Approaches to Safe and Efficient Robot Navigation

Benjamin Suger

Technische Fakultät
Albert-Ludwigs-Universität Freiburg

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

Approaches to Safe and Efficient Robot Navigation

Benjamin Suger

Dissertation zur Erlangung des akademischen Grades Doktor der Naturwissenschaften
der Technische Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

Dekan	Prof. Dr. Oliver Paul
Erstgutachter	Prof. Dr. Wolfram Burgard Albert-Ludwigs-Universität Freiburg
Zweitgutachter	Prof. Dr. Cyrill Stachniss Rheinische Friedrich-Wilhelms-Universität Bonn
Tag der Disputation	20. November 2017

Zusammenfassung

Autonome Navigation beschreibt ein aktuelles und aktives Forschungsgebiet. Dabei umfasst es einen weiten Rahmen von Themen und Problemstellungen, wie zum Beispiel: Lokalisierung, Kartierung, Bewegungsplanung und Steuerung sowie Hinderniserkennung. Ein klarer Indikator für die Aktualität des Themas ist, dass zahlreiche Firmen derzeit große Summen in Technologien investieren, welche autonome und intelligente Systeme alsbald zu einem festen Bestandteil unseres täglichen Lebens machen sollen. Eine klassische Problemstellung lässt sich an dem Beispiel eines autonomen Autos illustrieren: Es sollte jederzeit wissen wo es sich auf der Welt befindet (Lokalisierung und Kartierung), wissen welche Aktion als nächstes nötig ist, um dem angestrebten Ziel näher zu kommen (Pfadplanung und Steuerung) und flexibel auf andere Verkehrsteilnehmer und Hindernisse reagieren (Sicherheit und Hinderniserkennung). Um dies zu gewährleisten ist es notwendig, die Umgebung mit unterschiedlichen Sensoren zu erfassen und die erhaltenen Messdaten der Aufgabe entsprechend zu interpretieren. Dabei muss gewährleistet sein, dass die verfügbaren Ressourcen wie Rechenleistung und Speicherkapazität jederzeit ausreichen, um alle notwendigen Berechnungen hinreichend schnell auszuführen. Darüber hinaus soll das System unabhängig von Veränderungen der Umwelt zuverlässig arbeiten.

In dieser Doktorarbeit behandeln wir unterschiedliche Themen zu diesem Gebiet. Dabei sind wir speziell an der Effizienz, Robustheit und an der Sicherheit von autonomen Systemen interessiert. Zuerst behandeln wir das Thema Lokalisierung und Kartierung. Dabei beschäftigen wir uns zu Beginn (Kapitel 3) mit dem speziellen Problem der simultanen Lokalisierung und Kartierung (SLAM), bei welchem ein mobiler Roboter den Kartierungsprozess simultan mit dem Lokalisierungsprozess ausführen muss. Dieses Problem lässt sich mathematisch als Graph darstellen, wobei die Knoten des Graphen den Posen des Roboters entsprechen. Die Kanten, welche zwei Knoten verbinden, repräsentieren relative Transformationen. Diese werden aus den Sensordaten errechnet. Zur Lösung des SLAM-Problems wird ein äquivalentes nichtlineares Minimierungsproblem mit Hilfe der Methode der kleinsten Quadrate gelöst. Unser spezielles Augenmerk richten wir dabei darauf, solche Lösungen mit möglichst wenig Arbeitsspeicher zu berechnen. Hierbei ist zu beachten, dass im Allgemeinen der benötigte Arbeitsspeicher zum Lösen eines solchen Problems mit der Anzahl der Knoten und Kanten wächst. Unser Ansatz macht sich eine hierarchische “Teile- und Herrsche-Strategie” zunutze und zerlegt den Graphen in kleinere stochastisch unabhängige Teilprobleme, welche mit sehr viel weniger Speicheraufwand gelöst werden können. Im Zuge dieser Konstruktion müssen wir Randverteilungen approximativ darstellen, da ansonsten die Konnektivität des Graphen zu dicht wird und die Speichervorteile nicht realisiert werden können. Dies führt im Sinne der Optimierung zu leicht schlechteren Werten der Zielfunktion. In der experimentellen Auswertung unseres Ansatzes zeigen wir, dass der Qualitätsverlust verglichen zur exakten Lösung des Problems gering ist. Dabei benötigen wir signifikant weniger Arbeitsspeicher. Darüber hinaus erhalten wir im Vergleich mit bisherigen approximativen Lösungsansätzen mit unserer Methode bessere Lösungen. Unser Ansatz erweist sich daher speziell für Systeme

mit begrenztem Arbeitsspeicher als vorteilhaft.

Ist eine Umgebung einmal kartiert kann sich ein Roboter darauf beschränken, seine Position in der Karte zu ermitteln. Das Problem der Selbstlokalisierung behandeln wir in den zwei darauf folgenden Kapiteln. Hierbei gilt unser besonderes Interesse Situationen in denen sich die Umgebung über die Zeit verändert. In Kapitel 4 betrachten wir eine Problemstellung, welche wir besonders in industriellen Anwendungen als relevant ansehen. Unser Interesse gilt der relativen Position zwischen Roboter und einem Zielobjekt, z.B. ein Tisch oder eine Palette. Dabei nehmen wir an, dass der Roboter zur Wahrnehmung der Umgebung mit einem Laserscanner ausgerüstet ist. Wir betrachten ein Szenario, in welchem das Zielobjekt mit der Zeit seine Position verändert, beispielsweise wenn der Tisch durch Interaktion leicht verschoben wird. Dies führt dazu, dass die globale Positionsschätzung und die relative Positionsschätzung inkonsistent sind und das Problem somit nicht mit Standardmethoden gelöst werden kann. In unserem Ansatz modellieren wir das Zielobjekt explizit als Bestandteil der Karte und verwenden eine Formulierung, welche es erlaubt Referenzposen bezüglich mehrerer Objekte gleichzeitig zu schätzen. Um dies zu erreichen wird ein Algorithmus zur Berechnung der relativen Transformation zwischen zwei Messpositionen eines Lasersensors dahingehend erweitert, dass dieser die relativen Positionen zu mehreren Objekten gleichzeitig schätzt. Die experimentelle Auswertung zeigt, dass mit unserem Ansatz sehr genaue Positionsschätzungen erreicht werden, auch wenn die Zielobjekte mit der Zeit ihre Postion verändern. Dabei nutzen wir sowohl simulierte Daten, wie auch Daten von einem echten Roboter.

Im Vergleich zu Laserscannern bieten Kamera-basierte Systeme einen deutlichen Kostenvorteil und werden daher oft verwendet. Außerdem bieten sie Informationen über relativ große Bereiche der Umgebung und als Menschen sehen wir die auf Bildern basierte Wahrnehmung als natürlich an. In Kapitel 5 betrachten wir ein Kamera-basiertes Lokalisierungsproblem in welchem der Roboter starke Veränderungen der Umwelt erfährt. Verschiedene Beleuchtungsverhältnisse oder jahreszeitlich bedingte Veränderungen erschweren beispielsweise die Beurteilung darüber, ob zwei Bilder am gleichen Ort entstanden sind. In unserem Lösungsansatz machen wir uns die Tatsache zu Nutze, dass Bildsequenzen üblicherweise während einer kontinuierlichen Fahrt des Roboters aufgenommen werden. Unser spezielles Interesse gilt Situationen, in denen zwischen zwei Aufnahmen von Bildsequenzen eine große Zeitspanne vergangen ist und sich die Umgebung jahreszeitlich bedingt stark verändert hat. Dabei beschreiben wir eine Methode, um die Ähnlichkeit zwischen zwei Bildern zu quantifizieren und nutzen diese Werte als Sensormodell in einem probabilistischen Zustandsmodell. Das Verfahren zur Zustandsschätzung basiert auf einem Bayes-Filter. Darüber hinaus zeigen wir den Nutzen einer Datenvorbearbeitung auf, welche die Unterscheidbarkeit der erhaltenen Ähnlichkeitswerte erhöht. Dies führt zu einer Verbesserung in Bezug auf Genauigkeit und Robustheit der Lokalisierung. In einer umfassenden experimentellen Auswertung zeigen wir, dass der vorgeschlagene Ansatz eine Verbesserung gegenüber bisherigen Ansätzen darstellt.

Eine weitere, äußerst wichtige, Aufgabe für autonome Systeme ist, dass sie mit Hilfe von Sensordaten erkennen müssen welche Bereiche ihrer Umwelt sicher befahren werden können und wo sich Hindernisse befinden. Das ist besonders wichtig, da dies nicht nur die Sicherheit des Roboters, sondern auch die Sicherheit aller in der Umwelt befindlichen Gegenstände gewährleistet. Andererseits darf diese Erkennung nicht zu sensibel im Sinne der Sicherheit sein, da der Roboter sonst eventuell Hindernisse erkennt wo keine sind und somit an seiner Weiterfahrt gehindert wird. In dieser Arbeit betrachten wir dieses Problem auf zwei Arten, wobei wir stets einen 3D-Laserscanner zur Wahrnehmung der Umgebung verwenden.

Zuerst entwickeln wir in Kapitel 6 einen Ansatz der darauf abzielt, dass ein nicht speziell ausgebildeter Nutzer dem Roboter beibringen kann welche Bereiche befahrbar sind. Um diese Lernphase so einfach wie möglich zu gestalten, ist es lediglich notwendig den Roboter manuell sicher durch eine Umgebung zu steuern. Ein vollständiges Modell zur Traversierbarkeit leiten wir von den positiven Beispielen ab, welche von der Trajektorie des Roboters als eindeutig befahrbar klassifiziert worden sind. Zur Lösung dieses speziellen Lernproblems, bei dem wir keine Beispiele der negativen Klasse zur Verfügung haben, greifen wir auf bereits existierende Methoden zurück und zeigen auf, dass diese auch für die Traversierbarkeitsanalyse angewendet werden können. Unsere experimentelle Auswertung zeigt die Genauigkeit der abgeleiteten Traversierbarkeitsmodelle. Die vorgestellte Methode unterliegt der Annahme einer vollständig statischen Umgebung und hat den Nachteil, dass die Sensordaten zuerst akkumuliert werden und somit nicht unmittelbar in die Abwägung zur Traversierbarkeit einfließen.

Diesem Problem widmen wir uns in Kapitel 7, wobei wir uns die Erfahrungen aus dem vorhergehenden Kapitel zu Nutze machen. Hindernisse zeichnen sich oft durch grundlegende geometrische Eigenschaften aus, wie zum Beispiel die Höhendifferenz in der unmittelbaren Nachbarschaft. Dabei müssen wir die Art des zugrundeliegenden Geländes mit in Betracht ziehen, da zum Beispiel hohes Gras für einen großen Roboter kein Hindernis darstellt jedoch große Höhendifferenzen aufweist. Um auf diese Situationen angemessen zu reagieren nutzen wir zusätzliche semantische Informationen über das zugrundeliegende Terrain, welche wir ausschließlich aus den 3D-Laserdaten ableiten. Um zu entscheiden was ein Hindernis darstellt führen wir beide Informationen in einem probabilistischen Modell zusammen. Der Vorteil ist nun, dass wir die geometrischen Information extrahieren können sobald die Messungen gemacht wurden. Die Integration der Sensordaten zu Klassifizierung der Geländeart, welche zweifelsfrei als statisch angesehen werden kann, findet in regelmäßigen Abständen statt. Unsere Experimente mit einem autonom navigierenden Roboter zeigen die Anwendbarkeit und Effizienz der vorgeschlagenen Methode, im Sinne der Rechenlast.

Zum Ende dieser Arbeit (Kapitel 8) entwickeln wir einen Ansatz, um globale Navigation unter Verwendung von öffentlich zugänglichem Kartenmaterial zu ermöglichen, wofür wir die Daten von OpenStreetMap nutzen. Das Problem dabei ist, dass wir sowohl Ungenauigkeiten in der Karte, als auch in den GPS-Messungen in Betracht ziehen müssen. Diese Fehler führen dazu, dass Zielpunkte aus der Sicht des Roboters nicht immer auf dem angestrebten Weg, sondern manchmal mehrere Meter abseits liegen können. In unserem Ansatz korrigieren wir diese Fehler indem wir die Terrain-Klassifikation aus dem vorangegangenen Kapitel nutzen. In einem probabilistischen Modell assoziieren wir damit die Zielpunkte vom Straßennetz der Karte mit den Wegen in der Umgebung des Roboters. Unser Ansatz ermöglicht autonome Navigation in Bereichen, die zuvor noch nicht von einem Roboter erkundet wurden. Dies ist in herkömmlichen Systemen üblicherweise notwendig. Darüber hinaus modelliert der Roboter während er autonom navigiert ausschließlich seine lokale Umgebung und benötigt somit nur eine feste, zuvor berechenbare Menge an Hauptspeicher. Wir zeigen die Anwendbarkeit des Ansatzes mit Experimenten, bei denen ein echter autonomer Roboter in zuvor ungesehenen Gebieten navigiert.

Abstract

Autonomous navigation is a broad and active research area, which has attracted great interest in the last two decades. Nowadays, more and more companies invest lots of money and effort towards making autonomously navigating robots part of our daily lives. Key components of autonomous systems are mapping, localization, perception, planning and control. To achieve real autonomy, a robot needs to cope with limited on-board resources, changes of the environment and varying external conditions. Therefore, such systems need to infer meaningful and accurate interpretations from the available sensor data. In this thesis, we address several of these challenges, specifically: resource efficiency, changing and complex environments, traversability analysis and map interpretation.

First, we present an approximate solution to the problem of simultaneous localization and mapping (SLAM), particularly to the graph-based formulation of the problem. So far, state-of-the-art solutions to SLAM have put their main focus on speed and accuracy. Contrary to that, we provide a fast approximate solution to the SLAM problem that needs significantly less memory than other state-of-the-art methods. Subsequently, we address the problem of localization, focusing on two aspects related to changes of the environment. First, we consider an industrial scenario in which some crucial parts of the environment, *i.e.* at which there are objects that the robot needs to fulfill a task, may change their position over time. In this case, the global pose of the robot is not consistent with the pose relative to the object. To overcome this problem, we propose a variation for point-set registration, which can cope with multiple reference frames at the same time. Second, we discuss the problem of visual localization when facing severe changes of the environment. To cope with such extreme conditions, we exploit the sequential property of images recorded during continuous runs of the robot.

Besides the knowledge of its position in the world, an autonomous robot needs to know which parts of the world are safe to traverse and which are not. We discuss this topic from two perspectives. First, we present an approach to learning the traversability characteristics for a mobile robot from non-expert human demonstrations. The key challenge here is that the learning technique needs to cope with only positive examples of traversable ground. Second, we focus on the computational efficiency, taking into account that the crucial information about obstacles should be available to the robot as soon as possible. Thereby, we aim to keep the computational effort as low as possible, as resources may be limited.

Finally, we discuss an approach that makes publicly available map data from the OpenStreetMap (OSM) project useful to autonomous navigation in outer-urban environments. To cope with the errors in the map, we utilize semantic terrain information to align the tracks in OSM with the local vicinity of the robot. With our approach, a robot can navigate autonomously in previously unseen environments, which makes exploration and mapping of the environment in advance, unnecessary.

In this thesis, we present a variety of techniques and algorithms for autonomous navigation. In extensive real world experiments, we demonstrate the applicability and accuracy of the proposed methods.

Acknowledgments

Writing a PhD thesis is a big thing and without the support of many people this would have been much harder or even impossible. I am grateful for everyone of you, however, to keep the scope of this section in bounds I can only thank some outstanding supporters and companions.

First of all, I want to thank my supervisor Wolfram Burgard for his support, encouragement, inspiration and guidance. It was great, and cannot be seen as granted, to enjoy so much freedom, trust and opportunities in the exploration of ideas as a PhD student.

Great thanks to my co-advisers Bastian Steder, Gian D. Tipaldi and Luciano Spinello for all their support. It was great to discuss and solve problems as well as writing papers together. I really enjoyed the time spent with Bastian tracing down bugs and fixing undesired behavior of our lovely robot Viona, as well as seeing her navigating autonomously based on the code we developed.

Moreover, I want to thank all my colleagues who joint my way during the past years. We had great times, productive discussions and tons of chitchat. A special thanks goes to my office mates Jörg, Tayyab and Chau. It was great to explore the complementary effects of the different knowledge bases we had. Despite all the serious work we had a lot of fun. Furthermore, I want to thank Max for uncountable, valuable discussions and Christoph for his imperturbably willingness to help and especially for sharing his expertise in TiKZ. Furthermore, I want to thank Susanne Bourjaillat and Manuela Kniss for their support concerning all kinds of administrative stuff.

Infinite thanks to my beloved family. My children Liam and Iuna for making my life more exciting and Anki, my dear wife, for the inexhaustible support and her unshakable faith in me. We will never forget that Iuna was born in the night right before the submission deadline for ICRA, when I hurried from the office, where we were polishing our submission, directly to the delivery room. Furthermore I want to express my deep gratitude to my parents, which did never hesitate to offer help by taking care of the kids when the amount of work was large and time was scarce.

This work was partially founded by the DFG "Graduate School Embedded Systems" (GR1103), the EU projects LifeNav (ERC-AG-PE7-267686-LIFENAV) and EUROPA2 (FP7-610603-EUROPA2), and the German Federal Ministry of Education Research (BMBF) project NaRKO (01IS15044B-NaRKo). Many thanks to everyone who was involved in writing the proposals for these projects.

Contents

Contents **xiii**

1 Introduction **1**
1.1 Scientific Contribution . . . 2
1.2 Publications . . . 3
1.3 Collaboration . . . 4

2 Basics **5**
2.1 Notation . . . 5
2.2 Acronyms and Abbreviations . . . 6
2.3 Simultaneous Localization and Mapping . . . 7
2.3.1 Graph-SLAM . . . 7
2.4 Monte Carlo Localization . . . 8
2.5 Generalized Iterative Closest Point Algorithm . . . 8
2.6 Classification . . . 9
2.6.1 Random Forest Classifier . . . 10

3 Memory Aware Considerations of SLAM **11**
3.1 Mapping with Low Memory Consumption . . . 12
3.1.1 Graph Partitioning . . . 14
3.1.2 Leaves-to-Root Coarsening . . . 14
3.1.3 Root-to-Leaves Optimization . . . 16
3.2 Memory Consumption Analysis . . . 18
3.3 Experiments . . . 18
3.3.1 Memory Consumption . . . 19
3.3.2 Runtime on Systems with Restricted Main Memory . . . 19
3.3.3 Metric Accuracy . . . 20
3.4 Related Work . . . 22
3.5 Conclusions and Future Work . . . 24

4 Localization with Respect to Non-Stationary Objects **25**
4.1 Localization with Respect to Multiple References . . . 26
4.1.1 Generalized ICP for Multiple Rigid Bodies . . . 28
4.2 Point Cloud Generation . . . 30
4.2.1 Generating the Reference Point Cloud . . . 30
4.2.2 Local Point-set Registration . . . 31
4.3 Experiments . . . 32
4.3.1 Simulation Experiment . . . 33

4.3.2 Real-World experiments . . . 35
4.4 Related Work . . . 36
4.5 Conclusion and Future Work . . . 38

5 Camera-based Localization Facing Substantial Perceptual Changes 39
5.1 Visual Localization Utilizing Sequential Information . . . 41
5.1.1 Robust Image Matching . . . 41
5.1.2 Discrete Bayes Filter . . . 43
5.1.3 State Transition Model . . . 44
5.1.4 Sensor Model . . . 44
5.2 Forward Backward Smoothing . . . 45
5.3 Sequential Filtering . . . 45
5.4 Zero Component Analysis Whitening . . . 46
5.5 Experiments . . . 48
5.5.1 Scattered Trajectories . . . 51
5.5.2 Connected Trajectories . . . 53
5.5.3 NewCollege . . . 54
5.5.4 Parameter Discussion . . . 55
5.6 Related Work . . . 55
5.7 Conclusion and Future Work . . . 57

6 Semi-Supervised Learning of Traversability Models 59
6.1 Basic Structure . . . 61
6.2 Feature Design . . . 61
6.3 The Learning Problem . . . 62
6.3.1 Positive Naive Bayes . . . 63
6.3.2 Learning from Positive Only Examples . . . 66
6.3.3 Terrain Models . . . 66
6.3.4 Training . . . 67
6.4 Experiments . . . 67
6.4.1 Evaluation using Viona . . . 67
6.4.2 Evaluation using Obelix . . . 70
6.5 Related Work . . . 70
6.6 Conclusion and Future Work . . . 72

7 Adaptive Obstacle Detection 73
7.1 Online Obstacle Detection . . . 75
7.1.1 Velodyne Intrinsics . . . 75
7.1.2 Geometric considerations . . . 75
7.1.3 Basic Obstacle Detection . . . 76
7.2 Terrain Analysis . . . 78
7.2.1 Features . . . 79
7.2.2 Classification . . . 80
7.2.3 Terrain-Class-Map . . . 80
7.3 Terrain-Adaptive Obstacle Detection . . . 81

7.4 Experiments . . . 83
7.4.1 Illustration of Mixed Terrain Challenge . . . 83
7.4.2 Terrain Classification Accuracy . . . 84
7.4.3 Real World with Computational Analysis . . . 84
7.4.4 Effect of Approximations . . . 86
7.5 Related Work . . . 88
7.6 Conclusion and Future Work . . . 89

8 Outer-Urban OpenStreetMap-based Autonomous Navigation 91
8.1 Preliminaries . . . 93
8.1.1 Planning on OpenStreetMap . . . 94
8.1.2 Semantic Terrain Information . . . 94
8.2 Subgoal Alignment . . . 94
8.2.1 Probabilistic Formulation . . . 95
8.2.2 Process Model . . . 96
8.2.3 Measurement Model . . . 97
8.3 Sequential Markov-Chain Monte-Carlo Sampling . . . 97
8.4 Experiments . . . 98
8.4.1 System Setting . . . 99
8.4.2 Performance . . . 99
8.4.3 Intersections . . . 101
8.4.4 Runtime . . . 103
8.4.5 Limitations . . . 103
8.5 Related Work . . . 104
8.6 Conclusions . . . 105

9 Conclusion 107

Bibliography 115

Chapter 1

Introduction

Autonomous navigation embraces a broad range of problems, each of which constitute their own research area, *e.g.,* mapping and localization, planning, control and traversability analysis. When we think about a perfect autonomously navigating robot, our expectations of the capabilities of such a system are usually high. The robot should: know its exact position in the world at any point in time, be able to handle changes of environment that occur over time, and be able to know social behaviors and rules such as driving on the correct side of the street and circumvent pedestrians in a way we would regard as natural. It should be also able to perceive its nearby vicinity and react appropriately, based on a sophisticated interpretation of the sensor data. Moreover, we would expect that such a system could navigate autonomously, independent of external conditions, *e.g.,* season, weather and daytime.

We believe that a truly autonomous system should be able to perform all necessary operations on-board and, therefore, the algorithms need to take the problem of scarce resources into account. This becomes especially relevant when we consider autonomous long-term operation. Another important component is the choice of the perceptual sensor equipage of a robot. In this thesis, we will mostly rely on laser scanners, since they provide accurate geometrical data and are rather insensitive to lighting conditions, in contrast to other common sensors, *e.g.,* cameras. Especially in industrial applications, laser scanners are already commonly used, due to safety relevant issues. However, for the consumer market, cameras are of great interest, due to their low prices, and we will discuss a localization problem facing severe perceptual changes. When reasoning about traversability characteristics of the environment, the additional value of accurate geometric information, as provided by 3D-laser sensors, are of utmost importance, even though the pure geometric information alone may be not sufficient. So far, autonomous systems need an appropriate map of the environment to perform navigation tasks. Given that there are publicly available map-services that provide almost complete map data of road networks this effort seems avoidable. As humans, we are perfectly capable of navigating using these maps and a GPS device, even though the map and the GPS estimate is often several meters off. So, we would also expect that autonomously navigating robots could make use of this data in a similar way.

In this thesis, we will address several of the aforementioned problems. In the remainder of this chapter, we give an outline of the topics addressed in this thesis. Then, we briefly summarize the scientific contributions, followed by the list of publications, and discuss collaborations for the specific chapters.

Afterwards, we briefly introduce some basic methods that are used throughout this thesis in Chapter 2. In Chapter 3 we present an approach to solving the Graph-SLAM problem with a small memory footprint. In order to do this, we propose a hierarchical decomposition

of the problem, for which we use an efficient approximation of the marginal distribution and consequently subdivide the problem to multiple sub-problems, each of bounded size. The experimental evaluation shows that our approach needs significantly less memory than state-of-the-art SLAM solvers, and is more accurate than other state-of-the-art approximate solvers.

In the following two chapters, we discuss two challenging localization problems. First, in Chapter 4, we are interested in an accurate localization with respect to objects that may change their positions over time; when this happens, there is an inconsistency between the relative reference pose to the object and the global pose. To overcome this problem, we propose a variation of the ICP-algorithm that handles multiple reference frames for different rigid bodies. In real-world and simulated experiments, the proposed approach achieves an accuracy that is sufficient for industrial tasks. Secondly, in Chapter 5, we address the problem of visual localization facing substantial changes in the environment, caused by *e.g.,* seasonal changes. We explore the sequential properties of the image streams and apply a Bayesian filter framework to find correspondences between two image sequences recorded at substantially different times. In extensive experiments we show the advantages of the presented approach and the advancement over the state-of-the-art.

In Chapter 6 and Chapter 7 we address the problem of traversability analysis and obstacle detection. First, in Chapter 6, we present an approach that infers a model for traversability analysis from positive only demonstrations. The method is specifically designed so that a non-expert user could teach the robot by manually operating it safely through the environment. Second, in Chapter 7, we focus on computational efficiency for real-time obstacle detection in an all terrain scenario, *e.g.,* regular streets, dirt– and forest–roads and grassland. To achieve a reliable obstacle detection in such environments, we combine fast to compute geometric measures with semantic terrain information, which is updated with a lower frequency. We evaluate both approaches in real-world experiments, in which we applied the latter to a real autonomously navigating robot.

Finally, we present an approach to autonomous navigation using data from OpenStreetMap (OSM) in Chapter 8. This aims to overcome the requirement of an area having been explored before an autonomous system can navigate it. The presented approach makes data from OSM useful to global path planning in outer urban environments and, therefore, allows for autonomous navigation in previously unseen areas. We use the semantic information that we already extract for the obstacle detection in the former chapter, to account for errors that are induced by inaccuracies in OSM and the GPS measurements. In real world experiments, we use this system for autonomous navigation at different and previously unseen areas.

At the very end, in Chapter 9, we discuss the techniques and results presented in this thesis. Therein, we will also identify possible directions for future work, with the goal of further advancing the state-of-the-art in autonomous navigation.

1.1 Scientific Contribution

This thesis describes several contributions to different areas of autonomous navigation, *e.g.,* SLAM, localization, perception and traversability analysis. In summary, we present:

- an approximate solution to the SLAM problem that needs significantly less memory than state-of-the-art solutions (Chapter 3).

- a localization method that aims to localize a robot with respect to an object of interest, e.g., a work bench or a shelf. Thereby, we relax the static world assumption and take into account that this object may undergo movements over time (Chapter 4).
- an approach to localizing a robot, based on image sequences. We consider that the environment undergoes substantial changes, e.g., different seasons (Chapter 5).
- an approach to learning the traversability capabilities of a mobile robot from positive only demonstrations (Chapter 6).
- an approach to fast and efficient obstacle detection in mixed terrain scenarios, exploiting additional semantic terrain information (Chapter 7).
- an approach to outer-urban autonomous navigation, in previously unseen environments, utilizing the knowledge of a road-network provided by public map services, *e.g.,* OpenStreetMap (Chapter 8).

1.2 Publications

The content of this thesis is based on publications in international journals and conference proceedings. In the remainder of this section, we list this publications chronologically.

Journal Articles:

- Tayyab Naseer, Benjamin Suger, Michael Ruhnke und Wolfram Burgard. **Vision-based Markov Localization for Long-term Autonomy**. In *Journal of Robotics and Autonomous Systems*, vol. 89, pages 147 – 157, 2017.

Conference Articles:

- Benjamin Suger and Wolfram Burgard, **Global Outer-Urban Navigation with OpenStreetMap**, *To appear* in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, Singapore, 2017.
- Benjamin Suger, Bastian Steder und Wolfram Burgard. **Terrain-Adaptive Obstacle Detection**. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea, 2016.
- Jörg Röwekämper, Benjamin Suger, Wolfram Burgard und Gian D. Tipaldi. **Accurate Localization with Respect to Moving Objects via Multiple-Body Registration**. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015.
- Tayyab Naseer, Benjamin Suger, Michael Ruhnke und Wolfram Burgard. **Vision-Based Markov Localization Across Large Perceptual Changes**. In *Proceedings of the IEEE European Conference on Mobile Robotics (ECMR)*, Lincoln, UK, 2015.

- Benjamin Suger, Bastian Steder und Wolfram Burgard. **Traversability Analysis for Mobile Robots in Outdoor Environments: A Semi-Supervised Learning Approach Based on 3D-Lidar Data**. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, USA, 2015.

- Benjamin Suger, Gian Diego Tipaldi, Luciano Spinello und Wolfram Burgard. **An Approach to Solving Large-Scale SLAM Problems with a Small Memory Footprint**. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014. **Best Conference Paper Award Finalist**

1.3 Collaboration

This section describes the collaboration with other researchers during the process of thesis creation. The following list explains the role of the author for each individual chapter.

- Chapter 3: The approach presented in this chapter is the result of collaboration with Gian D. Tipaldi, Luciano Spinello and Wolfram Burgard. The implementation of the approach was fully done by the author of this thesis. The related publication, Suger et al. [113], was nominated for the conference best paper award in ICRA 2014.

- Chapter 4: The approach presented in this chapter is the result of collaboration with Jörg Röwekämper, Wolfram Burgard and Gian D. Tipaldi. The implementation was mainly done by J. Röwekämper, while the author of this thesis was involved in the theoretical discussion and the experimental evaluation. See Röwekämper et al. [99] for the related publication.

- Chapter 5: The approach presented in this chapter is the result of collaboration with Tayyab Naseer, Michael Ruhnke and Wolfram Burgard. The basics of the approach, the feature extraction and datasets have been contributed by my colleague, Tayyab Naseer. The author of this thesis implemented the recursive filter and the sequential selection algorithm. The chapter is based on Naseer et al. [84, 85]. The paper that describes the initial approach ([84]) was voted as one of the ten best conference papers by the participants.

- Chapter 6 and Chapter 7: The system presented in this section is the result of collaboration with Bastian Steder and Wolfram Burgard. The approaches presented in those chapters are part of the autonomous outdoor robot system that was used in the context of the EU project LifeNav (ERC-AG-PE7-267686-LifeNav). My colleague Bastian Steder, particularly, contributed to Chapter 7 with the implementation of the fast geometric feature. For related publications see Suger et al. [114, 115].

- Chapter 8: The approach presented in this chapter is the result of collaboration with Wolfram Burgard. The implementation of the approach was fully done by the author of this thesis, except for the A*-planner on the OSM, which was implemented by my colleague Philipp Ruchti. For the related publication see Suger and Burgard [112].

Chapter 2

Basics

In this preliminary chapter, we first give an overview of some notations and abbreviations, which we will use frequently throughout this thesis, in Section (2.1) and Section (2.2). In the remainder of this chapter, Section (2.3) to Section (2.6), we provide a brief introduction to some basic concepts, which build the base for the approaches that we will present in the course of this thesis.

2.1 Notation

Throughout this thesis, we will use the following mathematical notations:

Example	**Type**	**Description**
$x = 1.0$, $\varphi = 0.3$, $\alpha = 10^\circ$	Scalar	Lower case Latin and Greek letters.
$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix}$	Vector	Bold lowercase Latin letters.
$S = \Sigma = \begin{pmatrix} x_{11} & x_{12} & \cdots \\ x_{21} & x_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$	Matrix	Uppercase Latin or Greek letters.
$\mathcal{S} = \{x_1, x_2, \cdots\}$	Set	Uppercase calligraphic letters
$\mathfrak{M} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots\}$	Set of sets	Uppercase fracture symbols
$\lvert x \rvert = \begin{cases} x & \text{if } x \geq 0, \\ -x & \text{else.} \end{cases}$	Absolute value	A scalar enclosed by $\lvert\cdot\rvert$
$\lvert\mathbf{x}\rvert = \left\lvert \begin{pmatrix} x_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} \right\rvert = n$	Length/Size	A vector or a set enclosed by $\lvert\cdot\rvert$
$\lVert\mathbf{x}\rVert_p = \left(\sum_{i=1}^{\lvert\mathbf{x}\rvert} \lvert x_i \rvert^p \right)^{\frac{1}{p}}$	L^p-norm	A vector enclosed by $\lVert\ \rVert_p$, whereas $\lVert\ \rVert_2$ is also called the Euclidean norm

$\|\mathbf{x}\|_\Sigma = \sqrt{\mathbf{x}^\mathrm{T}\Sigma\mathbf{x}}$	Malahanobis norm	A vector enclosed by $\| \ \|_\Sigma$, whereas Σ is a positive definite matrix
$\oplus$ and $\ominus$	Compound operator and its inverse	Compound operator for transformations, see Smith et al. [108] for details.
$\delta_\mathcal{A}(x) = \begin{cases} 1 \text{ if } x \in \mathcal{A} \\ 0 \text{ else} \end{cases}$	Dirac/indicator function	The Dirac- or indicator function for the set $\mathcal{A}$ returns one if and only if the argument is an element of $\mathcal{A}$

2.2 Acronyms and Abbreviations

Besides the mathematical notations of the former section we will use the following acronyms and abbreviations:

Abbreviation	**Meaning**	**Description**
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping	A fundamental problem of mobile robotics, which concerns the solution of the localization and the mapping problem simultaneously.
MCL	**M**onte **C**arlo **L**ocalization	A localization method based on sampling.
ICP	**I**terative **C**losest **P**oint	Iterative method to estimate the relative transformation between tow point-sets observed from different positions.
RF	**R**andom **F**orest	An ensemble of trees classifier
IMU	**I**nertial **M**easurement **U**nit	A sensor measuring orientations
GPS	**G**lobal **P**ositioning **S**ystem	Satellite-based global positioning system
HOG	**H**istogram of **O**riented **G**radients	Features or descriptors for images, usually calculated at keypoints
SIFT	**S**cale-**I**nvariant **F**eature **T**ransform	
SURF	**S**peed **U**p **R**obust **F**eatures	
DCNN	**D**eep **C**onvolution **N**eural **N**etwork	Neural network with a broad range of applications
fps	**f**rames **p**er **s**econd	The number of frames a camera delivers per second
ZCA	**Z**ero **C**omponent **A**nalysis	Decorrelation method in data analysis

2.3 Simultaneous Localization and Mapping

The problem of simultaneous localization and mapping (SLAM) is a fundamental problem of autonomous navigation. The literature distinguishes between the cases of *mapping with known poses* and *localization* with a given map. Both problems have been solved previously, however, for localization: the map needs to be known in advance and for mapping: the poses of the robot are required. SLAM is the problem to solve these mutually dependent problems at the same time, which means that the robot needs to build the map from poses that are derived from the localization within that map. Not for nothing, this problem was regarded as a chicken-egg problem for some time. The key to solving the SLAM problem are so called *loop closures*, which means that the robot recognizes, from sensor observations, that it has been at the same area before.

Possible approaches to SLAM are based on filtering techniques, as they use the Kalman-filter [65, 108, 119] or the particle filter [39, 78]. A more recent method, models SLAM as a graph and solves an equivalent least-squares minimization problem [36, 40, 43, 57, 67, 92]. This method has gained popularity due to the robustness and quality of the solutions. As we rely on the formulation of this method in Chapter 3 and Chapter 4, we give a brief summary of this technique in the forthcoming section.

2.3.1 Graph-SLAM

The graph-based formulation of SLAM, usually called Graph-SLAM, models the SLAM problem as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The nodes $\mathbf{x}_i \in \mathcal{V}$ of the graph describe the poses of the robot, and an edge connecting two nodes $\mathbf{x}_i$ and $\mathbf{x}_j$, $(i, j) \in \mathcal{E}$, represents a measurement of the relative transformation $\mathbf{z}_{ij}$ between the two nodes, in conjunction with the information matrix Ω_{ij}, which models the uncertainty of the measurement. Basically, there are two types of measurements: those that connect two consecutive nodes, $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, are usually provided by *e.g.,* odometry or incremental scan matching, and those that represent a loop closure that associates two nodes, $\mathbf{x}_i$ and $\mathbf{x}_j$ with $|i - j| > k$, from different time steps. From the estimate of the robot poses and the measurement from the edges, we can define an error function e as

$$e\left(\mathbf{z}_{ij}, \mathbf{x}_i, \mathbf{x}_j\right) = e_{ij} = \mathbf{z}_{ij} \ominus \hat{\mathbf{z}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \mathbf{z}_{ij} \ominus \left(\mathbf{x}_i \ominus \mathbf{x}_j\right), \tag{2.1}$$

which measures the displacement of the relative transformation of the poses and the one estimated by the measurement $\mathbf{z}_{ij}$.

The probabilistic formulation of the SLAM problem aims to compute the most likely positions of the robot, given all observations $\mathcal{Z}$.

$$\left(\mathbf{x}_1^\star, \ldots, \mathbf{x}_n^\star\right) = \operatorname{argmax} P\left(\mathbf{x}_1, \ldots, \mathbf{x}_n \mid \mathcal{Z}\right) \tag{2.2}$$

This optimization problem is equivalent to minimizing the negative log-likelihood of this posterior. In a Gaussian model, this corresponds to the weighted error sum of the graph-based model above. The SLAM solution is then computed by solving the resulting non-linear least squares minimization problem

$$\left(\mathbf{x}_1^\star, \ldots, \mathbf{x}_n^\star\right) = \operatorname*{argmin}_{\mathbf{x}_1, \ldots, \mathbf{x}_n} \sum_{(i,j) \in \mathcal{E}} \left\| e_{ij} \right\|_{\Omega_{ij}}^2. \tag{2.3}$$

2.4 Monte Carlo Localization

The probabilistic formulation of localization aims to estimate the distribution of the current pose of the robot, given all of the data that has been observed so far, $P(\mathbf{x}_t \mid \mathcal{Z}_t, \mathcal{U}_t)$, with $\mathcal{Z}_t = \{\mathbf{z}_t, \ldots, \mathbf{z}_0\}$ being the set of observations and $\mathcal{U}_t = \{\mathbf{u}_{t-1}, \ldots, \mathbf{u}_0\}$ being the set of controls until time t. A common technique to recursively updating this probability is to apply a Bayesian filter, which detaches the sensor model and the transition model.

$$P(\mathbf{x}_t \mid \mathcal{Z}_t, \mathcal{U}_t) = \eta \underbrace{P(\mathbf{z}_t \mid \mathbf{x}_t)}_{\text{Sensor model}} \int \underbrace{P(\mathbf{x}_t \mid \mathbf{u}_{t-1}, \mathbf{x}_{t-1})}_{\text{Process model}} P(\mathbf{x}_{t-1} \mid \mathcal{U}_{t-1}, \mathcal{Z}_{t-1}) \mathrm{d}\mathbf{x}_{t-1} \tag{2.4}$$

Monte Carlo Localization (MCL) is a popular and commonly used probabilistic localization technique, which was introduced by Fox et al. [33]. The approach turns out to be robust and it is not bound to a specific distribution, as *e.g.,* the Kalman Filter [54]. MCL represents the distribution of the localization problem by samples. Each sample, also called particle, represents a pose hypothesis that is updated by sampling motions from the process model. From the sensor model, each particle is associated with a weight ω_i, which is proportional to the sensor model.

$$\omega_i \propto P(\mathbf{z}_t \mid \mathbf{x}_{t,i}) \tag{2.5}$$

After this step, a resampling scheme is applied that samples particles with a probability equal to the normalized weight. Particles that exhibit a large weight are likely to be sampled multiple times, and those with a low weight are likely to not be represented in the new set of particles. The state estimate is then derived from the set of samples, *e.g.,* by the mean or the mode. A drawback of MCL is that the computational complexity grows linearly with the number of particles, whereas the number of particles that is necessary for success depends mainly on the dimensionality of the state space and the size of the environment. Fox [32] proposed a technique called Kullback-Leibler divergence (KLD) sampling, which aims to compute the minimum amount of samples needed to represent the current distribution.

2.5 Generalized Iterative Closest Point Algorithm

The "Iterative Closest Point" algorithm (ICP) is a popular method for the efficient alignment of two point-sets $\mathcal{S}$ and $\mathcal{A}$ that observe the same scene from different positions. With the ICP algorithm, we seek to find the transformation that aligns the two point-sets best, which is equivalent to determining the relative transformation between the reference frame of $\mathcal{A}$ and the reference frame of $\mathcal{S}$. In order to simplify notations, we assume that the two point-sets $\hat{\mathcal{S}} = \{\hat{\mathbf{s}}_i\}$ and $\hat{\mathcal{A}} = \{\hat{\mathbf{a}}_i\}$ are sorted with respect to their correspondences and both sets are filtered for outliers. In this case, assuming perfect associations with no noise in the measurements, we can find a single affine transformation $\mathbf{T}$, which is composed of a rotaion-matrix $\mathbf{R}_\mathbf{T}$ and a translation vector $\mathbf{t}_\mathbf{T}$, such that

$$\hat{\mathbf{s}}_i = \mathbf{T}(\hat{\mathbf{a}}_i) \quad \forall 1 \leq i \leq |\hat{\mathcal{S}}| \left(= |\hat{\mathcal{A}}|\right). \tag{2.6}$$

However, in real world scenarios the data association will not be perfect, and the measurements are typically affected by noise. Therefore, the equation above will not hold, in general.

To overcome this problem, we assume that the point-sets are drawn from Gaussian probability distributions, i.e., $\hat{\mathbf{a}}_i \sim \mathcal{N}(\hat{\mathbf{a}}_i, \mathbf{\Sigma}_i^{\mathcal{A}})$ and $\hat{\mathbf{s}}_i \sim \mathcal{N}(\hat{\mathbf{s}}_i, \mathbf{\Sigma}_i^{\mathcal{S}})$. To quantify the differences between the point-sets we define the difference function $d^{\mathbf{T}}$ under the affine transformation $\mathbf{T}$ as

$$d^{\mathbf{T}}(\mathbf{a}_i, \mathbf{s}_i) = \mathbf{T}(\mathbf{a}_i) - \mathbf{s}_i. \tag{2.7}$$

The probabilistic formulation leads to a relaxation of Eq. (2.6) in the way that the distribution of the differences are again Gaussians with zero mean and appropriately transformed covariances:

$$d^{\mathbf{T}}(\hat{\mathbf{a}}_i, \hat{\mathbf{s}}_i) \sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Sigma}_i^{\mathcal{S}} + \mathbf{R_T}\mathbf{\Sigma}_i^{\mathcal{A}}{\mathbf{R_T}}^T\right) = \mathcal{N}\left(\mathbf{0}, \mathbf{\Sigma}_{i,T}\right) \tag{2.8}$$

Using this formulation we can find the optimal transformation $\mathbf{T}$ with a maximum likelihood approach, solving

$$\mathbf{T}^\star = \underset{\mathbf{T}}{\operatorname{argmax}} \prod_i p\left(d^{\mathbf{T}}(\hat{\mathbf{a}}_i, \hat{\mathbf{s}}_i)\right). \tag{2.9}$$

Transforming the optimization equation to the $\log$-space this results in

$$\mathbf{T}^\star = \underset{\mathbf{T}}{\operatorname{argmax}} \sum_i \log\left(p\left(d^{\mathbf{T}}(\hat{\mathbf{a}}_i, \hat{\mathbf{s}}_i)\right)\right) = \underset{\mathbf{T}}{\operatorname{argmax}} \sum_i \|d^{\mathbf{T}}(\hat{\mathbf{a}}_i, \hat{\mathbf{s}}_i)\|^2_{\mathbf{\Sigma}_{i,T}}. \tag{2.10}$$

Different assumptions on the distribution of the points $\mathbf{a}_i$ and $\mathbf{s}_i$ result in different distance models that are optimized. With $\mathbf{\Sigma}_i^{\mathcal{S}} = \mathbf{I}$ and $\mathbf{\Sigma}_i^{\mathcal{A}} = 0$ this formulation is equivalent to the original ICP approach by Besl and McKay [12]. For more details about other distances, e.g., point-to-point and point-to-plane, we kindly refer the reader to Segal et al. [104].

So far, we have assumed that the associations between the data in the two point-sets are given, which is usually not the case for real world data. As indicated by its name, a data point $\mathbf{a}_i$ is associated to the point $\mathbf{s} \in \mathcal{S}$ that has the minimal distance, based on the current transformation. More specifically, the solution of the following minimization:

$$\mathbf{s}_i^\star = \underset{\mathbf{s}\in\mathcal{S}}{\operatorname{argmin}} \|d^{\mathbf{T}}(\mathbf{a}_i, \mathbf{s})\|_2 \tag{2.11}$$

The ICP-algorithm iterates the assignment step of Eq. (2.11) followed by solving the optimization problem stated in Eq. (2.10). To initialize the transformation for the very first assignment, an initial guess needs to be given to the algorithm. In robotics, this is usually derived from odometry measurements or if no further information is available then it is initialized with the identity. Afterwards, the latest result of the optimization is used for the association step. This process is iterated until convergence or a maximum number of iterations is reached, since in general, there is no guarantee for the convergence of the algorithm.

2.6 Classification

Classification frameworks are typically employed to extract semantic information from sensor data. A common model is that the data is represented as a real valued vector $\mathbf{x} \in \mathbb{R}^d$, and a classifier F is a function that maps the data to semantic classes, which are commonly represented as a set of integers.

$$F : \mathbb{R}^d \longrightarrow \mathcal{C} \subseteq \mathbb{Z} \tag{2.12}$$

In general, one can not design a classifier by hand, due to the complexity of the values encoded in the feature data x. One can, however, employ machine learning techniques in order to learn a classifier from input data. Thereby, for a specific problem, the space of the possible classifiers is usually restricted to a subset $\mathcal{F}^1 \subseteq \mathcal{F}$, where $\mathcal{F}$ is the set of all possible functions as specified in the map (2.12). The classifier is trained on a training set $\mathcal{T}$, which consists of features along with their desired semantic label.

$$\mathcal{T} = \{(\mathbf{x}, y) \mid \mathbf{x} \in \mathbb{R}^d, y \in \mathbb{Z}\} \subseteq \mathbb{R}^d \times \mathbb{Z} \tag{2.13}$$

Learning the classifier is then usually formulated as an optimization problem:

$$F^\star = \underset{F \in \mathcal{F}^1 \subseteq \mathcal{F}}{\operatorname{argmin}} \sum_{(\mathbf{x},y)\in\mathcal{T}} L(F(\mathbf{x}), y) \tag{2.14}$$

with the loss function $L : \mathbb{R}^d \times \mathbb{Z} \to \mathbb{R}_{\geq 0}$ quantifying the penalty for wrong results.

Common classifiers are, *e.g.,* Support Vector Machine, Neural Networks, k-Nearest Neighbors, k-Means and Random Forests. In this thesis, we will use the Random Forest classifier, which we will, therefore, briefly introduce in forthcoming section.

2.6.1 Random Forest Classifier

The random forest classifier, as introduced by Breiman [16], is formed by an ensemble of decision trees. A decision tree is a binary tree that routes a data point $\mathbf{x} \in \mathbb{R}^d$ based on the value in a certain dimension and a threshold, both of which are stored in the corresponding node. We briefly explain how such a tree is trained.

The root node contains the full training set $\mathcal{T}$. Based on information theoretic measures, *e.g.,* gini-impurity or information gain, it exhaustively searches for the combination of split dimension and threshold that provides the most improvement of this measure for the resulting split. The data of the parent node is assigned to the child nodes, based on this split, and the process iterates until a break-condition is fulfilled, *e.g.,* a node contains only samples with the same label, a minimum amount of samples or a certain depth is reached. To classify a data point $\mathbf{x} \in \mathcal{R}^d$, the decision tree returns the label of the resulting leaf node. This process is fully deterministic and the trees tend to overfit the data when they are grown too deep.

The RF-classifier improves this behavior by randomizing the construction process. Instead of a single tree, it trains N trees and the classification result is provided by the accumulated results of each of the individual trees, *e.g.,* by majority vote or as probabilities. Each tree is trained on a training set $\mathcal{T}_i$ that is sampled, with replacement, from the original training set $\mathcal{T}$. This process is called bootstrapping or bagging, and as a consequence, each tree is trained on a slightly different training set. Second, instead of using deterministic decision trees, the RF-classifier trains randomized decision trees, which use random dimensions to determine the split instead of taking all dimensions into account. A nice characteristic of RF-classifiers is that it is proven by the strong law of large numbers that they do not overfit as the number of trees is increased [16].

In this thesis, we use the RF-classifier with the gini-impurity-based measure to construct the decision trees. Given the frequencies of the classes $p_1, \ldots, p_n$ for the data of a node, the gini-impurity is computed as

$$G(p_1, \ldots, p_n) = \sum_i p_i(1 - p_i) = \sum_{i \neq j} p_i p_j \tag{2.15}$$

Chapter 3

Memory Aware Considerations of SLAM

Building a map of an unknown environment and localizing within this map at the same time is known as the Simultaneous Localization and Mapping (SLAM) problem, and constitutes one of the fundamental problems in mobile robotics. In the past, solutions that are based on solving a non-linear optimization problem, which are highly effective due to an underlying sparse graph structure, have gained popularity, and are widely used. However, most approaches put their major focus on runtime and accuracy, rather than on memory consumption, which becomes especially relevant when large-scale SLAM problems have to be solved. In this chapter, we consider the SLAM problem with respect to memory consumption, and present a novel approximate approach to SLAM with a low memory footprint. The proposed approach achieves this, based on a hierarchical decomposition that consists of small sub-maps of limited size. We perform extensive experiments on synthetic and publicly available datasets to elaborate on the advantages of our approach. The results demonstrate that in situations in which the optimization of the complete map requires more than the available main memory, our approach, in comparison to state-of-the-art exact solvers, reduces the memory consumption and the runtime up to a factor of two, while still providing highly accurate maps.

Map building in SLAM is one of the fundamental problems in mobile robotics, as being able to learn what the environment looks like is typically regarded as a key prerequisite for truly autonomous systems. In the past, several techniques to solve the SLAM problem have been developed. One of those techniques, models the SLAM problem as a graph, where the poses of the robot are the nodes and measurements between different poses are encoded in the edges of the graph. The problem of SLAM is then reformulated as a nonlinear-least squares optimization problem. For more details on this technique see Chapter 2.3.1. Within this chapter, we refer to the Graph-SLAM formulation whenever we talk about SLAM. So far, highly effective SLAM methods have been developed and state-of-the-art SLAM solvers are able to achieve accurate solutions (meaning close to the optimum), in a minimum amount of time [40, 52, 92]. In this chapter, we will focus on memory consumption and seek for a SLAM solver that can produce fast and accurate solutions, while being effective with respect to memory consumption. This aspect is particularly relevant when one has to solve a large-scale SLAM problem on a memory-restricted system, at which the interpretation of large-scale may

be relative to the available memory of the system. In the case of large mapping problems, an algorithm that is not designed to be memory efficient will eventually try to allocate more than the available main memory on the computing unit. This typically triggers paging mechanisms of the operating system, during which parts of the memory are stored to or retrieved from the hard disk, thus largely slowing down the execution. It is worth to note that the very fast Solid-State-Disks (SSDs) are not appropriate to use for the paging mechanism due to the limited write cycles during their lifetime. We are convinced that the memory efficiency is highly relevant for the development of low-cost robotic systems, where hardware resources are often extremely limited to be competitive on the consumer market.

Due to the robustness of approaches to robot navigation and SLAM, the range of autonomous navigation for robots is rapidly increasing. City-scale autonomous navigation [58] is already possible, and autonomous cars have traveled hundreds of kilometers through the desert [121] and navigated for hours in city-like traffic environments [125]. Companies have started to test autonomous cars in cities and on highways, *e.g.,* Google, Tesla, Uber, Ford and Daimler, and it is somehow an open secret that they all rely on highly accurate maps of the environment in which they drive autonomously. Therefore, several modern techniques address the problem of learning large-scale maps, required by such applications [15, 30, 41, 88]. However, these approaches mostly concentrate on accuracy and runtime whilst memory consumption has not been their major focus.

In the remainder of this chapter, we present an algorithm that is able to solve large mapping problems with low memory consumption. Our method employs a *divide-and-conquer* principle to hierarchically subdivide the problem into many submaps of small size with limited dependencies, see Section (3.1.1), and to solve a fine-to-coarse, see Section (3.1.2), followed by a coarse-to-fine least-squares map optimization, see Section (3.1.3). At each level of the hierarchy, we subdivide the graph into subgraphs. For a schematic diagram see Figure (3.1). We optimize each subgraph independently of the others and approximate it coarsely (fine-to-coarse). All the approximated subgraphs, along with the edges that connect them, constitute the graph at the next level. We iterate this process until we reach a desired top level. Then, we carry out coarse-to-fine map adjustments by traversing the hierarchy in a top-down manner and performing an optimization at each level. Our algorithm does not require a specific graph clustering technique. Rather, every technique that is able to limit the number of nodes per cluster constitutes a valid choice. We present extensive experiments, see Section (3.3), in which we evaluate metric accuracy, memory footprint and computation time. The results are obtained by running the methods on publicly available datasets and demonstrate that our approach yields a precision that compares to that of other state-of-the-art methods. In addition, we perform evaluations on large-scale datasets consisting of hundreds of thousands of nodes and demonstrate that our method exhibits lower memory consumption than the state-of-the-art implementations. For memory-constrained systems, for which the entire data set does not fit into main memory, our approach is able to solve problems two times faster.

3.1 Mapping with Low Memory Consumption

In this chapter, we consider the SLAM problem in its probabilistic graph-based interpretation. Let $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_n]^T$ be the vector of robot poses, where $\mathbf{x}_i$ is the pose of the robot at time i. Let $\mathbf{z}_{ij}$ and $\mathbf{\Omega}_{ij}$ be the mean and information matrix of a measurement between pose $\mathbf{x}_i$ and

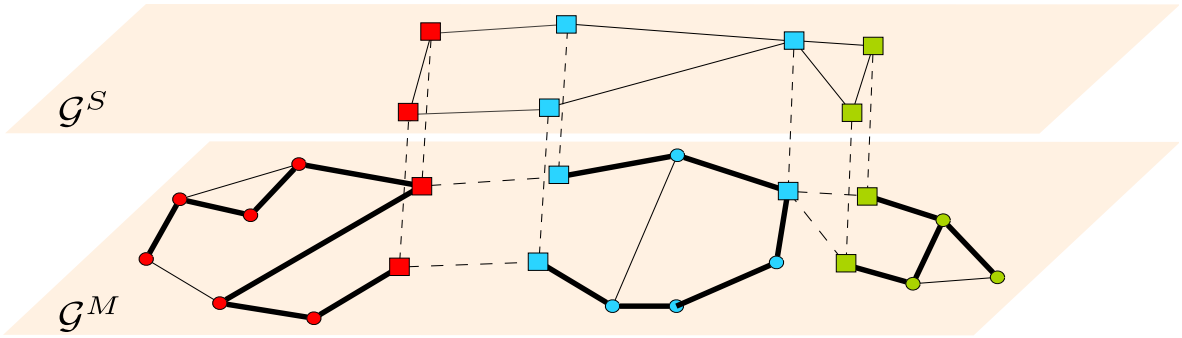


Figure 3.1: The proposed small memory footprint approach is based on a hierarchical graph partition. In each partition $\mathcal{G}_i^M$ (same color) we identify separator nodes $\mathcal{V}^S$ (squares) and interior nodes $\mathcal{V}^M$ (circles). This implicitly partitions the edges into separator edges $\mathcal{E}^S$ (dashed lines) and interior edges $\mathcal{E}^M$ (solid lines). We build the graph of the separator nodes in $\mathcal{G}^S$ by using a tree approximation on each subgraph (thick lines). We optimize on each layer and on disjoint subgraphs, from bottom to top and vice versa. With our algorithm, we never optimize the entire graph as a whole.

pose $\mathbf{x}_j$. This measurement can arise from odometry or be the resulting estimate of a local matching algorithm. Without loss of generality, we only consider pose-to-pose constraints in this chapter. For more general constraints, virtual measurements as proposed by Grisetti et al. [42] can be used.

Finding a solution to the SLAM problem is then equivalent to minimizing the negative log-likelihood of the joint distribution

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \left(-\log \left(p(\mathbf{x} \mid \mathcal{Z}) \right) \right) \tag{3.1}$$

$$= \operatorname*{argmin}_{\mathbf{x}} \left(-\log \prod_{\mathbf{z}_{ij} \in \mathcal{Z}} \phi_{\mathbf{z}_{ij}}(\mathbf{x}_i, \mathbf{x}_j) \right) \tag{3.2}$$

$$= \operatorname*{argmin}_{\mathbf{x}} \sum_{\mathbf{z}_{ij} \in \mathcal{Z}} \frac{1}{2} \|\hat{\mathbf{z}}(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{z}_{ij}\|^2_{\mathbf{\Omega}_{ij}} \tag{3.3}$$

where $\hat{\mathbf{z}}(\mathbf{x}_i, \mathbf{x}_j)$ is the prediction of a measurement given a configuration of $\mathbf{x}_i$ and $\mathbf{x}_j$, the function $\phi(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}\|\hat{\mathbf{z}}(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{z}_{ij}\|^2_{\mathbf{\Omega}_{ij}}\right)$ is the pairwise compatibility function, and $\mathcal{Z}$ is the set of all measurements. Our idea is to address the problem of mapping with low memory consumption by building a hierarchical data structure with a decreasing amount of detail such that, at each level, inference is always performed on subgraphs of bounded size. Our method applies the following inference procedure: First, a bottom-up inference process propagates information from the lower levels to the upper one (similar in spirit to Gaussian elimination). Second, a top-down inference procedure solves the top-most system and propagates the resulting information down to the lower levels (similar in spirit to back-substitution).

In the remainder of the section and for the sake of simplicity, we restrict the description of the approach to a two-level hierarchy. We can deal with multi-level hierarchies by iteratively applying this approach.

3.1.1 Graph Partitioning

Let the whole pose graph be $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ denotes the set of nodes and $\mathcal{E} = \{(i,j) \mid \mathbf{z}_{ij} \in \mathcal{Z}\}$ denotes the set of edges. Two nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ are adjacent if and only if there exists a measurement $\mathbf{z}_{ij} \in \mathcal{Z}$ or $\mathbf{z}_{ji} \in \mathcal{Z}$ between the two poses. We partition the set of nodes $\mathcal{V}$ into m pairwise disjoint subsets $\mathcal{V}^I = \{\mathcal{V}_1^I, \ldots, \mathcal{V}_m^I\}$ by an edge-cut, such that $\mathcal{V}_i^I \cap \mathcal{V}_j^I = \emptyset$ for $1 \leq i < j \leq m$ and $\mathcal{V} = \bigcup_i \mathcal{V}_i^I$. This directly induces a partitioning on the edge set into two disjoint subsets: $\mathcal{E}^S$ and $\mathcal{E}^M$, with $\mathcal{E}^M = \mathcal{E}_1^M \cup \cdots \cup \mathcal{E}_m^M$ and $\mathcal{E}^S \cup \mathcal{E}^M = \mathcal{E}$. The set $\mathcal{E}_i^M$ contains edges which connect only the nodes in V_i^I and $\mathcal{E}^S$ is the edge-cut set that connects nodes from two different subgraphs. Each $\mathcal{V}_i^I$ is then subdivided into the set of boundary nodes $\mathcal{V}_i^S$ and a set of interior nodes $\mathcal{V}_i^M$, where $\mathcal{V}_i^I = \mathcal{V}_i^M \cup \mathcal{V}_i^S$, $\mathcal{V}_i^S$ are those nodes of $\mathcal{V}_i^I$ which are incident with at least one edge from $\mathcal{E}^S$, and $\mathcal{V}_i^M \cap \mathcal{V}_i^S = \emptyset$.

Let $\mathcal{X}_n^M = \bigcup_{j \in \mathcal{V}_n^M} \{\mathbf{x}_j\}$ and $\mathcal{X}_n^S = \bigcup_{j \in \mathcal{V}_n^S} \{\mathbf{x}_j\}$. We can decompose the distribution $p(\mathbf{x} \mid \mathcal{Z})$ in Eq. (3.3) according to:

$$
\begin{aligned}
p(\mathbf{x} \mid \mathcal{Z}) &= p(\mathcal{X}^M, \mathcal{X}^S \mid \mathcal{Z}) && (3.4)\\
&= p(\mathcal{X}^M \mid \mathcal{X}^S, \mathcal{Z}) p(\mathcal{X}^S \mid \mathcal{Z}) && (3.5)\\
&\overset{Markov}{=} \prod_{n=1}^{m} p(\mathcal{X}_n^M \mid \mathcal{X}_n^S, \mathcal{Z}_n^M) p(\mathcal{X}^S \mid \mathcal{Z}), && (3.6)
\end{aligned}
$$

where $\mathcal{X}^M = \bigcup_n \mathcal{X}_n^M$, $\mathcal{X}^S = \bigcup_n \mathcal{X}_n^S$, $\mathcal{Z}_n^M = \{\mathbf{z}_{ij} \mid \{i,j\} \in \mathcal{E}_n^M\}$ and the rightmost part stems from the global Markov property.

Eq. (3.6) defines the building blocks of our hierarchy. The bottom level of the hierarchy consists of the m disjoint subgraphs $\mathcal{G}_n^M$ induced from the distributions $p(\mathcal{X}_n^M \mid \mathcal{X}_n^S, \mathcal{Z}_n^M)$. The top level is the graph of the separators $\mathcal{G}^S$ and is induced by the distribution $p(\mathcal{X}^S \mid \mathcal{Z})$. Figure 3.1 shows an example of a two-level hierarchy. Multiple levels are then iteratively built, considering the previously computed separator graph $\mathcal{G}^S$ as the input graph $\mathcal{G}$ and performing the decomposition on it.

To design a low-memory-consumption algorithm, we require that the size of all the partitions, on every level, is small enough to fit into memory. In principle, one can use any partitioning algorithm with this property. Potential candidates are METIS [55] or Nibble [109]. In our current implementation, we have opted for Nibble because it is a local algorithm and it is able to generate graph partitions in an incremental fashion. Accordingly, it does not need to store the whole graph in memory.

3.1.2 Leaves-to-Root Coarsening

The purpose of the leaves-to-root inference is to compute, at each level of the hierarchy, the marginal distribution $p(\mathcal{X}^S \mid \mathcal{Z})$ of the separator graph $\mathcal{G}^S$. Therefore, we exploit the pairwise

nature of the SLAM graphical model, as already been done in Eq. (3.3).

$$p\left(\mathcal{X}^S \mid \mathcal{Z}\right) = \int p\left(\mathcal{X}^M, \mathcal{X}^S \mid \mathcal{Z}\right) d\mathcal{X}^M \tag{3.7}$$

$$\underset{\text{Eq. (3.2)}}{\propto} \int \prod_{\mathbf{z}_{ij} \in \mathcal{Z}} \phi_{\mathbf{z}_{ij}}\left(\mathbf{x}_i, \mathbf{x}_j\right) d\mathcal{X}^M \tag{3.8}$$

$$= \prod_{\mathbf{z}_{ij} \in \mathcal{Z}^S} \phi_{\mathbf{z}_{ij}}(\mathbf{x}_i, \mathbf{x}_j) \prod_{n=1}^{m} \int \prod_{\mathbf{z}_{uv} \in \mathcal{Z}_n^M} \phi_{\mathbf{z}_{uv}}(\mathbf{x}_u, \mathbf{x}_v) d\mathcal{X}_n^M. \tag{3.9}$$

This decomposition tells us that the marginal distribution of the separator nodes is composed of the factors coming from the separator edges $\mathcal{E}^S$, connecting the respective boundary nodes of two different subgraphs, and the factors computed by marginalizing out the inner nodes of each individual subgraph, connecting the boundary nodes of the same subgraph. However, the process of marginalization will destroy the original sparseness of the pose-graph, leading to high computational costs and memory requirements. This is not in line with the target of our approach, since we want to be explicitly memory efficient. To overcome the problem of the fully connected marginalization graph, we propose to approximate the marginal distribution of the boundary nodes with a tree-structured distribution that preserves the marginal mean.

Chow and Liu [21] showed that the tree-structured distribution that minimizes the Kullback-Leibler divergence can be obtained by computing the maximum spanning tree on the mutual information graph. Although their proof considers discrete distributions, the result also holds for continuous densities. Unfortunately, computing the mutual information between boundary nodes involves inverting the information matrix relative to the entire subgraph, resulting in a computationally expensive operation.

We instead build a maximum spanning tree of the measurement information, where the graph structure is the same as $\mathcal{G}_n^M$ and the edges are weighted according to $\det(\mathbf{\Omega}_{ij})$.

We build the approximate separator graph incrementally. First, we optimize the subgraph without considering the separator edges. Second, we compute the maximum spanning tree $\mathcal{T}$ with one of the separators as root node and the path $\mathcal{P}^{ij}$ on the tree connecting the nodes i and j. By performing a depth-first visit on the tree, we select the edges of the graph such that any separator is connected to its closest parent in $\mathcal{T}$, resulting in the edge set $\mathcal{E}^T$. For each edge $(i, j) \in \mathcal{E}^T$, we compute a virtual measurement

$$\mathbf{z}_{ij}^* = \bigoplus_{k \in \mathcal{P}^{ij}} \hat{\mathbf{z}}_{k,k+1} \tag{3.10}$$

$$\mathbf{\Omega}_{ij}^{*-1} = \mathbf{J}^T \mathbf{\Omega}_{\mathcal{P}^{ij}}^{-1} \mathbf{J}, \tag{3.11}$$

where $\mathbf{J}$ is the Jacobian of Eq. (3.10), $\hat{\mathbf{z}}_{k,k+1} = \hat{\mathbf{x}}_k \ominus \hat{\mathbf{x}}_{k+1}$ is the transformation between node k and $k+1$ in the path, after optimization, and $\mathbf{\Omega}_{\mathcal{P}^{ij}}$ is a block diagonal matrix, whose diagonal elements correspond to the information matrix associated to the edges on the path. In practice, we compute the covariance matrix $\mathbf{\Omega}_{ij}^{*-1}$ by propagating the covariance matrices of the measurements along the compounding path [108]. Figure 3.2 shows an example construction of the separator graph for a single subgraph.

This results in a new graph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ with $\mathcal{V}^* = \mathcal{X}^S$ and $\mathcal{E}^* = \underset{n}{\cup} \mathcal{E}_n^{\mathcal{T}} \cup \mathcal{E}^S$ which constitutes the graph of the next level higher, in our hierarchical construction. We iterate this

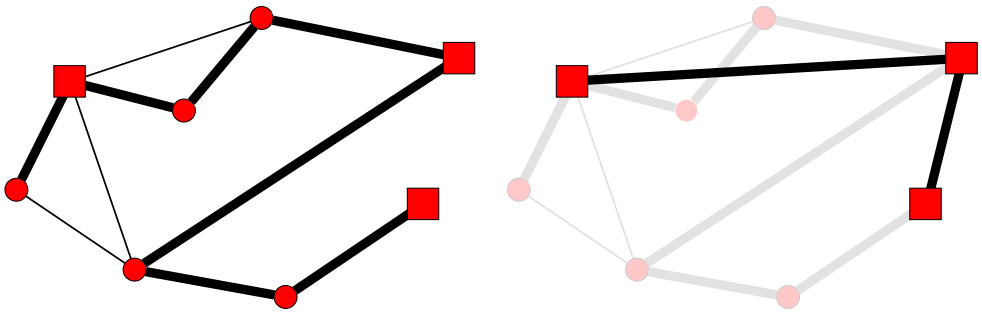


Figure 3.2: Example construction of the approximate separator graph for a single partition. The left image shows the original graph, where the separators are displayed as squares and thicker lines depict the maximum spanning tree. The right image shows the resulting separator graph obtained by performing the depth-first visit.

process of fine-to-coarse optimization until we reach the top level of the hierarchy. In principle, any non-linear least-squares optimizer can be used for the optimization of the subgraphs, however, in our current implementation we use g^{2}o [57] for that purpose.

3.1.3 Root-to-Leaves Optimization

Once the hierarchy has been built up to the top level L, we compute a solution for the top-most graph $\mathcal{G}_L = (\mathcal{V}_L, \mathcal{E}_L)$ by solving the corresponding least squares problem. By construction of the hierarchy, as explained in Section (3.1.2), each new level has fewer nodes and fewer edges than the current level. This is because the number of separator nodes is smaller than the original number of nodes and the edges are the union of existing edges, and the minimal amount of nodes to connect the separator nodes of the individual subgraphs. Therefore, we have $|\mathcal{V}_L| << |\mathcal{V}_0|$ and $|\mathcal{E}_L| << |\mathcal{E}_0|$ so that we can solve the top-level graph using substantially less memory than for the full graph.

Let us recall Eq. (3.6), where we factorized the target distribution for the individual subgraphs and the marginal distribution of the separator nodes. As derived from the previous section, we approximate the marginal distribution by the graph $\mathcal{G}_L$, which we have just optimized. To obtain the final map, we need to propagate the solution obtained to the leaves of the hierarchy. In the case of linear systems, this propagation is equivalent to back substitution. To overcome the non-linearities of SLAM, we propose to perform an additional optimization step on each subgraph, by fixing the separator variables and only minimizing with respect to the inner nodes. This step corresponds to N independent minimization problems of limited size, one for each subgraph $\mathcal{G}^M_{L-1,n}$. The process is then iterated for each level in the hierarchy.

Note, that the least square minimization algorithm is always applied to bounded problems. The partitions are bounded by construction, due to the partitioning algorithm. The separator graph is also bounded by construction. The number of separator nodes S is smaller than the number of original nodes, since the separators are a subset of the nodes of the original graph. The number of edges is also bounded, since we only consider the edges between separators from the original graph and the edges connecting the separators of the individual partitions, which are at most $S - 1$.

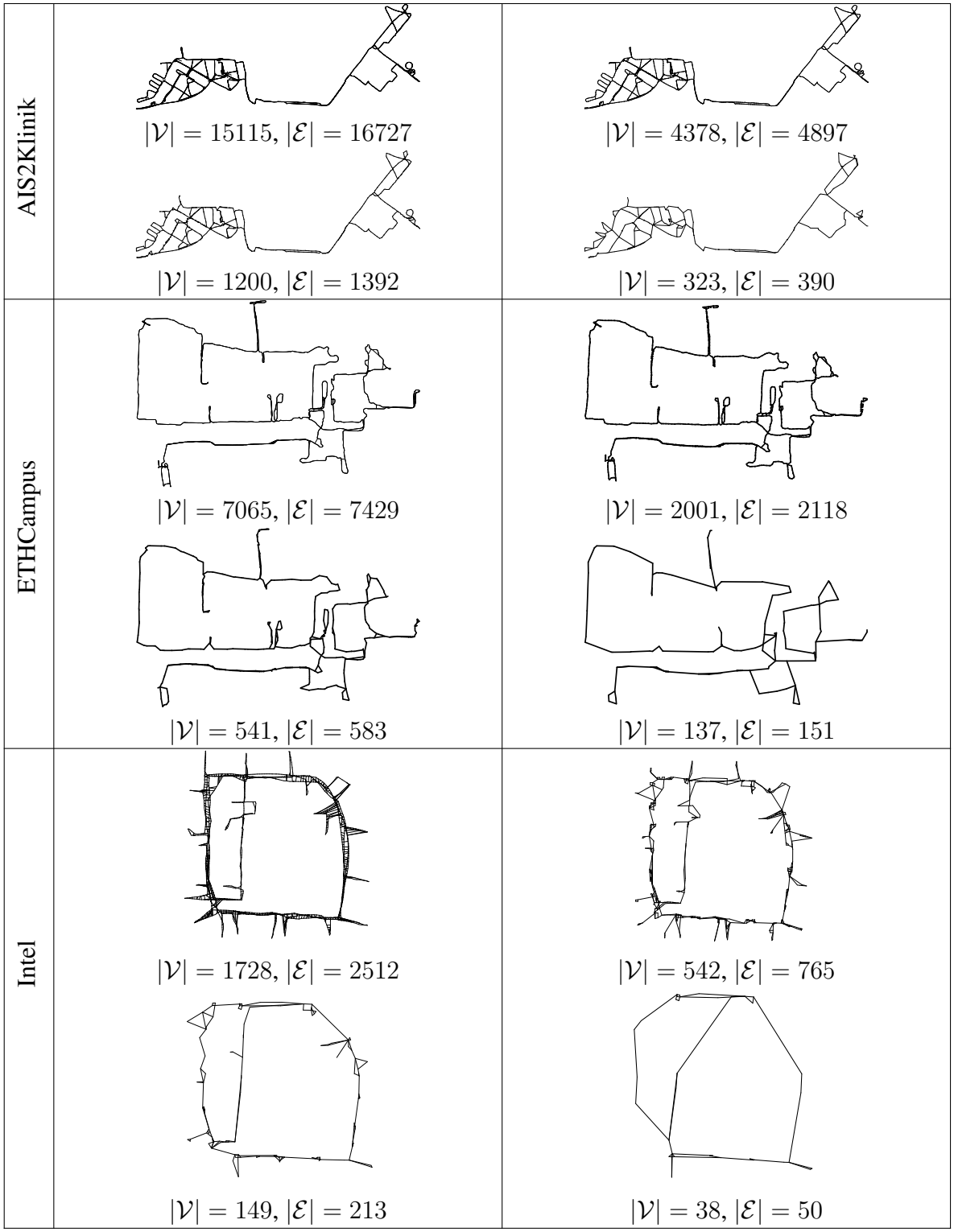


Figure 3.3: Hierarchies computed by SMF for different datasets. The picture shows the separator graph at different levels, with the number of nodes and edges.

3.2 Memory Consumption Analysis

In this section, we look briefly into the memory consumption of our approach and full SLAM solvers. Since the exact amount of memory used by an approach also highly depends on the implementation as well as on the specific instance of the problem, we will establish approximate estimates to get an idea of the magnitude of the memory consumption.

The graph that models the SLAM problem is typically sparse, which means that the average number of edges that intersect with a node is bounded, $|\mathcal{E}| \leq c|\mathcal{V}|$ for some $c > 0$. By our sparse construction, this holds for each level of hierarchy as well. First, let us analyze how a classic solution is computed. Solving the least squares minimization problem, incorporates the calculation of a solution to a linear equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ for a sparse $d|\mathcal{V}| \times d|\mathcal{V}|$ matrix $\mathbf{A}$, where d is the dimension of the pose representation, for which the number of edges determine the number of non-zero entries. With a good permutation of the columns of $\mathbf{A}$, the fill-in, which arises by solving the linear system, can be minimized and the number of non-zero values is in the same range as the number of non-zero values of $\mathbf{A}$. Even though finding the optimal permutation is known to be NP-complete [137] good approximations to the problem exist [4, 25, 37]. To sum up, the full solver needs to store the graph, the matrix $\mathbf{A}$ and the decomposition, in order to solve the linear system. This results in a memory consumption in the magnitude of at least $(3c + 1)|\mathcal{V}|$.

In contrast, our method needs to solve equations of bounded sizes only, and therefore our memory consumption on the fill-in is independent of the size of the graph. However, our current implementation needs to store the hierarchy, which we will briefly analyze. We assume that each level of our hierarchy satisfies the same sparseness inequality as above. Let us further consider that in our hierarchical construction, the number of nodes are at most, half of the prior level, $|\mathcal{V}_L| \leq \frac{|\mathcal{V}_{L-1}|}{2}$. The number of nodes in our hierarchical construction can be bounded as follows.

$$\sum_{i=0,\dots,K} |\mathcal{V}_i| \leq \sum_i \frac{|\mathcal{V}_0|}{2^i} \leq 2|\mathcal{V}_0| \tag{3.12}$$

Which leads to a memory consumption of our approach in the magnitude of $(2c + 2)|\mathcal{V}|$. These considerations show that our approach would need at most $2/3$ of the amount of memory a regular graph solver needs. The main advantage of our approach is that it never optimizes the graph as a whole, which leads to a tight bound of the dimensionality of the linear equation to solve.

In the forthcoming experimental section, we compare the memory consumption of different solvers. The results confirm the derived bound; in fact, our approach consumes only about half of the memory as regular solvers, in some problem instances.

3.3 Experiments

We evaluate our approach (SMF) with respect to memory consumption, runtime on low memory systems and metric accuracy, and compare it to other state-of-the art mapping approaches,

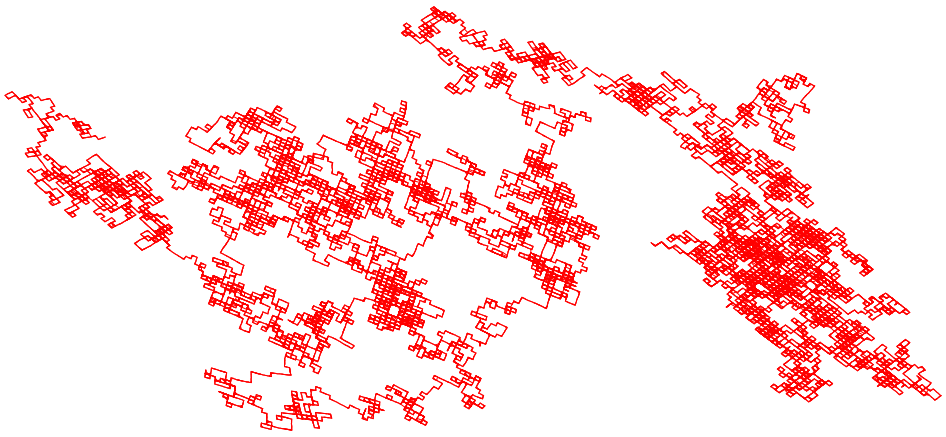


Figure 3.4: One example Manhattan world dataset used in the memory consumption experiment.

namely TSAM2[1] [88], GTSAM[2] [53], g^{2}o[3] [57]. To investigate the properties of our approximation, we compare our method to two other non-exact solvers, which are HogMan[4] [41] and a variant of TSAM2 that does not perform subgraph relaxation (T2-NOREL). Throughout the experiments, we limited the maximum submap size to 500 nodes. We ran all approaches on a single thread and without any parallelization, using Ubuntu 12.04 on an i7-2700K processor running at 3.5GHz. Note that HogMan runs incrementally, while all others are batch solvers.

3.3.1 Memory Consumption

In the first experiment, we evaluated the memory footprint on large-scale and synthetically generated datasets of Manhattan-like worlds, see Figure (3.4). The corresponding maps have graphs whose number of nodes ranges between 20,000 and 500,000 and whose connectivities lead to up to two millions edges. To investigate the memory consumption, we employed `valgrind` and `massif` in a native Linux environment. Both are open source tools for process memory analysis.

Figure 3.5 shows the results of the experiment. The graph shows two important aspects. First, our approach has the lowest memory consumption: it is consistently almost one order of magnitude more memory efficient than TSAM2, up to 2 times more than g^{2}o and GTSAM, and up to 1.3 times more than T2-NOREL and HogMan. Even though SMF, HogMan and T2-NOREL have a similar memory footprint, SMF is substantially more accurate, as shown in Section 3.3.3. Second, the approximate methods HogMan, SMF, and T2-NOREL) have lower memory consumption than the exact methods TSAM2, g^{2}o, and GTSAM.

3.3.2 Runtime on Systems with Restricted Main Memory

This experiment evaluates the computation time on systems with restricted main memory. For simulating these systems, we limited the available memory for the Linux kernel. We evaluated

[1] We thank the authors for providing their own implementation
[2] ver. 2.3.1 – https://collab.cc.gatech.edu/borg/gtsam/
[3] ver. git455 – https://github.com/RainerKuemmerle/g2o
[4] ver. svn20 – http://openslam.org/hog-man.html

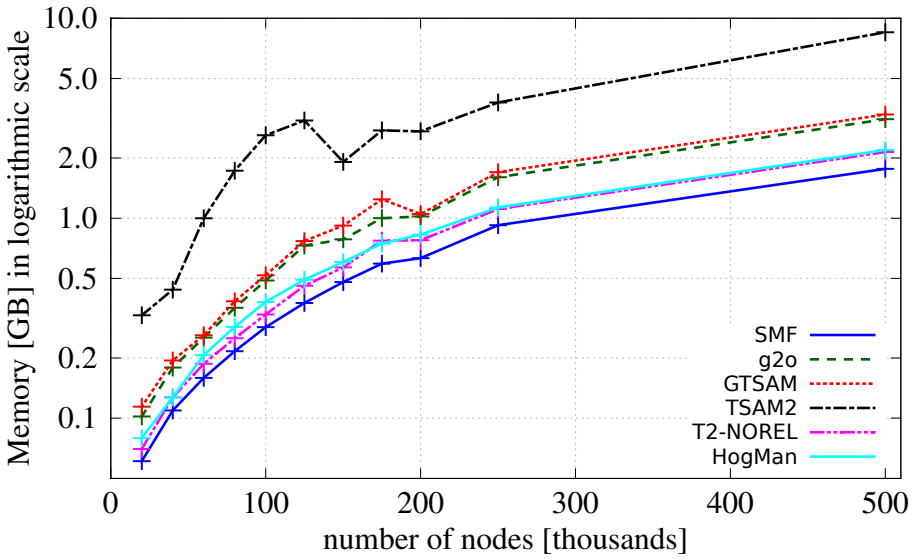


Figure 3.5: Memory consumption of modern SLAM solvers compared to our approach on large-scale graphs. Please note that the y-axis has logarithmic scale.

three budgeted memory settings (0.5GB, 0.75GB and 1.0GB) on 15 synthetic datasets with 200,000 nodes and varying connectivity. The mean number of edges was 640,000 with a standard deviation of 78,800.

Figure 3.6 shows the average runtime of all the batch solvers (g^{2}o, GTSAM, TSAM2, T2-NOREL and SMF). SMF is the fastest method at the lowest memory setting and comparable to T2-NOREL at increasing memory setups. All the other methods are significantly slower. In the lowest memory setup, TSAM2 was frequently killed by the kernel and was successful only 3 out of 15 trials. In the same setting, GTSAM never terminated.

To evaluate the statistical significance of the results, we performed a paired sample t-test with a significance level $\alpha = 0.05$. The test shows that SMF is 2 times faster than g^{2}o on 0.5GB, 1.6 times on 0.75GB, and 1.7 times on 1GB. With respect to TSAM2, SMF is 8 times faster on 0.75GB and 10 times faster on 1GB, where on 0.5GB no significant result can be given due to the limited amount of successful trials. SMF is also 7.2 times faster than GTSAM on 0.75GB and 3.7 times faster on 1GB. The timing performance of T2-NOREL is very similar to SMF, with SMF being 1.4 times faster on 0.5GB settings. Note, however, that SMF is substantially more accurate than T2-NOREL as shown in the next section.

3.3.3 Metric Accuracy

In these experiments, we quantified the metric precision of all the SLAM solvers and the time required to provide a solution without constraining the amount of available memory. Here, we run the solvers on several publicly available datasets: AIS2Klinik (15,115 nodes, 16,727 edges), ETHCampus (7,065 nodes, 7,429 edges), Intel (1,728 nodes, 2,512 edges) and Manhattan3500 (3,500 nodes, 5,542 edges). These datasets have been selected because they are good representatives of different environments (indoor and outdoor), simulated and real-data.

Table 3.1 summarizes the results with respect to the χ^2 error and runtime. Bold numbers

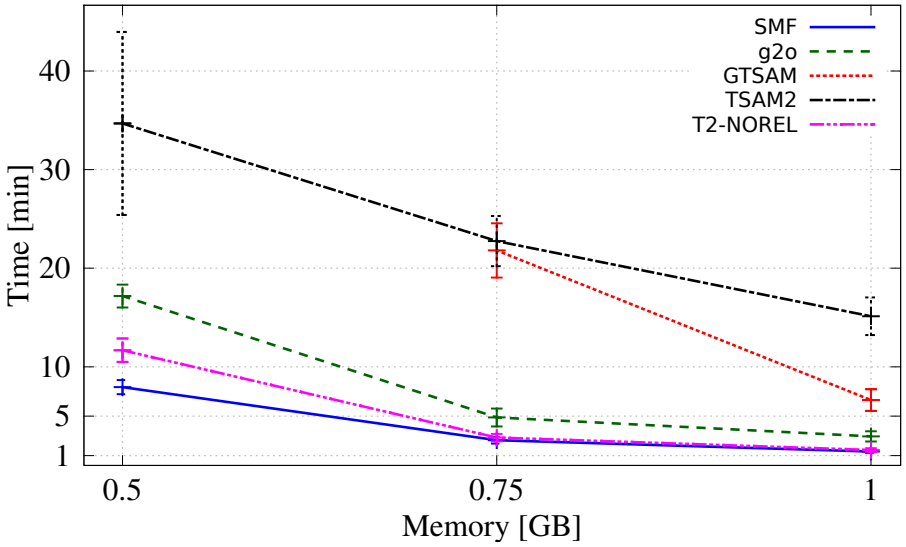


Figure 3.6: Runtime comparison with memory constraints.

indicate the best result among the approximate methods. Out of the approximate solvers, SMF has the lowest error and the lowest runtime in all the datasets but Intel and Manhattan3500, where T2-NOREL is slightly faster. SMF is more accurate than T2-NOREL: from 500 times on the ETHCampus dataset to 1.3 times on the Intel one. Compared to HogMan, SMF is up to 2 times more accurate and up to 10 time faster. However, the runtime comparison in this case is not meaningful as HogMan is an iterative solver that executes an optimization after each measurement. With respect to the exact solvers, SMF achieves comparable accuracy in all the datasets, being slightly slower than GTSAM and g^{2}o and slightly faster than TSAM2.

In order to precisely assess the quality of map reconstruction, we have also computed the reprojection error (RPE) between every edge of the optimized graph and the ground truth map computed using an exact solver – g^{2}o in our case.

Table 3.2 summarizes the results of the evaluation, showing the RPE mean, standard deviation and maximum error. SMF is more accurate than T2-NOREL and HogMan in all datasets. This is particularly evident on the two large outdoor datasets AIS2Klinik and ETHCampus where SMF is 3 times more accurate than T2-NOREL with respect to the mean error and more than an order of magnitude with respect to the maximum error. SMF is also up to 3 times more accurate than HogMan for the mean error and up to 4 times with respect to the maximum error. From a robot navigation standpoint, maximum-errors are indicators of how far some parts of the map are misaligned. Large values of this error may render the map unusable because, e.g., paths could not be computed: this happens with T2-NOREL in datasets AIS2Klinik ($\geq 10m$) and ETHCampus ($\geq 7m$). In those datasets, SMF instead achieves a maximum-error of $0.4m$ and $0.08m$.

Table 3.1: Small Memory Footprint Mapping (χ^2, Time)

Dataset	**Method**	χ^2 **error**	**time** [s]
AIS2Klinik	SMF	**471.0**	**0.86**
	T2-NOREL	108,977.8	1.00
	HogMan	647.0	15.53
	TSAM2	172.8	2.85
	GTSAM	172.8	1.00
	g^{2}o	172.8	0.21
ETHCampus	SMF	**38.9**	**0.36**
	T2-NOREL	22,457.2	0.50
	HogMan	79.3	2.55
	TSAM2	25.0	1.15
	GTSAM	25.0	0.36
	g^{2}o	25.0	0.06
Intel	SMF	**53.3**	0.11
	T2-NOREL	69.0	**0.08**
	HogMan	134.7	0.78
	TSAM2	45.0	0.18
	GTSAM	45.0	0.06
	g^{2}o	45.0	0.02
Manhattan3500	SMF	**287.1**	0.35
	T2-NOREL	733.8	**0.21**
	HogMan	521.9	3.25
	TSAM2	146.1	0.54
	GTSAM	146.1	0.23
	g^{2}o	146.1	0.07

3.4 Related Work

Over the past years, SLAM has been an enormously active research area and a variety of approaches to solve the SLAM problem have been presented. More recently, optimization methods applied to graph-based SLAM formulations have become popular. Lu and Milios [67] were the first to refine a map by globally optimizing the corresponding system of equations to reduce the error introduced by constraints. Subsequently, Gutmann and Konolige [43] proposed a system for constructing graphs and for detecting loop closures incrementally. Since then, several approaches for minimizing the error in the constraint network have been proposed, including relaxation methods [36, 47], stochastic gradient descent [92] and similar variants [40] as well as smoothing techniques [52]. In a recent approach, Kaess et al. [53] provide an incremental solution to the SLAM problem that relies on Bayes Trees.

Closely related to our work presented here are hierarchical SLAM methods. And several of them perform the optimization on multiple hierarchies. For example, the ATLAS framework [15] constructs a two-level hierarchy combining a Kalman filter and global optimization.

Table 3.2: Small Memory Footprint Mapping (RPE)

Dataset	**Method**	**mean** [m]	**std** [m]	**maxErr** [m]
AIS2Klinik	SMF	**0.0045**	**0.0110**	**0.41**
	T2-NOREL	0.0148	0.2454	10.59
	HogMan	0.0100	0.0300	2.12
ETHCampus	SMF	**0.0016**	**0.0031**	**0.08**
	T2-NOREL	0.0061	0.1496	7.73
	HogMan	0.0030	0.0070	0.15
Intel	SMF	**0.0026**	**0.0037**	**0.05**
	T2-NOREL	0.0038	0.0071	0.13
	HogMan	0.0090	0.0150	0.16
Manhattan3500	SMF	**0.0147**	**0.0161**	**0.25**
	T2-NOREL	0.0153	0.0434	1.85
	HogMan	0.0330	0.0350	0.44

Estrada et al. [30] proposed Hierarchical SLAM as a technique for using independent local maps, which they merge when the robot re-visits a place. Frese [35] proposed an efficient strategy to subdivide the map into a tree of subregions. In the case of an unbalanced tree and when the leaves contain many poses, his method suffers from high computational demands. HogMan [41] builds up a multi-layer pose-graph and employs a lazy optimization scheme for online processing. The hierarchy generated is always fixed and is based on special decimation principles. One can modify HogMan so as to address low-memory situations, at the price of performing optimization only on levels that can be loaded in memory. A divide and conquer approach has been presented by Ni et al. [89], who divide the map into submaps, which are independently optimized and eventually aligned. This method was later extended by Ni and Dellaert [88], which turns out to be the closest approach, with respect to the technique, to what is presented here. They employ nested dissection to decompose the graph into a multi-level hierarchy by finding node separators, and perform inference with the cluster tree algorithm.

Kretzschmar and Stachniss [56] apply graph compression using an efficient information-theoretic graph pruning strategy. They build a constant size map by actively removing nodes in the map. After pruning them away, their information can not be recovered any longer. In contrast to them, our method stores all of the information in the hierarchy. In a visual SLAM context, Strasdat et al. [111] presented a double window optimization approach. They perform local bundle adjustment (inner window) and approximate pose-graph minimization (outer window). They reduce the complexity of full map optimization but do not bound the size of the outer window. Our approach can be seen as a multiple window optimization, where each window is one level of the hierarchy. Another, similar approach to approximating subgraphs with tree-like subgraphs, has been independently proposed by Grisetti et al. [42]. The main purpose of their work is to construct an approximation of the original problem, which exhibits a larger convergence basin, to overcome optimization failures resulting from bad initialization, *e.g.,* caused by odometry with a high noise level. To achieve this, they *condense* local submaps into a set of virtual measurements with a star-topology to improve the initial estimate for a non-linear minimization algorithm on the whole graph. In contrast to our approach, they partition the graph by determining dissecting nodes, which are then

shared by different subgraphs, while our approach uses an edge-cut. Further differences are, that they use the unscented transform to determine the virtual measurement, while we use linearized propagation of the variables. Their construction consists of a one level hierarchy, while we do not limit the number of hierarchical levels in our approach. Despite the similarity of the underlying idea the approaches aim quite different goals. Their approach aims to find a better initial guess to achieve the global minimum in a batch optimization. Our approach aims to provide an accurate and efficient solution to large-scale SLAM problems in situations in which the entire map does not fit into the main memory.

3.5 Conclusions and Future Work

In this chapter, we presented a novel hierarchical and approximated SLAM technique that produces accurate solutions and thereby requires only a small memory footprint. Our solution is particularly appealing for solving large-scale SLAM problems or for systems with limited memory. Our experimental results suggest that our approach uses significantly less memory than state-of-the-art systems and is significantly faster on systems with restricted main memory. At the same time, it yields an accuracy that is comparable to that of state-of-the-art exact solvers and exhibits better results than state-of-the-art approximate solvers.

Future Work

With the approximation of the marginal distribution of the separator nodes, we do not see a straightforward way to recover the covariances of the solution by our approach. Therefore, we cannot estimate a posteriori uncertainties of the pose estimate. From a scientific probabilistic viewpoint it would be interesting to see how this uncertainty is affected by our approach and whether or not the results are consistent, *e.g.,* for all nodes the exact solution falls into the 95% uncertainty ellipse. Another, implementation related, continuation could be to show that the presented approach can run with constant memory when dumping all currently unused data to the hard disk. However, in this case we compete with the paging implementation of the operation system and therefore would need a sophisticated implementation to compete in terms of speed.

Chapter 4

Localization with Respect to Non-Stationary Objects

The ability to precisely execute manipulations tasks is of utmost importance for autonomous robots in many situations, *e.g.,* flexible production and shared workspaces. Especially for repetitive tasks, the robot needs to know its exact pose, relative to the object that is of interest for the manipulation. However, these objects may change their position over time and for consecutive visits of the robot, the global pose and the pose relative to the object may be inconsistent. In this chapter, we decouple the global pose of the robot and the relative pose to the object, to provide both an accurate relative and global position. Thereby, we relax the static world assumption that is included in regular localization problems. To achieve this, we introduce an extension of the ICP algorithm that handles multiple reference frames in a joint manner. In extensive experiments, which include simulated and real world scenarios, our approach achieves sub-centimeter accuracy, which is sufficient for regular industrial tasks.

In the previous chapter, we developed an approximate and memory efficient SLAM solution, which solves the problem of localization and mapping in a joint manner. In this chapter, we discuss the problem of localization in the case that a global map is already given. Hereby, we are particularly interested in situations where the robot needs to be localized accurately with respect to an object, which may have a relatively small footprint in the perceived sensor measurements. Moreover, this object may change its position between consecutive visits, e.g., pallets or boxes in a shared work-space. This implies that we need to explicitly relax the static map assumption, which is commonly used in localization approaches.

The ability of self-localization is a crucial prerequisite for autonomous robots that need to reliably navigate within a known environment. In recent decades, a multitude of approaches have been developed using different techniques and sensors. However, most approaches seek an accurate global pose of the robot and include the common static map assumption, and even if this assumption holds only for major parts of it, probabilistic techniques have shown accurate results. In contrast to that, we will consider the problem in which we are explicitly interested: both, an accurate global pose as well as an accurate relative pose with respect to an object that represents only a small fraction of the map. This becomes crucial in cases where the target object may change its position over time, since in this case the global and relative

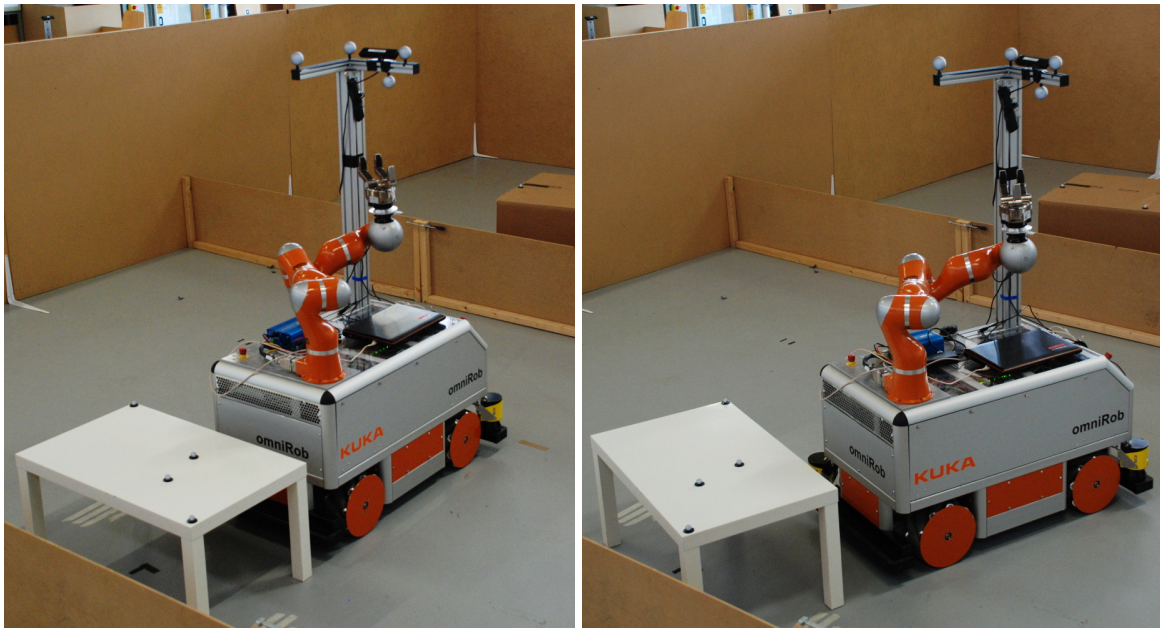


Figure 4.1: Localization example with the *omniRob* in front of a table. The two images show the same place, visited at different times. The left image illustrates the initial configuration during the mapping phase, when we drove the robot to the reference location and built a model of the table and the background. In the right image, the robot revisits this place, but in the time between the two visits, the position of the table has changed, having been shifted and rotated by an external force. Using our relative approach, the robot is able to localize and position itself relative to the table, at the same position as during the teaching phase.

poses are no longer consistent. We are convinced that this scenario is particularly relevant in industrial applications, *e.g.,* when pallets have to be moved. In this case, the objects may be found at a bound location in the map but they may be placed at a different position. An example scenario is illustrated in Figure (4.1), where the robot is required to position itself in front of an object that could be moved around.

In this chapter, we propose an approach that solely relies on a laser range finder. These sensors are commonly used for safety reasons, and therefore we expect most mobile robots in industrial facilities to be equipped with one. Instead of using a grid-map, which is a common method for map representation, we use a sensor-based point-set representation, which is composed of registered point-sets during the mapping phase. The advantage of this representation is that the noise of the sensor is implicitly reflected by the map. To solve the localization task, we reformulate the problem as a sensor registration problem with multiple rigid bodies. Therefore, our approach is able to retrieve a pose with respect to different parts of the environment, while keeping track of the global pose.

4.1 Localization with Respect to Multiple References

In this chapter, we address the problem of an accurate pose estimate that is consistent with respect to the global frame and with respect to the object frame. As long as the entire world

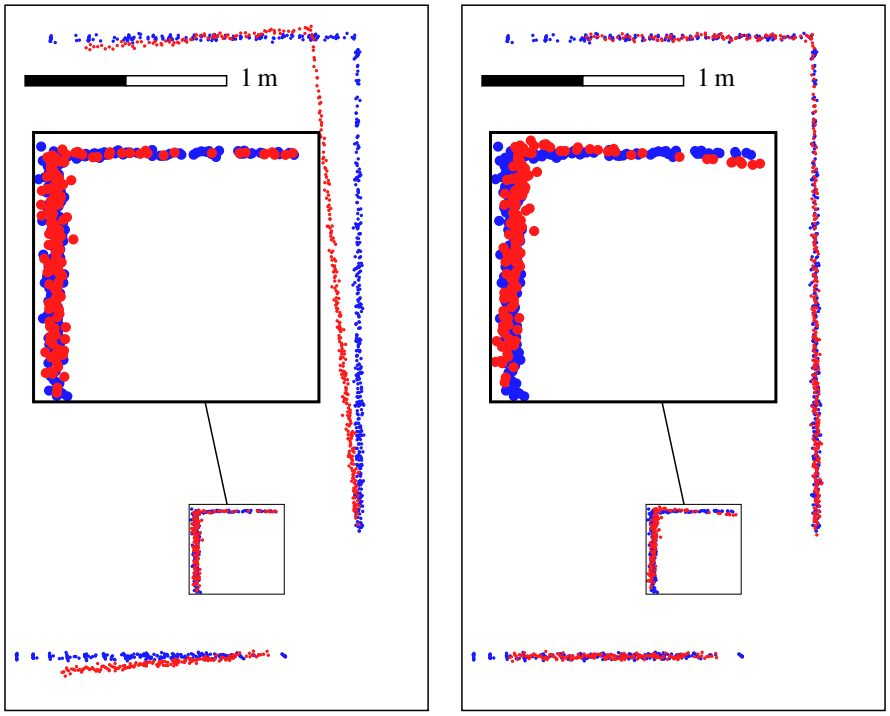


Figure 4.2: Illustration of a case in which the object (marked with the black box) was rotated after the mapping phase. This causes an inconsistency between the global pose and the relative pose to the object. On the left, the current scan (red) is projected with respect to the pose relative to the object. The misalignment to the background points (blue) is obvious, whilst we can see from the zoomed-in part that the object is aligned well. On the right, the current scan is drawn with respect to the background (global) pose. The walls align well, while the zoomed-in part shows the misalignment to the object, which is caused by the change of the position of the object.

is static, both poses are consistent, and one could be calculated directly from the other, but as soon as the object changes its position those poses become inconsistent. In Figure (4.2) we illustrate a case in which both frames are not consistent, because the object changed its position after it was mapped.

We decouple the problem of localization in two parts: first, a standard global localization, and second, the proposed approach to achieve accurate positions with respect to pre-defined objects. To solve the first part, we employ a common MCL approach, see Chapter (2.4). The second part is triggered whenever the robot arrives near the target object. Since we maintain the global pose in both cases, the transitions between the modes are smooth and straight forward. Therefore, we focus on the approach in the vicinity of the object, without limiting the generality of our approach. We regard this chapter as an extension of the work of Röwekämper et al. [98] towards localizing a robot with respect to a small fraction of the environment and relaxing the static world assumption. Röwekämper et al. [98] drove the robot to a location where a high accuracy was needed and stored local sensor measurements as reference observations in the map. Those measurements were recorded while the robot was standing still and then used for scan matching during runtime. To distinguish the objects

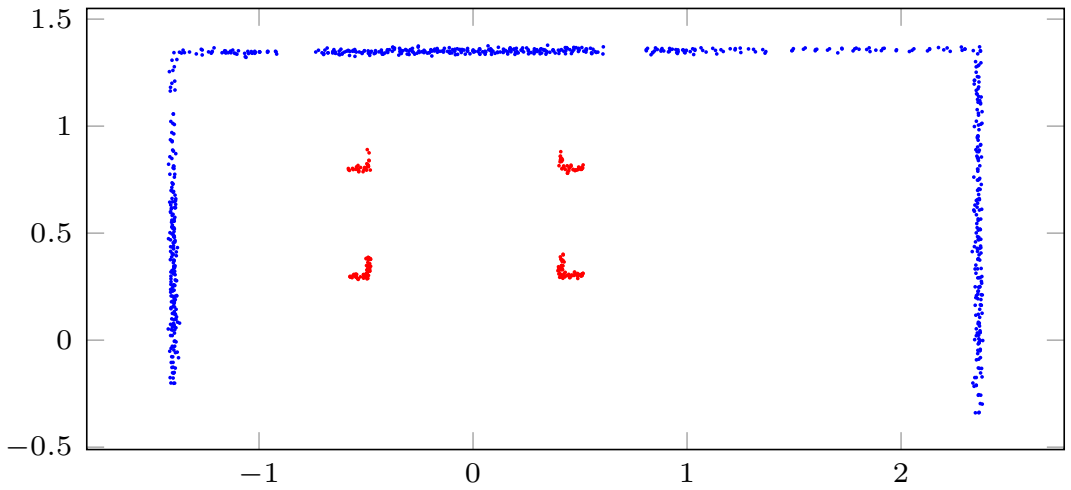


Figure 4.3: Example of a segmented reference point cloud that contains two objects. The red points stem from the legs of a table and the blue points from the background walls. Note the small footprint of the table in the sensor view.

that constitute their own reference frame, we augment the measurement points with semantic labels that determine the object of reference. Without limitations to the approach, we assume that the segmentation of the map is provided by the user, *e.g.,* using a graphical interface. Figure (4.3) illustrates an example of a reference model, where the object (a table) is depicted in red and the walls are depicted in blue.

4.1.1 Generalized ICP for Multiple Rigid Bodies

We express the problem of localization with respect to different reference frames in terms of a multi-body registration problem. Common approaches estimate a single affine transformation between two point-sets. In contrast to that, we model explicitly that different parts of the environment undergo different rigid-body motions. Given the reference point-set $\mathcal{S} = \{\mathbf{p}_i\}$, which is segmented for the rigid body objects $\mathcal{S}_j = \{\mathbf{p}_{i,j}\} \subseteq \mathcal{S}$, and a local point-set $\mathcal{Q} = \{\mathbf{q}_i\}$, regular ICP approaches calculate a single transformation that minimizes the reprojection error between the two sets and ignores the object segmentation, as described in Chapter 2.5. However, our goal is to estimate the reference pose $\mathbf{x}_j$ of each object point-set $\mathcal{S}_j$ relative to the robot position $\mathbf{x}_r$, which is expressed by the transformation $\mathbf{T}_{j,r} = \mathbf{x}_j \ominus \mathbf{x}_r$.

To simplify the notations, we assume that outliers are removed and the points of the local point-set $\mathbf{q}_i$ and the points from the reference scene $\mathbf{p}_{i,j}$ are ordered with respect to their associations. This means that point $\mathbf{p}_{i,j}$, which stems from object j, is associated to the point $\mathbf{q}_i$ of the local point-set. In general, the object correspondences of the query point are unknown. Therefore, we introduce a weighting factor $\omega_{i,j} \geq 0$, which we can use for variable purposes, *e.g.,* to model the probability that points $\mathbf{q}_i$ belong to object j or for variations of the approach adapting constraints like execution time. The objective function for the multi-body ICP is then

$$(\mathbf{T}_1^\star, \ldots, \mathbf{T}_n^\star) = \underset{(\mathbf{T}_1,\ldots,\mathbf{T}_n)}{\operatorname{argmin}} \sum_i \sum_{j \leq n} \omega_{i,j} \|\mathbf{p}_{i,j} - \mathbf{T}_{j,r}(\mathbf{q}_i)\|^2_{\Omega_{i,j}} \text{ with } \omega_{i,j} \geq 0 \;\; \forall i, j. \tag{4.1}$$

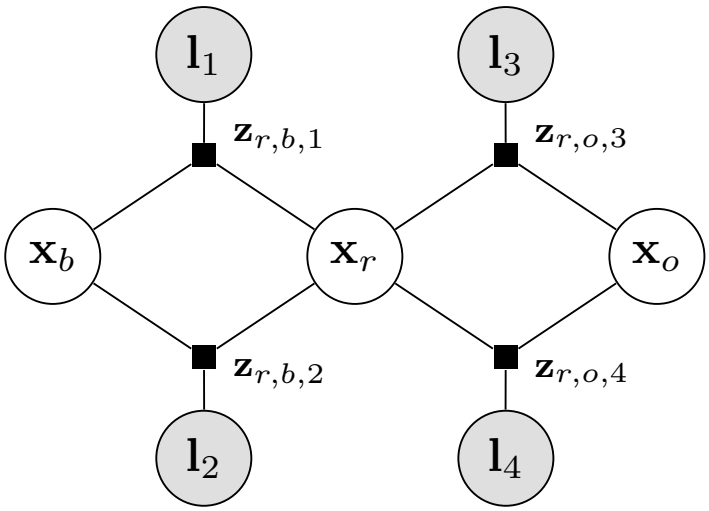


Figure 4.4: Illustration of a factor graph with two rigid bodies, $\mathbf{x}_b$ and $\mathbf{x}_o$, and the robot pose $\mathbf{x}_r$. Point associations are modeled as virtual landmark associations (shaded nodes) that induce pose-to-pose measurements $\mathbf{z}_{r,\{b,o\},i}$ (black squares) between the object and the current robot pose. With this graphical model, we can use nonlinear least-squares solvers, *e.g.,* g^{2}o, to estimate the transformation of the current robot pose and the objects.

In the case of $n = 1$, this optimization problem is the same as for the extended ICP-algorithm as given in Eq. (2.10).

In our current implementation, we use a Dirac-based weighting to compute the point associations. We assign a query point $\mathbf{q}_i$ to the closest point of any object with weight of one and zero for all the other objects, based on the euclidean reprojection error.

$$\mathbf{p}_{i^\star,j^\star} = \underset{\mathbf{p}_{l,j} \in \mathcal{S}}{\operatorname{argmin}} \|\mathbf{p}_{l,j} - \mathbf{T}_{j,r}(\mathbf{q}_i)\|_2 \tag{4.2}$$

This means, that we assign the query point $\mathbf{q}_i$ to the point $\mathbf{p}_{i^\star,j^\star}$ of object $j^\star$. Accordingly, the Dirac-based weight for the associations of the point $\mathbf{q}_i$ is $\omega_{i^\star,j} = \delta_{\{j^\star\}}(j)$. Modeling the object point correspondences in this way assimilates the regular ICP assignment of the nearest neighbor. The difference in our approach is that we take the reference pose of the object into account instead of estimating a single transformation. To find the desired transformation we apply the ICP scheme. Given an initial estimate for the transformations $\mathbf{T}_{i,r}$, we perform the weighted query point to object point correspondences assignment for each point $\mathbf{q}_i \in \mathcal{S}$ and then solve Eq. (4.1) to retrieve the next estimate of the transformations $\mathbf{T}_i$. We iterate these steps until convergence or a maximum number of iterations is reached.

In our implementation, we use the g^{2}o-optimizer to solve the minimization problem in Eq. (4.1), as we can transfer the formulation of multi-body ICP to a Graph-SLAM problem. To show this analogy we interpret the point correspondences as a common landmark observation from the respective poses, which are the reference poses of the objects and the robot pose. The measurement error of these observations is modeled as the reprojection error

$$\mathbf{z}_{i,j,r} = \mathbf{p}_{i,j} - \mathbf{T}_{j,r}(\mathbf{q}_i) \tag{4.3}$$

between the points of the reference model and the local point cloud, transformed with the current estimate of the pose of object j, $\mathbf{x}_j$. With the covariance matrix Σ_i, which models the uncertainties of the measurement, the objective function of multi-body ICP is equivalent to the Graph-SLAM problem

$$(\mathbf{x}_1^\star, \ldots, \mathbf{x}_n^\star) = \underset{(\mathbf{x}_1,\ldots,\mathbf{x}_n)}{\operatorname{argmin}} \sum_{i,j} \omega_{i,j} \| z_{i,j,r} \|^2_{\Sigma_i^{-1}} \tag{4.4}$$

The main difference to the common graph-based SLAM formulation is the iterative re-assignment of the landmark associations. However, the different error metrics of ICP, like plane-to-plane, point-to-plane and point-to-point, now become a variation of the covariance that models the uncertainties of the measurements, see Chapter (2.5), which is an intuitive interpretation of the different metrics. The correspondence association weighting factor $\omega_{i,j}$ determines the connectivity of the graph. For an illustration of such a factor graph see Figure (4.4). In our current implementation, we use the point-to-plane metric and a Levenberg-Marquard solver [80] for the non-linear least squares optimization problem.

4.2 Point Cloud Generation

In the previous section we derived the method for registration of two point-sets when multiple rigid bodies are present. Under the assumption that the entire environment is static, one can achieve sub-centimeter accuracy from a single sensor measurement, as shown by Röwekämper et al. [98]. However, the scenario we consider in this work contains rigid bodies that appear with a small footprint in the sensor measurements. This means that only a few measurements refer to some of the rigid bodies, which would not be sufficient for an accurate localization, e.g., normals can not be estimated reliably due to the sparse measurements. To overcome this problem, we propose registering multiple measurements, taken at different positions, for the generation of the local model.

4.2.1 Generating the Reference Point Cloud

We generate the reference point cloud in an offline process and one can interpret it as the teaching phase of our approach. This process is subdivided into two phases: first, the data recording, and second, the registration of the points into an accurate joint model. For the first step, we manually operate the robot in close proximity of the reference pose of the object, record the data obtained by the laser sensor and determine the reference pose. This accumulation of sensor measurements is necessary for our approach, since the target objects may have a small footprint in the sensor data and it ensures that we see the object from multiple viewpoints.

The second phase is the offline registration of the collected points. For the reference model, we want the registration to be as accurate as possible and computation time is not an issue, here. The formulation of this registration is similar to our multi-body variation of the ICP algorithm, see Eq. (4.1), except that we consider all pairwise combinations of references instead of the robot pose to the reference frames only. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be the poses of the robot during data collection. Now, for $2 \leq k \leq n$ the pose $\mathbf{x}_k$ is used as the pose $\mathbf{x}_r$ and the poses $\mathbf{x}_{k-1}, \ldots, \mathbf{x}_1$ are the reference poses, while the object is the full scan. In this process we set all

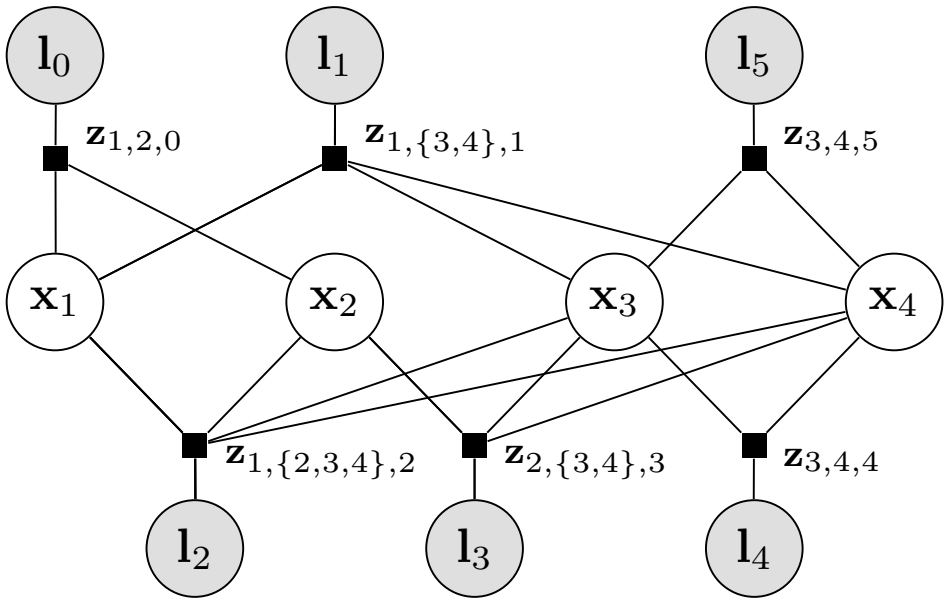


Figure 4.5: Illustration of a factor graph for the reference point cloud registration process. In this case, each pose $\mathbf{x}_i$ once acts as the robot pose and landmark associations are performed for each of the previous poses. Some landmarks are seen from more than one pose, which is the case if more than two poses share an edge with the same factor.

weights $\omega = 1$, which then results in a dense graph representation as illustrated in Figure (4.5). The objective is the accumulation of all of the measurements that arise from this construction. This again is similar to bundle adjustment, except that after each optimization, the landmark associations are recalculated, resulting in an iterative process of optimization.

4.2.2 Local Point-set Registration

At the time of operation, when the robot needs to localize itself with respect to the reference model, the time constraints from the robot control loop do not allow for running the procedure described in the previous section, as we generate a dense graph with too many associations. However, using only the current scan may lead to low accuracy, since only a few points of the object are present in a single scan, as stated before. Therefore, we will outline a procedure to incrementally register the current scan with respect to the previous N scans in the following way. Once the robot reaches the proximity of the reference object, we initialize the first pose of the procedure as retrieved from the regular MCL approach, which we use for the localization while navigating between the objects. For each of the following measurements, we initialize the pose with respect to the previously estimated pose, using the odometry readings from the robot. Let $\mathbf{x}_r$ be the initial estimate of the current scan, we then search for associations in the previous N point sets and weight each association with $\omega_{ir} = 1$. Instead of optimizing for the poses $\mathbf{x}_{r-1}, \ldots, \mathbf{x}_{r-N}$ we optimize for the current pose $\mathbf{x}_r^\star$, which leads to an accurate incremental pose estimate. The corresponding factor graph is depicted in Figure (4.6). Then we transform the local point-set with respect to the current pose $\mathbf{x}_r$ and run the multi-body ICP as described in Section (4.2) to retrieve the relative poses we are interested in.

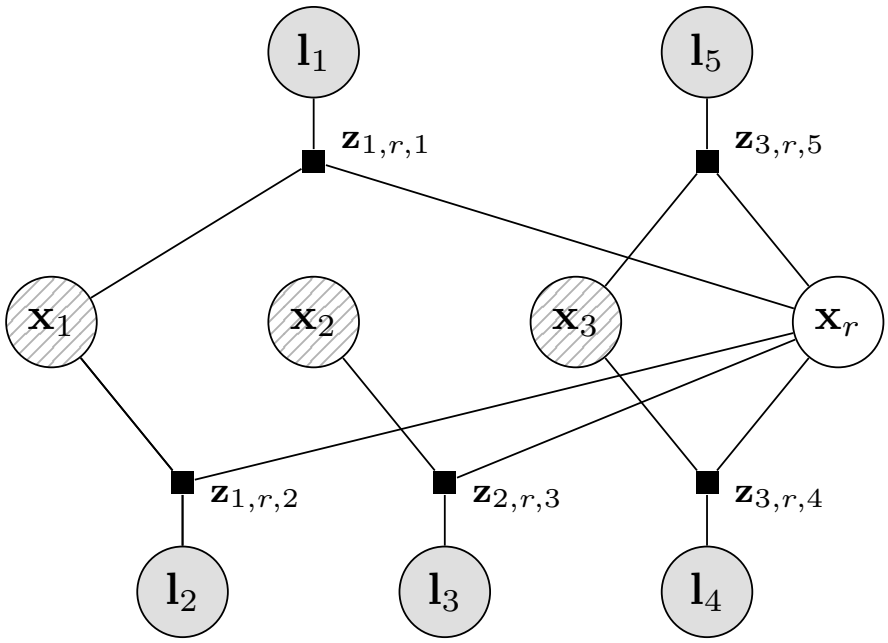


Figure 4.6: An example of the factor graph model, which we use to generate the local model when the robot is in the proximity of the target object. In this optimization, only the current robot pose $\mathbf{x}_r$ (white node) is free and the previous, already optimized poses (hatched nodes) are fixed at their positions. The point-to-point association, which is modeled as a common observation $\mathbf{z}_{j,r,i}$ of the landmark $\mathbf{l}_i$, is performed for each of the previous poses.

4.3 Experiments

For the experimental evaluation of our approach, we performed experiments in simulation as well as with a real robot. In both scenarios, we consider an environment that contains three different types of objects, which change their positions between consecutive visits of the robot. Those are: a table, a box and a shelf. We chose these objects specifically for their characteristic appearances when observed with a laser scanner. The convex shape of a box is easily detectable, as long as we observe at least one edge, from different viewpoints. The shelf has a concave, u-shaped, representation in the sensor data, which may result in self-occlusions from some viewpoints. Last, the table has a considerably small footprint in the sensor data, because the laser range finder only observes the four legs and its appearance is equivalent to a union of four small rigid boxes.

The procedure of the experiments is the same for both environments. First, we manually operate the robot and visit each object. Once we reach the proximity of the object, we start the reference model generation process as described in Section (4.2.1) and after the optimization has converged, we label the objects in resulting reference model. In the next phase, the robot visits the objects in a random order, several times. Once the robot reaches the proximity of the object we generate the local point-set, as described in Section (4.2.2). After integrating several sensor measurements, we run our multi-body ICP algorithm, as described in Section (4.1). Between consecutive visits, we change the position of the objects where we considered translations up to $10cm$ and rotations up to $10°$. To allow for better control of the

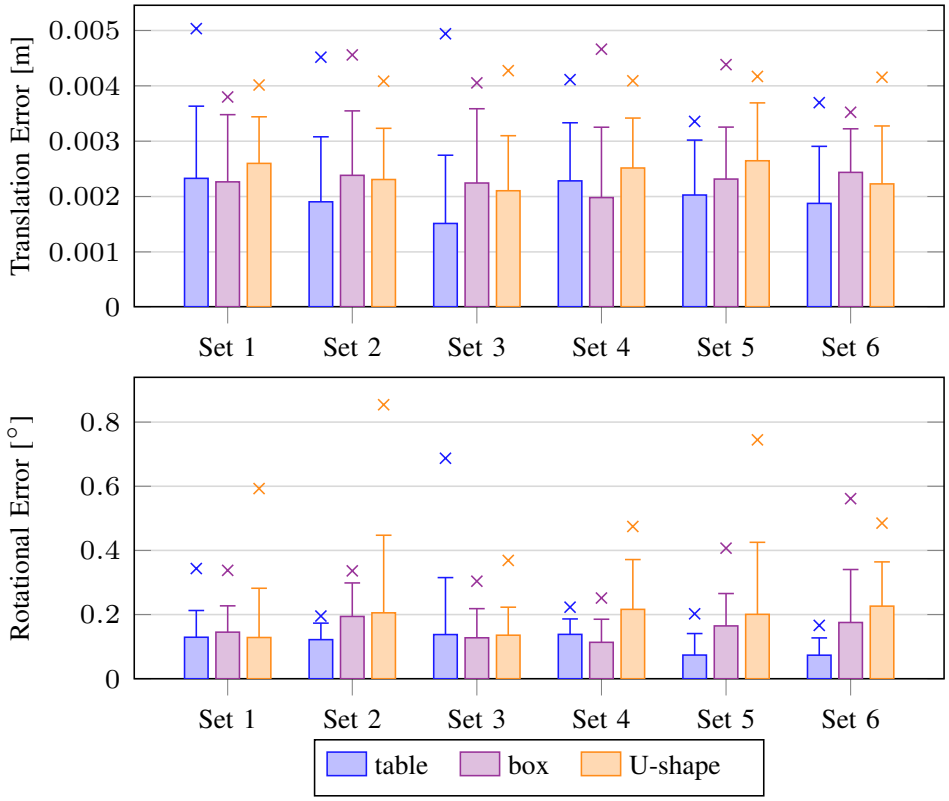


Figure 4.7: Results of the simulation experiments with the object shifted by 0.05m (Set 1), 0.1m (Set 2), rotated by $5°$ (Set 3), $10°$ (Set 4), simultaneously translated by 0.05m and rotated by $5°$ (Set 5), and simultaneously translated by 0.1m and rotated by $10°$ (Set 6). The height of the boxes show the mean error over all runs, and the bars show its standard deviation. The crosses mark the maximum errors.

viewpoint variety and to avoid viewpoint biased results, which may arise from the planner used in the system, we opted for a manual operation of the robot for this phase, as well. To evaluate the accuracy of our approach, we compare the estimated pose relative to the object to the ground truth.

4.3.1 Simulation Experiment

In the simulation experiment, we consider a differential drive robot with a laser range finder mounted parallel to the ground plane. To simulate the odometry of the robot, we use a velocity motion model with zero-mean Gaussian noise and a standard deviation of $0.1\frac{m}{s}$ for the translational velocity, and zero-mean Gaussian noise and a standard deviation of $0.1\frac{rad}{s}$ for the rotational velocity. The laser sensor is simulated with a field of view of $180°$ and an angular resolution of $0.5°$. The range readings are calculated with ray-tracing and we add random noise to each measurement, which is modeled as a zero-mean Gaussian distribution with a standard deviation of $0.01m$. The map of the environment that we used in the simulation experiments is pictured in Figure (4.8).

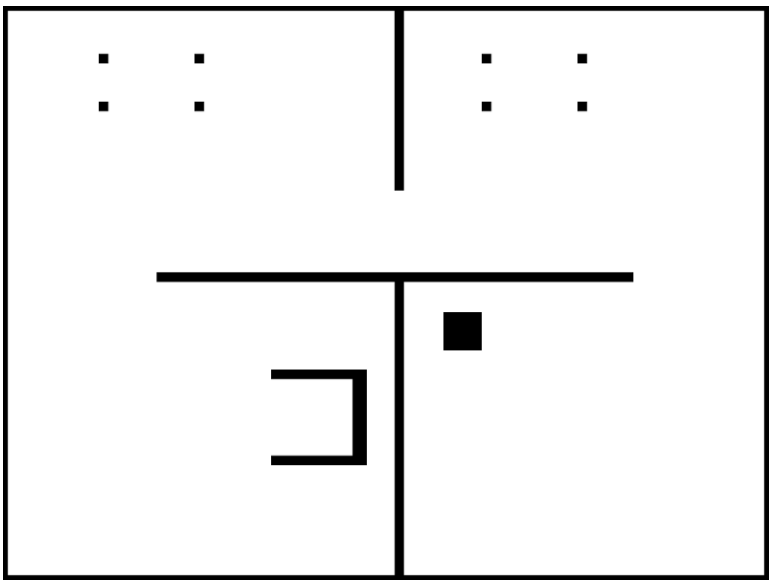


Figure 4.8: This is the a map of the environment that we used for the simulation experiments. The extent of the environment is $8\,\mathrm{m} \times 6\,\mathrm{m}$. For the evaluation of our approach, we used the box in the lower right, the u-shaped shelf in the lower left and the table in the upper right room.

In this experiment, we evaluate six different ways that the object may be moved between consecutive visits:

- Set 1: translation of $0.05m$
- Set 2: translation of $0.1m$
- Set 3: rotation of $5°$
- Set 4: rotation of $10°$
- Set 5: translation of $0.05m$ and rotation of $5°$
- Set 6: translation of $0.1m$ and rotation of $10°$

For each of the sets we perform ten localization runs at each object.

The results of this experiment are illustrated in Figure (4.7), where we draw: the mean error, which is the height of the box; the standard deviation, which is the error bar in the positive direction; and the maximum error, which is marked with the cross, in terms of translation and rotation, for each set and object. The error behavior is similar for all configurations, which indicates the robustness of our approach. The mean translational error is below $2.6mm$ for all settings and objects, with a standard deviation that is always less than $1.4mm$, and the maximum error is about $5mm$. The mean rotational error is lower than $0.23°$ for all settings with a maximum standard deviation of $0.24°$, which occurs for the u-shaped shelf, and a maximum error of at most $0.85°$. These results are promising and very accurate; we will see in the forthcoming section whether or not we can achieve a similar performance in the real world experiment.

Figure 4.9: An image of the real world environment that we used for the experimental evaluation of our approach. We modeled three rooms each of which contained one of the three objects we consider: a table in the lower left, a u-shaped shelf in the middle and a rectangular card box in the upper left room.

4.3.2 Real-World experiments

For the real world experiment, we set up an exemplary environment similar to the one of the simulation experiments, see Figure (4.9) for an image of the setting. We used KUKA omniRob as the mobile robot, see Figure (4.1), which is equipped with *SICK-S300 Professional* laser scanner, which exhibits a statistical standard deviation, including the systematic error, of $2.9cm$ [106]. To provide ground truth for this experiment, we employed an optical motion capture system, which is equipped with ten high speed Raptor-E cameras from Motion Analysis Digital, that use infrared light to detect and track reflective markers. The robot and each of the objects are prepared with a unique constellation of markers, which allows for both an exact pose estimate and the identification of the object.

In this environment we use the following sets of motion for the reference objects:

- Set 1: translation of $0.1m$
- Set 2: rotation of $10°$
- Set 3: translation of $0.1m$ and rotation of $10°$

In this scenario, we perform ten localization runs for each set at each object as with the simulation experiments.

The results of this experiment are depicted in Figure (4.10). For (Set 1) the translational error is similar for all objects, with mean error of less than $5mm$ and a maximum error less than $10mm$. However, for the rotational errors our approach achieves the best results for the table, and we observe the largest errors for the shelf, which is still less than $0.5°$. When the objects are rotated, in (Set 2), we observe that the translational errors increase substantially for the table and box, while they are stable for the shelf. The same accounts for the rotational errors. The combination of translation and rotation of the objects in (Set 3) has almost no

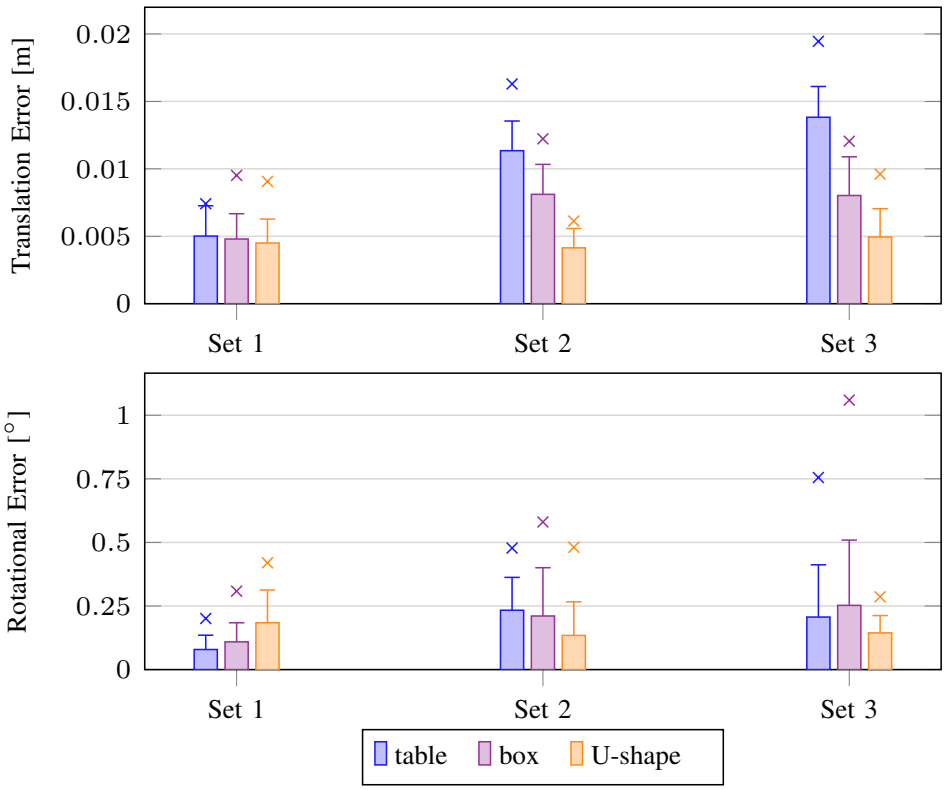


Figure 4.10: Results of the real world experiments with the object shifted by $0.1\mathrm{m}$ (Set 1), rotated by $10°$ (Set 2), and simultaneously rotated and translated (Set 3). The boxes indicate the mean error over all runs and the bars its standard deviation. The crosses mark the maximum errors.

effect for the mean of both measures, but for the rotational errors, we see a larger standard deviation for the box and table. However, for all experiments the mean translational error is always less than $15mm$ with a maximum error of at most $19mm$, and for rotational errors the mean is always less than $0.3°$ with a maximum error of at most $1.1°$.

It is worth noting that the loss of accuracy after a rotation of the object may partially be caused by small errors of the motion capture system and the distance of the robot to the object. If we consider that the reference pose of the robot is a distance of about one and a half meters from the object, a rotational error in the motion capture data of $0.25°$ may lead to an translational displacement of the reference pose of $6mm$. Taking this into account, we can safely consider the results obtained in this section as a lower bound of accuracy that is reached with our approach. However, in any case the results of our approach meet the accuracy bounds for loading tasks in industrial environments, which Saarinen et al. [101] claimed to be less than $30mm$.

4.4 Related Work

Accurate localization for industrial tasks is an active research area and has been addressed by several researchers. Saarinen et al. [101] proposed to use the Normal-Distribution-Transform in a MCL framework (NDT-MCL). The NDT uses a set of normal distributions that model the

environment in a piece-wise continuous way, to reduce the discretization effect of occupancy grid maps. Their approach achieves a positioning accuracy of $14mm$ and they claim that the minimal accuracy required for industrial loading tasks is at $30mm$. An MCL-based approach that positions a robot at a specific location with an accuracy of few millimeters, was presented by Röwekämper et al. [98]. As in our approach, their method uses multiple scans as map representations of the reference pose. Moreover, they employ a validity check of the pose estimate, comparing the results of two independent laser scanners that are mounted on different sides (front and rear) of the robot. In contrast to them, the approach we presented in this chapter achieves a high accuracy even with respect to objects that undergo rigid body motions.

Few authors address the problem of determining possible docking locations for objects from sensor data. Williamson et al. [132] present an approach that combines a mixture of Gaussians and an Expectation Maximization (EM) algorithm, for robot docking. Each mixture represents a possible object, and possible docking points are assigned by the EM algorithm. Jain and Argall [50] use the Kinect sensor to detect edges of tables, bowls, or cups to find possible docking candidates. This work is complementary to the approach we described in this chapter; they can be used to automate the finding of the reference pose for the respective object. However, the focus of their work is on the safety of people in wheelchairs and the accuracy they achieve is, as yet, too low for industrial settings.

Other authors address the problem non-stationary objects in an environment during the mapping and localization. To name two examples: the approaches of Biswas et al. [13] and Anguelov et al. [6] compute shape models of non-stationary objects. They compare maps created at different times using an EM-based algorithm in order to identify parts of the environment that have changed over time. Thereby, their main focus was understanding which parts of the environment can change their position over time. The accuracy of the approaches is beyond the purpose of the approach presented in this chapter, which satisfies industrial needs. The approach presented by Salas-Moreno et al. [102] integrated the location of objects in the context of SLAM. Although their approach achieves a good accuracy, the work still relies on the assumption that the entire map, including the objects, is static. These works could also be used to complement our approach, *e.g.,* for the automation of the object labeling during the training phase of our approach.

Another field of active research is the detection and tracking of dynamic objects, where dynamic (in this context) means that objects move continuously. For example, Ahmad et al. [3] extended the graph-based SLAM formulation to include dynamic objects. However, their work focuses on constantly moving objects for which the motion model is known. Another approach presented by Dewan et al. [28], targets the problem of detection and tracking objects in highly dynamic environments, *e.g.,* traffic. Although this approach is model free, it needs objects to move continuously in order to be detected. The approach of Tipaldi and Ramos [122] proposes a conditional random field clustering technique to perform motion segmentation simultaneously with the computation of the number of objects as well as their displacement. An extension of this approach, by modeling the data association within the probabilistic framework, was published by van de Ven et al. [128]. In a series of publications, Yang and Wang developed a multi-model extension for RANSAC, which makes it applicable to rapidly changing environments [134]. For better robustness against segmentation errors, the authors proposed a multi-scale algorithm [136] and include spatio-temporal information in stationary and dynamic objects [135]. Contrary to these works, we do not limit ourselves to just using two sequential scan measurements, instead we accumulate points over time, while the robot is

approaching the target, to improve the localization accuracy.

4.5 Conclusion and Future Work

In this chapter, we presented an approach to accurately localize a mobile robot with respect to a reference object that may change its position between consecutive visits. Our approach combines existing state-of-the-art global localization techniques, where we use a standard MCL in our current implementation, with the proposed ICP variant that explicitly decomposes the scene for different rigid bodies that may undergo different rigid body motions. Furthermore, we proposed methods to compute accurate models of the environment for the training and localization phase, using a variation of the multi-body ICP approach. This improves the accuracy of the localization and indicates the flexibility of the proposed formulation of our ICP variant. To analyze the localization accuracy of our approach relative to the target object, we performed simulation and real world experiments. The results show that our approach achieves an accuracy of at least $20mm$, which we believe is sufficient for typical industrial tasks even when the target object was shifted $10cm$ and rotated by 10° degrees.

Future Work

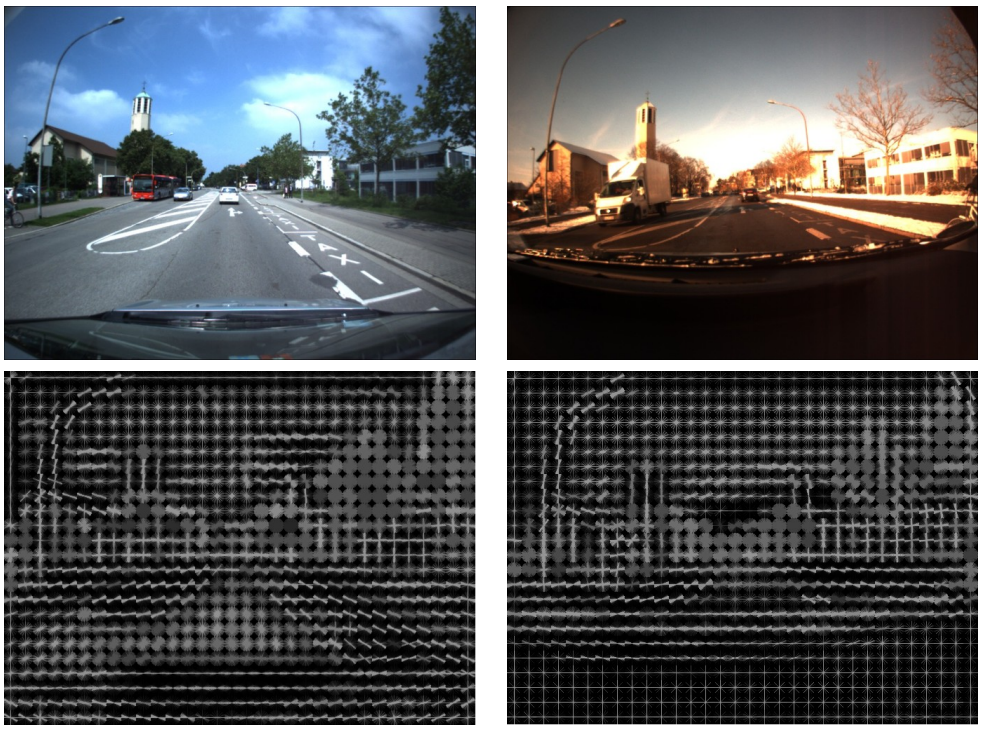
So far, we have used the nearest neighbor assignment of points and have taken the object label from the assigned point. This is straightforward, considering the ICP-based nature of the approach. However, this may lead to problems if the objects are close to each other or if an object is moved to a substantially different position, *e.g.,* several meters. One possibility to overcome this limitation could be to improve the point to object assignment, decoupling this phase from the point-to-point associations by actively searching and segmenting the point-sets for the objects. However, this would also lead to a higher computational burden. Another advancement would be to automate the phases that need manual assistance from a human during the training phase. This could be achieved by integrating approaches that aim to learn which objects change their positions and the mapping approaches that consider independent objects, as discussed in Section (4.4).

Chapter 5

Camera-based Localization Facing Substantial Perceptual Changes

Camera-based robot localization has become very popular in recent years. They provide an absolute cost advantage when compared to other perceptual sensors as, e.g., laser scanners. However, camera-based localization is prone to visual changes of the environment, which may be caused by different lighting conditions or seasonal changes. This problem becomes especially important when we think about life-long autonomy, where a robot needs to reliably navigate for a long period of time. In this chapter, we present an approach to camera-based localization on a sequence of images, in which the map sequence and the localization sequence exhibit major appearance changes, mainly caused by seasonal changes of the environment. Under such extreme circumstances, pairwise image matching techniques tend to fail. We exploit the sequence structure of the data and employ a recursive Bayesian filter technique, which results in a more robust localization. To determine the similarity between images, we use a whole image HOG descriptor and compare their cosine distances. Moreover, we investigate the benefits of applying a decorrelation technique to the image similarities to increase their distinctiveness. In extensive experiments, we show that our approach outperforms state-of-the-art techniques for a variety of challenging datasets.

In the previous chapter, we developed a method to cope with scenarios in which parts of the environment change their position over time. Thereby, we used a laser scanner as a perceptual sensor, which is a common sensor for industrial applications due to additional safety constraints. However, the main drawback of laser scanners is that they are relatively more expensive when compared to cameras. One drawback of using a single camera is that we do not get any metric information for what we see. Therefore, the data is sensitive to viewpoint and orientation changes, which constitutes one problem of camera-based approaches. Moreover, data perceived by a camera may be highly affected by changes in the lighting conditions, e.g., day and night or sun glare. In this chapter, we will focus on a scenario in which images recorded in a sequence are supposed to be matched to a previous run. We assume that both sequences share somewhat similar viewpoints but the appearances of parts of the environment have changed remarkably.



Summer Winter

Figure 5.1: Local gradient information in the images are robust to seasonal changes. The geometric information is stable, even though the appearance has substantially changed, e.g., see the streetlamps or the church. Our approach needs to cope with different view points, seasonal changes and illumination changes.

Vision-based localization has been deeply investigated in recent years [24, 76, 77]. However, varying environmental conditions still constitute a challenging problem for localization. So far, techniques that determine key-points and compute their descriptors, e.g., SIFT and SURF, have shown promising results when localizing under similar conditions. The main focus of these approaches are to find rotation- and scale-invariant key-points and descriptors, to avoid false positive matches, *e.g.,* Cummins and Newman [23]. When localizing a robot in varying conditions, such that the places exhibit a large perceptual difference, it turns out that key-point-based methods suffer not only from unstable descriptors but also from the fact that the keypoints are detected at spatially different pixels in the image, as claimed by Naseer et al. [82]. Therefore, we will instead use a whole image descriptor for which we calculate and stack HOG descriptors. The intuition for using HOG descriptors is that the gradient information stresses the geometrical structures that are persistent over time, e.g., buildings and poles. We depict this effect in Figure (5.1) with a pair of images and their HOG representations, where the appearance of the same place changed substantially due to seasonal changes and lighting conditions. Even though choosing a proper descriptor for the images is crucial for localization, similarities between images often exhibit high noise and relying purely on those would lead to a large number of false positive matches. To overcome this, recent methods explore the property that the images are recorded in a sequence Milford and Wyeth [77], Naseer et al. [82]. The work we present in this chapter will take this sequential characteristic into account as

well, and overcome some limiting assumptions of the aforementioned approaches. In our approach, we model the state of the robot with a recursive Bayes filter, which allows for a much more flexible transition model compared to state-of-the-art approaches, and therefore, our approach can handle more complex trajectories in a natural way due to the probabilistic formulation. Moreover, we show in our experiments that applying a whitening transformation to the image similarities reduces ambiguities and leads to a better performance.

5.1 Visual Localization Utilizing Sequential Information

In this chapter, a camera is the only sensor source for the mobile robot, which provides data at a low frame rate; let us consider at most four frames per second. This low frame rate makes visual odometry infeasible, and hence, no metrical information can be taken into account. Instead, we make the natural assumption that the images are recorded in a sequence during operation of a mobile robot. A dataset is composed of two such sequences, where we refer to the first as the database, $\mathcal{D} = \{d_1, \ldots, d_{n_D}\}$ with $n_D = |\mathcal{D}|$, and the second we will refer to as the query set, $\mathcal{Q} = \{q_1, \ldots, q_{n_Q}\}$ with $n_Q = |\mathcal{Q}|$. Now, each of the sequences are temporally ordered and we are particularly interested in cases where the two sequences are recorded with a large time gap between, e.g., spring and winter, which causes different appearances of the same places.

In the remainder of this section, we elaborate on the components of our approach. First, we explain how we compute the similarity matrix between the two datasets in Section (5.1.1). Afterwards, we briefly explain the model of the discrete Bayes filter and its components in Section (5.1.2) and Section (5.1.3) and show how to integrate the image similarities in Section (5.1.4). In Section (5.2), we outline how to compute the final probabilities for image q_j is perceived from the same place as image d_i for each combination of i and j, using the full data that is available. Afterwards, the sequential filtering is explained in Section (5.3). We close the technical part of this chapter with the explanation of the zero component analysis and outline how we apply it to our framework in Section (5.4).

5.1.1 Robust Image Matching

The input to our approach are images of a fixed resolution. To recognize places after substantial changes in their appearance, we need a descriptor that is not affected by those changes. As keypoint-based approaches are rather unstable and prone to errors after changes in appearances, we use a whole-image-descriptor based on gradient histograms. Specifically, we use the HOG descriptor, which works on a local patch of the image, as the base for the whole-image descriptor. We sub-divide the image in patches of size 32x32 pixels and compute the HOG descriptor for every patch. See Figure (5.1) for a visualization of the descriptors for each of the patches. Finally, we stack the descriptors to a single vector $\mathbf{f}$. Considering an image of size 1024×768 this would result in 32×24 HOG descriptors $\mathbf{h}_i$, which we stack to a single vector $\mathbf{f}$ of dimension $32 * 24 * |\mathbf{h}|$.

To quantify the similarity between two images, we calculate the *cosine*-similarity between the two whole image descriptors. For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ the real-valued *cosine*-similarity is calculated as:

$$sim(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^{\mathrm{T}} \cdot \mathbf{y}}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2} \tag{5.1}$$

In general, the *cosine*-similarity can take values in the full range $[-1, 1]$. However, in our case the values of the feature vectors are all positive, and therefore $sim(\mathbf{f}_1, \mathbf{f}_2) \in [0, 1]$ for all pairs of feature vectors we use in this chapter. The cosine similarity only accounts for the angle between the vectors and not for their magnitude. It is, for example, often used to compare text similarities based on term frequencies, e.g., see Adomavicius and Tuzhilin [1]. For computational efficiency we can normalize the whole-image descriptor $\mathbf{f} = \mathbf{f}/\|\mathbf{f}\|_2$ to avoid the frequent re-computation of the L_2-norm. Given our database set $\mathcal{D} = \{\mathbf{f}_1^d, \ldots, \mathbf{f}_{n_D}^d\}$ and the query set $\mathcal{Q} = \{\mathbf{f}_1^q, \ldots, \mathbf{f}_{n_Q}^q\}$ represented with the HOG descriptors of the respective image, we can now compute the similarity between image $q_i \in \mathcal{Q}$ and $d_j \in \mathcal{D}$ as

$$s_{ij} = sim\left(\mathbf{f}_i^q, \mathbf{f}_j^d\right) \tag{5.2}$$

which forms our similarity matrix $\mathbf{S}$ of size $n_Q \times n_D$ for a dataset. The top row of Figure (5.2) visualizes the similarity matrix. We see that the similarities suffer from noise, high values are present at random positions and often form blocks. Therefore, we will apply a two step pre-processing to the similarities in order to achieve more meaningful values.

First, we do a column-wise mean normalization, adjusting the mean of each column to one.

$$\hat{s}_{ij} = s_{ij} \left(\frac{1}{n_Q} \sum_{n=1}^{n_Q} s_{nj} \right)^{-1} \tag{5.3}$$

The effect of this normalization is visualized in the second row of Figure (5.2). As we can see, the parts where we had high similarities for each query are more smooth now. However, since we divide by the mean of the columns, the values between different columns may vary in a wide range. Therefore, we apply a second pre-processing step, in which we transform the values of each row to the interval $[0, 1]$. We do this by subtracting the minimum value and dividing by the remaining maximum value of the row.

$$\tilde{s}_{ij} = \frac{\hat{s}_{ij} - \min\limits_{m=1,\ldots,n_D} \hat{s}_{im}}{\max\limits_{n=1,\ldots,n_D} \left(\hat{s}_{in} - \min\limits_{m=1,\ldots,n_D} \hat{s}_{im} \right)} \tag{5.4}$$

The resulting similarity matrix $\tilde{\mathbf{S}}$ is visualized at the bottom of Figure (5.2). The zoomed-in part shows that the pre-processing procedure superimposes the path of the robot relative to the surroundings. For the filter-based localization, which is explained in the forthcoming sections, we are interested in the probability that the place at which image q_i is taken, is the same as the one where image d_j was taken. We will model this proportional to $\tilde{s}_{ij}$.

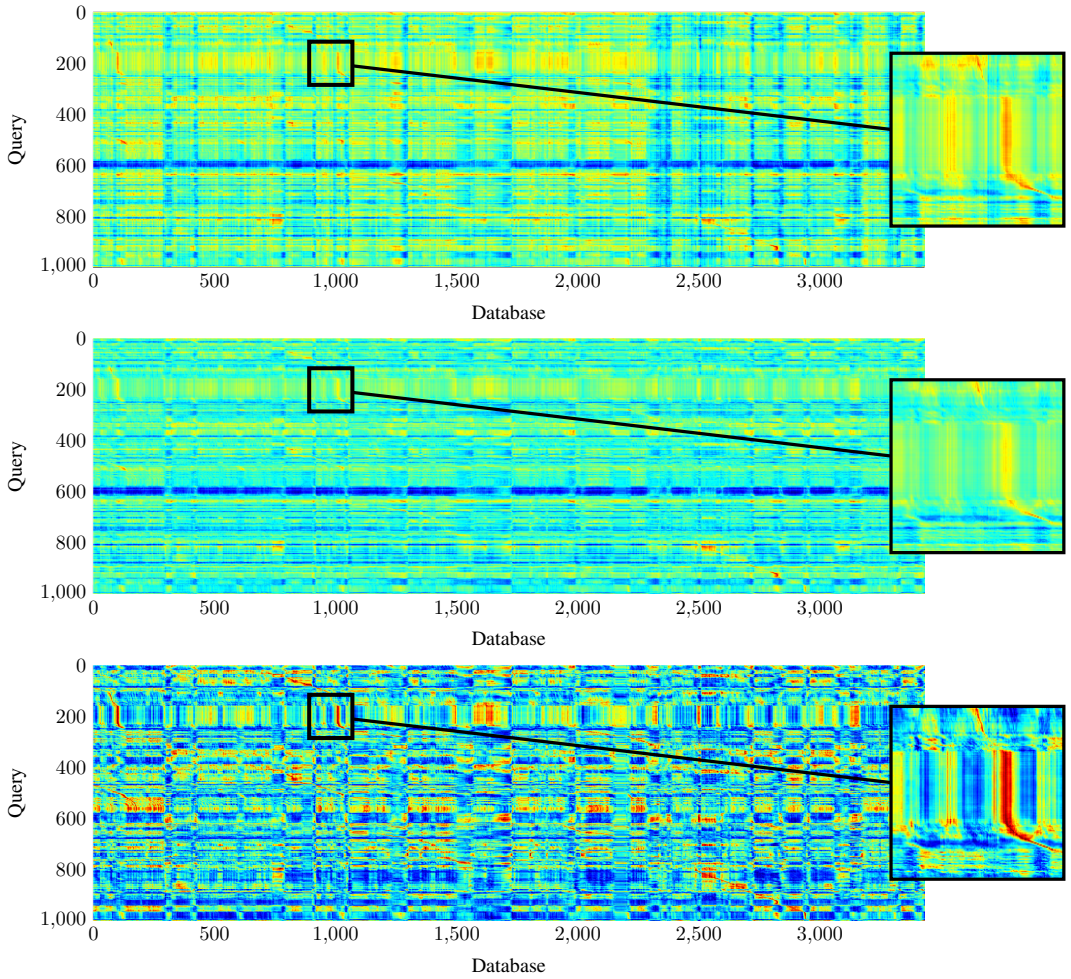


Figure 5.2: This figure shows the effect of the two pass normalization used in our approach. We show a zoomed-in part of the score matrix for better visualization. Column wise normalization in the first pass helps to disambiguate the confusing database images that could match against all query images and provide high scores. Although the scores do not look highly discriminative, it greatly reduces the noise in the matrix. In the second pass, we stretch the scores between 0 and 1 to achieve more distinctive scores.

5.1.2 Discrete Bayes Filter

For the localization task, we are interested in the pose of the robot at each time t. Since in our case, we do not have any further information except the images, we discretize the time according to the frequency of the images of the query set. In our probabilistic model, our goal is to estimate the belief of the robot's position x_t given the sensor data history $\mathcal{Z}_t = \{z_1, \ldots, z_t\}$ and the history of controls $\mathcal{U}_t = \{u_0, \ldots, u_{t-1}\}$.

$$Bel(x_t) = p(\mathbf{x}_t \mid \mathcal{Z}_t, \mathcal{U}_t) \tag{5.5}$$

Even though we consider that information of the controls is not available in our scenario, we keep them in the model, since our approach can integrate them when they are available.

It is a well known fact, that the complexity of the state estimate grows exponentially with the operation time, which makes it computationally infeasible to estimate the full distribution

at once (Fox et al. [34]). However, if the dynamics of the system, $P(x_t \mid x_{t-1}, u_{t-1})$, are known, we can compute the belief recursively without loss of information, which is known as the Bayes filter.

$$Bel(x_t) = \eta_t p(z_t \mid x_t) \sum_{x_{t-1}} P(x_t \mid x_{t-1}, u_{t-1}) Bel(x_{t-1}) \tag{5.6}$$

The sum goes over all possible states at time $t-1$, which is a discrete set, since for all timesteps we assume that $x_t \in \{1, \ldots, n_D\}$. The normalization constant η_t does not depend on the state x_t and ensures that the sum of the probabilities over all possible states is equal to one. In the forthcoming sections, we will briefly discuss the transition- and measurement model.

5.1.3 State Transition Model

In this section, we will describe the transition model, $P(x_t \mid x_{t-1}, u_{t-1})$, of the Bayes filter as it is used in our approach. In our case, the only available prior information we assume, is that the images are recorded in a sequence. We make use of this information, by setting the transition probabilities for nearby places relatively higher than for all other places. Therefore, we choose parameters $c_f, c_b, c_s > 0$ for forward, stationary and backward transitions:

$$P(x_t|x_{t-1}) = \alpha_t * \begin{cases} c_f & \text{if } f > x_t - x_{t-1} > 0 \\ c_s & \text{if } x_t - x_{t-1} = 0 \\ c_b & \text{if } -b < x_t - x_{t-1} < 0 \\ 1 & \text{else} \end{cases} \tag{5.7}$$

where α_t is a normalization factor, that ensures that we model an appropriate probability distribution. The interpretation of the parameters c_f, c_b and c_s is that it is c_f times more likely to do a transition within $]x_{t-1}, x_{t-1} + f[$ than anywhere else outside the interval $[x_{t-1} - b, x_{t-1} + f]$. The same applies to the parameters c_s and c_b. The intuition behind this is that both the database as well as the query images are recorded in a sequence, and if we go with a similar speed, it is very likely that if we are at position i at timestep $t-1$ then we will be at position $i+1$ at timestep t. Nevertheless, we can never be sure that we take exactly the same route in both runs, and therefore, the transition model needs to provide non-zero probabilities for all possible states. This property is not given by other state-of-the-art approaches that focus on the same problem. For example, the network-flow approach by Naseer et al. [82] only allows transitions in the forward direction with a strict maximum step width, i.e., $0 \leq x_t - x_{t-1} \leq k$ for some fixed constant k. SeqSLAM [76] considers a strict linear motion model. In contrast, we can handle trajectories with shortcuts, detours and loops. Moreover, our probabilistic formulation allows multiple modes, which we need in the case of revisiting the same place within the database run.

5.1.4 Sensor Model

For the last part, we need to specify for the Bayes filter described in Section (5.1.2) is the sensor model. For this, we use the similarity measure between two images as described in

Section (5.1.1). There we established similarity scores for each image pair and we set the likelihood for our sensor model as the normalized score

$$p(z = q_i \mid x_t = j) \propto \tilde{\mathbf{S}}_{ij} \tag{5.8}$$

This means that we model the probability of observing image q_i at position $x_t = j$ of the database is proportional to the pre-processed similarity between the images q_i and d_j. This completes the belief computation of our approach.

5.2 Forward Backward Smoothing

To compute the final belief matrix $\mathbf{B}$, where each row corresponds to a probability distribution over all possible places from the database, we use the procedure described in the previous section (Section (5.1)). Therefore, we initialize the recursive procedure with a uniform distribution, $Bel(x_0 = i) = 1/n_D$ for all $1 \leq i \leq n_D$, since we do not have any prior information about the starting point. However, if a more informed prior distribution would be available one should use it instead of the proposed uniform prior.

As the sensor model, which is based on the similarities of images, exhibits much noise, we compute the belief matrix by smoothing two passes of the filter. The first one, $\mathbf{B}_f$, is computed in the direction of movement of the robot, ordered by the provided time steps. The second one, $\mathbf{B}_b$, is computed in a time-reversed order. Therefore, we reverse the order of the database as well as of the query set. We then compute the final belief matrix $\mathbf{B}$ as the normalized geometric mean of the two passes.

$$\mathbf{B} = \lambda \cdot \sqrt{\mathbf{B}_f \cdot \mathbf{B}_b} \tag{5.9}$$

Where the operators $\cdot$ and $\sqrt{\ldots}$ denote element-wise operations, *i.e.* multiplication and square root, and λ is the row-wise constant $n_Q \times n_D$ normalization matrix, which ensures a valid probability distribution at each row of $\mathbf{B}$.

Our reasoning for choosing this two-pass smoothing is that the recursive nature of the Bayesian filter and the noisy measurements, may lead to an overconfident estimate in case of jumps. This would result in false-positive matches at the end of sub-sequences. On the other hand, when entering a sequence it may take some steps until the filter accumulates probability mass to the area. These effects are reversed for the backward pass.

5.3 Sequential Filtering

In our scenario, we have neither an exact transition model, nor a precise sensor model. Nevertheless, we have derived a meaningful posterior distribution in the previous chapters. The next step is inferring a path hypothesis from the belief matrix $\mathbf{B}$. At this point, the main issues are outliers, which are false positive matches with a high likelihood, which seem to occur randomly at different spots. However, these outliers are typically isolated, and we assume that they do not occur in long sequences. In contrast to that, the true positive matches will form sequences, since we know that the images are recorded at consecutive time-steps during a continuous run of the robot.

As a first step, we search the belief matrix for local peaks that are above a minimum probability of p_{min}. In this way, a neighborhood of relative high probabilities is represented by a single peak, which is the maximum likelihood relative to this neighborhood. This operation implicitly handles the case in which there is no true positive match for a query image q_i due to visits of so far unseen places during the query run, as we expect the probabilities to be equally low then.

Second, we check whether or not those remaining values form sequence. We will use three parameters to define: a sequence, the minimum length l_s, the maximum row gap g_r and the maximum column gap g_c. The latter two parameters matter for two reasons: first, the sequences may have been recorded at different speeds, which would result in no matching images between two consecutive time steps and second, the noisy observations may shadow consecutive matches.

More specifically, given a row-wise ordered set of matches

$$\mathcal{M} = \{(i_1, j_1), \ldots, (i_n, j_n) \mid \forall k = 1, \ldots, n-1 : i_k < i_{k+1}\} \tag{5.10}$$

with $\mathcal{M} \subset \mathcal{Q} \times \mathcal{D}$ we consider $\mathcal{M}$ as a sequence of matches if and only if $n \geq l_s$ and for all $k = 1, \ldots, n-1$ the conditions $i_{k+1} - i_k \leq g_r$ and $|j_{k+1} - j_k| \leq g_c$ are satisfied. The final estimated trajectory is the union of the sequences that satisfy these conditions. Obviously, we may loose some true positive matches and still have some false positive matches in our final trajectory. However, in our experimental evaluation, our approach, as described to this point, outperforms state-of-the-art approaches on most of the datasets. To improve the performance of the approach, we present a further pre-processing step utilizing a whitening transformation, which aims for more distinctive scores, in the upcoming section.

5.4 Zero Component Analysis Whitening

Data pre-processing with withening transformations is commonly done when analyzing image data. Typically, it is applied in the feature space, in order to de-correlate and normalize the features covariance. However, in our case the feature space has $98\,304$ dimensions and whitening in that space is seemingly infeasible due to the computational effort, which includes the computation of a SVD of the covariance, which is a dense matrix of size $98\,304 \times 98\,304$.

However, encouraged from the improvements that can be achieved using ZCA-whitening, we successfully experimented with applying the whitening transformation to the initial similarity matrix, as a pre-processing step. In our evaluation, we achieve an average improvement of 17% for the F_1-Scores compared to using the raw score-matrix as described in Section (5.1.1). We visualize the effect of the ZCA-whitening transform to the similarity matrix $\mathbf{S}$ in Figure (5.3). It is clear that the path of the robot is more distinguishable after the transform; particularly, the region highlighted with the blue rectangle, where we can see that the high values previously present in the original matrix have been greatly smoothed and which do not appear in the transformed matrix at the bottom. However, a drawback of the whitening transform is that we may lose correspondences in case of stoppages, as marked by the red rectangle. Nevertheless, the benefits of the whitening transform outweigh the drawbacks, *e.g.,* especially as stoppages could be removed in advance without losing the sequential characteristics of the data.

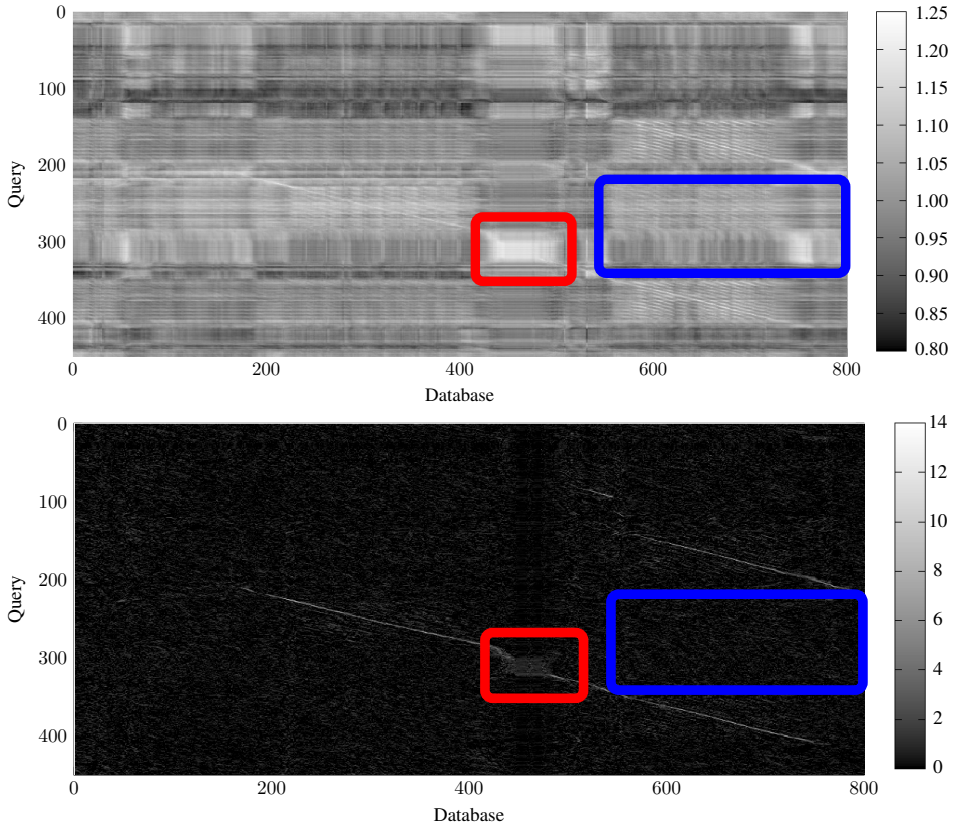


Figure 5.3: *Top:* This figure shows the similarity matrix using HOG-descriptors. The scores are not distinctive and we can notice noise scattered throughout the matrix.
Bottom: Whitening transformation yields highly distinctive scores, which results in removing most of the noise. This has a great impact on reducing the number of false positives in the similarity matrix.

For completeness, we briefly discuss the steps of calculating and applying the ZCA-whitening transform to our similarity matrix $\mathbf{S}$ with the raw *cosine*-similarities. First, we center every column of $\mathbf{S}$ by subtracting the mean for each column.

$$\mathbf{S} \mapsto \hat{\mathbf{S}} = (\mathbf{S}_{\mathrm{ij}} - \mu_{\mathrm{j}})_{\substack{1 \le i \le n_Q \\ 1 \le j \le n_D}}, \quad \mu_{\mathrm{j}} = \frac{1}{\mathrm{N}} \sum_{1 \le \mathrm{i} \le \mathrm{n_Q}} \mathbf{S}_{\mathrm{ij}} \tag{5.11}$$

Then we calculate the covariance matrix of $\hat{\mathbf{S}}$ for the rows, which results in a $n_D \times n_D$ matrix.

$$\mathbf{\Sigma} = \mathrm{Cov}(\hat{\mathbf{S}}) = \mathrm{E}[\hat{\mathbf{S}}^{\mathrm{T}} \hat{\mathbf{S}}] \tag{5.12}$$

This covariance matrix $\mathbf{\Sigma}$ is symmetric and semi-positive definite, therefore we can decompose the matrix, using the eigenvalue decomposition, into a product of a diagonal matrix $\mathbf{\Lambda}$ and a unitary square matrix $\mathbf{U}$ of size $n_D \times n_D$.

$$\mathbf{\Sigma} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^{\mathrm{T}} \tag{5.13}$$

The columns of the matrix $\mathbf{U}$ represent the eigenvectors of $\mathbf{\Sigma}$ and the diagonal elements of $\mathbf{\Lambda}$ are the corresponding eigenvalues. We obtain a diagonal covariance matrix for the

observations when we rotate the data using the matrix $\mathbf{U}$, resulting in $\hat{\mathbf{S}}\mathbf{U}$. Now, each dimension has a covariance corresponding to its eigenvalue. We get an identity covariance matrix by dividing the square root of $\mathbf{\Lambda}$. To ensure that the matrix is invertible, we add a small $\epsilon > 0$ to the diagonal elements of the diagonal matrix $\mathbf{\Lambda}$. Thus, the whitening transformation, also known as ZCA-Whitening, on our similarity matrix is given by

$$\mathbf{S} \mapsto \tilde{\mathbf{S}} = \hat{\mathbf{S}}\mathbf{U}\sqrt{(\mathbf{\Lambda} + \epsilon\mathbf{I})^{-1}}\mathbf{U}^{\mathrm{T}}. \tag{5.14}$$

5.5 Experiments

We analyzed the performance of the approach by performing an extensive set of experiments. Thereby, we provide comparison to state-of-the-art methods, namely: OpenSeqSLAM, an open-source implementation of the approach presented by Milford and Wyeth [77]; the network flow approach of Naseer et al. [82], which we will refer to as NFx with x being the number of flows used; and a baseline that takes the best match from the raw similarity matrix, which we will refer to as Raw_BM. OpenSeqSLAM uses a pixel-wise similarity measure on down-sampled and patch-normalized images to compute the similarity matrix. Matches are then inferred from searching the similarity matrix for linear patterns. However, OpenSeqSLAM is sensitive to changes of the viewpoint as reported by Sünderhauf et al. [116], likely originating from the pixel-wise similarity measure. In the original publication [77] the authors state that images were manually cropped to match viewpoints. For this evaluation we did no manual pre-processing on the images. The network flow approach NFx is based on similarity scores, similar to the approach we presented in this chapter. It models the problem as a graph and searches for the path in the graph that has the highest accumulated score. The main drawback of this method is that due to the restricted connections of the graph, transitions can only be to the nearby proximity of the current pose with respect to the order of the database. Moreover, we evaluate the performance of the proposed approach with the ZCA-whitening pre-processing, which we call *OursZCA*, and without, which we will refer to as *Ours*. To show the full potentials of our approach, we chose challenging datasets for the evaluation, which include extreme perceptual changes, *e.g.,* illumination, seasonal changes and different viewpoints, see Figure (5.4) and Figure (5.5), as well as trajectories that exhibit different characteristics. For a quantification of the results, we use measures of precision and recall, where we count a match as true positive if and only if the distance in the ground truth was less than six frames in the positive or negative direction of the database sequence. Considering a speed of $10m/s$ and a frame rate of $4\mathrm{Hz}$, this would result in matching bounds of $\pm 15m$. Within the experiments, we consider a single camera mounted on the robot and therefore we do not count images taken at the same spot but from the opposite direction as a true-positive match, since none of the considered methods handle this case. We use precision-recall curves to visualize the peformance of the approaches, since they are commonly used and intuitively to understand. To order precision-recall curves for performance, we use the maximum F_1-Score, which is the harmonic mean of precision and recall. The general definition of the F_β-Score for $\beta > 0$ is given by:

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}. \tag{5.15}$$

Figure 5.4: This figure shows true positive pairs of images, matched with our approach. For each pair: the left image is the query from the localization run and the right image is from the database. Our localization method is robust to: seasonal changes, partial occlusions and other sensor specific errors like sun glare.

Figure 5.5: These images show the successfully matched locations from the VPRiCE dataset using our current approach. Our approach handles large illumination changes, noisy and low resolution images captured across day and night, seasonal variations and adverse effects, like sun glare.

Setting $\beta = 1$ means that precision and recall is equally weighted, which is a common choice.

We present the quantitative results of the best F_1-Scores in Table 5.1. In Figure (5.6), Figure (5.7) and Figure (5.8) the quantitative precision-recall curves are depicted in the left column, the qualitative results of the path, corresponding to the best F_1-Score, of *Ours* and *OursZCA*, along with the ground truth, are drawn in the right column.

In order to structure the discussion, we divide the datasets into three parts, based on the characteristics of their trajectory in the database. The first part consists of the data-sets Seq 2, Seq 1 and Seq 5, which have a rather scattered ground truth trajectory, as they consist of many small chunks and exhibit many jumps forward and backward in the database. The results for these datasets are discussed in Section (5.5.1). The second part consists of the data-sets Seq 4, Seq 3 and VPRiCE, which have a ground truth trajectory that consists of a few rather long connected sequences. The results of those trajectories are discussed in Section (5.5.2). We discuss the NewCollege dataset separately in Section (5.5.3), as it is the only dataset in which database and query are identical and it exhibits a combination of the characteristics mentioned above: the rather trivial matches at the diagonal as a large connected sequence and many small sub-sequences, which stem from the loop closures during the run.

Our approach relies on several parameters, and we keep most of them fixed for the evaluation. First of all, we have the parameters for the transition model, $f,\ b,\ c_f,\ c_s$ and c_b (see Section (5.1.3)). For all experiments, we set those parameters to $f = 5,\ b = 3,\ c_f = 500,\ c_s = 100,\ c_b = 50$. In general, the performance of our approach depends on those parameters, because of the dependency of the Bayes filter on the correct transition model, and our approach uses an approximation due to the lack of additional information. Nevertheless, all of our experiments are done with the same parameter set for the transition model. Moreover, there are the parameters of the sequence filtering, $g_c,\ g_r,\ l_s$ and p_{min} (see Section (5.3)). The performance of our approach is mostly affected by the choice of l_s and p_{min}, while we keep the same values $g_c = g_r = 5$ for all datasets. For the experiments announced above, we vary the value of p_{min} to compute the precision recall curves and use a trajectory dependent, manually set value for l_s. To complete the evaluation of our approach, we discuss the variation

Method	Seq 2	Seq 1	Seq 5	Seq 4	Seq 3	VPRiCE	NewCollege
Raw_BM	0.38	0.48	0.43	0.51	0.55	0.53	0.68
OpenSeqSLAM	-	-	-	0.26	0.36	0.62	0.68
NF1	0.027	0.26	0.3	0.57	0.71	*0.66*	0.68
NF3/NF2 *	0.08	0.35	0.31	*0.70**	0.64*	-	0.79
Ours	*0.49*	*0.67*	*0.51*	0.58	*0.83*	0.55	*0.92*
OursZCA	**0.69**	**0.72**	**0.82**	**0.77**	**0.88**	**0.85**	**0.95**

Table 5.1: F_1-Scores over all datasets. Bold numbers are those with the best performance for the dataset, the italicized numbers show which approach performed best before our extension. For NF3/NF2* the starred numbers (*) indicate that NF2 was used, otherwise it is NF3.

of l_s in Section (5.5.4).

5.5.1 Scattered Trajectories

In this section, we discuss the results of the datasets Seq 2, Seq 1 and Seq 5, which we denote as the scattered trajectories, due to the characteristic structure of the ground truth trajectory. The datasets have been recorded at different seasons in Freiburg, Germany. For these datasets, the database as well as the query contain revisits of the same place, which stem from driving around blocks and taking multiple turns. Moreover, these trajectories exhibit several jumps forth and back in the database, since the car traverses the streets in a different order. In these kinds of datasets, the proposed approach shows the most performance improvement in our comparison, since neither NFx nor OpenSeqSLAM are able to handle such cases. Both approaches suffer from the limited transition model, which assumes that the trajectories visit the places in the same order. Thereby, these approaches can not deal with multiple queries matching the same database entry (query loop), and multiple matches for a single query (database loop) are not sufficiently handled, as well. Instead, the flexible motion model of the approach that we propose in this chapter can handle both cases well, which allows for arbitrary transitions between places in the database for consecutive query images. Consequently, our approach achieves good results for all of the three datasets, while NFx and OpenSeqSLAM do not give reasonable results at all; they are even below the baseline of Raw_BM for all datasets. At the same time, we see that *OursZCA* outperforms *Ours* for all datasets. This improvement is substantial for Seq 2 and Seq 5, where *OursZCA* achieves a precision of more than 0.8 for the same recall as the highest recall value achieved by *Ours*.

The scattered and complex trajectory of Seq 2 is depicted in Figure (5.6), where we show the ground-truth path with an overlay of the results from *OursZCA* and *Ours*. We can qualitatively observe what the precision-recall curve shows, *OursZCA* achieves a higher recall and produces fewer false-positives at the same time. For example, *OursZCA* matches the two short parts for the queries from 0 to 100 and only shares one of the false positive sequences of *Ours*. Referring to Table 5.1, the improvement of *OursZCA* (0.69) over *Ours* (0.49) for the best F_1-Score is 40.8%.

The ZCA-whitening of the data matrix has a huge effect on the results of Seq 5 as well, which is clearly noticeable in Figure (5.6). The precision-recall curve is way above the curve

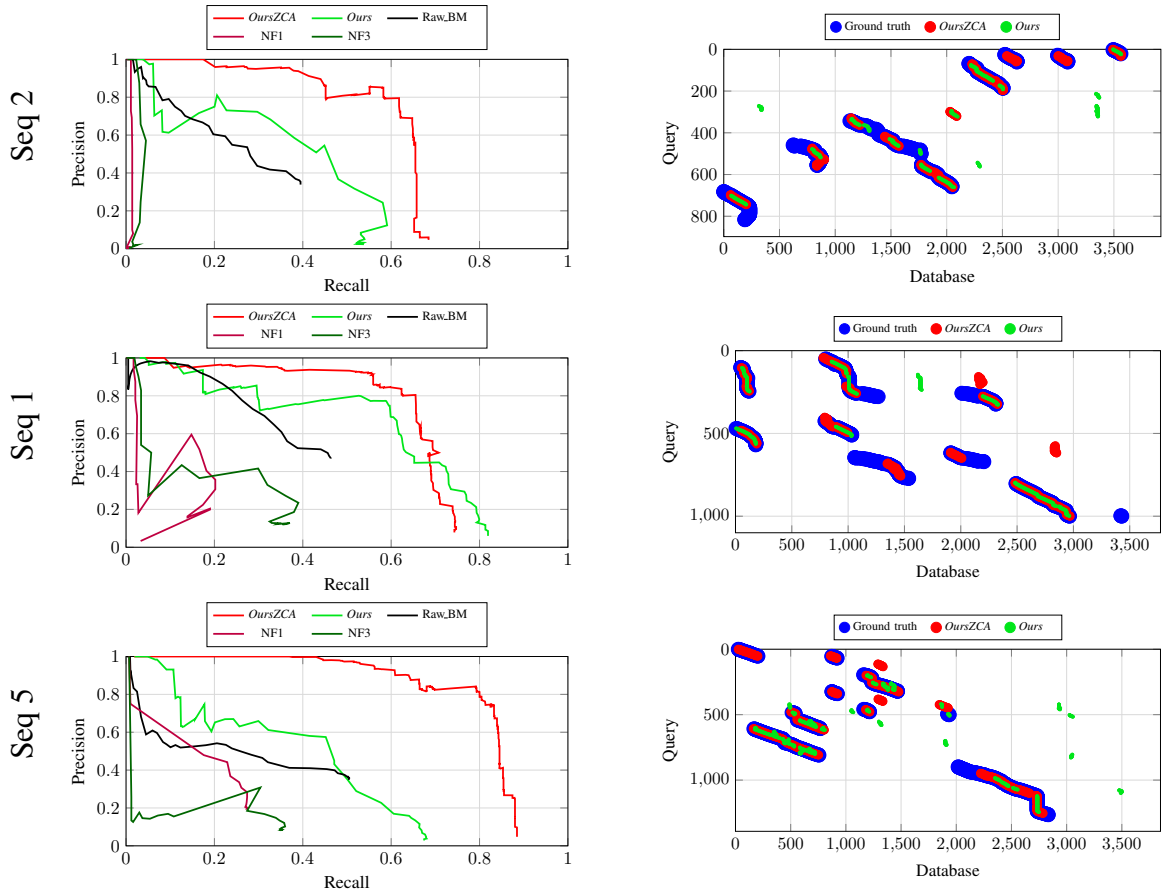


Figure 5.6: Experimental results. *Left:* Precision-Recall curves. *Right:* Paths for the best F_1-Score and ground-truth.

of *Ours*, and *OursZCA* achieves a much higher recall. The paths produced by the approaches reflect this fact. While *OursZCA* matches most of the trajectory with only a few false positives, *Ours* shows false positives all over the dataset and even the correctly matched parts of the trajectory frequently end up with some false positives. This results in an improvement of the best F_1-Score of *OursZCA* (0.82) over *Ours* (0.51) for the best F_1-Score is 60.8%.

The results of Seq 1 show a similar performance of *OursZCA* and *Ours*, as one can see from the similar shape of the respective precision-recall curves. When we take a look at the resulting paths, one can see that the ZCA-Whitening of the data matrix shades the matching scores for the stoppage at the beginning of the trajectory. Moreover, in this dataset *OursZCA* exhibits two false positive sequences while *Ours* returns only one. However, especially in the middle part of the trajectory *OursZCA* benefits from true-positive matches that are missed by *Ours*. This still results in an improvement in terms of the best F_1-Score of 7.5% when we apply the ZCA-whitening to the score matrix, see Table 5.1.

Altogether, these experiments show that the presented approach performs well on complex trajectories that can not be processed by other state-of-the-art methods. Moreover, the improvement of applying the ZCA-whitening transform to the similarity matrix is validated for all of the datasets considered in this section.

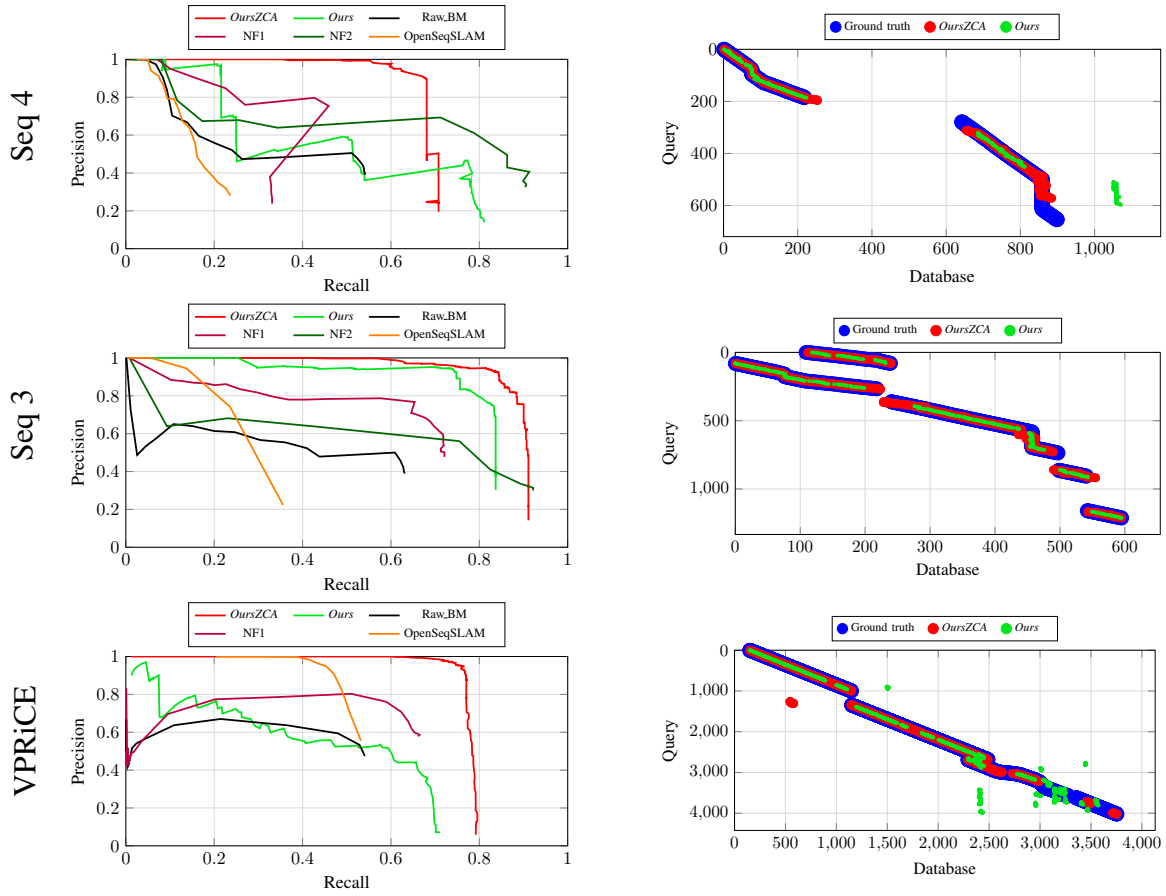


Figure 5.7: Experimental results. *Left:* Precision-Recall curves. *Right:* Paths for the best F_1-Score and ground-truth.

5.5.2 Connected Trajectories

In this section, we discuss the results of datasets Seq 4, Seq 3 and VPRiCE, which exhibit few and rather long, connected fragments on the ground truth matching between the query and the database, see Figure (5.7). These datasets exhibit fewer visits to new palaces (gaps) and almost no loops, whether in the database or in the query, and the ground truth is almost monotonic with respect to the order of the sequences. Therefore, these trajectories favor the comparison approaches OpenSeqSLAM and NFx.

For the first dataset, which is Seq 3, we observe that the precision-recall performance of *Ours* and *OursZCA* is quiet similar, see Figure (5.7). The ZCA-whitening exhibits similar precision values along with a higher recall. NF1 results in a precision of around 0.8 and a maximum recall of about 0.7. For this method, the recall suffers from the jump at the beginning of the trajectory, which can not be handled with a single flow. When we use an additional flow, NF2 results in higher recall values but shows a lower precision due to false positives gathered by the second flow. This results in maximum F_1-Scores of 0.71 for NF1, 0.83 for *Ours* and 0.88 for *OursZCA*.

The results for Seq 4 and VPRiCE again show that large gains are possible from using the ZCA-Whitening transform. In Seq 4, the results show that NF2 outperforms *Ours*, since in this case *Ours* exhibits too many false positives, resulting in a low precision. For the network

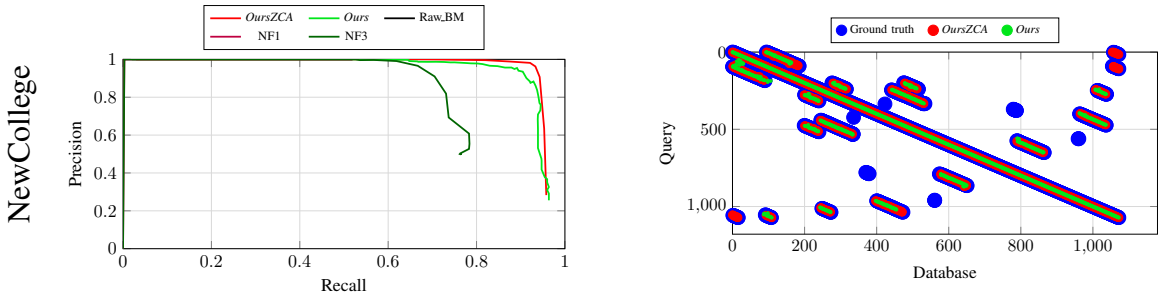


Figure 5.8: Experimental results. *Left:* Precision-Recall curves. *Right:* Paths for the best F_1-Score and ground-truth.

flow approach we need to use two flows, since the gap between the two subsequences is too large and can not be bridged by the rigid transition model using a single flow only. However, applying the ZCA-whitening transform results in a huge performance gain with high precision and a moderately lower recall. Therefore, *OursZCA* again outperforms NF2 and performs best of all of the compared methods. In terms of the best F_1-Scores this results in 0.58 for *Ours*, 0.70 for NF2 and 0.77 for *OursZCA*.

We observe similar results for the last dataset considered in this class of trajectories, which is the VPRiCE dataset. This dataset is composed of several sequential datasets, which are taken from different places around the world. Even though the dataset does not entirely satisfy our assumption of a full sequential dataset, this assumption *is* met for each subdataset used to compose it. For an impression of the different challenges in this dataset, see Figure (5.5). In this case, *Ours* suffers from many false positives in the lower right part, as shown in Figure (5.7). Without applying the ZCA-whitening preprocessing step, our approach is again outperformed by NF1, although it is important to mention that the network flow approach does not achieve reasonable results if multiple flows are used. For this dataset, OpenSeqSLAM achieves reasonable results as well, mostly due to the first part of this dataset, which is collected by a train and therefore provides a high viewpoint stability. However, *OursZCA* performs best among all of the other methods on this dataset again and achieves a best F_1-Score of 0.85, which is substantially higher than the next best results of 0.66 achieved by NF1.

5.5.3 NewCollege

The last dataset covers a typical single run mapping scenario with identical database and query set, and is well known in the community as the NewCollege [23] dataset. Nevertheless, we do not treat it differently from other datasets, so we expect to detect the diagonal as well as the loop closures of the dataset. In this case Raw_BM, NF1 and OpenSeqSLAM will only match the diagonal, since we compare identical images there, which necessarily leads to a maximum similarity score. Both versions of the approach that we have presented in this chapter, *Ours* and *OursZCA*, preform really well on this dataset, see Figure (5.8). Nevertheless, consistent with the former results, *OursZCA* achieves the best F_1-Score of 0.95 and outperforms *Ours* by 3.3%, see Table 5.1. This high measure of performance stems mainly from the fact that the similarity scores for this specific dataset are much more distinctive. Accordingly, the network flow approach with multiple flows also performs well on this dataset where NF3 achieves a

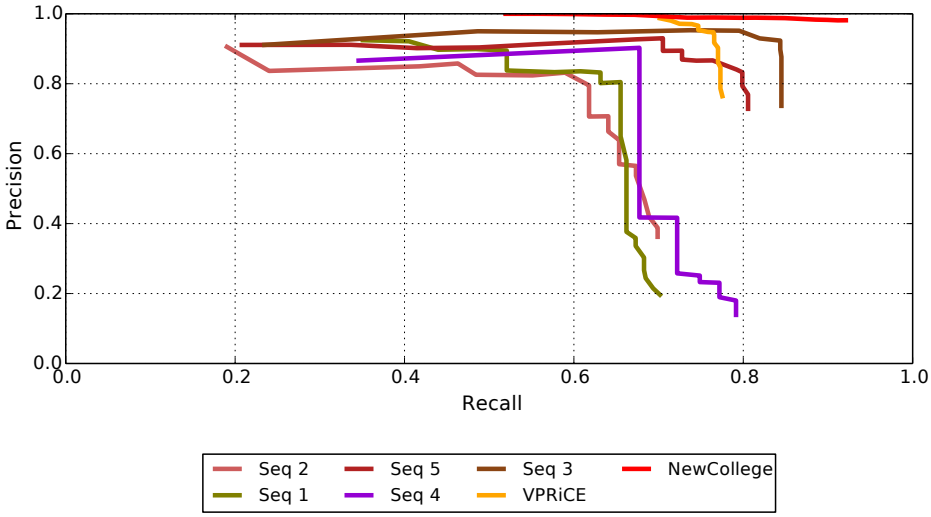


Figure 5.9: Precision-Recall curves for all datasets with *OursZCA* varying the minimum sequence length parameter l_s. With increasing l_s the curves go from bottom right to top left, since we filter false positives as well as true positives, therefore lowering the recall but increasing precision.

F_1-Score of 0.79. The use of three flows is obvious, due to the symmetry of the dataset.

5.5.4 Parameter Discussion

We complete the experimental evaluation of our approach by analyzing the influence of the minimum sequence length parameter of the sequential filtering l_s, see Section (5.3). Therefore, we use the p_{min} value of the best F_1-Scores obtained by the evaluation in the previous section and compute the precision recall curves for the varying values of l_s. One would expect that the values for l_s should take the shape of the trajectory into account, since for the scattered trajectories we may not retrieve sequences as long as those for the connected datasets. The trade-off is between losing true-positive matches and eliminating false-positive sequences. In fact, the values used for the scattered trajectories in Section (5.5.1) range form 16 to 27 and for the connected datasets in Section (5.5.2) range from 20 to 47.

The resulting precision-recall curves for varying l_s are depicted in Figure (5.9), where each dataset is drawn in a different color. Since false positive sequences are usually rather short, the curves run from bottom right to upper left, with increasing l_s. As expected, the scattered datasets are more sensitive to the recall when increasing l_s, while the connected datasets show a large step in precision, because we remove mainly false positives for a wide range of l_s. The VPRiCE dataset shows a different behavior, which is due to the challenging images in the last third of the dataset and the long connected first part of the dataset.

5.6 Related Work

Vision based systems have gained high popularity in the field of autonomous navigation and monocular camera-based approaches, in particular, are attractive due to the comparably

low sensor costs. Many researchers have addressed the problem of place recognition in recent years [18, 24, 26, 38]. Many approaches for visual place recognition assume that the appearance of places remains similar between the mapping and the localization run [11, 24]. The problem of reliable and robust localization becomes harder when the environment undergoes substantial perceptual changes between the mapping and localization task. These changes can be caused by seasonal changes, illumination occlusion and view point variations. So far, keypoint-based feature descriptor matching approaches have been heavily employed to visual localization problems; the most prominent features in the past have been probably SIFT [66] and SURF [10]. The feature and keypoint detection are focused on viewpoint and scale invariance and they are not robust against seasonal changes, as claimed by Naseer et al. [82], since the keypoints are no longer stable. Valgren and Lilienthal [127] agree that local feature matching alone is not sufficient for localization across seasons. However, they could show that high resolution panoramic images, combined with epipolar constraints, substantially improve the localization accuracy.

Another source of errors, which is tackled by several works, is changes of illumination between the matched images, since the features are usually not robust against them. Ranganathan et al. [97] tackle this problem by learning the matching function, and explicitly include various lightning conditions in the training data. Other authors propose transforming the images to an illumination invariant chromacity space [68, 74]. The approach of Glover et al. [38] combines RatSLAM [76] and FAB-MAP [24] to learn consistent maps over different times of a day. The combination of both systems exhibits fewer false positive matches for different conditions due to complimentary effects of both approaches. However, the approach still suffers from unstable features and is sensitive to longer detours. The approach of Milford and Wyeth [77] exploits sequential information, in combination with a pixel-based whole-image similarity, to localize across day and night. The approach achieves substantial improvements over keypoint-based approaches, even if the appearance of the environment undergoes extreme perceptual changes. The main limitations of the approach are that it assumes pre-aligned viewpoints across the two image sets and a linear trajectory. In the context of robot navigation, these assumptions are hardly met. Naseer et al. [82] model the matching problem as a network-flow optimization, using the inverse of the *cosine*-similarity of whole image HOG-descriptors, as in our approach, as edge costs to match sequences across seasons. This approach can handle non-linear trajectories, loops and detours. Although the approach can handle non-linear trajectories, it is limited in transitions due to a restricted graph connectivity. The approach presented in this chapter outperforms both approaches, as presented in the previous section. Additional sensor data, *e.g.,* range measurements or GPS, can aid the robustness of visual localization across seasons. Vysotska et al. [129] proposed an extension to the approach of [82], which utilizes GPS measurements to reduce the complexity of the constructed graph, gaining computational speed and accuracy. Badino et al. [8] use a Bayesian framework and combine range and visual data for robust localization across seasons. In contrast, our approach does not need additional sensor information.

Another approach to tackle cross season visual localization is to learn how the appearance changes over time. An approach to predict the appearance of an image for a new season was proposed by Neubert et al. [87]. They built vocabularies of superpixels learned over a long time span and use pixel-aligned training images for this purpose. Churchill and Newman [22] suggest an experience-based navigation framework, storing each image retrieved at a location, as one experience. The query image is then compared to all experiences. This approach can

handle severe perceptual changes but needs visual odometry. The approach of Carlevaris-Bianco and Eustice [19] analyses time-lapse videos and successfully learns feature point descriptors that are robust to changes. The feature descriptors map the corresponding image patches to a lower-dimensional space where the Euclidean distance provides a discriminable measure for matching images. McManus et al. [75] suggest learning regions of images that are stable over time. Instead of matching keypoint descriptors, they match the stable visual elements for robust long-term localization. All of these methods have in common the fact that they need training data collected over a long time span, which is not necessary for our approach.

More recently, the usage of Deep Convolutional Neural Networks (DCNN) has gained popularity. These networks learn the relevant features for a specific problem and have shown great robustness in several areas. Sünderhauf et al. [117] and Neubert and Protzel [86] show that DCNN-based descriptors, combined with region proposals, allow for image-pair matching across viewpoint and appearance changes. In an extension of their previous work, Naseer et al. [83] uses DCNN-based descriptors for the network-flow approach [82] and robot odometry for a visual SLAM approach, which aims to associate and join trajectories visited at different times. However, deep networks have a high computation power demand and, usually, GPUs are involved to speed up the computations, while our approach can be efficiently implemented on CPUs. Nevertheless, using DCNN-features for the proposed method should result in a gain of performance, as well.

5.7 Conclusion and Future Work

In this chapter, we presented an approach to robust and purely vision-based localization, facing substantial perceptual changes of the environment. We employ a Bayesian filter framework and measure the similarities of images based on whole-image HOG-based descriptors with the *cosine*-similarity. The performance of our approach is substantially improved by applying the ZCA-whitening transform to the data similarity matrix. On a wide range of datasets, the proposed approach outperforms state-of-the-art approaches, and it can handle more complex trajectories due to a probabilistic formulation of the transition model. Moreover, the approach is computationally lightweight and easy to implement.

Future Work

Possibilities for future work include a deeper analysis of the parameters of our approach. The derivation of the parameters from an analysis of the datasets, without manual intervention, would be a great advancement. Moreover, from our experiences during the development of the approach, it seems likely that different mounting positions of the camera may have an impact on the performance of the approach and possibly on visual localization approaches, in general. An analysis with respect to this could lead to interesting and useful conclusions.

Chapter 6

Semi-Supervised Learning of Traversability Models

A crucial pre-requisite for truly autonomous systems is the knowledge about which parts of the world can be traversed without causing harm to itself, the environment or, most importantly, to humans. On the other hand, traversability analysis also guarantees the mobility of the robot. The trade-off between safety and mobility constitutes the core of this important problem. In this chapter, we present an approach that infers a classification for traversability from demonstrations of a human. The only necessary capability of that human is that they are able to provide positive only examples of traversable ground. Our approach aims to learn a classifier for traversability from these positive-only samples. To achieve this, we extract features from 3d-lidar data, which expresses geometric and perceptual-based measures, to learn appropriate distributions for the traversability analysis. In extensive experiments, we demonstrate the capabilities of our approach, in terms of accuracy and usability, for different types of robots and discuss the expressiveness of the geometric and perceptual-based measures from our feature vector.

In the previous chapters, we discussed approaches to learn and use a map, *e.g.,* for localization, which is seen as one of the most important key pre-requisites for autonomously navigating robots. Decades ago, when research on autonomous navigation first started, researchers usually considered plane indoor environments. In combination with either 2d-lidar or ultrasonic sensors, solving the SLAM problem was sufficient for navigation. This was due to the fact that the commonly used occupancy grid map divided the world into free space and obstacles, which solves the problem of traversability analysis for such environments, as well. However, progress has been made and robots have started to conquer outdoor environments for which this assumption does not hold anymore and traversability analysis needs to be treated separately.

To discriminate traversable from non-traversable ground in unstructured and complex outdoor environments, a robot needs sensors that can perceive a dense model of the world. We can not use a single 2d slice, which is sufficient for planar environments, to determine whether or not an obstacle is present, since obstacles may exhibit various characteristics, *e.g.,* negative obstacles or slopes, which may appear as false positive obstacle detection. In these cases, 3d

Figure 6.1: Two mobile robot outdoor platforms with different capabilities and different fields of applications. On the top row Viona, is a large outdoor robot with great ground clearance and high motor power. On the bottom row Obelix, a robot that should only drive on the streets and walkways. The right column shows the resulting traversability map of our approach for the same area: the green depicts traversable ground and the red area depicts untraversable ground. Viona can traverse the grass in the upper right part, whereas Obelix cannot, in which case the grass is not traversable.

range data is necessary, as provided, *e.g.,* by stereo-cameras, radar or 3d-laser scanners, or a fusion of different sensors. The approach that we present in this chapter works on 3d-lidar data, not only using the purely spatial and geometric information but also including remission values to add a perceptual component to the process. The definition of traversability highly depends on the individual type of robot that is used in the application, since they can exhibit quite different capabilities in regards to: ground clearance, motor power, stability, and whether it is equipped with wheels or chains. For an example of different robot types, see Figure (6.1). Common approaches usually put a lot of effort into designing the model for traversability analysis, taking the platform and sensor specific characteristics into account. This is often a time consuming and costly process, which involves a real expert in that domain. It would be much easier if one could just manually operate the robot in that environment, to train the traversability analysis using the traversed path as positive examples for traversable ground. However, to make use of this data, we need to deal with the problem that the learning algorithm only receives the information of the traversed path during the training. This means that the training set contains only incomplete positive labels and, hopefully, no negative examples. Within the proposed framework, we will compare the performance of two learning algorithms, which are adapted from other domains and designed for this kind of problem, with respect to the quality of the traversability analysis.

The remainder of this chapter is organized as follows: first, we present the internal basic structures of the approach in Section (6.1). Afterwards, in Section (6.2), we describe the features, which we extract from the sensor data. With these features, we train the classifiers for traversability in Section (6.3). Finally, we evaluate the presented approach in extensive experiments for qualitative and quantitative measures in Section (6.4), using real-world data.

6.1 Basic Structure

In this chapter, we are particularly interested in traversability as a local and static characteristic of the environment. Furthermore, we assume that the mobile robot is equipped with a 3d-lidar sensor that additionally provides remission values, which is an important attribute, as we will see, later. For map representation and organization of the sensor data, we use a 2d grid structure $\mathcal{G}$ with a resolution $r \in \mathbb{R}_{>0}$, where each cell can temporarily store the corresponding raw sensor measurements and an arbitrary number of feature vectors, which we compute from the raw sensor data. We determine the correspondence of a measurement from the 3d-lidar with a cell of the grid by the 2d-projection of the respective 3d point.

$$\Pi_{\mathcal{G}} : \mathbb{R}^3 \longrightarrow \mathbb{Z}^2 \tag{6.1}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \longmapsto \begin{pmatrix} \lfloor \frac{x}{r} \rfloor \\ \lfloor \frac{y}{r} \rfloor \end{pmatrix} \tag{6.2}$$

A common limitation of 3d-lidar measurements is that they are locally sparse, and therefore it is hard to reason about the structure of a cell from a single scan. To overcome this, we collect measurements while the robot is moving, until it covers a distance of $d_P \in \mathbb{R}_{>0}$. We then compute and add a new feature vector for each cell. Details of the feature vector will follow in the next section.

As we are interested in the traversability characteristics of the environment, the goal of our approach is to estimate the probability that a cell is traversable given the observed features, $P(trav \mid f_1, \ldots, f_n)$. To achieve this, we aim to learn the distribution $P(f \mid state)$ with $state \in \{trav, \neg trav\}$, which we model as a multinomial distribution $P(\cdot \mid state) \sim Multinomial$. Since this is a discrete distribution, we assume that the set of features forms a vocabulary $\mathcal{V}$. To learn the parameters of the distribution, we need to estimate the frequencies for each $f \in \mathcal{V}$. Without loss of generality, we use the local grid maps that are only used for a distance of $d_M \in \mathbb{R}_{>0}$ and withdraw them afterwards, which circumvents accumulated registration errors, and for navigation, the robot first needs to know the traversability properties of its nearby vicinity.

6.2 Feature Design

Calculating features from raw sensor measurements is a common way of sensor data interpretation. The advantage of using features for our approach is that we can calculate them from a varying number of raw sensor measurements and receive a vector of a fixed dimension. Thereby, it is important that the features are designed to represent the world, which is perceived with the sensor, in an expressive way for the intended task. For our approach, this means

that they need to be suitable for distinguishing between obstacles and traversable ground, for different mobile robots. They have to distinguish flat solid ground from: moderate steps, different kinds of vegetation and solid obstacles of certain heights, since it strongly depends on the type of robot whether these are traversable or not. To achieve this, we combine the geometric information, *e.g.,* the height difference, and the remission values in our features, as they are useful to distinguish different materials, *e.g.,* grass and concrete.

For our approach, we use feature vectors that contain the following measures:

- The absolute maximum difference in the z-coordinate
- The mean of the remission values
- The variance of the remission values
- The roughness of the cell
- The slope

Each dimension should help to distinguish different types of terrain or relate to traversability constraints of the robot. The maximum height difference and the slope, reflect the ground-clearance of the robot as well as the motor power. The remission values and the roughness are expected to help when distinguishing terrain types, *e.g.,* concrete and vegetation.

Since the calculation of the first three dimensions is straight forward, we birefly explain the calculation of the latter two, which are based on the eigenvalues and the respective eigenvectors of the covariance matrix, which is calculated from the points that are registered to the same cell. The magnitude of the smallest eigenvalue is used as the roughness measure. The eigenvector that refers to the smallest eigenvalue is a least squares estimate of the normal vector, when estimating the parameters of a plane. Accordingly, the slope is the angle between this eigenvector and the vector of gravity. To ensure that these values are well defined, we ignore cells that contain less than five points. As our desired model contains a discrete distribution, as explained in the previous section, we discretize the features using bins of fixed increments for each dimension.

6.3 The Learning Problem

Providing a fairly easy method for acquiring training data is one of the goals in this chapter. Specifically, it should be possible that an arbitrary person, who has not been trained to design traversability models for mobile robots, could use it. To achieve this, we acquire the training data for our approach from demonstrations, in which the user manually operates the robot. From these demonstrations, we determine the traversed cells, which are those that intersect with the footprint of the robot, and assign a positive label to each of the features that were observed in these cells. With this method, the only thing that the user needs to know is that the robot should drive on all kinds of terrain and that the robot does not crash.

Using this process for training data generation has the advantage that it is fairly easy to execute but also has the drawback that we get scarce positive examples relative to a massive amount of unlabeled data, from which to learn. Fortunately, this kind of data is a common problem in other domains, *e.g.,* text classification or biological approaches like

Figure 6.2: Trajectory of Viona on the forest track that was used for the experiments on an aerial image.

protein categorization, and we can adapt existing methods for our approach. We use and compare two strategies in order to learn a classifier from this kind of training data, one called *Positive Naive Bayes* (PNB) introduced by Denis et al. [27] and *Learning Classifiers from Only Positive and Unlabeled Data* (POS) by Elkan and Noto [29]. The former was developed for text classification, for which this kind of data is very common, and also assumes a multinomial distribution of the words, given the document class. The latter is a more general approach, which can deal with a variety of distributions.

In the forthcoming sections, we discuss each learning method. As we have either positive or unlabeled training samples of the features, we denote the set of positive training samples as $\mathcal{P}$ and the set of unlabeled training samples as $\mathcal{U}$.

6.3.1 Positive Naive Bayes

Common applications for document classification are, *e.g.,* spam filtering or to determine whether or not a document is relevant for a user. Naturally, these cases have a large database of unlabeled examples and only a few positive examples available. The *Positive Naive Bayes Classifier*, as introduced by Denis et al. [27], aims to learn a multinomial distribution of the words, given the document class. In our case, the words are the discretized features f and

Figure 6.3: Aerial image with the training (blue) and evaluation (red) trajectory of Obelix on the Campus.

the documents are the cells of the grid. The maximum likelihood estimate for a multinomial distribution is estimated by the occurrence frequency in the training set. Therefore, we need the counting function C

$$C : \mathcal{V} \times 2^{\mathcal{G}} \to \mathbb{N} \tag{6.3}$$

$$C(f, \mathcal{S}) := \sum_{c \in \mathcal{S}} \sum_{f_c \in c} \mathbb{1}_f(f_c) \tag{6.4}$$

where $2^{\mathcal{G}}$ is the powerset of the grid cells. The sum takes each cell contained in the set $\mathcal{S}$ and counts the occurrences of the feature f with multiplicities. When we apply C to a set of cells $\mathcal{S}$ it counts the number of features contained in the cells of $\mathcal{S}$ with multiplicities.

$$C : 2^{\mathcal{G}} \to \mathbb{N} \tag{6.5}$$

$$C(\mathcal{S}) := \sum_{f \in \mathcal{V}} C(f, \mathcal{S}) \tag{6.6}$$

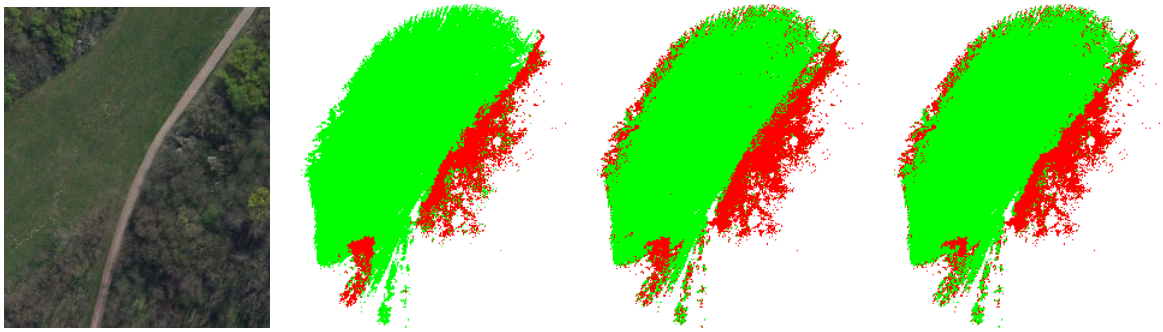
To estimate the frequencies of features in the case of traversable ground, we use the positive samples in our training set.

$$P(f \mid trav) = \frac{\alpha_p + C(f, \mathcal{P})}{\alpha_p |\mathcal{V}| + C(\mathcal{P})} \tag{6.7}$$

with the smoothing parameter α_p. The estimate of this frequency is the same as one would use in the case of fully labeled training data. However, in our case, the positive samples are not complete and there are no negative samples in the training set. Therefore, we aim to infer the frequencies for the negative class from the positive class estimate, and the prior for the positive class. This results in a virtual counting function C_N:

$$C_N : \mathcal{V} \to \mathbb{N}$$

$$C_N(f) := \max\left(0, \lfloor C(f, \mathcal{U}) - P(f|trav) P(trav) C(\mathcal{U}) \rfloor\right) \tag{6.8}$$



a) Aerial image b) Ground truth c) PNB-based classifier d) POS-based classifier

Figure 6.4: Traversability map from the forest run with Viona using our approach. From left to right: aerial image of the scene (a), ground truth labeled map (b), which was used for the evaluation, our approach using the PNB-based classifier (c) and our approach using the POS-based classifier (d). The grass on the mid-upper left side is correctly labeled as traversable (green) while the parts of the forest are labeled as obstacles (red). The POS-based classifier has false positives in the lower left and mid right. Both classifiers have problems with the measurements at the border of the map.

Now, we can apply this counting function to estimate the frequencies, given the negative class.

$$P(f \mid \neg trav) = \frac{\alpha_n + \eta C_N(f)}{\alpha_n |\mathcal{V}| + (1 - P(trav))C(\mathcal{U})} \tag{6.9}$$

$$\text{with } \eta = \left(\sum_{f \in \mathcal{V}} \frac{C_N(f)}{(1 - P(trav))C(\mathcal{U})} \right)^{-1} \tag{6.10}$$

where we use the smoothing parameter α_n and an additional normalization term η, which is necessary due to the approximate counting function C_N.

To sum up: we estimate the frequencies for features given traversable ground, directly from the positive training samples. To derive an estimate for the negative class frequencies, for which we do not have labeled samples in the training set, we compute the difference between the occurrences of a feature f in $\mathcal{U}$ and the predicted occurrences of f in traversable cells.

Finally, we need to determine the smoothing parameters $\alpha_{\{p,n\}}$. Let us consider what happens with features of the dictionary $\mathcal{V}$ that have not been observed in the training set. In this case, for the sake of safety, the desired result should satisfy the following inequality.

$$P(f \mid \neg trav) > P(f \mid trav) \tag{6.11}$$

To achieve this, we set $\alpha_p = |\mathcal{V}|^{-1}$ and $\alpha_n = 1$. The classifier then uses the *Naive Bayes* strategy to compute the probability $P(state \mid f_1, \ldots, f_n)$.

$$P(state \mid f_1, \ldots, f_n) \propto P(state) \prod_{i \leq n} P(f \mid state) \tag{6.12}$$

6.3.2 Learning from Positive Only Examples

The classifier proposed by Elkan and Noto [29] follows a different strategy. In their work, they first train a classifier to predict whether or not a feature f will be labeled, which is equivalent to being in the positive class, during training, $P(label \mid f)$. For this purpose, the sets $\mathcal{P}$ and $\mathcal{U}$ provide the full information and we can use standard learning methods. Once the distribution of $P(label \mid f)$ is estimated, Elkan and Noto elaborated a way to transfer this to the original classification problem, in our case $P(trav \mid f)$. For this, they assume that the positive samples are selected with a uniform probability from all positive cases, which they call the *selected completely at random assumption*. Given this assumption, Elkan and Noto could show that there exists a constant $c > 0$ such that

$$P(trav \mid f) = \frac{P(label \mid f)}{c} \tag{6.13}$$

While they provide different ways to estimate c using a validation set, we use the maximum estimate for c, since it should be the most conservative one. Nevertheless, since we have only incomplete data, it is still possible that we get $P(label \mid f) > c$ for some features. To cope with such cases we set

$$P(trav \mid f) = \min\left(\frac{P(label \mid f)}{c}, (1-\epsilon)\right) \tag{6.14}$$

In our approach, we estimate the distribution $P(label \mid f)$ using a regular Naive Bayes approach. To integrate multiple observed features, we use the efficient logit update strategy, utilizing the assumption that traversability is a static property of the environment.

$$\begin{aligned}&\mathrm{logit}(P(trav \mid f_1, \ldots, f_n)) = \\ &\mathrm{logit}(P(trav|f_n)) + \mathrm{logit}(P(trav \mid f_1, \ldots, f_{n-1})) + \mathrm{logit}(P(trav))\end{aligned} \tag{6.15}$$

6.3.3 Terrain Models

Since the learning algorithms can only use the positive data obtained by the trajectory, the unlabeled data may also contain features of different types of terrain that are traversable. The learning algorithms may get confused if we merge all of the data within one distribution. For example, if during the training, we traverse the street for most of the time and traverse grass for only a short time, then the ratio of labeled grass data is very small and, therefore, the learning algorithms can not adapt grass to be traversable. This kind of problem will occur whenever the training set of the terrain types is not perfectly balanced, which we cannot assume as it would induce constraints to the training procedure. Moreover, for the method described in Section (6.3.2) it will also violate the *selected completely at random* assumption, which is crucial for that approach. To overcome this problem, we use a set of different terrain models $\mathcal{M}$. The positive examples of a local map are compared to the existing terrain models using Pearson's χ^2-test, [94], with a significance level of $\alpha = 0.05$. If the test cannot discard the null hypothesis, we merge the data of the local map with the respective model. Otherwise, if the test discards the null hypothesis for all existing models, a new model is added to $\mathcal{M}$. For

the method described in Section (6.3.1), we use a one-vs-all strategy for the final classifier.

$$P(trav \mid f_1 \ldots, f_n) = \max_{m \in \mathcal{M}} P_m(trav \mid f_1, \ldots, f_n) \tag{6.16}$$

For the method described in Section (6.3.2) we need to specify how to compute $P(trav \mid f)$ in the context of terrain models. We use a featurewise one-vs-all strategy here.

$$P(trav \mid f) = \max_{m \in \mathcal{M}} P_m(trav \mid f) \tag{6.17}$$

6.3.4 Training

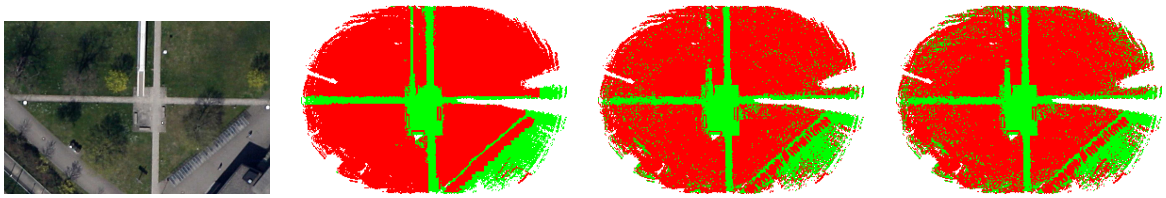
As claimed, the training phase is fairly easy to execute for the user. The robot is operated by a human over all kinds of terrain it can traverse. During this phase, the local maps are given to the learning algorithm; then, the statistic test is computed for the terrain models. Afterwards, the selected model (it may be an existing one or a new one), is merged with the data from the local map and the current distribution of the models are computed. More specifically, for a selected model $m \in \mathcal{M}$ the set of labeled data becomes $\mathcal{P}_m = \mathcal{P}_m \cup \mathcal{P}_l$ and the set of unlabeled data becomes $\mathcal{U}_m = \mathcal{U}_m \cup \mathcal{U}_l$. This sequential structure of our learning strategy also allows the retraining of the robot at any point in time. This might be interesting for scenarios where the robot acts mainly autonomously but is connected to a command center to which it can send requests, if, for example, it cannot find a path to the mission goal.

6.4 Experiments

In the experiments, we used two mobile robots with different capabilities, like in Figure (6.1). One robot is capable of urban as well as outer urban environments, providing: good motor power, high ground clearance and good stability (Viona). The other is only capable of urban environments, with small ground clearance and weak stability (Obelix). On both platforms, we evaluate the quality of the classification using hand labeled ground truth on suitable test tracks, *e.g.,* Figure (6.2) and Figure (6.3). Furthermore, we compare the quality of the classifier when we omit the remission values (NoRe) and when we omit the roughness and slope values (NoRS) of the feature vector, see Section (6.2). For the experiments, we used $d_M = 20m$, $d_P = 0.5m$ and limited the maximum range of our 3d-lidar sensor to $20m$ and the same parameters to discretize the feature vectors were applied to both robots. For the final classification, cells were classified as traversable if and only if $P(trav \mid f_1, \ldots, f_n) > 0.5$. As our approach aims to classify traversability as a static property of the environment, we assume that dynamic obstacles are removed from the scans. In our current implementation, we use an online dynamic obstacle detection approach based on scan differencing. To register the point clouds in our local map, we use the smoothed odometry estimate provided by an *Applanix* Navigation System. The robots are equipped with 3d-lidar sensors from *Velodyne*, providing $360°$ horizontal and $\sim 30°$ vertical fields of view.

6.4.1 Evaluation using Viona

We trained Viona on the Campus, by driving over grass of different heights and with different flowers, dirt, walkways and streets. For the evaluation of our classifier, we used a test track



Aerial image | Ground truth | PNB-based classifier | POS-based classifier

Figure 6.5: Traversability map from the campus run with Obelix using our approach. From left to right: aerial image of the scene; ground truth labeled map, which was used for the evaluation; our approach using the PNB-based classifier; and our approach using the POS-based classifier. Both classifiers produce similar results.

containing dirt roads, Figure (6.2), and on the campus where we traversed walkways as well as grass areas. For the quality measures of the classifiers, we labeled 30 local maps from the forest track and five from the campus track, which is about 10% of the local maps that were created during the run. The ground truth labeling was rather conservative, *i.e.* particularly in the forest environment, the cluttered areas between the trees were hard to classify for each and every cell, when in doubt they were classified as not traversable, since the measure of false positives is slightly more important for traversability analysis. Nevertheless, a false positive was counted if and only if the inspected cell and all eight adjacent cells were classified as positive (traversable).

In this experiment, our approach shows better results, in terms of precision and specificity, when we use the PNB-based classifier than with the POS-based classifier. On the combined data set, with the full feature vector, the PNB-based classifier reaches a precision of 0.992 while the POS-based classifier exhibits a precision of 0.945. The POS-based classifier has problems, especially with the forest data, 0.990 vs. 0.934, while this difference is not that substantial for the campus data set, 0.998 vs. 0.990. It is interesting to note that for the PNB-based classifier, the remission values, with a precision of 0.990 for NoRe, seem not to be as important as the roughness and slope, achieving a precision of 0.953 for NoRS, values. This changes for the POS-based classifier where the precision without remission is worse than without roughness and slope. However, for both classifiers the full feature vector is superior to the pruned feature vectors. The performance, in terms of recall, is antithetic to the precision. In this measure, the POS-based classifier (0.87) is superior to the PNB-based classifier (0.79). Like for precision, this difference is larger for the forest data set than for the campus data set.

The last quality measure we used in our evaluation is the specificity, as depicted at the bottom of Figure (6.6). This measure is of great importance, since it measures the rate of the true negative classifications. A *Type I Error* describes the wrong classification of a negative sample. In the case of traversability analysis, this means missing an obstacle. Here again, already indicated by the precision measure, the PNB-based classifier (0.987) is superior to the POS-based classifier (0.898). While we observed different gaps between the classifiers for precision and recall on the forest and campus data set, measuring the specificity, the difference is roughly the same for both data sets. Note that the results of this experiment do not prove that the PNB method is, in general, superior to the POS method, but for this data set and the way in which we used it.

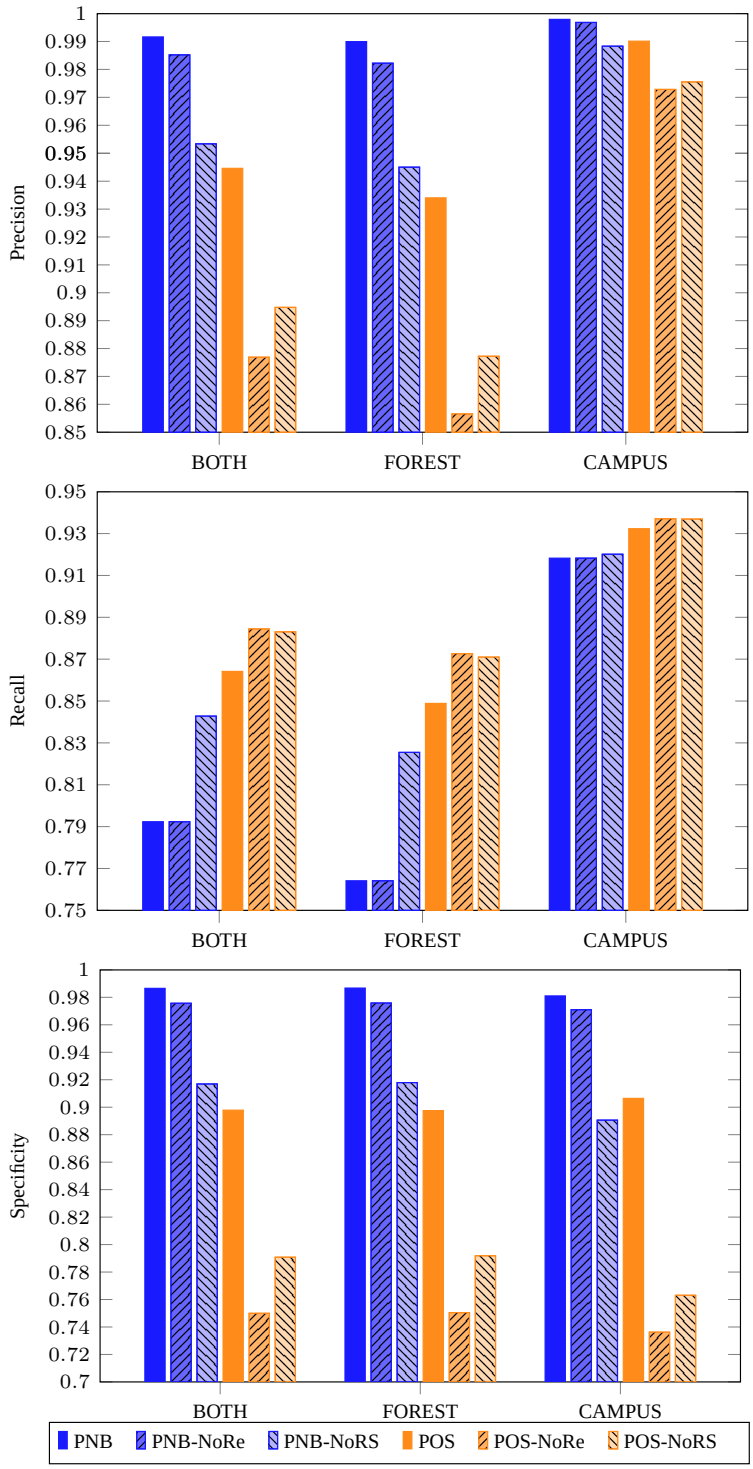


Figure 6.6: Precision, Recall and Specificity for the test trajectories of Viona. We compare the performance of the full feature vector, without remission values (NoRe) and without roughness and slope values (NoRS). The classifier based on PNB is shown in blue and the one based on POS is shown in orange. For both methods, using the full feature vector improves precision and specificity; the role of remission values and roughness and slope values behave differently for the two methods.

Table 6.1: Evaluation for Obelix on the campus trajectory.

Method	Measure	Full	NoRe	NoRS
PNB	Precision	0.978	0.924	0.958
	Recall	0.947	0.868	0.954
	Specificity	0.984	0.945	0.967
POS	Precision	0.975	0.637	0.840
	Recall	0.947	0.940	0.961
	Specificity	0.982	0.589	0.859

Results for Obelix on the campus trajectory (Figure (6.3)). Both learning methods behave quite similarly in this scenario when using the full feature vector. The absence of roughness and slope measures (NoRS) gives a better performance in this scenario than the absence of remission values (NoRe).

6.4.2 Evaluation using Obelix

For Obelix we used only a short training trajectory on the campus, see the blue part of Figure (6.3), since the complexity of the environment is much lower than for the forest data set. In this environment, both classifiers perform almost identically with the full feature vector, see Table 6.1. Both of the classifiers reach a precision of 0.98, a recall of 0.95 and specificity of 0.98. In this scenario, the remission values are more important than the roughness and slope parameters. Using the POS-based classifier the precision without remission values is 0.63 and without roughness and slope it is 0.84. Especially for the POS-based classifier, the combination of both improves the performance substantially, while for the PNB-based classifier the performance is similar.

On first view, it seems contrary that for Viona the roughness and slope features are more important than the remission features, and for Obelix it is the other way around. However, these measures are rather connected to the characteristics of the robots. As Obelix should only drive on the street and walkways, it is relatively important to distinguish them from grass, as the important measure of the step height can not necessarily distinguish between them. For this task, the remission values provide the more valuable information. On the other hand, Viona can easily drive over meadows but still needs to be aware of bushes. As our results suggest, the roughness and slope measures are more suitable for this purpose. Most importantly for the performance of our approach is that the full feature vector provides the best results in both cases. This means that the proposed features are suitable for different robots and can be used to determine traversability for operating in different environments.

6.5 Related Work

A comprehensive survey of traversability analysis methods for unmanned ground vehicles was presented by Papadakis [93]. He states that the predominant approaches to measuring traversability are based on occupancy grid maps, which are accredited to Moravec and Elfes [79]. Furthermore, they are based on the analysis of 2d elevation maps, where 3d information is represented in 2d maps. Pfaff et al. presented an approach, where the 2d

elevation maps were used for traversability analysis as well as for mapping and localization purposes [95]. A more general representation is a 2d grid map, where each cell stores features that provide enhanced information from the sensors. Papadakis identified this as the preferred choice when dense 3d point clouds are available [48, 51, 60].

Many methods for performing traversability analysis are based on heuristics that represent the capabilities of the robot, in combination with measurement models that describe the sensor noise [14, 17, 51]. These methods for classifying traversability typically work well for many environments, but they are limited in their generality, since they often do not explicitly distinguish different types of obstacles, like rocks or grass. Moreover, a specific heuristic has to be developed for every robot and also for different combinations of sensors and terrain. Murphy and Newman use a probabilistic costmap, which is generated from the combined output of a multiclass Gaussian process classifier and Gaussian process regressors. It models the spatial variation in cost for the terrain type, to perform fast path planning while taking the uncertainty of the terrain cost into account [81].

Another problem that is hard to tackle are so called negative obstacles, like holes in the ground or downwards leading steps. The sensor is not necessarily able to perceive the lower part of the structure and, therefore, the robot has to reason about the cause of missing data, which might result from untraversable gaps or simple visibility issues [64, 107]. This is of special interest in search and rescue scenarios after disasters, where the environment is very complex due to irregularities. This kind of analysis is particularly critical when the sensors used only provide a sparse representation of the environment, like rigidly mounted, downwards facing 2d-laser scanners.

It seems natural to let the robot learn about the traversability of the environment. This has the advantage that there is no need for a heuristic to interpret the sensor data. But in supervised scenarios, one has to provide labeled data to the approach, to learn from. Lalonde et al. [63] proposed segmenting 3d point clouds into the categories: *scatter*, to represent porous volumes; *linear*, to capture, *e.g.,* wires and tree branches; and *surface* to capture solid objects, like ground surfaces. The authors achieve this by training a Gaussian Mixture Model with an expectation maximization algorithm, on hand labeled data. A different way to perceive the environment is to use proprioceptive measures, like bumper hits, measuring slip, or vibration. Those can be combined with geometric measures and used, *e.g.,* to project the current measurements into the environment [5, 48, 105]. Yet, the use of proprioceptive measures requires an adequately robust robot that is physically able to traverse the terrain in question. Even though such methods allow the robot to autonomously learn a model of the environment, the trial and error part of this methodology involves a high risk of damage to the robot.

In contrast to this, our approach uses data provided by manual operation of the robot by a, non-specifically trained, human operator who drove a safe trajectory and, therefore, provided partially labeled training data. This is a very convenient, safe, and time-efficient way of training a classifier. An approach that follows a similar idea was presented by Ollis et al. [91]. Their system uses data from a stereo camera, radar, as well as 2d- and 3d-lidar sensors. Features are computed as multidimensional histograms, and a distribution is learned for the traversed cells. The approach makes use of a monotonicity assumption that states that cells with higher values of the features would be expected to be less traversable and the inferred probabilities were enforced to meet this assumption. The resulting values are mapped to a cost function that is then used for planning.

In our approach there are no heuristic assumptions about the features and their relation to traversability. To solve the restricted learning problem, we adapted the techniques of Denis et al. [27] and Elkan and Noto [29] to learn the probabilities from the available data.

6.6 Conclusion and Future Work

We presented an *easy to use* approach to learn traversability models for mobile robots. The proposed feature vector can capture the important information for traversability from 3d-lidar data. Furthermore, we successfully applied techniques for learning from training data that provides only partial information in terms of positive labels. In the experiments, we showed that our approach can be applied to different robots that exhibit major differences in their traversability characteristics. Moreover, our approach is usable in outdoor urban environments as well as in unstructured non-urban environments, like forest roads and grassland. Our experiments also include a discussion of the expressiveness of the proposed features in the context of different environmental characteristics.

Future Work

So far, the approach suffers from a growing size of the terrain models. It will be worth putting further efforts to this point. We expect that this is also connected to the lower recall results for unstructured environments. To apply the method to a real robot, an extension towards dealing with dynamics in the environment is necessary for obvious safety-related reasons. We tackle some of these points in the forthcoming chapter.

Chapter 7

Adaptive Obstacle Detection

Reliable detection and avoidance of obstacles guarantee safety and mobility, and are, therefore, crucial prerequisites for autonomously navigating robots. To ensure safe mobility, the obstacle detection needs to run online, meaning that the limited resources available to autonomous systems need to be taken into account. At the same time, robust obstacle detection is highly important. Here, an overly conservative approach might restrict the mobility of the robot, while a more reckless one might harm the robot or the environment in which it is operating. In this chapter, we present a probabilistic terrain-adaptive approach to obstacle detection, which relies on 3d-lidar data and combines computationally fast and cheap geometric features, like step height and steepness, which are updated with the frequency of the lidar sensor, with semantic terrain information, which is updated with at lower frequency. We provide experiments in which we evaluate our approach on a real robot on an autonomous run, over several kilometers containing different terrain types. The experiments demonstrate that our approach is suitable for autonomous systems that have to navigate reliably on different terrain types including urban streets, dirt roads and grass.

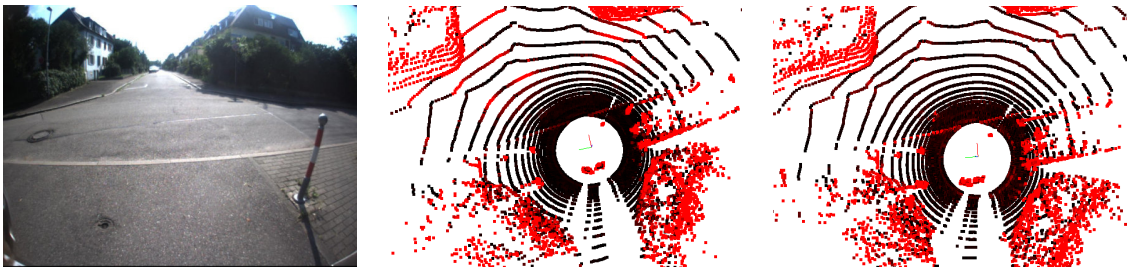
As stated in the previous chapter, inferring where a robot can safely drive and where it can not, is an important component for an autonomously navigating robot. In the previous chapter, we focused on an approach that is easy to use from the perspective of a non-expert user. Alhough the results of the approach were promising, a drawback was that it could not be updated with every measurement and assumes a static traversability map. In this chapter, we focus on the computational effort and propose an approach to online obstacle detection that enables the robot to detect obstacles with each measurement provided by the laser sensor. To achieve this, we combine a supervised learning method, yielding a low frequency terrain classification, which is based on accumulated measurements, with fast to compute geometric features that are computed for each measurement.

In this chapter, we are particularly interested in the situation in which a robot operates in environments with different characteristics, *e.g.,* urban, outer-urban and off road. The scene that is depicted in Figure (7.1) gives us an idea about the difficulties in mixed scenarios. The main problem is that the flatness assumption of man-made environments does not hold. Grass of various heights may easily appear as fake obstacles, because observations of the same

Figure 7.1: Our robot navigating autonomously on different types of terrain. It needs to cope with: grass of various heights, small trees, bushes, dirt roads and regular streets. The corresponding point clouds need to be dealt with differently depending on the type of terrain on which the robot moves.

height differences on a street may constitute a hazardous situation. This means that in mixed terrain settings, it is not possible to rely solely on the geometric features, because free space in one type of terrain may look like an obstacle in another, and vice versa. In the previous chapter, we derived the need of modeling the underlying terrain class, in the context of the learning problem there. While we used a statistical test in the former case, the number of terrain classes grew with the length of training examples. In this chapter, we propose using a classifier with preset terrain classes that is trained on hand labeled data. We take the burden of generating this hand labeled training data for the benefit of a more expressive power of the semantic information. We believe that our proposed approach, in this chapter, may be particularly relevant for autonomously navigating robots in agricultural or forestry applications.

As in the previous chapter, we will again rely only on a 3d-lidar as the perceptual sensor. We use lidar because it is more robust to different lighting conditions, while, *e.g.,* a camera in a forest would be highly affected by shadows and under/over exposed areas. More specifically, our 3d-lidar sensor is a Velodyne HDL 64, which provides a full 360° horizontal field of view, returning $\sim$1.3 million points per second with distance and remission information that needs to be processed. To cope with the resulting computational challenges, our approach takes advantage of the special structure of the Velodyne scans. The approach we present is able to detect traversable regions on medium high grass, dirt roads and regular streets using two threads on a quad core CPU i7@3.50GHz with a workload of $\sim$100% (of 400%) leaving enough computational power for other modules, like path planning or control.



(a) Onboard view (b) $stepHeight$ (c) $incline$

Figure 7.2: Example for the geometric measures extracted from Velodyne data. From left to right: A photo from the robots perspective in (a), a top view of the pointcloud with coloring based on the $stepHeight$ in (b), where black corresponds to low values and red to high values, and the pointcloud visualization of the $incline$ in (c), using the same color scheme.

The reminder of this chapter is organized as follows. First, in Section (7.1), we describe how we compute the fast geometric features and how we can get a basic online obstacle detection approach from it. Then we describe how our terrain classification approach works and how the terrain map is maintained in Section (7.2). To complete our combined approach, we discuss the fusion of both sources of information in Section (7.3). Finally, we evaluate the performance of our approach in Section (7.4), where we explicitly illustrate why we need terrain information in mixed terrain scenarios, evaluate the terrain classification and evaluate the computational cost of our approach in a real world experiment. At the end of the chapter, we discuss related work in Section (7.5), and conclude the chapter in Section (7.6).

7.1 Online Obstacle Detection

In this section, we discuss how we efficiently calculate the geometric measures from the 3d-lidar data. First, we give a short technical overview of the sensor characteristics. Then we explain how we exploit the special structure of the data to efficiently calculate the geometrical measures. Finally, we discuss how to use these measures for obstacle detection.

7.1.1 Velodyne Intrinsics

The Velodyne HDL sensors are very popular for autonomous robots. Currently, there are three versions with 16, 32 or 64 individual laser beams available. The individual lasers are mounted in the sensor with different pitch angles, setting the vertical field of view, which ranges from $-15°$ to $15°$ for the HDL 16, $-20°$ to $20°$ for the HDL 32 and $-24.8°$ to $2°$ for the HDL 64. The data of the sensor is delivered in spherical coordinates, providing the current azimuth and elevation angle, the distance, the remission and the id of the laser. This leads to the typical ring structure of the Velodyne scans, as, e.g., depicted in Figure (7.2).

7.1.2 Geometric considerations

Utilizing this structure, we can directly arrange the 3d-points as a matrix P_{ij}, where the elevation determines i and the azimuth j. For every point P_{ij} we search in each direction on

the same ring $P_{i(j\pm k)}$ and perpendicular to it $P_{(i\pm l),j}$, with $0 \leq k \leq K$ and $0 \leq l \leq L$, for the first point that has an Euclidean distance larger than x in 3d.

$$k^{\star+} = \min\{k \mid 0 < k \leq K \wedge \|P_{ij} - P_{i(j+k)}\|_2 > x\} \tag{7.1}$$
$$k^{\star-} = \min\{k \mid 0 < k \leq K \wedge \|P_{ij} - P_{i(j-k)}\|_2 > x\} \tag{7.2}$$
$$l^{\star+} = \min\{l \mid 0 < l \leq L \wedge \|P_{ij} - P_{(i+l)j}\|_2 > x\} \tag{7.3}$$
$$l^{\star-} = \min\{l \mid 0 < l \leq L \wedge \|P_{ij} - P_{(i-l)j}\|_2 > x\} \tag{7.4}$$

This procedure returns, at most, four neighbors with a maximum of $2(K + L)$ comparisons. From these points we calculate the step height ($stepHeight$), as the maximum absolute difference of the z-coordinates as well as the inclination angle ($incline$) of the line that connects the two points and the xy-plane, which corresponds to the steepness between the points.

$$stepHeight = \max\{\|\mathbf{p}_{ij} - \mathbf{q}\|_{\mathbf{\Pi_z}} \mid \mathbf{q} \in \{\mathbf{p}_{(i\pm l^{\star\pm})j}, \mathbf{p}_{i(j\pm k^{\star\pm})}\}\} \tag{7.5}$$

$$incline = \arcsin\left(\frac{\|\mathbf{p}_{ij} - \mathbf{q}^{\star}\|_{\mathbf{\Pi_z}}}{\|\mathbf{p}_{ij} - \mathbf{q}^{\star}\|_2}\right) \tag{7.6}$$

$$\text{with } \Pi_z = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{7.7}$$

Here $\mathbf{q}^{\star}$ is the point that corresponds to the argmax of the $stepHeight$ calculation. For a visualization of $stepHeight$ and $incline$ see Figure (7.2), and an example to clarify the computation of the geometric features is depicted in Figure (7.3). Due to the finite number of comparisons and the constant time look-up for the points, the overall computation of $stepHeight$ and $incline$ for one point in the scan can be done in constant time. In our implementation, we use $K = 5$, $L = 2$ and $x = 0.05m$, which is more than 2σ of the typical sensor noise of the Velodyne. Please note that this procedure requires knowledge about the robot's current pitch and roll angle, as, e.g., provided by an IMU on the robot.

As a pre-processing step, in order to reduce the influence of sensor measurement noise, we average close-by neighboring points along the ring, $P_{(i\pm k)j}$ with $0 \leq k \leq K$, thereby smoothing the surface structure. We compute this average efficiently using a sliding window so that the pre-processing does not break the constant time cost stated above.

In our current C++ implementation we can calculate $stepHeight$ and $incline$ for every point provided by the Velodyne with approximately 50% CPU load on a single core of an Intel i7@3.5GHz PC.

7.1.3 Basic Obstacle Detection

In our implementation, we maintain a rolling occupancy 2d-grid map with a fixed resolution, which is updated whenever a 3d-lidar measurement arrives, and the cells are shifted whenever the robot moves, such that the robot is always centered on the map. The 2d map is aligned perpendicular to the gravity vector, which is estimated from the IMU measurements. To

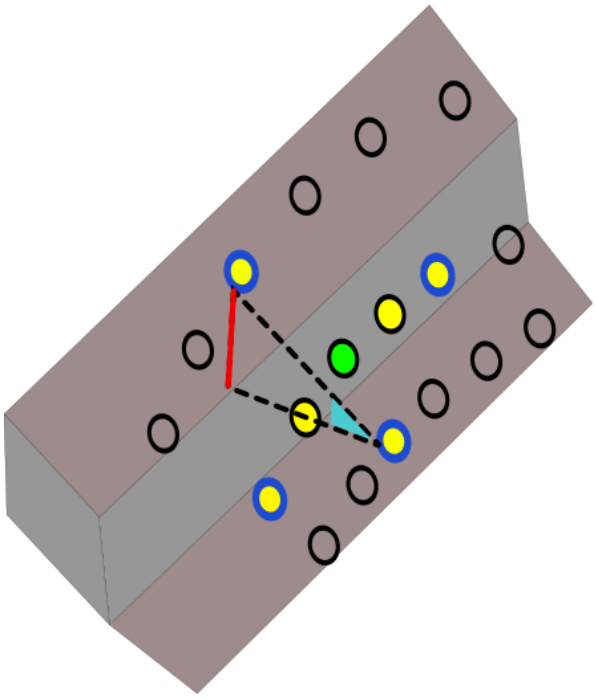


Figure 7.3: Schematic example for the computation of the geometric features. The green point is the point that is currently considered and the yellow points mark the local neighborhood for the computations. They are along the ring and perpendicular to it. The points with blue strokes satisfy our requirements of a minimum distance and are used for the computations. The two points connected with the dashed line are those with the largest difference in the z-coordinate. The $stepHeight$ is indicated by the red line of the triangle. Accordingly, we take the angle marked in cyan as $incline$.

determine the cell index for a 3d measurement we use the orthogonal projection to the xy-plane and chain it with the discretization determined by the resolution of the map $res\{X, Y\}$.

$$\mathbb{R}^3 \longrightarrow \mathbb{R}^2 \longrightarrow \mathbb{Z}^2 \tag{7.8}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \longmapsto \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} \lfloor \frac{x}{resX} \rfloor \\ \lfloor \frac{y}{resY} \rfloor \end{pmatrix} \tag{7.9}$$

The geometric properties, as computed in Section (7.1.2), are typically sufficient to perform a traversability analysis for environments such as offices, fabric halls or urban roads. Therefore, we use a recursive formula to update the map, which ensures that we integrate the information of each measurements. In this chapter, we are interested in obstacles that are persistent in the environment and therefore assume that whether or not an obstacle is present in cell c is a static property of the cell. For the recursive estimate of the probability, that cell c contains an obstacle, given the history of measurements, $P_c(o \mid g_1, \ldots, g_n)$ we can use the efficient logit representation.

$$\begin{aligned} &\mathrm{logit}(P_c(o \mid g_1, \ldots, g_n)) \\ &= \mathrm{logit}(P(o \mid g_n)) + \mathrm{logit}(P_c(o \mid g_1, \ldots, g_{n-1})) - \mathrm{logit}(P_c(o)) \end{aligned} \tag{7.10}$$

To perform the recursive calculation, we need only to specify the distribution for $P(o \mid g)$. In Figure (7.2) we depict the magnitude of the geometrical measures for an urban-scene. Comparing the values and the scene, we clearly see that all vertical structures, *e.g.,* the pole near the robot and the bushes on the opposite side of the street, exhibit high values (red) for both measures. On the other hand, we can observe a contrary behavior of the two measures at potential false-positive detections of obstacles, *e.g.,* the lowered curbs right in front of the robot show a strong response on the $incline$ values in contrast to low values of the $stepHeight$, and vice versa on the pavement on the opposite side of the street. The observation of this behavior is not unexpected, as the $incline$ is rather sensitive to noise at points nearby the robot, due to the smaller distances of the points, while at more distant points the incline is robust to such noise. The $stepHeight$ suffers from noise at the more distant points, since it calculates the absolute differences in the z-coordinate.

Taking this into account, the distribution $P(o \mid g)$ should exhibit a high probability if and only if both values are relative large. So far, we have used two different models for this. The first, a straightforward and efficient way, is to model it as a Dirac distribution using the thresholds $maxStepHeight$ and $maxIncline$.

$$P(o \mid g) = \begin{cases} 1 \text{ if } incline > maxIncline \wedge stepHeight > maxStepHeight \\ 0 \text{ else} \end{cases} \tag{7.11}$$

For urban or indoor environments, we usually choose $maxStepHeight = 0.05$ and $maxIncline = 20.5°$. One drawback of this threshold-based modeling is that we cannot apply Eq. (7.10), since the border values 0 and 1 are not defined for the logit-representation. As a workaround, we propose marking a cell as occupied for a fixed time span if $P(o \mid g) = 1$, assuming that obstacles are frequently observed, when using the threshold-based measure.

However, modeling the probability that an obstacle is present, given the geometric measures, continuously, taking their extent into account, should lead to a better estimate. In order to take into account the characteristics specific to the geometric measures, we propose using

$$P(o \mid g) = (1 - \exp(-\alpha stepHeight)) \sqrt{1 - \exp(-\beta \sin(incline))}, \tag{7.12}$$

with $\alpha = 25$ and $\beta = 1.2$ in urban environments. This function shows a similar but smoother behavior, like the step-function of the Dirac distribution. The values of both functions are depicted in Figure (7.4) for a comparison of the returned values. In this case, we can apply Eq. (7.10) and thereby take the extent of the geometric measures into account.

Independent of the specific model of $P(o \mid g)$, when the robot enters outer-urban environments, the probabilities that an obstacle is present depends on the type of terrain on which the robot operates. For an idea, see the top row, right column of Figure (7.6). In this case, the robot drives on a meadow. When we use the probabilities as modeled for the urban scenario, fake obstacles appear all around the robot, which makes progress on this terrain impossible. Accordingly, our goal is to infer about the terrain type and dynamically adapt the obstacle probability.

7.2 Terrain Analysis

When the environment is more scattered in its geometry, like a meadow or forest environment, it is not possible to rely purely on the geometrical measures, since *e.g.,* grass may appear

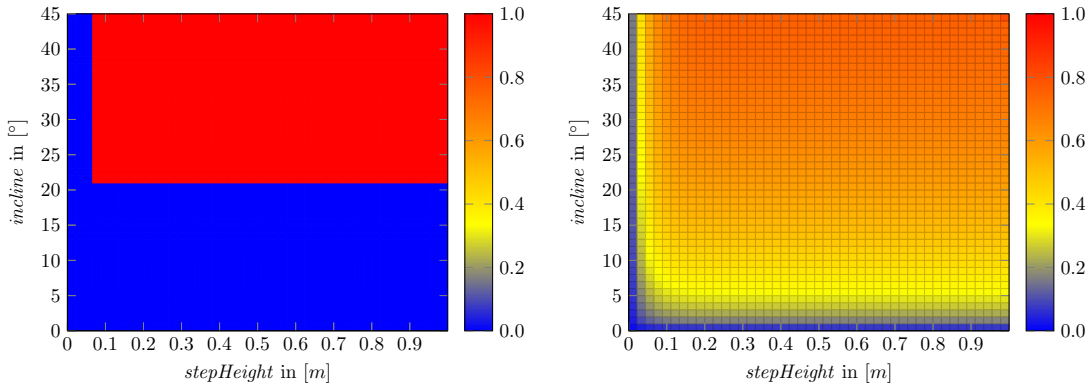


Figure 7.4: Comparison of the probabilities that an obstacle is present, given the geometric measure. On the left is the Dirac distribution on the left the smooth distribution.

as fake obstacles with step heights even larger than the ground clearance of the robot. On the other hand, a robot that should not drive on grass, may consider a mowed meadow as traversable ground and drive over it. To avoid those situations, we use a terrain classification based on features, extracted from points that are registered into a fixed resolution moving grid-map, as in the previous section, and infer the terrain class using a random forest (RF) classifier. We decided to use a RF-classifier, since it is fast and known for robustness and accuracy, it also handles the different magnitudes of the individual dimensions of the feature vector internally. Yet, in general, every multiclass classifier could be employed. Since we are only interested in the local vicinity of the robot, we utilize the raw odometry aided by an IMU sensor for the registration of the points. Points from the lidar are mapped with these pose estimates until the robot has covered a distance of d meters. Then, we compute one feature for each cell of the map, if more than a predefined value of $minPoints$ are registered in that cell. Using the inverse sensor model from the random-forest classifier, we can maintain a probability distribution over the terrain class for each cell.

7.2.1 Features

The features we use for this approach are similar to those we used in the last Chapter 6, as they were working well in the context of traversability analysis. However, since the maximum step height is already represented by our geometrical features and we believe that it is closer related to the traversability characteristics than to the terrain type, we exchange it with the maximum remission value of a cell. For completeness, our five-dimensional feature is composed of the following measures:

- Maximum remission
- Mean remission
- Standard deviation of the intensities
- Roughness
- Slope

The first three dimensions represent the perceptive or appearance-based information that we get from the intensity values of the 3d-lidar sensor and the last two represent characteristic geometric information of the terrain. The results of the previous chapter suggest that both measures are important when facing mixed terrain scenarios.

7.2.2 Classification

In our approach, we apply an RF-classifier, as introduced by Breiman [16] with the gini-impurity as split criterion for the nodes of the trees, see Chapter 2, Section (2.6.1). We hand-labeled data from several scenes, which were recorded in different environments, which include different terrain types like streets, grass, dirt- and forest roads, in order to train the classifier. For the labeling, we apply a custom tool to determine areas of the corresponding terrain type. All of the labeled data were assigned to one of the following classes:

- street
- grass
- dirtroad
- other

The first three classes constitute the terrain we consider as potentially traversable, while the last class ideally unifies all others that are likely to be untraversable. We train the random forest to model the inverse sensor model, *i.e.* it returns a probability estimate $P(t \mid f)$ for the terrain class t given the feature f.

7.2.3 Terrain-Class-Map

The terrain class map is a fixed resolution rolling grid-map, where we temporarily store the raw points until we compute a feature. Then, we calculate a probability distribution over the terrain class, based on the probabilities provided from the classifier. We assume that the cells in the terrain map are independent from each other and that the state of a cell is static. Therefore, we can maintain the probability distribution over the terrain class given the registered features, $P\left(t \mid f_1, \ldots, f_n\right)$, for each cell independently. Since we want to update the information incrementally as soon as they are available, we apply the Bayes rule twice with the assumptions stated above to derive:

$$P(t \mid f_1, \ldots, f_n) = \frac{P(f_n \mid t, f_1, \ldots, f_{n-1}) P(t \mid f_1, \ldots, f_{n-1})}{P(f_n \mid f_1, \ldots, f_{n-1})} \tag{7.13}$$

$$= \frac{P(t \mid f_n) P(f_n) P(t \mid f_1, \ldots, f_{n-1})}{P(t) P(f_n \mid f_1, \ldots, f_{n-1})} \tag{7.14}$$

$$= \eta \frac{P(c \mid f_n) P(t \mid f_1, \ldots, f_{n-1})}{P(t)} \tag{7.15}$$

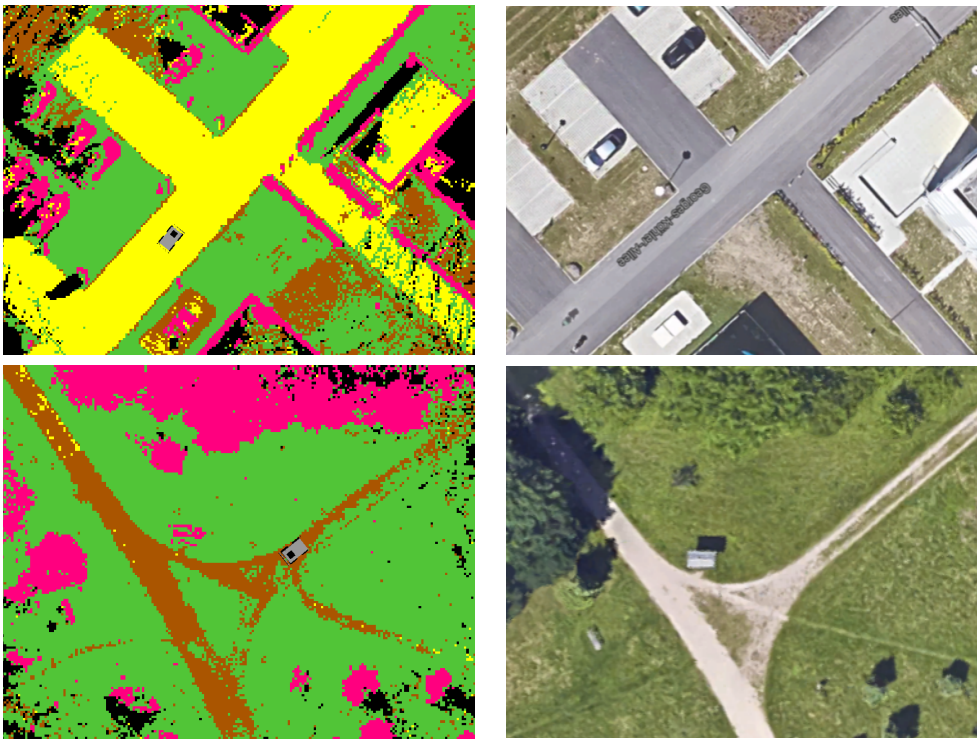


Figure 7.5: Left: results of our terrain classification. Yellow is street, green is grass, brown is dirt and pink is other. Right: aerial image of the scene. Our terrain classification accurately classifies different terrain types in urban and outer urban environments.

For Eq. (7.13), we apply the Bayes rule and further assume that the current feature f_n, given the cell class, does not depend on other features. This, and again Bayes rule, for $P(f_n \mid t)$ leads to Eq. (7.14). Finally, we summarize the terms that do not depend on the class of the cells as a normalization constant η to derive Eq. (7.15). In our implementation, we use a uniform prior for the class and the inverse sensor model from the random-forest classifier, to compute the class probabilities of individual cells.

To save computational resources, we apply the assumption of the static terrain class and mark a cell as classified, if it has been seen more than ten times and if the probability of the most likely terrain class exceeds 0.9. Classified cells are not considered for updates of the terrain class but keep their state. In order to counteract cases where the state of the cells may change from time to time, *e.g.,* a parking car that may start driving, we reconsider classified cells for updates after a fixed amount of time.

7.3 Terrain-Adaptive Obstacle Detection

Our obstacle detection module combines the methods explained in Section (7.1) and Section (7.2). In our implementation, we use two asynchronous threads: one that computes the geometric measures and the other the terrain analysis. To circumvent race conditions, we maintain two independent maps: one for the terrain class and the other fuses the information of our combined approach, which is then given to a planner. Therefore, the system has critical

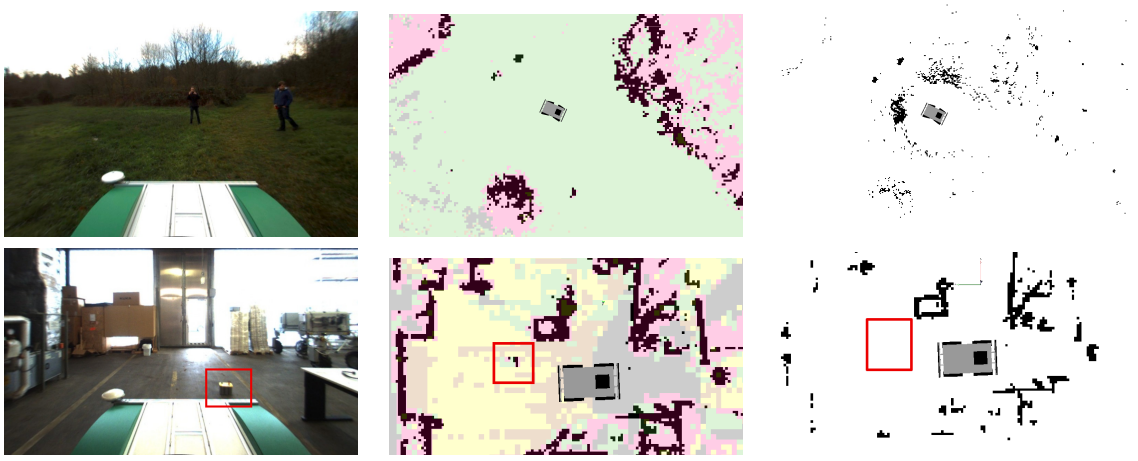


Figure 7.6: Left: an on-board view of the scene. Middle: the resulting obstacle map blended with the terrain classification of our approach. Right: the resulting obstacle map of the basic approach. In the top row example, our approach guarantees full mobility while the basic approach is trapped by many fake obstacles induced from the grass. In the lower row example, we see what happens if we set the threshold of the basic approach to the same as for grass in our combined approach. Our approach uses adaptive thresholds and correctly finds the obstacle. The basic approach misses the massive battery in front of the robot, as marked with the red rectangle.

information available as soon as it arrives but can still gain from the slower updates of the terrain class model.

We model the terrain type as a latent variable to obstacle detection, which allows the use of both measures in a probabilistic sound formulation. Given the feature set $\mathcal{F}$ and the geometric feature g this results in the following computation:

$$P(o \mid g, \mathcal{F}) = \sum_t P(o \mid g, t) P(t \mid \mathcal{F}) \tag{7.16}$$

Thereby, we assume that $P(o \mid g, \mathcal{F}, t) = P(o \mid g, t)$, which means that given the terrain class the probability of containing an obstacle is independent from the observed feature f and $P(t \mid g, f) = P(t \mid f)$, which means that the terrain type t is independent from the geometric feature g. With Eq. (7.16) we can use a different probability model for each terrain type due to the dependency of $P(o \mid g, t)$ on the terrain type t.

A computationally efficient alternative is to approximate Eq. (7.16) in a one-vs-all manner, which uses the obstacle probability for the most likely terrain type t only.

$$P(o \mid g, \mathcal{F}) = P(o \mid g, t^\star), \text{ with } t^\star = \underset{t \in \mathcal{T}}{\operatorname{argmax}} P(t \mid \mathcal{F}) \tag{7.17}$$

Here we lose potential information, since we take only the most likely terrain type into account; however, the most likely terrain class can easily be tracked by the terrain map. With this computation, we have fewer evaluations of $P(o \mid g, t)$, multiplications and memory access operations, which leads to faster estimates and lower CPU load. The effect of this approximation is discussed in the Section (7.4.4).

Table 7.1: Terrain class dependent parameters

Terrain Class	$maxStepHeight$ [m]	$maxIncline$ [°]	α	β
Street	0.05	20.5	25	1.2
Grass	0.50	20.5	10	1.2
Dirt	0.25	20.5	3	1.2
Other	0.05	20.5	25	1.2

7.4 Experiments

We designed our experiments to highlight the different aspects of the proposed approach. We provide qualitative and quantitative results about the performance of our approach. In addition, we present a real world experiment of an autonomous four km run through challenging terrain. Furthermore, we provide a detailed analysis of the computational requirements, illustrating the economical use of computational resources by our approach. For all experiments, we used the same parameters and the same random forest, which consists of six trees. The trees were grown without a depth limit. Each split considered two randomly chosen variables, while the minimal number of features to split a node was 100 and the split criterion was based on the gini-index. See Chapter 2 Section (2.6.1) for more details. The terrain analysis grid had a resolution of $0.2m$, a size of 300×300 and d was set to $0.5m$. The map that was given to the planner, which is updated by our combined approach, had a resolution of $0.1m$ and a size of 600×600. The maximum speed of our robot was set to $1.2m/s$. In order to circumvent inconsistent lidar data, we applied the calibration approach by Steder et al. [110] for our Velodyne HDL64.

The remainder of this section is organized as follows. First, we illustrate qualitatively that the knowledge about the terrain type can be important for obstacle detection in Section (7.4.1). Afterwards, we briefly evaluate the accuracy of our terrain classification with quantitative measures in Section (7.4.2). Then we present the results of our approach when successfully applied to a real autonomous robot in Section (7.4.3). Finally, we analyze the effect of the approximations to compute the obstacle map, as pointed out during the chapter, as well as the respective computational effort in Section (7.4.4).

7.4.1 Illustration of Mixed Terrain Challenge

If no additional semantic information is available, we can only make a decision based on a single setting of the parameters. When we apply the parameters for street, as presented in Table 7.1, the system will easily run into problems if we are not actually on a street. In Figure (7.6), we show two examples of situations where the naive approach, without terrain class considerations, is either blocked by fake obstacles like grass or misses the detection of real obstacles if more reckless settings are used, *e.g.,* the one for grass. In the first row of Figure (7.6), we use the street parameters from Table 7.1, the rightmost image shows the result of the obstacle detection, where the robot is trapped by grass which has shown up as fake obstacles all around the robot. A simple but dangerous fix for the naive approach would be to increase the parameters such that larger step heights are allowed, *e.g.,* the grass parameters of Table 7.1, but then, as seen in the second row of Figure (7.6), we may encounter

Table 7.2: Terrain Classification Confusion Matrix

Class	Street	Grass	Dirt	Other
Street	0.79 (0.64)	0.06 (0.15)	0.14 (0.19)	0.01 (0.004)
Grass	0.02(0.005)	0.91 (0.86)	0.05 (0.13)	0.02 (0.002)
Dirt	0.02 (0.10)	0.18 (0.18)	0.78 (0.72)	0.03 (0.002)
Other	0.04 (0.06)	0.10 (0.10)	0.10 (0.03)	0.76 (0.82)

hazardous situations, since on the rightmost image the rather massive obstacle, a battery, is not recognized as an obstacle anymore. In contrast, with our terrain-adaptive classification, we can classify both scenes correctly, as depicted in the middle column of Figure (7.6).

7.4.2 Terrain Classification Accuracy

To evaluate the performance of the terrain classification, we evaluated scenes as shown in Figure (7.5). We used partially labeled scenes from urban environments taken from our campus as well as from data in the forest. All in all, $\sim$100K labeled cells were evaluated, since only cells for which the class was obvious were labeled. The resulting confusion matrix is shown in Table 7.2, which presents the results of our cell-wise integrated classification, see Eq. (7.15) as well as the most likely class per feature, which is given in brackets. Our classifier can reliably distinguish the four classes from each other, but there are some confusion with street and dirt as well as with dirt and grass. One reasons for this is probably, as we observed, that streets are often covered with a little bit of dirt from construction or agricultural machines, which has a high influence on the remissions of the lidar. Also, the remissions on dirt seems very similar to those of grass, depending on its composition. It can also be seen in Figure (7.5), on the top left image, where grass and dirt are mixed in some regions, but the aerial image on the right indicates that it may not be totally clear what the correct labeling should be. Nevertheless, classifying grass reliably is important, since most fake obstacles appear in grass and the false classifications of grass did no harm as far as we could observe, since they are rather isolated.

7.4.3 Real World with Computational Analysis

In this experiment, we evaluate the capabilities and the computational economy of our approach in a real world scenario with a robot navigating autonomously on a four kilometer trajectory. The route starts at our campus and follows a circuit to a nearby forest before returning to the campus. The GPS-trajectory of this run is depicted in Figure (7.7). As we are interested in the computational efficiency, we used the Dirac-based formulation, see Eq. (7.11), for the obstacle detection, combined with the most likely terrain class, see Eq. (7.17), on the robot. During this experiment, we measured the time taken by the different tasks. Our algorithm was evaluated using a Intel(R) Core(TM) i7-3840QM @ 2.80GHz with four physical cores. We used a global planner, which sends short term goalpoints to a local planner, which itself has planned a path, based on the obstacle map of our approach.

In Figure (7.8) we visualized the CPU percentages of the components of our approach, in a pie-chart. The whole run took about $4400s$, which in return results in an average speed of

Figure 7.7: The trajectory of the outdoor experiment on an aerial image. The trajectory contains regular streets, grass and forest roads.

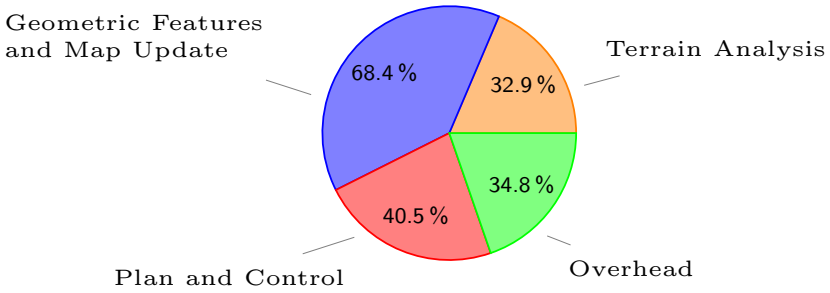


Figure 7.8: CPU usage of our approach during the real world experiment.

$0.9m/s$, while the maximum speed of the robot was limited to $1.2m/s$. The overall processor clock time was $7760s$, where also: path planning, collision checking, motion control, a GUI and the message management was included, which leads to an overall average processor usage of about 180%. For the terrain classification, including the feature computation, the system used on average about 33% of its processing power, while for the high frequency geometric measures, including the obstacle grid map update, it needed about 68%. This underlines the economy of our approach, since on our quadcore CPU, more than half of the processing power was available when our navigation software was running. Moreover, we showed that our approach can be applied, both online and in the real-world, on an autonomous robot for safe navigation in forested and outer urban environments, even with the applied approximations.

Finally, we took the recorded data and re-evaluated the CPU-usage of our approach when we use the full probabilistic model, Eq. (7.12) for the obstacle probabilities and Eq. (7.16) to include the terrain class. In this case we observe an average processor usage that is about 26% higher than using the approximations for the geometric feature and map update. On our system it turned out that this is critical since messages may queue up or get ignored. Therefore,

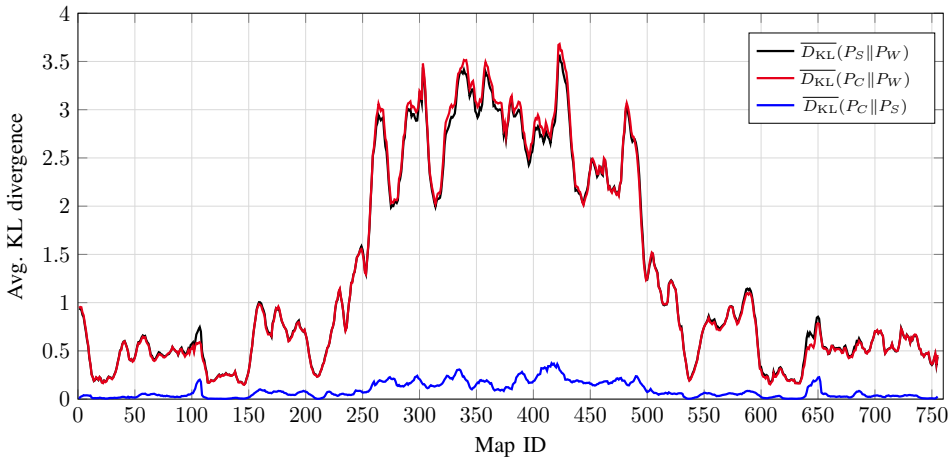


Figure 7.9: Average KL divergence for each map. Note the different extend after the robot enters (map ID 250) and leaves (map ID 500) the forest environment. The difference between P_C and P_W is more substantial, however, the difference between P_C and P_S is visible as well.

we will investigate the effect of the approximations in the next section.

7.4.4 Effect of Approximations

In this section, we discuss the differences between the possible approximations, which we have discussed earlier in this chapter, and the mathematically exact model. We compare two approximations (P_W, P_S) with the exact model (P_C):

- P_W denotes the rough approximation that was used for the online experiment in the previous section, which is the combination of the Dirac approximation (Eq. (7.11)) and the most likely terrain class (Eq. (7.17))
- P_S denotes the approximation using the smooth probabilities (Eq. (7.12)) in combination with the most likely terrain class (Eq. (7.17))
- P_C denotes the mathematically exact model, which is the combination of the smooth probabilities (Eq. (7.12)) and integrates over the full estimate of the terrain classification (Eq. (7.16))

We processed the data from the autonomous run offline and saved a snapshot of the obstacle probability map every $5m$ for the evaluation of the approximations. From these snapshots, we compare the probability distributions of the individual cells to contain an obstacle. To quantify the differences between the distributions, we compute the Kullback-Leibler (KL) divergence for each of the corresponding cell pairs. The KL divergence is commonly used to compare the quality of approximations to probability measures. For two probability measures P and Q over the same domain Ω the KL divergence is defined as follows:

$$D_{\mathrm{KL}}\left(P \| Q\right) = \sum_{x \in \Omega} P(x) \log_2 \left(\frac{P(x)}{Q(x)} \right) \tag{7.18}$$

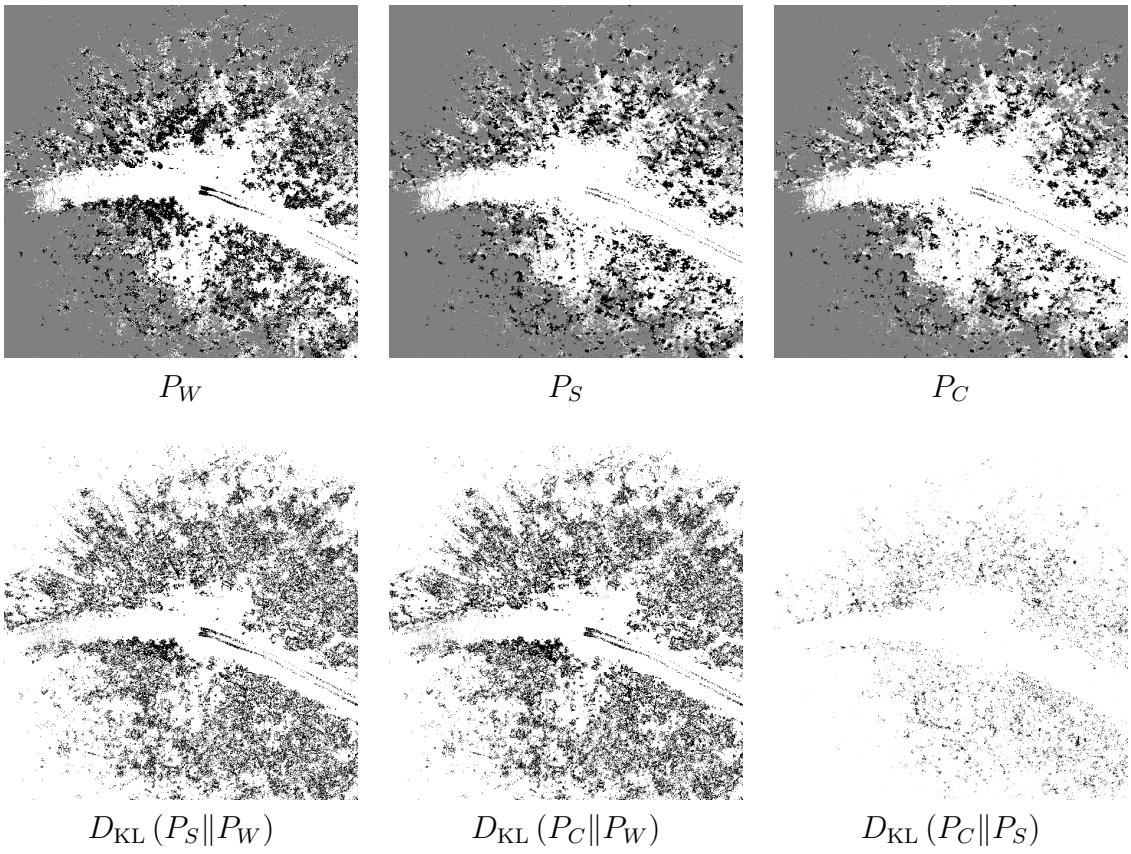


Figure 7.10: *Top:* Probabilistic obstacle maps for the probability measures. *Bottom:* Cell-wise KL divergence for the approximations depicted as gray scale images.

We calculate the average KL divergence over all cells that contain observations to quantify the differences between two maps, which we denote as $\overline{D_{\mathrm{KL}}}(P\|Q)$. The results are depicted in Figure (7.9) that shows the comparison between the $\overline{D_{\mathrm{KL}}}(\cdot\|\cdot)$ for each of the maps computed under both the approximations and the exact model. It can be observed that the difference between P_C and P_W is more substantial than the difference between P_C and P_S. However, the difference between P_C and P_S is also non-negligible. To illustrate this difference more clearly, we depicted the obstacle maps and the KL divergence for each cell of the map ID 421, which exhibits the largest average KL divergence ($\overline{D_{\mathrm{KL}}}(P_C\|P_S)$) of 0.37, in Figure (7.10). One can observe that the KL divergence for the cells on the path (center of the map) is lower than the cluttered surroundings that usually include trees and bushes. This can be explained by two reasons: first, if the terrain classification is confident, which means that it exhibits a low entropy, then P_S is a close approximation of P_C. On the other hand, if the obstacle probability, $P(o \mid g, t)$, is similar for all the terrain classes, *e.g.,* on a flat surface, both the probabilities would be similar again. This effect is also visible in Figure (7.9) where we see that the values are larger between the map IDs 250 and 500, which is the part the robot's trajectory in the cluttered forested environment.

Finally, we measured the average computational time spent for the map update using our implementation of the different measures on an Intel(R) Core(TM) i7-2700K CPU @ 3.50GHz. The average computation time for an obstacle map update is $(5.9ms, 11.2ms, 47.7ms)$ for (P_W, P_S, P_C) respectively. For a map update, memory accesses, the evaluation of $P(o \mid g, t)$,

multiplications and additions are the most computationally expensive operations. Comparing the computation times of P_C and P_S, we notice that the run time difference, as expected, is directly proportional to the number of terrain classes, which is four in our case.

7.5 Related Work

Papadakis [93] categorizes traversability analysis methods for unmanned ground vehicles, non-exclusively, by their usage of proprioceptive or exteroceptive sensor data, which can be either geometry- or appearance-based. Our approach relies purely on exteroceptive sensor data (perceived with a 3d-lidar) from which to interpret the geometric and appearance characteristics. Moreover, Papadakis points out that the 2d-digital-elevation-maps [61, 90] are the most common choice when a 3D-Lidar is employed. The elevation maps were later used for mapping [95] and were extended to multilevel surface maps [124] to capture more complex environments. In this work, we also employ a 2d-grid-based map in order to organize and interpret the sensor data, where each cell stores features and information about the terrain type as well as the geometry.

Most approaches concentrate on a certain terrain scenario, either structured outdoor urban environments or unstructured, rough off-road terrain, since both environments typically underlie very different assumptions. In urban environments, it is more important to detect and track dynamic objects, like cars, pedestrians and cyclists, and reasoning about the street lane or sidewalks [7, 59, 72]. For rough terrain analysis, estimating the load bearing surface seems promising as done by [130, 131], but these approaches typically have a high computational burden, *e.g.,* 3d-ray-tracing and also rely on different types of sensors. Another way, is the estimation of the step height of cells, calculating the difference in the z-coordinate of points that fall into the cell, like in [20] where range-dependent thresholds were applied to determine obstacles. A probabilistic formulation was utilized by Thrun et al. [120]. They employed five 2d-lidar sensors, mounted in different angles, and performed statistical reasoning about the step height whilst also taking time-differences into account. Manduchi et al. [70] utilize RGB-D data from stereo-cameras and similar geometrical considerations, as we do. Instead of a grid-map based approach they reason about the obstacle nature of points, by checking truncated cones, which are placed on ground plane points. Terrain classification is deduced from the RGB-D information using a mixture of Gaussians. Moreover, they rely on a 2d-lidar sensor for terrain classification, analyzing histograms of distances. In contrast to this, our approach relies purely on 3D-Lidar data, which is less sensitive to lighting conditions and typically provides more accurate data.

Adding semantic information about the terrain type can be very useful, since different terrain types may follow different rules for different mobile robot platforms. Assuming a pure forest environment, McDaniel et al. [73] classify the ground plane and trees, estimating the tree center by using a least-squares fit on circles for candidate points, from 3d-lidar data. Besides an online time-of-flight camera based basic obstacle detection, Santamaria-Navarro et al. [103] perform an offline terrain classification using a Gaussian process classification approach; however, the classification is only for traversable and not traversable. Wurm et al. [133] use a tilted 2D-Lidar scanner to distinguish low grass vegetation from street in structured outdoor environments. The system utilizes self-supervised classification learning by employing a vibration classifier (proprioceptive sensor information) to train a support vector

machine. The features applied for classification are: range, incident angle and remission. Laible et al. [62] employ camera and lidar sensors to distinguish: asphalt, big tiles, grass, gravel and small tiles. In our work, we utilize distance and remission values from the 3d-lidar in order to compute various features that we employ to determine the terrain class, where we apply a random forest classifier [16] to distinguish between: regular street, dirt roads and grass. Finally, we combine the semantic terrain class estimate with geometrical measures such as step height and steepness, calculated from points of single scans, exploring the special structure of the sensor for efficient computations.

More recently, Valada et al. [126] exploited different spectra in a camera-based approach to classify traversable paths in forested environments. The approach employs a DCNN-based classifier, which requires heavy computations on a GPU. In contrast to this, the presented approach needs fewer than two cores on a standard laptop for fully autonomous navigation.

7.6 Conclusion and Future Work

In this chapter, we presented a probabilistically sound approach to online obstacle detection, adapted according to semantic terrain information. Our method combines low frequency terrain analysis and high frequency basic obstacle detection, and provides a reliable obstacle classification with a moderate usage of computational power. In our experiments, we identified situations where a naive approach, without the terrain adaptation, would yield suboptimal results for safety or mobility reasons. Moreover, we demonstrated that our approach is: economical with computational power, fast enough to run onboard, and capable of controlling an autonomous robot along a challenging four km track which includes different terrain types like grass, dirt roads and regular streets.

Future Work

We see potential in the combination of camera-based approaches with the proposed approach. The fusion of two such approaches could improve the quality of the obstacle detection, as the two sensors provide different characteristics, which may lead two complimentary effects.

Another, more general, perspective would be to develop an evaluation scheme, which makes approaches comparable. A protocol that contains specific challenges to solve for different environments and claims, could help to make approaches more comparable. We think that this is the way to go, as providing benchmark datasets for a wide variety of robots, sensors and environments does not seem feasible.

Chapter 8

Outer-Urban OpenStreetMap-based Autonomous Navigation

Autonomous navigation typically requires an accurate global map of the environment to perform global path planning from the current pose to the desired goal. This map is commonly acquired from pre-mission exploration and mapping runs. However, publicly available map services are widely used by humans for global navigation and, nowadays, provide an almost complete road network data. When using them for autonomous navigation with mobile robots, one is faced with the problem of inaccuracies in the map and the uncertainty about the position of the vehicle relative to the map, which are often in the magnitude of meters. In this chapter, we present a probabilistic approach that makes the street network data of OpenStreetMap applicable to autonomous navigation. The approach associates subgoals, retrieved from global path planning on OpenStreetMap, in the vicinity of the robot with local sensor data obtained from 3d-lidar. To achieve this, we combine semantic terrain information with a Markov-Chain Monte-Carlo technique. With our approach, the autonomously navigating robot stays on the trails while following the path planned in OpenStreetMap, even though the system is repeatedly exposed to substantial errors. The proposed approach was successfully used on a real robot, navigating autonomously at different and previously unseen locations. We perform extensive experiments to show the robustness of our approach regarding the alignment between the robot pose and the OpenStreetMap data.

At the beginning of this thesis we discussed the importance of mapping and localization for autonomous navigation. However, in order to build a map, a robot needs to explore the environment, which is a time and energy consuming process. Afterwards, pre-mission global planning is limited to the area that has already been explored, and to extend this area the robot needs to start the exploration and mapping process again. Given that a multitude of map-services provide almost complete maps of road networks and additional data, like the compendiums of buildings, this procedure of exploration and mapping seems avoidable. The problem to overcome is that those maps are not as accurate as maps that are built using sophisticated SLAM techniques. In this chapter, we propose a solution to this problem for outer-urban environments, which uses only the street network data, as no additional data is

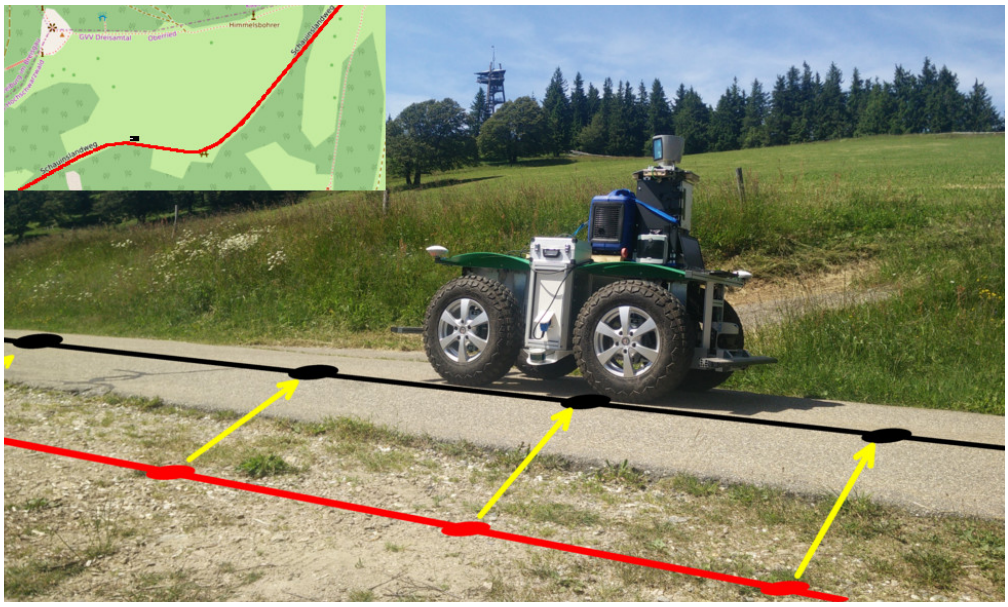


Figure 8.1: Our approach addresses robot navigation in outer-urban environments and aims to match a path given from OpenStreetMap (red) to the part in the sensor data that may most likely correspond to the street (black). It is designed to correct map and GPS errors.

typically available for such environments.

Global path planning is an important pre-condition for reliable and accurate mission execution of an autonomous robot. A large class of state-of-the-art solutions for large scale navigation uses solutions to the SLAM problem to create an accurate map of the environment. Some of them generate dense occupancy grid maps plus a topo-metrical graph, from it. The topo-metrical high level graph is then used for global path planning purposes and the fine grained map is loaded dynamically as map tiles for the local vicinity of the robot. In this case, assuming that the robot is accurately localized, the goalpoints planned in the global topo-metrical map are consistent with the local map. However, this has the drawback that unknown areas need to be explored and mapped first and then, at operation time, an active localization needs to run online. Moreover, planning is only possible in areas that have been explored already, and a lot of data needs to be stored.

In this chapter, we show that autonomous navigation in outer-urban environments is possible without the exploration and mapping task. To achieve this, we substitute the topo-metrical map with the street network data from OpenStreetMap[1] (OSM) and instead of computing globally consistent fine-grained grid-maps, we model the vicinity of the robot online, reducing the requirements of locally consistent grid-maps. On the other hand, this raises the problem that the global and the local map are not necessarily consistent. To overcome this, we track the position of the robot in the OSM using smoothed GPS measurements and align the trails in OSM with the local vicinity of the robot by exploiting semantic terrain information, as we need to deal with errors from GPS and OSM. This allows for autonomous navigation at previously unseen locations, provided that the street network of those is present in OSM.

The idea is inspired by the fact that humans use maps provided by cartographic services frequently and successfully for global path planning and navigation purposes. The cartographic

[1] https://openstreetmap.org

data provided by OpenStreetMap is based on "Volunteered Geographical Information" (VGI), see Haklay and Weber [45], and is publicly available under a "Open Database License (ODbL)". The problem within the context of robotics is that VGI maps contain a high uncertainty in their quality and even in what is considered a good quality map, the accuracy is often far from those generated with SLAM. We need to take several sources of errors into account, *e.g.,* nodes may be placed in incorrect positions by: careless contributors, inaccurate GPS estimates (during map creation and at operation time), and the sparse approximation of trails by line segments. Humans can easily resolve these inaccuracies. We hypothesize this originates from the ability to classify trails and associate them with the map, instead of relying purely on the GPS estimate. More specifically, humans know that they are currently standing on the street and not several meters away, as the GPS pose in the map suggests. Furthermore, the source of error does not matter for a human, we simply make our way along the trails and usually find our desired goal.

In this chapter, we present an approach to imitate human behavior and make maps like OpenStreetMap useful for global path planning in outer-urban environments, which includes navigating on small streets, dirt- and forest roads. To achieve this, we compare samples of the trails from the map with semantic terrain classification, using the approach from the previous chapter, and use a Markov-Chain Monte-Carlo technique to determine the most likely position of the trail in our local frame. With this correction, it is possible to use data from OSM for global path planning on a real autonomous robot, whereas frequently short term subgoals are sent to a local planner, which navigates the robot within its nearby vicinity.

The remainder of the chapter is organized as follows. First, we give a brief overview about the components of the system in Section (8.1). Subsequently, we discuss how we utilize the terrain classification to match the trails contained in OSM with the local vicinity of the robot in Section (8.2). Afterwards, we describe how the MCMC technique can be used for our purpose in Section (8.3). Finally, we evaluate the robustness and timings of our approach in Section (8.4). At the end, we provide an overview of related work in Section (8.5) and conclude the chapter in Section (8.6).

8.1 Preliminaries

In this section, we introduce the framework that we use in this chapter. We aim to use the road network from OpenStreetMap as a global topo-metrical map in order to provide short term subgoals to a local planner, which is applied to the local vicinity of the robot and is in charge of planning a path from the current robot position to the next subgoal. Thereby, it will take sensor information provided by GPS, IMU, odometry and a 3D-LiDAR into account. For obstacle detection, the local planner uses the approach provided in the previous chapter. The challenge is to correctly align the frame of OpenStreetMap, which is given in UTM coordinates, and the local frame of the robot such that the subgoals are located on the street and the position in the OSM is correct in order to send the correct upcoming subgoal at the right time. Thereby, we need to take into account that the GPS and odometry measurements are corrupted by noise and that the map may exhibit errors of varying extend. A failure of this alignment will result in subgoals that are not on the street and the robot may drive off, or if the subgoal is unreachable, it might even get stuck.

8.1.1 Planning on OpenStreetMap

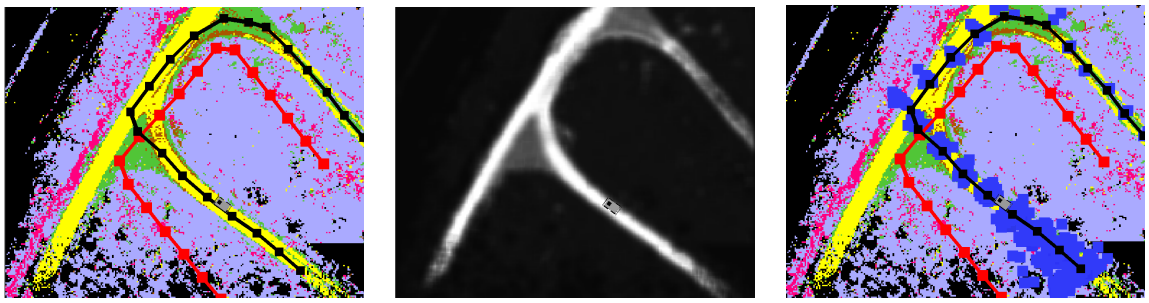
For efficient planning on the street-network graph, we use a standard A*-planner to plan a path to the desired goal location on the OSM graph. Assuming that the vehicle is located on a road, we determine the start point is, from the orthogonal projection, the closest street in OSM, according to the GPS pose estimate. In order to provide incremental subgoals to the local planner, we sub-sample the path generated by the A* algorithm equidistantly. Accordingly, we obtain a sequence of proposed subgoals $g_0^G, \ldots, g_n^G$ in the global UTM coordinate system. During the autonomous navigation, we keep track of the robot pose within OSM and whenever a subgoal is almost reached, we send the next one, until the final goal-point is sent. An example of this sub-sampled path is depicted in red in Figure (8.2) (a). As we can also see there, this approach, solely based on GPS, can not be used directly due to diverse sources of errors, such as inaccuracies in the map or the GPS pose estimate. From our observations, we can confirm the claim made by Máttyus et al. [71] that these errors mostly matter in the direction perpendicular to the road. To correct this error and thus make the navigation plan useful, our approach includes a dedicated classifier for the terrain in the vicinity of the robot to identify the road. In Figure (8.2) (a), we see that the uncorrected path, depicted in red, as retrieved from OSM is heading offroad and the robot could not reach those poses at all. In contrast to that, the corrected path, depicted by black lines, is nicely aligned with the street (yellow), which enables the robot to follow the desired path on the trail.

8.1.2 Semantic Terrain Information

To integrate the global subgoals consistently into the local frame of the robot, we will rely on our 3d-lidar-based semantic classification of the terrain type, which we have already introduced in Chapter 7.2. We distinguish between the terrain classes *street, dirt road, grass, vegetation* and *other*, see Figure (8.2) (a) for a visualization of the terrain class map in a forest environment. With this information, we can compute relative weights between potential subgoal positions using decreasing scores from *street* to *other*. In our system, the terrain class map is maintained independently to model the local vicinity of the robot, which is also explained in the previous chapter. In the forthcoming section, we explain, in detail, how we use the semantic information to make the global OSM plan useful to an autonomously navigating robot.

8.2 Subgoal Alignment

In this section, we explain how our approach combines the information from the two independent maps. From the global planner on OSM, we receive a series of subgoals $g_0^G, \ldots, g_n^G$, which are represented by their UTM-coordinates. We transform the received subgoals to our local odometry frame, leading to $g_0^L, \ldots, g_n^L$. With the subgoals in the robot frame, we calculate the difference $G_k^L = g_k^L - g_{k-1}^L$, which we represent by the orientation $\theta_k = \mathrm{atan2}(G_k^L(y), G_k^L(x))$ and the distance $d_k = \|G_k^L\|_2$, and model it as values of a random variable u_k. The goal of our work is to find the best configuration of the local subgoals, taking the observations from the 3d-lidar into account. To achieve this goal, we derive a probabilistic formulation in Section (8.2.1) and provide details of its components in Section (8.2.2) and



(a) (b) (c)

Figure 8.2: Subfigure (a) visualizes the semantic classification, distinguishing *street* (yellow), *dirt-road* (brown), *grass* (green), *vegetation* (purple) and *other* (pink). The red line visualizes the OSM representation without correction and the black line shows positions estimated by our approach. Subfigure (b) visualizes the weighting function, bright colors correspond to high weights. In subfigure (c), blue markers represent the distribution of the samples for our subgoals.

Section (8.2.3). Finally, to estimate the positions of the most recent subgoals we use this formulation in a Markov-Chain Monte-Carlo approach in Section (8.3).

8.2.1 Probabilistic Formulation

Our goal in this chapter is it to find the most likely arrangement of the local subgoals $\hat{g}_0, \ldots, \hat{g}_n$, such that every subgoal is on the street, given the relative positions $u_1, \ldots, u_n$ and the observations, which are the terrain classification results that are stored in the terrain class map, $\mathcal{Z}$.

$$\hat{g}_{0:n} = \operatorname{argmax}\ p\left(g_{0:n} \mid \mathcal{Z}, u_{1:n}\right) \tag{8.1}$$

This results in a $2(n+1)$-dimensional optimization problem and to the best of our knowledge there is no efficient method to directly solve it. Therefore, we aim for a recursive solution in order to reduce the dimensionality of the problem. By construction, we can assume that the subgoals $g_{0:n}$, given $u_{1:n}$, satisfy the Markov property and therefore form a Markov-Chain.

$$p(g_n \mid g_{0:n-1}, u_{1:n}) = p(g_n \mid g_{n-1}, u_n) \tag{8.2}$$

Moreover, we make the assumption that the cells of the terrain map are independent and therefore, given the subgoals g_i are well separated, the observations $z_{0:n}$ corresponding to the individual subgoals are independent given the subgoals.

$$p(z_{0:n} \mid g_0, \ldots, g_n) = \prod_{i \leq n} p\left(z_i \mid g_i\right) \tag{8.3}$$

Now, we can iteratively apply Bayes rule and combine it with Eq. (8.2) and Eq. (8.3) to derive the following factorization of the posterior.

$$\hat{g}_{0:n} = \operatorname*{argmax}_{g_{0:n}} p\left(g_0\right) \prod_i p\left(z_i \mid g_i\right) p\left(g_i \mid g_{i-1}, u_i\right) \tag{8.4}$$

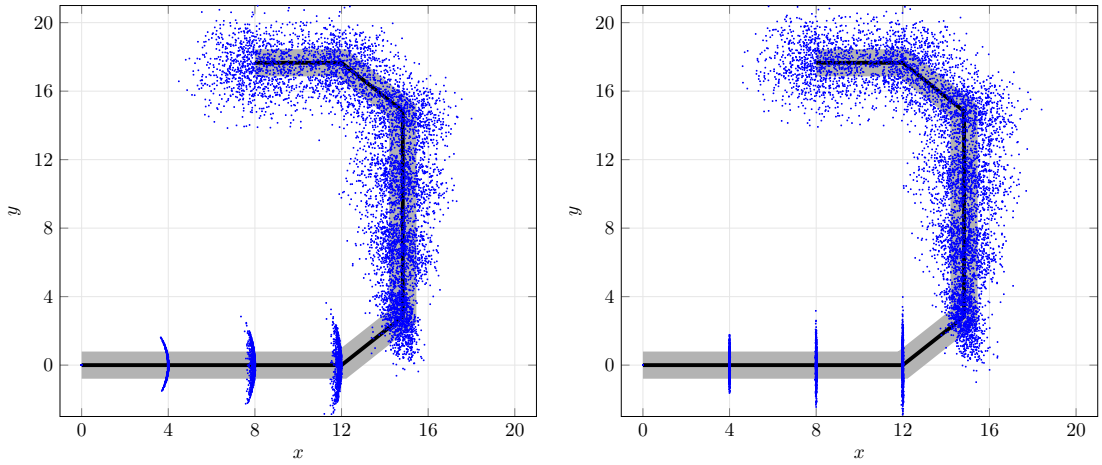


Figure 8.3: Visualization of the different process models. On the left is the well-known banana-shaped distribution [118], and on the right is the process model distribution with the discounting factor cos^{-1}, as introduced in Eq. (8.6).

Here, $p(g_0)$ is the prior distribution, $p(z_t \mid g_t)$ is the observation likelihood and $p(g_t \mid g_{t-1}, u_t)$ is the process model. Even though each factor can be computed efficiently now, the search space for a full trajectory remains too large. Therefore, we use an iterative solution to overcome this.

8.2.2 Process Model

We assume that the main source of error occurs perpendicular to the street direction given in OSM. We take this into account for our process model as follows:

$$g_k = h(g_{k-1}, u_k, \Delta), \text{ with} \tag{8.5}$$

$$h(g_{k-1}, u_k, \Delta) = g_{k-1} + \frac{d_k + \Delta d}{\cos(\Delta\theta)} \begin{pmatrix} \cos(\theta_k + \Delta\theta) \\ \sin(\theta_k + \Delta\theta) \end{pmatrix} \tag{8.6}$$

and Δ is a zero mean Gaussian noise variable.

$$\Delta = (\Delta d,\ \Delta\theta) \sim \mathcal{N}\left((0,\ 0), \begin{pmatrix} \sigma_d & 0 \\ 0 & \sigma_\theta \end{pmatrix}\right) \tag{8.7}$$

The additional factor $(cos(\Delta\theta))^{-1}$ accounts for the angular deviation and adjusts the distance, unfolding the commonly used banana-shaped distributions perpendicular to the street direction, as depicted in Figure (8.3). For the initial distribution, we assume that the direction of the street is known and therefore model

$$g_0 \sim \mathcal{N}\left(g_0^L, \Sigma_0\right) \tag{8.8}$$

with an appropriately rotated covariance matrix. The visualization of the sample distribution is shown in Figure (8.2)(c).

8.2.3 Measurement Model

To calculate the observation likelihood, we use the terrain class map, which we derive from the measurements as described in Section (8.1.2). For a quantification of the semantic information given a local subgoal g, $P(\mathcal{Z} \mid g)$, we map the terrain classes to costs, which increase as it becomes more unlikely to see a class on a road. More precisely, we define a function $f : \mathcal{T} \rightarrow \mathbb{R}^{+}_{>0}$ from the set of classes $\mathcal{T} = \{street, dirtroad, grass, vegetation, other\}$ to a discrete subset of the real half-line. Given the mapping f we can calculate a value for each cell $C \in \mathcal{Z}$ using the results of the fuzzy classification, by calculating the expected value of the function f over all terrain classes of cell C as

$$f^{E}(C) = \sum_{t \in \mathcal{T}} f(t) P(t \mid C), \tag{8.9}$$

where $P(t \mid C)$ is maintained by the terrain map. Now, as we aim to find evidence that a subgoal is placed on the trail, and since our map has a fixed resolution r, we do not only use a single cell but a square grid neighborhood with the center at the cell corresponding to the subgoal g. We refer to this neighborhood as $\mathbf{N}(g)$, which has a size of $(2N+1) \times (2N+1)$. With respect to this we compute the final costs as an equally weighted average of f^{E} over the cells in $\mathbf{N}(g)$.

$$f^{\mathbf{N}}(g) = \frac{1}{|\mathbf{N}(g)|} \sum_{C \in \mathbf{N}(g)} f^{E}(C) \tag{8.10}$$

To transform this cost value into a likelihood, we use an exponential distribution with parameter λ in our current implementation, stating the observation likelihood as

$$P(z \mid g) \sim \lambda \exp\left(-\lambda f^{\mathbf{N}}(g)\right). \tag{8.11}$$

Recall that we used the term of well separated subgoals in the derivation of Eq. (8.3), now it is clear that this characteristic is defined as $\mathbf{N}(g_i) \cap \mathbf{N}(g_j) = \emptyset, \forall i \neq j$. This also shows the tight connection between the assumption that observations are independent given the subgoals and the independence of the cells in the terrain map $\mathcal{Z}$. Figure (8.2)(b) visualizes the observation likelihood for every cell of the map.

8.3 Sequential Markov-Chain Monte-Carlo Sampling

To solve the optimization problem stated in Eq. (8.4), we employ a MCMC method for an incremental solution. In the real world, one of the main problems for our approach is visibility. Especially in forested environments, an intersection is often not observed before the robot actually arrives there. Moreover, the process model, given in Eq. (8.6), models the incremental, moderate error and cannot resolve a larger error, which typically occurs at intersections, in a single step. On the other hand, we are interested in subgoals that are as far away as possible, which is in favor of the local planner. Therefore, we do not only sample the most recent

subgoal, but a sequence of the most recent K subgoals.

$$p\left(g_{n-K+1:n} \mid \mathcal{Z}, u_{1:n}\right) = \int \Bigg(\prod_{i=n-K+1}^{n} p\left(z_i \mid g_i\right) p\left(g_i \mid g_{i-1}, u_i\right) \cdot p(g_{n-K+1} \mid g_{n-K}, u_{n-K}) p\left(g_{n-K} \mid \mathcal{Z}, u_{1:n-K}\right) \Bigg) \mathrm{d}g_{n-K} \tag{8.12}$$

Unfortunately, this increases the complexity of the distribution and especially in online scenarios, computation time is crucial. Therefore, we need to find a proper trade-off between accuracy and computational time, which we analyze in our experimental section. To complete the description of our approach, we briefly state the different steps, which are *sampling*, *weighting* and *resampling*. In our case, a sequence hypothesis G_j consists of samples of the most recent K subgoals, sequentially sampled from the proposal distribution

$$G_j \sim \prod_{i=n-K+1}^{n} p\left(g_{i,j} \mid g_{i-1,j}, u_i\right). \tag{8.13}$$

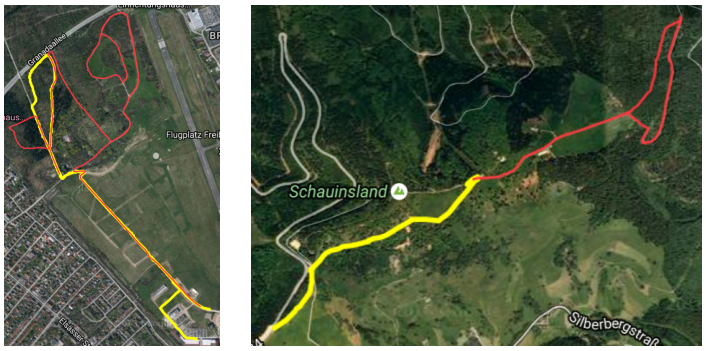
In practice, we sample a sequence hypothesis consecutively using Eq. (8.6) with samples from the distribution given in Eq. (8.7). Furthermore, we calculate the importance weight for G_j as

$$\mathbf{w}_j \propto \prod_{i=n-K+1}^{n} p\left(z_{i,j} \mid g_{i,j}\right). \tag{8.14}$$

Both computations can be done efficiently due to the recursive structure and the independence of observations. However, the high dimensionality of the state space requires substantially more samples. After the weighting, we calculate the so-called number of effective particles $n_{\mathit{eff}} = \left(\sum \mathbf{w}_j^2\right)^{-1}$. If n_{eff} is fewer than half of the number of samples, we perform a resampling procedure in which we use stochastic universal sampling as introduced by Baker [9]. For the current state, we use the expectation of the sampled trajectory.

8.4 Experiments

We evaluated our approach using real world experiments, with the robot shown in Figure (8.1), which navigated autonomously in previously unexplored environments as well as with data that was recorded navigating the robot via remote control. We give a detailed overview of our system settings in Section (8.4.1). To evaluate our approach, we measure the displacement of the proposed subgoals relative to the true path in our local frame. Thereby, we will investigate the influence of the sequence length K and the number of samples we use to approximate the distribution in Eq. (8.12) in Section (8.4.2). As intersections are critical situations that need to be resolved correctly for the success of autonomous navigation in real world applications, we analyze the behavior of our approach at such points in Section (8.4.3). Especially in online settings, runtime becomes important and results concerning this are discussed in Section (8.4.4). Finally, we reason limitations of the approach and discuss cases in which our approach may fail to find the correct path in Section (8.4.5).



Mooswald Schauinsland

Figure 8.4: GPS tracks of the datasets we used for evaluation on an aerial image. Autonomous trajectories are colored in yellow, remotely controlled trajectories are colored in red. Mooswald datasets are collected nearby our campus and Schauinsland datasets are collected from a mountain with the same name, near Freiburg.

8.4.1 System Setting

For all experiments we used data from the same robot, equipped with a Velodyne HDL-64E 3d-lidar sensor, Kalman-filtered GPS and internal odometry aided by gyroscope and IMU. The terrain classification integrates measurements for a driven distance of $d = 0.5m$ between the feature computation steps. The resolution of the grid was set to $0.2m$ with a rolling grid of size 300×300, see Section (7.2). Subgoals were generated with a distance of four meters, and the map service we used was OpenStreetMap. For all experiments, we set the neighborhood size $N = 2$, which corresponds to a patch size of $1m \times 1m$, and $\lambda = 4$, see Section (8.2.3). The terrain cost function was set to $\{street : 0.1, dirt : 0.2, grass : 0.4, vegetation : 0.8, other : 1.0\}$. If a cell without observations was requested, a default value, equal to that of $other$, was returned. For the evaluation, we considered datasets from two locations, one nearby our campus (Mooswald) and one from a mountain near Freiburg, the Schauinsland. The GPS-tracks of the trajectories are visualized on aerial images in Figure (8.4). All in all, we evaluated data with an overall trajectory length of 14 kilometers.

8.4.2 Performance

First to mention, we successfully applied our approach to a real robot, navigating autonomously in both areas, see Figure (8.4), with an overall trajectory length of 5.2 km, in which we used 1,000 samples and a sample sequence length of four. Our approach made the difference between the robot leaving the road and getting stuck, and successfully reaching its goal. To quantify these qualitative results, we calculate the root mean squared error (RMSE) measuring the distance of the subgoals, to the middle of the desired trails in the local frame. Due to the lack of ground truth data, we approximate the error utilizing the knowledge that the robot was driving near the middle of the trails most of the time. Given a sequence of robot poses $p_{1:m}$ and a sequence of subgoals $g^K_{1:n}$, corrected with a sample sequence length K, and assuming a medium street-width s of $3m$, we calculate the RMSE as:

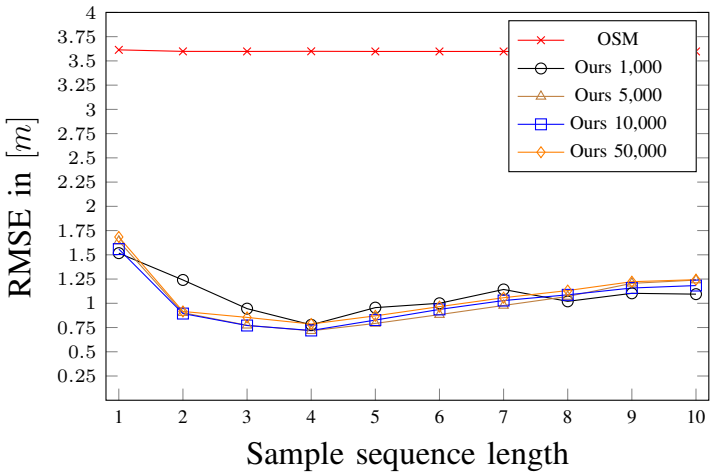


Figure 8.5: Average RMSE over ten instances for Ours (N) and the uncorrected data from OSM, where N denotes the number of samples. Our approach achieves similar RMSE for different sample sizes and the minimum for the sequence length of four.

$$RMSE_K = \sqrt{\frac{\sum_{i \leq n} \max\left(\min_{l \leq m} \|g_i^K - p_l\|_2 - \frac{s}{2},\ 0\right)^2}{n}} \tag{8.15}$$

For the computation of the RMSE, we concatenate the poses and subgoals of all trajectories. We examine samples sizes between 1,000 and 50,000 and vary the sequence length from one to ten. Due to the randomized nature of our approach, we run ten instances for each setting and calculate the RMSE for each instance.

We compare the average RMSE of the proposed method with different settings, to the raw data without correction from OSM, the results are depicted in Figure (8.5). The sequence length of four achieves the minimum RMSE for all the sample sizes, ranging from $0.78m$ for 1,000 samples to $0.72m$ for 10,000 samples. Compared to the raw subgoals from OSM, which exhibits a RMSE of $3.6m$, our method yields an error reduction of up to 80%. The results indicate, that for longer sequence lengths the dimension for the sampling is too large, such that even when using 50,000 samples the RMSE is higher than with fewer particles and shorter sequence sample lengths. In the forthcoming section, in which we inspect the results of the approach at intersections, we give an explanation for this behavior.

In Figure (8.6) we compare the errors at each subgoal, corrected with our approach using 1,000 samples and a sequence length of four to the errors of the raw subgoals from OSM. The errors of OSM exceed ten meters several times, while the errors of our approach are typically below $1m$; except for one situation, which we explain in Section (8.4.5), the single peaks that exhibit a higher error for our approach, stems from crossings where the area of possible street points is large and the robot often takes a shortcut relative to the shape representation in OSM, which is not considered by our error measurement. Nevertheless, comparing the errors from our approach to OSM in a single tailed paired student's-t-test, the better performance of our approach was significant ($\alpha = 0.05$) for all settings and instances.

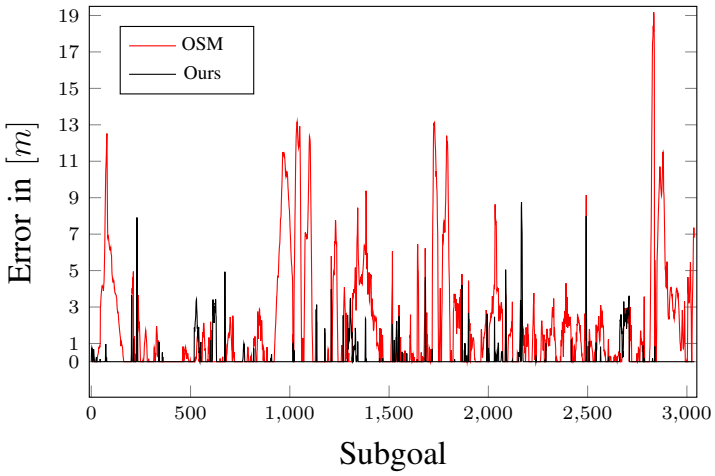


Figure 8.6: The error of our approach compared to uncorrected data from OSM for every subgoal used in our evaluation set. For our approach, we use 1,000 samples and a sequence length of four in this case.

8.4.3 Intersections

In the previous section, we measured the average displacement between the middle of the trails and the estimated position of the subgoals over the whole trajectory. This measure shows a similar behavior for a wide range of settings. This is because keeping the subgoals on the trails in the local frame, works robustly for most of the provided settings. However, in practice, the most critical situations occur at intersections, as there the robot will get stuck if it cannot find the desired path. In this section, we focus on such situations and take a closer look at the results of our approach at six intersections. Successful examples at those intersections, as achieved by our approach with 1,000 samples and a sample sequence length of four, are depicted in Figure (8.7). For this evaluation, we again recorded the results of ten instances of our approach and checked whether or not it could estimate feasible subgoals on the desired trails. We examine situations in which the intersections in OSM are located too far ((a) and (b)) and too close ((c) and (d)) relative to the driving direction, as well as an intersection that is exposed to large shape approximation errors, intersection (e), and for validation: an intersection that is well aligned, intersection (f).

The mapped nodes in OSM at the intersections (a) and (b) are located farther in the map than they appear in the local frame. As the robot has already passed the intersection while following the subgoals, the terrain map contains rich information about the intersection and in this situation the subgoals could be aligned correctly for all settings. Notably, intersection (b) offers several options for failure, as our approach may detect the wrong street as the desired path. However, we passed this intersection several times from both directions and never observed the approach to fail there.

In contrast to this case, at intersections (c) and (d), the junction is located closer in the map than in the local frame. This situation is more critical to our approach, as by visibility constraints, the terrain class map may not be as informative as in the former case. When the robot reaches the presumed point of the junction, only a partial view of the intersecting track is available. This situation is more severe at intersection (d), as the OSM is more off there. In these cases, we observe different success rates for different settings of our approach. When

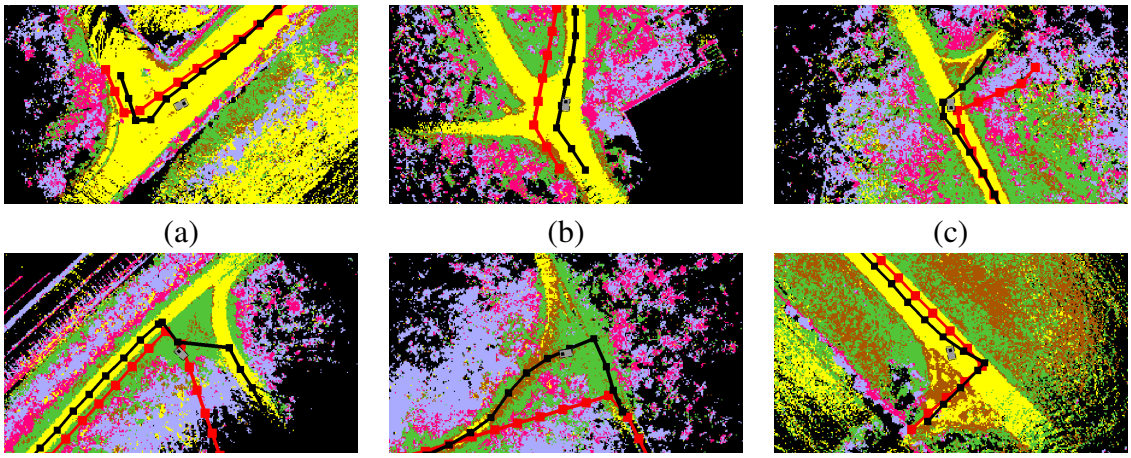


Figure 8.7: Intersections constitute challenging situations for our approach. For intersections (a) and (b) the junction point is located too far in the map (red), a situation that is typically well handled by our approach (black). At intersections (c) and (d), the junction point appears too early in the map. These situations are challenging as there is only partial visibility and not all parameter settings can resolve this. At intersection (e) we face substantial approximation errors and for intersection (f) the map is well aligned with the local frame.

the chosen sequence length K is too small, *e.g.,* $K = 1$, the degree of freedom is not sufficient to successfully resolve the situation. On the other hand, when we set K too large, the number of samples may not be sufficient to explore the full state space, leading to inferior results. More specifically, with a sequence length $K = 1$, our approach can not resolve situations at both intersections, no matter which sample size was used. When we applied a sequence length between two and four, $K \in \{2, 3, 4\}$, the results improve substantially. At intersection (c) our approach was able to successfully find the desired path, with qualitatively better results when the K was set to two or three. However, at intersection (d) with $1,000$ samples the sequence length of four was superior, with this setting, our approach found the desired path in nine of the ten trials, while it was only found in four cases when K was set to two. Raising the number of samples improves the results, as *e.g.,* for $K = 2$ and a sample size of $50,000$, our approach was successful in nine of the ten trials. For a larger sequence length, we tested up to $K = 6$, and the success rate drops, as the space to explore is too large. Even with $50,000$ particles, no successful trial was observed at intersection (d).

At intersection (e) we see that the OSM approximation of the path does not match the true conditions, as the trail takes a short detour to the left. In this case, the path is erroneously approximated in the OSM but the intersecting street is matched well, our approach can resolve this situation successfully with each setting. Finally, intersection (f) is well aligned in the OSM and we have included it in this evaluation as a verification example. As already expected from the former cases, this situation is handled successfully for each of the settings.

The results in this section explain the shape of the RMSE curves of the former section. As visibility is good and differences occur incrementally, each setting achieves relatively good results; however, in cases like intersection (c) and (d) the sample sequence length and the number of particles matters. As behind the crossing our method usually recovers, and this results in only a moderate increase of the RMSE.

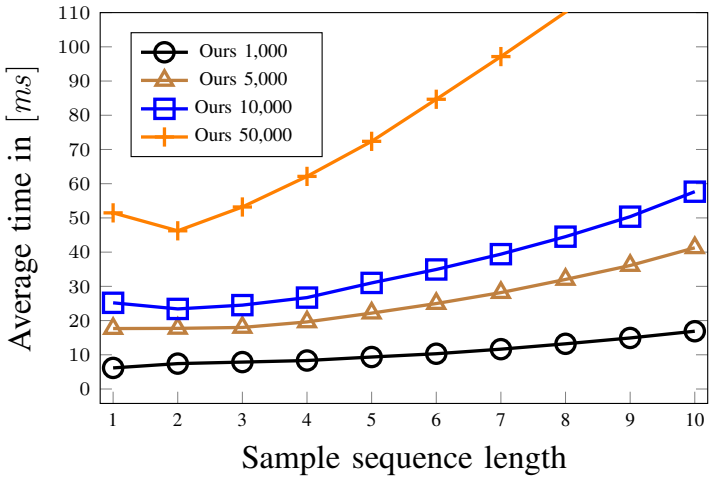


Figure 8.8: Average runtime for calculating an update for Ours N, where N denotes the number of samples.

To conclude, the results in real world navigation tasks depend on a complex interaction between: the sensor visibility, the behavior of the local planner, and the implementation of how the global planner sends a new subgoal. In our implementation, we found that sequence lengths between two and four show good results, and we chose four when the robot operates autonomously, as, in general, the results look qualitatively best in this case.

8.4.4 Runtime

Execution time of the approach is important, especially when deployed in a real autonomous system. Our approach is efficiently implemented in C++ and the runtime was measured for the different settings on a single core of an Intel(R) Core(TM) i7-2700K CPU @ 3.50GHz. The reported computation time does not include updates of the terrain map, which were computed every 0.5m in a separate thread and took on average 0.3s. The runtime is almost linear in the number of samples and the sequence length, see Figure (8.8). With 1,000 samples and a sequence length of four, the average update time is 8.3 ms, whereas this increases to 62 ms with 50,000 samples. Our choice of parameters is fully supported by our runtime and accuracy evaluation. This setting of the system yields realtime performance, which is integral for autonomous navigation in real world scenarios.

8.4.5 Limitations

The performance of our approach depends on the quality and correctness of the terrain classification. In the remotely controlled experiments, we encountered a situation in which the path was covered by grass and several meters away the classifier found evidence for a dirt road; accordingly, the estimation of the subgoals drifts towards that region, see the left image in Figure (8.9). Nevertheless, as soon as the path is distinguishable from the surroundings some meters later, our approach can successfully recover from this failure, see the right image in Figure (8.9). This means that an autonomously navigating robot may deviate from the desired path in such a region and return to it as soon as our method recovers from its failure.

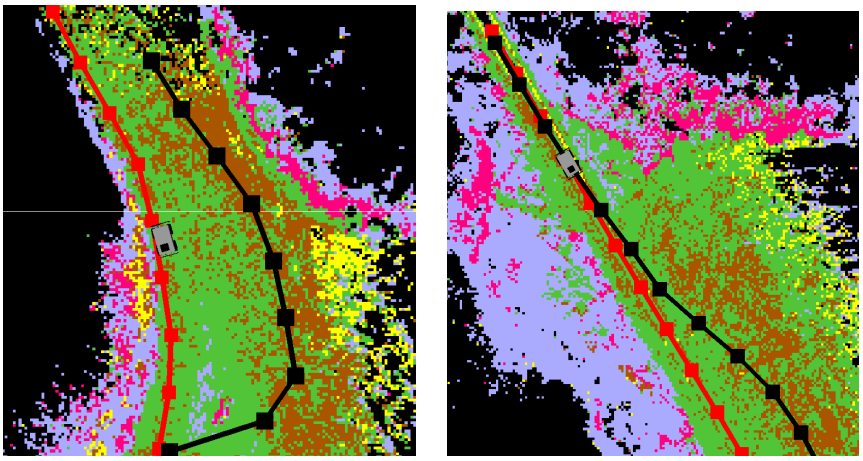


Figure 8.9: An example where our approach fails to find the correct path (left), due to an ambiguous classification outcome. The path from OSM (red) matches precisely. Our method (black) estimates the subgoals at an area that is classified as dirt road, whereas the actual path is classified mostly as grass. As soon as the classification improves, our method recovers (right).

Another potential source of errors was already identified in the previous section, as we have seen there that intersections that are mapped earlier than they appear are difficult to resolve. If the intersection node is placed too far off, our approach as presented, may not find the junction point, as it is out of sight. However, this could be solved by detecting such situations and then employing an active search for the junction point.

8.5 Related Work

In recent years, publicly available maps like OSM have gained interest in robotic applications and several authors have utilized such maps for localization purposes by matching the road network against data observed by the robot. For example, Floros et al. [31] use the Chamfer-distance to compare chunks of visual-odometry trajectories from the robot with the network structure of OSM. Mandel and Birbach [69] fuse GPS and odomtery data for OSM localization. Ruchti et al. [100] segment single 3d-lidar scans into *street* and *no street* areas and match the street segments against the road-network, while also taking into account the negative information of *no street*. Whereas these approaches show an impressive robustness for global localization, they only provide limited accuracy with localization errors typically exceeding several meters. In the context of our work, this error is too large.

Other approaches use maps of geotagged images as a source for localization and match information retrieved from images against the images in the map. There are also feature-based approaches that match panoramas, *e.g.,* from GoogleStreetView, against the local camera images [2, 123, 138]. However, these approaches assume a rich image-based map, which is not available everywhere, especially not in outer-urban and forested environments. Another, camera-based, approach by Radwan et al. [96] extracts textual information from images and matches it against georeferenced text of the map for localization, a source of information that is sparse in outer-urban environments.

More similar to our approach is Máttyus et al. [71], who combine monocular aerial images with stereo images from a robot, estimating their pose in OSM and enhancing the map with semantic information that is derived from deep learning image classification. The authors assume the error of the OSM alignment to be perpendicular to the road direction, an assumption that we make in our approach, as well. However, they also assume that the error is within a fixed bounds of four meters, which seems sufficient for the urban dataset from KITTI. In our case, we cannot restrict the error to fixed bounds, since in forested environments we face errors of up to ten meters and more.

Another approach by Irie et al. [49] extracts boundaries from the structural representation of GoogleMaps and projects them to images recorded by the robot. The matching score is based on squared-loss mutual information between the image features and projected labels from the map. This approach is dedicated to urban environments and needs manual preprocessing for map conversion.

An approach to OSM navigation was recently presented by Hentschel and Wagner [46] where 3d-lidar data is matched against the rich information of landmarks also provided by the map, *e.g.,* the compendium of buildings. The approach is particularly suited for urban environments and cannot directly be applied to outer-urban environments, as they are not guaranteed to contain a sufficient amount of the required features.

From the related work, it is clear that the uncertainty in the quality of OSM is a well-known problem. In the context of urban environments, approaches often assume error bounds ranging from two to four meters, which is typically a strong assumption for less frequently used outer-urban environments. Haklay [44] evaluated the quality of OSM with a focus on London and England. His analysis shows that positions recorded by Ordnance Survey (OS) at different areas in London are on average six meters away from the corresponding positions in OSM. He also compared the overlap of smaller roads in OS and OSM with results varying from 5% to 100%. This shows the varying quality of such maps and also provides a clear motivation for our work presented in this chapter.

8.6 Conclusions

In this chapter, we presented an approach to using publicly available map data from OpenStreetMap for autonomous navigation in outer-urban environments at previously unseen locations. Our approach relies only on perceptions of the local vicinity of the robot, and therefore, requires constant memory only. Furthermore, it is highly efficient and can operate online on a robot navigating autonomously in real world settings. The proposed approach needs neither an accurate map nor an accurate localization on the map for autonomous navigation. In comparison to the raw data from OSM, the corrections of our approach are significant and the differences between the trails in the map and the trails in the local frame exceed ten meters several times.

Future Work

In future work, we can imagine investigating deep learning methods for terrain classification to further increase the robustness of our approach. On the planning level, a more sophisticated method that actively searches for the crossing in the case of larger errors, which should

increase the robustness, since it would overcome the visibility constraints of our approach. Most interesting would be to apply an optimization scheme in order to improve the OSM itself. This could be a great improvement, especially in the less frequently visited outer-urban areas.

Chapter 9

Conclusion

In this thesis, we discussed a number of techniques for highly relevant topics in the field of Autonomous Navigation. Our contributions are dedicated to the problems of: SLAM, localization facing changes in the environment, traversability analysis, and the interpretation and usage of third party maps. Each contribution covers a specific problem for autonomously navigating robots and has advanced the state-of-the-art in respective fields.

First, we addressed the problem of large scale SLAM under limited resources, specifically limited main memory. We proposed a hierarchical construction, subsequently dividing the large problem into independent sub-problems, each of bounded size. Thereby, we efficiently approximate the marginal distribution of nodes using a tree. The loss of accuracy is moderate, as shown in the experiments. Moreover, we showed our this approach needs significantly less memory than state-of-the-art exact solvers and outperforms other approximate solvers in terms of accuracy and memory consumption. When the main memory is limited, our approach computes the solution faster than state-of-the-art solvers, which shows the advantage of the hierarchical construction when compared to the paging mechanism of the operation system.

Once a map is computed, the purpose is to achieve a robust localization for a mobile robot. A remaining challenge in the area is to cope with changes in the environment, which may occur for multiple reasons. Subsequently, we address such challenges in two different domains. At first, we presented an approach to robustly localize a mobile robot relative to an object, *e.g.,* a table, which is dedicated to industrial environments, using a 2d-laser sensor. We proposed an extension to the popular ICP-algorithm, which takes multiple reference frames into account and computes a joint solution for this task. Our approach can cope with moderate movements of predefined objects and achieves an accuracy that is considered sufficient for industrial applications. The other problem we addressed in this domain is dedicated to robust visual localization, facing severe changes in the appearance of the environment. The goal of the presented approach is to enable visual localization, given two image sequences that are recorded during continuous runs of a mobile robot with a substantial time difference between them. During the time between the first and second run, the appearance of the environment is substantially affected by seasonal changes. The presented approach employs a Bayesian filter, taking the sequential nature of the recorded images into account, and a sensor model, which is based on HOG-features, in order to infer the correspondences between the two image sets. As shown in extensive experiments, our approach outperforms state-of-the-art techniques and can handle more complex datasets, for which other approaches cannot return meaningful results.

We then discussed the safety relevant topic of traversability analysis. This topic is of utmost importance to autonomously navigating robots, since it is responsible for the safe success of autonomous tasks. Thereby, it needs to take both sides into account: safety and mobility.

First, we presented a learning approach to infer traversability characteristics of a robot from human demonstrations. In this approach, we proposed a framework that successfully inferred a traversability model from only positive demonstrations of traversable ground. Our method does not require the users to have special training to create traversability models for robots. In the experimental evaluation, we showed that the same approach is applicable to different robots and we achieved good results for traversability with respect to the safety relevant measures. However, the approach assumes traversability as a cell-wise static property of the environment and provides only slow updates, which is sufficient if no dynamics are present in the environment. We tackled this shortcoming in the following chapter, where we presented an approach to fast and efficient obstacle detection. To achieve this, we combine slowly updated semantic terrain information with quickly updated information to compute geometric features. We showed that the overall computational effort of the full navigation system can be accomplished by fewer than two cores, despite the complex nature of the outer-urban outdoor environment, in which the robot navigated autonomously. For both approaches, the perceptual sensor was a 3d-lidar, which was used for both tasks: inferring the terrain type and computation of the geometric features.

Finally, we presented an approach to autonomous outer-urban navigation using the map data from OpenStreetMap. We showed that semantic terrain information can be used to successfully cope with the errors of the system, introduced by inaccurate map data and inaccurate GPS measurements. This approach makes it possible to perform navigation tasks at previously unseen locations, which is not possible with common autonomous navigation systems. This navigation approach and the online obstacle detection were successfully deployed on a real autonomously navigating robot, which was developed within the framework of the EU-Project LifeNav.

To sum up, in this thesis, we presented a number of techniques that advanced the state-of-the-art in different areas of Autonomous Navigation. The presented approaches are designed for real autonomy, which means that all computations can be done on-board with a standard CPU and without the need for communication between robots or other external computation power.

Future Work

As it is the nature of scientific problems, there are manifold possible starting-points for future work. We briefly show some examples that, from our point of view, would be the most interesting.

For the memory efficient SLAM approach, it could be beneficial to recover the covariances of the final estimate. Therefore, it is probably necessary to change the approximation of the marginal distributions; however, results towards this would make the approach stronger. So far, the approach is used as a batch approach, an incremental version of which would make the approach applicable to online settings with low computational resources.

Recently, the methods using deep neural networks have gained in popularity and it looks like that they have started to conquer more and more topics related to autonomous navigation. Even though the limits of such approaches need to determined, they could be integrated within the context of this work at several points. For example, one could use features computed by convolutional neural networks for the visual localization approach, since they seem to provide more expressiveness than previous state-of-the-art features. They could also be integrated

within the traversability approach, *e.g.,* to aid the terrain classification, or to directly reason about obstacles.

Finally, it seems possible to extend the method for outer-urban OSM navigation to correct the OSM data, based on the estimates made within the approach. GPS-based methods for crowded environments already exist; however, they rely on a large amount of data that is not available for less frequently visited outer-urban areas.

List of Figures

3.1 Hierarchical graph partitioning . . . 13
3.2 Construction of the approximate separator graph . . . 16
3.3 Hierarchies computed by SMF for different datasets. . . . 17
3.4 Manhattan world dataset example. . . . 19
3.5 Memory consumption comparison. . . . 20
3.6 Runtime comparison with memory constraints. . . . 21

4.1 Illustration of the relative localization problem. . . . 26
4.2 Illustration of inconsistent global and relative pose. . . . 27
4.3 Illustration of a segmented point-cloud . . . 28
4.4 Graph representation of ICP with multiple reference frames. . . . 29
4.5 Graph representation for the reference point cloud registration. . . . 31
4.6 Graph representation for the local point cloud registration. . . . 32
4.7 Quantitative results of the simulation experiments . . . 33
4.8 Map of the simulation environment. . . . 34
4.9 Image of the real world environment. . . . 35
4.10 Quantitative results of the real world experiments . . . 36

5.1 Local gradient information . . . 40
5.2 Illustration of the similarity normalization . . . 43
5.3 Illustration of whitening transformation . . . 47
5.4 True positives of our localization approach. . . . 49
5.5 Successful matches for the VPRiCE dataset . . . 50
5.6 Results for the scattered trajectories . . . 52
5.7 Results for the connected trajectories . . . 53
5.8 Results for NewCollege . . . 54
5.9 Evaluation varying the sequence length . . . 55

6.1 Two robots with different capabilities . . . 60
6.2 Illustration of training and evaluation trajectories as used with Viona . . . 63
6.3 Illustration of training and evaluation trajectories as used with Obelix . . . 64
6.4 Qualitative results for learned traversability maps . . . 65
6.5 Qualitative results for learned traversability maps . . . 68
6.6 Evaluation of the mixed outer urban and urban scenario . . . 69

7.1 Illustration of different terrain types . . . 74
7.2 Example for geometric measures . . . 75
7.3 Geometric feature calculations . . . 77
7.4 Obstacle probabilities . . . 79

7.5 Qualitative evaluation of the terrain classification 81
7.6 Illustration to show necessity for terrain adaption 82
7.7 GPS-track of the evaluation trajectory 85
7.8 CPU usage of our approach during the real world experiment. 85
7.9 Average KL divergence comparison 86
7.10 Differences between approximations 87

8.1 Motivation for OSM navigation 92
8.2 Approach overview 95
8.3 Difference between the process models 96
8.4 GPS tracks of the trajectories for the evaluation 99
8.5 Average RMSE 100
8.6 Error at each time step 101
8.7 Challenges at intersections 102
8.8 Average runtime 103
8.9 Example for deviation and recovery 104

List of Tables

3.1 Small Memory Footprint Mapping (χ^2, Time) 22
3.2 Small Memory Footprint Mapping (RPE) 23

5.1 F_1-Scores comparison 51

6.1 Evaluation for Obelix on the campus trajectory. 70

7.1 Terrain class dependent parameters 83
7.2 Terrain Classification Confusion Matrix 84

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Tran. on Knowledge and Data Engineering (TKDE)*, 17(6):734–749, 2005.

[2] P. Agarwal, W. Burgard, and L. Spinello. Metric localization using google street view. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[3] A. Ahmad, G. D. Tipaldi, P. Lima, and W. Burgard. Cooperative Robot Localization and Target Tracking based on Least Square Minimization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

[4] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[5] A. Angelova, L. Matthies, D. Helmick, and P. Perona. Learning and prediction of slip from visual information. *Journal on Field Robotics*, 2007.

[6] 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 *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2002.

[7] P. Babahajiani, L. Fan, and M. Gabbouj. Object recognition in 3d point cloud of urban street scene. In *Computer Vision-ACCV 2014 Workshops*, 2014.

[8] H. Badino, D. Huber, and T. Kanade. Real-time topometric localization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[9] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proc. of the Second Int. Conf. on Genetic Algorithms*, 1987.

[10] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[11] M. Bennewitz, C. Stachniss, W. Burgard, and S. Behnke. Metric localization with scale-invariant visual features using a single perspective camera. In *European Robotics Symposium 2006*. Springer, 2006.

[12] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Tran. on Pattern Analysis and Machine Intelligence (TPAMI)*, 14(2):239–256, 1992.

[13] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun. Towards object mapping in non-stationary environments with mobile robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2002.

[14] I. Bogoslavskyi, O. Vysotska, J. Serafin, G. Grisetti, and C. Stachniss. Efficient traversability analysis for mobile robots using the kinect sensor. In *Proc. of the European Conference on Mobile Robots (ECMR)*, 2013.

[15] M. Bosse, P. M. Newman, J. J. Leonard, and S. Teller. An ATLAS framework for scalable mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2003.

[16] L. Breiman. Random forests. *Machine learning*, 2001.

[17] G. Broten, D. Mackay, and J. Collier. Probabilistic obstacle detection using 2 1/2 d terrain maps. In *Proc. of the 9th Conf. on Computer and Robot Vision (CRV)*, 2012.

[18] C. Cadena, D. Gálvez-López, J. D. Tardós, and J. Neira. Robust place recognition with stereo sequences. *IEEE Tran. on Robotics (T-RO)*, 28(4):871–885, 2012.

[19] N. Carlevaris-Bianco and R. Eustice. Learning visual feature descriptors for dynamic lighting conditions. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[20] T. Chang, T. Hong, S. Legowik, and M. Abrams. Concealment and obstacle detection for autonomous driving. In *Proc. of the Int. Assoc. of Science and Technology for Development-Robotics and Application*, 1999.

[21] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Tran. on Information Theory*, 14(3), 1968.

[22] W. Churchill and P. Newman. Practice makes perfect? managing and leveraging visual experiences for lifelong navigation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[23] M. Cummins and P. Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *Int. Journal of Robotics Research*, 27(6):647–665, 2008.

[24] M. Cummins and P. Newman. Highly scalable appearance-only SLAM – FAB-MAP 2.0. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[25] E. Cuthill. Several strategies for reducing the bandwidth of matrices. In *Sparse matrices and their applications*. Springer, 1972.

[26] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Tran. on Pattern Analysis and Machine Intelligence (TPAMI)*, 29 (6), 2007.

[27] F. Denis, R. Gilleron, M. Tommasi, et al. Text classification from positive and unlabeled examples. In *Proc. of the 9th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*, 2002.

[28] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard. Motion-based detection and tracking in 3d lidar scans. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2016.

[29] C. Elkan and K. Noto. Learning classifiers from only positive and unlabeled data. In *Proc. of the 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2008.

[30] C. Estrada, J. Neira, and J. Tardós. Hierachical SLAM: Real-time accurate mapping of large environments. *IEEE Transactions on Robotics*, 21(4):588–596, 2005.

[31] G. Floros, B. van der Zander, and B. Leibe. Openstreetslam: Global vehicle localization using openstreetmaps. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

[32] D. Fox. Adapting the sample size in particle filters through kld-sampling. *Int. Journal of Robotics Research*, 22(12):985–1003, 2003.

[33] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1999.

[34] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2(3):24–33, 2003.

[35] U. Frese. Treemap: An $o(logn)$ algorithm for indoor simultaneous localization and mapping. *Autonomous Robots*, 21(2):103–122, 2006.

[36] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Tran. on Robotics (T-RO)*, 21(2):196–207, 2005.

[37] A. George and J. W. Liu. The evolution of the minimum degree ordering algorithm. *Siam Review*, 31(1):1–19, 1989.

[38] A. Glover, W. Maddern, M. Milford, and G. Wyeth. FAB-MAP + RatSLAM: Appearance-based slam for multiple times of day. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[39] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Tran. on Robotics (T-RO)*, 23(1):34–46, 2007.

[40] G. Grisetti, C. Stachniss, and W. Burgard. Non-linear constraint network optimization for efficient map learning. *IEEE Tran. on Intelligent Transportation Systems*, 10(3), 2009.

[41] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[42] G. Grisetti, R. Kümmerle, and K. Ni. Robust optimization of factor graphs by using condensed measurements. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[43] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE Int. Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, 1999.

[44] M. Haklay. How good is volunteered geographical information? a comparative study of OpenStreetMap and Ordnance Survey datasets. *Environment and Planning B: Planning and Design*, 37(4):682–703, 2010.

[45] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4), 2008.

[46] M. Hentschel and B. Wagner. Autonomous robot navigation based on OpenStreetMap geodata. In *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2010.

[47] A. Howard, M. Matarić, and G. Sukhatme. Relaxation on a mesh: a formalism for generalized localization. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2001.

[48] A. Howard, M. Turmon, L. Matthies, B. Tang, A. Angelova, and E. Mjolsness. Towards learned traversability for robot navigation: From underfoot to the far field. *Journal on Field Robotics*, 2006.

[49] K. Irie, M. Sugiyama, and M. Tomono. A dependence maximization approach towards street map-based localization. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[50] S. Jain and B. Argall. Automated perception of safe docking locations with alignment information for assistive wheelchairs. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[51] D. Joho, C. Stachniss, P. Pfaff, and W. Burgard. Autonomous exploration for 3D map learning. In *Fachgespräche Autonome Mobile Systeme (AMS)*, 2007.

[52] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Fast incremental smoothing and mapping with efficient data association. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[53] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Int. Journal of Robotics Research*, 31(2):216–235, 2012.

[54] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[55] G. Karypis and V. Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

[56] H. Kretzschmar and C. Stachniss. Information-theoretic compression of pose graphs for laser-based slam. *Int. Journal of Robotics Research*, 31(11):1219–1230, 2012.

[57] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[58] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. A navigation system for robots operating in crowded urban environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

[59] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. Autonomous robot navigation in highly populated pedestrian zones. *Journal on Field Robotics*, 32(4): 565–589, 2015.

[60] S. Kuthirummal, A. Das, and S. Samarasekera. A graph traversal based algorithm for obstacle detection using lidar or stereo. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[61] I. Kweon and T. Kanade. High resolution terrain map from multiple sensor data. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 1990.

[62] S. Laible, Y. N. Khan, K. Bohlmann, and A. Zell. 3d lidar-and camera-based terrain classification under different lighting conditions. In *Autonomous Mobile Systems 2012*. Springer, 2012.

[63] J.-F. Lalonde, N. Vandapel, D. F. Huber, and M. Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal on Field Robotics*, 23(10):839–861, 2006.

[64] J. Larson and M. Trivedi. Lidar based off-road negative obstacle detection and analysis. In *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2011.

[65] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(4):376–382, 1991.

[66] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110, 2004.

[67] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333–349, 1997.

[68] W. Maddern, A. Stewart, C. McManus, B. Upcroft, W. Churchill, and P. Newman. Illumination invariant imaging: Applications in robust vision-based localisation, mapping and classification for autonomous vehicles. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2014.

[69] C. Mandel and O. Birbach. Localization in urban environments by matching sensor data to map information. In *Proc. of the European Conference on Mobile Robots (ECMR)*, 2013.

[70] R. Manduchi, A. Castano, A. Talukder, and L. Matthies. Obstacle detection and terrain classification for autonomous off-road navigation. *Autonomous Robots*, 18(1):81–102, 2005.

[71] G. Máttyus, S. Wang, S. Fidler, and R. Urtasun. HD Maps: Fine-grained road segmentation by parsing ground and aerial images. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[72] J. Maye, R. Kaestner, and R. Siegwart. Curb detection for a pedestrian robot in urban environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[73] M. W. McDaniel, T. Nishihata, C. A. Brooks, P. Salesses, and K. Iagnemma. Terrain classification and identification of tree stems using ground-based lidar. *Journal on Field Robotics*, 29(6):891–910, 2012.

[74] C. McManus, W. Churchill, W. Maddern, A. Stewart, and P. Newman. Shady dealings: Robust, long- term visual localisation using illumination invariance. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2014.

[75] C. McManus, B. Upcroft, and P. Newman. Learning place-dependant features for long-term vision-based localisation. *Autonomous Robots*, 39(3):363–387, 2015.

[76] M. Milford and G. Wyeth. Persistent navigation and mapping using a biologically inspired slam system. *Int. Journal of Robotics Research*, 29(9):1131–1153, 2010.

[77] M. Milford and G. F. Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[78] M. Montemerlo, S. T. D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, 2003.

[79] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1985.

[80] J. J. Moré. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical Analysis*. Springer, 1978.

[81] L. Murphy and P. Newman. Risky planning on probabilistic costmaps for path planning in outdoor environments. *IEEE Tran. on Robotics (T-RO)*, 29(2):445–457, 2013.

[82] T. Naseer, L. Spinello, W. Burgard, and C. Stachniss. Robust visual robot localization across seasons using network flows. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2014.

[83] T. Naseer, M. Ruhnke, C. Stachniss, L. Spinello, and W. Burgard. Robust visual SLAM across seasons. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[84] T. Naseer, B. Suger, M. Ruhnke, and W. Burgard. Vision-based markov localization across large perceptual changes. In *Proc. of the European Conference on Mobile Robots (ECMR)*, 2015.

[85] T. Naseer, B. Suger, M. Ruhnke, and W. Burgard. Vision-based markov localization for long-term autonomy. *Robotics & Autonomous Systems*, 89:147–157, 2017.

[86] P. Neubert and P. Protzel. Local region detector+ cnn based landmarks for practical place recognition in changing environments. In *Proc. of the European Conference on Mobile Robots (ECMR)*, 2015.

[87] P. Neubert, N. Sünderhauf, and P. Protzel. Superpixel-based appearance change prediction for long-term navigation across seasons. *Robotics & Autonomous Systems*, 69: 15–27, 2015.

[88] K. Ni and F. Dellaert. Multi-level submap based slam using nested dissection. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[89] K. Ni, D. Steedly, and F. Dellaert. Tectonic SAM: Exact; Out-of-core; Submap-based SLAM. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[90] K. E. Olin and D. Y. Tseng. Autonomous cross-country navigation: an integrated perception and planning system. *IEEE Expert*, 6(4):16–30, 1991.

[91] M. Ollis, W. H. Huang, and M. Happold. A bayesian approach to imitation learning for robot navigation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[92] E. Olson, E. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2006.

[93] P. Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, 26(4):1373–1385, 2013.

[94] K. Pearson. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.

[95] P. Pfaff, R. Triebel, and W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *Int. Journal of Robotics Research*, 26(2): 217–230, 2007.

[96] N. Radwan, G. D. Tipaldi, L. Spinello, and W. Burgard. Do you see the bakery? leveraging geo-referenced texts for global localization in public maps. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2016.

[97] A. Ranganathan, S. Matsumoto, and D. Ilstrup. Towards illumination invariance for visual localization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.

[98] J. Röwekämper, C. Sprunk, G. D. Tipaldi, C. Stachniss, P. Pfaff, and W. Burgard. On the position accuracy of mobile robot localization based on particle filters combined with scan matching. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[99] J. Röwekämper, B. Suger, W. Burgard, and G. D. Tipaldi. Accurate localization with respect to moving objects via multiple-body registration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[100] P. Ruchti, B. Steder, M. Ruhnke, and W. Burgard. Localization on openstreetmap data using a 3d laser scanner. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2015.

[101] J. Saarinen, H. Andreasson, T. Stoyanov, and A. J. Lilienthal. Normal distributions transform monte-carlo localization (NDT-MCL). In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[102] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[103] À. Santamaria-Navarro, E. H. Teniente, M. Morta, and J. Andrade-Cetto. Terrain classification in complex three-dimensional outdoor environments. *Journal on Field Robotics*, 32(1):42–60, 2015.

[104] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. *Robotics: Science and Systems*, 2(4), 2009.

[105] M. Shneier, T. Chang, T. Hong, W. Shackleford, R. Bostelman, and J. S. Albus. Learning traversability models for autonomous mobile vehicles. *Autonomous Robots*, 24(1): 69–86, 2008.

[106] *S300 Safety laser scanner*. Sick AG, 2016.

[107] A. Sinha and P. Papadakis. Mind the gap: detection and traversability analysis of terrain gaps using lidar for safe robot navigation. *Robotica*, 31(07):1085–1101, 2013.

[108] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*. Springer, 1990.

[109] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. of the 36th annual ACM Symposium on Theory of Computing*, 2004.

[110] B. Steder, M. Ruhnke, R. Kümmerle, and W. Burgard. Maximum likelihood remission calibration for groups of heterogeneous laser scanners. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2015.

[111] H. Strasdat, A. J. Davison, J. M. M. Montiel, and K. Konolige. Double window optimisation for constant time visual slam. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 2011.

[112] B. Suger and W. Burgard. Global outer-urban navigation with OpenStreetMap. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2017.

[113] B. Suger, G. D. Tipaldi, L. Spinello, and W. Burgard. An approach to solving large-scale SLAM problems with a small memory footprint. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2014.

[114] B. Suger, B. Steder, and W. Burgard. Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3d-lidar data. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2015.

[115] B. Suger, B. Steder, and W. Burgard. Terrain-adaptive obstacle detection. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[116] N. Sünderhauf, P. Neubert, and P. Protzel. Are we there yet? challenging seqslam on a 3000 km journey across all four seasons. In *Proc. of the ICRA Workshop on Long-Term Autonomy*, 2013.

[117] N. Sünderhauf, S. Shirazi, A. Jacobson, E. Pepperell, F. Dayoub, B. Upcroft, and M. Milford. Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free. In *Proc. of Robotics: Science and Systems (RSS)*, 2015.

[118] 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.

[119] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *Int. Journal of Robotics Research*, 23(7/8):693–716, 2004.

[120] S. Thrun, M. Montemerlo, and A. Aron. Probabilistic terrain analysis for high-speed desert driving. In *Proc. of Robotics: Science and Systems (RSS)*, 2006.

[121] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. Stanley: The robot that won the DARPA Grand Challenge. *Journal on Field Robotics*, 23(9):661–692, 2006.

[122] G. D. Tipaldi and F. Ramos. Motion Clustering and Estimation with Conditional Random Fields. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[123] A. Torii, R. Arandjelovic, J. Sivic, M. Okutomi, and T. Pajdla. 24/7 place recognition by view synthesis. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[124] 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.

[125] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal on Field Robotics*, 25(8):425–466, 2008.

[126] A. Valada, G. L. Olivera, T. Brox, and W. Burgard. Deep multispectral semantic scene understanding of forested environments. In *International Symposium on Experimental Robotics (ISER)*, 2016.

[127] C. Valgren and A. Lilienthal. SIFT, SURF & Seasons: Appearance-based long-term localization in outdoor environments. *Robotics & Autonomous Systems*, 58(2):149–156, 2010.

[128] J. van de Ven, F. Ramos, and G. D. Tipaldi. An integrated probabilistic model for scan-matching, moving object detection and motion estimation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[129] O. Vysotska, T. Naseer, L. Spinello, W. Burgard, and C. Stachniss. Efficient and effective matching of image sequences under substantial appearance changes exploiting gps priors. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2015.

[130] C. Wellington and A. Stentz. Online adaptive rough-terrain navigation vegetation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2004.

[131] C. Wellington and A. Stentz. Learning predictions of the load-bearing surface for autonomous rough-terrain navigation in vegetation. In *Field and Service Robotics*, 2006.

[132] M. Williamson, R. Murray-Smith, and V. Hansen. Robot docking using mixtures of gaussians. *Proc. of the Conf. on Neural Information Processing Systems (NIPS)*, 1999.

[133] K. Wurm, R. Kümmerle, C. Stachniss, and W. Burgard. Improving robot navigation in structured outdoor environments by identifying vegetation from laser data. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[134] S.-W. Yang and C.-C. Wang. Multiple-model ransac for ego-motion estimation in highly dynamic environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

[135] S.-W. Yang and C.-C. Wang. Simultaneous egomotion estimation, segmentation, and moving object detection. *Journal of Field Robotics*, 28(4):565–588, 2011.

[136] S.-W. Yang, C.-C. Wang, and C.-H. Chang. Ransac matching: Simultaneous registration and segmentation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[137] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

[138] A. R. Zamir and M. Shah. Accurate image localization based on google maps street view. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2010.