of the input data. This is usually called *mapping*. However, as explained in chapter 1, perception also includes the semantic annotation of the data obtained from the sensors. We will refer to this as *scene analysis* or *object recognition*. The main concept behind this is that of *classification*: For each element $\mathbf{x}$ of a given set $\mathcal{X}$ of input data we search a class label y which belongs to a set $\mathcal{Y}$ of possible labels. The mapping between $\mathcal{X}$ and $\mathcal{Y}$ is called a classifier. Such a classifier can be obtained by using supervised learning from a previously labeled input data set. We will describe this briefly in section 8.2. We will also see that the approaches that exist to do supervised learning can be subdivided into three main categories, namely the *discriminant function*, the *discriminative model* and the *generative model* approach. There is a large number of different learning algorithms that can be found in the literature, where good summaries are given in the textbooks by Theodoridis and Koutroumbas [2003] and Bishop [2006]. From these, we will briefly describe three very common approaches which we will use later in chapter 9 for comparison and extension towards a new object classification algorithm. These approaches are the naïve Bayes classifier (NBC) in Section 8.3, which follows the generative model, the nearest-neighbor classifier (NN) in Section 8.4, which uses a discriminative model, and the AdaBoost algorithm in Section 8.5, which learns a discriminant function. A summary and the conclusions are given in Section 8.6.

8.2 Supervised Learning

Assume we have a set $\mathcal{P}$ of input data that consists of N data points. This can, for example, be a point cloud as described in Section 4.2. Our goal is to find a class label y_n, where $n = 1, \dots, N$ for each data point $\mathbf{p}_1, \dots, \mathbf{p}_N \in \mathcal{P}$. Usually, this is not done directly on the input space, but rather on a set $\mathcal{X}$ of *features* that are computed from the input data. This has the advantage that the labeling does not depend on the particular 3D coordinates of the data points. Instead, local properties of the environment such as planarity or smoothness can be expressed with the features, which makes them more useful for classification. If we assume that a vector $\mathbf{x}_n$ of M features is computed for each data point $\mathbf{p}_n$, we can define a classifier as follows:

Definition 8.1. A *hypothesis* or a *classifier* is a mapping from a set $\mathcal{X} \subset \mathbb{R}^M$ of N feature vectors $\mathbf{x}_1, \dots, \mathbf{x}_N$ to a set $\mathcal{Y} = \{1, 2, \dots, K\}$ of class labels $y_1, \dots, y_N$.

A common way to obtain a classifier is known from the area of machine learning as *supervised learning* (see for example Bishop [2006]), where a training data set $\widehat{\mathcal{X}}$ is given, for which the assigned class labels $\widehat{\mathcal{Y}}$ are known. Then, a classifier is derived from this training data set. There exist three distinct approaches to do this:

- The *discriminant function* $h(\mathbf{x})$: This function maps from $\mathcal{X}$ to $\mathcal{Y}$ and is therfore a classifier. An example is Fisher's linear discriminant, which maximizes the ratio of the between-class covariance to the within-class covariance of the training data set. Details are described in Bishop [2006].

- The *discriminative model* $p(y \mid \mathbf{x})$: This is a probabilistic approach that models the conditional probability of a label $\hat{y}$ for a given feature vector $\hat{\mathbf{x}}$ from the training set. Then, a label can be found for a new input vector $\mathbf{x}$ by maximizing this conditional probability.

- The *generative model* $p(y, \mathbf{x})$: This approach is also probabilistic, but it is more complex than the discriminative model. It seeks to model the joint distribution of features and labels or, likewise, the probability $p(\mathbf{x} \mid y)$ and the class prior $p(y)$. The posterior $p(y \mid \mathbf{x})$ is then obtained using Bayes' rule.

There is an ongoing discussion in the literature whether to favor discriminative or generative models when using a probabilistic approach, while the tendency seems to be towards discriminative models (see for example Goutte et al. [2004], Ng and Jordan [2002], Ulusoy and Bishop [2005]). In general, we can say that the generative model is more powerful, because it can generate new data points by sampling from the learned distribution $p(y, \mathbf{x})$. It also seems to give better results when the training data set is small. However, if enough training examples are given, discriminative approaches seem to perform better (see Ng and Jordan [2002]). Compared to these two, the non-probabilistic discriminant function approach is less complex, but it is not capable of modeling the posterior $p(y \mid \mathbf{x})$, which is useful and needed in many situations.

8.2.1 Formulation of the Followed Approach

For the scope of this work, we will focus on the discriminative model, but we will also compare the results of our discriminative classifier presented in chapter 9 to the naïve Bayes classifier (NBC) described in Section 8.3, which is generative, and to AdaBoost (see Section 8.5), which can be viewed as a discriminant function learner. Thus, in this work we assume we are given a model description of the posterior distribution $p_\omega(y_1, \dots, y_N \mid \mathbf{x}_1, \dots, \mathbf{x}_N)$ where ω is a set of parameters defining the actual posterior distribution. As mentioned above, the posterior is obtained from a training data set $\widehat{\mathcal{X}}$ of $\hat{N}$ examples with known labels $\widehat{\mathcal{Y}}$. This is done by finding the model parameters ω that maximize $p_\omega(\hat{y}_1, \dots, \hat{y}_{\widehat{N}} \mid \hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_{\widehat{N}})$. We will refer to this as the *learning step*. Using the parameters ω obtained in the learning step, we can find a set of labels $y_1, \dots, y_N$ for a given new set of features $\mathbf{x}_1, \dots, \mathbf{x}_N$ so that the posterior $p_\omega(y_1, \dots, y_N \mid \mathbf{x}_1, \dots, \mathbf{x}_N)$ is maximized. This means that we have a maximum-a-posteriori (MAP) formulation. To simplify the notation, we will collect all feature vectors $\mathbf{x}_1, \dots, \mathbf{x}_N$ in one vector $\mathbf{x}$ and all class labels $y_1, \dots, y_N$ in a vector $\mathbf{y}$. Then, we can summarize the two steps performed in supervised learning as

$$\begin{aligned} \textbf{learning:} \quad & \text{given } \hat{\mathbf{y}} \text{ and } \hat{\mathbf{x}}, \text{ find } \omega^* = \operatorname*{argmax}_\omega p_\omega(\hat{\mathbf{y}} \mid \hat{\mathbf{x}}) \\ \textbf{inference:} \quad & \text{given } \omega^* \text{ and } \mathbf{x}, \text{ find } \mathbf{y}^* = \operatorname*{argmax}_\mathbf{y} p_{\omega^*}(\mathbf{y} \mid \mathbf{x}) \end{aligned} \tag{8.1}$$

In the following sections we will describe three standard approaches to do supervised learning: the generative naïve Bayes classifier (NBC), the discriminative nearest-neighbor classifier (NN), and the AdaBoost algorithm.

8.3 The Naïve Bayes Classifier

The idea of the naïve Bayes classifier is to find a model of the likelihood $p_\omega(\mathbf{x}_n \mid y_n)$ and the prior $p(y)$. Thus, for each class label $k = 1, \dots, K$ and each data point $\mathbf{p}_n$ that has an associated feature vector $\mathbf{x}_n \in \mathbb{R}^M$ of size M we know $p_\omega(\mathbf{x}_n \mid y_n = k)$ and $p_\omega(y_n = k)$. From this, we can compute the posterior $p_\omega(y_n \mid \mathbf{x}_n)$ using Bayes' rule:

$$p_\omega(y_n \mid \mathbf{x}_n) = \frac{p_\omega(\mathbf{x}_n \mid y_n) p(y_n)}{\sum_{k=1}^K p_\omega(\mathbf{x}_n \mid y_n = k) p(y_n = k)}. \tag{8.2}$$

Usually, a Gaussian distribution is used to model the likelihood. This means that the parameters ω are defined as the mean vectors $\boldsymbol{\mu}_k$ and the covariance matrices Σ_k for each class $k = 1, \dots, K$. Thus, we can formulate the likelihood as

$$\begin{aligned} p_\omega(\mathbf{x}_n \mid y_n = k) &= \frac{1}{(2\pi)^{M/2} \sqrt{\det(\Sigma_k)}} \exp\left(-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right) \\ \text{where} \quad \omega &= (\boldsymbol{\mu}_1, \Sigma_1, \dots, \boldsymbol{\mu}_K, \Sigma_K) \end{aligned} \tag{8.3}$$

In the learning task the individual class means and covariances ($\boldsymbol{\mu}_k$, Σ_k) are computed. This can be done by determining the sample mean and covariance matrix from the training data set (see Theodoridis and Koutroumbas [2003]). For the prior $p(y)$ we can either assume a uniform distribution or estimate the probability $p(y) = k$ for each class k from the fraction of data points that belong to class k.

In the inference task we search for labels y_n that maximize the posterior (8.2). This is equal to maximizing the numerator of the right hand side of 8.2, because the denominator does not depend on y_n. If we assume uniform priors, the inference task reduces further to maximizing the likelihood term $p_\omega(\mathbf{x}_n \mid y_n)$. This means that for each new data point $\mathbf{x}_n$ we simply compute the likelihood (8.3) for each class and assign it the label of the class with maximal likelihood.

8.3.1 Discussion

The NBC is optimal in the sense that it minimizes the classification error probability (see Theodoridis and Koutroumbas [2003]). It performs well in cases where the assumption of a Gaussian distribution of the feature vectors for each class is reasonable. However, in many cases this does not hold. For example, if the feature vectors of a given class are distributed in a multi-modal distribution, then the use of the class mean and covariance matrix results in a bad approximation. Also, the class borders learned by the NBC are restricted to be second order curves, which also reduces the flexibility of the classification approach.

8.4 The Nearest-Neighbor Classifier

As mentioned above, the Nearest-Neighbor classifier (NN) is a discriminative model. The idea of NN is to estimate the posterior directly from the training data by looking at a set of l points in the training data set that are neighbors of an input data point $\tilde{\mathbf{x}}$ to be labeled. From these neighbors the local posterior $p_\omega(y \mid \tilde{\mathbf{x}})$ is estimated by computing the fractions of neighboring points that belong to the particular classes. In the learning phase, the NN classifier does not compute any parameters like the NBC, but instead simply stores the entire training data set as parameters ω. Thus, the classification uses the particular instances of data points from the training data set, which is why the NN classifier is called an *instance-based* classifier.

To describe the NN classifier more formally using the general framework of supervised learning from equation (8.1), we first define the local neighborhood $\mathfrak{N}$ of size l for a data point $\tilde{\mathbf{x}}$ and a given data set $\mathcal{X}$ as

$$\begin{aligned} &\mathfrak{N}(l, \tilde{\mathbf{x}}; \mathcal{X}) := \{\mathbf{x} \in \mathcal{X} \mid \mathrm{card}(\mathfrak{R}(\tilde{\mathbf{x}}, \mathbf{x}, \mathcal{X})) \leq l\} \\ &\text{where} \quad \mathfrak{R}(\mathbf{x}_1, \mathbf{x}_2, \mathcal{X}) := \{\mathbf{x} \in \mathcal{X}, \mathbf{x} \neq \mathbf{x}_1 \mid d(\mathbf{x}, \mathbf{x}_1) \leq d(\mathbf{x}_2, \mathbf{x}_1)\} \end{aligned} \tag{8.4}$$

Here we used the function 'card()' that returns the number of elements or the *cardinality* of a given finite set. Using (8.4) we can estimate the posterior distribution as

$$p_\omega(y_n = k \mid \mathbf{x}_n) = \frac{\text{card}(\{\hat{\mathbf{x}} \in \mathfrak{N}(l, \mathbf{x}_n, \hat{\mathcal{X}}) \mid \hat{y} = k\})}{l}, \qquad k = 1, \dots, K \tag{8.5}$$

where $\omega = (\hat{\mathbf{x}}_1, \hat{y}_1, \dots, \hat{\mathbf{x}}_{\widehat{N}}, \hat{y}_{\widehat{N}})$ is the parameter set obtained in the learning step. The parameter l defines the size of the neighborhood and should not be a multiple of the number of classes K (see Theodoridis and Koutroumbas [2003]). In the simplest case, l is set to 1, which means that the classifier always assigns the label of the closest training example in feature space.

8.4.1 Discussion

The major disadvantages that have been mentioned for the Bayes classifier do not apply for the nearest-neighbor classifier. In particular, the NN classifier does not assume any model for the distribution of the feature vectors. Thus, it can handle arbitrary distributions in the feature space. We can see this as a sample-based representation of the posterior distribution, where the samples are the training examples, also called *instances*. This directly shows us the major drawbacks of the NN approach, namely the need of storing many training instances to obtain an accurate representation of the posterior distribution. Especially in cases where the feature space is high-dimensional this causes problems, because the number of required training instances grows exponentially with the dimension of the feature space. This is sometimes referred to as the *curse of dimensionality*.

8.5 AdaBoost

The *AdaBoost* algorithm was introduced by Freund and Schapire [1996, 1997] as a special case of boosting. The idea of boosting is to generate a classifier out of a set of given base classifiers, that only give poor classification results. In the context of learnability, these base classifiers are usually called *weak* classifiers, while the classifier that results after boosting is a *strong* classifier. Intuitively, the distinction of strong and weak learning is that a strong learner finds with high probability a classifier that is arbitrarily accurate. A weak learner in contrast, has only a bounded accuracy. For example, an algorithm that is only slightly better than random guessing is a weak learner. The concept of boosting is based on Valiant [1984] who states that it is possible to construct a strong learner from any given weak learner. A more formal definition of learnability and more details on this can be found in Jain et al. [1999].

Algorithm 5 shows the AdaBoost algorithm for a 2-class problem. Here, it is assumed that we are given a set of T weak classifiers $h_1, \dots, h_T$ and a training data set $\widehat{\mathcal{X}}$ with known labels $\widehat{\mathcal{Y}}$. Each weak classifier returns either -1 for the first class or 1 for the second class. The main idea of the algorithm is to introduce a vector $\mathbf{w}$ of importance weights for each data point from the training data set (see line 1). These weights are adapted in each iteration of the algorithm, where the loop runs over all weak classifiers that are to be learned. In the first step of each iteration t, here shown in line 3, the weak classifier h_t is obtained from the training data with the current weight vector $\mathbf{w}^t$ for the

Algorithm 5 AdaBoost – Compute a strong classifier h_f from a set of weak classifiers h_t

```
Input:
      labeled training examples $(\widehat{\mathcal{X}}, \widehat{\mathcal{Y}}) = \{(\hat{\mathbf{x}}_1, \hat{y}_1), \ldots, (\hat{\mathbf{x}}_{\widehat{N}}, \hat{y}_{\widehat{N}})\}$
      number of weak classifiers $T$
1: Initialize weights $\mathbf{w}^1 \in \mathbb{R}^{\widehat{N}}$ such that $w_n^1 = 1/\widehat{N}$ for all $n = 1, \ldots, \widehat{N}$
2: for $t = 1, \ldots, T$ do
3:     $h_t \leftarrow$ L $(\mathbf{w}^t, \mathcal{Z})$
4:     $$\epsilon_t \leftarrow \frac{\sum_{i=1}^{\widehat{N}} w_i^t \, |h_t(\hat{\mathbf{x}}_i) - \hat{y}_i|}{2 \sum_{i=1}^{\widehat{N}} w_i^t}$$
5:     $$\alpha_t \leftarrow \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$
6:     $$w_n^{t+1} \leftarrow w_n^t \exp\left(\alpha_t \frac{|h_t(\mathbf{x}_i) - y_i|}{2}\right)$$
7: end for
8: $$h_f(\mathbf{x}) \leftarrow \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$
```

data points. Then, in line 4, the performance of h_t on the training data is measured using the weighted average error ϵ_t. Here, we have to divide by two, because the absolute value of the difference between $h_t(\hat{\mathbf{x}}_i)$ and $\hat{y}_i$ can be either 0 or 2 and we want to have a bounded error measure between 0 and 1. Based on this error ϵ_t, we compute a coefficient α_t for the classifier h_t in line 5. This coefficient is small for big errors and large when the error is small. In line 6, the weights for the data points are updated according to the classification error of the current classifier. This way, the data points that have been classified wrong by h_t will have a higher weight for the next weak classifier h_{t+1}. The resulting strong classifier h_f is shown in line 8. It consists of a weighted sum of the weak classifiers, where the weights are the learned coefficients α_t.

8.5.1 Multiple Classes

The AdaBoost algorithm, as shown here, is designed for a 2-class problem. However, it can be extended for cases with multiple classes. This can be done, for instance, by concatenating several binary classifiers that distinguish one class from all others to obtain a *sequential* classifier. Here, the order in which the particular binary classifiers are applied is important for the performance of the overall classifier. Also, a more elaborate technique named AdaBoost.M2 has been proposed by Freund and Schapire [1995] to extend the binary AdaBoost to multiple classes. At this point, we will not discuss these extensions in more detail, but we instead refer to the masters thesis by Martínez Mozos [2004].

8.5.2 Discussion

The strength of the AdaBoost algorithm lies in its ability to use a set of classifiers that perform comparably bad on the training data set to come up with a classification method, that is adapted to the misclassified data points and performs better than all single classifiers. Thus, it is particularly useful in cases where many features can be extracted from the data points, which can be turned into weak classifiers by simple thresholding. In this case, the subroutine *weakLearn* in line 3 of algorithm 5 returns the threshold that minimizes the classification error when using the feature with index t.

However, as mentioned already, the AdaBoost algorithm can be viewed as a discriminant function learner, because it only returns a direct mapping h_f from the feature space $\mathcal{X}$ to the set of class labels $\mathcal{Y}$. There is no means to represent the uncertainty of the classification.

8.6 Conclusion

In this chapter, we gave a short overview of existing techniques for object classification using supervised learning. In particular, we described the three standard approaches naïve Bayes, nearest-neighbor and AdaBoost, which all are examples of different supervised learning approaches. None of these approaches can be viewed as the method of choice for all classification tasks. In fact, the choice of a classification algorithm depends on the application for which it is needed, which is a major issue in the context of object classification and pattern recognition. We refer to Theodoridis and Koutroumbas [2003] for a more detailed discussion on this.

Despite the fact that all three approaches give good results in certain situations, they all have in common that they operate on each input data point independently and do not take into account the possible labeling of data points in the vicinity of the query data point. This often leads to unrealistic classification results, because the features extracted from nearby data points maybe different from each other due to noise or imperfections during the feature extraction, although the probability that they belong to the same object class is comparably high. For example, if we want to distinguish the two object classes ’wall’ and ’door’ in a 3D point cloud of a building and we consider the local planarity of a scan point as a feature, it may happen that the class label ‘wall’ is assigned to a data point, although all other points in the vicinity of this point belong to the class ‘door’. Methods that additionally incorporate the information of neighboring data points for classification are called *collective classification* methods (see Chakrabarti and Indyk [1998]). They will be the subject of the next chapter.

It is dangerous to be right in matters on which the established authorities are wrong.

Voltaire (1694-1778)

9

Collective Classification

9.1 Introduction

In the previous chapter, we summarized some of the most common techniques for the task of classification by supervised learning. One property that is common to all these approaches is that the classification decision is made only on the basis of the *local evidence* of the particular class label. This means that for any data point $\mathbf{z}_i$, the assignment of a special class label only depends on the information that is available for $\mathbf{z}_i$. This local information is usually collected in a feature vector $\mathbf{x}_i$. Statistically, we can say that the label y_i that is to be assigned to the data point $\mathbf{z}_i$ is assumed to be independent of the label assigned to any other data point $\mathbf{z}_j$ in the vicinity of $\mathbf{z}_i$. Or, in other words, the conditional probability $p(y_i \mid \mathbf{x}_i)$ is assumed to be equal to the probability $p(y_i \mid \mathbf{x}_i, y_j)$ for $i \neq j$.

However, in most situations, this independence assumption is invalid. As a simple example, consider the case where the classification of a set of 3D points is made based on the orientation of the local normal vector at each particular data point. If we assign the label 'floor' to each data point where the normal vector points upwards and the label 'wall' where the normal vector points to the side, it may happen due to noise in the data, that a data point is labeled as 'floor', even though all its neighboring points are labeled as 'wall'. Thus, we will have an incorrect labeling. To avoid such situations, we will apply the concept of *collective classification*, in which the statistical dependency between labels of neighboring data points is also considered. This is the subject of the remainder of the chapter.

The chapter is organized as follows. In section 9.2 we present the major concepts of associative Markov networks (AMNs), which are a special instance of relational Markov random fields (see Taskar et al. [2002]). AMNs have been introduced by Taskar [2004] as a collective classification framework, for which efficient learning and inference algorithms can be formulated. In section 9.3 we show how the learning task can be performed even more efficiently by appropriately reducing the training data set. In experiments, we will see that this data reduction has no influence on the good classification results. In section 9.4, we further extend the AMN based classification method by introducing a feature transform τ, which is applied to the features before feeding them into the AMN. This improves the classification result, because the feature space becomes better separable by hyper planes. The improvement will be visible in our experiments. Finally, we conclude the chapter in section 9.5.

9.2 Associative Markov Networks

In this section, we will describe the concept of associative Markov Networks (AMNs). They have been introduced by Taskar [2004] and are a special case of conditional Markov Random Fields, which were briefly presented in section 3.4.2. As we saw there, a conditional Markov Random Field is an undirected graph with a set of cliques C and a *clique potential* ϕ_c, which is a non-negative function associated to each $c \in C$. It defines the distribution

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{\prod_{c \in C} \phi_c(\mathbf{x}_c, \mathbf{y}_c)}{\sum_{\mathbf{y}'} \prod_{c \in C} \phi_c(\mathbf{x}_c, \mathbf{y}'_c)} \tag{9.1}$$

where $\mathbf{x}_c$ and $\mathbf{y}_c$ are the features and labels of all nodes in the clique c. Here, the potential ϕ_c is a mapping from features and labels to a positive value. This value is often called the *compatibility* between the features and the labels of the data points in c. The higher the compatibility is, the more likely it is that the labels $\mathbf{y}_c$ are correct for the features $\mathbf{x}_c$.

The denominator in equation (9.1) is called the *partition function*, usually denoted Z, and is essentially a sum over all possible labelings. In all but the simplest cases the calculation of the partition function constitutes the major problem in the learning task because of its exponential complexity. We will later see how learning can be done without calculating Z.

In AMNs, the size of the cliques is usually restricted to be either one or two. This results in a *pairwise* MRF, where only *node* and *edge potentials* φ and ψ are considered. For a pairwise MRF with the set of edges $E = \{(ij) \mid i < j\}$ equation (9.1) simplifies to

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{N} \varphi(\mathbf{x}_i, y_i) \prod_{(ij) \in E} \psi(\mathbf{x}_{ij}, y_i, y_j). \tag{9.2}$$

Again, $Z = \sum_{\mathbf{y}'} \prod_{i=1}^{N} \varphi(\mathbf{x}_i, y'_i) \prod_{(ij) \in E} \psi(\mathbf{x}_i, y'_i, y'_j)$ denotes the partition function. Note that in equation (9.2) there is a distinction between node features $\mathbf{x}_i \in \mathbb{R}^{d_n}$ and edge features $\mathbf{x}_{ij} \in \mathbb{R}^{d_e}$. Thus, the number d_n of node features and the number d_e of edge features is not necessarily the same.

It remains to describe the potentials φ and ψ. As mentioned above, the potentials reflect how well the features fit to the labels. One simple way to define the potentials is the *log-linear* model (see Taskar et al. [2004]). In this model, a weight vector $\mathbf{w}^k$ is introduced for each class label $k = 1, \ldots, K$. The node potential φ is then defined so that $\log \varphi(\mathbf{x}_i, y_i) = \mathbf{w}_n^k \cdot \mathbf{x}_i$ where $k = y_i$. Accordingly, the edge potentials are defined as $\log \psi(\mathbf{x}_{ij}, y_i, y_j) = \mathbf{w}_e^{k,l} \cdot \mathbf{x}_i$ where $k = y_i$ and $l = y_j$. Note that there are different weight vectors $\mathbf{w}_n^k \in \mathbb{R}^{d_n}$ and $\mathbf{w}_e^{k,l} \in \mathbb{R}^{d_e}$ for the nodes and edges.

For the purpose of convenience we use a slightly different notation for the potentials, namely

$$\log \varphi(\mathbf{x}_i, y_i) = \sum_{k=1}^{K} (\mathbf{w}_n^k \cdot \mathbf{x}_i) y_i^k \tag{9.3}$$

$$\log \psi(\mathbf{x}_{ij}, y_i, y_j) = \sum_{k=1}^{K} \sum_{l=1}^{K} (\mathbf{w}_e^{k,l} \cdot \mathbf{x}_{ij}) y_i^k y_j^l, \tag{9.4}$$

where y_i^k is an indicator variable which is 1 if point p_i has label k and 0, otherwise.

Additionally, a generalized version of the Potts model (see Potts [1952]) is applied for the potentials in an AMN. This is done by introducing the constraints $\mathbf{w}_e^{k,l} = \mathbf{0}$ for $k \neq l$ and $\mathbf{w}_e^{k,k} \geq \mathbf{0}$ and results in $\psi(\mathbf{x}_{ij}, k, l) = 1$ for $k \neq l$ and $\psi(\mathbf{x}_{ij}, k, k) = \lambda_{ij}^k$, where $\lambda_{ij}^k \geq 1$. The idea here is that edges between nodes with different labels should be penalized over edges between equally labeled nodes.

9.2.1 Learning and Inference with AMNs

In this section, we describe how learning and inference can be done with AMNs according to equation (8.1). In a first step, we reformulate the problem so that, instead of maximizing $P_\omega(\mathbf{y} \mid \mathbf{x})$, we maximize $\log P_\omega(\mathbf{y} \mid \mathbf{x})$. The parameters ω are represented by the weight vectors $\mathbf{w} = (\mathbf{w}_n, \mathbf{w}_e)$. By plugging in equations (9.3) and (9.4), we obtain

$$\max \sum_{i=1}^{N} \sum_{k=1}^{K} (\mathbf{w}_n^k \cdot \mathbf{x}_i) y_i^k + \sum_{(ij) \in E} \sum_{k=1}^{K} (\mathbf{w}_e^k \cdot \mathbf{x}_{ij}) y_i^k y_j^k - \log Z_{\mathbf{w}}(\mathbf{x}). \tag{9.5}$$

Note that the partition function Z only depends on $\mathbf{w}$ and $\mathbf{x}$, but not on the labels $\mathbf{y}$.

Learning

The problem arising in the learning task is that the partition function Z depends on the weights $\mathbf{w}$. This means that when maximizing $\log P_{\mathbf{w}}(\hat{\mathbf{y}} \mid \mathbf{x})$ the intractable calculation of Z needs to be done for each $\mathbf{w}$. However, if we instead maximize the *margin* between the optimal labeling $\hat{\mathbf{y}}$ and any other labeling $\mathbf{y}$ defined by

$$\log P_\omega(\hat{\mathbf{y}} \mid \mathbf{x}) - \log P_\omega(\mathbf{y} \mid \mathbf{x}), \tag{9.6}$$

the term $Z_{\mathbf{w}}(\mathbf{x})$ cancels out and the maximization can be done efficiently. This method is referred to as *maximum margin* optimization. The details of this formulation are omitted here for the sake of brevity. We only note that the problem is reduced to a quadratic program (QP) of the form:

$$\begin{aligned}
&\min \frac{1}{2}\|\mathbf{w}\|^2 + c\xi && (9.7)\\
&\text{s.t. } \mathbf{w}\mathbf{X}\hat{\mathbf{y}} + \xi - \sum_{i=1}^{N} \alpha_i \geq N; \quad \mathbf{w}_e \geq 0; \\
&\alpha_i - \sum_{ij,ji\in E} \alpha_{ij}^k - \mathbf{w}_n^k \cdot \mathbf{x}_i \geq -\hat{y}_i^k, \ \forall i,k; \\
&\alpha_{ij}^k + \alpha_{ji}^k - \mathbf{w}_e^k \cdot \mathbf{x}_{ij} \geq 0, \ \alpha_{ij}^k, \alpha_{ji}^k \geq 0, \ \forall ij \in E, k
\end{aligned}$$

Here, the variables that are solved for in the QP are the weights $\mathbf{w} = (\mathbf{w}_n, \mathbf{w}_e)$, a slack variable ξ and additional variables α_i, α_{ij} and α_{ji}. Again, we refer to Taskar et al. [2004] for details.

Inference

Once the optimal weights $\mathbf{w}$ are calculated, we can do inference on an unlabeled test data set. This is done by finding the labels $\mathbf{y}$ that maximize $\log P_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x})$. As mentioned above, Z does not depend on $\mathbf{y}$ so that the maximization in equation (9.5) can be done without considering the last term. With the constraints imposed on the variables y_i^k this leads to a linear program of the form

$$\begin{aligned}
&\max \sum_{i=1}^{N}\sum_{k=1}^{K}(\mathbf{w}_n^k \cdot \mathbf{x}_i)y_i^k + \sum_{ij\in E}\sum_{k=1}^{K}(\mathbf{w}_e^k \cdot \mathbf{x}_{ij})y_{ij}^k && (9.8)\\
&\text{s.t. } y_i^k \geq 0, \ \forall i,k; \ \sum_{k=1}^{K} y_i = 1, \ \forall i \\
&y_{ij}^k \leq y_i^k, \ y_{ij}^k \leq y_j^k, \ \forall ij \in E, k
\end{aligned}$$

Here, we introduced variables y_{ij}^k representing the labels of two points connected by an edge. The last two inequality conditions are a linearization of the constraint $y_{ij}^k = y_i^k \wedge y_j^k$.

9.3 Reduction of the Training Data

Unfortunately, the learning task we described in the previous section is computationally expensive in run time as well as in memory requirement. For each scan point there is one variable and one constraint in the quadratic program (9.7). Furthermore, we have two variables and two constraints per edge. This results in a large computational effort; Anguelov et al. [2005] report one hour run time for about 30,000 scan points. Fortunately, in usual data sets, a huge part of the data is redundant. For instance, to reduce the data

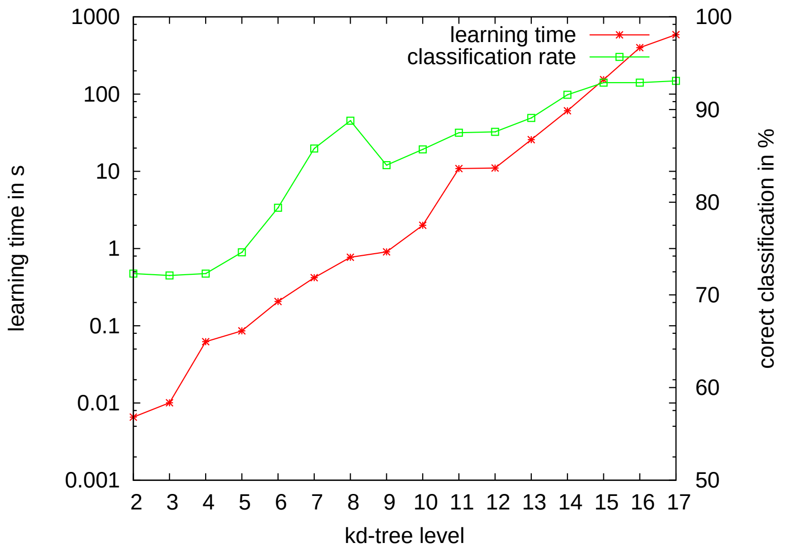


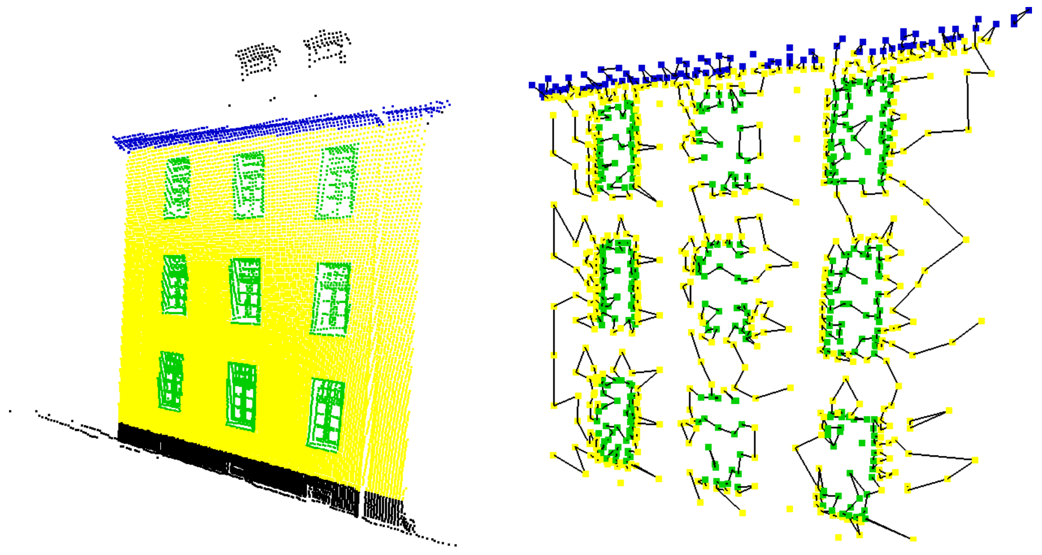
Figure 9.1: *Processing time for the learning task (red with crosses) and classification rate (green with boxes) for different values of d_{max}. The left y-axis is for the time and is shown in log-scale, the right one is for the classification results.*

set we can randomly draw a smaller set of scan points from the whole scan. In our experiments this gave good results. The run time dropped down from about 20 minutes down to less than a minute while the detection rate was still around 92%. However, it is not clear how many samples are necessary to obtain good detection rates, because this depends on the data set. A scene with many small objects should not be down-sampled as much as a scene with only few, big objects. Therefore we reduce the data *adaptively*.

9.3.1 Adaptive Reduction

The idea of adaptive data reduction is to obtain as much information as necessary from the training data so that still a good recognition rate can be achieved. This is dependent on the data set, so we need an adaptive data structure. One way to adaptively store 3D data is by means of *kD*-trees, as we have seen in section 4.3.1. When using *kD*-trees we can reduce the data set by considering only scan points in the tree that are stored in leaf nodes up to a given maximum depth d_{max}. All points in deeper branches are then merged into a new leaf node of depth d_{max}. The data point in this new leaf node is calculated as the mean of all points from the corresponding subtree. Apart from the reduction in the data complexity, this has the advantage of a sampling that is less dependent on the data density. The only question here is how to select d_{max}.

To investigate the influence of the maximal tree depth we ran the recognition process with different values for d_{max}. Figure 9.1 shows the time the training process took and the corresponding classification rate for each value of d_{max}. What can be seen from the figure is that the processing time grows exponentially whereas the recognition rate does



a. *Training data* **b.** *AMN of reduced data*

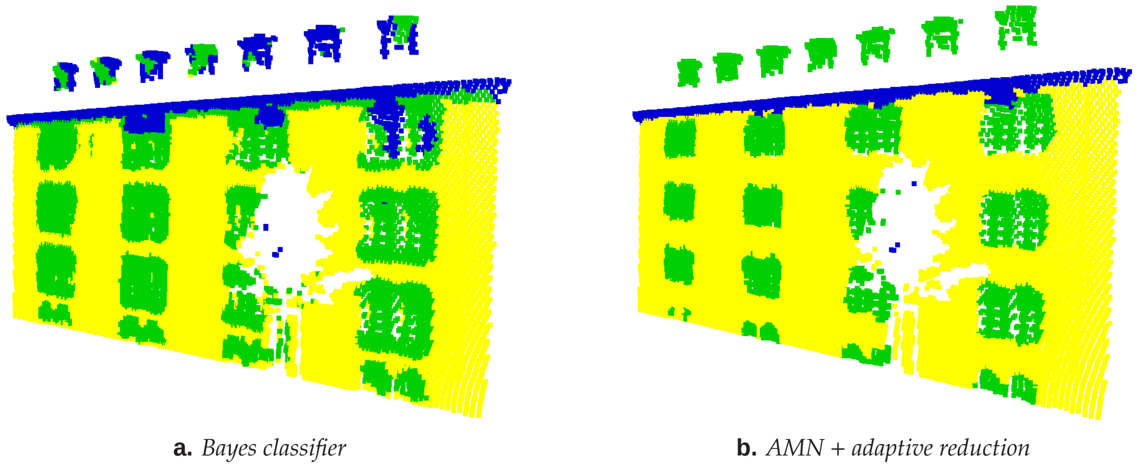
Figure 9.2: *Data set used for training (fig. a). The black points are unlabeled and are not considered in the training process. Fig. b shows the resulting AMN after reducing the training data. By applying the adaptive reduction, the borders between labeled regions are emphasized while areas in which the labels do not change are represented in a higher level of abstraction.*

not improve for d_{max} higher than 15. For $d_{max} = 15$ we obtained a classification rate of 92.9% while the run time for the training was only 2.5 minutes. In this case, the training set consisted of 6558 points.

9.3.2 Parameterless Reduction

When using *kD*-trees to reduce the training data in the described way, it still remains to find a good value for d_{max}. As for the uniform down-sampling, this is dependent on the data set. In our current system, we therefore modify the reduction algorithm so that it is parameter-free: We still use a *kD*-tree to store the data points, but instead of pruning at a fixed level, we merge all points in a subtree whenever all of its labels are equal. The idea here is that for large homogeneous areas, where all points have the same label, we can assume a higher level of abstraction as in heterogeneous areas.

Figure 9.2 shows an example of a training set that was reduced in this way. The original data set is shown in figure 9.2a. The reduced set consists only of 919 points, while the original scan contained 16, 917 labeled points. As the next section will show, training on a data set, which was reduced this way, took only a few seconds. This is a substantial speed-up without a serious reduction of the classification performance.



a. *Bayes classifier* **b.** *AMN + adaptive reduction*

Figure 9.3: *Results of the inference on a test data set, where the learning was performed on the training data from figure 9.2a. In figure (a) we see the classification result using Bayes classification. Especially at the borders between classes the classification is poor. In contrast, the AMN classifier reduces this problem as shown in figure (b).*

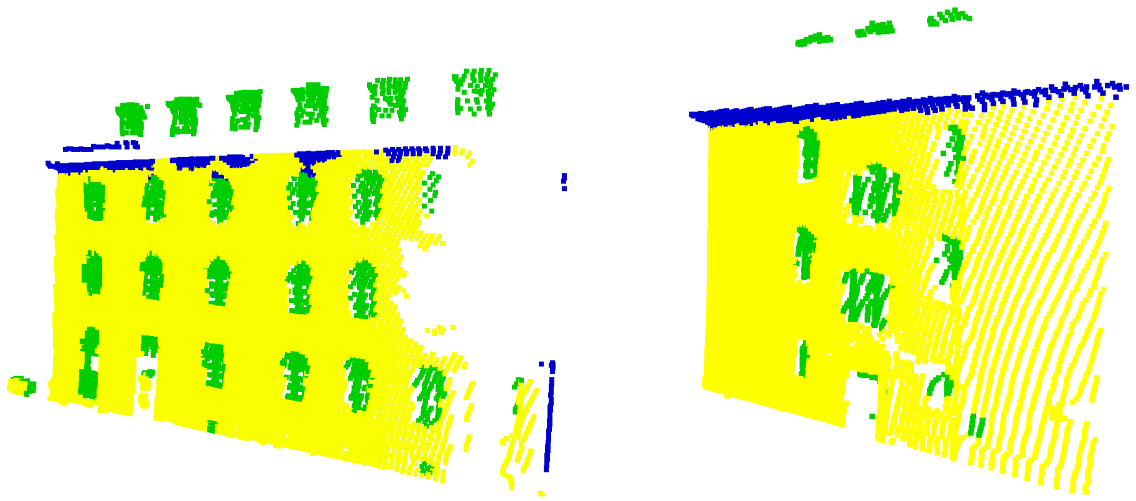


Figure 9.4: *Additional results on two other data sets for the AMN learner with adaptive data reduction.*

9.3.3 Tests on a Real Data Set

We applied the described classification algorithm to real data collected with a SICK laser range finder mounted on top of a pan/tilt unit. The data consisted of 3D outdoor scans from a building with different kinds of windows. In a first step, we divided the data into walls by using a simple plane extraction algorithm based on region growing (see section 5.2.1). For each wall we obtained a normal vector $\mathbf{n}$ and a mean point $\mathbf{q}$. Then we extracted all points that had a distance of at most $0.5m$ from the planes. In this way we achieve robustness against noise from the normal vector calculation.

The goal was to classify the scan points into the classes 'window', 'wall' and 'gutter'. One of the data sets together with its manually created labeling is shown in figure 9.2a. It represents one wall of the building with only single-size windows. For the evaluation we used three different scans of walls of the building with single- and double-size windows.

9.3.4 Feature Extraction

We evaluated different types of features. It turned out that good results can be achieved with feature vectors that represent a local distribution of some value. One such distribution we used in the experiments was that of the cosine of the angles between the local normal vectors in the vicinity of each point and the plane normal vector $\mathbf{n}$. For a neighbor $\mathbf{p}'_i$ of a given scan point this value is calculated by $\alpha := \mathbf{p}'_i \cdot \mathbf{n}$. The distribution over α is represented as a local histogram.

Another set of features was obtained by considering the distribution of neighbors in front of and behind the wall plane. To be more precise, at each scan point $\mathbf{p}_i$ we counted the number of neighbors $\mathbf{p}'_i$ so that $|\mathbf{p}'_i \cdot \mathbf{n}| > |\mathbf{p} \cdot \mathbf{n}| + \epsilon$ where ϵ was used as a threshold to get robustness against noise. In our experiments, ϵ was set to $0.05m$. Accordingly, we counted the neighbors so that $|\mathbf{p}'_i \cdot \mathbf{n}| < |\mathbf{p} \cdot \mathbf{n}| - \epsilon$ and $|\mathbf{p} \cdot \mathbf{n}| - \epsilon \leq |\mathbf{p}'_i \cdot \mathbf{n}| \leq |\mathbf{p} \cdot \mathbf{n}| + \epsilon$. This way we obtained a histogram with three bins.

The last feature we used was the normalized height of each scan point. Here, we assumed a maximum scan height h_{max} of $15m$ which is reasonable considering that objects that are higher than $15m$ cannot be scanned accurately. For points with negative height, this feature was set to 0. For all others it was the quotient of the local height and h_{max}. This feature was especially used to distinguish 'gutter' from the other classes.

9.3.5 Building the Markov Network

An important implementation detail is the way the nodes are connected in the network. If we take too many neighbors, the learning and the inference will be less efficient and require a lot of time. Also, the way in which the connections are defined has an influence on the classification result. For our experiments, it turned out that a sampling strategy similar to the one reported by Anguelov et al. [2005] gives the best results. In our case, we randomly sample neighbors for each scan point $\mathbf{p}_i$ using a Gaussian distribution. Then we connect $\mathbf{p}_i$ to its neighbors so that no point is connected to more than three others. This guarantees that learning and inference can be carried out efficiently and at the same time provides enough information from the neighboring points.

9.3.6 Evaluation

The experimental results are shown in Figures 9.3 and 9.4. For comparison, we ran a Bayes classification on the same data set, yet with a different set of features. The reason for this is that in Bayes classification the features are assumed to be distributed normally and this did not hold in the case of our features. The best result we obtained is shown in figure 9.3a. It can be seen that in some regions the classes are locally inconsistent. Especially in the roof windows, the classification is wrong. This is because the Bayes classifier only decides locally on the labels and does not take the neighbors into account.

Figure 9.3b, shows the result for the same data set obtained with the AMN approach. Additionally, figure 9.4 shows the results for two other test instances. For solving the QP in the learning step we used the C++ library OOQP Gertz and Wright [2003].

For quantitative evaluation we labeled one of the test sets by hand and compared the results with this labeling. We obtained 85.5% correct classifications for the Bayes classifier and 93.8% for the AMN with adaptive data reduction. The computation time for the learning step was between 5 and 7 seconds on a Pentium 4 with 2.8 GHz.

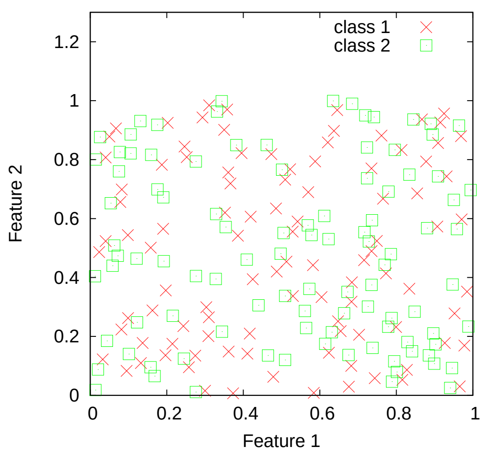
9.4 Instance-based Extension

The main drawback of the AMN classifier, which is based on the log-linear model, is that it separates the classes linearly. This assumes that the features are separable by hyper-planes, which is not justified in all applications. This does not hold for *instance-based* classifiers such as the nearest-neighbor classifier (NN). In NN classification, a query data point $\tilde{\mathbf{p}}$ is assigned to the label that corresponds to the training data point $\mathbf{p}$ whose features $\mathbf{x}$ are closest to the features $\tilde{\mathbf{x}}$ of $\tilde{\mathbf{p}}$. In the learning step, NN simply stores the entire training data set and does not compute a reduced set of training parameters, which is why it is also called *lazy classification*.

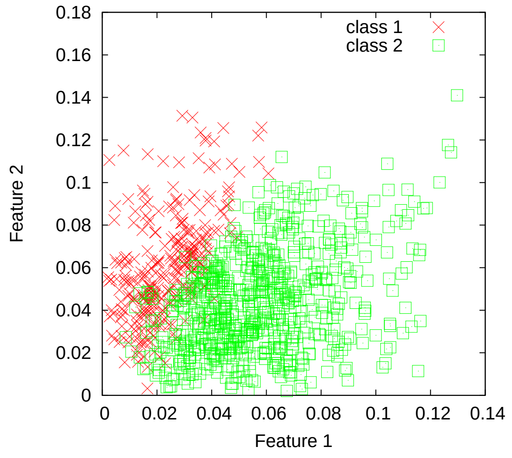
The idea now is to combine the advantage of instance-based NN classification with the AMN approach to obtain a collective classifier that is not restricted to the linear separability requirement. This will be presented in the next section.

9.4.1 The Transformed Feature Vector

Consider a simple 2-class classification problem with 2 features for each data point. Suppose that the training data consists of the feature space shown in Fig. 9.5 on the left. We can see that the two classes can not be separated by a line. However, if we assume that the correct label for a new query data point $\tilde{\mathbf{p}}$ corresponds to the label of its closest training data point in feature space, then the NN algorithm yields the optimal classification. We can think of this as first *transforming* each feature vector $\tilde{\mathbf{x}}$ to the vector $(d(\tilde{\mathbf{x}}, \mathbf{x}^1), d(\tilde{\mathbf{x}}, \mathbf{x}_2))$ where $\mathbf{x}^1$ is the feature point of class 1 that is closest to $\tilde{\mathbf{x}}$, $\mathbf{x}^2$ the closest class-2-point and $d(.,.)$ is the distance in feature space. Applying this transform $\tau : \mathbb{R}^2 \to \mathbb{R}^2$, $\tau(\tilde{\mathbf{x}}) =: \tilde{\mathbf{t}}$ to each feature point $\tilde{\mathbf{x}}$ yields a transformed feature space which can be separated by the



a. *Synthetic feature space*



b. *Transformed feature space*

Figure 9.5: *Application of the feature transformation on a synthetic data set. (a): Original feature space, (b): Feature space after applying transform τ. For details see text.*

line $\tilde{t}_1 = \tilde{t}_2$ where $\tilde{\mathbf{t}} = (\tilde{t}_1, \tilde{t}_2)$. In other words, all feature points for which $\tilde{t}_1 < \tilde{t}_2$ will be classified as 1 and all points where $\tilde{t}_2 < \tilde{t}_1$ will be assigned the label 2.

In practice, it is not always possible to find the correct label using the NN rule. However, if we instead assume that the probability that $\tilde{\mathbf{x}}$ has label $\tilde{y}$ is *proportional* to the distance of $\tilde{\mathbf{x}}$ to its nearest neighbors in each class, we still obtain linear separability, even though there will be some classification errors.

To illustrate this, we randomly drew 1000 samples from the feature space shown in Fig. 9.5a. The label y that was assigned to each sample $\tilde{\mathbf{x}}$ was randomly drawn where the probability of $y = k$ was governed by a Gaussian distribution $\mathcal{N}(\mu^k, \sigma^k)$ with $\mu_k = d(\tilde{\mathbf{x}}, \mathbf{x}^k)$ and $k \in \{1, 2\}$. This means, the closer $\tilde{\mathbf{x}}$ is to a feature point $\mathbf{x}^k$ with label k, the more likely it is that label $y = k$ is assigned to it. After applying the described feature transform τ to these samples, we obtain the feature points shown in Fig. 9.5b. It can be seen that the feature space is separable by a line passing through the origin, although there will always be some classification error. However the classification result will be much better than obtained by linearly separating the original feature space in Fig.9.5a.

9.4.2 *M* Nearest Neighbors

One problem of the NN classifier is that the assignment of a label to a query point $\tilde{\mathbf{p}}$ only depends on the labeling of *one* instance in the training set, namely the one whose feature vector is closest to $\tilde{\mathbf{x}}$. However, it is possible that the features of other training instances are also very close to $\tilde{\mathbf{x}}$, although they are labeled differently. For example, suppose that the distances of $\tilde{\mathbf{x}}$ to the two closest training features $\hat{\mathbf{x}}^1$ and $\hat{\mathbf{x}}^2$ are very similar, and the corresponding labels $\hat{y}^1$ and $\hat{y}^2$ are different, the decision of assigning the label $\hat{y}^1$ to $\tilde{\mathbf{p}}$ may be wrong, especially in the presence of noise. Therefore we proceed as follows: For the feature vector $\tilde{\mathbf{x}}$ that corresponds to $\tilde{\mathbf{p}}$ we compute the M nearest training instances in each of the K classes $C_1, \dots, C_K$ and the corresponding distances $d(\tilde{\mathbf{x}}, \hat{\mathbf{x}}_k^m)$ where $k = 1, \dots, K$ and $m = 1, \dots, M$. These are used to define the transformed feature vector $\tau(\tilde{\mathbf{x}})$ as

$$\tau(\tilde{\mathbf{x}}) = (\dots, d(\tilde{\mathbf{x}}, \hat{\mathbf{x}}_k^1), \dots, d(\tilde{\mathbf{x}}, \hat{\mathbf{x}}_k^M), \dots) \tag{9.9}$$

Experiments show that higher values of M increase the classification rate, but for large M the improvement is very small. In our experiments, $M = 5$ turned out to be a good choice.

9.4.3 Implementation Details

To speed up the learning process, we need to represent all feature vectors such that the a nearest neighbor search in the feature space can be performed efficiently. To this end, we use kD-trees $\mathcal{K}_1, \dots, \mathcal{K}_K$ to store the training feature vectors of each class $C_1, \dots, C_K$. This way, the computational effort of the nearest neighbor lookup becomes logarithmic in the number of the stored instances.

Thus, the training step consists of computing the features for the training data, transforming these features according to equation (9.9) and assigning the transformed features to the nodes of the AMN. The edge features of the AMN consist of a constant scalar value, as described by Anguelov et al. [2005]. After solving the quadratic program we obtain the weight vectors $\mathbf{w}^k$. Then, in the inference step, we use the transformed features $\tau(\mathbf{x})$ of the test data as node features for the AMN. Again, the edge features are constant.

Feature Computation Depending on the input data used, we computed different types of features for the data points. In the case of the 2D grid data, each point in the map is represented by a set of geometrical features which are extracted from a laser range scan covering 360^o field of view. Each laser is simulated and centered at each point in the map representing a free space Martínez Mozos et al. [2005]. Because the number of geometrical features is big (more than 300) we select the best ones using the AdaBoost algorithm.

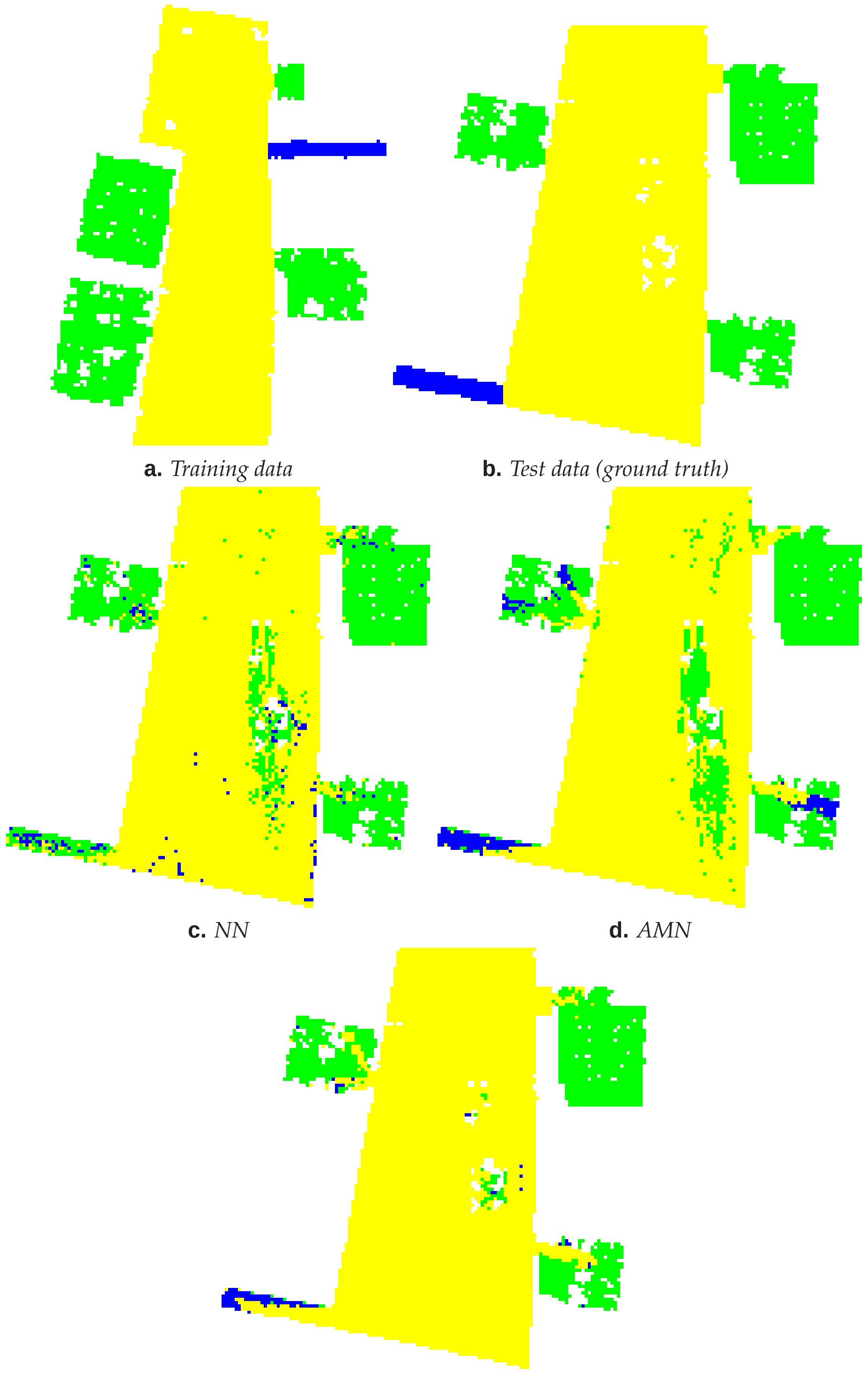
For the 3D data set we computed *spin images* Johnson [1997] with a size of 5×10 bins. The spherical neighborhood for computing the spin images had a radius between 10 and 15cm, depending on the resolution of the input data.

9.4.4 Experimental Results

We performed a series of experiments with 2D and 3D data to compare our *instance–based AMN* (iAMN) algorithm, with the NN and the AMN classifier. The results of these experiments demonstrate that our iAMN outperforms the two other algorithms, independent of the features used.

2D Map Annotation

For the 2D classification experiment we used an occupancy grid map of the interior of a building. The map was annotated with three different labels, namely 'corridor', 'room' and 'lobby'. Then the map was divided into two non-overlapping sub-maps, one for training and one for testing. Fig. 9.6a shows the sub-map used for training and Fig. 9.6b



a. *Training data* **b.** *Test data (ground truth)*

c. *NN* **d.** *AMN*

e. *iAMN*

Figure 9.6: *Results on 2D data of an occupancy grid map.*

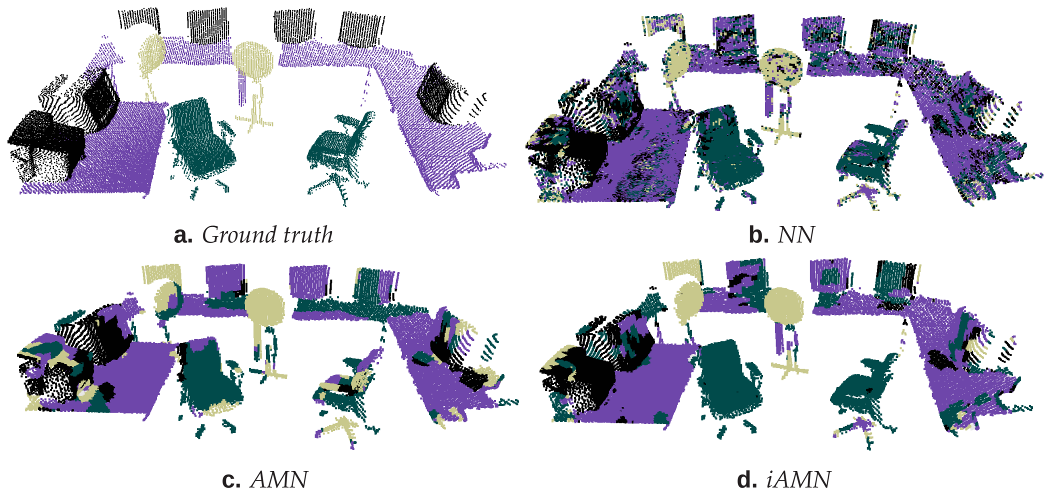


Figure 9.7: *Classification results for one scene from the* M *data set.*

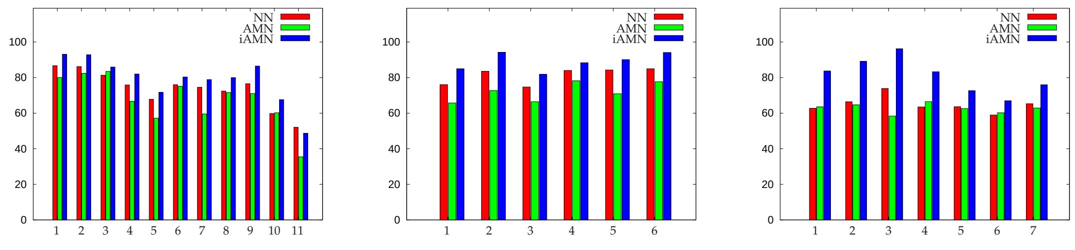
the ground truth for the classification. The results obtained with the NN and the standard AMN approach are shown in Figs. 9.6c and 9.6d, while Fig 9.6e shows the result of our iAMN algorithm. It can be seen that the iAMN approach gives the best result. The classification rate of the NN is with 92.2% higher than that of the standard AMN, which was 89.8%, but the classification is very noisy. In contrast, the iAMN result is more consistent with respect to neighboring data points and has the highest classification rate with 95.5%.

3D Scan Point Classification

Furthermore, we evaluated the classification algorithms on three different 3D data sets with an overall number of 38 scanned scenes. The scans were obtained with a 3D laser range scanner. The first data set is called H and consists of 11 recorded scenes with two humans in varying poses. The scans were labeled into the four classes ’head’, ’torso’, ’legs’, and ’arms’. The second data set named S consists of 20 different 3D scans of the object classes ’chair’, ’table’, ’screen’, ’fan’, and ’trash can’. Each scan in the S data set contains just one object of each class, apart from tables, which may have screens standing on top of them. The last data set is named M and consists of seven scans with multiple objects of the same types as in the S data set.

The H data set was evaluated using 11-fold cross validation. From the S data set we created six training examples for each object and evaluated the classification on the rest of the scans. For the M data set we used the object instances from the S set for training, because those were not occluded by other objects.

Fig. 9.7 shows a typical classification result for a scan from the M test set. We can see that NN assigns wrong labels to different points while their neighbors are classified correctly. The AMN results show that areas of points tend to be classified with the same label. However, due to the restriction to linear separable data, many object parts are



a. *Data set* H **b.** *Data set* S **c.** *Data set* M

Figure 9.8: *Individual classification rates for all scenes.*

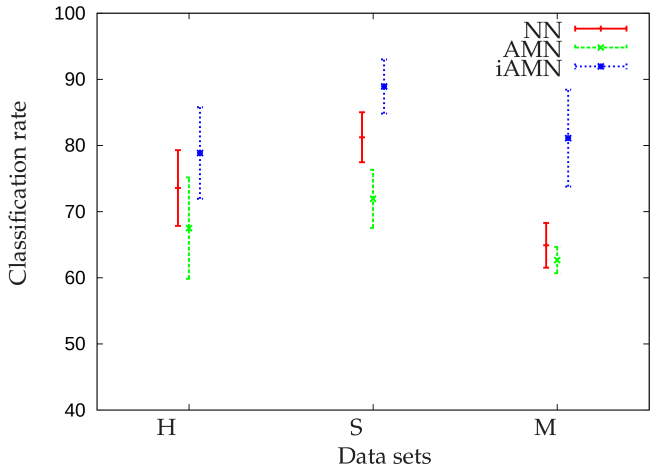


Figure 9.9: *Statistical analysis of the three different data sets. For the* M *data set we obtain a significant improvement over NN and standard AMN classification*

misclassified, especially in complex objects like chairs and fans. In contrast, the results obtained with iAMN are better even in these complex objects, because the transformed feature vectors computed by iAMN are better suited for a classification based on separating hyper-planes. Table 9.1 shows the resulting classification rates. We can see that the iAMN classifier outperforms both of the others in all three data sets.

A statistical analysis is shown in Figure 9.9. As can be seen, for the M set our algorithm performs significantly better than both, the iAMN and standard AMN. A detailed analysis of the classification results is depicted in Figure 9.8. These results demonstrate, that our iAMN yields highly accurate results for all three data sets.

9.5 Conclusions and Future Work

In this chapter, we presented a framework for classifying objects in 2D and 3D range data sets, which follows the concept of collective classification. In collective classification, the statistical independence assumption of neighboring data points is relaxed which leads to a

Data set	NN	AMN	iAMN
H	75%	69%	80%
S	81%	72%	89%
M	63%	62%	76%

Table 9.1: *Classification results for all three data sets*

more realistic modeling. As the core mechanism for the classification we used associative Markov networks, which are a special instance of conditional Markov random fields. The major contributions in this chapter are an intelligent way to reduce the training data set for faster learning, and an extension of the AMN classifier that reduces the problems arising from the linear separability requirement of AMNs. In extensive experiments we showed the improvements of the new approaches over the standard AMN classification. However, the work does not end here. Many issues are still to be solved, one of them being the occurrence of more and more ambiguities as the number of classes increases. A possible way to tackle this problem may be a hierarchical approach, in which a set of geometric primitives are detected by one AMN, while another one on a higher level detects more complex objects from these primitives. Current work is already directed into this line of investigation.

Part IV

Applications

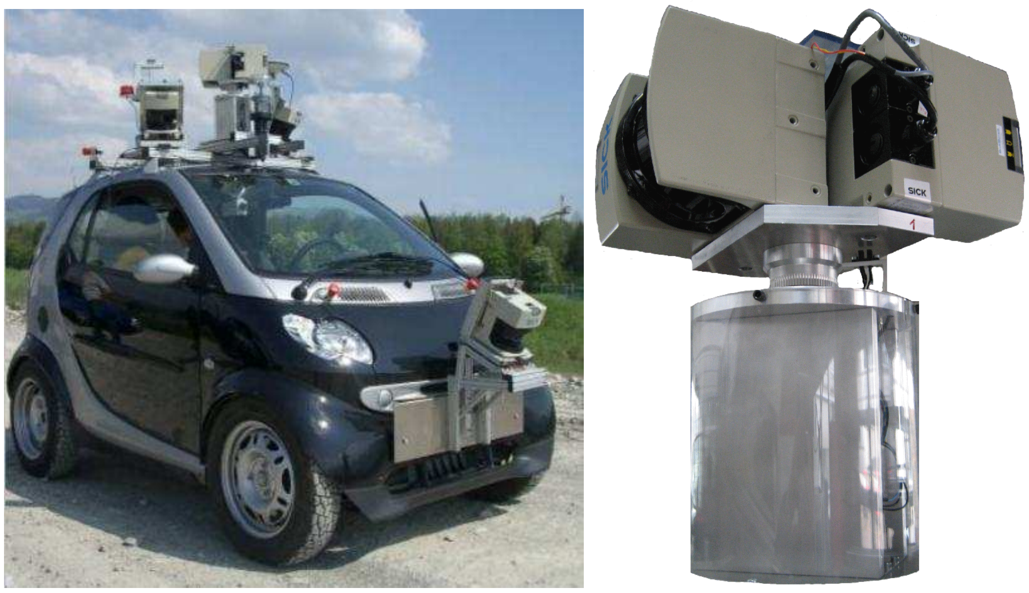
10

3D Mapping with a Robotic Car

10.1 Introduction

In this chapter, we want to give an example of how to apply the concepts of 3D map building presented in chapters 6 and 7 to a real world problem. The work we will present here has been part of a project that was mainly developed at the universities of Lausanne, Switzerland and Freiburg, Germany. The goal of that project was to develop an autonomous car that is able to create high-resolution and accurate 3D maps of the environment while moving in this environment. A detailed description of the entire project can be found in [Lamon et al., 2006a; Lamon et al., 2006b; Pfaff et al., 2007b]. In the following, we will focus on the 3D mapping aspect of the system, rather than on the autonomous driving, because it is more relevant to the aim of this thesis. In particular, we will show how the concepts of building multi-level surface maps and loop closing can be used in this particular application. The combination of a highly accurate pose estimation algorithm developed in the project, together with the map registration and loop closing techniques described earlier will be presented as a solution to the online 3D outdoor map building problem.

The chapter is organized as follows: First we describe in section 10.2 the details of the system used for the 3D mapping task. We will focus on the localization and the mapping part, as these are particularly relevant for the presented application. Then, in section 10.3 we present the results of the extensive experiments that were carried out in the project. Finally, section 10.4 concludes the chapter and presents future work.



a. *The* S T *robotic system for outdoor 3D terrain mapping.* **b.** *Close-up view of the rotating 3D laser scanner.*

Figure 10.1: *The robotic car* S T *and its 3D scanning device.*

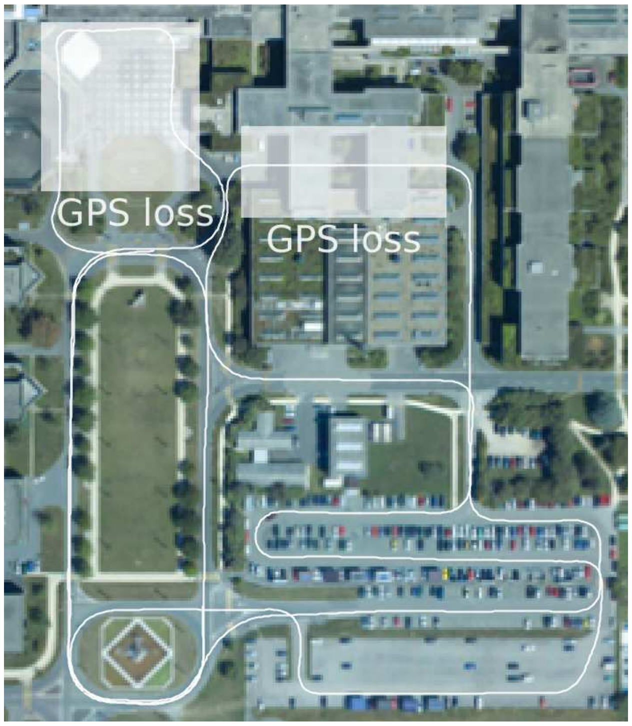
10.2 Description of the System

Figure 10.1a shows a picture of the robotic car S T used in the 3D mapping experiments. The car is a Smart ForTwo that has been modified for driving in off-road terrain. It is equipped with a variety of different sensor systems, including an inertial measurement unit (IMU), a differential GPS, omni-visual and perspective CCD cameras and five outdoor laser range finders by SICK. Two of these laser range finders are mounted vertically onto a rotatable socket. The power supply and the data line of the scanners are provided by means of sliding contacts inside the rotating device so that a constant one-directional rotation can be performed. After each full rotation, a signal is triggered that indicates the completion of the 360° rotation. Figure 10.1b shows a close-up view of this 3D scanning device.

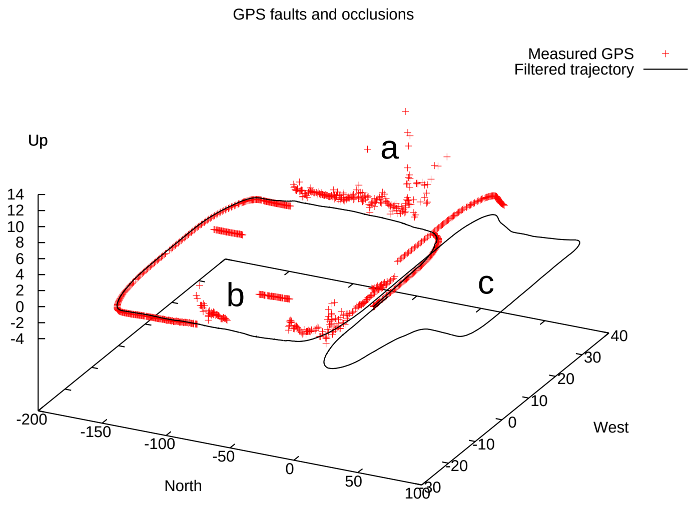
The software architecture used for the S T robot consists of many modules, where two of them are particularly important. These are the localization module and the 3D mapping module, which we will describe in the following.

10.2.1 The Localization Module

As mentioned, several sensors are utilized for an accurate localization of the vehicle. In particular, these consist of an IMU, a differential GPS, an optical gyroscope and the odometry measurements from the car. The data acquired with these sensors is fed into an information filter, which is the inverse form of a Kalman filter. The outcome of this



a. *Overlay of the estimated trajectory and the ortho-photo of EPFL. The zones where the GPS was not available are highlighted. The total traveled distance is* 2350*m.*



b. *During the run, the GPS signal was disturbed by many objects (trees, buildings, etc.) and GPS faults are of high amplitude. The localization algorithm was able to reject erroneous GPS fixes and to provide accurate estimations. The labels a,b and c mark areas where GPS is of poor quality (a, b) or unavailable (c). The graph represents only a part of the trajectory depicted in Fig.(a)*

Figure 10.2: *Example run of* S T *over the campus of EPFL in Lausanne.*

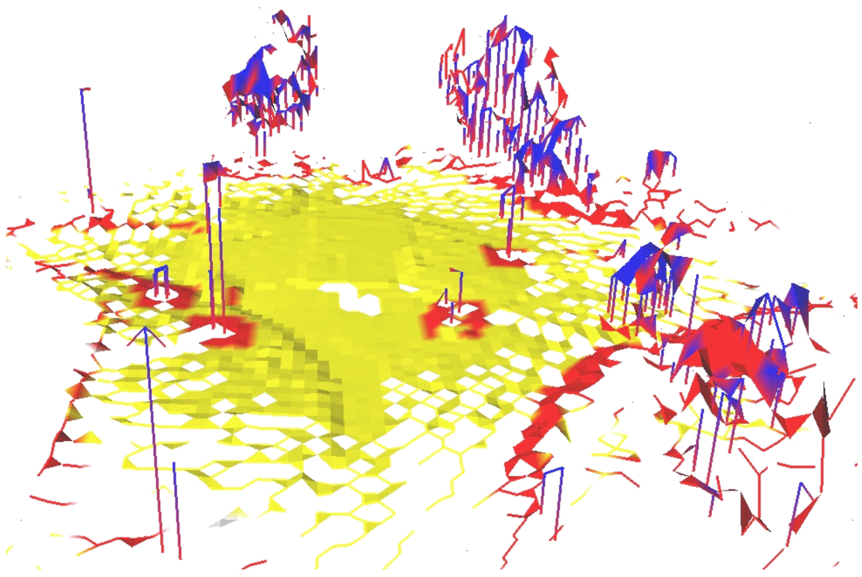


Figure 10.3: *Example of a local MLS map. The map was computed from a point cloud while the vehicle traveled through the campus environment shown in Fig.10.2a. The colors reflect the three different classes of surface patches: traversable (yellow), non-traversable and horizontal(red) and non-traversable and vertical (blue).*

information filter is a very precise estimate of the 3D position and orientation of the vehicle, even in cases where no GPS signal is available or when the GPS is disturbed by buildings or other objects. The details of the algorithm are described in Lamon et al. [2006b]. In Figure 10.2 we see an example of a run where the vehicle was driven over the campus of EPFL in Lausanne. In some areas, no GPS signal was available because the car traversed a parking garage or an underpass. This is shown as white boxes in the aerial image in Fig. 10.2a. However, the estimated location of the vehicle is still very accurate. Figure 10.2b shows the discrepancy between the raw GPS data and the estimated vehicle path. As can be seen, the localization algorithm yields a position estimate even when the GPS signal is disturbed or lost. The comparison of the estimated path with the aerial image in Fig. 10.2a shows that the path fits very well into areas of paved ground. This indicates the high accuracy of the localization module.

10.2.2 The 3D Mapping Module

The second major part of the system architecture is the 3D mapping module. It consists of three sub-modules that fulfill different tasks for the 3D mapping process, mainly in real time. The sub-modules are arranged in a data pipeline where the output of one is the input of the next one. The sub-modules are:

1. **The Point Cloud Generator:** The inputs into this sub-module are the range measurements from the rotating laser scanners (see Fig. 10.1b), the signals triggered by

the rotation device, and the 6D position and orientation estimate from the localization module. The point cloud generator computes a point cloud from all 3D range measurements that are acquired during one rotation of the scanners. This is done every 2 seconds.

2. **The Local MLS Map Generator:** This sub-module receives point clouds and computes local MLS maps as described in section 6.5.2. It also classifies the surface patches of the MLS map into the classes "traversable", "non-traversable and horizontal" and "non-traversable and vertical" (see sections 6.5.5 and 7.2.3). An example of such a local MLS map is shown in Fig. 10.3.

3. **The Global MLS Map Generator:** The task of this sub-module is to compute globally consistent MLS maps from the particular local maps. This is necessary, because – despite the high accuracy of the localization module – the local maps are still slightly misaligned. Therefore, we register consecutive local maps using the method described in section 7.2.3. In addition, the loop closing algorithm described in section 7.3.3 is applied whenever the vehicle is likely to close a loop. This is implemented as an input signal to the global MLS mapper which is triggered either manually or automatically from the localization estimate.

As mentioned, the whole 3D mapping module runs in real-time, with the exception of the loop-closing procedure, which fails to fulfill the real-time constraint. For scenarios where many big loops have to be closed, we therefore decided to compute the global MLS map offline.

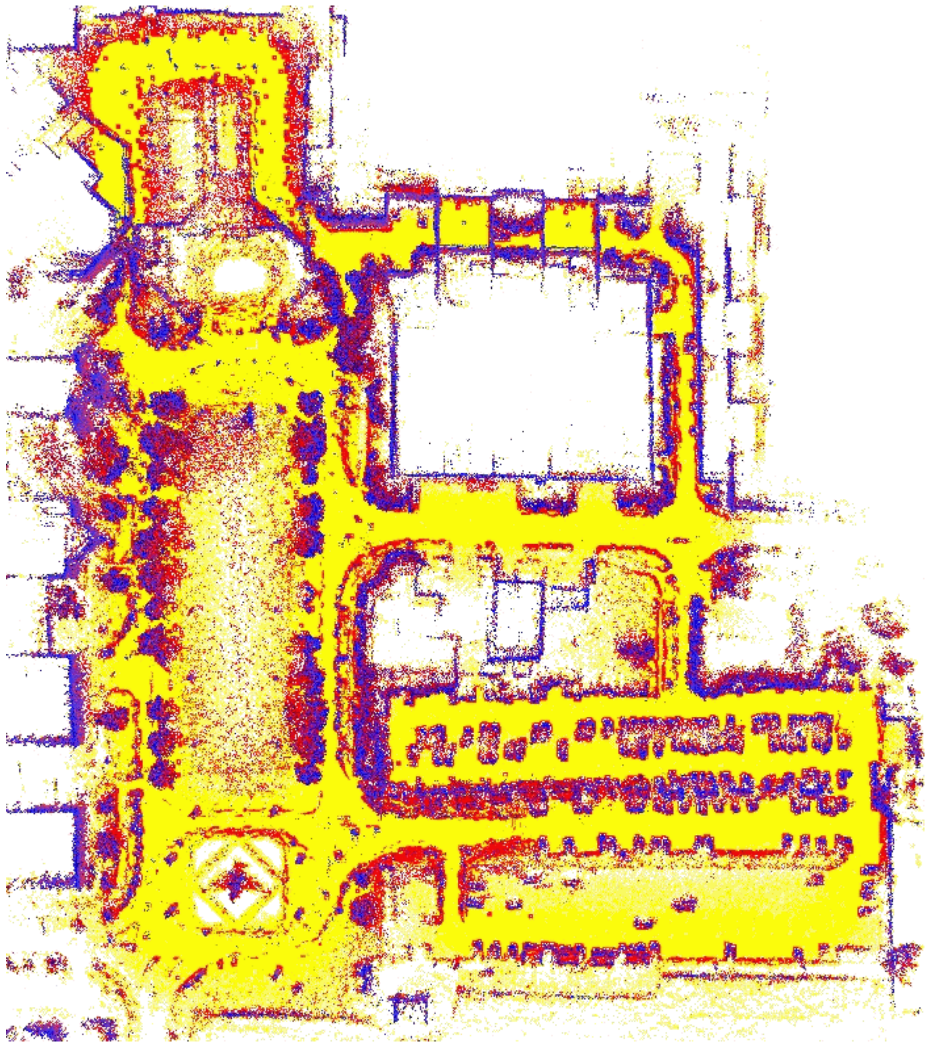
10.3 Experimental Results

We tested the described 3D mapping system on the 3D data set that was acquired while the vehicle traveled along the path shown in Fig. 10.2a. On this path, the robot encountered five nested loops. Additionally, it traversed through an underground parking lot and through several underpasses. The two main goals of our experiments where

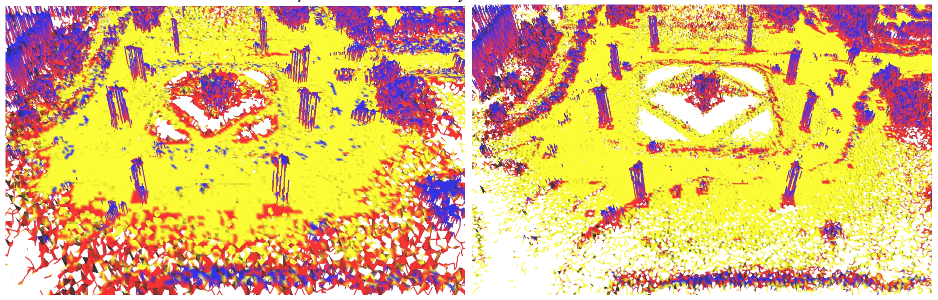
- to demonstrate that the 3D map representation yields a significant reduction of the memory requirements compared to a point cloud representation, while still providing sufficient accuracy;

- to show the necessity of the loop closing procedure despite the high accuracy of the localization module and the pose correction based on local map registration.

In the experiment we acquired 374 local point clouds consisting of $27,264,000$ data points. The area scanned by the robot spans approximately 300 by 250 meters and the overall path length was approximately $2,300m$. Figure 10.4a shows a top view of the resulting MLS map with a cell size of $50cm \times 50cm$. As before, the yellow surface patches are classified as traversable. It requires $55.16MB$ to store the computed map, where 34% of $300,000$ cells are occupied. Compared to this the storage of the $27,264,000$ data points requires $654,33$ megabytes.



a. *Top view of the resulting MLS map with a cell size of* $50 \times 50cm$. *The yellow surface patches are classified as traversable.*



b. *Mapping result without loop closing.* **c.** *Mapping result with loop closing.*

Figure 10.4: *Full view of the campus map (a) and two close-up views of the lower left corner. Fig. (b) depicts the resulting MLS map when only local map matching is applied. Fig. (c) displays the same part of the MLS map where the loop closing algorithm was applied.*

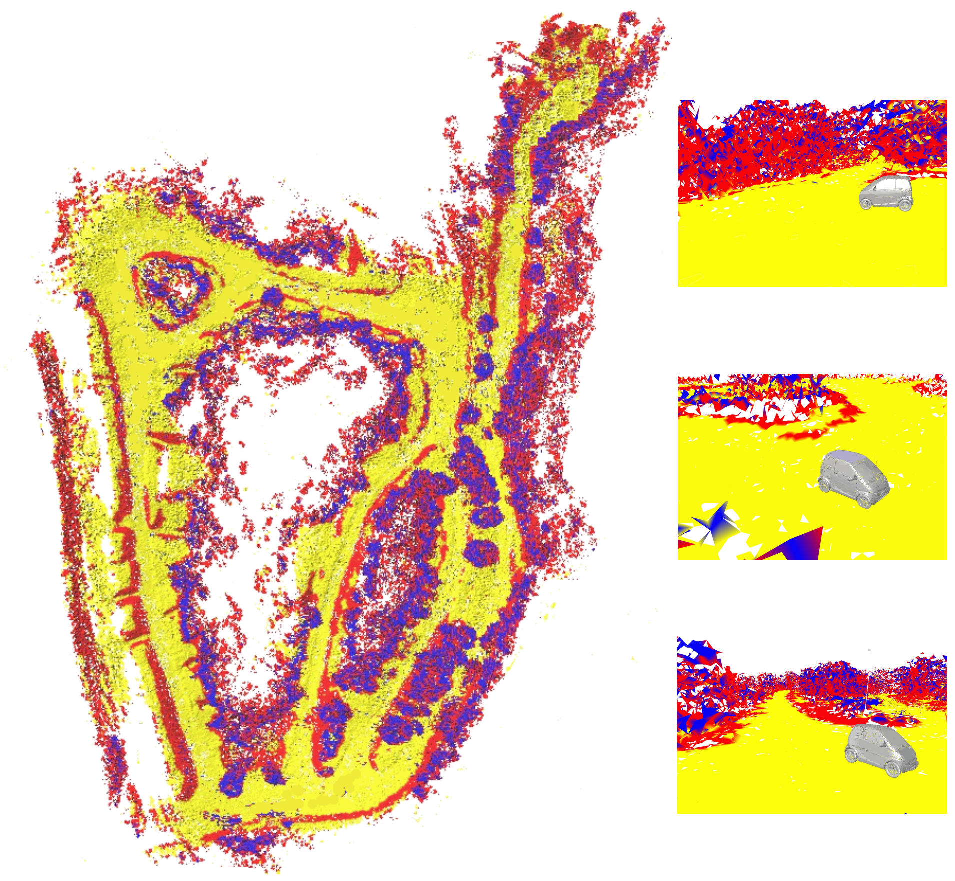


Figure 10.5: *The left image shows a top view of the resulting MLS map of a military test site with a cell size of* $50cm \times 50cm$*. The area scanned by the robot spans approximately 250 by 200 meters. During the data acquisition, the robot traversed three nested loops with a length of approximately 1,200m. On the right hand side three cutouts with a visualized smart are depicted.*

Figures 10.4b and 10.4c show two different MLS maps that were computed from the same data set. The left image depicts the resulting MLS Map when only local map matching is applied. The right image displays the same part of the MLS map where we additionally applied our loop closing algorithm. We can see several inconsistencies in the left map, especially when looking at the vertical poles, which in reality correspond to street lamps. Furthermore, several traversable patches have been classified as non-traversable due to the misalignment of the maps.

In a second experiment, the robotic car was steered over a military test site. Here, the robot's path consisted of three nested loops. We acquired 312 local point clouds consisting of $22,700,000$ data points. The site area was approximately 250 meters long and 200 meters wide. The distance traveled by the vehicle was about $1,200m$. Fig. 10.5 shows a top view of the resulting MLS map. Again, the cell size is $50 \times 50cm$. On the right side of Fig. 10.5 we can see three cutouts with a visualized smart. In this case, the computed map requires 17.15MB of memory where 36% of $200,300$ cells are occupied. In contrast, the memory required to store a point cloud of $22,700,000$ data points would be $544,8MB$.

10.4 Conclusions and Future Work

In this chapter, we showed how the novel concept of multi-level surface maps and the computation of globally consistent maps can be applied to a real-world problem, namely the creation of accurate and compact 3D maps of large-scale outdoor environments. Furthermore, we showed that the combination of an information-filter based localization algorithm and an online version of the 3D map building algorithm described in chapters 6 and 7 is capable of fulfilling this task. In particular, we saw that the loop-closing technique described in section 7.3.3 is necessary and appropriate to obtain consistent 3D maps.

The next step is to show experimentally that these maps are useful and needed for localization and planning, which was the design goal mentioned in chapter 6. This will strengthen the theoretical considerations made there. Some first experiments have already been made and the results were encouraging

Every man takes the limits of his own field of vision for the limits of the world.
Arthur Schopenhauer (1788 - 1860)

11 Occlusion Handling

11.1 Introduction

As a second application example, we want to consider the problem of *occlusions*. Occlusions may occur in any kind of perception task, i.e. in a 2D camera image, in a 3D range scan or in data acquired with other sensors such as sonar measurements. An occlusion occurs whenever an object is partially hidden by others and therefore can not be perceived entirely. For example, when looking into a room from outside, we might see chairs and tables, but they are only partially visible. Nonetheless, humans are still able to recognize these objects correctly as chairs or tables, because they have some kind of knowledge about the shape of the entire object. In this chapter, we will present a method to automatically solve the occlusion problem in 3D laser range scans using previously acquired knowledge about the shape of the objects. This will be done by first applying the object recognition techniques developed in chapter 9. Once the inference task is performed on a new test data set, the idea is to compare objects to each other to which the same class label has been assigned. This enables us to infer the shape of a single object based on the shapes of the other objects in the class. This assumption behind this is that the objects in a class are somehow similar to each other. To compare the object entities of a given class, we first cluster the corresponding set of data points into singular contiguous point clouds. Then we match these point clouds to each other using the ICP algorithm described in section 7.2. This results in a *prototype* of the particular object. In the next step, we create a mesh representation out of this prototypical point cloud. The obtained mesh is then remapped to the position where the original object entities were. As a result, the scene representation is more realistic due to the mesh computation and partial object occlusions are resolved.

The chapter is organized as follows. In Section 11.2 we describe the details of the occlusion handling algorithm. Section 11.3 presents some experimental results.

11.2 Description of the Algorithm

Assume we are given a point cloud $\mathcal{P}$ of a 3D scene, that contains several entities of different kinds of objects. In our example, we will use the wall of a building with windows as shown in Fig. 9.3 and 9.4. Assume further that we have applied the AMN classification method described in section 9.2 (or its improved version from section 9.4). As a result, we have a class label $l_i \in \{1, \dots, c\}$ for each data point $\mathbf{p}_i \in \mathcal{P}$ where c is the number of classes. These class labels are used to split $\mathcal{P}$ into c subsets $\mathcal{P}_1, \dots, \mathcal{P}_c$ so that all points in a subset $\mathcal{P}_i$ have the same label l_i. Then, for each of these class specific subsets, we perform the steps "clustering", "entity matching" and "mesh generation and remapping". These steps will be described in the following.

11.2.1 Clustering

This step actually consists of two hierarchically applied clustering steps. The first clustering is done in Euclidean 3D space on all data points of each cloud $\mathcal{P}_i$. The aim of this step is to find contiguous subsets in $\mathcal{P}_i$ that correspond to object entities. The clustering is done using a region-growing algorithm similar to the one described in Algorithm 2 on page 59 with the difference that no test is done on the angle of the normal vectors – in fact no normal vectors are computed here. As a result, each cluster contains points that are not farther away from each other than a given threshold. This threshold is defined in the function N in line 11 of Algorithm 2. As before, the neighborhood of two points is computed efficiently using a kD-tree to store the point cloud $\mathcal{P}_i$. We will denote the object entities found for the class $\mathcal{P}_i$ as $P^i_1, P^i_2, \dots$.

In the second clustering step, the entities are clustered into subclasses, which we will call *kinds*. This is motivated by the fact that usually a class consists of different kinds of objects, e.g. in a class "window" we can find single- and double-size windows as well as windows of different shapes. Of course, the distinction of these subclasses could be done when labeling the training data already. In this case the AMN would automatically yield appropriate labels for the individual subclasses. However, we decided to separate the division into kinds from the labeling process, because it turned out to be difficult to define features for the different entities on the point level. For this second clustering step, we use three entity features based on the oriented bounding box (OBB) B^i_j of the entity's point cloud P^i_j (see also Section 4.3.2 on OBBs). The features are the volume of B^i_j, the quotient of the second-longest and the longest edge of B^i_j and the *radius* of P^i_j, defined by the maximal distance of a point in P^i_j and the centroid of P^i_j. Again, for the clustering we apply region-growing, in this case in 3D feature space. The result of this second clustering step is a set of kinds $\mathcal{K}^i_1, \mathcal{K}^i_2, \dots$ for the class i and a mapping that assigns to each entity P^i_j a kind $\mathcal{K}^i_k$.

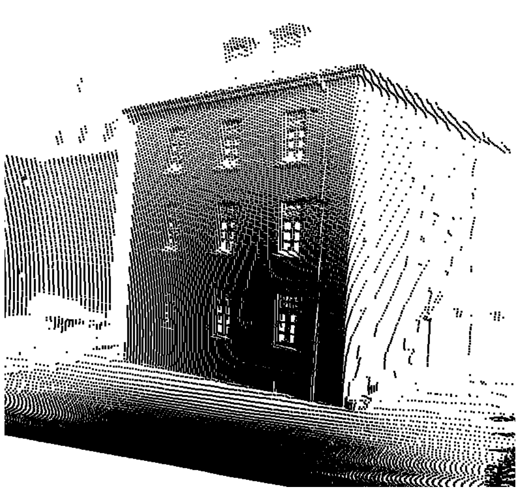
11.2.2 Entity Matching

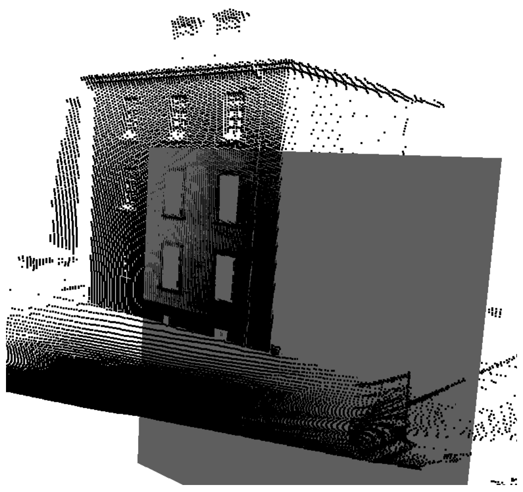
In the next step, the entities that belong to the same kind are matched to each other. This is the step in which information about the shape of one entity is used to complete the shape of another entity of the same kind. This assumes that all entities of a kind have the same shape and that a good matching between entities can be found. The matching is done using the Iterative Closest Point algorithm (ICP) described in Section 7.2.1, where one entity is selected as a reference frame and all other entities of the same kind are matched to this reference entity. This is a simple method and has the advantage that the number of computed matches is only linear in the number of entities of a given kind. One could also think of connecting all entities into a clique and match all entities to each other. Then, the matching errors could be reduced by performing a global optimization technique such as the one described in Section 7.3.3 on all entity poses. This would result in a number of matchings that is quadratic in the number of entities, and thus the process to find a prototype for each kind would be much slower. In our experiments, we obtained good results with the one-reference-frame technique. The result of the entity matching step is a point cloud $\hat{P}^i_k$ for each kind $\mathcal{K}^i_k$ that consists of all matched entities of that kind. This point cloud is defined as the *prototype* of the kind $\mathcal{K}^i_k$.

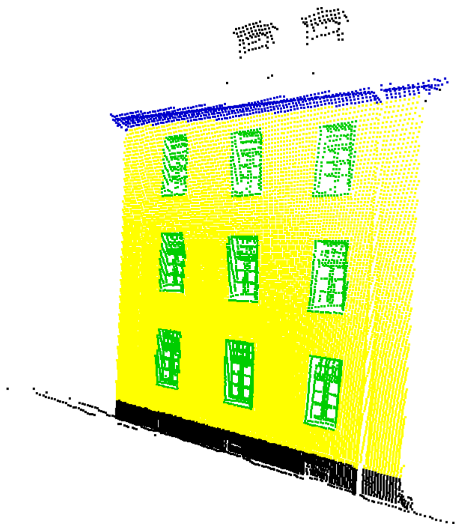
11.2.3 Mesh Generation and Remapping

Finally, to get a better visualization, we generate triangulated meshes from the prototypes $\hat{P}^i_k$ resulting from the previous step. This can be done with one of the triangulation methods that are described in section 4.5. For our application, we used a contouring algorithm which is mentioned in Section 4.5.2. The reason for this choice is that the prototype point clouds often contain many isolated data points due to small errors in the matching process or to imperfections of the scanning device. These *outliers* become more apparent, the more entities are used for creating the prototype. Furthermore, the residual error resulting from the entity matching process leads to uncertainties in the relative positions of the entities to each other. This means that a triangulation that connects the points of the prototype directly, such as an alpha-shape (see Section 4.5.3) would give a poor result, because the position of these points is not certain. Instead, a triangulation method that reflects this uncertainty, such as the grid-based contouring algorithm, gives a better result. For a discussion on different triangulation methods see also Section 4.5.4.

After applying the marching-cubes contouring algorithm by Lorensen and Cline [1987] (again, see Section 4.5.2 for details) on each of the prototypes $\hat{P}^i_k$ we obtain a triangular mesh representation $\hat{M}^i_k$, that approximates the volume represented by the prototype $\hat{P}^i_k$. These meshes are then mapped to all original positions of the singular entities. The rotation and translation parameters of this mapping are obtained by inverting the affine transforms that resulted from the entity matching process.


a. *Original scan*


b. *Plane extraction*


c. *Training data labeled by hand*

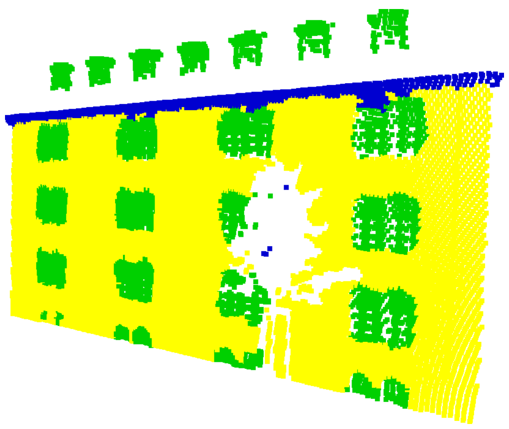
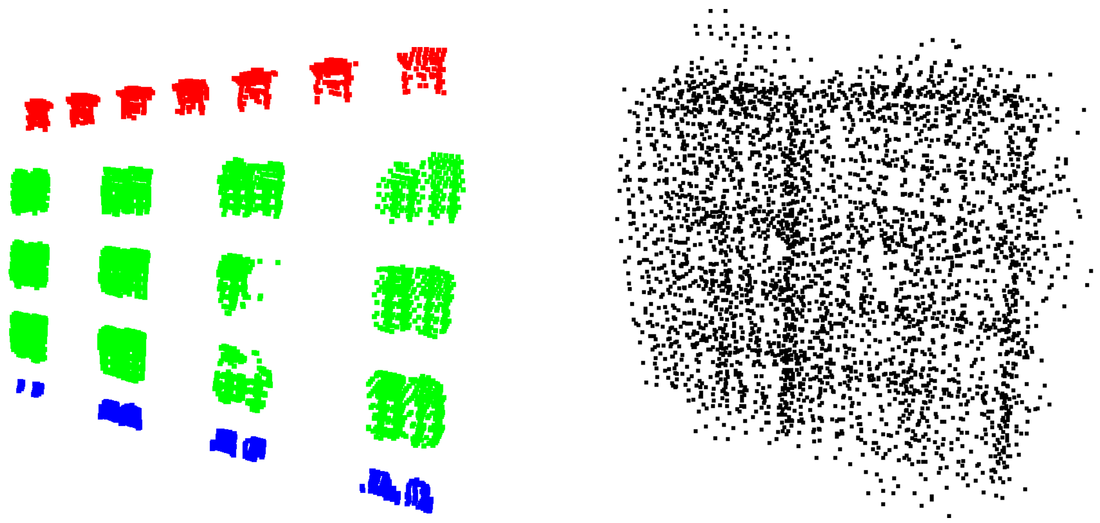

d. *Output of AMN classifier(test set)*

Figure 11.1: *Example of the occlusion handling algorithm. First, the AMN classification described in chapter 9 is applied: From the original 3D point cloud (a) we extract vertical planes to get the walls of the building (b). Then, we label one of these walls – which becomes the training data set – by hand (c). After applying the AMN classifier on a different wall (the test data set), we obtain the classification in Figure (d). Note that some windows and the wall were occluded by a tree during the data acquisition.*



a. *Clustering* **b.** *Entity matching*

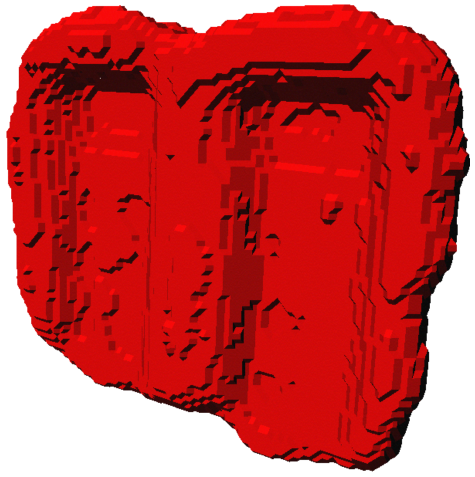
Figure 11.2: *The next steps of the occlusion handler. After the clustering step, we obtain three different kinds of windows (see Figure (a)). For each of these kinds, we perform the entity matching step and obtain a prototype. Figure (b) shows the prototype for the big windows.*

11.3 Experimental Results

We implemented the described occlusion handling algorithm and tested it on the data sets shown in Figure 9.2 and 9.3. As mentioned above, the first step is the application of the AMN classifier to the test data set. The classes that were learned by the algorithm are "window", "wall" and "gutter". The steps that were performed to obtain the training and test data set, as well as the classification result are depicted in Figure 11.1, where the latter is shown here for completeness. Again, we note that the shape of the windows in the training data set differs slightly from that of the test data set, but the classifier was still able to detect the windows. Furthermore note that there are gaps in the test data set, which are caused by the occlusions of a tree in front of the building. This results in windows that are only partially seen.

Figure 11.2 shows the results of the clustering and the entity matching step. As we can see from figure 11.2a, after clustering we obtain three different *kinds* of windows, namely roof windows, double-size windows and basement windows of a smaller size. The two big windows which are occluded are still correctly clustered, because their shape is most similar to that of the other big windows. After matching all entities of big windows we obtain the prototype point cloud shown in Figure 11.2b. We can see that the alignment is not perfect and some outliers occur on the window borders. However, the window shape is represented in more detail than by any of the point clouds from the particular entities.

The next step of the occlusion handling algorithm computes a triangular mesh from the prototypes. This is visualized in Figure 11.3. We can see that the contouring algorithm


a. *Mesh generation I*

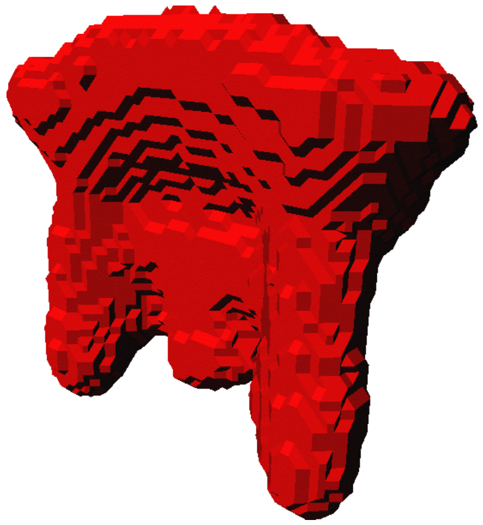

b. *Mesh generation II*

Figure 11.3: *Last step of the occlusion handling. A 3D mesh is generated from each prototype. Fig. (a) shows the mesh for the big windows and Fig. (b) the prototype mesh for the roof windows.*

is relatively robust against outliers and that the shape of the windows is modeled very accurately. Of course we also see the drawback of the contouring algorithm which consists in the artifacts caused by the underlying 3D grid structure.

After applying the last step, namely the back-projection into the scene, we obtain the mesh shown in Figure 11.4. In the scene, all objects have been replaced by the prototypes of their respective kinds. Note that this holds for all objects in the scene, including the wall and the gutter. The difference compared to the window class is only that for these classes only one object occurs in the data. This means that the prototype of the class is equal to the object encountered. However, for the partially occluded objects, the algorithm was able to recover the full structure.

11.4 Conclusions and Future Work

We presented a possible application of the object classification algorithm derived in chapter 9. Our goal was to recover the shape of objects that were partially occluded during the data acquisition process. The way this is done is by using the knowledge about the shape of other objects of the same kind that occur in the same scene. The motivation behind this was that humans resolve occlusions in a similar way, although they don't need to see objects of the same kind in the same scene. Thus, a possible extension of the algorithm would be to maintain a database of possible kinds of objects that can be used in the entity matching step after the occluded object is recognized by the AMN classifier as an object of a known kind. This would lead to a more general algorithm which is not limited to the

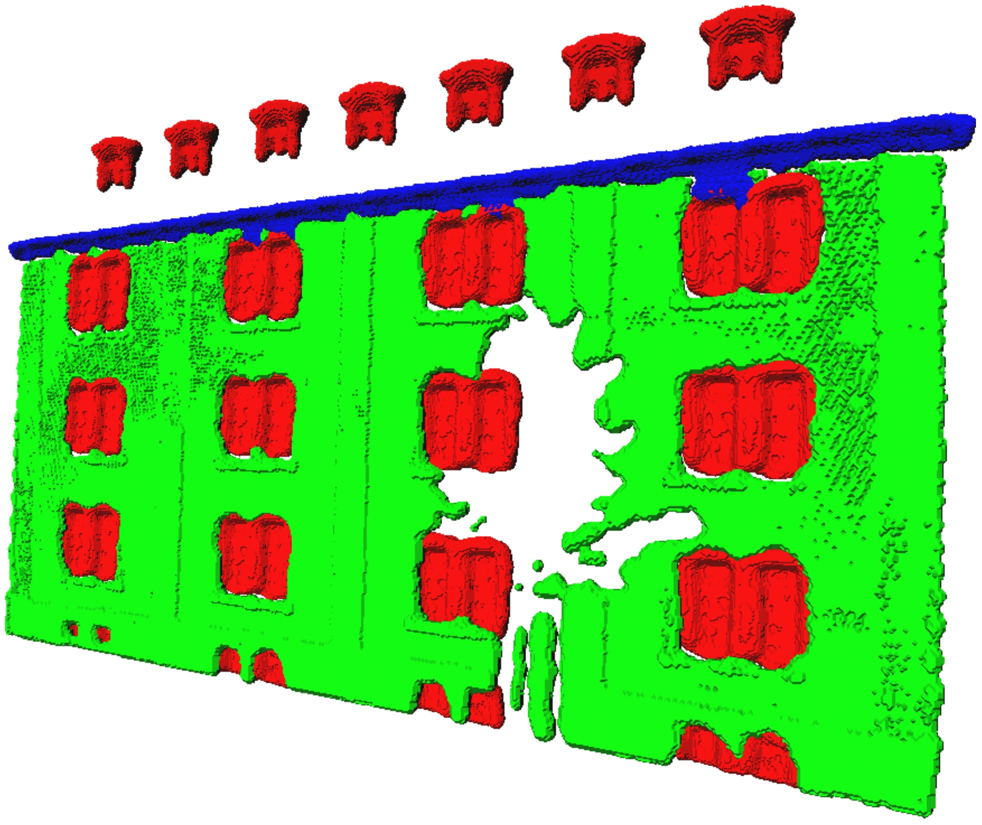


Figure 11.4: *Result obtained with our algorithm. Note that two windows in the second column have been restored. In the original data (see Figure 11.1d) these windows were occluded by a tree. Also note that the wall could not be restored, because only one wall object was encountered in the data set so that no prototype containing data in the occluded areas was obtained.*

occurrence of several objects of the same kind in the same scene.

Furthermore, we note that the shape of the objects from the data set used in our experiment is comparably simple. Therefore, it is more likely that a good matching can be found in the entity matching step. In the case of more complex objects such as chairs or tables, this does not hold, because the scan matcher can get stuck in many more local minima. Therefore, a more robust scan matching algorithm is required in such cases.

I have made this [letter] longer, because I have not had the time to make it shorter.

Blaise Pascal (1623 - 1662)

12 Discussion

12.1 Conclusions

All mobile robots operate in real environments, either indoors, for example in office environments, or outdoors where they have to cope with any different kind of terrain. The common property of these environments is that they are – as well as the robots themselves – three-dimensional, i.e. the objects that are contained in the environment extend spatially in the three directions width, depth and height. This means that any kind of representation of the environment that is based on only 1 or 2 dimensions is a projection and therefore involves the risk of omitting important information for a safe navigation through these environments. In this thesis, we address this problem by providing methods to efficiently represent and process three-dimensional data both for the task of mapping (using the MLS maps in chapter 7) as well as for the recognition of objects (using the *kd*-tree pruning in chapter 9). Furthermore, we showed that some problems from 3D environments can be solved using methods from the 2D case and adapting them adequately. Examples include the entropy-based exploration (see chapter 4) or the global scan registration (see chapter 7). In addition to these insights, we followed a conceptual strategy in this thesis, which we will present in the next section.

12.1.1 Main Concept Followed in this Thesis

This thesis presents a way to perform the task of three-dimensional perception for mobile robots. The perception task mainly consists of a compact and consistent representation, as well as the recognition of three-dimensional objects. For both of these subtasks we presented methods that either improved existing approaches or introduced new concepts to the field. The common concept to all these methods is the incorporation of additional knowledge about the particular problem. This knowledge is derived from observations that a human makes when looking at the problem, but it is not hard-coded into the algorithm. In contrast, the particular parameters that are needed to represent this additional knowledge are always obtained by the algorithm itself, rather than by the human who implements the algorithm. We believe that this combination of human modeling and the automated learning of the model parameters is the key to address complex problems in mobile robotics, not only in the field of robot perception. To illustrate the use of this concept, we explicitly analyze it in three examples from this thesis.

- **Plane Extraction** (chapter 5)**:** The main idea of the presented hierarchical EM approach is that in most indoor environments the planar structures that occur are either parallel to each other or they intersect each other in a predefined angle, which is in most cases 90 degrees. However, the algorithm does not assume this angle to be fixed, but rather estimates it adaptively. This way, it can also deal with environments where the planes are not perpendicular to each other. Thus, the modeling of the environment is made by the human, but the model parameters are estimated by the algorithm.

- **Global Pose Estimation** (chapter 7)**:** In this example the main idea is to incorporate additional knowledge about the environment in terms of a repeated occurrence of certain features. In the example we showed, a building is mapped from all sides and the fact that certain line features extracted from the particular views also occur in other views is used to improve the global view registration. Again, the model assumption of this co-occurrence of features is made by a human, but the actual position of these features is estimated adaptively by the algorithm. In essence, we can say that the algorithm is told *that* there are common line features in different views, but not *where* they are.

- **3D Object Recognition** (chapter 9) **:** For this task, we apply and extend an approach from the field of machine learning, that uses a predefined model of the posterior distribution which is to be maximized. This model is a special instance of a Markov random field (MRF) and incorporates the statistical dependence of neighboring data points. However, the particular parameters of the model, namely the weight vectors that correspond to the feature vectors of each class, are learned by the algorithm. In fact, the concept of combining a human modeler and a machine learning algorithm is applied in all approaches where an MRF is given by its structure. In recent years however, a new concept is followed in the machine learning community, which is known as *structural learning*, where the number of nodes and the edges are also learned by the algorithm. For our application, we consider this as an interesting approach that should be followed in future work.

The main advantage of this concept is that the modeling can be made by a human on a more abstract level, which makes the algorithms applicable in a more general set of situations. The author is convinced that guiding research into this direction will lead to more reliable and adaptable solutions to complex problems.

12.2 Future Work

Although we have shown that the 3D perception task can be performed in an efficient and concise way, there still remain some open questions that should be addressed in future work. We will name them here, separated by the subtasks of perception.

12.2.1 Efficient and Consistent 3D Mapping

The use of multi-level surface (MLS) maps provides a probabilistic framework to efficiently and accurately store 3D data, as we have seen in chapter 6. This approach could be further improved in several ways. For example, by adding further information such as texture or local surface normals, the expressive power of MLS maps would be improved, yet to the price of more memory requirement. This additional information would then aid in the map matching process to disambiguate feature points that have to be matched.

12.2.2 3D Object Recognition and Scene Analysis

The results of the object recognition algorithm presented in chapter 9 are promising, but they still offer room for some improvement. One problem seems to be the similarity of different objects in small scale regions. For example, the surface of a table often appears locally like the surface of a chair, because it is simply planar. Also other parts, such as the legs of a chair might look similar to corresponding parts of other objects, such as ventilators. A possible way to handle this problem might be to formulate a hierarchical approach to recognize the objects. For example, one might think of a low-level recognition of object parts and a classifier operating on a higher level that incorporates the relationship between nearby object parts. This high-level classifier might then also be collective and relational to be able to model statistical dependencies between object parts.

Furthermore, future research should focus on the question of how the object classification could generalize better. Until now, only objects can be recognized that have been learned before in the exact same shape. In an application where objects of the same kind, but with different shapes need to be recognized, this means that each object shape needs to be learned separately, which in general is impossible due to time and memory constraints. A possible way to address this problem might be to use a more abstract representation of the objects, for example by computing a graph-based structure of the objects. This would reduce the amount of data and at the same time extract more relevant features from the raw input data.

Bibliography

P. Allen, I. Stamos, A. Gueorguiev, E. Gold, and P. Blaer. Avenue: Automated site modeling in urban environments. In *Proc. of 3rd Conference on Digital Imaging and Modeling in Quebec City, Canada"*, pages 357–364, May 2001.

H. Andreasson, R. Triebel, and W. Burgard. Improving plane extraction from 3d data by fusing laser data and vision. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2005.

D. Anguelov, R. Biswas, D. Koller, B. Limketkai, S. Sanner, and S. Thrun. Learning hierarchical object maps of non-stationary environments with mobile robots. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.

D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. Discriminative learning of markov random fields for segmentation of 3d scan data. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 169–176, 2005. ISBN 0-7695-2372-2. doi: http://dx.doi.org/10.1109/CVPR.2005.133.

J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. R. L. Whittaker. Ambler: An autonomous rover for planetary exploration. *IEEE Computer Society Press*, 22(6):18–22, 1989.

J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/361002.361007.

F. Bernardini and C. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. Technical report, Departement of Computer Sciences, Purdue University, 1997.

J. E. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society*, B 36:75–83, 1974.

P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Journal on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2), February 1992.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Y. Boykov and D. Huttenlocher. A new bayesian approach to object recognition. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999.

P. Buschka and A. Saffiotti. A virtual sensor for room detection. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 637–642, Lausanne, Switzerland, 2002.

S. Chakrabarti and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proc. of the ACM SIGMOD*, Seattle, Washington, 1998.

Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1991.

P. Clifford. *Markov random fields in statistics*, chapter Markov random fields in statistics, page Markov random fields in statistics. Oxford University Press,, 1990.

L.B. Cremean, T.B. Foote, J.H. Gillula, G.H. Hines, D. Kogan, K.L. Kriechbaum, J.C. Lamb, J. Leibs, L. Lindzey, C.E. Rasmussen, A.D. Stewart, J.W. Burdick, and R.M. Murray. Alice: An information-rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics*, 2006.

DARPA. Darpa gran challenge rulebook. Website, 2004. http://www.darpa.mil/grandchallenge05/Rules 8oct04.pdf.

A.J. Davison, Y. Gonzalez Cid, and N. Kita. Real-time 3d SLAM with wide-angle vision. In *Proc. of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, 2004.

P. A. de Alarcón, A. D. Pascual-Montano, and J. M. Carazo. Spin images and neural networks for efficient content-based retrieval in 3d object databases. In *Int. Conf. on Image and Video Retrieval (CIVR)*, 2002.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

T. G. Dietterich. Machine learning for sequential data: A review. In T. Caelli, editor, *Structural, Syntactic, and Statistical Pattern Recognition*, Lecture Notes in Computer Science, page 15âĂŞ30. Springer-Verlag, 2002.

G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

N. Elmqvist and P. Tsigas. A taxonomy of 3d occlusion management techniques. In *Proceedings of the IEEE Conference on Virtual Reality*, 2007.

T.-J. Fan, G. Medioni, and R. Nevatia. Segmented descriptions of 3D surfaces. *IEEE Journal of Robotics and Automation*, RA-3(6):527–538, 1987.

D. Ferguson and A. Stentz. Field D*: An interpolation-based path planner and replanner. In *Proc. International Symposium on Robotics Research (ISRR)*, 2005.

G. Fischer. *Lineare Algebra*. Vieweg, 1995. in german.

M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. Assoc. Comp. Mach.*, 24(6):381–396, 1981.

F. Fleuret, R. Lengagne, and P. Fua. Fixed point probability field for complex occlusion handling. In *Proceedings of the Tenth IEEE International Conference on Computer Vision*, 2005.

U. Frese. *An O*(log *n*) *Algorithm for Simulateneous Localization and Mapping of Mobile Robots in Indoor Environments*. PhD thesis, University of Erlangen-Nürnberg, 2004.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory (Eurocolt)*, 1995.

Y. Freund and R.E. Schapire. Experiments with a new boosting algortihm. In L. Saitta, editor, *Thirteenth International Conference on Machine Learning*, 1996.

Y. Freund and R. E. Shapire. A decision-theoeretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977. ISSN 0098-3500. doi: http://doi.acm.org/10.1145/355744.355745.

A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik. Recognizing objects in range data using regional point descriptors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2004.

C. Früh and A. Zakhor. An automated method for large-scale, ground-based city model acquisition. *International Journal of Computer Vision*, 60:5–24, October 2004.

E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29:58–81, 2003.

S. Gottschalk. *Collision Queries using Oriented Bounding Boxes*. PhD thesis, University of North Carolina, Chapel Hill, 2000.

C. Goutte, E. Gaussier, N. Cancedda, and H. Déjean. Generative vs discriminative approaches to entity recognition from label-deficient data. In *JADT 2004 : 7es Journees internationales d'Analyse statistique des Données Textuelles*, 2004.

S. Granger and X. Pennec. Multi-scale EM-ICP: A fast and robust approach for surface registration. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2002.

J. Guivant, E. Nebot, and S. Baiker. Autonomous navigation and map building using laser range sensors in outdoor applications. *Journal of Robotics Systems*, 17(10):565–583, 2000.

J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, Monterey, CA, USA, 1999.

D. Hähnel, W. Burgard, and S. Thrun. Learning compact 3d models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 44:15–27, 2003.

R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

M. Hebert, C. Caillas, E. Krotkov, I.S. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 997–1002, 1989.

A. Hoover, J.-B. Gillian, X. Jiang, P. J. Flynn, H. Bunke, D. Goldgof, K. Bowyer, D. W. Eggert, A. Fitzgibbon, and R. B. Fisher. An experimental comparison of range image segmentation algorithms. *IEEE Journal on Pattern Analysis and Machine Intelligence (PAMI)*, 18(7):673–689, 1996.

B. K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4:629 – 642, April 1987.

D. Huber and M. Hebert. Fully automatic registration of multiple 3D data sets. *Image and Vision Computing*, 21:637–650, 2003.

D. Huber, A. Kapuria, R. R. Donamukkala, and M. Hebert. Parts-based 3d object classification. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

E. Hygounenc, I.-K. Jung, P. Souères, and S. Lacroix. The autonomous blimp project of laas-cnrs: Achievements in flight control and terrain mapping. *International Journal of Robotics Research*, 23(4-5):473–511, 2004.

L. Iocchi, K. Konolige, and M. Bajracharya. Visually realistic mapping of a planar environment with stereo. In *In Proc. of Seventh International Symposium on Experimental Robotics (ISER)*, Hawaii, 2000.

S. Jain, D. N. Osherson, J. S. Royer, and A. Sharma. *Systems That Larn: An Introduction to Learning Theory*. MIT Press, 1999.

A. Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1997.

A. E. Johnson and S. B. Kang. Registration and integration of textured 3D data. In *Int. Conf on Recent Advances in 3D Digital Imaging and Modeling*, 1997.

L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic road maps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, pages 566–580, 1996.

S. Koenig and M. Likhachev. D* lite. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2002.

P. Kohlhepp, M. Walther, and P. Steinhaus. Schritthaltende 3D-Kartierung und Lokalisierung für mobile inspektionsroboter. In *18. Fachgespräche AMS*, 2003. In German.

K Konolige. Large-scale map-making. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2004.

S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury and; M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains: Functions and integration. *International Journal of Robotics Research*, 21(10-11):917–942, 2002.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML-2001)*, 2001.

R. Lakaemper and L. J. Latecki. Extended EM for planar approximation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2006.

P. Lamon, S. Kolski, R. Triebel, R. Siegwart, and W. Burgard. The SmartTer for ELROB2006 – a vehicle for fully autonomous navigation and mapping in outdoor environments. Technical report, Ecole Polytechnique Fédérale de Lausanne and Albert-Ludwigs-Universität Freiburg, 2006a.

P. Lamon, C. Stachniss, R. Triebel, P. Pfaff, C. Plagemann, G. Grisetti, S. Kolski, W. Burgard, and R. Siegwart. Mapping with an autonomous car. In *Workshop on Safe Navigation in Open and Dynamic Environments, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2006b.

J.J. Leonard and H.J.S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *Proceedings of the Ninth International Symposium on Robotics Research*, 1999.

M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3D scanning of large statues. In *Proc. SIGGRAPH*, pages 131–144, 2000.

X. Li and I. Guskov. Multiscale features for approximate alignment of point-based surfaces. In *Eurographics Symposium on Geometry Processing*, 2005.

B. Limketkai, L. Liao, and D. Fox. Relational object maps for mobile robots. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1471–1476, Edinburgh, Scotland, 2005.

Ming C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkely, 1993.

Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, and S. Thrun. Using EM to learn 3D models with mobile robots. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (SIGGRAPH)*, 21(4):163–169, 1987.

F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

P. D. Lyons, M. Rioux, and R. T. Patterson. Application of a three-dimensional color laser scanner to paleontology an interactive model of a juvenile tylosaurus sp. basisphenoid-basioccipital. In *Palaeontologia Electronica*, volume 3, 2000.

Ó. Martínez Mozos. Supervised learning of places from range data using boosting. Master's thesis, Albert-Ludwigs-Universität Freiburg, 2004.

Ó. Martínez Mozos and W. Burgard. Supervised learning of topological maps using semantic information extracted from range data. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

Ó. Martínez Mozos, C. Stachniss, and W. Burgard. Supervised learning of places from range data using adaboost. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 1742–1747, Barcelona, Spain, 2005.

P. S. Maybeck. The Kalman filter: An introduction to concepts. In *Autonomous Robot Vehicles*. Springer Verlag, 1990.

A. S. Mian, M. Bennamoun, and R. A. Owens. Matching tensors for automatic correspondence and registration. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2004.

M. Montemerlo and S. Thrun. A multi-resolution pyramid for outdoor robot terrain perception. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2004.

H. P. Moravec. Robot spatial perception by stereoscopic vision and 3d evidence grids. Technical Report CMU-RI-TR-96-34, Carnegie Mellon University, Robotics Institute, 1996.

H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9:61–74, 1988.

P. Moutarlier and R. Chatila. An experimental system for incremental environment modeling by an autonomous mobile robot. In *1st International Symposium on Experimental Robotics*, Montreal, 1989.

A. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6d SLAM with approximate data association. In *Proc. of the 12th Int. Conference on Advanced Robotics (ICAR)*, pages 242–249, 2005.

K. Okada, S. Kagami, M. Inaba, and H. Inoue. Plane segment finder: Algorithm, implementation and applications. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Seoul, Korea, 2001.

C.F. Olson. Probabilistic self-localization for mobile robots. *IEEE Transactions on Robotics and Automation*, 16(1):55–66, 2000.

E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor estimates. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2006.

R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Matching 3d models with shape distributions. In *Shape Modeling International*, Genova, Italy, 2001.

C. Parra, R. Murrieta-Cid, M. Devy, and M. Briot. 3-d modelling and robot localization from visual and range data in natural scenes. In *1st International Conference on Computer Vision Systems (ICVS)*, number 1542 in LNCS, pages 450–468, 1999.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2nd edition, 1988.

K. Pervölz, A. Nüchter, H. Surmann, and J. Hertzberg. Automatic reconstruction of colored 3d models. In *Proc. Robotik*, 2004.

P. Pfaff and W. Burgard. An efficient extension of elevation maps for outdoor terrain mapping. In *Proc. of the International Conference on Field and Service Robotics (FSR)*, pages 165–176, Port Douglas, QLD, Australia, 2005.

P. Pfaff, R. Triebel, and W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *International Journal of Robotics Research*, 26(2), 2007a.

P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, W. Burgard, and R. Siegwart. Towards mapping of cities. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007b.

R. B. Potts. Some generalized order-disorder transformations. *Proc. Cambridge Phil Soc.*, 48, 1952.

K. Pulli. *Surface Reconstruction and Display from Range and Color Data*. PhD thesis, Univ. of Washington, 1997.

K. Pulli. Multiview registration for large data sets. In *Proc. of Int. Conf. on 3D Digital Imaging and Modeling (3DIM)*, 1999.

K. Pulli and M. Pietikäinen. Range image segmentation based on decomposition of surface normals. In *Proc. Scandinavian Conference on Image Analysis*, Norway, 1993.

C. Qiang, F. Li, and C. Ge. Feature extraction from range images in 3d modeling of urban scenes. In *Proc. of Robotics, Intelligent Systems and Signal Processing*, 2003.

J. Rodgers, D. Anguelov, H.-C Pang, and D. Koller. Object pose detection in range scan data. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

S. Ruiz-Correa, L. G. Shapiro, and M. Meila. A new paradigm for recognizing 3-d object shapes from range data. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 2003a.

S. Ruiz-Correa, L. G. Shapiro, M. Meila, and G. Berson. Discriminating deformable shape classes. In *Advances in Neural Information Processing Systems (NIPS)*, 2003b.

S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *tdim*, 2001.

K. Sabe, M. Fukuchi, J.-S. Gutmann, T. Ohashi, K. Kawamoto, and T. Yoshigahara. Obstacle avoidance and path planning for humanoid robots using stereo vision. In *Proceedings of the International Conference on Robotics and Automation (ICRA'04)*, New Orleans, April 2004.

Hanan Samet. *Applications of Spatial Data Structures*. Addison-Wesley Publishing Inc., 1989.

W. Schroeder, K. Martin, and B. Lorensen. *The Visualisation Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice Hall, 2nd edition, 1997.

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

J. A. Sethian. *Level Set Methods and Fast Marching Methods – Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.

S. Singh and A. Kelly. Robot planning in the space of feasible actions: Two examples. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1996.

C. Stachniss and W. Burgard. Exploring unknown environments with mobile robots using coverage maps. In *Proc. of the International Conference on Artificial Intelligence (IJCAI)*, 2003.

I. Stamos and M. Leordeanu. Automated feature-based range registration of urban scenes of large scale. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

A. Stentz. The focussed d* algorithm for real-time replanning. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1995.

J Sun, Y. Li, S. Bing Kang, and H.-Y. Shum. Symmetric stereo matching for occlusion handling. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

B. Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University, 2004.

B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2002.

B. Taskar, V. Chatalbashev, and D. Koller. Learning Associative Markov Networks. In *Twenty First International Conference on Machine Learning*, 2004.

S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Elsevier Academic Press, 2003.

S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2000.

S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker. A system for volumetric robotic mapping of abandoned mines. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2003a.

S. Thrun, C. Martin, Y. Liu, D. Hähnel, R. Emery Montemerlo, C. Deepayan, and W. Burgard. A real-time expectation maximization algorithm for acquiring multi-planar maps of indoor environments with mobile robots. *IEEE Transactions on Robotics and Automation*, 20(3):433–442, 2003b.

S. Thrun, Y. Liu, D. Koller, A.Y. Ng, Z. Ghahramani, and H. Durant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *International Journal of Robotics Research*, 23(7-8):693–704, 2004.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of Field Robotics*, 2006. To appear.

R. Triebel and W. Burgard. Improving simultaneous mapping and localization in 3d using global constraints. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2005.

I. Ulusoy and C. M. Bishop. Generative versus discriminative models for object recognition. In *Proceedings IEEE International Conference on Computer Vision and Pattern Recognition, CVPR*, 2005.

S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Journal on Pattern Analysis and Machine Intelligence (PAMI)*, 13(4), April 1991.

C. Urmson. *Navigation Regimes for Off-Road Autonomy*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2005.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

N. Vandapel, D. Huber, A. Kapuria, and M. Hebert. Natural terrain classification using 3-d ladar data. In *IEEE International Conference on Robotics and Automation*, 2004.

H. Wallach. Efficient training of conditional random fields. Master's thesis, Div. of Informatics, Univ. of Edinburgh, 2002.

S. Weik. Registration of 3-d partial surface models using luminance and depth information. In *International Conference on Recent Advances in 3-D Digital Imaging and Modeling (3DIM '97)*, May 1997.

J. Weingarten and R. Siegwart. Ekf-based 3d slam for structured environment reconstruction. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2005.

Carl Wellington, Aaron Courville, and Anthony Stentz. Interacting markov random fields for simultaneous terrain modeling and obstacle detection. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

Z. Wu, Y. Wang, and G. Pan. 3D face recognition using local shape map. In *Proc. of the IEEE International Conference on Image Processing*, 2004.

O. Wulf, K-A. Arras, H.I. Christensen, and B. Wagner. 2d mapping of cluttered indoor environments by means of 3d perception. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 4204–4209, New Orleans, apr 2004. IEEE.

C. Ye and J. Borenstein. A new terrain mapping method for mobile robot obstacle negotiation. In *Proc. of the UGV Technology Conference at the 2002 SPIE AeroSense Symposium*, 1994.

K. Yoshida and H. Saito. Registration of range images using texture of high-resolution color images. In *IAPR Workshop on Machine Vision Application (MVA02)*, December 2002.

D. Zwillinger, editor. *Standard Mathematical Tables and Formulae*. CRC Press, 30th edition, 1996.