Accurate and Compact Surface Models for Mobile Robots

Michael Ruhnke

Technische Fakultät
Albert-Ludwigs-Universität Freiburg im Breisgau

UNI FREIBURG

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

Oktober, 2014

Accurate and Compact Surface Models for Mobile Robots

Michael Ruhnke

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

Dekan:	Prof. Dr. Yiannos Manoli
Erstgutachter:	Prof. Dr. Wolfram Burgard
Zweitgutachter:	Prof. Dr. Cyrill Stachniss
Tag der Disputation:	10.10.2014

Abstract

Mobile robots require environment models to perform autonomous navigation tasks, like the delivery of goods, cleaning up homes, surveillance and environmental monitoring. Besides environment models, mobile robots also need 3D models of objects to fulfill other complex tasks that include object detection and manipulation to provide better services for users.

Most mobile robots perceive the environment with range sensors and it seems desirable to learn 3D models directly from the collected data. The problem of learning a consistent model from range measurements with a mobile robot is known as simultaneous localization and mapping (SLAM). The goal of SLAM approaches is to estimate a spatial configuration of robot poses so that the sensor information accurately describes the environment. Given a SLAM solution, we can transfer the sensor measurements into a 3D model representation. The accuracy of the representation needs to be sufficiently high to let the robot successfully perform its tasks. Consider a robot that is autonomously navigating through a narrow passage, to find a valid, collision free path the robot needs an accurate model with a sufficient resolution.

This thesis addresses the problems of creating highly accurate 3D models and learning compact representations for those models. We present a novel algorithm to improve the accuracy of SLAM results by creating a joint optimization problem that takes into account small errors in robot poses and noise in sensor measurements. We discuss accurate sensor models of the underlying sensor to capture the typical noise characteristics of laser range finders and RGBD cameras. Furthermore, we estimate correspondences between different measurements of the same surface in different range scans to relate the available information. In this way, we minimize errors in the pose estimates and the sensor measurements and compute highly accurate models.

We also consider the memory requirements of high-resolution models since we intend to use them on resource constrained mobile robots. Typically, accuracy and compactness behave contrary to each other in a model representation. Therefore, we present an elegant way to describe surfaces in a model to achieve a data reduction. Many surfaces in human made environments have a similar shape, for example, flat surfaces, edges, or corners. Therefore, we organize shape and texture descriptions in dictionaries and describe a scene with respect to those. We learn the dictionaries during the model creation on the provided range data. Besides being compact, the models are more expressive than occupancy or point based representations since they allow us to search for specific surface attributes in a model to find an object. Additionally, we present a place recognition approach, which also uses a dictionary of reference features to speed up the search for similar places. One important application for environment models is navigation. Therefore, we present a fully integrated robotic system to perform pedestrian like autonomous navigation in highly crowded urban environments.

In this thesis, we present novel techniques and algorithms to learn accurate and compact 3D surface models with mobile robots. We evaluate the described techniques on real-world datasets and in realistic scenarios. Furthermore, we demonstrate that our methods enhance the state-of-the art.

Zusammenfassung

Eine wichtige Grundfähigkeit mobiler Roboter ist die autonome Navigation. Diese ist unerlässlich damit Roboter selbständig Aufgaben erledigen können, wie beispielsweise den Transport von Waren, das Säubern von Wohnungen, oder die automatische Überwachung der Umwelt. Um ihre Umwelt intern zu repräsentieren und mit ihr zu interagieren, benötigen Mobile Roboter Modelle. Ebenso wichtig wie Umgebungsmodelle sind Objektmodelle, die benötigt werden, um Objekte in der Umwelt zu detektieren und zu manipulieren. Ein praktisches Beispiel hierfür ist das Füllen oder Entleeren einer Geschirrspülmaschine durch einen mobilen Roboter, der die einzelnen Objekte erkennen muss, um sie an den richtigen Platz zu stellen.

Viele mobile Roboter verfügen über Entfernungssensoren, wie Lasermesssysteme oder 3D Kameras, die benutzt werden können, um 3D Modelle von Umgebungen oder Objekten direkt aus Sensordaten zu lernen. Dabei werden konsistente Modelle mit SLAM Algorithmen erstellt. SLAM ist die Abkürzung für „Simultaneous Localization and Mapping" und bedeutet übersetzt die gleichzeitige Lokalisierung in einer Karte und Kartenerstellung. Ein typischer SLAM Ansatz schätzt die räumliche Anordnung der Messdaten einer Umgebung. Generell gilt, dass die Genauigkeit des Umgebungsmodells die Präzision beeinflusst, mit der ein Roboter Aufgaben erfüllen kann. Um beispielsweise ein sehr kleines Objekt oder Detail in einem Modell zu erkennen, brauchen wir ein möglichst genaues und hoch aufgelöstes Modell. Daher ist es erstrebenswert, solche Modelle zu lernen. Da Roboter typischerweise über beschränkte Rechen- und Speicherkapazitäten verfügen, ergibt sich daraus eine weitere Herausforderung: die effiziente und platzsparende Repräsentation von hochauflösenden Umgebungsmodellen.

Im Rahmen dieser Doktorarbeit befassen wir uns mit der Erstellung von hochgenauer 3D Umgebungsmodellen aus den verrauschten Messungen von Abstandssensoren sowie mit dem Lernen kompakter Repräsentationen für diese Umgebungsmodelle. Die Arbeit ist wie folgt aufgebaut. In Kapitel 1 geben wir einen Überblick über die Inhalte dieser Arbeit und diskutieren relevante Problemstellungen. Anschließend fassen wir unsere wissenschaftlichen Beiträge zusammen und geben einen Überblick über unsere wissenschaftlichen Veröffentlichungen, sowie veröffentlichte quelloffene Software. In Kapitel 2 führen wir dann die Notation ein, die wir in dieser Arbeit verwenden, und grundlegende Datenstrukturen und Algorithmen auf die wir im Folgenden aufbauen.

In Kapitel 3 beschreiben wir einen Ansatz, um die Genauigkeit von Umgebungsmodellen zu verbessern, die mit einem SLAM Algorithmus erstellt wurden. Dabei berücksichtigen wir sowohl kleine Fehler in den Positionsschätzungen als auch Messrauschen in den Sensordaten und modellieren dies als quadratisches Optimierungsproblem, bei dem Fehler und Messrauschen gleichzeitig minimiert werden. Im Zuge der Optimierung werden 3D Beobachtungen in einer Größenordnung verformt, die dem typischen sensorspezifischen Messrauschen entspricht. Dafür stellen wir akkurate Sensormodelle sowohl für Laserscanner als auch RGBD Kameras vor. Dies führt zu hoch genauen 2D oder 3D Umgebungsmodellen.

Beginnend mit Kapitel 4 beschäftigen wir uns mit speichereffizienten 3D Repräsentationen für Umgebungsmodelle. Die zugrunde liegende Idee der vorgeschlagenen Surface Primitive

Modelle ist es, die Umgebung mit einer kleinen Anzahl von Beispieloberflächen bzw. Oberflächenprimitiven zu beschreiben. Die Oberflächenprimitiven sind dabei als eine Art Wörterbuch zu verstehen, mit deren Hilfe eine Umgebung beschrieben werden kann. Jede dieser Primitiven kann dabei mehrfach verwendet werden. Um ein Modell einer Umgebung zu erstellen, muss für jede Oberfläche die Oberflächenprimitive gefunden werden, welche sie am besten beschreibt. Zusätzlich muss auch ein Referenzwörterbuch gelernt werden. Zu diesem Zweck schlagen wir einen Lernansatz für Wörterbücher vor, der die Genauigkeit eines Modells mit deren Komplexität bzw. der Größe des Referenzwörterbuchs abwägt. Die auf diese Art gelernten 3D Modelle sind typischerweise klein bzw. kompakt.

In Kapitel 5 führen wir die Idee hinter Surface Primitive Modellen weiter, um farbige 3D Modelle zu lernen. Dabei betrachten wir Oberflächen und Texturen als getrennte Kanäle und lernen für jeden Kanal ein eigenes Referenzwörterbuch. Zusätzlich benutzen wir eine flexible Kodierung, die es uns ermöglicht Texturen oder Oberflächen als Mischung von Wörterbucheinträgen zu beschreiben. Das führt dazu, dass kleinere Wörterbücher gelernt werden und die typischen Variationen, besonders in der Textur, besser beschrieben werden können. Des weiteren haben wir das Verfahren zum Lernen der Referenzwörterbücher erweitert, um es robust bezüglich Verdeckungen und nicht gesehener Teile der Oberfläche zu machen.

In Kapitel 6 beschreiben wir eine hierarchische Erweiterung des Ansatzes aus dem vorherigen Kapitel, welche Umgebungen mit verschieden großen Oberflächen repräsentieren kann. Dabei wird der Rekonstruktionsfehler benutzt, um zu entscheiden in welcher Größe bzw. mit welcher Auflösung eine Oberfläche beschrieben wird. Dies führt zu Umgebungsmodellen, in denen Oberflächen mit wenig Variation in Struktur oder Textur mit großen, gering aufgelösten Oberflächen repräsentiert werden und Bereiche mit viel Variation mit kleinen, hoch aufgelösten Oberflächen.

In Kapitel 7 beschreiben wir einen Ansatz, mit dem ein Roboter anhand seiner 3D Beobachtungen erkennen kann ob er an einem Ort schon einmal gewesen ist. Dies kann dazu verwendet werden den akkumulierten Fehler in der Positionsschätzung zwischen dem vorherigen Besuch eines Ortes und dem aktuellen Besuch zu minimieren. Dies ist beispielsweise für einen graphbasierten SLAM Algorithmus relevant. Analog zu den vorherigen Kapiteln, werden hier Ähnlichkeiten in 3D Beobachtungen gesucht, auch wenn der Fokus nicht darauf liegt die Umgebung zu repräsentieren. Ausgehend von einer 3D Beobachtung ist es das Ziel unseres Ansatzes alle 3D Beobachtungen aus einer Datenbank zu finden, welche denselben Ort zeigen. Die zugrunde liegende Idee ist, das wir ähnliche Strukturen in 3D Beobachtungen finden wenn diese dieselbe Umgebung zeigen. Um die Suche nach ähnlichen Strukturen zu beschleunigen, benutzen wir die relativen Häufigkeiten von Merkmalen aus einem Referenzwörterbuch als Ähnlichkeitsmaß. Für potentielle Kandidaten wird eine Transformation geschätzt basierend auf den Merkmalen in den beiden Beobachtungen. Mit dieser Transformation wird dann geometrisch überprüft, ob es sich um eine konsistente Lösung handelt.

Wir präsentieren ein vollständig integriertes Navigationssystem für mobile Roboter in Kapitel 8. Besonderer Schwerpunkt bei diesem System ist die robuste und sichere Navigation auf Gehwegen und in dicht bevölkerten Innenstädten wie beispielsweise Fußgängerzonen. Die Schlüsselkomponenten unseres Systems sind ein SLAM Modul, das Karten in Stadtgröße erstellen kann, ein effizienter Planer, um kollisionsfreie Pfade zu berechnen, ein Modul zur Erkennung befahrbarer Oberflächen sowie eine robuste Lokalisierung für dynamische Umgebungen. Das vorgestellte System wurde auf einem realen Roboter implementiert und in mehreren großen Feldtests evaluiert.

Abschließend besprechen wir die Schlussfolgerungen, die wir aus dieser Arbeit ziehen und beschreiben mögliche zukünftige Erweiterungen für die vorgestellten Ansätze.

Acknowledgements

I would like to thank all people who supported me in writing this thesis. My special thank goes to my adviser Wolfram Burgard for his time, encouragement, guidance and support. He provided me with the freedom to pursue my own ideas, created a highly motivating work environment and had always an open door for me.

My co-advisor Giorgio Grisetti supported me and encouraged me to learn various important things around scientific work, and writing papers. My deepest thanks for talking me into doing a Ph.D., all the fruitful discussions, and the immense amount of fun we had together. Furthermore, I would like to thank Cyrill Stachniss for reviewing this thesis.

Likewise, I thank all colleagues and co-authors for interesting discussions and collaborations: Rainer Kümmerle, Bastian Steder, Slawomir Grzonka, Jürgen Sturm, Christian Dornhege, Cyrill Stachniss, Alexander Kleiner, Liefeng Bo, Juan D. Tardós, and Dieter Fox.

Special thanks go to Dieter Fox, Liefeng Bo, Evan Herbst, Peter Henry, Cynthia Matuszek, Jinna Lei and Heinrich Mellmann for a scientifically interesting and very pleasant stay at the University of Washington.

Moreover, I would like to thank my office mates Rainer Kümmerle and Bastian Steder for the great atmosphere, countless advices, and deep friendship. Additional thanks go to all colleagues, friends, and former colleagues in the AIS group. Special thanks go also to Christoph Sprunk, Axel Rottmann for their latex and xfig support. I would also like to thank Rainer Kümmerle, Bastian Steder, Barbara Frank, Slawomir Grzonka and Christoph Sprunk for the valuable feedback they provided for this thesis. Furthermore, I would like to thank Susanne Bourjaillat, Kristine Haberer, Michael Keser, Bettina Schug, and Daniela Wack for their administrative and technical support.

The EUROPA project was a big success and I would like to thank all people involved in creating the system. Datasets are a very important resource in robotics. Therefore, I would like to thank all people who made their datasets either publicly available or available to me. In particular, I thank Dirk Hähnel, Cyrill Stachniss, Martin Magnusson, Bastian Steder, Peter Henry, and Jürgen Sturm.

My deepest thanks go to my family and especially to my wife Heidi for their support and endless love.

This work has partly been supported by the European Commission under contracts number FP7-231888-EUROPA, ERC-AG-PE7-267686-LifeNav and FP7-610603-EUROPA2. The support is gratefully acknowledged.

Contents

Chapter 1

Introduction

Humans are used to live in a three-dimensional world and to reason about space and appearance. As a consequence, it seems natural to use textured 3D models to represent an environment in a computer. This has the advantage of an intuitive user experience for applications such as virtual reality or games. Often, 3D environment models are manually created from scratch in this context. Besides the effort to manually design a model, it is hard to build models that are comparably rich in details as models captured in the real world. Therefore, it seems desirable to construct 3D models of environments with help of 3D sensors. Accurate high-resolution models of real-world locations have many potential applications. They can be used to record crime scenes and make them virtually available for investigators. Models of historical artifacts or sites could preserve a specific state at a specific point in time and allow for virtual visits and to study details even if an artifact or site gets damaged. Furthermore, such models can be used to track changes over time and help to autonomously monitor an environment, for example, to detect diseases of plants or corals. Given the recent advances in 3D printing technology, we can also print out 3D models and replicate objects.

When we think about mobile robots, it is essential to model the world and enable robots to reason about their surroundings and interact with the environment. Since most mobile robots are equipped with range sensors, we would like to learn 3D models directly from the sensor data. This problem is known as simultaneous localization and mapping (SLAM). The goal is to estimate a spatial configuration of robot poses so that the available sensor data accurately resemble the information about the environment. Therefore, a SLAM system needs to recognize a place it has visited before. This helps to correct accumulated errors along the driven trajectory between the previous visits of that location and the current visit. In the SLAM community, this is called a loop-closure. Loop-closures are essential to build a globally consistent model. Therefore, it is a key challenge to recognize previously seen places based on 3D range data in an efficient manner. Furthermore, the desired resolution of a model as well as the scale of the environment can introduce additional challenges in the model estimation. A high-resolution model of an environment needs a highly accurate SLAM solution and the noise introduced by sensors like laser range finders or RGBD cameras introduces additional challenges in the model estimation. To solve a SLAM problem, we can also use the information of additional sensors like an inertial measurement unit (IMU) or a GPS device. This is especially useful for creating large-scale environment models of cities. Furthermore, we also have to consider the resource constraints of a mobile robot in the context of large-scale models and organize models accordingly.

An important application for 3D environment models is the autonomous navigation of a mobile robot. To successfully navigate, a mobile robot needs to estimate its pose in the model with

help of a localization algorithm. With a known pose, a robot can plan and execute a collision-free trajectory from the current location to a specified goal location. Such a navigation system can be used for the delivery of goods, food or medication. Another important application for autonomous or semi-autonomous navigation are wheelchairs. With help of such a technology, an old or disabled person could gain enhanced mobility and therefore substantially improve his or her quality of life.

Given a robot is equipped with a manipulator, we could also think about pick and place tasks. In the context of an industrial application, mobile robots could deliver needed parts just in time to speedup or optimize a manufacturing process. With groups of mobile robots, a manufacturing process could also be made more flexible since either different products or several variants of the same product, could be assembled just by changing the software. Furthermore, we envision mobile service robots to make our life more comfortable, e.g., by cleaning up our homes, filling and emptying the dishwasher, or mowing the lawn. To fulfill such tasks, we would like to create mobile service robots that are aware of their surroundings, reason about objects, and manipulate them. One challenging aspect of this scenario is the hard underlying perception problem one has to solve to create such robots. Due to the complexity of the task, we usually try to detect only a specific set of objects in the sensor data of a robot, needed for a specific application.

This is also related to the underlying representation of the environment model. Usually, a task-specific representation is chosen, like landmark-based representations for localization, grid maps for planning, and feature-based representations for object detection tasks. But what happens if we do not know the application beforehand? In such cases, we would like to include all available sensor information into one accurate and compact model and let an application designer decide later on, which information he needs for his specific task. The main challenge is to transfer the model into a representation that can be used for a broad variety of applications. Point clouds or occupancy grid maps are well suited for tasks like localization or planning but detecting an object in an occupancy grid map is a difficult task. Consider a representation in which we apply 3D perception algorithms to encode similarities of local surfaces or textures. In such a model, we can easily search for similar surfaces or textures to find an object. As we will see later in this thesis, such models yield richer representation. Of course, learning such a model introduces additional perception challenges, but this can be seen as a first step towards a robot that is aware of its surrounding in his internal representation. Out of the described challenges for mobile robots we identified the following research questions that we will address in this thesis:

- How can we estimate highly accurate models, considering sensor noise and pose uncertainties at the same time?
- How to deal with the massive amount of data to build accurate 3D models?
- How can we use 3D perception techniques to provide a semantically richer representation?
- How can we encode similarities in shape or appearance at different scales?
- How can we reliably detect places the robot has visited before based on 3D range data?
- How can we build a fully integrated robotic navigation system to successfully navigate in large-scale urban environments?

In the following, we will give an outline of the topics of this thesis. In the remainder of this chapter, we will briefly summarize our scientific contributions, give an overview of the released

open source software, and discuss collaborations. In Chapter 2, we will describe the notation used throughout the thesis, important data structures, and relevant methods as foundation for this thesis.

In Chapter 3, we describe a method to improve the accuracy of SLAM results. Often the accuracy of a SLAM model suffers from noise in the sensor observations and small remaining errors in the pose estimates. We construct an optimization problem that takes into account the typical errors for laser range finders and RGBD cameras. In this way, we can allow scans to deform and at the same time optimize the corresponding poses, leading to highly accurate 2D and 3D models.

Starting with Chapter 4, we focus on compact representations for 3D range data. The underlying idea of the presented surface primitive models is to represent the surfaces of an environment with a fixed set or dictionary of example surface primitives. Each primitive can be used several times to describe the surfaces of an environment. To model the input data, we need to detect which primitive can describe which part of the scene. Furthermore, we introduce a learning scheme for surface primitive dictionaries that trades off accuracy of the model versus complexity. As a result, we can learn highly compact models of 3D range data.

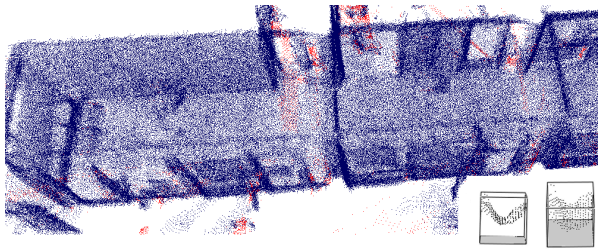
In Chapter 5, we extend the surface primitive idea to deal with textured 3D data and apply a more flexible scheme to learn dictionaries based on sparse coding. We represent range data and texture as different channels and learn a dictionary per channel. Instead of inserting one surface primitive at a location we use a mixture of discrete surface descriptions to express surface attributes. This flexible coding scheme allows us to learn smaller and more expressive dictionaries. Furthermore, we extended the dictionary learning procedure to better deal with occlusions.

In Chapter 6, we describe a hierarchical extension to deal with surface patch descriptions of different sizes. We use the reconstruction error to guide the decision at which level of the hierarchy we should model a part of the surface. This leads to variable resolution models that represent areas with few variations at low resolution and areas with large variations at high-resolution.

After the surface representations, we present a place recognition system for 3D range data in Chapter 7. Like in the previous chapters, we search for similarities in 3D range observations but the focus is to compute loop-closure constraints for estimating a SLAM solution and not to represent models. Given we have a new 3D range scan and a database of scans, we would like to find scans in our database that correspond to the same location, observed in the new scan. We use a dictionary of feature descriptors to guide the search for scans potentially showing the same location to speedup the search. In the next step, we compute scan wise feature correspondences and apply geometric validation to filter out false positives.

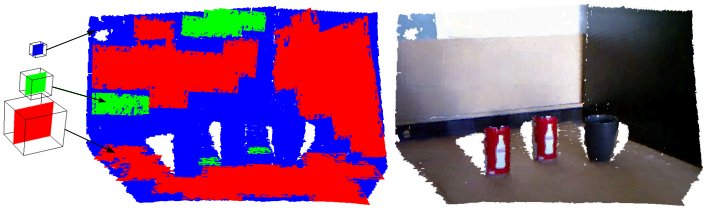
All previously mentioned chapters focus on specific subproblems to create mobile robots that autonomously perform tasks. In Chapter 8, we present a fully integrated navigation system for mobile robots designed to operate in crowded urban environments. We describe the different components of our system including a SLAM module for dealing with city scale maps, a planning component for inferring feasible paths, a traversability and vegetation analysis tool, and a module for accurate localization in dynamic environments. The described navigation system has been implemented on a real robot and has been tested in several city-scale field tests. In Chapter 9, we discuss the conclusions arising out of this thesis and describe ideas about future work building on top of the presented topics.

(a) High-resolution maximum likelihood models

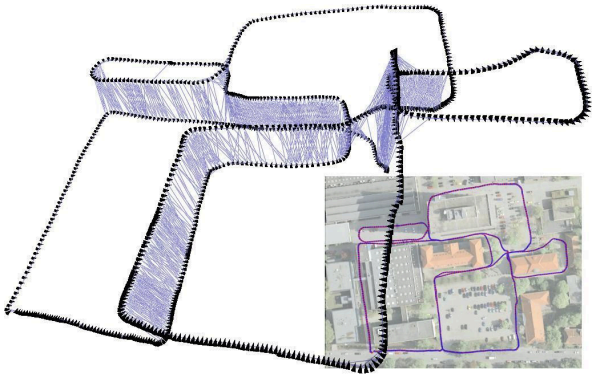


(b) Compact 3D models

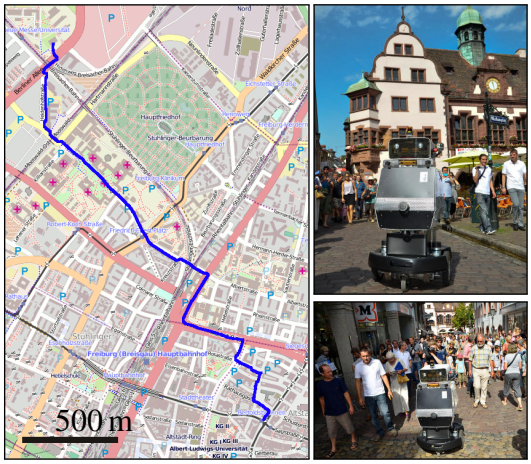
(c) Compact RGBD models



(d) Hierarchical RGBD models



(e) Place recognition with 3D range data



(f) City-scale robot navigation

Figure 1.1: Motivating figures of the individual chapters of the thesis. **(a)** Chapter 3: Sparse Surface Adjustment, joint optimization of sensor poses and range measurements for highly accurate models. **(b)** Chapter 4: Surface Primitives, compact 3D models based on the detection of recurrent surface patterns. **(c)** Chapter 5: Sparsely Coded Surface Models, accurate and compact RGBD models with sparse coding. **(d)** Chapter 6: Hierarchical Sparsely Coded Surface Models, even more compact surface descriptions, due to different surface patch sizes and resolutions. **(e)** Chapter 7: Place recognition on 3D range data. **(f)** Chapter 8: City-scale navigation with a mobile robot in highly crowded environments.

1.1 Scientific Contribution

In this thesis, we propose several novel methods to learn highly accurate and compact surface models with mobile robots. Such models can represent objects or large environments. Furthermore, we present a robust navigation system for mobile robots suited to operate on sidewalks and in densely populated city centers. Figure 1.1 shows the motivating images of the individual chapters. To abstract the key contributions, we propose:

- a novel method to compute highly accurate 2D maps or 3D models, taking into account typical sensor uncertainties of lasers and RGBD cameras (Chapter 3).
- a compact representation for range data exploiting repetitive structures based on a dictio-

nary, together with a dictionary learning scheme that directly trades off accuracy versus compactness to learn repetitive structures (Chapter 4).

- a novel dictionary learning method that can be utilized on textured range data based on sparse coding. In this context we propose an extension to the machine learning technique k-svd, to learn better dictionaries on range data in the presence of occlusion (Chapter 5).
- an approach to model textured range data in a hierarchical fashion based on sparse coding (Chapter 6). In this way, we can capture similarities in texture or shape at different scales and build more complex models.
- a novel place recognition system that exploits surface appearances based on a dictionary to speedup the candidate search for similar places (Chapter 7).
- an integrated and operational navigation system for mobile robots where the main contributions lie in the large-scale environment model and the robust localization in such models (Chapter 8).

1.2 Open-Source Software

In the context of this thesis, we published the following open source software:

- SSA (*https://github.com/MichaelRuhnke/ssa*) is an open-source C++ tool for post optimization of 2D or 3D SLAM solutions. SSA iteratively refines robot poses and surface points in one global graph optimization system and produces highly accurate 3D point clouds. Sensor observations are not treated as rigid body and might be refined during the optimization procedure. This leads to substantially less accumulated noise in the resulting model. SSA builds upon g2o as optimization back-end. SSA was developed in collaboration with Rainer Kümmerle, Giorgio Grisetti, and Wolfram Burgard.
- SLAM Benchmarking toolkit (*http://ais.informatik.uni-freiburg.de/slamevaluation/*) is an open-source C++ tool to compare the solutions of different SLAM algorithms based on a set of manually verified relations between key poses on a SLAM trajectory. As part of the toolkit we also provide relation sets for several publicly available SLAM datasets. This toolkit is a result of the collaboration with Rainer Kümmerle, Bastian Steder, Christian Dornhege, Giorgio Grisetti, Cyrill Stachniss, Alexander Kleiner, and Wolfram Burgard.

1.3 Publications

The work presented in this thesis is based on several papers published in international conference proceedings, workshops, book chapters and submitted to an international journal. A chronological overview of the individual publications is presented in the following list:

Book Chapters and Journal Articles

- R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. Autonomous robot navigation in highly populated pedestrian zones. In *Journal on Field Robotics*, 2014. (to appear).
- M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. *3D Environment Modeling Based on Surface Primitives*, pages 281–300. Springer Berlin/Heidelberg, 2012a.

Conference Proceedings and Workshops

- M. Ruhnke, Liefeng. Bo, D. Fox, and W. Burgard. Hierachical sparse coded surface models. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2014. (to appear).

- M. Ruhnke, Liefeng. Bo, D. Fox, and W. Burgard. Compact RGBD surface models based on sparse coding. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2013.

- 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 and Automation (ICRA)*, 2013.

- M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Range sensor based model construction by sparse surface adjustment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012b.

- B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard. Place recognition in 3D scans using a combination of bag of words and point feature based relative pose estimation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011a.

- M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Range sensor based model construction by sparse surface adjustment. In *Proc. of the IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 2011b.

- M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Highly accurate maximum likelihood laser mapping by jointly optimizing laser points and robot poses. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011a.

- M. Ruhnke, B. Steder, G. Grisetti, and W Burgard. Unsupervised learning of compact 3D models based on the detection of recurrent structures. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010. **Finalist Best Paper Award**.

The following publications are not covered by this thesis

- R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Autonomous Robots*, 27(4):387–407, 2009. doi: http://dx.doi.org/10.1007/s10514-009-9155-6.

- W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and Juan D. Tardós. A comparison of SLAM algorithms based on a graph of relations. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

- M. Ruhnke, B. Steder, G. Grisetti, and W Burgard. Unsupervised learning of 3D object models from partial views. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

1.4 Collaborations

Due to the academic nature of this work, I collaborated with several researchers over a couple of years to create this thesis. Such collaborations can be fruitful discussions leading to novel ideas as well as bringing together the expertise to build a system that fulfills a specific task.

- Chapter 3 is the result of a collaboration with Rainer Kümmerle, and Giorgio Grisetti. The author of this thesis was mainly involved in the problem formulation (Sections 3.1 and 3.2), the sensor models for laser range finders (Section 3.3.1) and RGBD cameras (Section 3.3.2), and solving the data association problem (Section 3.4), whereas the contribution of Rainer Kümmerle was the construction of the optimization problem described in Section 3.5. Related publications are Ruhnke et al. (2011b, 2012a).

- Chapter 4 is joined work with Bastian Steder who was mainly involved in the interest point and feature extraction described in Section 4.1.3. Related publications are Ruhnke et al. (2010, 2012b).

- Chapters 5 and 6 are the result of fruitful discussions with Liefeng Bo and Dieter Fox. Related publications are Ruhnke et al. (2013, 2014).

- The place recognition system described in Chapter 7 is the result of joined effort with Bastian Steder and Slawomir Grzonka. While we present the full approach for completeness, only the appearance based similarity candidate ordering to perform an informed search described in Section 7.2.2 was contributed by the author of this thesis. The related publication is Steder et al. (2011a).

- Chapter 8 describes a navigation system for a robotic pedestrian assistant and is joined work together with Rainer Kümmerle, Bastian Steder, and Cyrill Stachniss. The described work is the outcome of the EU project EUROPA. We present the whole navigation system for completeness. The main contribution of the author of this thesis is the world model data structure (Section 8.2.3) and the robust localization (Section 8.2.4) in such models. Related publications are Kümmerle et al. (2013, 2014).

Chapter 2

Basics

In this chapter, we briefly discuss basic concepts and algorithms as foundation for the methods we will describe later in this thesis. First, we give an overview of our notation, used through the thesis, followed by basic data structures for 2D and 3D range data. We conclude the chapter with a brief description of the SLAM problem and the graph-based formulation for SLAM algorithms.

2.1 Notation

Overview of the notations and symbols used in this thesis:

Symbol	Description
$a, b, c, \dots$	scalar value
$\mathbf{x}, \mathbf{y}, \dots$	vector
$\hat{\mathbf{x}}$	estimated value of $\mathbf{x}$
$\mathbf{x}^*$	optimal value of $\mathbf{x}$
$\|\mathbf{x}\|_0 = \#(i \|x_i \neq 0)$	l_0 norm, number of non-zero entries of a vector
$\|\mathbf{x}\|_1 = \sum_i^n \|x_i\|$	Manhattan norm
$\|\mathbf{x}\|_2 = \sqrt{\sum_i^n \|x_i\|^2}$	Euclidean norm
$A, M, \dots$	matrix
$\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \dots\}$	set
$\mathcal{D}$	dictionary as set of dictionary entries
$\mathfrak{S} = \{\mathcal{S}_1, \mathcal{S}_2, \dots\}$	set of sets
$\|\mathcal{S}\|, \|\mathbf{x}\|$	size of set (number of elements) or vector (number of dimensions)
$\langle \dots \rangle$	tuple (ordered list)
$\mathbf{x}_i \oplus \mathbf{x}_j$	motion composition operator as described in Smith et al. (1990)
$\mathbf{x}_i \ominus \mathbf{x}_j$	inverse of the motion composition operator
$A \odot B$	element wise matrix multiplication

2.2 Data Structures

2.2.1 Point Cloud

Throughout this thesis, we use mostly point clouds as data structure for raw range data. A point cloud $\mathcal{P} = \{\mathbf{p}_1, ..., \mathbf{p}_{|\mathcal{P}|}\}$ is a set of points $\mathbf{p}_i, i \in 1, \ldots, |\mathcal{P}|$. Each point is an n-dimensional vector. The dimensionality is dependent on the data channels provided by the sensor. Each point can store a 2D/3D location together with the measured range, intensity or RGB information. We create a point cloud from a range scan by inserting a point at the end of each projected beam, according to the pose of the sensor. Figure 2.1 shows a point cloud in (a) and a colored point cloud in (b).

(a) **(b)**

Figure 2.1: Example of a point cloud and a corresponding point cloud with color information.

2.2.2 Range Image

A range image or depth map can store 3D data obtained at one viewpoint in a 2D image. An example range image can be seen in Figure 2.2. The value in every pixel resembles the range to the closest surface from the chosen viewpoint. Darker pixels are closer to the observer and blue pixels correspond to invalid range measurements. A range image can be easily converted into a point cloud by inserting a point at the given range in the direction of the pixel. With the z-Buffer algorithm (Foley et al., 1994) we can calculate a range image out of a point cloud. For a fused point cloud containing information from different viewpoints a corresponding range image cannot represent all information. Different points might occlude each other and only the point closest to the observer is represented. Neighborhood queries are very fast on range images, since nearby points are located in neighboring pixels.

2.3 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is one of the fundamental problems for mobile robots. The goal is to estimate a model of an environment and localize a vehicle or robot in this model at the same time. The general problem in this context is that an accurate pose estimate is needed to render an consistent model and on the other hand one needs a consistent

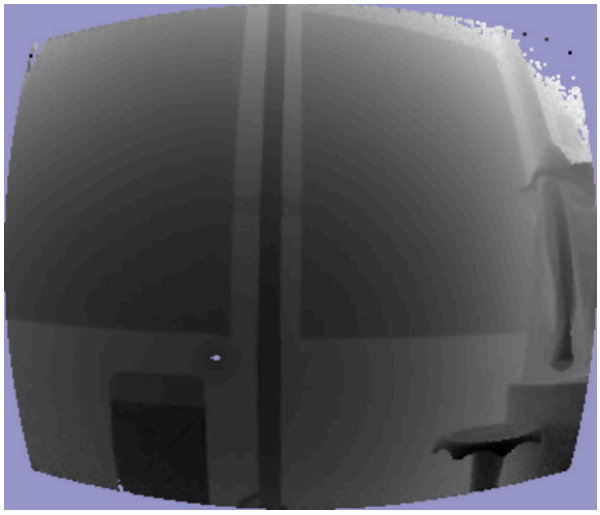


Figure 2.2: Example of a range image of the point cloud presented in Fig. 2.1. The depth is encoded with gray values and the darker the pixel color the closer it is to the observer.

model to localize the robot with respect to that model. The problem of fusing the available sensor information into one model is referred to as *mapping with known poses* in the literature. In case of a 2D laser sensor, this can be achieved by projecting the laser beams into a common model based on the pose of the robot and the offset between robot and sensor. This can be solved easily with perfectly known robot poses but if the poses have small errors, we accumulate errors by fusing the data into a common model. Accumulated errors lead to a growing uncertainty in the model, what makes it harder localize a robot in the model. This again leads to growing errors in the pose estimate and as consequence to even worse models.

Localization and mapping with known poses are mutually dependent on each other and there are two different paradigms to deal with this dependency. One option is to treat them independently in an alternating fashion. Very prominent approaches in this area are PTAM (Klein and Murray, 2007) for monocular vision and Kinect Fusion (Newcombe et al., 2011) for RGBD cameras. In general this works well in small environments but does not scale to larger environments where loop closures can be used to correct accumulated errors in the pose estimates over long trajectories and their impact on the map.

To overcome this problem, one has to estimate the environment model and pose of the robot at the same time. This can either be done with information filters like extended Kalman filters (EKFs) (Smith et al., 1990; Leonard and Durrant-Whyte, 1991), sparse extended information filters (Thrun et al., 2004; Eustice et al., 2005), particle filters (Montemerlo et al., 2003; Grisetti et al., 2007) or with least square error minimization techniques (Lu and Milios, 1997; Gutmann and Konolige, 1999; Frese et al., 2005; Olson et al., 2006). Due to quality and robustness of the solutions, the error minimization formulation of the SLAM problem as graph of spatial relations has become very popular in the SLAM community. We will present a brief overview of graph-based SLAM since this is highly related to the methods described in Chapters 3 and 7.

2.3.1 Graph-based SLAM

Given a robot trajectory, we can model a SLAM problem as a graph where each pose $\mathbf{x}_i$ is described with a node. Two poses $\mathbf{x}_i$ and $\mathbf{x}_j$ are connected with an edge $\mathbf{z}_{ij}$ which corresponds to a spatial relation. Figure 2.3 illustrates an exemplary pose-graph structure. Spatial relations can

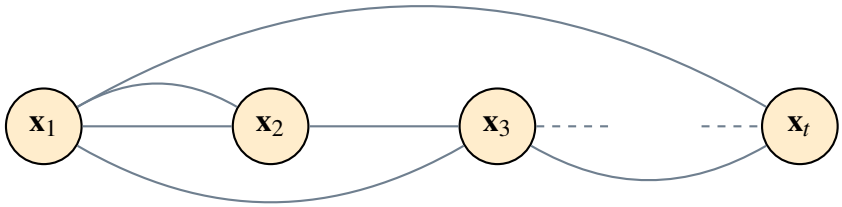


Figure 2.3: Overview of the graph structure of a SLAM problem. Every node $\mathbf{x}_i$ corresponds to a robot pose and the grey edges correspond to spatial relations between the nodes.

be introduced by different sources and correspond to a measured relative displacement between two poses. Such measurements can come from robot odometry or be the result of a transformation estimated from the sensor data, like the result of a scan matcher. Especially interesting are spatial relations between robot poses in situations where a robot revisits a place where it has been before. In the SLAM literature this is referred to as loop-closure relations. Loop closures allow us to correct accumulated errors in the pose estimates along a robot trajectory on the driven loop. We can compute the error for a relation $\mathbf{z}_{ij}$ with the following equation:

$$\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}) = \mathbf{z}_{ij} \ominus (\mathbf{x}_i \ominus \mathbf{x}_j). \tag{2.1}$$

The weighted error is then

$$\mathbf{e}_{ij} = \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^\intercal \Omega_{ij} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}), \tag{2.2}$$

where Ω_{ij} denotes the information matrix used to weight the error for a relation $\mathbf{z}_{ij}$ according to the uncertainty of a measurement. In this way, we avoid a uniform distribution of the accumulated error and distribute it in an informed fashion. In case of odometry relations the uncertainty can be evaluated based on a motion model. The estimated uncertainty of a correlative scan matcher can also be used as measurement uncertainty for a scan pair.

Based on the weighted error function we can construct the following minimization problem

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{i,j} \mathbf{e}_{ij} \tag{2.3}$$

With Equation 2.3 we can compute a graph configuration $\mathbf{x}^*$ that minimizes the weighted error. This is also the maximum-likelihood configuration of the graph given the measurements according to $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{z})$ since the optimized error function $\sum_{i,j} e(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^\intercal \Omega_{ij} e(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ corresponds to the negative log-likelihood of the maximization problem. In this thesis, we use the publicly available General Graph Optimizer (g^2o) framework (Kümmerle et al., 2011) to solve graph-based minimization problems.

Chapter 3

Sparse Surface Adjustment

In the previous chapter, we briefly described graph-based SLAM as a basic technique to estimate consistent models of an environment. In this context, range data is usually treated as rigid body and sensor noise is not considered. Furthermore, the alignment uncertainty for edges between two poses is modeled with a Gaussian distribution, which is only an approximation of the true surface dependencies. Often this leads to blurred surfaces in the model for high-resolution renderings. Therefore, we describe a post optimization approach to obtain highly accurate 2D and 3D models from SLAM solutions in this chapter. The key idea of this method is to jointly optimize the poses of the sensor and the positions of the surface points, measured with a range scanning device. Instead of optimizing a set of constraints between sensor pose pairs, we optimize the position of every point together with the sensor poses. To relate surface points, we connect corresponding points of different scans with constraints. Furthermore, our approach applies a physical model of the underlying range sensor (laser or RGBD camera) to constrain the relations between surface points and sensor poses. To solve the optimization task it employs a state-of-the-art graph-based optimizer and iteratively refines the structure of the error function by recomputing the data associations after each optimization. We present our approach and evaluate it on data recorded in different real-world environments with RGBD cameras and laser range finders. The experimental results demonstrate that our method is able to substantially improve the accuracy of SLAM results.

♦ ♦ ♦ ♦ ♦ ♦ ♦

In the previous chapter, we introduced graph-based SLAM as a technique to build maps or surface models for mobile robots. Such maps are typically globally consistent (see Figure 3.1(a)) but tend to have uncertainties in the surface estimates leading to blurred walls (see Figure 3.1(b)). There are two potential sources that cause this effect, either small remaining errors in the pose estimates or accumulated sensor noise. Errors in pose estimates can be caused by sensor noise, approximations in the estimate computation or resolution limitations of a scan matcher used to create constraints in a pose graph. Furthermore, the covariance estimate of a scan matcher can also be sub-optimal in cases where the underlying distribution is not unimodal

(a) grid map of the Intel dataset **(b)** enlarged part of the map

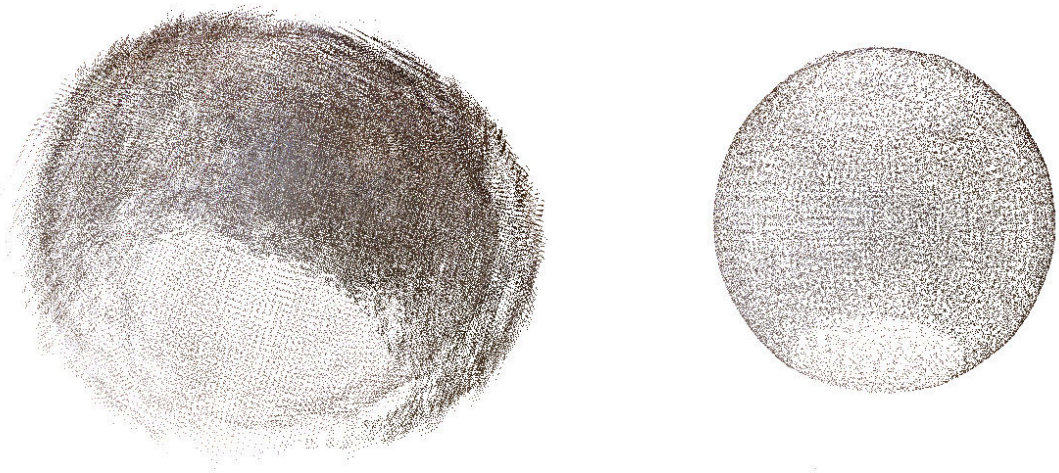
Figure 3.1: (a) shows the grid map obtained by a standard SLAM approach. (b) illustrates the uncertainty of the walls in a grid map with 5 cm resolution. The walls are mostly 2 to 3 cells thick which corresponds to 10 to 15 cm.

or simply not a Gaussian. An additional challenge is that the true data association is typically unknown on range data. Due to runtime constraints, scan matching approaches usually operate with a limited search range and at a manageable resolution. Furthermore, not all scans that share at least one common point have a constraint between their sensor poses.

In this chapter, we propose a graph-based approach for solutions to 2D and 3D SLAM problems by considering it as a joint optimization task that simultaneously estimates the robot poses and the surfaces in the environment. Our approach applies an accurate uncertainty model of the underlying range sensor and considers the endpoints of a range scan as *samples* generated by the surfaces of the environment. We iteratively refine the graph structure by recomputing the data association between each individual distance measurement and the local surface points. In this way, we can relax the assumptions that each range scan is a rigid body so that we obtain more accurate maps. Additionally, we extend the accuracy and robustness with respect to errors in the data association by combining two strategies with different search ranges. Our method is a post-processing procedure to be applied to the output of graph-based SLAM systems and is designed to provide locally more consistent models.

We evaluate the accuracy of the models computed by our method by means of ground truth measurements. We furthermore characterize the error of several laser range finders as well as the Microsoft Kinect both theoretically and experimentally. In particular RGBD cameras are affected by a quantization error in the disparity measurements. This error leads to noise in the range data that grows quadratically with the distance. By specifically modeling this error, our approach is able to use long range and noisy measurements, to capture wide regions, and to assess the structure of the scene. At the same time it can refine the local structures given the more accurate short range measurements, resulting in a very accurate super resolution model.

Figure 3.2 shows an RGBD example illustrating the accuracy that can be achieved with the approach described in this chapter for the reconstruction of a spherical object with a radius of 75 mm. The average error in the measured radius is below 1.5 mm. The remainder of this chapter is organized as follows, we will start with a brief overview of the approach, followed



(a) input data **(b)** Sparse Surface Adjustment result

Figure 3.2: (a) shows an accumulated model of a metallic sphere used as input to our method. (b) shows the result of our method after 20 iterations. As can be seen, the resulting model has a smooth surface. Furthermore, the model is also highly accurate with a mean error of 1.4 mm.

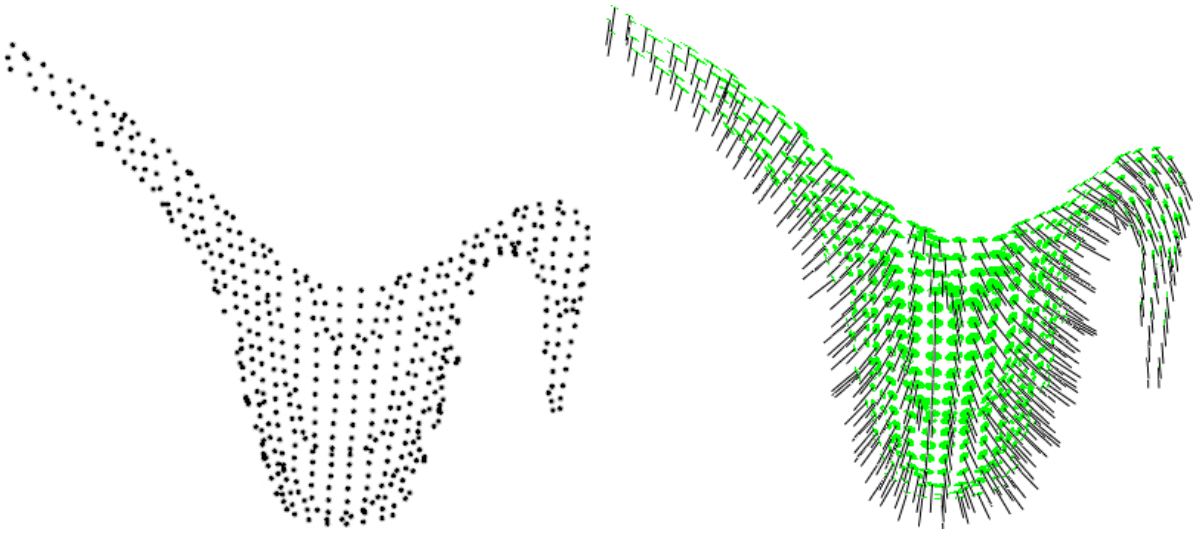
by a description of our surface representation in Section 3.2. In Section 3.3, we will introduce sensor models for laser range finders and RGBD cameras. The computation of surface correspondences in described in Section 3.4. In Section 3.5, we present the construction of the minimization problem to compute the maximum likelihood configuration of sensor poses and surface points. An extensive evaluation can be found in Section 3.6, followed by a discussion of relevant related work in Section 3.7 and our conclusions in Section 3.8.

3.1 Overview

The goal of our approach is to construct a consistent high-resolution 3D model of the environment from a set of roughly aligned 3D range scans. The input of our approach can be obtained from a traditional SLAM algorithm. Our approach is able to compensate for small errors in the sensor pose and it takes into account the noise affecting the range measurements. Typical man-made environments consist of regular surfaces and a range reading can be understood as sample generated by the underlying observed surface. We exploit the regularity assumption by approximating a surface by a set of small locally planar patches, characterized by their normals in direction to the sensor. We will refer to these as surface elements (*surfels*).

The main idea is to construct an optimization problem that adjusts the poses of the sensor and of the surfels to find a maximum likelihood configuration. To achieve this task, we assess the surfels structure from each individual measurement endpoint. We then minimize the distance between nearby surfels acquired from different sensor poses. This is done, by taking into account the uncertainties of the sensor measurements via appropriate sensor models. We extract locally planar surfels from each range scan, as described in Section 3.2. Each measured distance is then connected to the corresponding surfel by a constraint that depends on the sensor used (laser range finder in Section 3.3.1 or RGBD camera in Section 3.3.2). Each surfel is parametrized by a state variable.

Nearby surfels arising from different scans are then connected by "virtual" measurements that "pull" the two surfels onto each other, while allowing them to slide along the tangential directions. We assume locally smooth and planar surfaces and that the tangential directions



(a) input point cloud **(b)** corresponding surfels

Figure 3.3: This figure illustrates our surface model. (b) shows the surfels extracted from the measured points shown in (a), together with the normal computed from the corresponding local neighborhood.

provide a good a approximation of the local surface around a surfel. These virtual measurements are iteratively refined, as explained in Section 3.4. Finally, in Section 3.5 we will explain how to construct a sparse optimization problem that incorporates all the above constraints. Despite the high number of variables these problems can be solved in a relatively fast manner by using modern optimization techniques.

3.2 Surface Elements

In our approach, we represent the surface of a 3D range observation as a set of surfels. We assume the sensed surface to be piecewise smooth and that we are able to extract local normals around the endpoints of a scan. We represent every single measurement endpoint by a small planar region, which gives a local description of the surface. The local characteristics of a surfel are modeled by a Gaussian distribution and are calculated from the neighboring endpoints of the same observation within a global coordinate frame. These Gaussians encode the orientation of the normal and how well the local surface can be represented by a plane. The mean μ_{nk} of a Gaussian represents the center of the surfel that is measured by the k^{th} measured endpoint $\mathbf{r}_{nk}$ that originates from the n^{th} sensor pose $\mathbf{x}_n$. We initialize μ_{nk} with

$$\mu_{nk} = \mathbf{x}_n \oplus \mathbf{r}_{nk} \tag{3.1}$$

and compute the covariance based on the endpoints in the local neighborhood of μ_{nk} within the same scan. Once the Gaussian is computed, the estimated normal of the surface $\hat{\mathbf{n}}_k$ is the eigenvector of the smallest eigenvalue of the covariance matrix oriented towards the sensor. Figure 3.3(b) shows an example of the computed local characteristics for the corresponding point cloud shown in Figure 3.3(a).

Since we update the poses of the surfels during the optimization procedure we need to re-compute the local characteristics after each optimization run. We cannot avoid the re-computation by choosing a relative coordinate frame because the surfel optimization changes the relative configuration of the neighborhood.

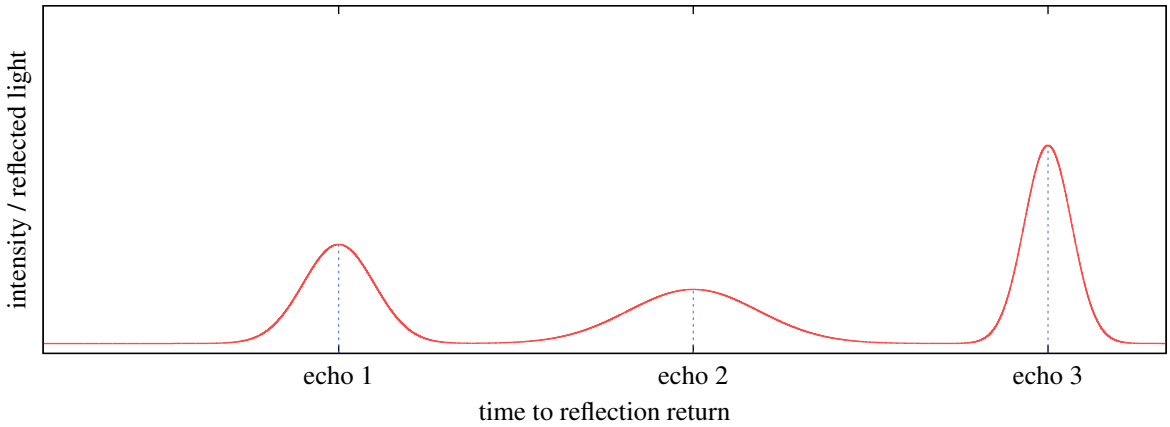


Figure 3.4: This plot illustrates the dependency between the time until a reflection is returned and the amount of reflected light for a single laser measurement. In this example, a laser range finder measures three echoes. The first and the second echo could reflect dust, fog, or rain, whereas the strongest echo reports a true surface in the environment.

3.3 Sensor Models

It is very important to constrain the region in which a surfel can be moved in a reasonable way while deforming a scan. Otherwise a model could be stretched or shrinked with wrong assignments in the data association. In the other extreme a scan would simply not be adjusted and the resulting model would not improve, given constraints with very small covariances. Therefore, we need accurate sensor models, which account for measurement uncertainties. These uncertainties depend on the sensing technique as well as on the specific parameters of the used sensor. In the following we present a sensor model for laser range finders in Section 3.3.1 and a sensor model for RGBD cameras in Section 3.3.2.

3.3.1 Laser Range Finder

A laser range finder measures the length of a set of light beams emitted by a laser source. The length of a beam is obtained by measuring the time duration between the emission of a laser pulse and the return of its reflection. Due to the optical characteristics of the laser, a beam has a conic shape (see also technical description LMS151 (Sick AG, 2012a)). For this reason, a single laser beam does not measure the surface at a specific point, but rather returns an aggregated measure of the distances of the surface within the spot of the beam. Every surface or obstacle that is hit by the beam returns an echo. Modern laser range finders are able to distinguish several echoes and return either the first, the strongest or multiple echoes. Each echo corresponds to a peak in the measured light intensity at a given time, as illustrated in Figure 3.4. Fog, dust and rain can cause echoes that do not report a surface in an environment. Due to ambient light, the intensity is typically larger than zero even if no reflection is measured. Identifying several echoes improves the accuracy since the range is only computed on the reflections around the chosen echo instead of averaging over the return times of all reflections. Throughout our experiments, we experienced that the accuracy of a range measurement depends on the range and therefore on the size of the beams spot on a surface. A possible explanation is that a larger spot means that less light will return to the sensor and the signal to noise ratio becomes worse. Figure 3.5 illustrates the range dependency of laser spot diameters for several laser range finders and their relation to the angular resolution. This effect is usually negligible when one is interested in low-resolution maps or the robot operates in narrow environments only, but it

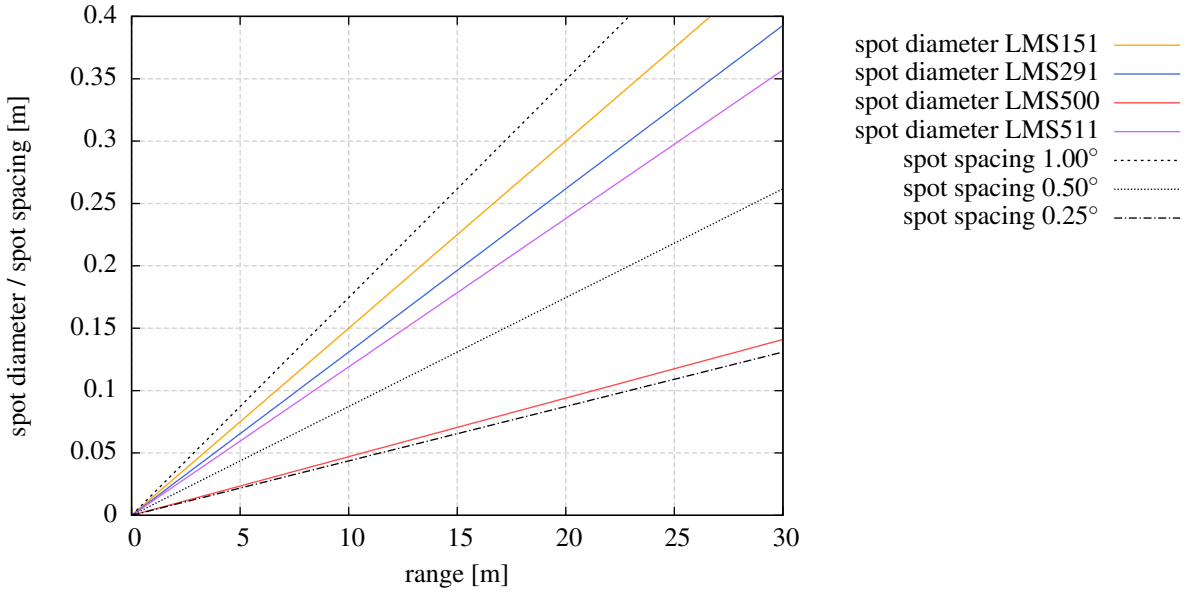


Figure 3.5: This plot shows the range dependent behavior of the beam spots for several different laser scanners and the distance between neighboring spot centers for 0.25°, 0.5° and 1°. All plotted lasers have spot diameters that are wider than 0.5° except of the LMS 500. The typical angular resolution of a laser range finder is between 0.1° and 0.5°. As consequence the beam spots overlap for neighboring range measurements. Source Sick AG (2006)

becomes evident in all other situations.

Furthermore, the incidence angle on the surface usually plays a major role in the error affecting a laser beam. This is another consequence of the conic shape of the beam: the area of the spot on the surface increases with the angle between the normal of the surface and the center of the beam. Since the distance measure is obtained by averaging over a larger region, it tends to be less accurate. The main intuition behind this is that the variance in the measurements grows with the region the spot covers and as a result the corresponding echo gets less peaked. For example the second echo in Figure 3.4 is less peaked than the third echo and therefore, has a worse signal to noise ratio. Figure 3.6 shows an exemplary laser scan where each beam has a range averaged over $2,429$ laser scans recorded at the same location. In this way, we can reduce the impact of statistical errors in the range readings. The remaining error is up to $1\,cm$ and has a systematic source. The uncertainty affecting a single laser beam hitting a surface typically depends on the systematic error η^{sys} of the device, the range, the incidence angle η^{inc} and a statistical error η^{stat}. The systematic error is influenced by ambient light and temperature and is device specific. Since this error dependents on non observed quantities, we use the typical systematic error provided in the data sheet of the corresponding device to account for this error (see Table 3.1). We assume a uniform distribution of this error in our model. It might be possible to calibrate this error for a specific device and use less conservative parameters to perform high quality mapping tasks, but the general model stays the same.

The statistical error σ^{stat} seems to be range dependent as can be seen in Figure 3.7. For both plots we see an increase in the measured standard deviation σ with growing range. Therefore we model σ^{stat} as linear function between a minimum deviation at short-ranges and a maximum deviation at the maximum range r^{max}.

$$\eta^{\text{stat}} = (\sigma^{\text{stat}}_{\text{max}} - \sigma^{\text{stat}}_{\text{min}}) \frac{r_k}{r^{\text{max}}} + \sigma^{\text{stat}}_{\text{min}} \tag{3.2}$$

It would also be possible to fit a polynomial function through our measurements, but again this

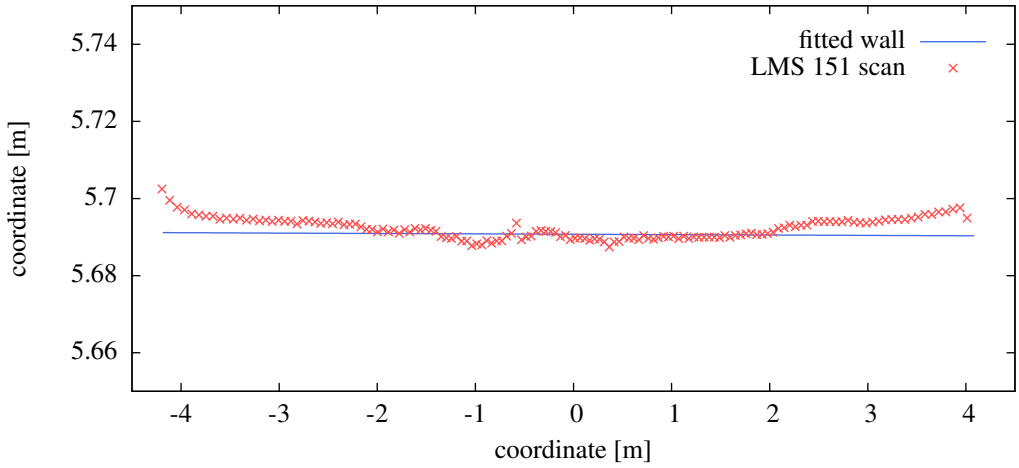


Figure 3.6: This figure shows an averaged laser scan over 2,429 individual scans of a 8 m wide wall captured with a stationary robot at a distance of ≈ 5.69 m. We assumed a planar wall and fitted a line into the scan as reference wall. As can be seen the center of the scan gives an accurate description of the wall whereas the outer parts tend to be up to 1 cm away from the wall. This is related to the increased range of the beams which hit the wall and the angle of incidence on the surface.

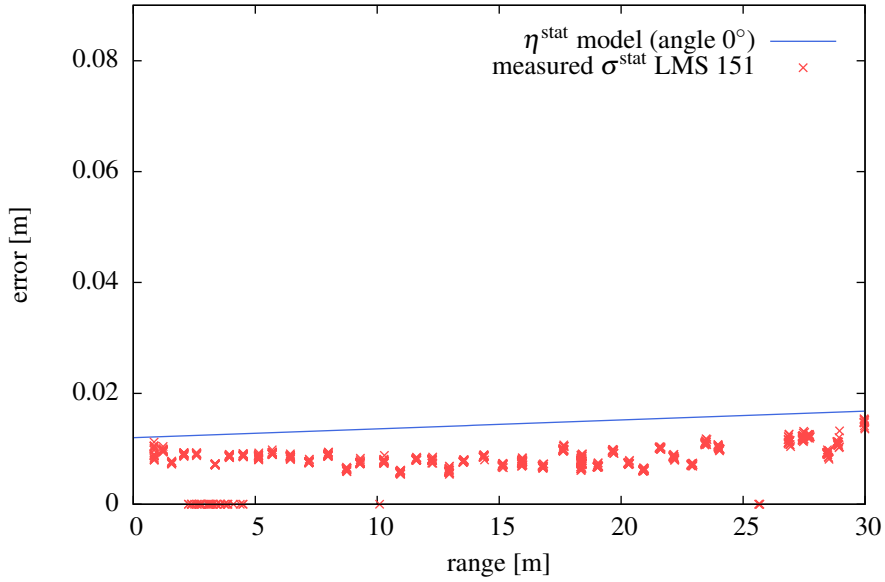
would mainly improve the results for one specific sensor and we are interested in an conservative upper bound for all laser devices of a type. For generating the plots in Figure 3.7, we recorded various datasets of a concrete wall with a distance between 1 m and 30 m. For every distance we collected over 2000 measurements and fitted a plain through the range measurements similar to Figure 3.6. We computed the average, the standard deviation and the incidence angle for every beam. For the plot, we selected all beams with an incidence angle of $0^\circ \pm 1$ (orthogonal to the wall, see also Figure 3.9). The standard deviations provided in the UTM-30LX data sheet (Hokuyo Ltd., 2012) does not seem to provide a good explanation for the measured deviations as can be seen in Figure 3.7. Since we recorded the data only with one UTM, it is unclear if this behavior is just specific for our sensor or if the noise behaves similar for all UTM-30LX.

The observed wall has a width around of 8 m and we observe the wall at various angles with different beams. Therefore, we can also evaluate the dependence between the incidence angle and the resulting errors for various ranges from the same dataset. Figure 3.8 shows two examples for the relation between incidence angle and range errors for two different laser range finders. As can be seen for larger incidence angles the average error in range grows rapidly.

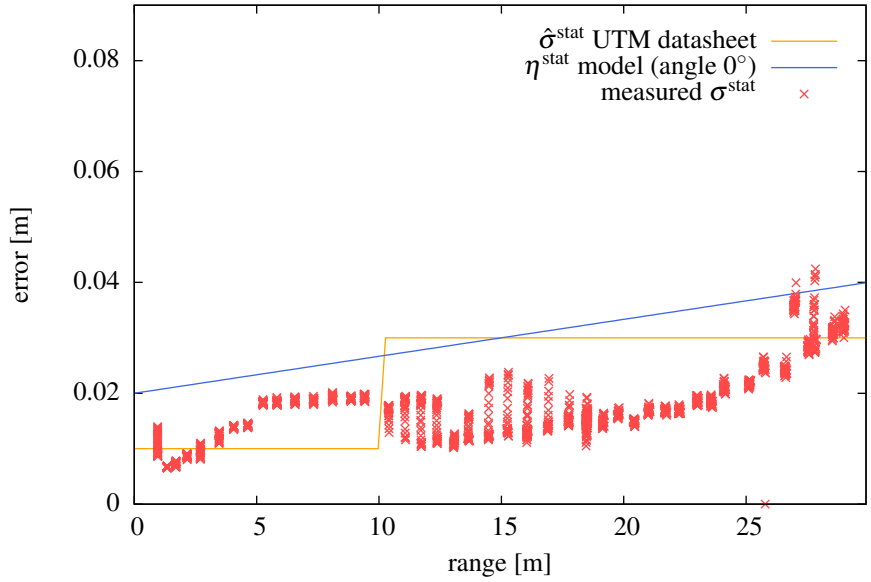
To model this error, we consider the geometric relation between the laser beam and the underlying surface, an illustration can be found in Figure 3.9. The diameter d_k^{surface} of the beams spot on the surface increases with the length of the measured beam and it depends on the incidence angle α_k to the surface. The spot diameter d_k is proportional to the beams opening angle β and to the norm $\|\mathbf{r}_k\|_2$ of the beam's vector $\mathbf{r}_k$. Note that β is usually larger than the angular resolution of the sensor and depends on the optical properties of the sensor. Figure 3.5 illustrates the relation between the angular resolution of the sensor and the optical opening angle for several laser range finders. Due to the wider opening angle of the beams compared to the angular resolution most laser scanners measure distances with overlapping neighboring beams.

The spot diameter is

$$d_k = 2\|\mathbf{r}_k\|_2 \tan\left(\frac{\beta}{2}\right), \tag{3.3}$$



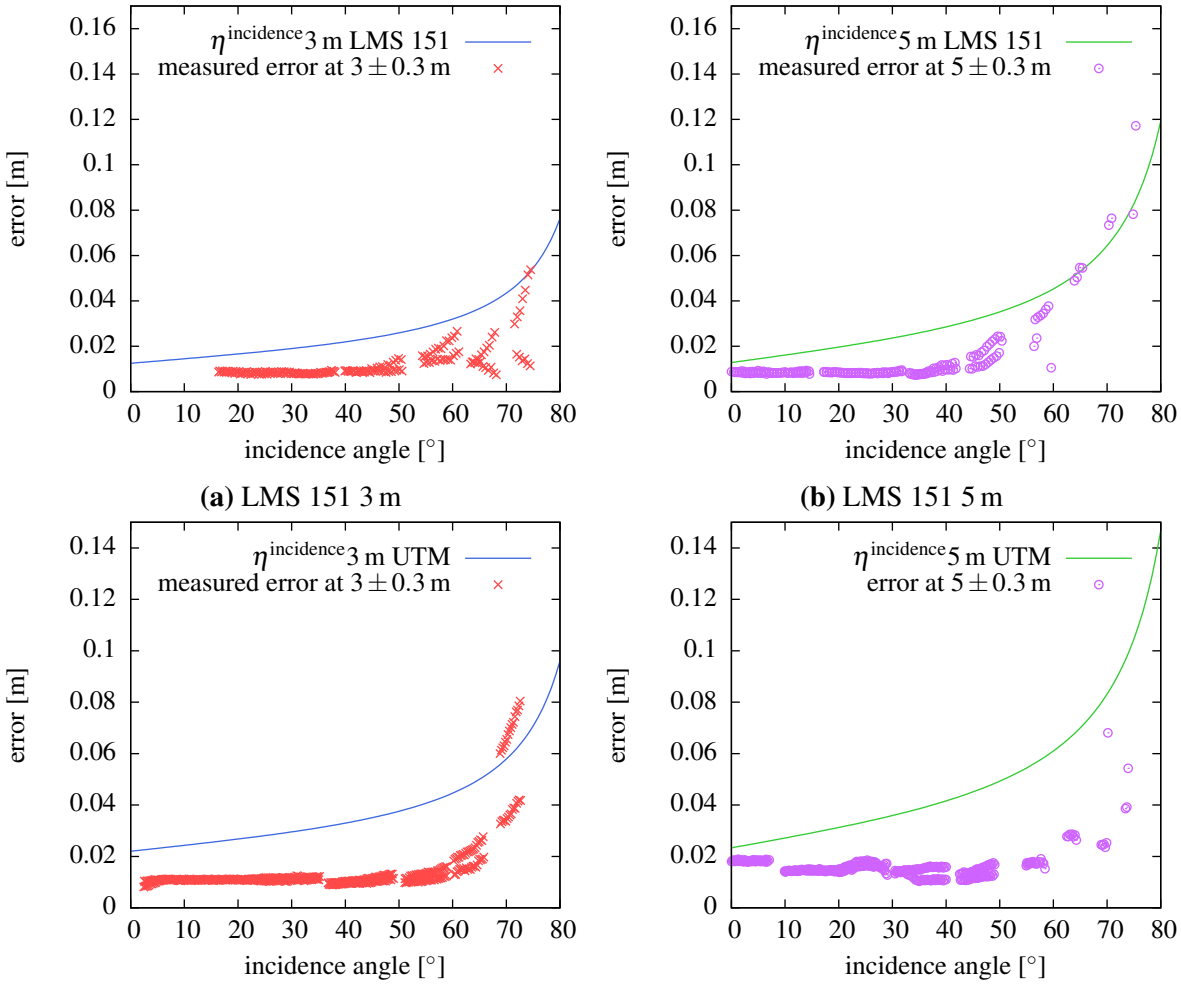
(a) LMS 151



(b) UTM-30LX

Figure 3.7: Overview of the predicted standard deviations η^{stat} of our model for LMS 151 (a) and UTM-30LX (b) laser range finders. For comparison we recorded laser measurements of a flat wall at different ranges with an incidence angle of 0°. To measure the standard deviation, we collected over 2,000 measurements at every range. In (b) we additionally show the standard deviation function provided in the data sheet Hokuyo Ltd. (2012)

.



(a) LMS 151 3 m **(b)** LMS 151 5 m

(c) UTM 3 m **(d)** UTM 5 m

Figure 3.8: This figure illustrates the dependency between the error in range and the incidence angle for certain ranges. The error in this plots corresponds to the average distance between ranges measured for a particular beam and the fitted line on the wall. (a) and (b) show data measured with an LMS 151 and the error our model predicts for incidence angles between 0° and 80°. Similar plots for an UTM-30LX are shown in (c) and (d). Note that we did not measure any ranges with an incidence larger 80°.

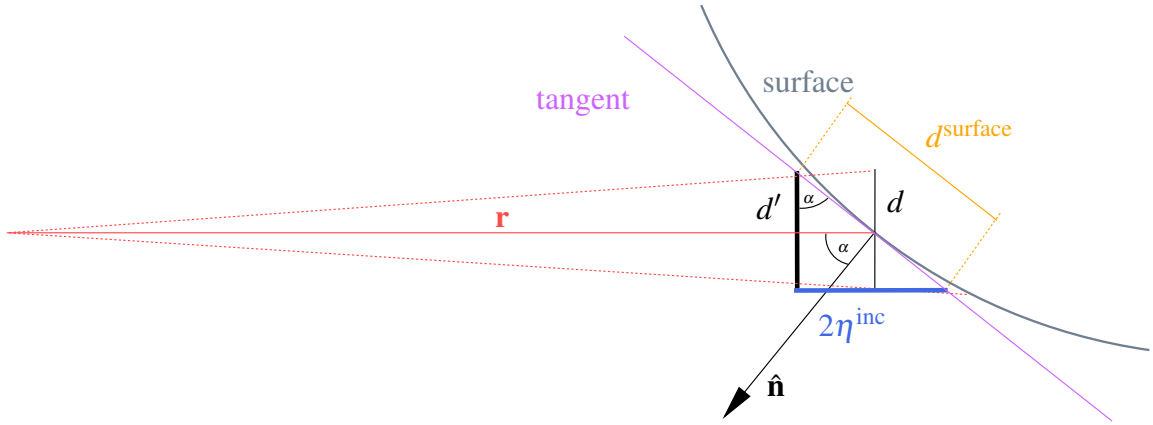


Figure 3.9: We assume smooth surfaces that can be well approximated with a tangent and were we can compute a normal. For convenience we dropped the indices. $\mathbf{r}$ corresponds to the beam of the laser vector. $\hat{\mathbf{n}}$ is the normal of the surface. α corresponds to the incidence angle. d is the diameter of the beam's cone on the surface at range $\|\mathbf{r}\|_2$. The intersections of the cones (red dotted) with the tangent (purple) defines a right triangle d', e, d^{surface}. d^{surface} corresponds to the diameter of the beam's spot on the tangent as approximation of the surface and e corresponds to the uncertainty along the beam's direction. We can compute e and d^{surface} with known angle and $d' \approx d$.

and the possible error in direction of the beam η^{inc} is

$$\eta^{\text{inc}} = \frac{d'_k \tan(|\alpha_k|)}{2}. \tag{3.4}$$

Assuming $d \simeq d'$, we can compute

$$\eta^{\text{inc}} \simeq \frac{d_k \tan(|\alpha_k|)}{2}. \tag{3.5}$$

If we have an estimate of the normal $\hat{\mathbf{n}}_k$ from our surface model and we obtain a laser measurement $\mathbf{r}_k$, the potential measurements will be distributed approximately as a Gaussian oriented along the beam's direction. The mean will lie at the center of the surface element and the uncertainty is represented by the covariance Σ^{meas}. The standard deviation σ_{11} along the beam's direction is proportional to the uncertainty along the beam's direction η^{stat}, the incidence related uncertainty η^{inc} and a quantity proportional to the systematic error η^{sys}

$$\sigma_{11} = \eta^{\text{stat}} + \eta^{\text{inc}} + \eta^{\text{sys}}. \tag{3.6}$$

The standard deviation σ_{22} along the direction orthogonal to $\mathbf{r}_k$ will depend on the radius of the spot:

$$\sigma_{22} = \tan\left(\frac{\beta}{2}\right) \|\mathbf{r}_k\|_2. \tag{3.7}$$

In the above equations σ_{11} and σ_{22} are the respective entries in the covariance matrix Σ^{meas}. Once computed we have to rotate the covariance according to the beam direction in the scans coordinate frame. Figure 3.9 illustrates how we compute the distribution of the point on a surface that generated a range measurement. In case of a 3D laser scanner, we extend the covariance with

$$\sigma_{33} = \sigma_{22} \tag{3.8}$$

assuming a symmetric laser cone. In case of a non-symmetric laser spot, β has to be split for both dimension and adapted accordingly. This is similar to what we describe in the next section.

sensor	β in rad	m. range	η^{stat} (min/max)	η^{sys}
LMS 151 (Sick AG, 2012a)	0.015	50 m	$1.2/0.2$ cm	30 mm
LMS 291 (Sick AG, 2006)	≈ 0.01274	80 m	$0.1/0.2$ cm	35 mm
LMS 500 (Sick AG, 2012b)	0.0047	80 m	$0.07/0.09(20\,\text{m})$ cm	35 mm
LMS 511 (Sick AG, 2012b)	0.0119	80 m	$0.06/0.14(30\,\text{m})$ cm	35 mm
UTM-30LX (Hokuyo Ltd., 2012)	$(0.0174)^*$	30 m	$(0.1/0.3\,\text{cm})^{**}$	50 mm

Table 3.1: Example specifications and corresponding covariances for different standard laser types. The β value of the LMS 291 is read out of the plot on page 7 in the data sheet (Sick AG, 2006). *Note that there is no information about the opening angle in the UTM-30LX data sheet and that the opening angle is usually larger than the angular resolution. ** The reported standard deviations did not match our experimental results. We used $0.02/0.04$ m as parameters for our sensor model.

Table 3.1 gives an overview over the sensor specific parameters for our sensor model. For the systematic error, we chose the average error values for our uncertainties. Note that the UTM-30LX lacks a description of the optical properties and we assumed 1° as upper bound for the opening angle.

In Figure 3.10, we plotted σ_{11} for an LMS 291 dependent on the incidence angle for various ranges. As can be seen in Figure 3.10(a) the influence of the incidence angle grows with the range. An incidence angle of 80° has only minor impact at a range of 1 m on the uncertainty. In contrast for a range of 30 m the uncertainty is more than 0.4 m. A close up view of the uncertainties around an incidence angle of 0° is shown in Figure 3.10(b). A comparison of the uncertainty functions for various lasers at a range of 30 m is shown in Figure 3.10(c) and Figure 3.10(d). The UTM-30LX has the largest uncertainty and is already close to its maximum range. Due to the small opening angle of the laser beam, the LMS 500 seems to be the laser range finder that is most certain for incidence angles larger 10°.

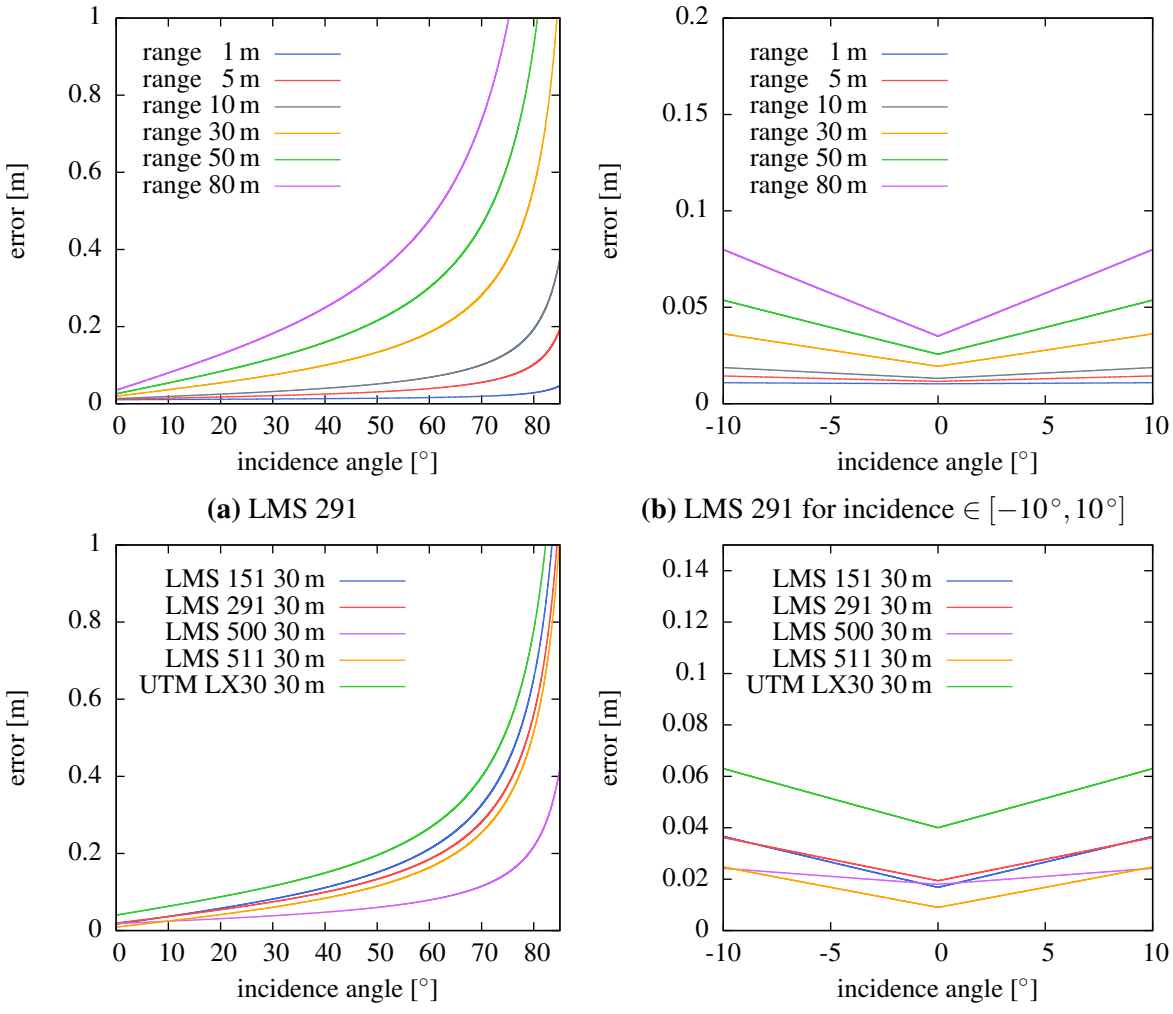
3.3.2 RGBD Cameras

In case of RGBD and stereo cameras, the depth value of a point in the scene is obtained from at least two projections of that point measured in the image planes of two different observers. In case of a stereo camera these two observers are in fact two cameras, and the correspondences between the two images are found with a stereo matching algorithm. The Kinect sensor approaches the problem by coupling an infrared projector and an infrared camera, thus making the correspondences more robust and easier to find. In this case the projector can be seen as an additional observation point. The displacement between two projections of the same point in the two image planes is called disparity and is quantized in sub-pixels, thus suffers of a systematic error. The depth is inversely proportional to the disparity. Therefore, the impact of the quantization error in the disparity on the depth estimate increases with the range.

According to the ROS kinect calibration tutorial by Konolige and Mihelich (2011), the range r of the Kinect is calculated using the following equation:

$$r = \frac{q_{\text{pix}} b f}{d_{pix}}. \tag{3.9}$$

Here q_{pix} is the subpixel resolution of the device in the calculation of the disparity, b is the baseline, f is the focal length and d_{pix} the normalized disparity. In the case of the Kinect $q_{\text{pix}} = 8$. Accordingly b and f should come out of the intrinsic calibration of the camera. Default values are $f = 525$ and $b = 0.075$m. If we transform this equation to get the disparity



(a) LMS 291

(b) LMS 291 for incidence $\in [-10°, 10°]$

(c) Comparison of lasers from Table 3.1

(d) Closeup comparison at 30m

Figure 3.10: The plots show the predicted standard deviations of our sensor model for the parameters of the LMS 291 and dependent on the incidence angle for various ranges in (a) and (b). A comparison between various laser range finders for a range of 30 m is shown in (c) and (d). The laser with the smallest beam opening angle (LMS 500) has the lowest uncertainty for incidence angles larger 10°.

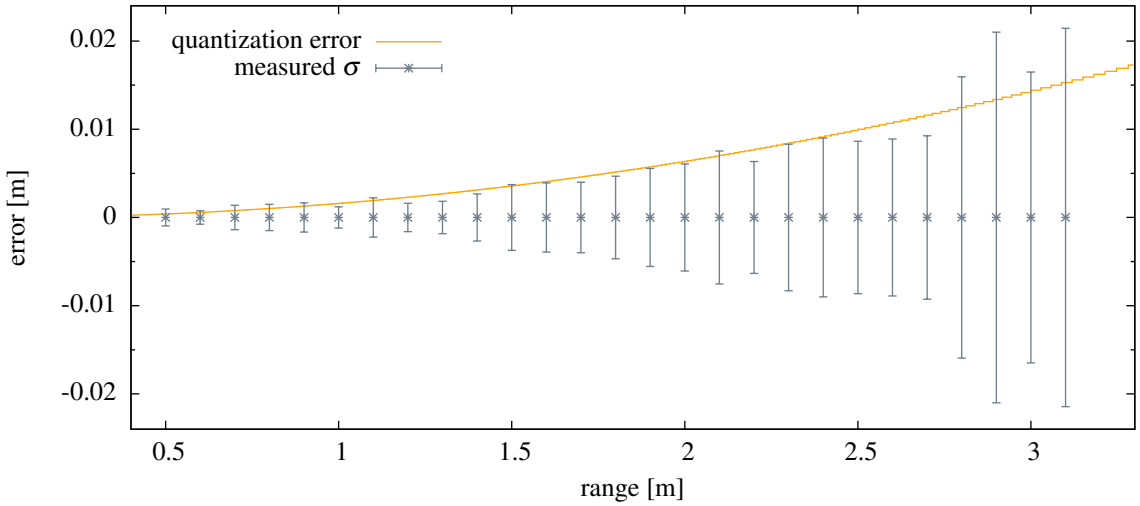


Figure 3.11: This figure shows the range dependent theoretical quantization error in orange. The standard deviation calculated from experimental Kinect data is shown in grey.

for a range r, we obtain

$$d = \frac{q_{\text{pix}} bf}{r}. \tag{3.10}$$

The quantization error can be assumed to be uniformly distributed, and the width η^{quant} of the distribution depends on the sensed range, according to the following equation:

$$\eta^{\text{quant}}(r) = \frac{q_{\text{pix}} bf}{2} \left[\frac{1}{\text{round}\left(\frac{q_{\text{pix}} bf}{r} + 0.5\right)} - \frac{1}{\text{round}\left(\frac{q_{\text{pix}} bf}{r} - 0.5\right)} \right].$$

Figure 3.11 illustrates the quantization error. We evaluated the statistical error of a Kinect facing a planar surface in a range between 0.5 m and 3.1 m. Every 0.1 m we acquired 100 scans and calculated the error to a fitted plane. The resulting standard deviation is shown in Figure 3.11. The standard deviations behave similar to the analytic error function. Note that the Kinect data have larger error bounds as the quantization error for ranges over 2.5 m. This depends on the plane fitting routine, whose performance decreases in presence of the high noise that occurs in this range leading to additional errors.

To construct the least squares problem we approximate the uniform distribution along the direction of an endpoint with a Gaussian, whose covariance is proportional to the variance of the uniform distribution of width η^{quant}. To be more detailed, we represent the uncertainty of a measured endpoint with a covariance Σ^{meas}. The covariance σ_{11} along the direction of k^{th} endpoint $\mathbf{r}_k$ is modeled as

$$\sigma_{11} = \eta^{\text{quant}}(\|\mathbf{r}_k\|_2). \tag{3.11}$$

An RGBD sensor observes neighboring measurements at the same time and there is no additional dynamic involved. Therefore, we assume a range dependent error between neighbors in X-direction of the image, modeled in σ_{22}, and the Y-direction of the image, modeled in σ_{33}. This error depends on the size of the surface "covered" by a pixel. Given the angular resolutions β_X, β_Y we can model the error as

$$\sigma_{22} = \tan\left(\frac{\beta_X}{2}\right) \|\mathbf{r}_k\|_2 \tag{3.12}$$

$$\sigma_{33} = \tan\left(\frac{\beta_Y}{2}\right) \|\mathbf{r}_k\|_2. \tag{3.13}$$

Here, σ_{11}, σ_{22}, and σ_{33} are the respective entries in the covariance matrix Σ^{meas}.

3.4 Surface Correspondences

In the Section 3.2, we described our surface representation for the individual endpoints of a single range scan by assuming that the sensed surface is locally regular. In this section, we will explain how we determine potential correspondences between regions of surfaces sensed from different sensor positions.

Given a surfel we search for potential correspondences between this surfel and all other surfels observed from different sensor poses within a search radius. Since our method is able to deform scans, we have to carefully select the search range. A too large range might result in stretched or fragmented surfaces and erroneous sensor poses introduced by false correspondences and a too small range might result in unconnected surfaces, which might lead to an inconsistent model.

Instead of searching a trade off between accuracy and robustness we combine two different strategies. To ensure the local connectivity we apply nearest neighbor (NN) in close range and to be more robust against initial alignment errors from missing or suboptimal loop closures we apply "normal-shooting" (NS) proposed by Chen and Medioni (1991) with a larger search range. Given an initial configuration of a surfel $\langle \mu_{ni}, \Sigma_{ni} \rangle$, whose normal is well defined, we search along its normal direction and in the nearby neighborhood to seek for the closest surfel belonging to a *different* observation. Let this surfel be $\langle \mu_{mj}, \Sigma_{mj} \rangle$. If the normals of the two surfels have a similar orientation, we add a constraint between them since it is very likely that they belong to the same surface. If the normals differ more than a given threshold (20 degrees in our current implementation), we assume that we hit another surface and we reject the match. As maximum search distance in the object model experiments we used 2 cm for normal shooting and 5 mm for nearest neighbor. In the environment model experiments we used 15 cm for normal shooting and 5 cm for nearest neighbor.

3.5 Least Squares Optimization

The overall goal of our approach is to determine the model and the configuration of robot poses that are maximally consistent with the observations. In the remainder of this section we propose error functions and error weights for the previously introduced constrains, with the goal to construct a least squares optimization problem.

The model $\mathcal{M}$ is represented as a set of surfels. Each surfel is described by a Gaussian $\langle \mu_{nk}, \Sigma_{nk} \rangle$, where μ_{nk} denotes the k^{th} measurement of the n^{th} robot pose $\mathbf{x}_n$, as described in Section 3.2.

Each surfel $\langle \mu_{nk}, \Sigma_{nk} \rangle$ is connected to the robot pose by a measurement $\mathbf{r}_{nk}$. This error is distributed according to the covariance Σ^{meas} computed either from the corresponding laser sensor model described in Section 3.3.1 or the RGBD camera sensor model discussed in Section 3.3.2:

$$e^{\text{me}}_{nk} = ((\mu_{nk} \ominus \mathbf{x}_n) - \mathbf{r}_{nk})^{\top} (\Sigma^{\text{meas}}_{nk})^{-1} ((\mu_{nk} \ominus \mathbf{x}_n) - \mathbf{r}_{nk}). \tag{3.14}$$

The error of corresponding surfels ni and mj belonging to *different* scans is modeled with the following error vector:

$$\mathbf{v}^{\text{cor}}_{nimj}(\mu_{ni}, \mu_{mj}) = \mu_{ni} - \mu_{mj} = \Delta\mu_{nimj}. \tag{3.15}$$

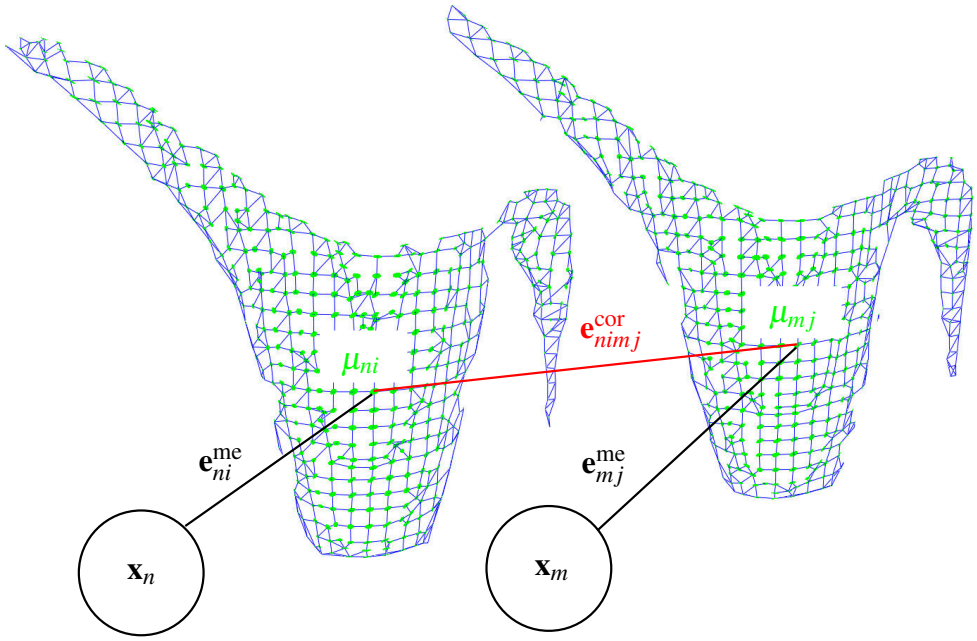


Figure 3.12: This figure illustrates the graph structure of our optimization problem. $\mathbf{x}_n$ and $\mathbf{x}_m$ are the sensor poses from which two 3D scans have been acquired. Two corresponding surfels μ_{ni}, and μ_{mj}, extracted from the different scans are connected by a "virtual" measurement $\mathbf{e}^{\text{cor}}_{nimj}$. A measurement generated by a surfel depends on the surfel characteristics and on the position of the sensor. In this example a beam's measurement of the patch μ_{ni} sensed from $\mathbf{x}_i$ is captured by the error function $\mathbf{e}^{\text{me}}_{ni}$. Similarly, a measurement of μ_{mj} sensed from $\mathbf{x}_j$ is captured by the error function error function $\mathbf{e}^{\text{me}}_{mj}$.

Accordingly, the error is minimal if both surfels are in the same location. We intend to allow surfels to "slide" onto each other along their tangent directions, while we want them to be more constrained along their normals. This can easily be modeled by the sum of the inverses of the covariance matrices of the surfel ellipsoids:

$$\Omega^{\text{cor}}_{nimj} = \Sigma^{-1}_{ni} + \Sigma^{-1}_{mj}. \tag{3.16}$$

The quadratic error function then is

$$e^{\text{cor}}_{nimj} = \Delta\mu^{\top}_{nimj}\Omega^{\text{cor}}_{nimj}\Delta\mu_{nimj}. \tag{3.17}$$

An odometry measurement $\mathbf{u}_n$ between two consecutive robot poses $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ contributes to the error function by the following term:

$$\mathbf{e}^{\text{od}}_i = (\mathbf{u}_i \ominus (\mathbf{x}_{i+1} \ominus \mathbf{x}_i))^{\top}\Sigma^{-1}_i(\mathbf{u}_i \ominus (\mathbf{x}_{i+1} \ominus \mathbf{x}_i)). \tag{3.18}$$

With the introduced quadratic error functions, we can formulate a least squares minimization problem to find the configuration of sensor poses $\mathbf{x}^*_{1:n}$ and surfels $\mu^*_{1:n,1:k}$ that minimize the following function:

$$\left\langle \mathbf{x}^*_{1:n}, \mu^*_{1:n,1:k} \right\rangle = \underset{\mathbf{x}_{1:n},\mu_{1:n,1:k}}{\operatorname{argmin}} \sum_{\langle n \rangle} e^{\text{od}}_n + \sum_{\langle n,m,i,j \rangle} e^{\text{cor}}_{nimj} + \sum_{\langle n,k \rangle} e^{\text{me}}_{nk}, \tag{3.19}$$

where we consider the odometry measurements between two successive frame acquisitions $\mathbf{e}^{\text{od}}_n$ only if it is available.

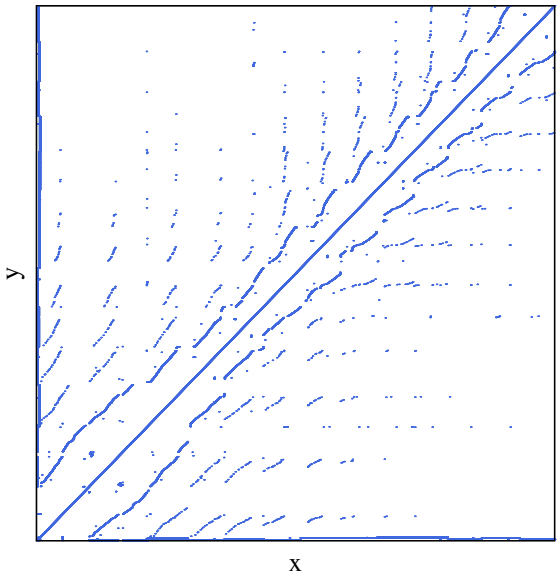


Figure 3.13: Non-zero entries of the approximated Hessian obtained by our optimization procedure. The "dense" bands on the bottom and on the left correspond to constraints between the robot poses and the surfels.

Relevant 3D datasets typically have millions of measurements and robot poses, which results in an optimization problem with millions of variables. Nonetheless, the obtained optimization problems are usually relatively sparse. The objective function is the sum of factors involving only pairs of state variables. Thus, the approximated Hessian contains a number of non-zero entries that is proportional to the number of constraints. The number of introduced constraints inside an observation is linear in the number of measurements. Since the applied sensors have a limited field of view and a limited range, the constraints between observations are in general only local and result in a sparse approximated Hessian. Therefore, we can efficiently compute this minimum by utilizing the g^{2}o framework (Kümmerle et al., 2011), which applies sparse linear algebra libraries. Figure 3.13 shows the typical non-zero pattern of the sparse Hessian.

Since we do not know the true correspondences, we have to iteratively refine the correspondences after every optimization run. Additionally, we recompute the surfel properties for the updated system and construct a new optimization problem. We perform this procedure until the changes in the objective function fall below a threshold or the maximum number of iterations has been reached.

3.6 Experiments

In this section, we evaluate our approach on 2D and 3D real-world datasets and discuss its advantages. The main purpose of our approach is to improve the consistency of models. We evaluate this consistency both visually and quantitatively by measuring the entropy of the generated models. For further quantitative evaluations, we used a simulated map resembling the Freiburg indoor building 079 map.

3.6.1 Improvement for 2D SLAM datasets

In the first experiment, we evaluated the impact of the combined pose and observation optimization on the following real-world datasets: Freiburg indoor building 079, the Intel Research

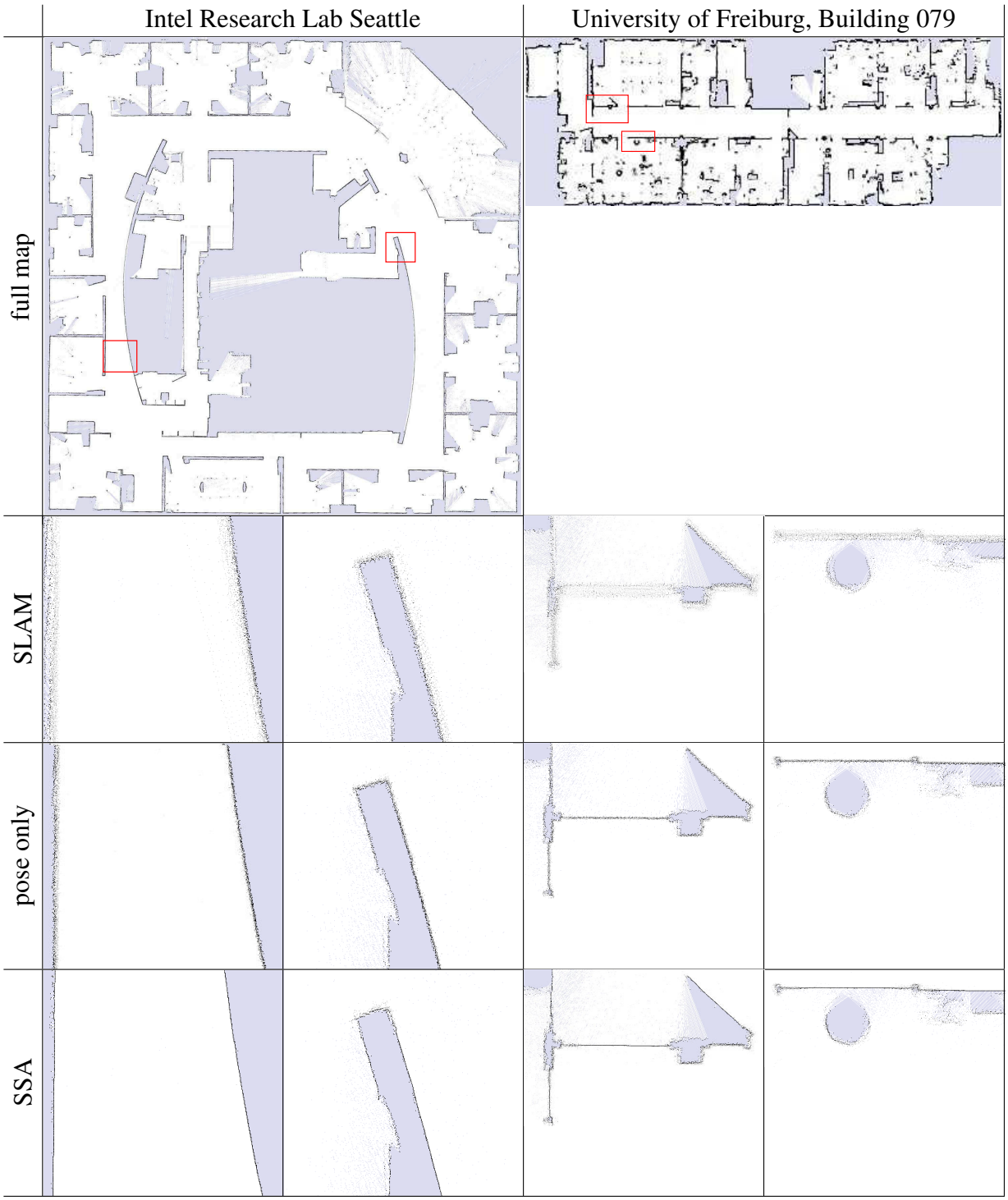


Figure 3.14: The top row of this figure shows the resulting maps for Intel and Freiburg 79 datasets. We enlarged some regions to visualize the differences in the SLAM results used as input (second row), the pose-only optimized result (third row), and the output of our method (bottom row). The bottom row shows only few blur and sharp borders between free and occupied cells.

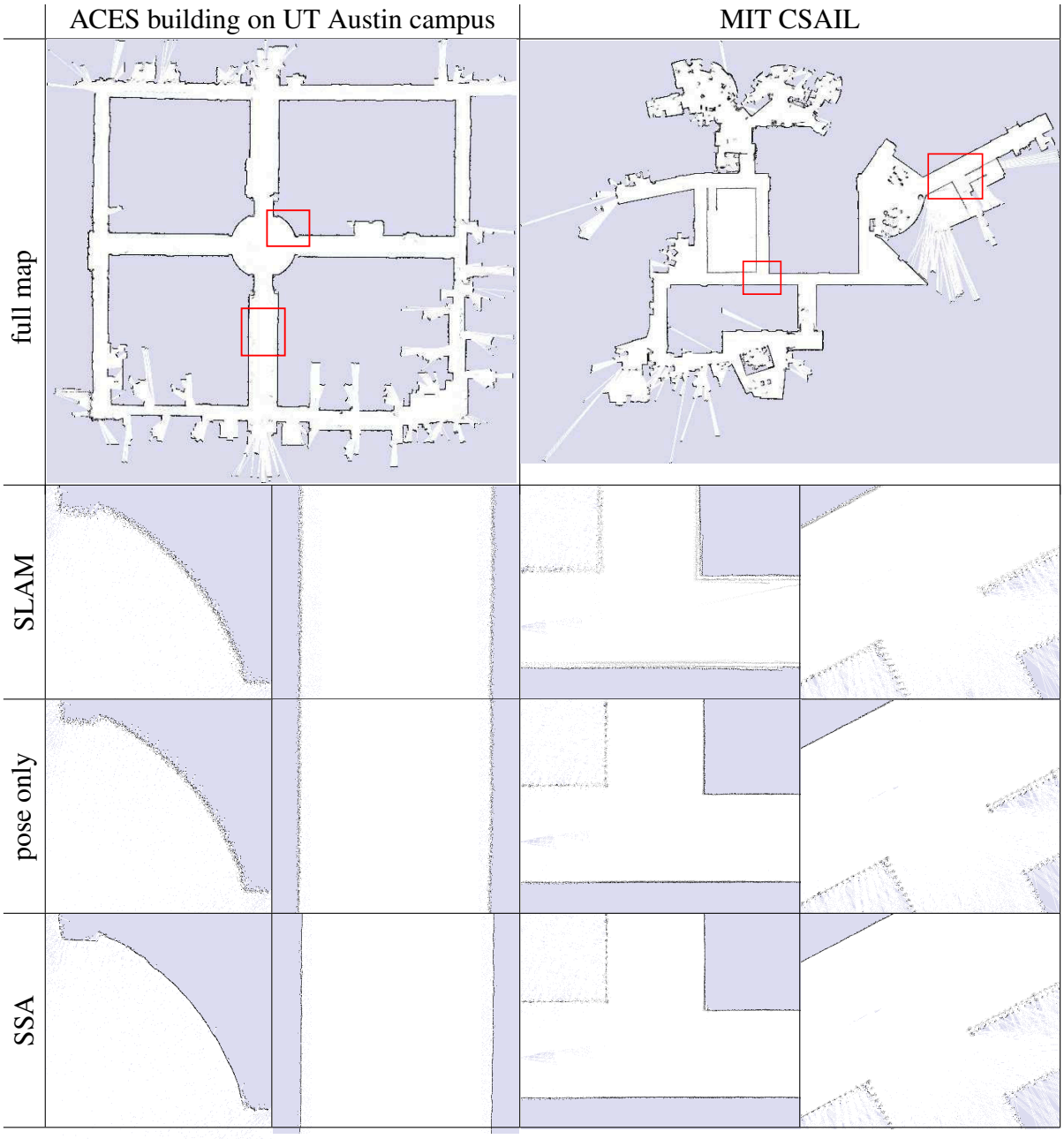


Figure 3.15: The top row of this figure shows the resulting maps for ACES and MIT CSAIL datasets. Again, we enlarged some regions to visualize the differences in the SLAM results used as input (second row), the pose-only optimized result (third row), and the output of our method (bottom row).

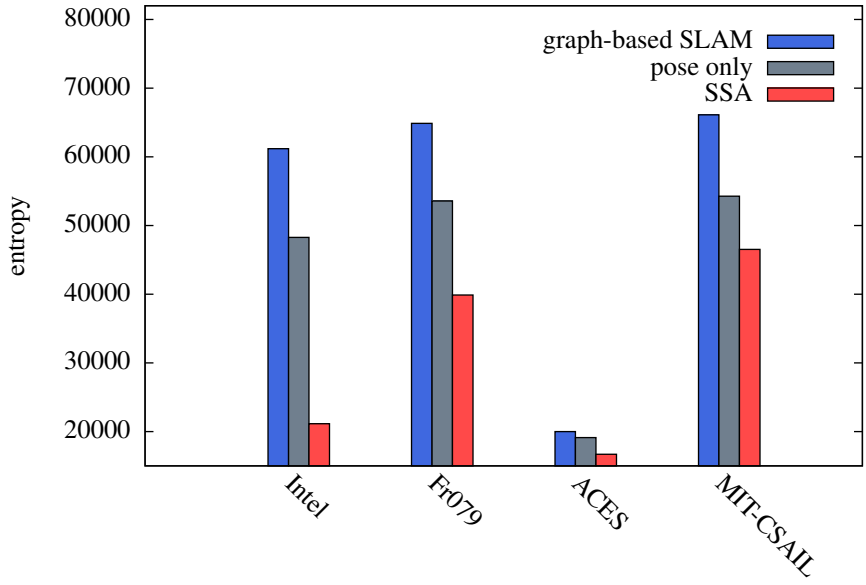


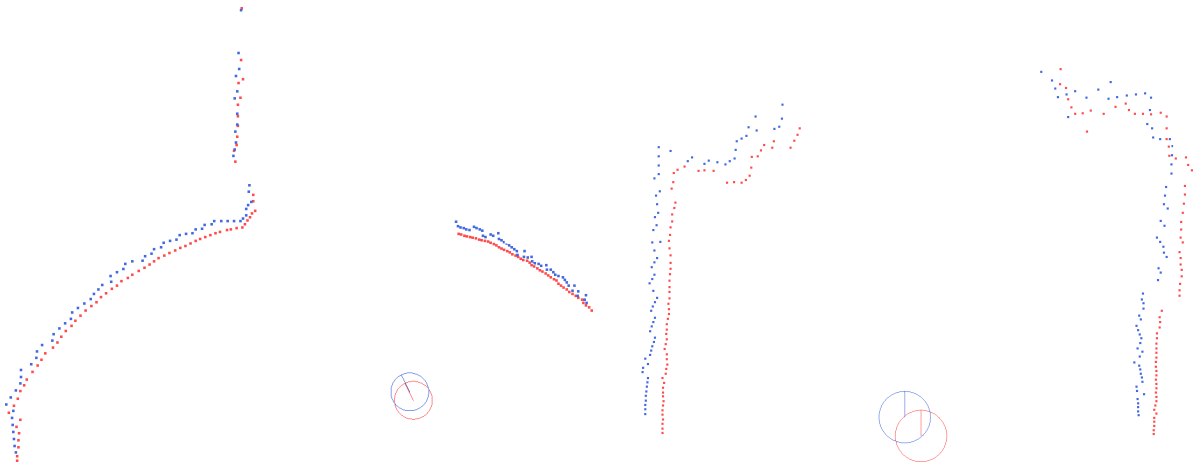
Figure 3.16: This figure shows the computed entropy for the first experiment. The entropy of the optimized result is lower than the entropy of the input data on all maps.

Lab, the MIT CSAIL Building, and the ACES Building. We choose those datasets because they are publicly available and well known in the SLAM community. Another important advantage of this dataset collection is the fact that it was acquired with different laser sensors and gives an intuition of the generality of our sensor model. The main purpose of our method is to produce accurate maps, which typically results in a low entropy value for the corresponding occupancy grid map. The entropy measures the uncertainty about the state of the environment. Our approach is not intended as a solution for the entire SLAM problem but rather serves as a post-optimization step that maximizes the accuracy of the map. In this experiment, we therefore computed a standard graph-based SLAM solution for each dataset and applied our method to the resulting estimates. The occupancy maps generated based on the estimates of the standard SLAM algorithm serve as a baseline for the quality of the maps obtained with our approach. To analyze the impact of the laser point optimization, we additionally generated maps using only the robot poses estimated by our approach but generating the maps from the original scans. We generated grid maps of 5 mm resolution for both methods as well as our approach and analyzed the resulting maps visually and numerically by measuring the entropy. We compute the entropy $H(M)$ of a grid map M as proposed by Stachniss et al. (2005) with the following equation:

$$H(M) = -\sum_{c \in M} p(c)\log(p(c)) + (1-p(c))\log(1-p(c)), \tag{3.20}$$

whereas c denotes to a grid cell of the map M and $p(c)$ is the corresponding occupancy probability of that grid cell. The first row of Figures 3.14 and 3.15 illustrate the resulting maps. The SLAM results used as input for out method are shown in the second row. The corresponding parts of the map for pose-only optimization and our approach are depicted in the third and the bottom row. As can be seen from the figure, our approach yields maps showing the highest accuracy compared to both other approaches. The entropy of the resulting model is plotted in Figure 3.16. Furthermore, our approach yields a substantially lower entropy value, which is consistent with the visual inspection of the models.

The lower entropy comes from the fact that the optimization of laser points reduces the impact of sensor noise. Figure 3.17 illustrates the differences before and after optimization on two partial laser readings taken from the ACES Building dataset. The optimized points



(a) (b)

Figure 3.17: Two partial scans of the ACES Building dataset before optimization (blue) and after optimization (red). In both cases the black points show typical sensor errors. In contrast the red points give a more likely explanation of the surface structure.

(red) give a visually better description of the observed structure than the original sensor reading (blue). The possible gain in entropy depends one the noise characteristic of the laser range finder and the size of the map.

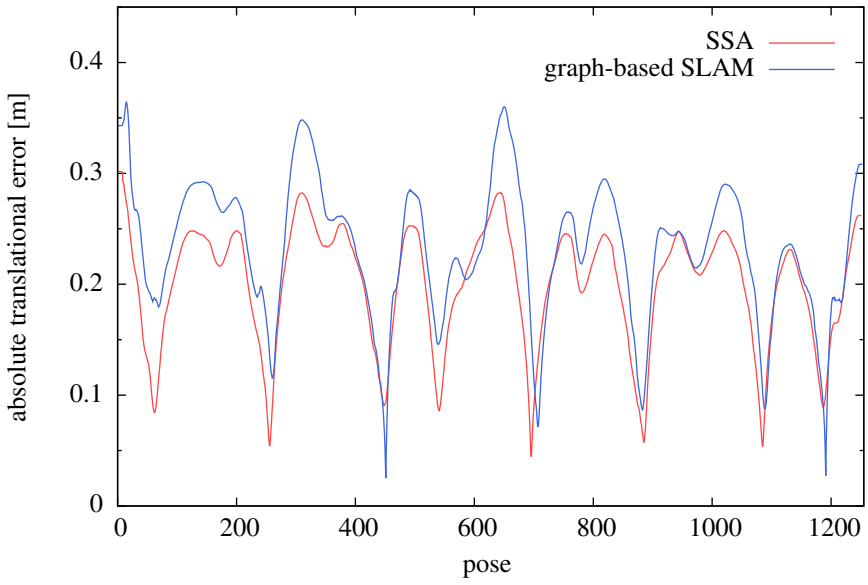
3.6.2 Localization and Map Accuracy

Since we lack precise ground truth for the real-world datasets, we evaluated the approach in a simulation scenario. Therefore, we measured the root mean squared error (RMSE) as described in the work of Olson and Kaess (2009). Additionally, we computed the relative error between localization pose pairs as additional measure. We refer to this measure as relative pose error (Burgard et al., 2009) in the following.

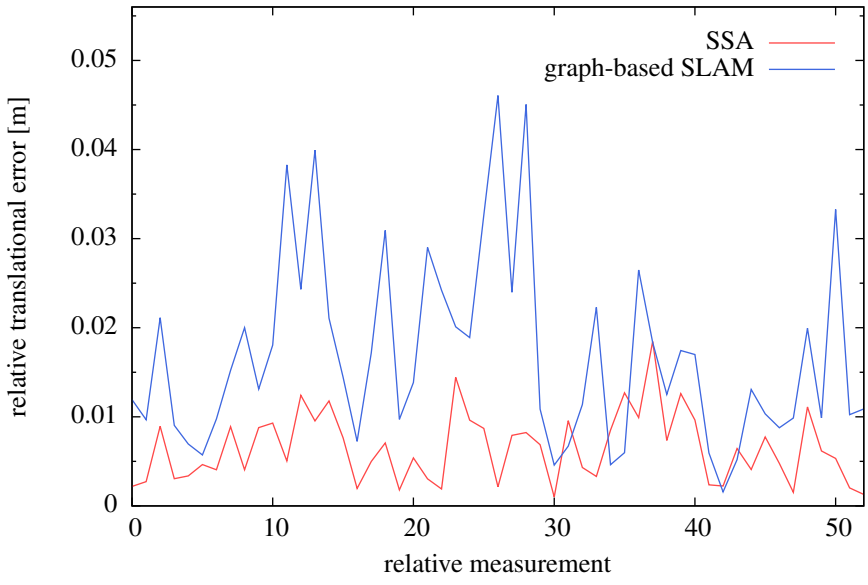
As first step of the experiment, we manually constructed a map with a cell size of 5 mm and recorded two datasets on this map using a simulator. From the first dataset we constructed a map with graph-based SLAM and a post-optimized map with our method. For both maps we ran a standard Monte-Carlo-Localization (Dellaert et al., 1999) given the second dataset and evaluated the corresponding localization accuracy. The plot in Figure 3.18(a) illustrates the absolute error for all poses calculated during the computation of the RMSE. This error plot represents the accumulated map and localization error and gives an upper bound to the global error in the map. The RMSE of the SLAM result is 0.0636 m and the RMSE of the optimized map is 0.0469 m. The plot in Figure 3.18(b) shows the relative pose error of localization pose pairs that were selected sequentially on the robot trajectory and have a distance of at least 1 m to each other. Both plots demonstrate that the localization gets substantially better on a map optimized with our approach than on a standard SLAM map.

3.6.3 3D Environment Models

Furthermore, we evaluated our method on three different real-world 3D datasets. The first dataset was recorded with a Kinect sensor on a mobile robot in the corridor of building 79 on the Freiburg campus. The second dataset was recorded in the same corridor using a tilting laser range finder. As third dataset we choose the publicly available AASS-loop dataset (Magnusson,



(a) absolute trajectory error



(b) relative pose error

Figure 3.18: Localization error on a graph-based SLAM (blue) and an SSA optimized map (red) in the simulator experiment. (a) shows the absolute errors for all poses. It gives an upper bound for the global error of the underlying map. (b) shows the relative errors between pose pairs that were selected sequentially on the trajectory with a distance of at least 1 m.

Table 3.2: Detailed overview of all datasets used in our experiments.

dataset	figure	# points	time	res.	entropy SLAM	entropy SSA	improvment
Fr079	3.14	218374		5mm	54864.2	39881.5	27.3%
Intel	3.14	344381	5 min	5mm	61172.1	21145.1	65.4%
MIT CSAIL	3.15	223581	6 min	5mm	66122.5	46514.3	29.7%
ACES	3.15	106346	3 min	5mm	19994.5	16694.5	16.5%
B. 079 RGBD	3.19	1,434,298	24 min	50 mm	174,092	143,087	17.8%
B. 079 Laser	3.19	688,977	11 min	50 mm	143,569	92,003.6	35.9%
AASS	3.20	2,266,519	48 min	50 mm	287,433	265,585	7.6%
Planar Surface	3.21	313,673	5 min	50 mm	12,493	6,911.43	44.7%
Black Cup	3.22	107,281	6 min	2 mm	8,184.52	6,781.88	17.1%

2007), which was recorded with a spinning laser range finder. We computed SLAM solutions for both corridor datasets and used the given SLAM solution of the AASS dataset as baseline for our consistency comparison. Afterwards we applied our method on all three datasets and computed the entropy on a 3D grid with cell size 5 cm. The first row of Figures 3.19 and 3.20 gives an overview of the used datasets. A detailed view of an example region before optimization and after optimization is shown in the second and third row. Table 3.2 gives detailed statistics of the involved datasets.

Our approach substantially reduced the entropy in all three datasets. The entropy of the Kinect dataset was reduced from 174,092 down to 143,087, which corresponds to a reduction of 18 %. The entropy of the Building 079 corridor dataset was reduced from 143,569 down to 92,003.6, which is a reduction of 35.9 %. The entropy reduction of 7.6 % on the AASS dataset is lower than on the other datasets, because of the very specific sampling strategy of the rotating laser. The performances of our approach depend on the characteristics of the environment, on the accuracy of the initial solution computed by the SLAM algorithm, and on the characteristics of the sensor. Clearly, the better the initial SLAM solution is, the more accurate the result of our model will be since there will be less ambiguities in the data association. Finally, the denser is the dataset, the better our approach can perform the alignment. For these reasons, the magnitude of the entropy reduction differs between the datasets but it is always positive.

There are two effects, which enable our approach to improve the results of graph-based SLAM algorithms. The optimization of a combined problem, using all available sensor information, instead of optimizing over fixed constraints, introduced by pairwise scan-matching and secondly the possibility to refine observations. Especially in the case of range dependent sensor errors, like the quantization errors of the Kinect, the measurement refinement is essential to build consistent models. As already stated in the introduction, we performed an experiment to evaluate the impact of this effect. To this end, we recorded a set of colored point clouds, while approaching a planar surface. The upper part of Figure 3.21 shows the complete model and the magnified view of an example region before and after optimization. The accumulated quantization effects of the model are substantially reduced after optimization. In this example, the entropy was reduced by 44 %.

3.6.4 Object Models

In our fourth experiment we demonstrate that our approach can also efficiently build consistent objects models up to a resolution of 1 mm. Therefore, we applied our method on an object

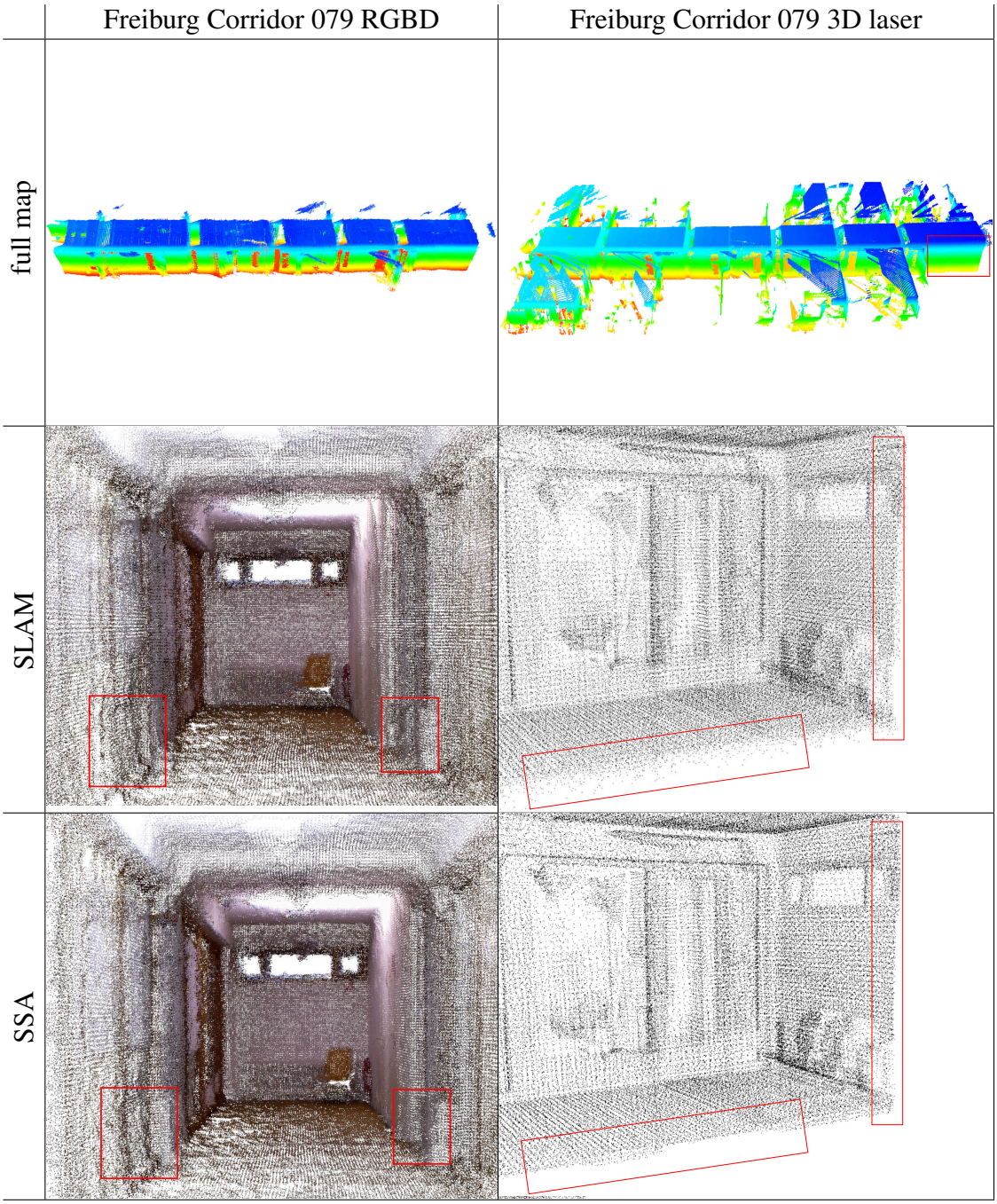


Figure 3.19: The first row illustrates two datasets captured in Building 079 on the Freiburg campus. The left dataset was recorded with a Kinect and the other dataset with a 3D laser (tilted 2D laser). The second row depicts closeup views of the datasets before optimization and the bottom row shows the same datasets after optimization. We highlighted with rectangles regions where the effect of the optimization is seen best. In the left example the double walls are optimized out and in the right example most of the noise around ground and walls is gone.

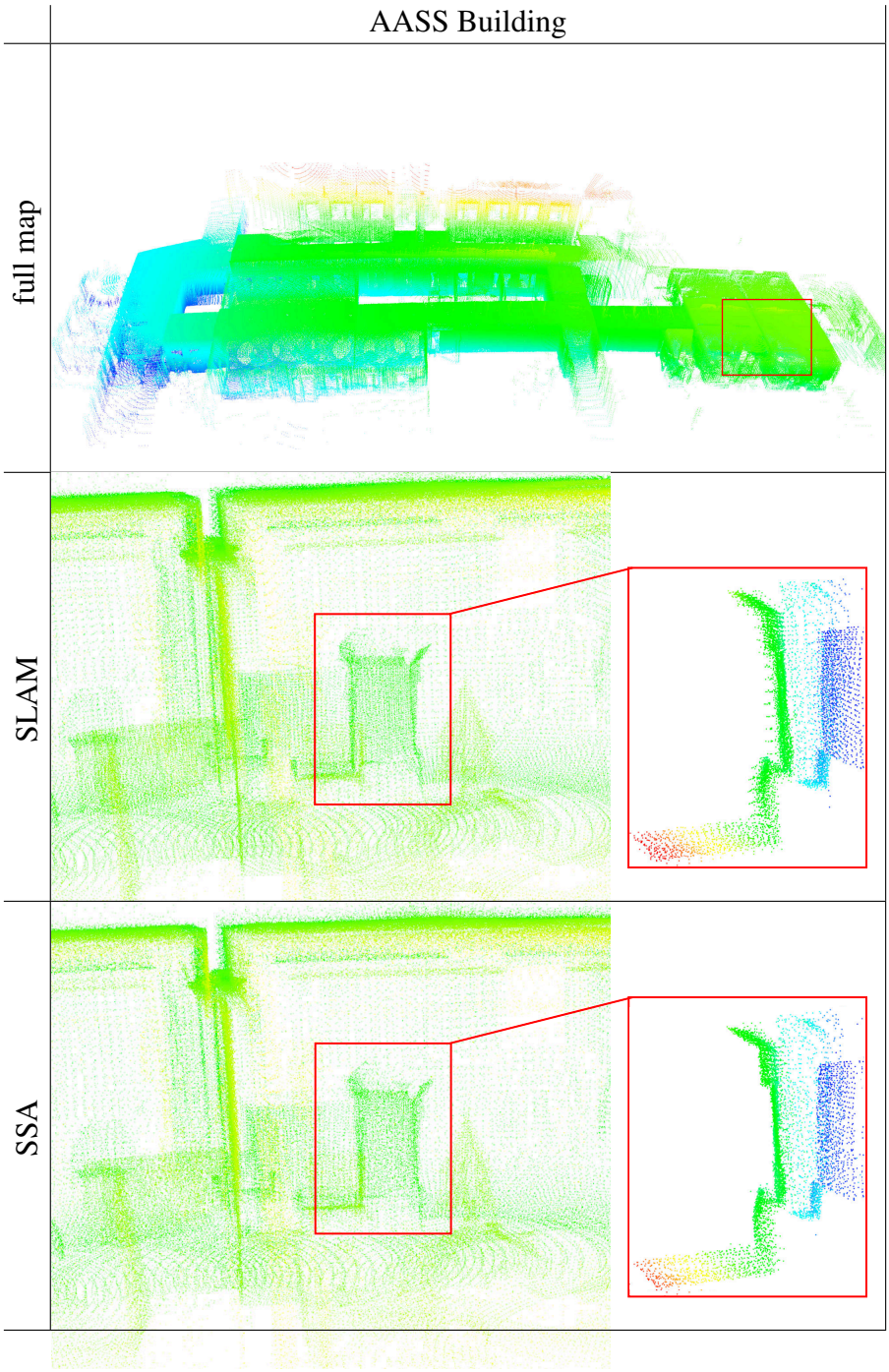


Figure 3.20: The first row shows an overview of the AASS dataset. The dataset was recorded with a 3D laser. The second row shows closeup views of the datasets before optimization and the bottom row shows the same datasets after optimization. We highlighted with rectangles regions where the effect of the optimization is seen best.

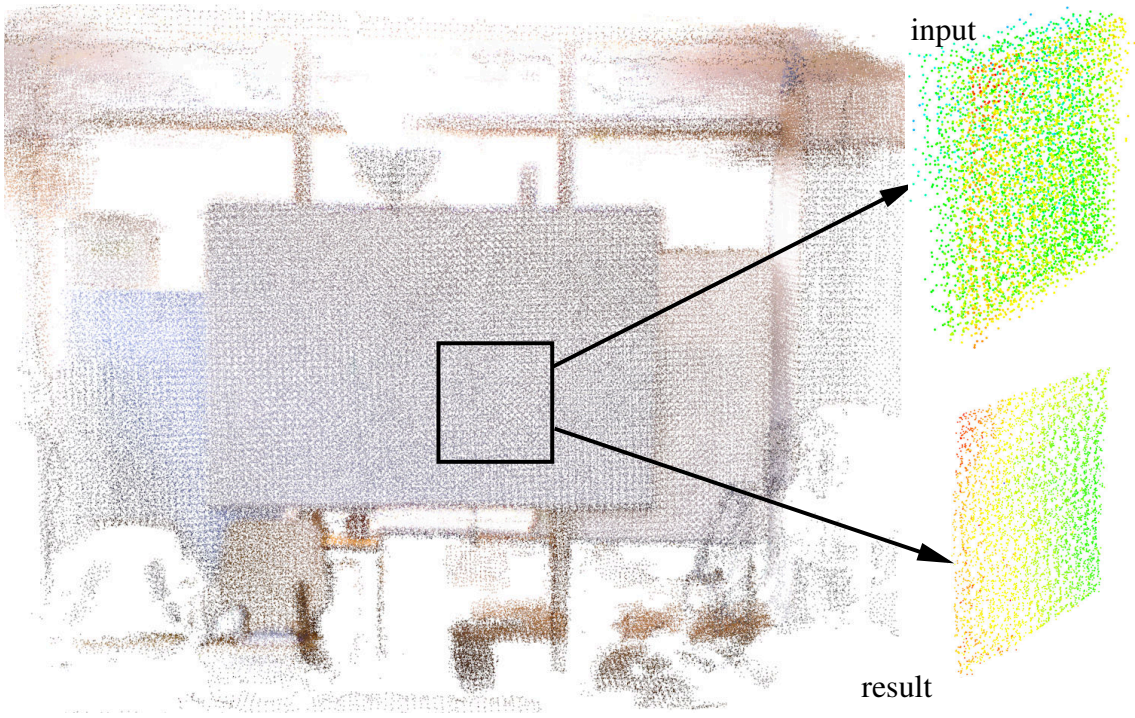


Figure 3.21: The left picture shows the point cloud of a model recorded in an office environment, which consists of 20 Kinect scans. The two magnified views on the right represent a part of the surface from a different viewpoint, whereas the top image depicts the input data and the bottom image shows the same region after optimization. The accumulated errors are substantially reduced.

model dataset, acquired with a Kinect on a rotating turntable. For the initial registration of the point clouds we applied incremental scan matching, introduced loop closure constraints, and optimized the result. Afterwards we applied our method and computed the entropy before and after optimization with a grid size of 2 mm.

Figure 3.22 shows the input data (a) and the resulting model for a black cup (b), computed on point clouds with a resolution of 1 mm. The resulting model looks more consistent. The entropy was reduced by 17.1 %. Additionally, we compared the outcome of our method with the results of the Moving Least Squares (MLS) (Fleishman et al., 2005) implementation provided in the Point Cloud Library (PCL) (Rusu and Cousins, 2011). Figure 3.22(c) shows the resulting point cloud for MLS applied on the input point cloud with default parameters and a search radius of 1 mm. Since MLS does not consider errors in the robot poses, the resulting model is not consistent.

3.6.5 Ground Truth Data

Obtaining a ground truth for models of arbitrary objects is rather difficult, thus we approached the problem with an object having a simple geometry. We decided to evaluate the accuracy achievable with our method on a geometrically simple object like a sphere with a known radius of 7.5 cm shown in Figure 3.23(a). The spherical geometry allowed us to sample a perfect object model as ground truth and the data association is more challenging since the normals provide no distinctive information. We recorded a dataset using a hand held Kinect capturing a metallic sphere on a table and measured the poses of the camera with a motion capturing system. The freiburg2/metallic sphere 2 dataset is publicly available here (Sturm et al., 2012). Afterwards, we selected reference scans every 0.2 m and manually removed all points not belonging to the

(a) object picture

(b) ICP aligned input

(c) Moving Least Squares result

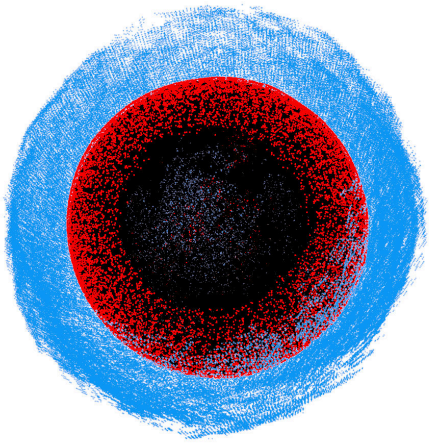
(d) Sparse Surface Adjustment result

Figure 3.22: This figure illustrates differences between Sparse Surface Adjustment and Moving Least Squares. Therefore we acquired multiple scans of the cup shown in **(a)**. Picture **(b)** shows the point clouds registered by using ICP and **(c)** illustrates the effect of MLS on the input data. Whereas the model looks smoother than the input it shows inconsistencies arising from errors in the estimate of the initial positions of the sensor. The picture in **(d)** depicts the results obtained by our approach on the same input data. The result appears to be more consistent.

(a) object used to acquire the data

(b) input model based on motion capturing poses

(c) result of SSA after 20 iterations



(d) overlay of input, SSA and a sampled sphere

Figure 3.23: For the direct comparison between the input model (blue) and the resulting model (red) we put both models into (d) together with a sampled sphere as ground truth (black). Note for visibility reasons the picture shows the bottom of the spheres, where the blue and red spheres have no points.

sphere. The accumulated model using the motion capturing poses can be seen in Figure 3.23(b). We used this model as reference for comparison and as input to our method. The result obtained with our technique can be seen in Figure 3.23(c). We aligned both spheres to a sampled sphere with the radius of 7.5 cm. The RMSE for the input model is 1.94 cm with a standard deviation of 1.1 cm and a maximum error of 4.43 cm. For our method the RMSE is 1.4 mm with a standard deviation of 0.7 mm and the maximum error is 6.5 mm. Therefore, our model is substantially more accurate than the input data as can also be seen in Figure 3.23(d).

3.7 Related Work

Several approaches have been proposed in the past to characterize the error affecting the result of scan-matching algorithms and thus to accurately estimate the constraints of the optimization problem. The standard technique for the registration of 3D range scan pairs is the Iterative Closest Point Algorithm (ICP) proposed by Besl and McKay (1992). ICP iteratively minimizes the distance between corresponding points, while updating the data association. Bengtsson

and Baerveldt (2003), proposed to analyze the residual of the re-projection error around the minimum by either sampling the error function or by approximating it by a quadratic form. Subsequently, Censi (2009) presented a closed form minimization algorithm to determine this covariance. All these approaches rely on point-to-point correspondences. Olson (2009b) proposed a hierarchical correlative algorithm derived from Konolige and Chou (1999) that can compute the histogram of the possible robot positions around the minimum. This method does not rely on specific point-to-point correspondences. Instead it uses a histogram approximation of the likelihood function on a grid. Segal et al. (2009) presented a variant of 3D ICP scan matching that minimizes the matching error between corresponding planar patches extracted from the input scans. Biber and Strasser (2003) proposed to solve the scan-matching problem by approximating the reference scan by a set of Gaussians. In this way, the alignment can be computed in closed form given the data association. Compared to the point-to-point criterion minimized by traditional IPC scan matchers (Besl and McKay, 1992; Lu and Milios, 1994) using the Gaussians allows to weight the error along different directions based on the shape of the matched surfaces. Subsequently, Magnusson et al. (2007) extended this approach to 3D.

Chang and Zwicker (2011) proposed a model reconstruction approach for articulated objects, which is able to automatically determine the joints of an object and deform scan parts according to the configuration of the object. This method also estimates the stiffness between surface points and deforms several scan parts to get a consistent model, but does neither consider the sensor uncertainties nor deform each scan point individually.

The Non-Rigid ICP variant proposed by Li et al. (2008) is able to deform all points on a surface graph (data) onto a second surface graph (model) by introducing a deformation model and applying least square optimization to find the optimum for deformation and registration. Since this method uses a weighted least square approximation for the model surface it is not easily extendable to the case of multiple scans and does not account for systematic sensor errors, while our method does not consider surface deformations.

Fleishman et al. (2005) presented Robust Moving Least Squares (MLS), a smoothing technique based on robust regression. They assign the points to piecewise smooth surfaces and are able to obtain accurate models. In a similar context Andersen et al. (2010) propose a Markov Random Field formulation that optimizes the parallelism between neighboring surface elements (surfels) and their overlap. This smooths out noise while maintaining sharp features. Both methods can be used to obtain accurate surface models, but ignore the model of the sensor that has been used to generate the point cloud and do not optimize over the sensor poses. Compared to our method, MLS is less robust to noisy initial configurations.

In this chapter, we presented an approach to determine the optimal position of both the points in the scans and the robot poses. Our method is similar in spirit to traditional bundle adjustment problems in computer vision (Triggs et al., 2000). A bundle adjustment algorithm seeks to find the configuration of a set of distinguishable features and camera poses that minimizes the re-projection error over the measured sequence of images. In contrast to bundle adjustment, our method does not require point features and it considers the local structure of the model while performing the optimization. The main difference between a bundle-adjustment algorithm and our method is that in our domain, a scan point is not a distinguishable physical point, being not generated by a specific feature, but rather a point sampled from the model's surface. Instead of optimizing the position of a feature based on n observations of that particular feature, our method optimizes the position of all observed points on a surface. We globally minimize a local plane-to-plane error metric similar to Segal et al. (2009). Furthermore, we refine the data associations after every optimization run. This yields models with highly accurate surfaces and drastically reduces the impact of accumulated sensor noise.

3.8 Conclusions

In this chapter, we described an approach that improves the results of traditional SLAM approaches by jointly optimizing robot poses and range data. Our method treats range readings as samples of a regular surface and is able to improve the consistency of 3D models by efficiently solving a least squares optimization problem that is constructed based on an accurate model of the sensor. For this purpose, we presented two sensor models, one for laser range finders and one for RGBD cameras. The laser sensor model takes into account the growing uncertainty with range and additionally the influence of the incidence angle. The RGBD sensor model accounts for the quantization effects introduced by the sub pixel discretization of the computed disparity. Experimental results obtained with a RGBD camera and a laser range finder in real-world 2D and 3D settings and for known objects demonstrate that our approach yields substantial improvements compared to a standard SLAM technique. Furthermore, we can apply this technique to high-resolution data to compute very accurate object models. So far, we focused on high accuracy models and did not consider the size of the model as relevant quantity. Of course, the size is important for a resource constrained mobile robot and the efficiency of algorithms that use a model. Therefore, we will focus on learning compact representations in the next chapters.

Chapter 4

Surface Primitives

In this chapter, we describe an approach to learn a compact representation for 3D range data. The key idea is to use a set of sample surfaces or surface primitives to describe 3D scenes. The surface primitives are organized in a reference dictionary. Each entry in such a dictionary is represented by a small sample point cloud and can be used several times to describe the observed surfaces. This leads to a highly compact description of an environment. The dictionary is directly learned on 3D data captured by a mobile robot. Furthermore, we optimize our model in terms of complexity and accuracy by minimizing the Bayesian information criterion (BIC). Experimental evaluations on large real-world data sets show that the described method allows robots to accurately reconstruct environments with as few as 78 words. Furthermore, the experiments suggest that the proposed representation gives a richer semantic description than purely occupancy-based representations.

♦ ♦ ♦ ♦ ♦ ♦ ♦

In the previous chapter, we proposed a technique to optimize the accuracy of models based on depth data. Throughout that section we used point clouds as data representation. Point clouds can represent data accurately but have high memory requirements. Given we have $n \in \mathbb{N}$ scans from a range sensor pointing at the same surface, a joined point cloud would store one point per scan, leading to n points for the same part of a surface. While this redundancy is essential to compute a super resolution model, we would like to have a more compact representation of an environment for resource-constrained mobile robots. Therefore, we will focus on representations that have a good tradeoff between accuracy and compactness in the following.

In this chapter, we present an approach to compactly represent 3D range data by determining repetitive patterns and using these patterns to reconstruct the environment. The key idea is to build a dictionary of common 3D patterns from dense 3D point clouds and to use this dictionary to compactly represent the entire scene. Besides reducing the amount of data, the approach has the potential to facilitate tasks such as place recognition or object detection. The idea to compress data by replacing multiple occurrences of the same data block by one word of a dictionary is at the root of classical compression approaches like, e.g., LZW (Nelson, 1989). These algorithms work on symbolic data and mostly focus on the loss-less reconstruction of the input without relying on additional knowledge about the data. In our context, the computationally intensive part is the recognition and the matching of recurrent patterns within noisy point

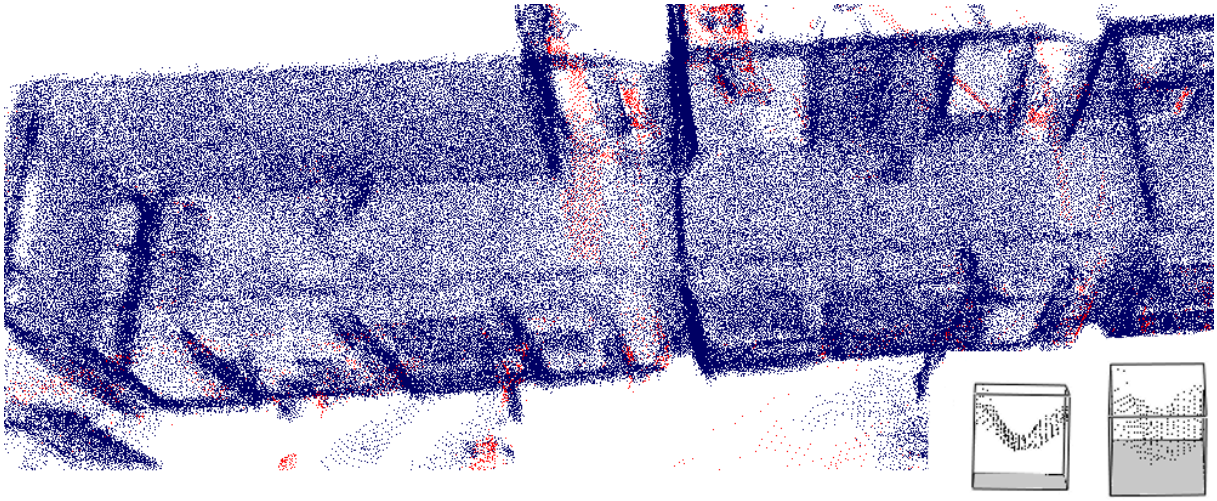


Figure 4.1: This figure shows a point cloud reconstruction obtained with our approach. The reconstruction was built from a dictionary of size 114 (blue). Points of the original scene that are not covered by the dictionary are red. In the bottom right we show two different views of one of the dictionary words chosen by our method.

clouds. We will address this problem with efficient data representations that encode the surface properties of a 3D scene and allow us to quickly retrieve occurrences of pattern in the data.

Our method constructs increasingly accurate models and optimizes them by minimizing the Bayesian information criterion (BIC) to tradeoff the complexity of the model against its accuracy. We evaluate our approach on large real-world data sets. One advantage of our problem formulation is that only few parameters need to be defined. The experiments reveal that our approach can compute accurate models with as few as 78 words, due to the recurrence of the local structures. The number of words is automatically chosen by our algorithm and depends on the structural complexity of the environment. Figure 4.1 shows an example of a 3D scene reconstructed with our approach.

In the next section of this chapter, we will describe our representation, followed by the dictionary selection procedure based on the BIC. We then describe the words in our dictionaries and the features we use to find repetitive structures. After presenting a final reduction step for the scene instances of the words in the model, we present experimental results using different 3D data sets. Furthermore, we describe how our method can be applied to the object detection task. Subsequently, we provide a detailed discussion of related work.

4.1 Learning Compact 3D Models

Point clouds are a common representation for 3D scenes consisting of an unordered set of 3D points $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{|\mathcal{P}|}\}$ sampled at the border between the occupied and the unoccupied space. The number of points required to represent a scene is directly proportional to the area of the visible surface of the objects and to the desired resolution. Furthermore, the resolution has a strong impact on the specific applications that operate on the point cloud. For instance, applications like object recognition require a resolution higher than the size of the largest distinctive detail of the different objects to be detected. In this context, representing large scenes with the appropriate resolution leads to high memory requirements.

In most man-made environments, however, the scenes exhibit locally repetitive patterns. For

instance, flat surfaces, corners, doorways, or multiple instances of the same object in the scene result in similar point distributions at different positions in the cloud. In this chapter, we use these repeating structures to represent the scene in a way that is both compact and accurate. In the remainder of this section, we first give a formal problem description, followed by a discussion of the Bayesian information criterion and a description of our likelihood function to estimate the accuracy of a model. Subsequently, we discuss the heuristics used to reduce the search space and the creation of the dictionary and the model for a given scene. We conclude the section by describing a method to reduce the overlap between nearby words and further improve the result.

4.1.1 Problem Formulation

We construct a *dictionary* of repeating local structures, which constitute the *words* of our dictionary. We then represent the overall scene based on these words, where a single word can occur multiple times at different locations in the scene. Each word w of our dictionary $\mathcal{D}$ is a subset of the points of the original point cloud $\mathcal{P}$, selected from the neighborhood of a given point p with a maximum distance of δ:

$$w = \{s \mid s \in \mathcal{P}, \|s - p\|_2 < \delta\}. \tag{4.1}$$

Given a dictionary $\mathcal{D} = \{w_1, \ldots, w_n\}$, we can reconstruct a 3D scene that approximates the original one by combining the words in the dictionary. We parametrized a surface primitive (SP) model as $\mathcal{M}_{\text{SP}}(\mathcal{D}, \mathcal{I})$, where the instances $\mathcal{I} = \{\mathbf{i}_1, \cdots, \mathbf{i}_{|\mathcal{I}|}\}$ describe a given scene based on a dictionary $\mathcal{D}$. The scene instances $\mathcal{I}$ are a set of tuples $\mathbf{i}_j = \langle \text{id}_j, T_j \rangle$, where id_j is the index of a word in the dictionary and T_j is a 3D transformation. An instance $\mathbf{i}_j$ of the model specifies that the word with index id_j occurs in the scene at location T_j. Based on this notation, we can define the reconstructed scene $\mathcal{R}$ as the union of a set of words in the dictionary, each one positioned according to its corresponding transformation T_j:

$$\mathcal{R} = \bigcup_{j=1}^{|\mathcal{I}|} T_j \cdot w_{\text{id}_j}. \tag{4.2}$$

Here, $T_j \cdot w_{\text{id}_j}$ denotes a set obtained by transforming all points in the word w_{id_j} according to T_j. There is one remaining issue regarding $\mathcal{R}$ that has to be addressed. The positions of the words in the reconstructed scene might have a high overlap, leading to multiple points describing the same part of the original scene. Therefore, we introduce a sparsification step on $\mathcal{R}$ leading to $\hat{\mathcal{R}}$, where every point has a minimum distance ε to every other point in the set:

$$\hat{\mathcal{R}} = \{r_i \mid r_i \in \mathcal{R} \wedge \|r_i - r_j\|_2 \geq \varepsilon \; \forall r_j \in \hat{\mathcal{R}}\} \tag{4.3}$$

In the following, we will describe how to learn a dictionary and present an interest point method to find stable positions for the words on he surface.

4.1.2 Dictionary Leaning with the BIC

Given a perfect dictionary, the reconstruction and the point cloud are identical $\hat{\mathcal{R}} = \mathcal{P}$. The disadvantage of a perfect dictionary is that it is typically big since it has to describe not only every surface configuration but also noise in the sensor measurements and all possible point distributions on similar surfaces. To avoid overfitting to noise and point distribution, we focus

on building an approximate reconstruction and learn a memory efficient dictionary. To achieve this, we have to tradeoff the accuracy of the approximation against the complexity of the learned model. This tradeoff is naturally captured by the BIC, which is a statistical model selection criterion. The BIC score is computed based on the sum of the negative log-likelihood of the model, given the data, and the model complexity with respect to the number of data points and the number of free parameters. Given the input scene $\mathcal{P} = \{s_1, ..., s_N\}$ of size N and the reconstructed scene $\hat{\mathcal{R}} = \{r_0, \cdots, r_M\}$ of size M created from the dictionary $\mathcal{D}$, the BIC is defined as:

$$\mathrm{BIC}(\mathcal{P}, \hat{\mathcal{R}}) = -2 \cdot \log(\mathcal{L}(\mathcal{P}, \hat{\mathcal{R}})) + k \cdot \log(N). \tag{4.4}$$

Here $\mathcal{L}(\mathcal{P}, \hat{\mathcal{R}}))$ is the likelihood function that measures how well the data $\mathcal{P}$ is represented by the reconstructed point cloud $\hat{\mathcal{R}}$, and k is the number of parameters (see below). Minimizing the BIC therefore results in finding a model, which is maximally consistent with the input (small negative log-likelihood), while having a limited complexity (small k). In our problem, the count of parameters is defined as follows:

$$k = 3 \cdot \sum_{i=1}^{|\mathcal{D}|} |w_i|, \quad w_i \in \mathcal{D}, \tag{4.5}$$

which is the sum of the point cloud sizes of every word in the dictionary times 3, since a point has dimension 3. By assuming the points in the cloud to be independent, we define the log-likelihood of the reconstruction $\hat{\mathcal{R}}$ as follows:

$$\log(\mathcal{L}(\mathcal{P}, \hat{\mathcal{R}})) = \log\left(\prod_{i=1}^{N} P(s_i \mid \hat{\mathcal{R}})\right) \tag{4.6}$$

$$= \sum_{i=1}^{N} \log(P(s_i \mid \hat{\mathcal{R}})) \tag{4.7}$$

Here s_i represents the i^{th} point in $\mathcal{P}$ and the probability of s_i is modeled depending on the data associations a_{s_i,r_j} between s_i and the j^{th} point in $\hat{\mathcal{R}}$:

$$P(s_i \mid \hat{\mathcal{R}}) = \sum_{j=1}^{M} P(s_i \mid r_j, a_{s_i,r_j}) \cdot P(a_{s_i,r_j}) \tag{4.8}$$

The need to model the data associations arises from $\mathcal{P}$ and $\hat{\mathcal{R}}$ having different resolutions in corresponding areas. Since we lack any knowledge about the data associations, we assume a uniform distribution of the prior $P(a_{s_i,r_j})$, leading to

$$P(s_i \mid \hat{\mathcal{R}}) = \frac{1}{M} \sum_{j=1}^{M} P(s_i \mid r_j, a_{s_i,r_j}). \tag{4.9}$$

The main influence of the data association is typically in the close proximity around the target data point s_i. Therefore we only consider data associations within a distance of 3σ (see below for a definition of σ). Let $\hat{\mathcal{R}}^{s_i} = \{r'_1, \cdots, r'_{M'}\} \subseteq \hat{\mathcal{R}}$ be the points in $\hat{\mathcal{R}}$ that are in this range of s_i:

$$\hat{\mathcal{R}}^{s_i} = \left\{ r_j \in \hat{\mathcal{R}} \mid \|s_i - r_j\|_2 \leq 3\sigma \right\}, \tag{4.10}$$

Using this set, we can approximate Equation 4.9 as

$$P(s_i \mid \hat{\mathcal{R}}) \approx \frac{1}{M} \sum_{j=1}^{M'} P(s_i \mid r'_j, a_{s_i,r'_j}). \tag{4.11}$$

Given an association a_{s_i,r_j}, the probability for a data point pair is modeled as a spherical Gaussian

$$P(s_i \mid r_j, a_{s_i,r_j}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}\|s_i - r_j\|_2^2\right). \tag{4.12}$$

Since we typically know the characteristics of the range sensor used to acquire a point cloud, we can model the deviation σ range dependent for each point s_i proportional to the resolution of the original range scan at the range of s_i. For a laser range finder with an angular resolution of α, we compute σ_{s_i} in the following way:

$$\sigma_{(s_i)} = \lambda \cdot \tan\left(\frac{\alpha}{2}\right) \cdot \text{range}(s_i) \tag{4.13}$$

A uniform σ would overestimate the importance of the low resolution parts in the input data. Consider a typical 3D range scan taken in a planar world, in which every patch can be described by a planar point cloud. In this context, the only difference between the planar point clouds is their resolution. By using our formulation of the likelihood function, the best BIC rated word is the one having the "average" resolution. Reconstructing the scene with this best candidate reduces the accuracy in high-resolution areas, while it improves low-resolution parts. So far we described how to evaluate the BIC of a given model $\mathcal{M}_{\text{SP}}(\mathcal{D},\mathcal{I})$ for a dictionary $\mathcal{D}$ and a scene $\mathcal{P}$. In the following sections, we will describe how we construct a dictionary and a model given a new scene.

4.1.3 Interest Points

Let us recall, that a model $\mathcal{M}_{\text{SP}}(\mathcal{D},\mathcal{I})$ consists of a dictionary $\mathcal{D}$ and a scene description $\mathcal{I}$, which is a set of tuples $\{\mathbf{i}_1, \cdots, \mathbf{i}_{|\mathcal{I}|}\}$. Every $\mathbf{i}_j = \langle \text{id}_j, T_j \rangle$ describes an *instance* of the word $w_{\text{id}_j} \in \mathcal{D}$ that is supposed to be placed in the reconstructed scene at the position described by the transformation T_j.

Given a new scene, actually finding the optimal dictionary and model with respect to the BIC is a hard problem. For the dictionary, an arbitrary number of words with no restriction regarding shape and size of the words is possible. For each of those dictionaries, an arbitrary number of instances with all possible combinations of positions in the scene would have to be evaluated. To reduce the search space, we use mainly two heuristics: we use words of a fixed size and shape (spheres with a fixed radius, which we call the *support size*), and we use fixed poses for the instances, which are computed on the scene in advance. The latter is done according to the following criteria:

- Given a recurrent surface, we would like to have stable interest points. Whenever we observe a specific surface, the interest point should be at the same location on the surface.
- We have to ensure a full coverage of the input point cloud $\mathcal{P}$ with respect to the word radius δ.

The first aspect is at the heart of typical key point algorithms. Therefore, we use a key point detector for 3D point clouds and fill the remaining surface area via uniform sampling. In this way, we get stable interest points in areas with well-defined changes in the surface, like edges or corners and uniformly distributed interest points in areas with only few changes in the surface. On a flat surface, the interest point position is not stable. For our application this is not problematic since the surface looks similar in a wide range and we can easily describe the surface with the same patch, in the local neighborhood on the surface.

To compute the interest points, we use the interest point extractor of the Normal Aligned Radial Feature (NARF) (Steder et al., 2010b). The NARF interest point detector selects points at locations with a stable surface, for which we can robustly estimate normals, and sufficient changes in the surface in the surrounding to ensure a stable position, even in the presence of sensor noise. To apply the NARF interest point extractor, we have to compute range images from the point $\mathcal{P}$. Since a range image cannot represent the data of a point cloud fused from different viewpoints, we have to compute a range image for every original viewpoint and fuse the interest points of the range images again. To select interest point candidates, a score is computed on every point in a range image, which is computed from the point cloud. This score reflects the surface stability and the amount of variation, as can be seen in Figure 4.2(a,b). We use non-maximum suppression to select the final interest points. We made some modifications in the scoring function to adapt it to our needs. To get interest points on straight edges, we added a term to the scoring function that reflects high curvature. The result can be seen in Figure 4.2(c) and the result of the standard interest point extractor is shown in (b). We select the final interest points $\mathcal{K}$ from the scored pixels of a range image as described in Algorithm 1.

Algorithm 1 Surface primitive interest point selection.

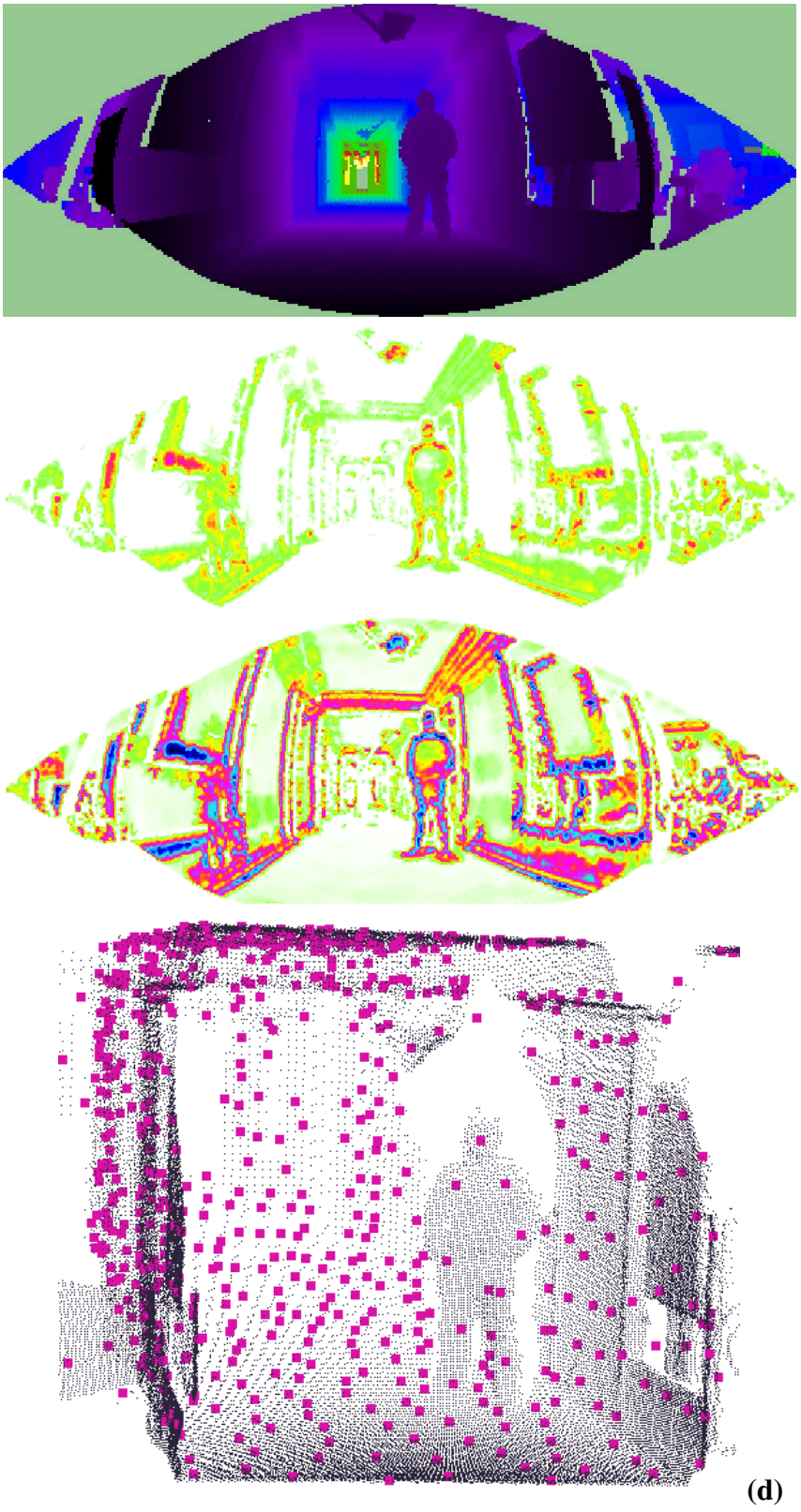
while $s_{max} > 0$ **do**
  for all points $p \in \mathcal{P}$ **do**
    find maximum score s_{max} and corresponding point p_{max}
  if $s_{max} == 0$ **then**
    break
  add p_{max} to $\mathcal{K}$
  set scores of neighborhood $\mathcal{N} = \{p | p \in \mathcal{P}, \|p - p_{max}\|_2 < \hat{\delta}\}$ to 0
end while

With this greedy selection, we first add points on well-defined complex structures and later points on straight edges. The positions on the straight edges are not fully defined since they can be shifted along the edge. By selecting these points after the well-defined points and forcing a distance between neighboring interest points, we align them according to the scene structure. In the last step, we add the points on uniform surfaces as can be seen in Figure 4.2(d). The radius $\hat{\delta}$ for changing the neighborhood scores to 0 is set between δ and 2δ to guarantee a sufficient overlap.

Once we have the final interest points $\mathcal{K}$, we compute a NARF descriptor on all interest points. Each descriptor is a vector of real numbers and can be used for fast similarity comparisons based on a metric, such as the Manhattan distance. Figure 4.3 shows two examples obtained with the descriptor-based similarity measure. Furthermore, the NARFs provide a unique 6 degree-of-freedom (DOF) local coordinate frame, computed from the surface structure. We will use the 6 DOF transformations to describe the locations of the words instances in a scene.

In the next step, we construct an initial model with a temporary dictionary. To achieve that, we create a word w_j for every interest point $k_j \in \mathcal{K}$ from of the input point cloud $\mathcal{P}$ by selecting the neighborhood around k_j in $\mathcal{P}$ and applying the inverse of the corresponding transformation T_j. This gives us a small point cloud, which is centered in the coordinate origin and contains all points that fall into a cube in w_j. Out of this procedure, we also get the corresponding instance description $\mathbf{i}_j = \langle \mathrm{id}_j, T_j \rangle$, which contains the position T_j and dictionary index of the extracted word. The result of this step is a model with $|\mathcal{I}| = |\mathcal{D}_\mathcal{P}| = |\mathcal{K}|$ and our goal, however, is to learn a small dictionary $\mathcal{D}$ with $|\mathcal{D}| < |\mathcal{D}_\mathcal{P}|$.

We learn $\mathcal{D}$ in an iterative fashion, starting with an initial dictionary of size 1. The first entry



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

Figure 4.2: This figure illustrates the interest point selection. **(a)** shows an input range image of a corridor with a person standing in this corridor. **(b)** depicts the scores for the standard NARF interest point detector. The scores shown in **(c)** correspond to the described modified scoring function for the interest point extraction. The corresponding point cloud with the final interest points is illustrated in **(d)**.

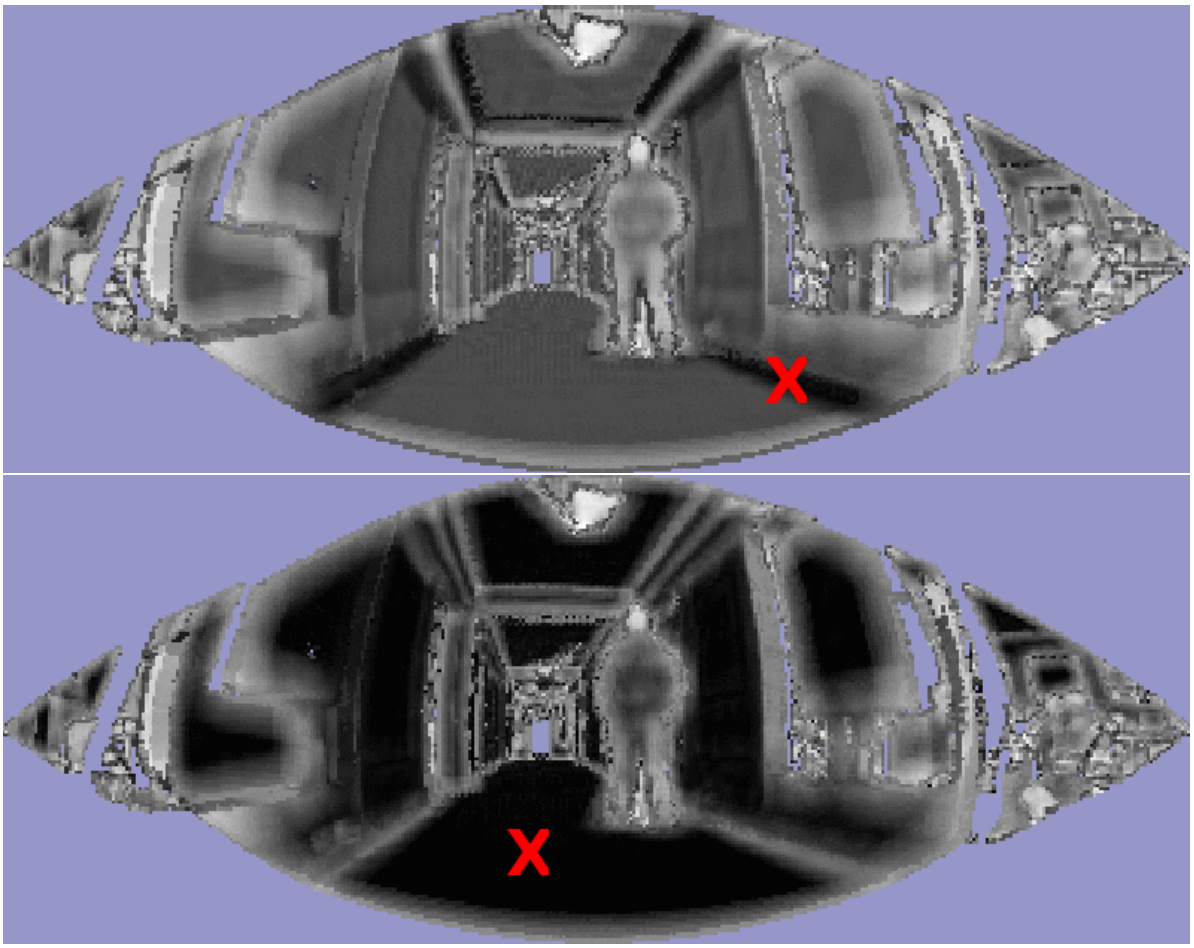


Figure 4.3: This figure illustrates the use of features for similarity queries. **(a)** shows a similarity query for an edge feature. The cross corresponds to the selected feature and the grey value in the image reports the similarity. A dark pixel is highly similar, whereas a bright pixel corresponds to a low similarity. A similarity query for a flat surface is shown in **(b)**.

is chosen randomly among all candidate words in $\mathcal{D}_\mathcal{P}$. In each iteration, we check if we can improve the BIC by inserting, deleting or replacing a word in the current dictionary. For the insert operation, we randomly select a previously not selected word. Deleting a word from the dictionary is only allowed for $|\mathcal{D}| > 1$. We replace a word by randomly selecting one word in the dictionary and one word that is not in the current dictionary and by swapping them.

Whenever we change the dictionary and compute the BIC, we first have to assign all instances $\mathbf{i}_j = \langle \mathrm{id}_j, T_j \rangle$ in the scene to the corresponding words in the dictionary. We could compute a similarity score based on the point clouds of the original word extracted at a location and every entry of our dictionary. Since we already have NARF descriptors for all instances, we do the assignments based on the minimum descriptor distance, resulting in a faster matching.

After changing the dictionary, we compute the BIC score and compare it with our previous score. If the BIC score is not improved by the last action, we reject the last operation and restore the previous dictionary. The dictionary learning terminates either if a maximum number of iterations is hit or the dictionary did not change for a certain number of iterations.

4.1.4 Reducing the Instances in the Model

Since words can have an arbitrary overlap, we can reduce the number of instances of words in the model, while preserving the likelihood value. This procedure does not affect the dictionary $\mathcal{D}$, but the number of instances used to reconstruct the scene $|\mathcal{M}|$. To this end, we attempt to sequentially remove the redundant instances of words in the model. We analyze one word w at a time, and we consider all occurrences of that word in the model. The order, in which we process the words is given by the number of occurrences of each word in the model. For each occurrence of w in the model, we measure the change in the likelihood when removing it from the scene. If this value does not decrease, the occurrence is redundant and we remove it from the model.

4.2 Experiments

In this section, we present experiments to evaluate the performance of our approach. We apply our approach to data obtained in different environments and we measure the following quantities:

- The number of words in the dictionary $|\mathcal{D}|$.
- The number of instances $|\mathcal{I}|$ in the model.
- The relative file size, i.e., the ratio between the memory consumption of the model generated by our approach and the memory consumption of the original data. We use a standard binary representation of the data, meaning every floating point value takes 4 bytes of memory. Each 3D point thereby takes 12 bytes, each 6 DOF transformation takes 24 bytes, and a reference to a word in the dictionary takes 4 bytes.
- The number of inliers, i.e., the data points that are correctly matched by the model.
- The number of outliers, which are the points in the reconstructed model that are not contained in the original scene.

The number of inliers and outliers is calculated based on nearest neighbor queries using a *k*d-tree. A point is considered to have a correct correspondence if a neighbor is in the distance of $\sigma_{(s_i)}$. In our experiments the fraction of outliers was never approaching zero even in situations where the reconstructed scene exhibited a high accuracy. The reason for this lies in the predictive behavior of the words that can be partially matched with the scene and complete parts of the environment that are not fully observable.

The first experiment deals with the reconstruction of an indoor scene, described in the Scene Compression section (4.2.1). Next, we will investigate, how a dictionary learned on one scene can be applied to reconstruct a second scene in the Dictionary Transfer section (4.2.2). Furthermore, we will evaluate the parameters of the likelihood function in Section 4.2.3. Afterwards, we will present results obtained with large in- and outdoor datasets in the Compression of Large Datasets section (4.2.3). Finally, we present a naive object detection as example application in Section 4.2.5.

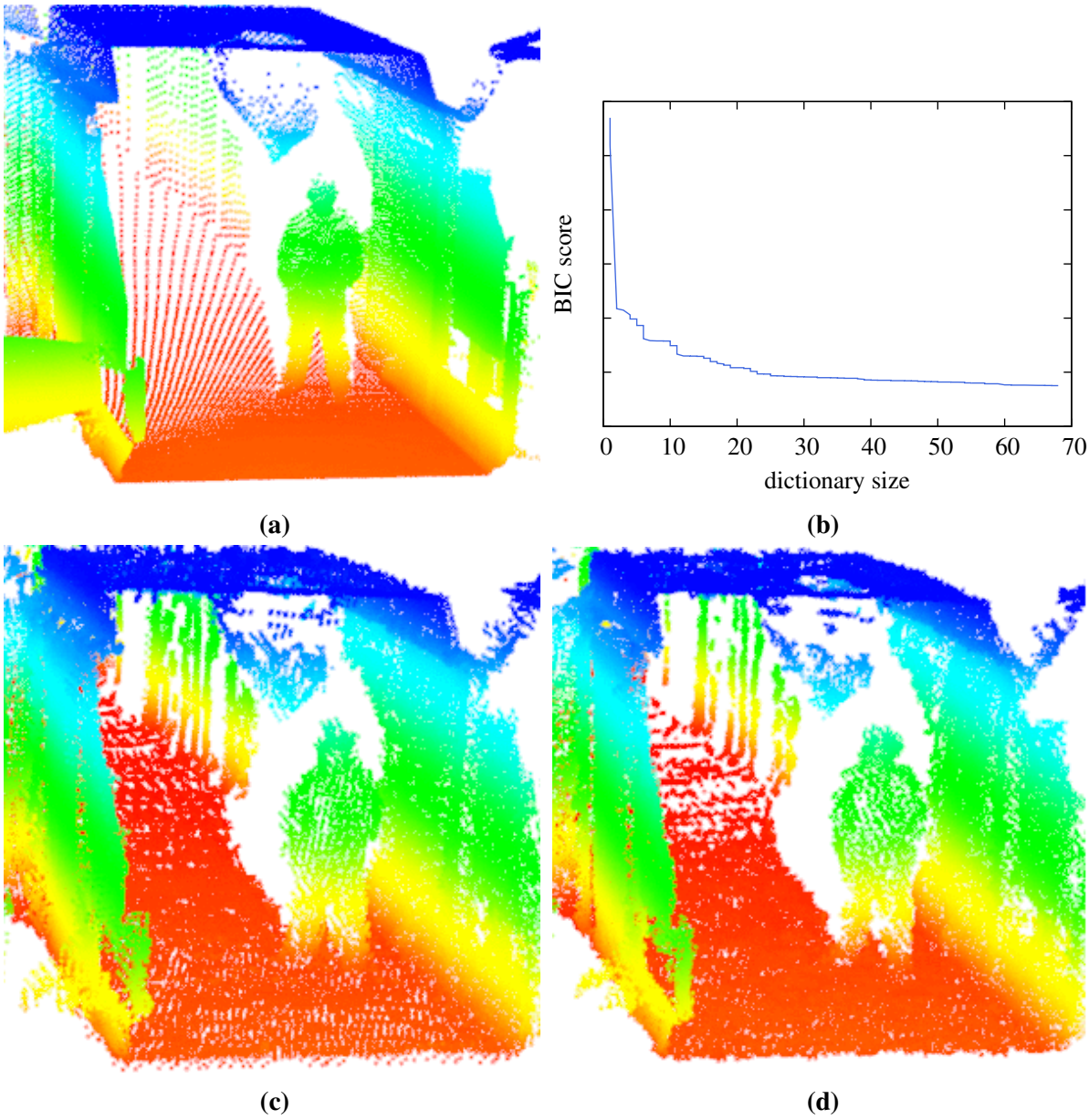


Figure 4.4: This figure shows **(a)** the original point cloud, **(b)** the evolution of the BIC during the learning process, **(c)** point cloud reconstructed based on a dictionary of size 10 and **(d)** based on the final dictionary with size 68. The planar corridor is already well explained with a dictionary of size 10. The accuracy of the reconstruction gets better with an increasing number of words.

4.2.1 Scene Compression

In this experiment, we measure the accuracy of a model obtained in an indoor scene. The input data correspond to a scene, in which a person is standing in a corridor and in front of the robot (see Figure 4.4(a)). After 1000 iterations, our method converged to a model containing 68 words. Figure 4.4(b) shows the evolution of the BIC score. For illustration purposes, Figures 4.4(c) and 4.4(d) show two different models reconstructed with 10 and 70 words. We chose a value of $\delta = 15\,\text{cm}$ in this experiment, meaning our dictionary words covered spheres with a diameter of 30 cm. The resulting size of the model is 200 Kb, which, given an initial point cloud of size 1.2 Mb, corresponds to a relative file size of 16%, compared to the input data.

The reconstructed point cloud of the final dictionary with size 68 covers 89% of the input point cloud and yields 4.2% outliers according to the point cloud evaluation method described above. See Table 4.1 for more details. The reconstruction shows a more uniform point resolution than the original point cloud in Figure 4.4(a). This is caused by the fact that a word encodes a surface prototype and every instance of a word in the scene uses this prototype for reconstruction. In this example, our encoding scheme introduces higher resolutions at more distant surfaces.

4.2.2 Dictionary Transfer

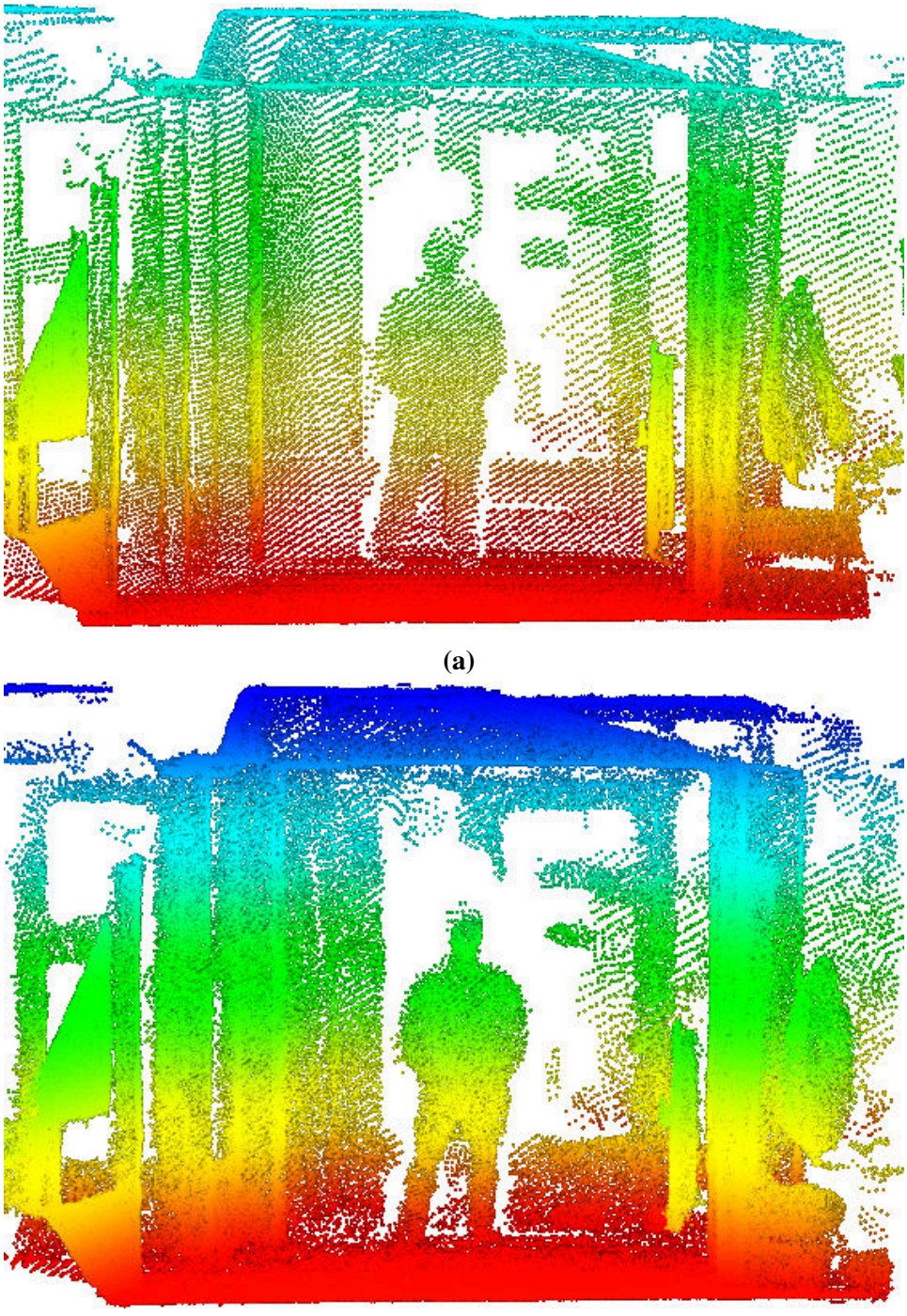
In a second experiment, we evaluated, how well a dictionary learned on a given scene can be applied to describe different scenes. To this end, we used the dictionary acquired during the experiment described in Section 4.2.1 to reconstruct a new scene acquired in the same building. To reconstruct the scene, we generated a model $\mathcal{M}_{\mathrm{SP}}$ by using the known dictionary and the scene instances computed with the interest point selection method explained in Section 4.1.3. Based on this model, we then reconstructed the input scene and compared it to the original input.

Figure 4.5 illustrates the original point cloud in (a) and the reconstruction in (b). The picture in (a) shows the same person as in the previous scene but in another location. The resulting model has 5263 instances and the reconstruction has 89% inliers and 12% outliers. Compared to 89% inliers and 4.2% outliers of the originally reconstructed scene, the result for the dictionary transfer is slightly worse. In general, this means that learned dictionaries generalize to similar environments. Of course, novel surfaces with no similar dictionary entry, cannot be represented accurately.

4.2.3 Parameters of the Likelihood Function

The proposed method has only one adjustable parameter, the standard deviation σ used in the computation of the likelihood model. This parameter influences the stiffness of the likelihood function and therefore the accuracy of the model approximation. In this experimental setting, we evaluated the impact of σ on the small point cloud that represents a chair shown in Figure 4.6 (a). We computed a model for every σ in the interval $[0.0001, 0.02]$ with a step size of 0.0001. Figure 4.6 (b) shows the dictionary sizes of the results for every step. It can be seen that the number of words drops from 80 to under 10. This result highlights the ability of our method to reduce the model complexity depending on the desired accuracy.

Figure 4.6 shows examples of model approximations built with dictionary sizes of 34 (c) and 12 (d) words. The model shown in (c) has a similar structure compared to the original data in (a). In contrast to that, the model in (d) has a flat back part and most of the fine structure at the chair feet is not well represented. Figure 4.7 depicts the detailed evolution of the BIC score and the dictionary size for the model from Figure 4.6(c). The BIC score drops to a nearly constant level after 200 iterations. In contrast, the dictionary size needs more than 400 iterations to converge. The intuition behind this is that the words chosen in the first 200 iterations have a large impact on the likelihood, whereas the words inserted in the later iterations give only locally better approximations. Since we are interested in the quality of the approximation, we use the changes in the dictionary as convergence criteria instead of changes in the BIC. In our current implementation, we run the BIC dictionary selection until the dictionary does not change for 1000 iterations.



(a)

(b)

Figure 4.5: The point cloud of a scene with a person standing in a corridor is illustrated in **(a)**. The reconstructed point cloud, using only words from the dictionary extracted from the scene in Figure 4.4(a) is shown in **(b)**. As can be seen, our approach yields accurate reconstructions even in such situations.

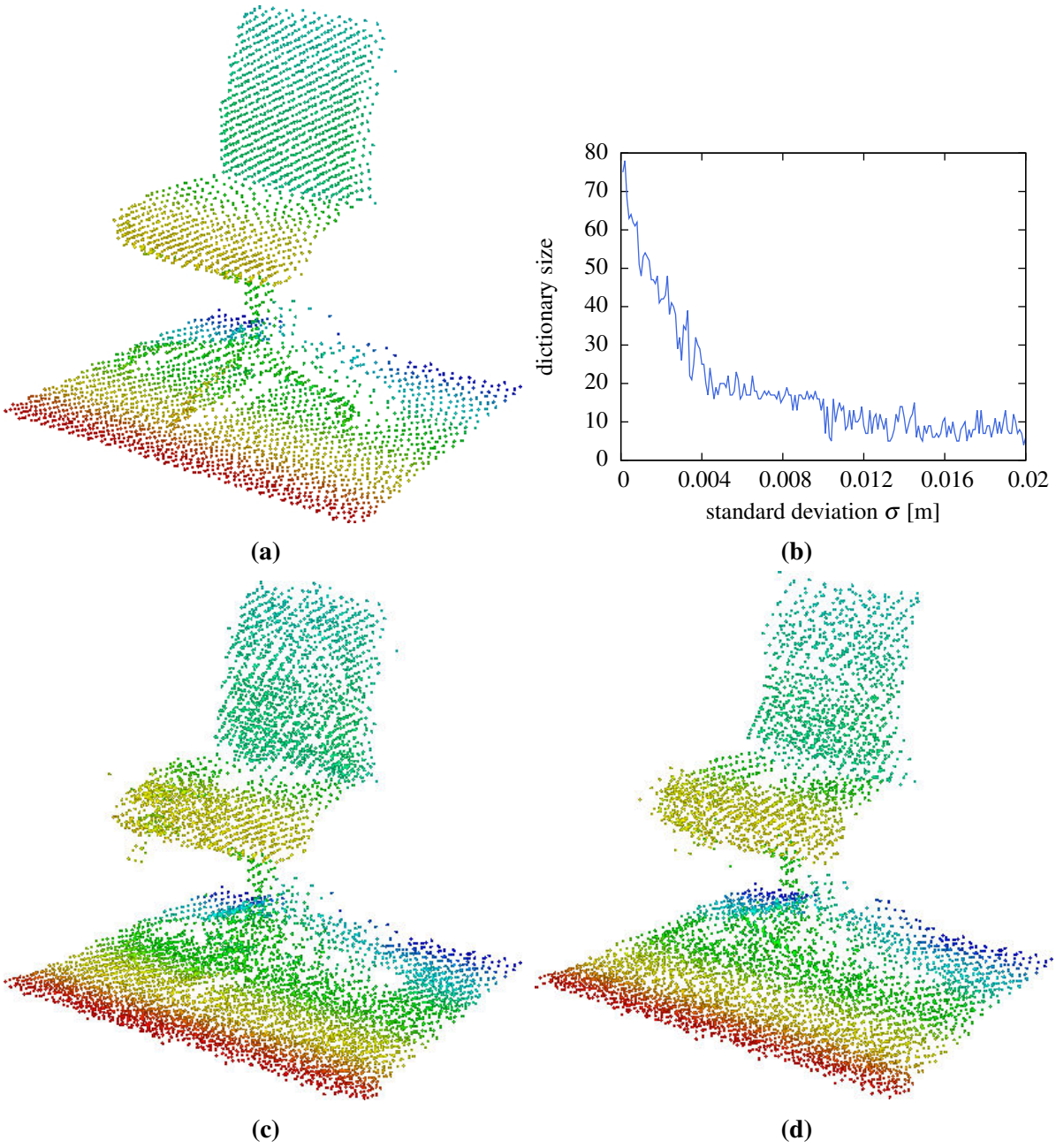


Figure 4.6: The input point cloud of a standard office chair is illustrated in **(a)**. The dictionary size for different values of σ is plotted in **(b)**. The model approximations of the point cloud shown in (a) are depicted in **(c)** and **(d)** using a dictionary sizes of 34 and 12. The model approximated with 34 words has nearly the same structural appearance as the point cloud (a). In contrast, the model approximated with 12 words has a flat back part and less structure at the feet of the chair.

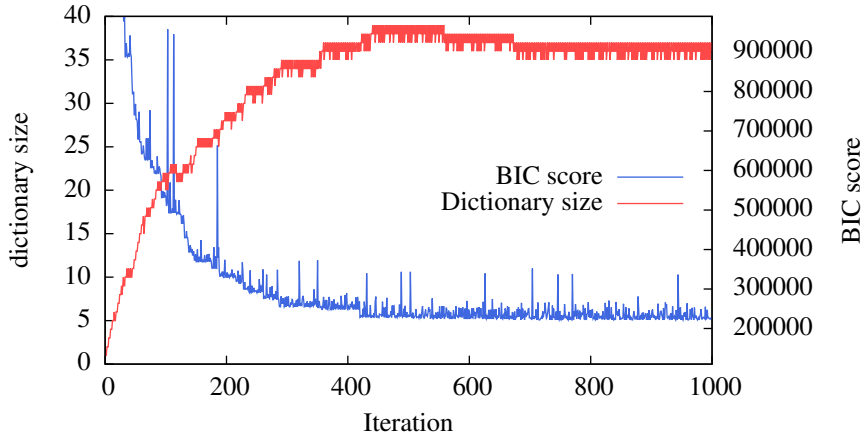


Figure 4.7: This plot shows the evolution of the BIC score and the dictionary size during computation of the model shown in Figure 4.6(c). While the BIC stays nearly constant after only 200 iterations, the dictionary size needs almost 400 iterations to converge to a nearly constant level

4.2.4 Compression of Large Datasets

The advantages of our representation are particularly evident on large datasets, where the size of the dictionary is bounded by the repetitiveness of the environment. To demonstrate this, we recorded an indoor dataset in the main corridor of building 79 on our campus. The dataset consists of 31 three-dimensional range scans. Figure 4.8 shows the reconstructed point cloud and a plot of the occurrence frequencies for all words. Whereas the points that originate from the most frequent word are colored in yellow, the points of the remaining words are colored in blue. The yellow points cover most of the scene but the occurrence count for the most frequent word is not proportional to the observed coverage. This is caused by the fact that words can describe the surface with a different amount of points and that words can overlap. Redundant information of overlapping words is not included in the point cloud reconstruction.

As second dataset, we chose a portion of a publicly available 360° outdoor dataset (Steder, 2009), which was acquired on our campus. For the outdoor setting, we chose a word radius δ of 0.25 m. The resulting statistics are shown in Table 4.9. Note that the relative file size is computed using the sparsified original point cloud as reference. Additionally, we give the compression rate for the case that both, the original point cloud and our model were compressed using the standard file compression method gzip afterwards. The relative file size is about 18% in both cases. Gzip leads only to a minor compression and can similarly be used to compress our model. Our experiments suggest, that typical real-world scenes can accurately be described by a small number of reoccurring words. While the run time for selecting a dictionary is not yet feasible for online applications, a previously learned dictionary might be sufficient for many applications.

4.2.5 Example Application: Object Detection

In this section, we briefly discuss object detection as one example application of the presented model representation. We follow the standard terminology and speak of an object model and of a scene, in which we search for object models. Standard object detection approaches represent object models and scenes with features like NARF (Steder et al., 2010b), VFH (Rusu et al., 2010) or Spin Images (Johnson and Hebert, 1999). The approaches search for corresponding features between object models and a scene. Furthermore, the spatial relations of the features

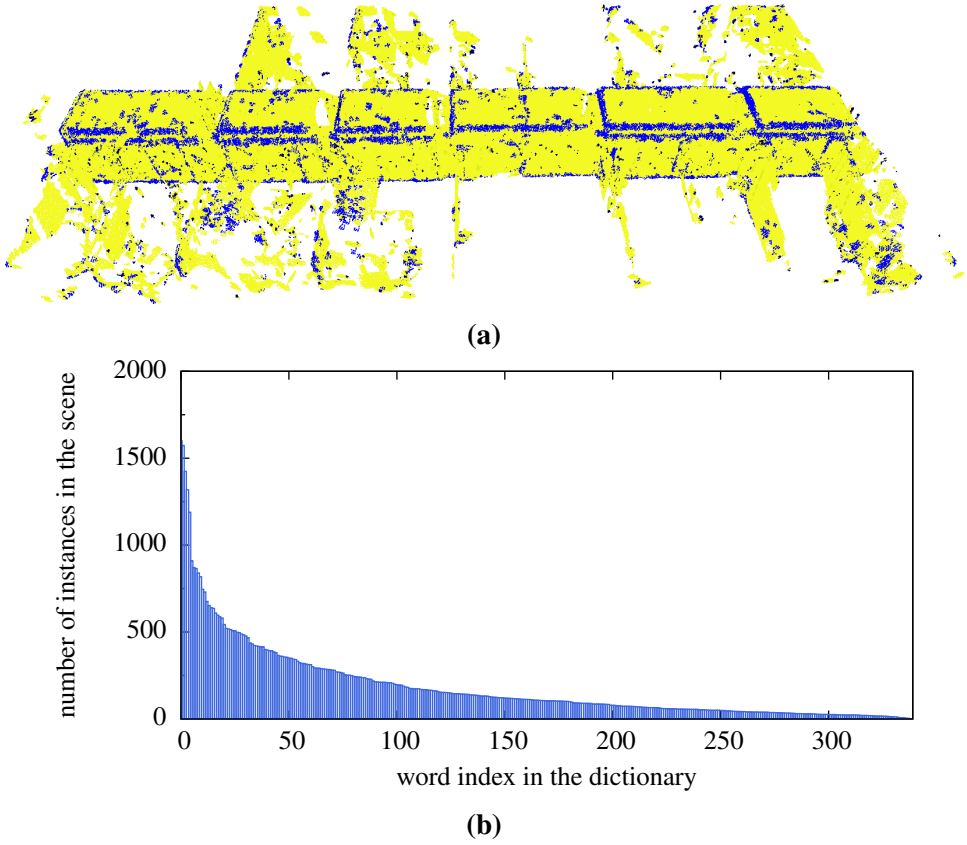
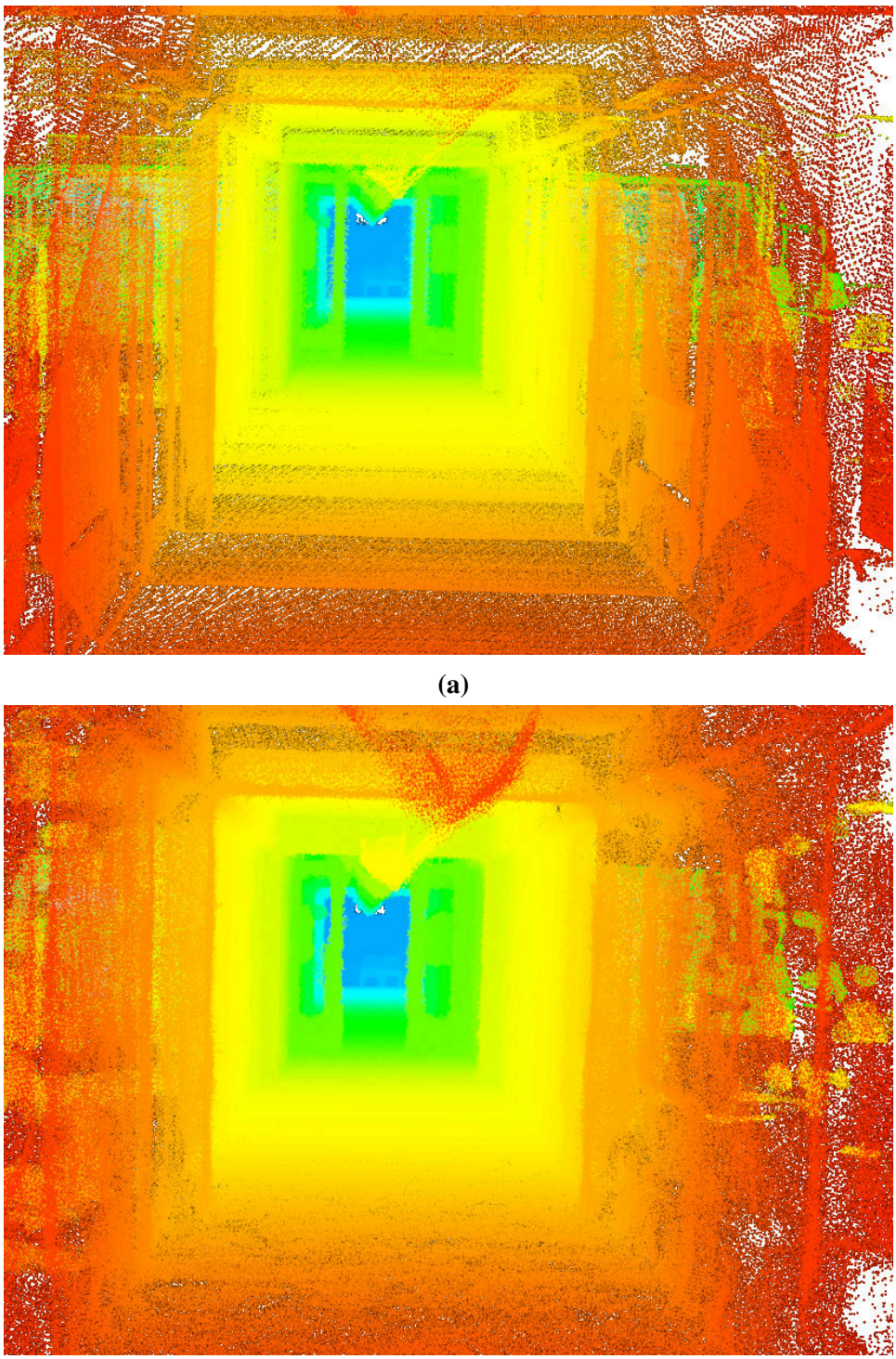


Figure 4.8: A map generated from the corridor dataset is illustrated in **(a)**. Whereas the dominant planar word in the dictionary is colored in yellow, the remaining words are colored in blue. The instance occurrences per word are shown in **(b)**. The occurrence frequency does not directly reflect the percentage of the covered area since the count of points for a word can be different and words may overlap.

Dataset	Single scene	Corridor	360° outdoor
figure	4.4	4.8	4.10
dictionary size $\|\mathcal{D}\|$	68	339	159
word radius	0.1 m	0.15 m	0.25 m
word occurrences	6,599	62,792	873,464
file size input point cloud	1.21 MB	37.77 MB	137.47 MB
compression with gzip	1.09 MB	33.99 MB	122.11 MB
file size of our model	0.20 MB	1.85 MB	24.47 MB
model with gzip compression	0.16 MB	1.67 MB	21.25 MB
relative file size of model	16.5%	4.9%	17.8%
relative file size using gzip	14.7%	4.9%	17.4%
inliers	88.3%	98.7%	99.6%
outliers	4.2%	0.3%	1.2%
runtime	11 min	20 min	22.3 h

Table 4.1: Overview of the experimental results. The timings were measured with an Intel i7 quad core CPU.



(a)

(b)

Figure 4.9: This figure gives a closeup view of the original point cloud in **(a)** and a similar view of the computed model in **(b)**. The point distribution in the original point cloud is range dependent. In contrast the model approximation gives a denser reconstruction of the scene and has only 5% of the storage requirements compared to the original data.

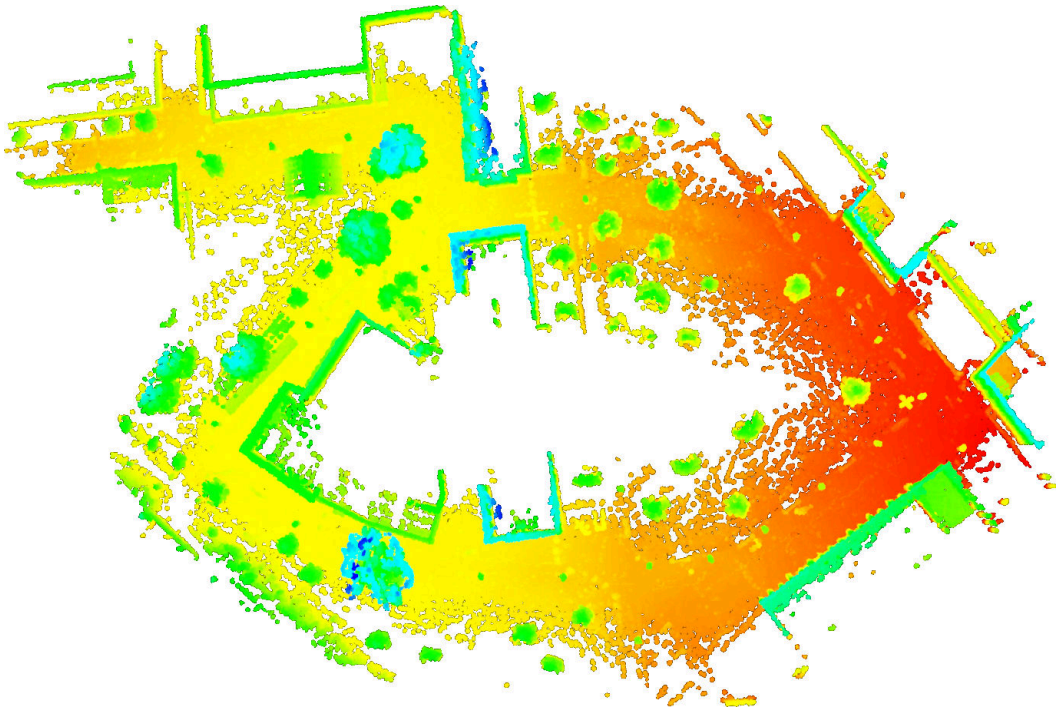


Figure 4.10: Point cloud reconstructed from the 3D outdoor dataset. The colors denote the height of a point. The model covers the observed environment by 99.6% and has 1.2% outliers.

can be used to perform a geometric validation and reject false positives.

Our model representation allows us to directly use the words in the dictionary as features. A word describes one type of repetitive surface structure. Given an expressive dictionary, we can accurately describe objects in a model database and in a scene. In principle, this description should be similar in our object model and in every scene we describe with the same dictionary. One straightforward way to represent an object given a dictionary is to calculate a histogram of word appearances. Representing an object in that way is strongly related to Bag-of-Words approaches such as FABMAP (Cummins and Newman, 2009). In such a histogram $H = \{h_1, ..., h_n\}$, the entry h_i tells how often the word w_i occurred in an object model, divided by the total number of word appearances. We can directly compute this histogram on a model by counting the words that are linked with the word instances describing an object. Since we are interested in the local word distribution in the scene, we compute the histograms only based on the surrounding words. More concretely, we compute one histogram with radius r for every word instance in a scene by counting the surrounding words. This can be done efficiently with *k*d-trees. We choose the radius of the spheres according to the size of the object model. Given the object model histogram and the local histograms for all word instances in a scene we can compare every local histogram with the object model histogram. For this comparison, we choose the Euclidean metric. Every histogram pair that has a difference below a given threshold can be considered as match.

To test the described method, we manually extracted 8 partial views of the office chair shown in the top picture of Figure 4.11 from different scenes and computed a common dictionary. Then we calculated histograms for every partial view as object model reference. Finally, we tested the method on the point cloud shown in the bottom picture of Figure 4.11. This scene shows a cluttered office environment that includes five instances of the office chair. We applied the object model dictionary on the model of the scene in the same fashion as described in Section 4.2.2.

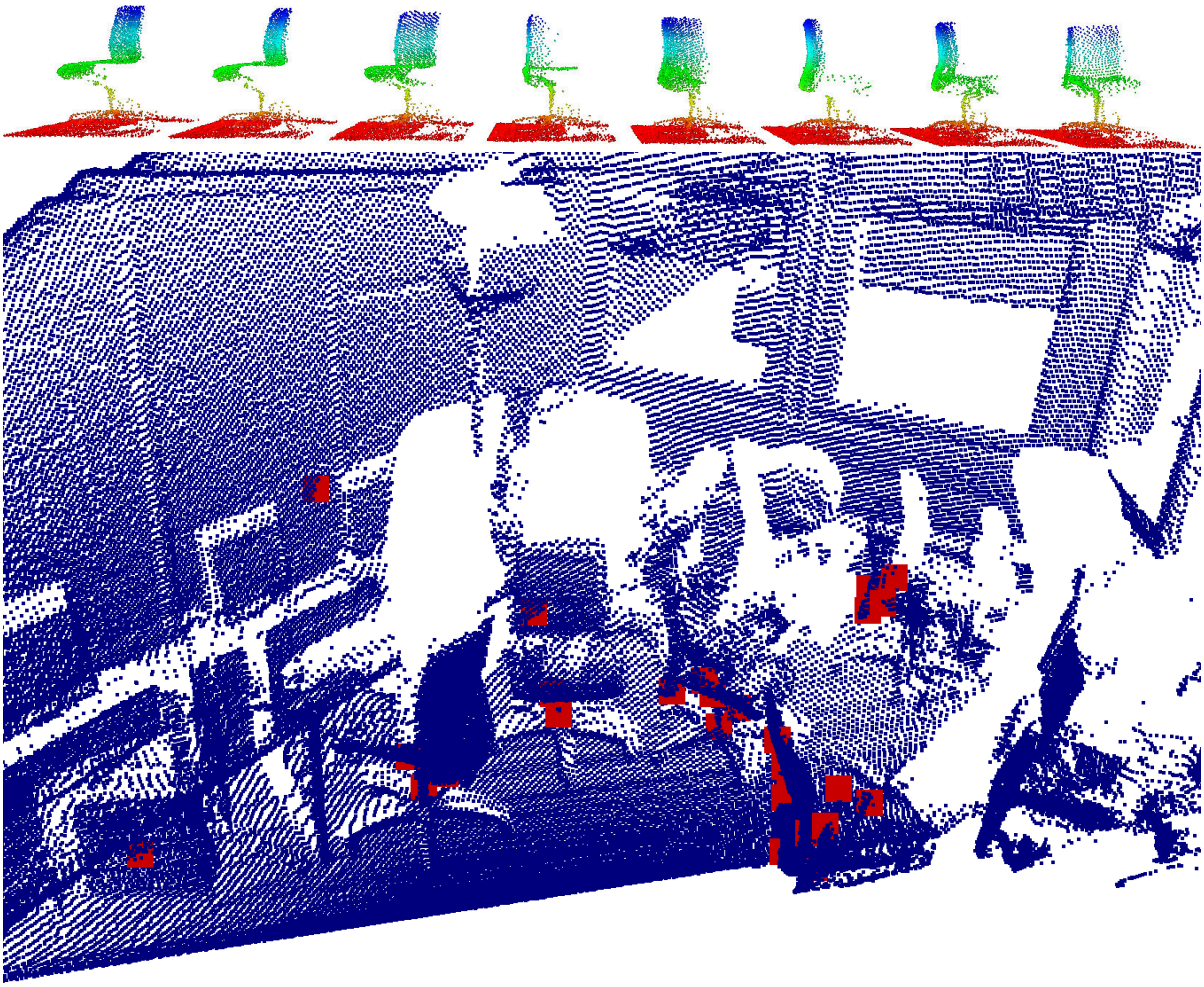


Figure 4.11: This figure shows the point clouds of 8 partial views used to construct the dictionary for the object detection example. The bottom picture shows the point cloud of a scene in blue and the results of our naive object detector highlighted with red cubes.

Afterwards, we computed local histograms for all word instances and compared those with the reference histograms. The best matches are visualized with red squares in Figure 4.11. Our naive detector was able to find all five visible object instances in the chosen scene in less than 100 msec. The result includes only two false positives in the left part of the image. This naive approach could easily be extended by considering spatial relations between words to make it more robust, similar to what we will describe in Chapter 7.

4.3 Related Work

Accurate representations of 3D scenes or models are required for a wide range of applications. Recent developments on Simultaneous Localization and Mapping (SLAM) made it possible to obtain large metric models, however, these models can be larger than the available memory. Most 3D data representations aim at being used within these SLAM algorithms. In these contexts, it is usually more important to focus on a reliable subset of the features in the environment than to attempt to represent the scene in its whole complexity.

A straightforward way to describe 3D data is to directly represent the unordered set of

3D points in the scene. This representation is commonly referred to as point cloud. Popular algorithms in SLAM (Nüchter et al., 2005) and computer vision (Besl and McKay, 1992) rely on these kinds of representations. The advantages of point clouds are that they capture the world at an arbitrary level of resolution and that one can easily apply spatial transformations to them. Their disadvantage is that they lack structural information and that they can be arbitrarily redundant, resulting in large memory overheads.

In computer graphics octrees are a common approach to represent scarcely occupied volumetric regions. In the context of 3D laser-based SLAM, Fairfield et al. (2007) proposed to use oct-trees to represent the range measurements. The advantage of octrees is that they compactly represent the 3D space. Their main shortcoming is the complexity of applying 3D transformations, which basically requires to rebuild the entire octree. Additionally, the octrees describe the pure occupancy information, disregarding the fact that the 3D data originates from a surface. To recover properties of the surface from an octree representation, one needs to use a high-resolution tree that may result in enormous memory requirements.

Surface maps are another popular representation for 3D data. They can be thought of as 2D grids, in which every cell encodes the height of an object. While these structures are clearly attractive to represent information related to the traversability of a given environment, they are inadequate to represent surfaces whose normals are not aligned with the z axis of the world. Additionally, it is not possible to represent full 3D environments with multiple levels using normal surface maps. To overcome this limitation, Triebel et al. (2006) proposed the multilevel surface maps (MLSM). A MLSM can be thought of as a stack of surface maps, each one describing one of the folds of the environment, plus a series of "vertical" entries that capture the information poorly represented by various surfaces. MLSM are effective to represent man-made environments. The bounded resolution of the cells in the surface grid may be a problem for applications requiring a high accuracy like object detection.

An approach for 3D mapping based on laser range data that relies on extracting planar patches from the observations has been proposed by de la Puente et al. (2009). Extracting planes is rather efficient, however, the approach is restricted to environments where everything is planar. Furthermore, the unavoidable noise that affects the measurements, limits the effectiveness of this representation in real-world scenarios.

A more general approach that extracts a predefined set of geometric primitives like planes, spheres, cylinders, cones, and tori has been proposed by Schnabel et al. (2009). The primitives are detected with a RANSAC based approach, which is accelerated by using a specific octree based sub-sampling strategy. Whereas the main focus of this work is the completion and reconstruction of 3D object models it shares the common idea of describing surfaces with a set of primitives with our method. The main differences between the two methods are that the method described in this chapter learns the basic shape types from the data and that it uses a fixed patch size. Learning the primitives on the data has the advantage of better representing surfaces that occur frequently, which cannot accurately be represented by the five primitives of Schnabel et al. but at the cost of being probably less compact.

Limketkai et al. (2005) show the importance of richer semantic descriptions by introducing *Relational Object Maps* that are classical 2D grid maps augmented with information of object locations. The objects are classified using features and are used as landmarks for tasks like localization or place recognition

The method presented in this chapter aims to represent complex 3D models by utilizing the redundant information that occurs in different locations of an environment. We store prototypes of recurrent structures in the form of partial point clouds as words in our dictionary. These prototypes are then used to replace the occurrences of the corresponding partial point clouds in

the original scene. Since we only need to store the references to words in the dictionary and their positions in the scene, we obtain a substantial compression of the original point cloud. As a result, this approach produces highly accurate and compact representations that can be manipulated as efficiently as point clouds. At the same time, this representation has the potential for making additional tasks like place recognition or object detection substantially more effective compared to procedures that operate on the full point cloud.

4.4 Conclusions

In this chapter, we presented an approach for learning compact models of 3D environments. The key idea of our method is to learn a set of repetitive patterns in 3D range data and to use these patterns for approximating the entire point cloud. We apply the Bayesian information criterion to select a dictionary that gives a good tradeoff between storage requirements and model accuracy. Experimental results demonstrate that our approach is able to compute accurate models of complex in- and outdoor scenes. The presented method encodes the environment by a set of surface primitives and, compared to standard occupancy-based representations, provides a step towards semantic annotations. Furthermore, such annotations are applicable to tasks like object recognition. In the following chapter, we will extend the ideas presented in this chapter to deal with colored point clouds. To compactly describe shape and texture, we will use a more flexible coding scheme for the dictionary to better deal with variations in surface or texture descriptions.

Chapter 5

Sparsely Coded Surface Models

In this chapter, we focus on compact models for RGBD data and the specific challenges arising for jointly encoding shape and high-resolution textures. In contrast to the previous chapter surface patches are no longer described with a sample point cloud. Instead we model a local point cloud with a range image and a RGB image and learn independent dictionaries for shape and texture. Furthermore, we introduce a novel dictionary learning scheme based on sparse coding, leading to better surface approximations with less computational effort. We introduce a novel interest point scheme to decompose a registered point cloud into surface patches instead of the scan wise interest point extraction described in the previous chapter. The resulting models are highly compact since we store only a dictionary, surface patch positions, and an abstract description of the depth and if available RGB attributes for every patch. Again, we learn the dictionary in an unsupervised fashion from surface patches sampled from indoor and outdoor maps. We show that we can learn better dictionaries by extending the K-SVD method with a binary weighting scheme that ignores undefined surface pixels. Through experimental evaluation on real-world laser and RGBD datasets we demonstrate that our method produces compact and accurate models.

♦ ♦ ♦ ♦ ♦ ♦ ♦

In the previous chapter, we described a surface representation based on surface primitives. We described each primitive with a sample point cloud. While we could transfer this to RGBD data in a straightforward way, it seems desirable to describe depth and color channel with independent dictionaries. In typical human made environments texture has more variations than shape and in most cases they are not correlated. In other words, a red surface can have arbitrary shape and flat surfaces can have different textures. Splitting depth and RGB channel of a small sample point cloud introduces additional challenges in the dictionary learning process. The BIC based dictionary learning presented in Chapter 4 cannot easily be extended to deal with texture data. Therefore, we approach this problem by representing the surface with a RGB image and a depth image of fixed metric size and a fixed resolution. This discretization of the data introduces an additional parameter (surface resolution) and an additional discretization error. Therefore the method presented in the following is less general than the one described in the previous chapter

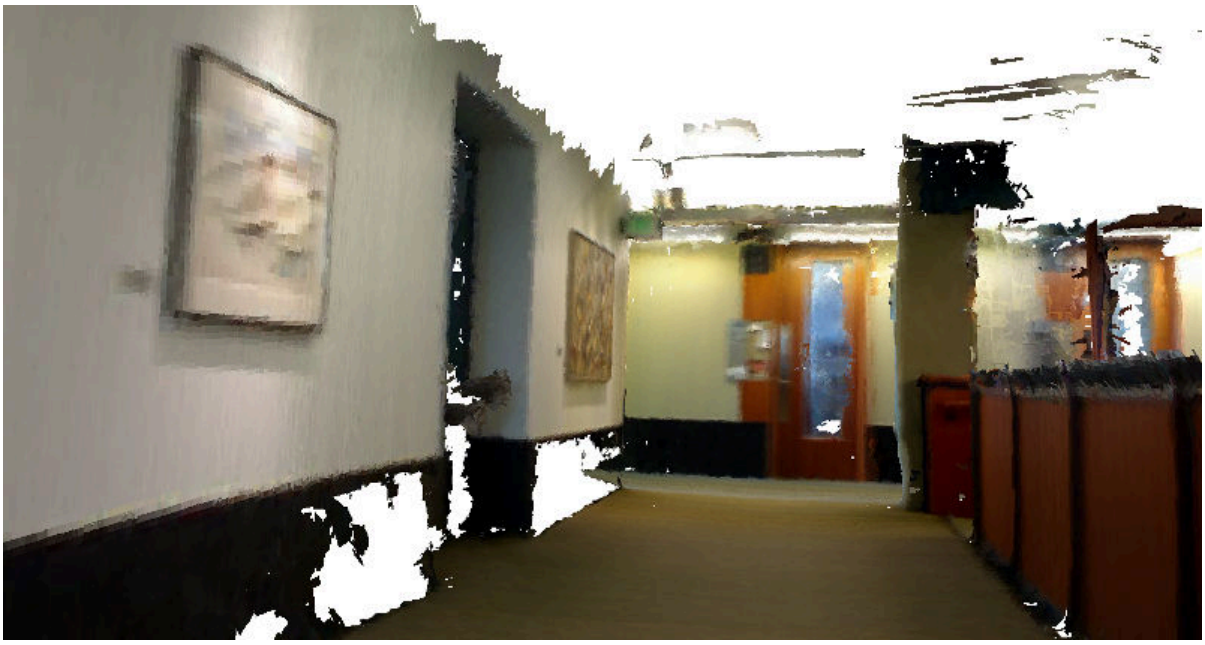


Figure 5.1: This picture illustrates the quality of a model built with our method at a resolution of 2 cm. We represent the surface with patches of size 20×20 cm and encode each patch with a sparse code of an overcomplete dictionary. Generated from a 3.35 GB dataset, the resulting file size is 10.2 MB. For comparison, a colored occupancy Octomap of the same dataset has a size of 44.5 MB.

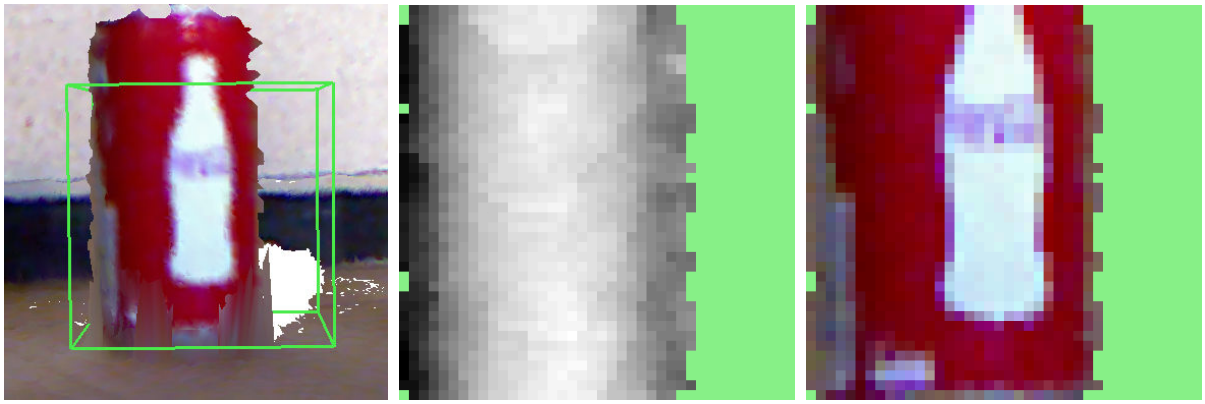
but it can deal with colored point clouds and the fixed image dimensions enable us to use sparse coding as dictionary learning technique.

Sparse coding is a machine learning technique that describes data based on an overcomplete dictionary and a sparse set of linear factors (Olshausen et al., 1997). This introduces a flexible way to scale and combine dictionary entries to approximate the input data (texture and shape). In the context of this chapter, we apply sparse coding to reduce the amount of data necessary to store the surface description. This reduction exploits similarities in shape and texture, which occur in most human-made environments and will be encoded in our dictionary. Based on such dictionaries, we are able to describe shape and texture with a sparse code containing only a few linear factors and the indices of the corresponding dictionary.

One interesting property of sparse coding is that chunks of similar data are usually described with similar sparse codes. This makes it applicable for feature learning in the context of object detection (Bo et al., 2012) because a search or comparison in the compact sparse code space is more efficient and more robust than on noisy raw data. Building compact and accurate 3D models with sparse coding makes it possible to directly search for similar structures, textures, or full objects in the sparse code descriptions, given an expressive dictionary. This is comparable to what we described for surface primitives in Section 4.2.5 but with additional texture information. Accordingly, the approach presented in this chapter is able to produce richer models for mobile robot tasks.

5.1 Overview and Problem Formulation

The main goal of our method is to build textured 3D surface models based on sparse coding. In the following we will refer to local surfaces as surface patches or simply patches. Such patches need a well-defined location in order to compute them from the input data. Furthermore, we



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

Figure 5.2: This figure shows an example RGBD patch **(a)**, the corresponding depth description **(b)**, and the color description **(c)**. The light green areas in **(b)** and **(c)** correspond to undefined areas; marked as 0's in the patch bitmask.

need to learn a dictionary from the surface patches as reference for our codes. Once we have a dictionary, we approximate the surface patches with sparse codes referring to that dictionary. Given the locations, the dictionary and the sparse codes, we can approximate every patch and place it at the right location to recover the entire surface model of the environment.

In our context the input data is a pre-registered set of colored point clouds $\mathcal{P}$. To apply sparse coding, we have to decompose $\mathcal{P}$ into chunks of a particular size and discretize every chunk. We can do this either in pixel or in metric space. Since we intend to jointly represent the information from multiple point clouds acquired at different locations, choosing pixel space would result in differently scaled surface descriptions, making it hard to combine the surface information observed from multiple scans. On the other hand, a discretization in metric space has the disadvantage of introducing additional errors since we usually observe nearby surfaces at a high resolution and store the information at a lower resolution. We represent the surface with small RGBD patches of a given size and a fixed metric resolution on the surface. The depth and RGB values are oriented in the direction of the surface normal. In this way, we discretize only in the two dimensions of the surface and have full accuracy in the third dimension. Let us consider a surface patch of size $10\,\text{cm} \times 10\,\text{cm}$ with a resolution of $1\,\text{cm}$. In our surface representation this would result in a 10×10 matrix for the depth channel and a $10 \times 10 \times 3$ matrix for the color channel. Figure 5.2 shows an example surface patch and the corresponding depth and color channels.

To describe a set of colored point clouds $\mathcal{P}$ we define a Sparsely Coded Surface Model (SCSM) $\mathcal{M}_{\text{SCSM}}(\mathcal{D}^{depth}, \mathcal{D}^{rgb}, \mathcal{I})$ which contains reference dictionaries for both channels $\mathcal{D}^{depth}$ and $\mathcal{D}^{rgb}$ and a scene description $\mathcal{I} = \{\mathbf{i}_1, \ldots, \mathbf{i}_l\}$. The entries $\mathbf{d}_i^{depth}$ of the depth dictionary $\mathcal{D}^{depth} = \{\mathbf{d}_1^{depth}, \ldots, \mathbf{d}_n^{depth}\}$ have the same size as the depth channels of the surface patches and likewise the entries $\mathbf{d}_i^{rgb}$ of the RGB dictionary $\mathcal{D}^{rgb} = \{\mathbf{d}_1^{rgb}, \ldots, \mathbf{d}_m^{rgb}\}$ have the same size as the surface patch color channel. Since shape has less variations than texture, the depth dictionary $\mathcal{D}^{depth}$ is usually substantially smaller than $\mathcal{D}^{rgb}$.

Every $\mathbf{i}_j = \left\langle T_j, \mathbf{c}_j^{depth}, \mathbf{c}_j^{rgb}, \mathbf{b}_j \right\rangle$ stores a 3D pose for the surface patch, T_j, one sparse code for the depth $\mathbf{c}_j^{depth}$, one for the RGB channel $\mathbf{c}_j^{rgb}$, and a bitmask $\mathbf{b}_j$ that is 1 for all observed pixels and 0 for undefined pixels (see also Figure 5.2). In this way, we can decouple the impact of occlusions from the surface description and get sharp surface borders. The storage requirements are rather moderate. Given a patch size of 10×10 we have to store 100 bits or 12 byte, which

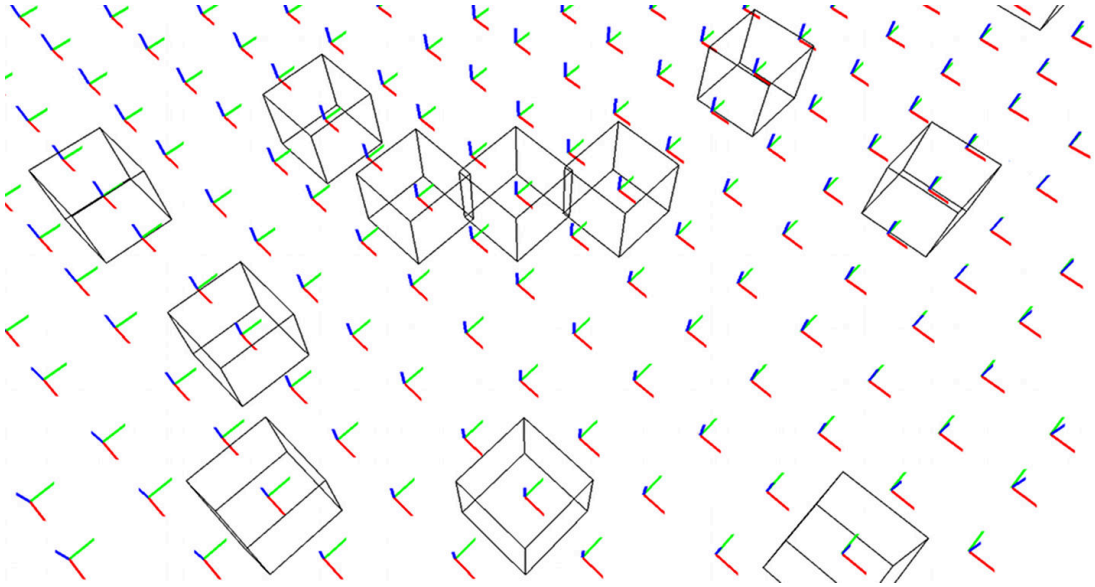


Figure 5.3: This figure illustrates the resulting surface patch locations on a wall. Each triplet of red, green and blue lines correspond to a local coordinate frame for a surface patch whereas the blue lines corresponds to the local surface normal. Each local coordinate frame will be the center of a surface patch cube. As can be seen in this figure, our method seeks to extract parallel surface patch locations to have a good alignment of neighboring cubes. Once computed, we will select a local point cloud that falls into the cube and compute a range image and a texture image.

corresponds to three floating point numbers, per extracted surface patch.

In the following, we will discuss how we partition the input data, learn a dictionary from the extracted patches and calculate a sparse description for every patch.

5.2 Surface Patch Locations

To decompose a point cloud $\mathcal{P}$ into a set of surface patches $\mathcal{S}$ we have to decide where to place our surface patches $\mathbf{s}_i, i \in \{1,\ldots,|\mathcal{I}|\}$. Each surface patch describes a subset of points of $\mathcal{P}$ that fit into a cube of a particular size. Possible locations for the surface patches are defined by the points in $\mathcal{P}$. For compression we are interested in the minimum number of subsets that contain all elements of $\mathcal{P}$ and the locations of those subsets. This is a set cover problem, known to be NP-complete (Feige, 1998). Instead of searching for an optimal solution, we focus on a computationally inexpensive and approximate solution. Please consider that $\mathcal{P}$ may contain millions of points and that the computation of surface patches for every point is already expensive in terms of computation time and memory requirements.

In Section 4.1.3 we described a NARF-based interest point method. This method tends to provide less stable interest points on RGBD data. Another aspect is that the previously described method extracted interest points on every range observation instead of the fused point cloud. This made it necessary to test all extracted instances in a later step whether we can drop them, as described in Section 4.1.4. As consequence, we decided to use a global strategy that focuses more on coverage than on stable locations to extract interest points and rely on the more flexible sparse coding scheme to account for possibly worse surface patch locations.

We uniformly distribute the locations for our surface patches on $\mathcal{P}$ and keep the amount of overlapping regions low. We approach this with a spatial sub-sampling strategy using a voxel grid as described in Algorithm 2. Since it is possible for centroids of neighboring voxels to be

Algorithm 2 Pseudo code for target location computation.

```
for all points p ∈ P do
    Insert p into target voxel
compute centroids for all voxels
for all centroids c ∈ C do
    for all neighboring centroids n ∈ C do
        if ||c − n||_2^2 < minDistance then
            drop centroid and continue with next c
        else if ||c − n||_2^2 > maxDistance then
            add (c+n)/2 to target locations
    add c to target locations
```

very close to each other, we check for such neighbors and neglect the centroid computed from less points if we do not loose coverage. Neighboring centroids can also be too far away from each other resulting in gaps in the surface. To get a maximum coverage we fill all remaining gaps by computing additional centroids for the remaining non-covered points. In case of a grid-aligned flat ground plane our strategy will result in an optimal non-overlapping solution. Computing the surface patch locations in the described way leads to overlapping surface patches in structural complex regions and on surface borders. As result of this procedure, we will have a sparse uniformly distributed set of target locations.

In the next step, we calculate a surface patch $\mathbf{s}_i$ for every computed target location. Additionally, we need the orientation of the surface around every target location. Since we do not compute the NARF descriptor any longer, which provided us with a well-defined transformation, we have to manually compute a well defined transformation for every patch. Therefore we extract a local point cloud for every target location with a diameter similar to our surface patch size and compute the mean and the covariance of this local point cloud. The rotation is computed via singular value decomposition (SVD) on the covariance matrix. The eigenvector of the smallest eigenvalue corresponds to the normal and will be the z-axis direction of the surface patch. The remaining two eigenvectors will correspond to the x- and y-axes, in order of their ascending eigenvalues. Furthermore, we check the ratio between the two larger eigenvalues. If the ratio is close to one, the orientations of the corresponding eigenvectors are not well defined and tend to be strongly influenced by sensor noise. To avoid this situation, we select alternative local x- and y-axes by choosing the two global coordinate axes that have the larger angle to the computed normal and make them orthogonal to the normal. This results in much more regular surface patch orientations on flat surfaces. If less than three points fall into one target location we set the rotation to the identity matrix. Combining the target location and the computed orientation we can define a transformation T_i as a local coordinate frame of our surface patch. A typical result of this procedure can be seen in Figure 5.3. Once we computed T_i, we apply T_i^{-1} to the point cloud neighborhood and compute the values for each patch pixel in color and depth space. Therefore, we project all neighboring points into their target patch pixel and compute the mean for each defined pixel.

5.3 Dictionary Learning with K-SVD

The extracted surface patches $\mathcal{S}$ contain a lot of redundant information on both channels and we intend to compute a sparse approximation to find a compact representation of the data. Let

S be the data matrix that contains every $\mathbf{s}_i$ as i-th column vector. The idea of K-SVD is to learn an overcomplete dictionary D and a sparse description X to approximate S. We can formulate this as a minimization problem with the following equation:

$$\min \|S-(W\odot(DX))\|_F^2 \quad s.t.\forall i\,\|\mathbf{x}_i\|_0 \leq k, \quad i\in\{0,\ldots,|\mathcal{I}|\}. \tag{5.1}$$

Here, $\|A\|_F$ denotes the Frobenius norm, $\mathbf{x}_i$ is the i-th column vector of X, $\odot$ denotes the element-wise matrix multiplication and k is the maximum number of nonzero entries for each column $\mathbf{x}_i$ that approximates a corresponding surface patch $\mathbf{s}_i \approx D\mathbf{x}_i$. To deal with undefined values we extend the standard K-SVD formulation with a binary weighting matrix W that has an entry of 1 for data pixels that were actually observed and a 0 entry otherwise. This information is represented in the bitmasks of the patches. In this way, we ignore the reconstruction results for undefined values and focus the reconstruction accuracy on the observed values. Undefined pixels store a value of zero in S and by multiplying W in an element-wise fashion we ensure that the reconstructed values of undefined pixels are ignored during the optimization. In the following, we will outline our proposed extension to K-SVD, called weighted K-SVD (wK-SVD). Pseudo code of wK-SVD is provided in Algorithm 3. A more detailed description of an efficient K-SVD implementation without weighting can be found in Rubinstein et al. (2008).

Algorithm 3 Pseudo code for wK-SVD algorithm.

```
Initial D_0 filled with randomly selected columns of S
for all iterations do
    for all x_i ∈ X do                                   ▷ compute sparse code matrix X
        x_i = argmin ||s_i − (w_i ⊙ (D x_i))||_2^2
    for all d_j ∈ D do                                   ▷ optimize dictionary D
        L := Indices of X columns with d_j ≠ 0
        (U,S,V) = SVD(S^(L) − W^(L) ⊙ (D X^(L) − d_j X_row(j)))
        d_j = normalize(U)
        X_row(j) = S · V
```

Starting from an initial dictionary D_0, wK-SVD iteratively achieves an improved representation of S. In each iteration wK-SVD alternates between computing the sparse code matrix X based on the current dictionary and optimizing each dictionary entry based on the current sparse code matrix. We compute the sparse code matrix X with orthogonal matching pursuit (OMP) (Pati et al., 1993). OMP is a greedy algorithm that decouples the computation of X into $|\mathcal{I}|$ sub-problems, one for every data item $\mathbf{s}_i$. This makes it easy to parallelize OMP, which results in a major speedup in our implementation. Given a dictionary D, OMP iteratively solves the following minimization problem:

$$\hat{\mathbf{x}}_i = \underset{\mathbf{x}_i}{\operatorname{argmin}} \|\mathbf{s}_i-(\mathbf{w}_i\odot(D\mathbf{x}_i))\|_2^2 \quad s.t.\,\|\mathbf{x}_i\|_0 \leq k, \quad i\in\{0,\ldots,|\mathcal{I}|\}. \tag{5.2}$$

Again, we have added a weighting vector $\mathbf{w}_i$ which is the i-th column vector of the weighting matrix W to neglect undefined values. In every iteration we search the code word that best matches the current residual and append it to the current $\mathbf{x}_i$. Once a new code word is selected, the patch $\mathbf{s}_i$ is orthogonally projected to the span of the selected code words, and the residual is recomputed for the next iteration until either the maximum number of entries k is computed or the residual error is below a minimum error.

Once we have computed the sparse code matrix X, we update the dictionary on a per-entry basis. For each dictionary entry $\mathbf{d}_j$ we select the patches that have a non-zero entry in their

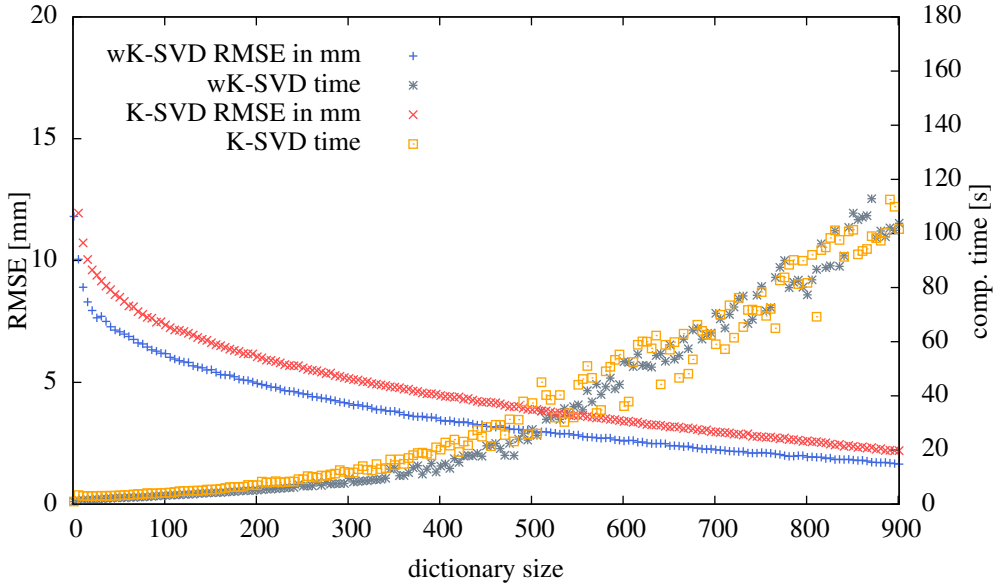


Figure 5.4: This plot illustrates the accuracy and time requirements for our method on a range only reference dataset. We applied wK-SVD and K-SVD with varying dictionary sizes between 1 and 901 and a step size of 5. As can be seen, the RMSE of wK-SVD is smaller than the RMSE of K-SVD for all dictionary sizes.

sparse code. Put another way, we select the patches that use $\mathbf{d}_j$ for their approximation. A codeword $\mathbf{d}_j$ is updated via SVD on the residual matrix containing all differences between the selected patches and their approximations using all codewords and their sparse codes without $\mathbf{d}_j$. The outline of wK-SVD is provided in Algorithm 3. Again we force the residuals of the patches and their approximations to be zero for undefined values. In general, K-SVD and wK-SVD can get stuck in a local minimum, depending on the initial dictionary. However, for practical scenarios Aharon et al. (2006) reported that K-SVD usually converges to a good dictionary for a wide range of initializations. In our implementation, we fill the initial dictionary of size K with randomly chosen input data entries.

Once we computed all parts of our model $\mathcal{M}_{\text{SCSM}}(\mathcal{D}^{depth}, \mathcal{D}^{rgb}, \mathcal{I})$, we can rebuild a point cloud out of the model. This is done by going through all elements of $j \in \mathcal{I}$ and decode the j-th element with

$$\hat{\mathbf{s}}_j^{depth} = \mathcal{D}^{depth} \cdot \mathbf{c}_j^{depth}. \tag{5.3}$$

With $\mathbf{c}_j^{depth} = [\lambda_j^1, \ldots, \lambda_j^N]$ this can be rewritten as

$$\hat{\mathbf{s}}_j = \lambda_j^1 \cdot \mathbf{d}_1^{depth} + \ldots + \lambda_j^N \cdot \mathbf{d}_N^{depth}. \tag{5.4}$$

Remember that the sparse code $\mathbf{c}_j$ is a sparse vector with only k nonzero entries. The same scheme is applied for decoding the color channel. The joint information is projected into a colored 3D point cloud for all defined values in $\mathbf{b}_j$. As a final step, we apply T_j to put the patch point cloud at the right location in the resulting point cloud.

5.4 Experiments

In this section, we evaluate our method on various datasets. The interesting evaluation quantities for our approach are mainly the accuracy and compactness of the resulting model and the

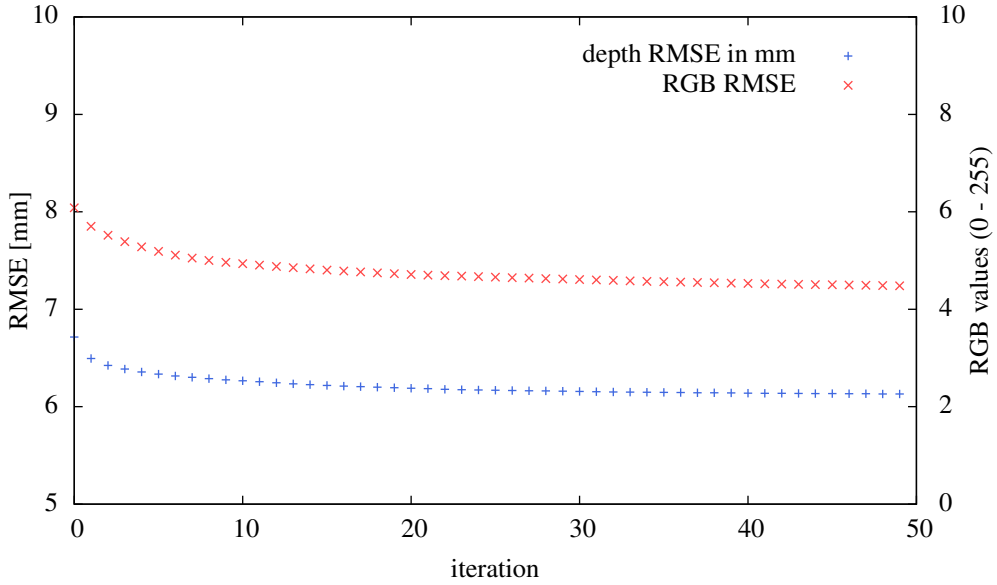


Figure 5.5: This plot illustrates the RMSE of the depth and RGB dictionary learning with wK-SVD . In both cases wK-SVD converges very fast to an RMSE of 6 mm for depth and a RMSE of 4.5 for RGB.

time needed to compute the result. As measure of accuracy, we use the root mean squared error (RMSE) of the dictionary learning method wK-SVD, which is calculated on the discrete surface patches and their approximations according to Equation 5.1. While this error calculation provides insights into how close we get to the optimal solution given our discretization, we are additionally interested in the error on point level for comparison with other methods and different discretizations. Therefore, we search for every point in the resulting point cloud the nearest neighbor in the reference point cloud and vice versa. Over all pairs, we compute the RMSE.

To demonstrate how compact the results are, we provide the file sizes of the input dataset and the resulting file sizes in their corresponding representation in Table 5.1. Furthermore, we provide timings required for learning the dictionary since this is usually the computationally most expensive step. All results were computed on a standard desktop machine with an Intel i7 CPU at 3 GHz and with four cores. Through all experiments we chose a sparsity factor $k = 5$.

5.4.1 Accuracy and Runtime versus Dictionary Size

In our first experiment, we evaluated the accuracy and runtime requirements of our method on a range only dataset, which we already used in Chapter 4.2.1. We computed a point cloud decomposition containing 2,585 surface patches of size 16 cm and a resolution of 2 cm. To this model we applied standard K-SVD and wK-SVD to learn a dictionary. We varied the dictionary size between 1 and 901 with a step size of 5. Figure 5.4 gives a detailed overview of the resulting RMSE per patch pixel and the run time depending on the dictionary size. A dictionary of size 70 already results in an RMSE of approximately 1 cm with a computation time around 2.5 seconds using four threads. The difference between wK-SVD and K-SVD is also very interesting. The RMSE in the wK-SVD is smaller than for K-SVD for all dictionary sizes. The timings for wK-SVD are faster for dictionaries smaller than 400 and it performs comparable to K-SVD for dictionaries larger than 500. The faster computation time probably is related to the typically fewer entries in the sparse code matrix for wK-SVD for smaller dictionaries.

Figure 5.6(a) shows the input point cloud, (b) a corresponding SP model and (c) the SCSM. Both methods used a dictionary of size 70. As can be seen, the model computed by our method

Dataset	Scene 5.6c		RGBD Corridor		Surface Detection
method	Chapter 4	Chapter 5	Octomap	Chapter 5	Chapter 5
dict. size (D/RGB)	70 / -	70 / -	- / -	100 / 500	120 / 114k*
patch size	0.1 m	0.16 m	-	0.2 m	0.1 m
resolution	-	0.016 m	0.02 m	0.02 m	0.005 m
file size:					
input	1.2 MB	1.2 MB	3.35 GB	3.35 GB	4.7 MB
result	431 KB	357 KB	44.5 MB	10.2 MB	559 MB
relative size	35%	29%	1.3%	0.3%	-
RMSE depth	0.058 m	0.016 m	0.016 m	0.017 m	-
RMSE RGB**	-	-	25.1	19.9	-
time	7 min	2.5 s	56 s	8 min	5 s

Table 5.1: This table gives an overview of the experimental evaluation for each dataset and method. The dictionary size and RMSE errors are split in depth and RGB. The timings are measured on a standard desktop CPU with 3 GHz. (*) The RGB channel was not compressed. (**) Unit is intensity value.

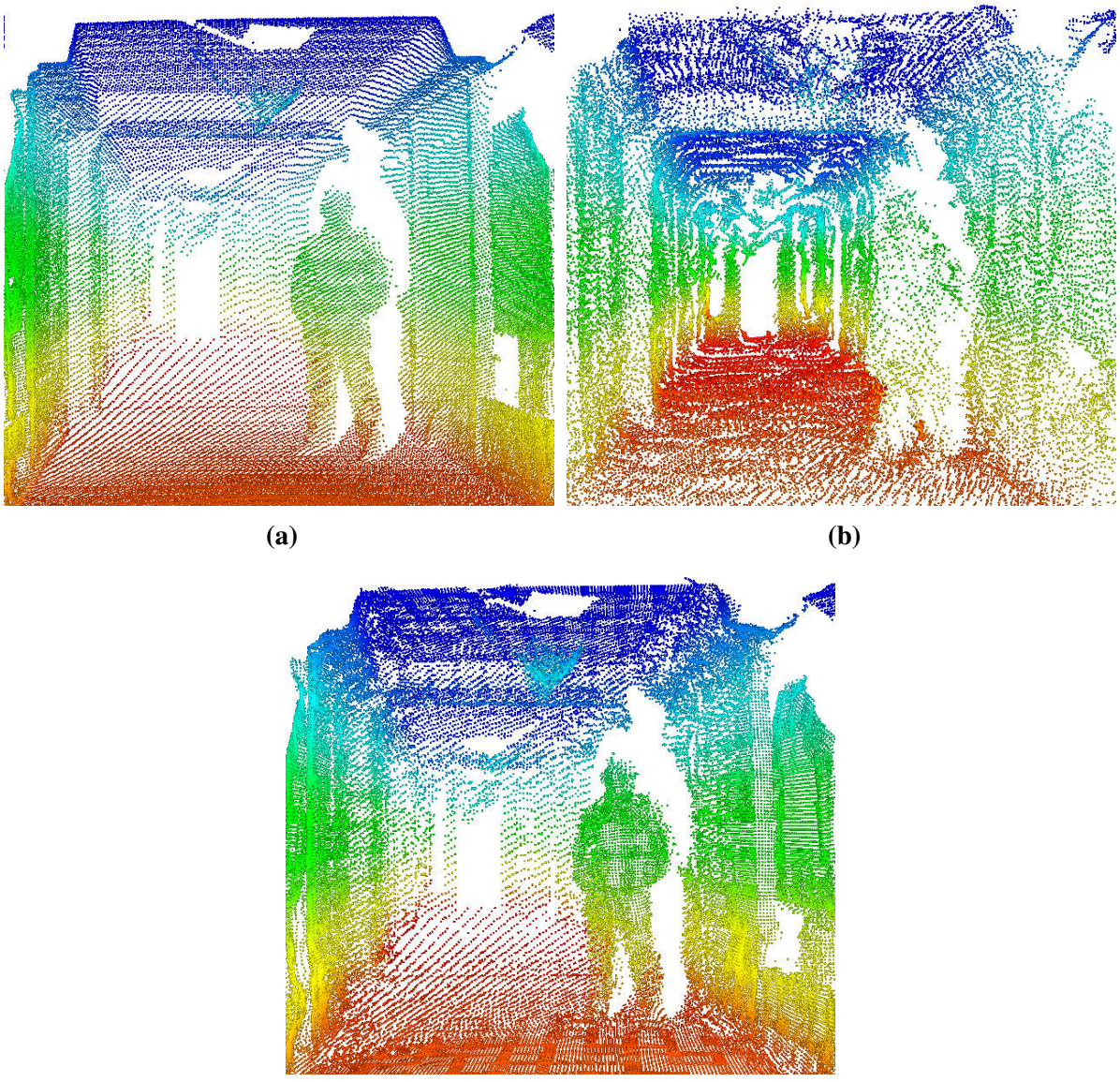
partially maintains the original scan structure as a result of the stored bitmasks and additionally provides a substantially better reconstruction accuracy. The resulting RMSE per point is approximately 1.6 cm and the resulting file size is 357 KB, compared to an RMSE of 5.8 cm and a file size of 421 KB. Thus, the described method clearly outperforms the surface primitives model in terms of the resulting file size, the RMSE, and the time needed to learn the dictionary, at least for the chosen patch resolution in the SCSM. The main reason for the better results is that we use discrete surface patches instead of a sample point cloud for our dictionary entries. In this way, a depth value is stored with one float instead of three in the point cloud representation. Furthermore, we use the gained space to store additional bitmasks and sparse codes with every surface description. Together this results in a better reconstruction quality. The SCSM results could look different for other patch resolutions.

5.4.2 Compact Models

In the next experimental setting we applied our method on a SLAM solution[1] containing 707 registered RGBD scans. This dataset was acquired in the corridor of a typical office building. An overview of the dataset can be seen in Figure 5.7(a). We applied our method to the accumulated point cloud and built a model with 41,358 surface patches of size 20 cm and a resolution of 2 cm. It took eight minutes to compute the depth dictionary of size 100 and the RGB dictionary of size 500. Figure 5.5 plots the evolution of the RMSE during the dictionary learning for both dictionaries over the different iterations.

For comparison, we created a colored occupancy Octomap of the dataset with a voxel size of 2 cm, which resulted in the colored model shown in Figure 5.7(c). An enlarged view of the same region in our model is shown in 5.7(d). Our model looks much smoother since we discretize only in two dimensions and the sparse coding also reduces the noise in the RGB data as can be seen on the wall on the left hand side. The depth and RGB errors of the Octomap and our model have the same magnitude and mainly reflect the discretization. Furthermore, our model is more than four time smaller with 10.2 MB versus 44.5 MB at the cost of more computation time. Note that the maximum root mean squared error is $\sqrt{255^2+255^2+255^2} \approx 441.7$ for intensity values on RGB data.

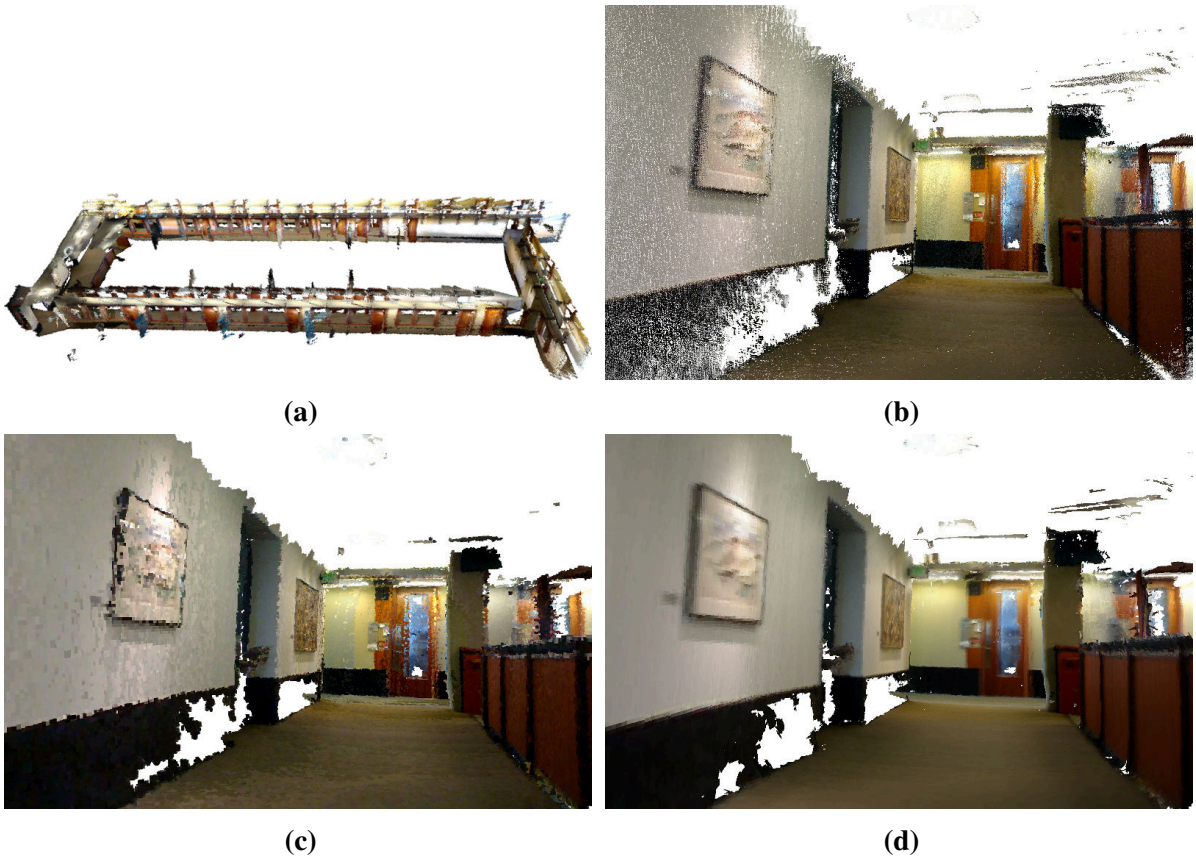
[1] Courtesy of Peter Henry



(a) **(b)**

(c)

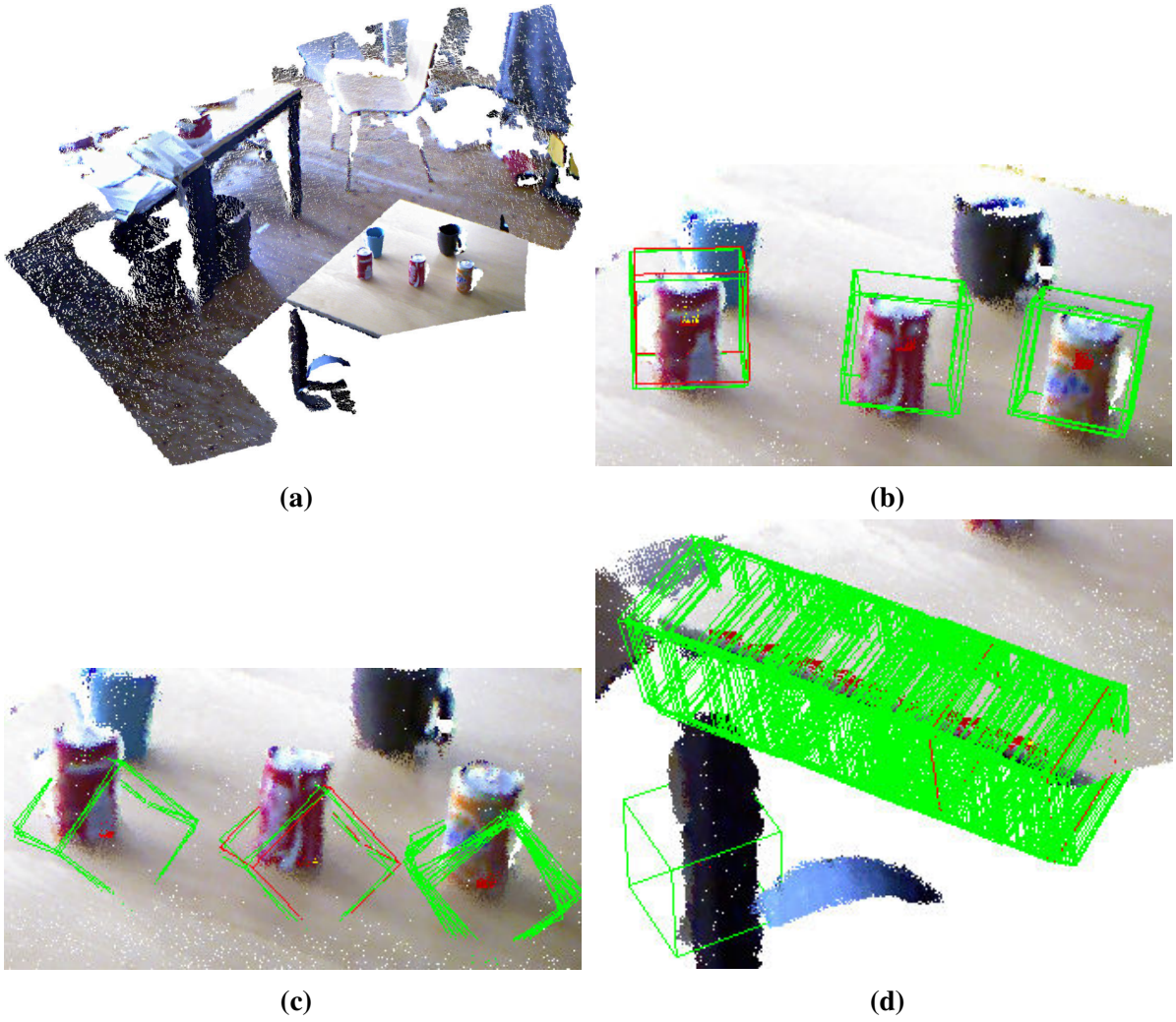
Figure 5.6: The input point cloud of this experiment is shown in **(a)**. The cloud contains only range information and the displayed colors are height-dependent. The reconstruction quality of the corresponding SP model with a dictionary size of 70 is shown in **(b)**. A model built by our method with a dictionary of the same size is shown in **(c)**. As can be seen, the model computed by our method maintains part of the original scan structure as a result of the stored bit masks and at the same time provides a substantially better reconstruction accuracy. The resulting RMSE per point is approximately 1.6 cm and the resulting file size is 357 KB compared to an RMSE of 5.8 cm and 421 KB achieved by the SP model.



(a) **(b)**

(c) **(d)**

Figure 5.7: An overview of the resulting model for the RGB Corridor dataset is shown in **(a)**. A representative part of the model is shown as a raw point cloud **(b)**, Octomap **(c)** and our method **(d)**. The Octomap occupancy grid has a file size of 44.49 MB with a voxel size of 2 cm, an RMSE of 0.016 m on the depth channel and an RMSE of 25.1 intensity value on the RGB channel. Our model has a file size of 10.2 MB and stores a depth dictionary with 100 entries and a RGB dictionary with 500 entries. The RMSE is 0.017 m for the depth channel and 19.9 intensity value for the RGB channel. In comparison, Octomap has a slightly lower error in the depth data and a slightly larger error in the RGB data. The major difference between the two results is the resulting file size, which is four times larger for Octomap.



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

Figure 5.8: We performed an object detection experiment on the RGBD scan shown in (a). For this, we computed a model and created a surface patch for every point in the point cloud and learned a dictionary. From the resulting model we selected a surface patch and searched for similar sparse codes of the depth channel in the entire model. (b) shows the results for upper can parts search, (c) for lower parts of a can standing on a flat surface and (d) shows the results for the edge of a table. The green cubes mark highly similar surface patches for the corresponding search queries. In all three cases we successfully found all surface patches that contain similar surface descriptions. Note that there is only one false positive, in the lower part of the table leg shown in (d).

Object Detection

Since sparse coding has previously been applied successfully in object recognition applications, the question arises whether we can directly search for objects in our models by comparing sparse codes. To evaluate this, we acquired an RGBD scan containing multiple small soda cans on a table (see Figure 5.8(a)). To demonstrate similarities in the sparse code space we extracted surface patches for every point in the model instead of applying the proposed point cloud decomposition. In this way, we avoid the problem of extracting the sparse codes on suboptimal positions since our point cloud decomposition scheme does not explicitly search for stable key points or patch locations.

For a real object detection scenario one would compute a model of the environment with our method along with a reference model for the object we are searching for, with surface patches in every point, based on the same dictionary. In such a setting, searching an object in an environment boils down to finding surface patches that are very similar to a representative subset of the sparse description of the reference model.

Figures 5.8(b), (c) and (d) show the results of three different search queries. The corresponding reference surface is marked with a red cube, whereas the green cubes correspond to the most similar surface patch regions. We forced an 80% similarity on the indices and used the Euclidean norm on the linear factor as similarity score. Furthermore, we show only results with a value below a threshold of 0.05. As can be seen for all three settings, our approach reliably found similar surface patches. It produced only one false-positive in the lower part of the table leg. This indicates that our approach can be applied for object recognition tasks despite of the simplistic nature of the detection algorithm. It furthermore highlights the richer description of our models compared to standard representations.

5.5 Related Work

There exist a wide variety of different representations for 3D environments. Depending on the desired application they focus either on accuracy, compactness, or rendering performance. Recent developments in Simultaneous Localization and Mapping (SLAM) and the availability of RGBD-cameras made it possible to obtain large colored metric models. Whereas a major advantage of colored point clouds lies in their accuracy, their drawback is the amount of data that has to be stored, since every single observed pixel is represented in both 3D and RGB-space. Therefore, most RGBD-based SLAM systems either use only a subset of features internally (Henry et al., 2010) or represent only small, local spaces (Newcombe et al., 2011).

Fairfield et al. (2007) proposed a compact voxel representation based on Octrees in the context of mobile robotics. Wurm et al. (2010) developed an open source implementation called Octomap which can store additional RGB information together with 3D occupancy. Kammerl et al. (2012) used Octrees to compress colored point cloud streams by transmitting only the differences in successive frames. For large voxel sizes, Octrees are very compact but introduce major discretization errors. If a small voxel size is chosen the resulting models are more accurate but not compact anymore.

In Chapter 4, we proposed to model the geometric structure of an environment based on surface primitives. In this approach, a surface primitive is a small point cloud that can be inserted in a model at multiple locations. The corresponding model uses a dictionary of primitives and stores a set of 6 degrees-of-freedom locations together with the corresponding dictionary indices, describing where to place which of the small point clouds to reconstruct the input data. In principle, this can be regarded as a variant of bag-of-words models.

A technique similar to bag-of-words methods is sparse coding. Instead of choosing one dictionary entry of a bag-of-words to describe an input signal, sparse coding allows to describe a signal as a linear combination of multiple dictionary entries. Sparse coding has already been successfully applied in the context of signal approximation (Rubinstein et al., 2010), image denoising (Hyvarinen et al., 2001; Elad and Aharon, 2006) and more recently to learn feature descriptions for object recognition tasks on both vision only and RGBD data (Bo et al., 2012; Yang et al., 2009), resulting in superior recognition results compared to standard bag-of-words models.

In this chapter, we follow a similar idea to describe models as a set of local descriptions. However, in contrast to the method described in the previous chapter, we use discrete local surface descriptions and apply sparse coding to approximate the surfaces. Instead of a greedy dictionary learning using the Bayesian information criterion to guide the search, we apply an efficient variant of K-SVD (Rubinstein et al., 2008) and learn smaller dictionaries while achieving substantially lower reconstruction errors. In addition, our method is also able to represent full RGBD information instead of just the 3D surface model.

5.6 Conclusions

In this chapter, we presented a novel approach to construct compact colored 3D environment models representing local surface attributes via sparse coding. The key idea of our approach is to decompose a set of colored point clouds into local surface patches and describe them based on an overcomplete dictionary. The dictionaries are learned in an unsupervised fashion from surface patches sampled from indoor maps. We demonstrate that our method produces highly compact and highly accurate models on different real-world laser and RGBD datasets. Furthermore, we show that this method outperforms the approach described in the previous chapter in terms of compactness, accuracy and computation time for a well-chosen surface resolution. One disadvantage of the method presented in this chapter is that we have to represent the full surface at the same resolution. In contrast, the SP method presented in Chapter 4 has the advantage of learning dictionaries with entries of different resolution. In this way, it can better adapt to sparse surface data measured at higher ranges. To overcome this limitation, we will present a hierarchical extension of Sparsely Coded Surface Models in the following chapter.

Chapter 6

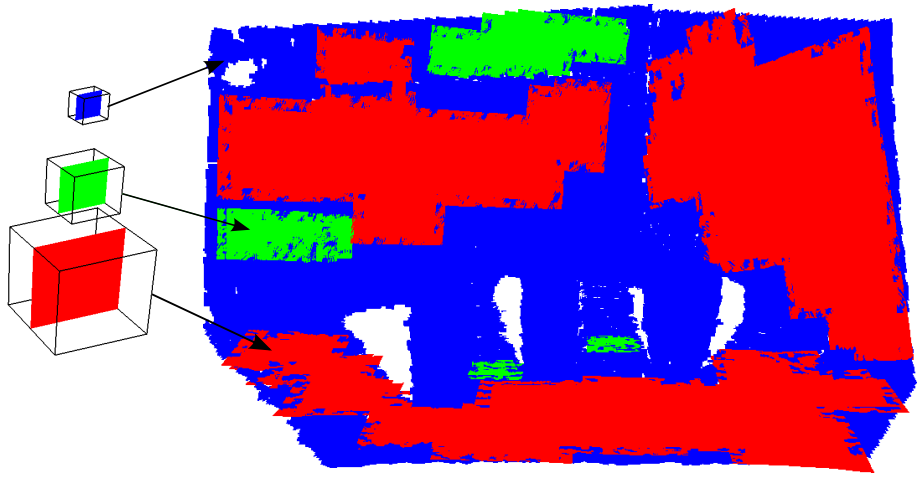
Hierarchical Sparsely Coded Surface Models

In this chapter, we present a hierarchical extension to the Sparsely Coded Surface Models described (SCSM) in Chapter 5. The main idea of this method is to construct textured 3D environment models in a hierarchical fashion based on local surface patches. Compared to the approaches presented in the previous chapters, the hierarchy enables our method to represent the environment with differently sized surface patches. The reconstruction scheme starts at a coarse resolution with large patches and in an iterative fashion uses the reconstruction error to guide the decision as to whether the resolution should be refined. This leads to variable resolution models that represent areas with little variation at low resolution and areas with large variations at high resolution. Again we compactly describe local surface attributes via sparse coding based on an overcomplete dictionary. Furthermore, we learn a dictionary directly from the input data and independently for every level in the hierarchy in an unsupervised fashion. Practical experiments with large-scale RGBD data sets show that the hierarchical method computes even more compact models than the regular SCSM approach at a comparable accuracy.

♦ ♦ ♦ ♦ ♦ ♦ ♦

In this chapter, we propose a hierarchical extension of the Sparsely Coded Surface Models described (SCSM) method presented in the previous chapter to construct compact and textured 3D models of an environment from RGBD data. In Chapter 4 and Chapter 5 we had to select a reasonable size for the surface patches. Therefore, the methods could only exploit similarities at the scale of the chosen patch size. In human made environments, similarities in texture and shape can occur at an arbitrary scale. Representing similarities at arbitrary scale would mean that we have to learn an infinite number of dictionaries, from atomic level to planetary level and would have to find the best patch locations and scale for a surface to represent it. Obviously this leads to an immense search problem.

For RGBD cameras, we can limit the minimum scale at which we can represent something according to the sensor resolution. The limited field of view and the maximum range also provide a maximum scale for observable similarities. Furthermore, we intend to systematically



(a) Surface coverage per patch size

(b)

Figure 6.1: This figure illustrates our hierarchical surface description scheme. The large red areas in picture **(a)** correspond to the regions that are represented at the highest level using the large patches. The green areas are represented with smaller patches at level two and the blue area is represented with small high-resolution patches. Image **(b)** shows the corresponding model with a size of 343 KB compared to 4.7 MB of the original point cloud.

choose only a subset of possible patch sizes and represent the environment accordingly. Analog to Chapter 5, we describe local surface attributes with sparse codes which refer to an overcomplete dictionary of the corresponding patch size. This leads to models with high resolution in areas with high variation and low resolution in areas with minor variations. Figure 6.1 shows an example for a hierarchical sparsely coded surface model (HSCSM). The red surfaces shown in (a) correspond to the largest surface patches and the blue surfaces to the smallest patches. The picture in (b) shows the corresponding model with RGB information.

We construct such hierarchical models in a greedy fashion by starting at the highest level of our hierarchy with the largest surface patches and use the standard deviation in depth and RGB-space to decide as to whether a surface should be represented at the current level. The idea is to describe everything that cannot be accurately represented at the current level at a lower level with higher resolution. To guarantee coverage we describe all remaining data at the lowest level. Once we know the level of detail for all data, we apply K-SVD to learn a dictionary for every level in the hierarchy and calculate a sparse code for every local surface.

The remainder of this chapter is organized as follows: in Section 6.1 we will start with the formal description of the hierarchical model. In Section 6.2, we will describe our approach to compute surface patch locations at different scales. Afterwards, we describe our multilevel dictionary learning scheme in Section 6.3. In Section 6.4, we present our experimental evaluation, followed by a discussion of related work in Section 6.5. We conclude the chapter in Section 6.6.

6.1 Overview and Problem Formulation

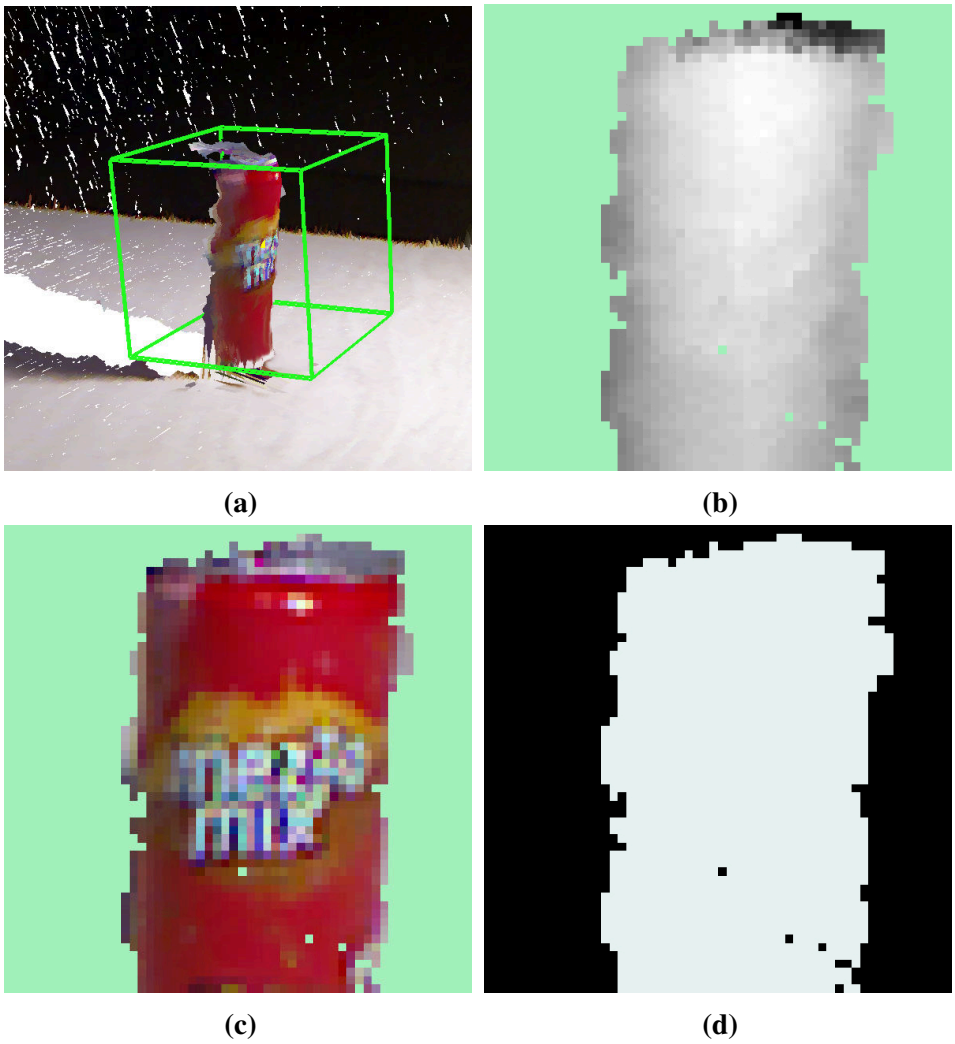
Similar to Chapter 5, we represent colored 3D data with a set of surface patches and encode every of these local surfaces using sparse coding based on a reference dictionary. In this way, we can exploit the repetition of structures and textures and build highly compact models. Note that we need more than one surface patch to describe a volume with this strategy. One major challenge in this context is that the scale at which structure or texture reappears can be arbitrary. Therefore, it seems desirable to introduce surface patches of different sizes to capture and exploit similarities at different scales.

Similar to SCSM we encode local colored 3D data at a well-defined location as set of 2D images for three channels, which are depth, RGB, and bitmask (see Fig. 6.2d). The grey value on the depth channel, shown in (b), corresponds to the weighted mean of the distance to the surface in direction of the normal. We compute a RGB value for a pixel as the weighted mean of the RGB values that fall into the corresponding pixel, an example is shown in (c). Note that we can compute a local range image and a corresponding texture at every location in an unorganized point cloud as described in Section 2.2.2. We use the RGBD sensor model described in Section 3.3.2 to weight the error according to the error of the maximum range of the sensor. The bitmask, shown in 6.2d (d), encodes which pixels have a valid value. In this way, we can encode surfaces that are smaller than the given patch size and can also deal with occlusions (see Figure 6.2). In our current implementation, we start with a maximum patch size and a minimum resolution at the highest level and halve the size in every dimension of the patches and double the resolution for every following level.

The main intuition of the hierarchical modeling is that we try to explain everything on the current level and use coverage and error distributions to guide the decision as to whether a surface is well represented or should be represented at a lower level and higher resolution.

Starting with a pre-registered set of colored point clouds as input data we build a model using large patches with low resolution to represent the full data set. If a patch does not cover more than 90% we can easily describe it at the next lower level. The standard deviation ρ_c of a pixel in a patch gives us the information how good the patch resolution can represent the data at the current level of the hierarchy. We introduce a maximum standard variation ρ_{max}^{depth} and ρ_{max}^{rgb} for both channels to control how accurate a model will represent details. If the standard deviation of a pixel is above one of these thresholds we set the bitmask of this pixel to 0 and postpone this surface for the next level. Figure 6.3 illustrates the procedure. The left picture shows the input point cloud and the right picture shows the point cloud of all remaining points after modeling the first level. The latter correspond to the input for the next level. At the lowest level we accept all patches independent of the computed standard deviation to ensure that the model is complete and represents all data.

We define a hierarchical sparsely coded surface model as $\mathcal{M}_{\text{HSCSM}}(\mathcal{H}, \mathcal{D}^{depth}, \mathcal{D}^{rgb}, \mathcal{I})$ with the hierarchy description $\mathcal{H} = \{h_1, \ldots, h_{|\mathcal{H}|}\}$ consisting of a tuple $h_j = \langle ps_j, pr_j \rangle$ for the j-th level, which defines the patch size ps_j and patch resolution pr_j, reference dictionaries for both channels, $\mathcal{D}^{depth}$ and $\mathcal{D}^{rgb}$, and a scene description $\mathcal{I} = \{\mathbf{i}_1, \ldots, \mathbf{i}_{|\mathcal{I}|}\}$. The reference dictio-



(a) **(b)**

(c) **(d)**

Figure 6.2: This figure shows an example RGBD patch **(a)**, the depth description **(b)**, the color description **(c)** and the bitmask **(d)**. The light green areas in **(b)** and **(c)** correspond to undefined areas, which are marked as zeroes in the patch bitmask **(d)**.

(a) **(b)**

Figure 6.3: This figure illustrates the greedy-based hierarchical modeling scheme we apply. The left picture shows the input point cloud and the right picture shows all points that could not be accurately represented at the first level of the hierarchy. These points are the input for the next level.

nary of every channel has entries that correspond to a particular level of the hierarchy $\mathcal{D}^{depth} = \{\mathcal{D}_1^{depth}, \ldots, \mathcal{D}_{|\mathcal{H}|}^{depth}\}$ that we construct by concatenating the learned dictionaries for every level. All entries $\mathbf{d}_k^{depth}$ of a depth dictionary $\mathcal{D}_j^{depth} = \{\mathbf{d}_1^{depth}, \ldots, \mathbf{d}_n^{depth}\}$ with $n = |\mathcal{D}_j^{depth}|$ have the same number of rows and columns as the depth channels of the surface patches of their corresponding level. Currently we us the same dimensions for all levels, but this is not mandatory. Every $\mathbf{i}_j = \langle T_j, \mathbf{c}_j^{depth}, \mathbf{c}_j^{rgb}, \mathbf{b}_j, l_j \rangle$ stores a transformation T_j, consisting of the 3D pose and the orientation for the surface patch, one sparse code for the depth $\mathbf{c}_j^{depth}$ and one for the RGB channel $\mathbf{c}_j^{rgb}$, a bitmask $\mathbf{b}_j$ that is 0 for undefined pixels and 1 for defined pixels and the index of its level l_j. In the following we will discuss how we compute the positions for our surface patches and give an overview of our dictionary learning scheme.

6.2 Hierarchical Surface Patch Locations

To fully represent a set of input point clouds $\mathcal{P}$ we need to find locations for the surface patches such that all data is covered with the patches. As already mentioned in the previous chapter, finding the minimum number of subsets that contain all elements of $\mathcal{P}$ is a set covering problem known to be NP-complete Feige (1998). Since we have different patch sizes for every level of our hierarchy, we can neither follow the strategy described in Section 4.1.3 nor the strategy described in Section 5.2 since both try to uniformly distribute surface patches of similar size. Therefore, we propose to solve this problem by applying a greedy algorithm that searches for the best patch locations at every level starting with the largest patches.

Possible ways to define the quality of patch locations are the area they cover, the error introduced by the patch discretization, and the error of the reconstruction based on our dictionary. Since we compute the dictionary in a later step, we can only compute the reconstruction error by either introducing a two pass encoding, which results in better patch locations in the second pass, or using a general pre-computed dictionary for every patch size and resolution. The error introduced by representing the data by a patch depends on the level. Since we reject patch pixels that introduce high errors and try to represent the corresponding data at a lower level with smaller patches and with a higher resolution, this directly influences the area covered by a patch. Therefore, we sub-sample the point cloud $\mathcal{P}$ to a resolution close to the resolution of a patch pixel, resulting in a point cloud $\mathcal{P}'$ and calculate the approximate coverage value for $\mathcal{P}'$ by computing surface patches and counting the number of valid pixels. We use the coverage value to guide our greedy selection of possible locations and start with the location that corresponds to the best coverage value. Once selected, we update the coverage of the neighborhood of a location according to the area the corresponding patch covers. To compute a coordinate frame of a patch we compute the normal of the range data in the neighborhood of a candidate location and chose x- and y-axes from the two global coordinate axes that have the larger angle to the computed normal and make them orthogonal to the normal. In this way, we constrain the surface patches in their rotation around the normal according to a global coordinate frame. Given constrained orientations for surface patches, we can extend the greedy search to first select a maximum coverage location and then search in the neighborhood along the defined x,y-directions for possible candidate locations which are well-aligned to our surfaces patches. This works best in cases were most of the surface normals show in the directions of the base vectors of the global coordinate frame. To achieve this, we can either compute a statistic over all normals in the dataset or, if an IMU is available, we can use the gravity vector to align the global coordinate frame. As a result of this alignment the average distance between neighbor-

ing surface patch locations is increased compared to distance-only based methods while still covering the same surface but with less overlap.

6.3 Hierarchical Dictionary Learning

The extracted surface patches $\mathcal{S}$ contain a lot of redundant information on RGB and depth channels. Thus, we intend to compute a sparse approximation to find a compact representation of the data.

To learn our reference dictionary we apply wK-SVD , as described in Section 5.3, independently on every level for the depth channel and the RGB channel and concatenate the dictionaries per channel and update the indices of the sparse codes accordingly. Since the impact of errors on the resulting model scales with the size of a patch, we require a low reconstruction error for the higher levels. We start with a maximum number of allowed dictionary entries or the maximum number of data entries for the current level. Since wK-SVD reports back the number of unused entries, we crop the dictionary afterwards and proceed with the next level.

We can decode a Hierarchical Sparsely Coded Surface Model $\mathcal{M}_{\text{HSCSM}}(\mathcal{H}, \mathcal{D}^{depth}, \mathcal{D}^{rgb}, \mathcal{I})$ back into a point cloud if needed. This can be done by iterating through all elements of $\mathcal{I}$ and decoding the j-th element with

$$\hat{\mathbf{s}}_j^{depth} = \mathcal{D}^{depth} \cdot \mathbf{c}_j^{depth}. \tag{6.1}$$

With $\mathbf{c}_j^{\text{depth}} = [\lambda_j^1, \ldots, \lambda_j^N]$ this can be rewritten as

$$\hat{\mathbf{s}}_j = \lambda_j^1 \cdot \mathbf{d}_1^{depth} + \ldots + \lambda_j^N \cdot \mathbf{d}_N^{depth}. \tag{6.2}$$

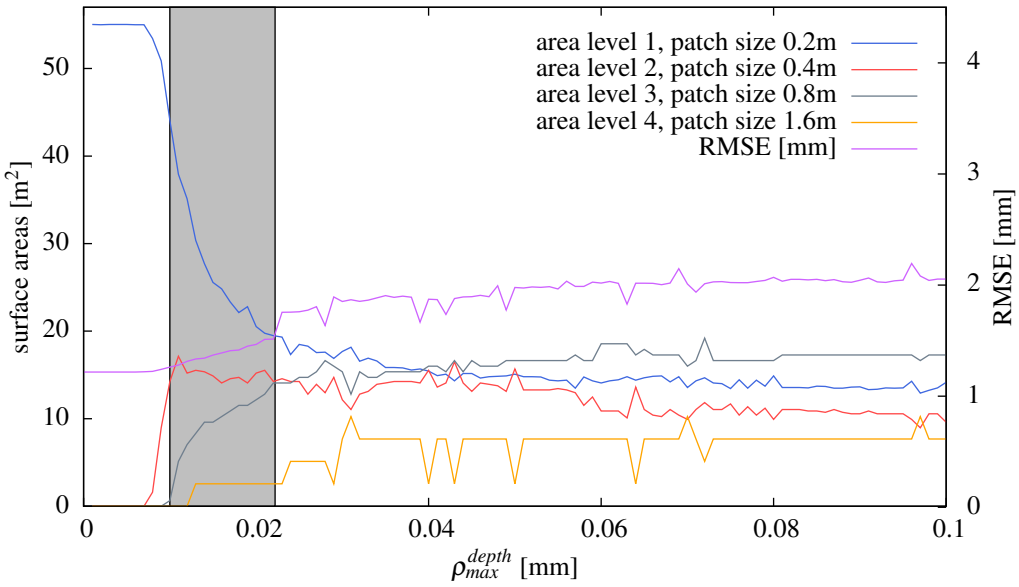
Note that the sparse code $\mathbf{c}_j$ is a sparse vector with only k nonzero entries. We apply the same scheme for decoding the color channel and project the joint information into a colored 3D point cloud according to the corresponding scale information $h_{\text{level}_j} \in \mathcal{H}$ for all values defined in $\mathbf{b}_j$. As the final step, we apply T_j to put the patch point cloud at the right location in the resulting point cloud $\hat{\mathcal{P}}$.

6.4 Experiments

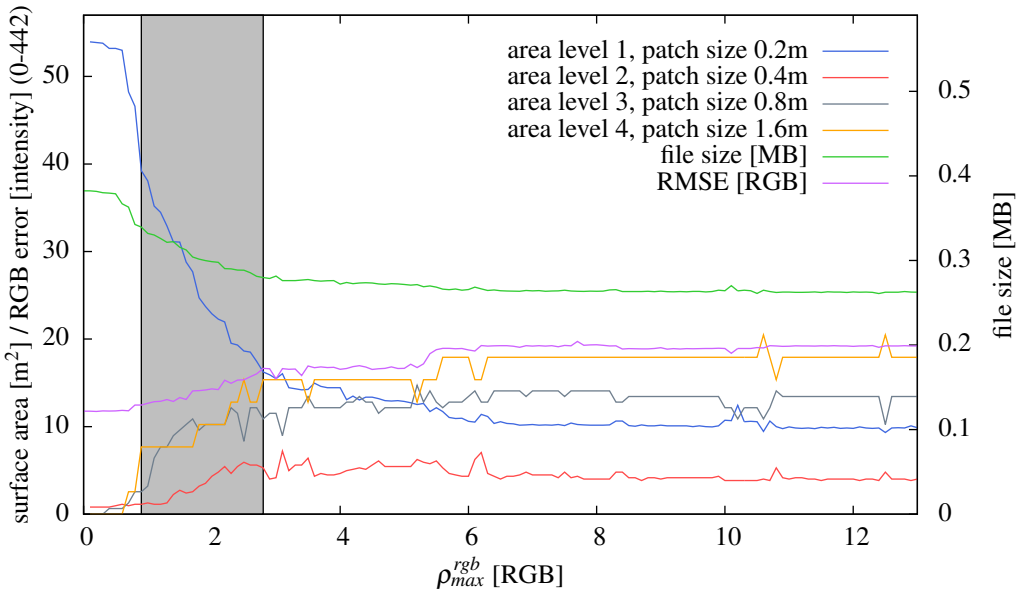
The interesting quantities for our models are accuracy, compactness and the time needed to compute a HSCSM. We will follow a similar evaluation scheme as described in Section 5.4. As measure of compactness we take the file size of the resulting models. As measure of accuracy we compute the root mean squared error (RMSE) between the input point clouds $\mathcal{P}$ and the point cloud reconstruction of our model $\mathcal{R}$. Obviously, this depends on the sensor noise and the discretization scheme we apply in our models. Still, the order of magnitude of the error is a good indicator of the accuracy. We compute the error for every point in $\mathcal{R}$ by searching for the nearest neighbor in $\mathcal{P}$ and vice-versa. In this way, the error measures inliers and outliers.

6.4.1 Influence of Maximum Standard Deviation

Crucial questions are how much we gain from introducing multiple levels with different patch sizes and if the maximum standard deviation is a good criterion to guide our decision what to model on a certain level. Therefore, we conducted a corresponding experiment on the RGBD scene shown in Figure 6.3. We chose to use 4 levels with patch sizes of $24/12/6/3$ cm and corresponding resolutions of $24/12/6/3$ mm. In the first setting, we varied the maximum allowed



(a)



(b)

Figure 6.4: This figure illustrates the impact of the maximum standard deviation parameters ρ_{max}^{depth} and ρ_{max}^{rgb} on our hierarchical model. The area covered for every level of the hierarchy and the resulting RMSE for the final model are plotted in **(a)**. For small values for ρ_{max}^{depth} our model is fully covered with the small patches of the lowest level. By increasing ρ_{max}^{depth} we shift the contribution in coverage to higher levels. This drastically reduces the number of patches needed to describe the surface. A similar evaluation for ρ_{max}^{rgb} is shown in **(b)**. Again, the area covered with larger patches at higher levels increases while relaxing this parameter. We marked the most interesting parameter ranges with a light gray box. In this area the file size is reduced by approximately 25%, while introducing only a moderately increasing error.

standard deviation for the depth channel ρ_{max}^{depth} for a fixed maximum standard deviation of the RGB channel $\rho_{max}^{rgb} = 100$. In the second setting, we used a fix $\rho_{max}^{depth} = 1.0$ and varied ρ_{max}^{rgb}. Figure 6.4 shows the resulting plots. The plot in 6.4(a) shows the area covered for every level of the hierarchy and the resulting RMSE for the corresponding model. For small ρ_{max}^{depth} the models are fully covered with small patches of the lowest level. By increasing ρ_{max}^{depth} the contribution in coverage is shifted to higher levels. This drastically reduces the number of patches needed to describe the surface and results in more compact models. The plot in 6.4(b) shows a similar evaluation for ρ_{max}^{rgb}. Again, the area covered with larger patches of higher levels increases while relaxing the parameter. We highlighted the most interesting parameter ranges with a light gray box. In this range, the file size is reduced around 25% while only moderately increasing the error. One interesting aspect is the difference in the area covered by the largest patches at level 4 in the two experimental settings. For $\rho_{max}^{rgb} > 4$ the plot in Figure 6.4(b) level 4 covers the majority of the surface while this is never the case for the full parameter range of ρ_{max}^{depth} the plot in Figure 6.4(a). At least for the considered scene the 3D structure constrains the possible patch sizes stronger than the texture.

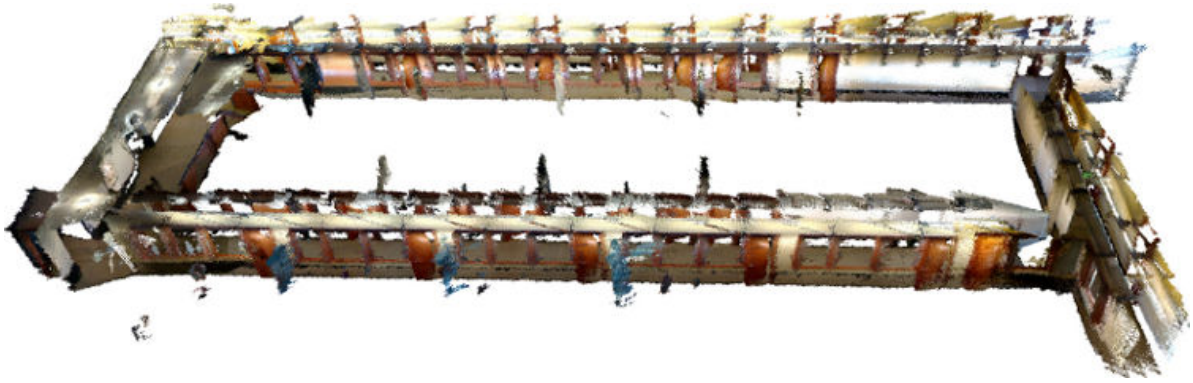
6.4.2 Comparison to Octomap and SCSM

In this section, we compare HSCSM to Octomap (Wurm et al., 2010) and SCSM (Chapter 5). Therefore we applied our method on three different data sets, the example scene illustrated in Figure 6.3, a SLAM solution[1] containing 707 registered RGBD scans and the publicly available fr1/room data set (Sturm et al., 2012), both shown in Figure 6.5. Table 6.1 gives an overview of the relevant statistics for Octomap, SCSM, and HSCSM. Note that we chose the minimum resolution according to the average distance between sensor and surfaces in the data set to avoid over-fitting to sensor noise. On the three data sets, our method never extracted patches on level 4 or higher. Therefore, we provide timings for the larger data sets only with 3 levels.

For the corridor SLAM data set, we applied our method to the accumulated point cloud and built a model with 35,824 surface patches on three hierarchy levels (251 / 801 / 34,772) with patch sizes of 80 cm, 40 cm and 20 cm and corresponding resolutions of 8 cm, 4 cm and 2 cm. The full process of creating the model took 19 min including the dictionary learning with a dictionary size of 1,000 for the depth channel and 3,500 for the RGB channel. The model created with our hierarchical method outperforms the Octomap in terms of accuracy and also visually as can be seen in Figure 6.6 (b) and (d). Regarding runtime, Octomap is fastest with less than a minute but introduces a higher error in both depth and RGB space. Compared to SCSM, the hierarchical model is 24% more compact and even more accurate in the depth channel. The higher accuracy is either the result of the larger dictionary used in the HSCSM compared to the dictionary size SCSM or the result of better surface patch positions. The hierarchical organization scheme allows us to store a larger dictionary and still produce more compact models but the hierarchical coding scheme itself cannot perform better than the non-hierarchical. The error in RGB space is slightly higher. This is due to the fact that encoding errors in larger patches have a bigger impact on the overall error calculation. Therefore we had to increase the dictionary sizes compared to the non-hierarchical method.

The fr1/room data set was captured in a cluttered office environment and is a challenging benchmark for our method. Clutter introduces high entropy and makes it harder to find similar structures. Figure 6.7 shows an example view for the raw point cloud, Octomap, SCSM and HSCSM. Again, our method outperforms Octomap in terms of accuracy and compactness,

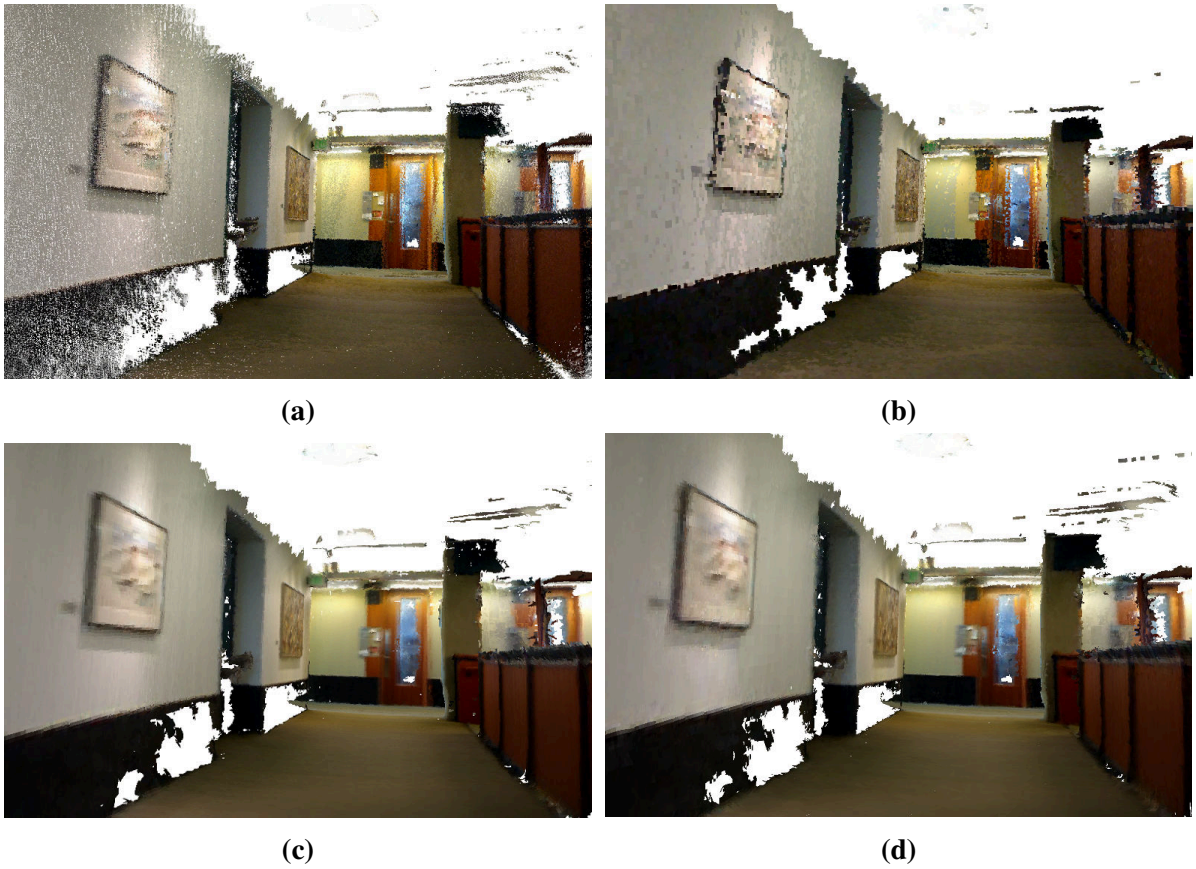
[1] Courtesy of Peter Henry



(a)

(b)

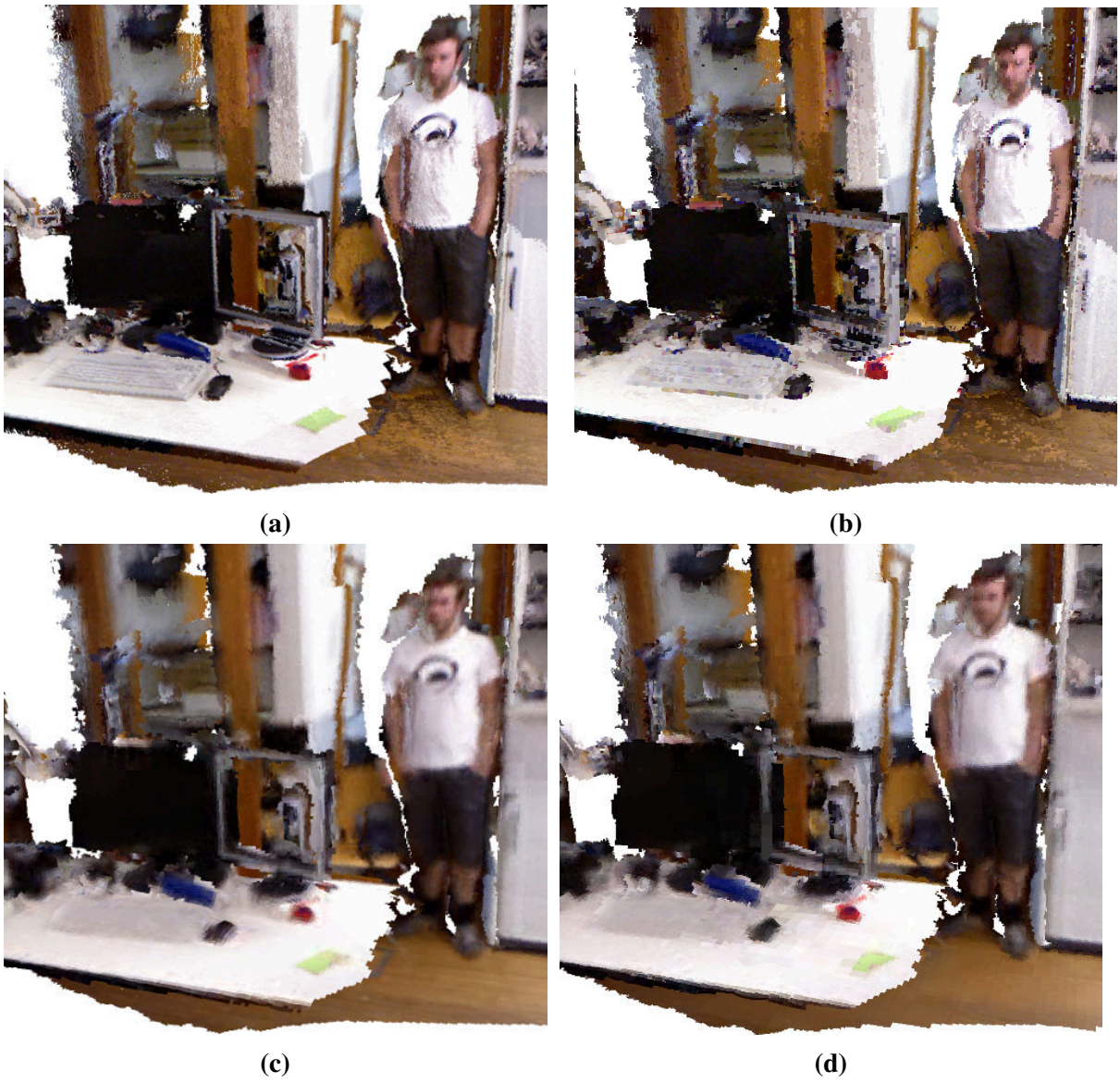
Figure 6.5: This figure shows the data sets used for comparison with Octomap, Sparsely Coded Surface Models (SCSM), and Hierarchical Sparsely Coded Surface Models (HSCSM). Picture **(a)** shows an overview of a RGBD data set acquired in a typical corridor environment and **(b)** shows the publicly available fr1/room (Sturm et al., 2012) data set.



(a) **(b)**

(c) **(d)**

Figure 6.6: Resulting model for the RGB Corridor data set, a representative part of the model as a raw point cloud **(a)**, an Octomap **(b)**, SCSM **(c)** and the hierarchical variant we propose in **(d)**. The Octomap occupancy grid has a file size of 44.49 MB with a voxel size of 2 cm, an RMSE of 0.016 m on the depth channel and an RMSE of 25.1 intensity value on the RGB channel. The model without hierarchy has a file size of 10.2 MB and stores a depth dictionary with 100 entries and a RGB dictionary with 500 entries. The RMSE is 0.017 m for the depth channel and 19.9 intensity value for the RGB channel. The hierarchical model has an RMSE of 0.013 m for the depth and of 21.7 intensity value for the RGB channel. In comparison, the hierarchical model has the lowest error in the depth data and an error in the RGB data that is between the Octomap and Sparsely Coded Surface Model errors. Organizing the data with the hierarchical model reduces the required storage from 10.2 MB to 7.8 MB.



(a) **(b)**

(c) **(d)**

Figure 6.7: Resulting model for the fr1/room data set, a representative part of the model as raw point cloud **(a)**, an Octomap **(b)**, SCSM **(c)** and the hierarchical variant propose in this paper. The Octomap has a resolution of 1 cm and a file size of 45 MB, an RMSE of 0.006 m on the depth channel and an RMSE of 39.6 intensity value on the RGB channel. The SCSM model has a file size of 8.1 MB and stores a depth dictionary with 500 entries and a RGB dictionary with 3500 entries. The RMSE is 0.005 m for the depth channel and 29.9 intensity value for the RGB channel. The hierarchical model has RMSE 0.005 m for the depth and 30.0 intensity value for the RGB channel. In comparison, the hierarchical model has the lowest error in the depth data and an error in the RGB data that is between the Octomap and Sparsely Coded Surface Model errors. Organizing the data in a hierarchical fashion enables us to store the data with 7.2 MB instead of 8.1 MB.

data set	Scene Figure 6.3		
method	Octomap	SCSM	HSCSM
dictionary size (D/RGB)	- / -	100 / 200	100 / 200
patch size	-	3 cm	24 / 12 / 6 / 3 cm
resolution	0.3 cm	0.3 cm	2.4 / 1.2 / 0.6 / 0.3 cm
input file size	4.7 MB		
resulting file size	918 KB	423.9 KB	343.6 KB
RMSE (D/RGB*)	0.0014 m / 8.2	0.0016 m / 14.3	0.0017 m / 15.2
time	0.25 s	7 s	8 s

data set	RGBD Corridor		
method	Octomap	SCSM	HSCSM
dictionary size (D/RGB)	- / -	100 / 500	1000 / 3000
patch size	-	0.2 m	0.8 / 0.4 / 0.2 m
resolution	0.02 m	0.02 m	0.08 / 0.04 / 0.02 m
input file size	3.35 GB		
resulting file size	44.5 MB	10.2 MB	7.8 MB
RMSE (D/RGB*)	0.016 m / 25.1	0.017 m / 19.9	0.013 m / 21.7
time	56 s	8 min	19 min

data set	fr1/room		
method	Octomap	SCSM	HSCSM
dictionary size (D/RGB)	- / -	500 / 3500	500 / 3500
patch size	-	0.05 m	0.2 / 0.1 / 0.05 m
resolution	0.01 m	0.01 m	0.04 / 0.02 / 0.01 m
input file size	2.7 GB		
resulting file size	45 MB	8.1 MB	7.2 MB
RMSE (D/RGB*)	0.006 m / 39.6	0.005 m / 29.9	0.005 m / 30.0
time	2 min	31 min	36 min

Table 6.1: Experimental evaluation for each data set and method. The dictionary size and RMSE errors are split into depth and RGB. The provided timings were measured on a standard Intel i7 CPU with 3 GHz. (*) Unit is intensity value.

while it requires more computation time. The model also looks smoother while being 84% smaller in size. In comparison to SCSM we gain 9% in terms of compactness while introducing a slightly higher error in RGB. In general, the gain in compactness is highly influenced by the amount of variations in structure and texture. In the worst case the full surface needs to be represented at the lowest level so that the resulting model performs similar to SCSM.

Overall, the gain in compactness compared to SCSM is between 9% and 24% on the data sets with a comparable accuracy. Since HSCSM tries to represent the full scene at three different scales, the run time is increased.

6.5 Related Work

The basic representations for textured 3D data are either colored point clouds, voxels or polygon meshes. Point clouds can store every measured surface observation as 3D point together with

a color. Therefore, all data can be accurately stored but since we need to store every single measurement including multiple observations of the same surface, this representation has high storage requirements.

A voxel representation is a 3D grid where each cell stores information like an occupancy or color value. The storage requirements for voxel grids depend on their resolution. For high resolutions a voxel grid has even higher storage requirements than point clouds since all grid cells have to be stored. For low resolutions, a voxel grid can be compactly stored but the grid structure leads to discretization artifacts. A more memory efficient variant of voxel grids are octrees but they also suffer from discretization artifacts. Octrees represent occupancy and implicitly free space in a hierarchical grid structure by dividing every voxel into eight octants. The advantage of octrees is that we only need to represent one voxel if all octants of this voxel have the same content. In this way, some parts of the scene are represented with big voxels and only parts that need a higher resolutions are represented with small voxels. An open source implementation for textured occupancy octrees called Octomap was presented by Wurm et al. (2010) and was used as baseline for our experimental evaluation. Kammerl et al. (2012) used octrees to compactly transmit differences in successive RGBD frames captured with a RGBD camera.

In the context of SLAM many recent GPU-driven systems utilize the Truncated Signed Distance Function (TSDF) (Curless and Levoy, 1996) as representation. The main advantages of this approach are the smoothness and the resolution of the estimated surface. However, this approach requires to maintain a cubic voxel grid in memory, which restricts the size of such a grid to fit into GPU memory. Recent extensions use either a moving TSDF volume representation (Whelan et al., 2013) and maintain data outside of the GPU or use local TSDF volumes or patches to represent only local aspects of the environment (Henry et al., 2013). Since TSDF volumes have a limited compactness and require ray casting for visualization, the volumes are typically converted into meshes for that purpose. Meshes, however, are a highly accurate representation but lack compactness and furthermore cannot be easily updated.

Recently, Ma et al. (2013) presented a planar simplification scheme for large meshes to drastically reduce the number of vertices on planar surfaces. In contrast to our work the quadtree-based triangulation scheme produces a mesh without overlap, while our method produces overlapping surfaces. On the other side it reduces only the structural information and does not exploit repetitive structures or textures. It furthermore is limited to planar surfaces.

In the previous chapter, we introduced a surface-patch-based representation that uses Sparse Coding to compactly describe the surface attributes in both, 3D and RGB space. Sparse Coding (Olshausen et al., 1997) is a machine learning technique used for signal approximation (Rubinstein et al., 2010), image denoising (Hyvarinen et al., 2001; Elad and Aharon, 2006) and for learning features (Bo et al., 2012; Yang et al., 2009). In this chapter, we presented a hierarchical extension that better adapts to the level of detail needed to accurately describe local surfaces. This extension leads to even more compact models. Additionally, we presented a novel interest point method driven by maximizing coverage and alignment to reduce the number of patches needed to represent the input data. Furthermore, we introduce a distance-based weighting scheme to compute weighted means for the surface descriptions. This has been applied successfully in the context of TSDF-based SLAM methods (Newcombe et al., 2011; Whelan et al., 2013; Henry et al., 2013). As a result, the represented surfaces suffer less from noisy far range measurements of RGBD cameras.

6.6 Conclusions

In this chapter, we presented an extension to Sparsely Coded Surface Models to construct textured 3D models using a hierarchy of local surface patches. The described method employs differently sized surface patches thus leading to more compact models. Furthermore, we described a greedy algorithm that uses a maximum allowed standard deviation for RGB and depth channel to guide the decision about what to model at which level of detail. Our method exploits similarities in structure and texture by using sparse coding to describe surface attributes at different scales. The required dictionaries are learned during the model creation process in an unsupervised fashion. Practical experiments carried out with data sets of different scale demonstrate that our method compares favorably with two state-of-the-art techniques. Besides learning compact and accurate surface models, some of the techniques described in the previous chapters can also be applied in the context of place recognition on 3D range data. In the next chapter, we will use a dictionary of features to efficiently search for similar places.

Chapter 7

Place Recognition

Recognizing previously visited places based on sensor data is an essential ability for mobile robots. It can be used for initializing a localization algorithm (global localization), loop-closure detection as part of a SLAM system or change detection in dynamic environments. Most related work on place recognition focuses on camera images and relatively few approaches try to solve this problem on 3D range data. In this chapter, we present a place recognition system that works with 3D range data and reliably detects similar structures. In contrast to Granström et al. (2009) or Magnusson et al. (2009), we estimate transformations between a query scan and possible matches in our database. This allows our method to perform geometric validation of possible matches to robustly avoid false positives. Furthermore, we introduce a bag-of-words of geometric features to speedup the search on large-scale datasets. This is highly related to the surface models and dictionary learning techniques described in Chapter 4 and Chapter 5. Instead of building a 3D model of the environment, as presented before, we describe every scan in the database dependent on an overcomplete dictionary. We exploit similarities in feature appearance, based on the dictionary, as a heuristic to guide our search for similar places in our database. Throughout extensive experiments on publicly available indoor and outdoor datasets we evaluate our method and compare favorable to other state-of-the-art approaches in terms of the recall rate.

♦ ♦ ♦ ♦ ♦ ♦ ♦

7.1 Introduction

In place recognition, we would like to detect that a robot is currently in an area it has previously visited based on its sensor observations. This is a key ability to perform global localization with local sensors if GPS is not available (for instance indoors) or not accurate enough to successfully localize. In the context of SLAM, place recognition is a key component to find loop closures and correct accumulated drift in order to build large-scale maps. Both examples are essential for a mobile robot to successfully perform navigation tasks. Most published place recognition systems rely either on 2D laser data or use cameras and only few use 3D range data. In the

context of 3D range data it seems also desirable to estimate a relative transformation between an observation and the best database matches to perform geometric validation.

In this chapter we present a place recognition system based on 3D range data. We represent individual scans as range images and use point features to estimate a relative transformation between scan pairs. Instead of matching features for every scan pair, we learn a reference dictionary of 3D point features, using techniques like the ones we described in chapters 4 and 5. Based on this dictionary we can pre-compute feature appearances for the entire scan database. In the literature this is referred to as bag-of-words. To compare a new scan with our database, we only have to compute the feature appearances for this scan according to the reference dictionary and search for database entries with similar appearance. This informed search scheme saves run time and reduces the number of possible candidates for estimating relative transformations. Such estimated transformations can be used as relations in a SLAM pose graph as shown in Figure 7.1. The black nodes correspond to the robots trajectory based on the odometry information of the robot and loop closures. The resulting relative transformations of the place recognition are shown as blue loop closure constraints. Together with a graph based optimization framework, similar to what we used in chapters 3 and 8, a consistent map can be computed.

We evaluated the proposed method on ground robots as well as on flying vehicles. To benchmark our approach, we used publicly available 3D datasets. In this way a comparison with other approaches can be easily performed. Furthermore, we recorded a dataset with a flying vehicle and made the data publicly available.

7.2 Our Approach

The input to our algorithm is a scan database, a pre-computed representative dictionary of appearing features and a query scan that should be matched to the database. As result, the system returns a set of scans that are potential matches, together with a set of corresponding scores and relative transformations between query scan and matching scan. The score reflects the quality of the match.

Given a database of 3D range measurements $\mathcal{DB}$ and a query scan z^*, our approach calculates a set of candidate tuples $\mathcal{C}(z^*) = (\langle z_1, T_1, s_1 \rangle, \ldots, \langle z_n, T_n, s_n \rangle)$. Each tuple $\langle z_i, T_i, s_i \rangle$ consists of a scan $z_i \in \mathcal{DB}, i \in \{1, \ldots, n\}$, which is potentially similar to the query scan. The second element is the estimated relative transformation T_i between z^* and z_i and the third element is a score s_i, which reflects the confidence of the given match. Our algorithm computes $\mathcal{C}(z^*)$ as follows:

1. We calculate a reference dictionary based on the extracted features from a 3D training dataset $\mathcal{DB}'$. Usually this bag-of-words has to be computed once on a representative dataset and can be used for several datasets.

2. Pre-compute feature appearances based on the dictionary for the full scan database $\mathcal{DB}$.

3. Compute a histogram of feature appearances for the query scan z^* based on the reference dictionary and sort database scans according to the similarity in feature appearances. The resulting ordered list will be $\hat{\mathcal{DB}}(z^*) = \langle \hat{z}_1, \ldots, \hat{z}_{|\mathcal{DB}|} \rangle$.

4. Calculate candidate transformations for possible pairs $\langle z^*, \hat{z}_k \rangle, \hat{z}_k \in \hat{D}(z^*)$, starting from the best ranked pair. Such candidate transformations are estimated via point feature

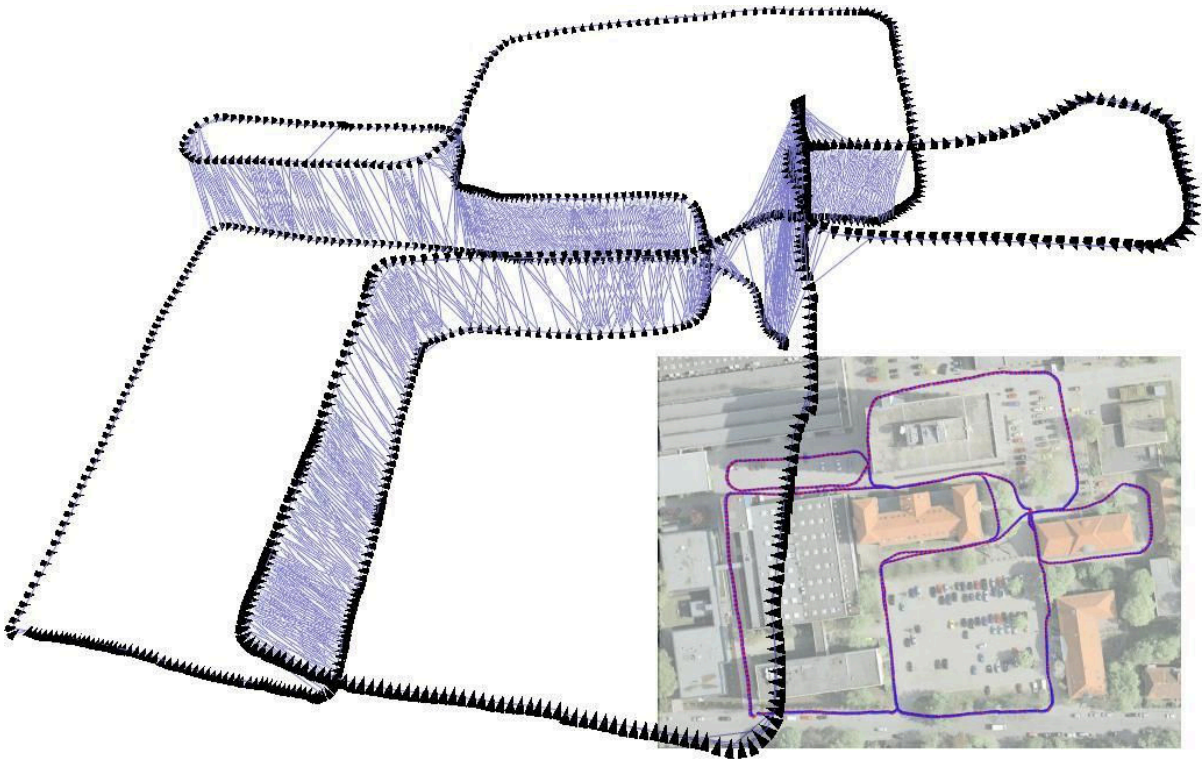


Figure 7.1: This figure shows the resulting pose graph including loop closure relations generated by our place recognition system. The black nodes correspond to the trajectory the robot had taken in the Hanover2 dataset. The found loop closures connecting similar poses are shown as blue lines. To make the loop closures visible, we varied the height of the nodes according to their node index starting with node 0 at ground level. An aerial image from Google Earth (© Google) with the robots corrected trajectory can be seen in the bottom right image. More details for the dataset and our results can be found in the experimental section.

matching between extracted features from z^* and $\hat{z}_k$. Note that this procedure typically results in many transformation candidates for each pair.

5. We score each possible transformation based on a scheme described in section 7.2.4. We take the highest scored transformation T_k and if the score is above a threshold we add a candidate $\langle \hat{z}_k, T_k, s_k \rangle$ to the result set.

The appearance based ordering allows us to perform an informed search. In this way we can constraint the maximum search time and still get good results. Since appearance based similarity highly depends on the overlap between two scans and might suffer from occlusions, we do not ignore scan pairs that have a low similarity as long as we still have time left for our search. Therefore, we repeat the last two steps either until we reach the maximum search time or $k = |D|$. Without a time constraint all scans in $\mathcal{DB}$ are checked.

Instead of working with 3D point clouds as the underlying representation, we chose to use range images (see Figure 7.2) to better represent knowledge about maximum range readings and scene parts that were not observed. A range image can capture the range information captured from a single viewpoint. Representing range data of a sensor moving during data acquisition can lead to data loss because of occlusions. In the following, we will give a detailed description of the individual parts of our method.

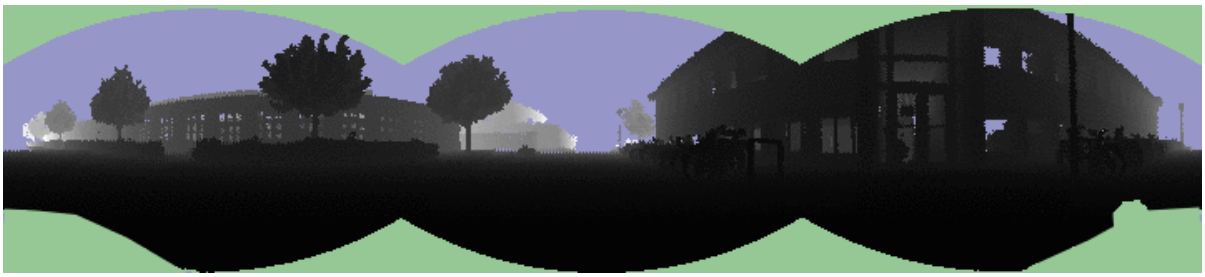


Figure 7.2: This figure shows an example range image with 360° field of view. The data were acquired with a 2D laser range finder on a pan-tilt unit with three sweeps at different orientations and is part of the FreiburgCampus360_3D dataset. A pixel position corresponds to the spherical coordinates of a point. The range is represented with a grey value where dark pixels are close to the observer and brighter pixels are further away. Pixels with maximum ranges have blue color and unobserved pixels have green color.

7.2.1 Feature Extraction

In order to build a dictionary and perform feature matching for transformation calculation we have to extract features or surface descriptions on non-textured 3D range images. We could easily apply the techniques described in Chapter 4 or Chapter 5. For the method presented in this chapter, we chose to use NARFs (Normal-Aligned Radial Features) Steder et al. (2011b). The main reason is that the NARF extraction is faster compared to surface primitives or sparse coded surface descriptions. We will later on see that the time needed to extract features plays a major role in the system timings. Furthermore, NARFs tend to perform well on noisy data, bring their own robust keypoints, and have already been successfully applied for object detection tasks (Steder et al., 2011b) on comparable 3D data.

Before we can extract NARFs, we need to select three parameters, namely the size of the feature vector, the support size and the maximum number of features we would like to extract. The size of the feature vector influences the resolution of the feature, whereas the support size corresponds to the radius of the region the feature will describe. Throughout our experiments we chose a descriptor size of 36. In our experiments the appearance based bag-of-words similarity did work best with a high number of features describing relatively small parts of the environment. Therefore, we set the maximum number of features we extract in a scan to 2000 and the support size to $1/10$ of the average range in the database $\mathcal{DB}$. In this way we extract features that describe smaller regions for indoor datasets and larger regions in outdoor datasets.

For calculating possible transformations via feature matching we are interested in a smaller number of more distinctive features. Therefore, we extract 200 features with a support size of $1/5$ of the average range in our dataset. The intuition behind this is that features with small support sizes are more influenced by noise and therefore less distinctive. On the other side, features with large support sizes are more expensive to compute and less robust to occlusions and missing data. The above mentioned values seemed to provide a good tradeoff between the desired properties and led to good results.

There are several ways to define a distance measure for feature descriptors, like Manhattan distance or Euclidean distance. For our experiments we chose the Manhattan distance as a measure to describe the similarity between feature vectors (*descriptor distance*). NARFs can be extracted either rotationally invariant or without invariance. A rotationally invariant feature of a top left corner of a cube would look like a feature of a top right corner of a cube and thus make the feature matching more difficult since we would prefer to match top left corners only with top left corners. This makes sense in our context where a wheeled robot has only minor variations in roll and pitch during the capturing of 3D range data. This is also applicable to flying vehicles

or other robots that are equipped with an IMU. Furthermore, this reduces the amount of possible false feature associations and saves computation time. A detailed comparison between NARFs with and without rotational invariance is provided in Section 7.3.2.

7.2.2 Appearance Based Similarity Ordering

To perform an informed search in our database $\mathcal{DB}$, we use feature appearances to compute a measure of similarity and search for candidates on the most similar database entries first. For describing the feature appearances, we learn a small representative set of features (bag-of-words) to have a common dictionary to refer to similar features. We learn such a dictionary on sample feature descriptors from a training dataset $\mathcal{DB}'$. For every 3D scan in this dataset we compute 2000 NARFs, leading to $n \cdot 2000$ feature descriptors of size 36. In chapters 4 and 5 we described different dictionary learning techniques: BIC driven clustering and K-SVD. In principle we could apply any of these techniques. We chose to use k-means, because the dictionary learning time was shortest compared to the other candidates. K-means is a special case of K-SVD, with only one dictionary reference and a fixed scaling factor of 1. Therefore, we clustered the reference dictionary with k-means setting $k = 200$, leading to 200 reference feature vectors from the clusters centroids. To describe a scan $z_i \in \mathcal{DB}$ we have to find the most similar word in the dictionary for every extracted feature. We use the Euclidean distance as the measure of similarity and assign every feature to the closest word in our dictionary. To express the word appearances, we compute a histogram H_i, which has a value for each dictionary entry. Such a value corresponds to the number of features that are assigned to a specific dictionary entry for a scan z_i.

We define the similarity between a query scan z^* and a scan z_i as the Euclidean distance between their appearance histograms $\|H^* - H_i\|_2$. Since the histograms for the database can be pre-computed, we just have to compute H^* and the distances to the stored histograms for each scan. Based on this distances we create an ordered list $\hat{\mathcal{DB}}(z^*)$ with the most similar scan (smallest distance) as the first item. In the next step we describe how we compute a set of candidate transformations between two scans.

7.2.3 Feature Matching and Candidate Transformations

Since each NARF has a well defined 3D Transformation we need only one feature correspondence to compute a relative six degree of freedom transformation between two scans. Therefore, we search for every feature in z^* corresponding features in z_i. We use the distance of the feature descriptors as measure of similarity and skip all feature pairings with a descriptor distance over a threshold. Again we rank the feature pairs based on their descriptor distance and start to evaluate the transformations for the best matches. The evaluation procedure, in which we compute a score for each transformation, is described in Section 7.2.4.

Throughout our experiments we considered only the best 2000 feature pairs and their transformations. For the rotationally invariant NARF setting we evaluated up to 5000 transformations, since we get more feature matches due to the additional degree of freedom

7.2.4 Scoring of Candidate Transformations

Given a candidate pair $\langle z^*, \hat{z}_k \rangle, \hat{z}_k \in \hat{D}(z^*)$ the feature matching returns a list of relative transformations $\hat{T}_k = \{\hat{T}_{k_1}, \ldots, \hat{T}_{k_n}\}$. We would like to determine the best transformation for each candidate pair. Therefore, we compute a score or likelihood for each candidate transformation

$\hat{T} \in \hat{T}_k$ per candidate pair and chose the best scored transformation. This score reflects the likelihood of a transformation based on our sensor model. Since we use 3D data, we can evaluate the quality of a candidate transformation point wise, assuming mutual independence.

To speedup the process we first select a subset of our query scan z^* as a set of *validation points* $\mathcal{V}$. Of course the described procedure could also be done with a set $\mathcal{V}$ containing all points in z^*. As first step we apply the candidate transformation $\hat{T} \in \hat{T}_k$ to all validations points $v \in \mathcal{V}$, which are in the coordinate frame of z^* to transform them into the coordinate frame of $\hat{z}_k$, resulting in $v' \in \mathcal{V}'$. Now we can re-project all v' in the range image of $\hat{z}_k$, compute the pixel location $v'(x,y)$, and compare the range of that pixel $p_k(x,y) \in \hat{z}_k$ with the range of v'. Based on this comparison we calculate a score $s_{\hat{T}}(v') \in [0,1]$ and a weight $w_{\hat{T}}(v') > 0$, which tells us how good the range r' of projected point v' explains the observed $r_k(x,y)$. The score reflects the re-projection error and is normalized between $[0,1]$. The weight comes out of our sensor model and enables us to weight the re-projection error. In this way, we can better deal with occlusions and other typical error cases based on the sensor characteristics. The overall likelihood for a candidate transformation $s(\hat{T})$ can be calculated with the following equation:

$$s(\hat{T}) = \frac{\sum_{\forall p \in P} w_{\hat{T}}(p) \cdot s_{\hat{T}}(p)}{\sum_{\forall p \in P} w_{\hat{T}}(p)}. \tag{7.1}$$

We define the difference between expected and measured range as $\Delta r = r_k(x,y) - r'$. For a meaningful interpretation of Δr we have to consider the physical sensor capabilities. A laser scanner emits a light pulse at a certain direction and the range is calculated by measuring the time until the reflected light returned along the line of sight. Each pixel in a range image stores the range measured with one beam. A more detailed discussion of the laser sensor capabilities can be found in Section 3.3.1. We consider the following different cases for Δr:

1. The observation fits the expectation, i.e., $|\Delta r| \approx 0$. To account for typical noise in the sensor observation we define a maximum typical sensor error $\Delta r_{\max} > 0$ and check for $|\Delta r| < \Delta r_{\max}$ to detect this case. We calculate the resulting score with $s_{\hat{T}}(p) = 1 - \frac{|\Delta r|}{\Delta r_{\max}}$ and set the corresponding weight to $w_{\hat{T}}(p) = 1$. This score function represents a triangular distribution. The most obvious choice would have been a Gaussian distribution. Since we have to compute this for many pixels and many candidate transformations, we favored the triangular distribution to save computation time. For the following cases we set the score to $s_{\hat{T}}(p) = 0$. In this way only the associated weight $w_{\hat{T}}(p)$ reflects the quality of the candidate transformation $\hat{T}$.

2. Given we observe a range larger than the expected range, i.e., $\Delta r > \Delta r_{\max}$. In this case we see through a point v' and observed something behind it. This could have been caused either by an almost transparent object or by a dynamic obstacle, but in most cases this is caused by a wrong transformation. To penalize the overall score of the solution we set a high weight of $w_T(p) = w^{\text{st}} \geq 1$ for seeing through an obstacle.

3. The range of the observation is shorter than the expected range, i.e., $\Delta r < -\Delta r_{\max}$. This leads to two possibilities:

   (a) The backward projection $\hat{T}^{-1} \cdot v_k(x,y)$ is visible in z^*. Therefore, the occlusion comes from an already known obstacle and seems to be consistent. In this case we give a low weight $w_{\hat{T}}(p) = w^{\text{ko}} \leq 1$ leading to a small penalty for known obstacles. In this way we can still get high scores even if two scans have only a small overlap.

(b) The backward projection $\hat{T}^{-1} \cdot v_k(x,y)$ is not visible in z^*. This means that both projection directions report an inconsistency. A dynamic obstacle, similar to what we described in case 2, could have caused this. Again, in most cases this is a strong indicator for a wrong transformation. Therefore, we introduce a high weight $w_{\hat{T}}(v) = w^{\text{uo}} \geq 1$ for unknown obstacles.

4. The point $v_k(x,y)$ corresponds to an unobserved pixel in $\hat{z}_k$. In most cases this comes from the fact that v' could not be observed because it is outside of the scan. This is weighted similar to 3(a).

5. The range measurement $r_k(x,y)$ is a maximum range reading and no valid range was measured in the corresponding pixel in $\hat{z}_k$. Again we have to consider two possibilities for the expected range r of v in z^*:

   (a) The range of the projected point is shorter than the range in the original scan, i.e., $r' \leq r$. This clearly speaks against a maximum range reading of v', since we could already observe the corresponding point at a larger distance in z^*. Therefore, we handle this case similar to 2.

   (b) The range of point v is larger after the projection in scan $\hat{z}_k$, i.e., $r' > r$. In this scenario it could be a likely explanation that the point corresponding to v is too far away for a proper range reading. In such cases with give a medium weight of $w_{\hat{T}}(p) = w^{\text{fr}} \geq 1$ for far ranges.

Even small errors in correct transformations have a high impact on the projective correspondence search. Especially around surface borders it is possible to assign a background pixel to a point lying on an object, resulting in a bad score. Therefore, we perform a local search in pixel space around the projected target pixel $v_k(x,y)$ and evaluate all possible assignments for the query point v'. Throughout our experiments we searched in a radius of 3 pixels. For the overall scan score we selected the best score out of the scores of all possible correspondences.

So far we just assumed that there are validation points. In the following we will explain how we compute a set of validations points $\mathcal{V}$. We could easily select all scan points as validation points, but this would be rather computationally expensive. Therefore, we select only a subset of the scan z^*. We could either use a random subset or do the validation in a coarse-to-fine fashion, but not all points are equally important to detect inconsistencies. Uniform surfaces like walls or the floor can be aligned easily without providing substantial information about the quality of the match, resulting in a high score. Whereas a coarse-to-fine strategy would result in a uniform 2D distribution we are interested in a uniform 3D resolution. In this way we avoid problems with the typical range dependent resolution of 3D scans. To cover the structural informative parts of a scan we use the NARF key-points $\hat{\mathcal{V}}$, which we already extracted, as initial validation point set. In the next step we perform a spatial sub-sampling by randomly selecting an initial point from $\hat{\mathcal{V}}$ and add it to the final validation point set $\mathcal{V}$. Then we iteratively add the point that has the largest distance to all points in $\mathcal{V}$ until we reach a maximum number of points. Throughout our experiments we used a maximum of 200 validation points.

In this way, we generate a sub-sampled ordered list $\hat{V} = \langle v_0, \cdots, v_j \rangle$ which has almost equidistant points out of $V = \langle v_0, \cdots, v_{|V|} \rangle$ for all values of j. Therefore, we already cover the full scene after evaluation of a minimum set of validation points at a coarse resolution and getting a higher resolution the more validation points we consider. Since we evaluate the score for a candidate transformation in the order of the given validation points, we have already a rough approximation of the final score after a certain minimum number of evaluation steps.

We can utilize this to reduce run time for obviously bad transformation candidates and stop the evaluation after a minimum number of steps if the score is below a lower bound.

Of course we can calculate the score $s(\hat{T})$ for a candidate transformation $\hat{T}$ either between z^* and $\hat{z}_k$ or in the other direction by applying $\hat{T}^{-1}$. Due to the projective data association this might lead to different results for both cases. Therefore, we compute both directions and take the minimum. This leads to the scoring function $s'(\hat{T}) = \min(s(\hat{T}), s(\hat{T}^{-1}))$ for a scan pair $\langle z^*, \hat{z}_k \rangle$.

7.2.5 Self-Similarity

To successfully estimate a transformation on 3D range data, we have to rely on distinctive geometric features. In many human made environments we have few distinctive features like in long corridors. In such cases two scans of the same corridor, taken meters apart, might get a high score because the scans look very similar. To detect such situations we compute a self-similarity score for all scans. This is done by matching the scan z to itself as described before and reject all transformation candidates that have a transformation matrix close to the identity matrix. Then we select the best remaining transformation and use the corresponding score as measure of self-similarity $self(z)$. To make our scoring function more robust to self-similarity we extend the final scoring for a transformation between z^* and $\hat{z}_k$ as follows: $s^*(\hat{T}) = (1 - (self(z^*) + self(\hat{z}_k))/2) \cdot s'(\hat{T})$. Once we have computed all candidate transformation scores $\hat{T} \in \hat{T}_k$ for a scan pair, we select the transformation with the highest score. Given this score is above a threshold, we consider it as a valid match. Based on this match we could initialize a localization algorithm or in the context of SLAM consider our match as a potential loop closure.

7.2.6 Implementation details

To further improve results we try to optimize the best transformation candidates for a scan pair $\langle z^*, \hat{z}_k \rangle$ with ICP and recompute the scores. In this way we can reduce the impact of small alignment errors. It might happen that the correct transformation is not best scored due to a small misalignment. Therefore, we have to almost consider the full candidate set. Of course we can neglect most already low scored candidates. Furthermore, we can reduce the number of candidates by clustering the transformations and take only the best scored candidate for a group of very similar transformations. As the last step, we perform ICP on the remaining candidates and update transformation and score accordingly. To speed up ICP, we use only correspondences between one validation point set and the nearest neighbor in the other scan.

7.3 Experiments

To evaluate our method we chose several publicly available 3D range datasets. In this way a comparison to other methods can be performed easily. Furthermore, we selected a set of two indoor and two outdoor datasets to provide a sophisticated evaluation. Note that the 2D laser scanners were actuated in different fashions to capture 3D data throughout all datasets. In the following we will give a brief description of the chosen datasets and their challenges.

7.3.1 Datasets

We evaluated our method on the following datasets:

- As first dataset we used **AASS-loop** (Magnusson, 2007). The dataset was collected in an indoor environment and contains highly challenging ambiguous areas such as long corridors. Note that this dataset was also used in related work (Granström et al., 2009; Magnusson et al., 2009).

- We recorded a dataset with a flying **Quadrotor** robot which was equipped with a static 2D laser scanner (Grzonka et al., 2009). The 3D scans were recorded in an indoor office environment by maintaining the pose and varying altitude. Due to dynamics of the platform the challenges of this dataset lie in the amount of noise in the 3D scans and the high self-similarity, since the scans were recorded in a corridor environment.

- The **FreiburgCampus360_3D** dataset (Steder, 2009) was recorded in an outdoor campus environment and was also used in related work (Steder et al., 2010a). The 3D scans show 360° and have a high resolution. The main challenge here is that the scans were recorded relatively far apart.

- We used **Hanover2** (Wulf, 2007) as second outdoor dataset. Again this dataset was used in previous work (Granström et al., 2009; Magnusson et al., 2009; Steder et al., 2010a), which makes a comparison to our results straightforward. The 3D range scans in this dataset are relatively sparse due to the chosen rotational velocity of the laser scanner during data recoding.

To generate ground truth we used the provided odometry and a scan matcher to create a SLAM pose graph. Loop closures were selected manually and refined by a scan matcher. As final step, the pose graph was optimized with g^2o (Kümmerle et al., 2011). The resulting trajectories were the baseline for our evaluation. We used the distance between scan poses to decide whether a result is a false/true positive or a false/true negative. In case of the quadrotor we used the pose estimates of the internal navigation system as described in Grzonka et al. (2009) as ground truth. Figures 7.3 and 7.4 provide an overview of the used datasets.

Throughout our experiments we used the following weights for the scoring function as described in Section 7.2.4: $w^{\text{st}} = 25$, $w^{\text{ko}} = 0.5$, $w^{\text{uo}} = 15$, and $w^{\text{fr}} = 5$.

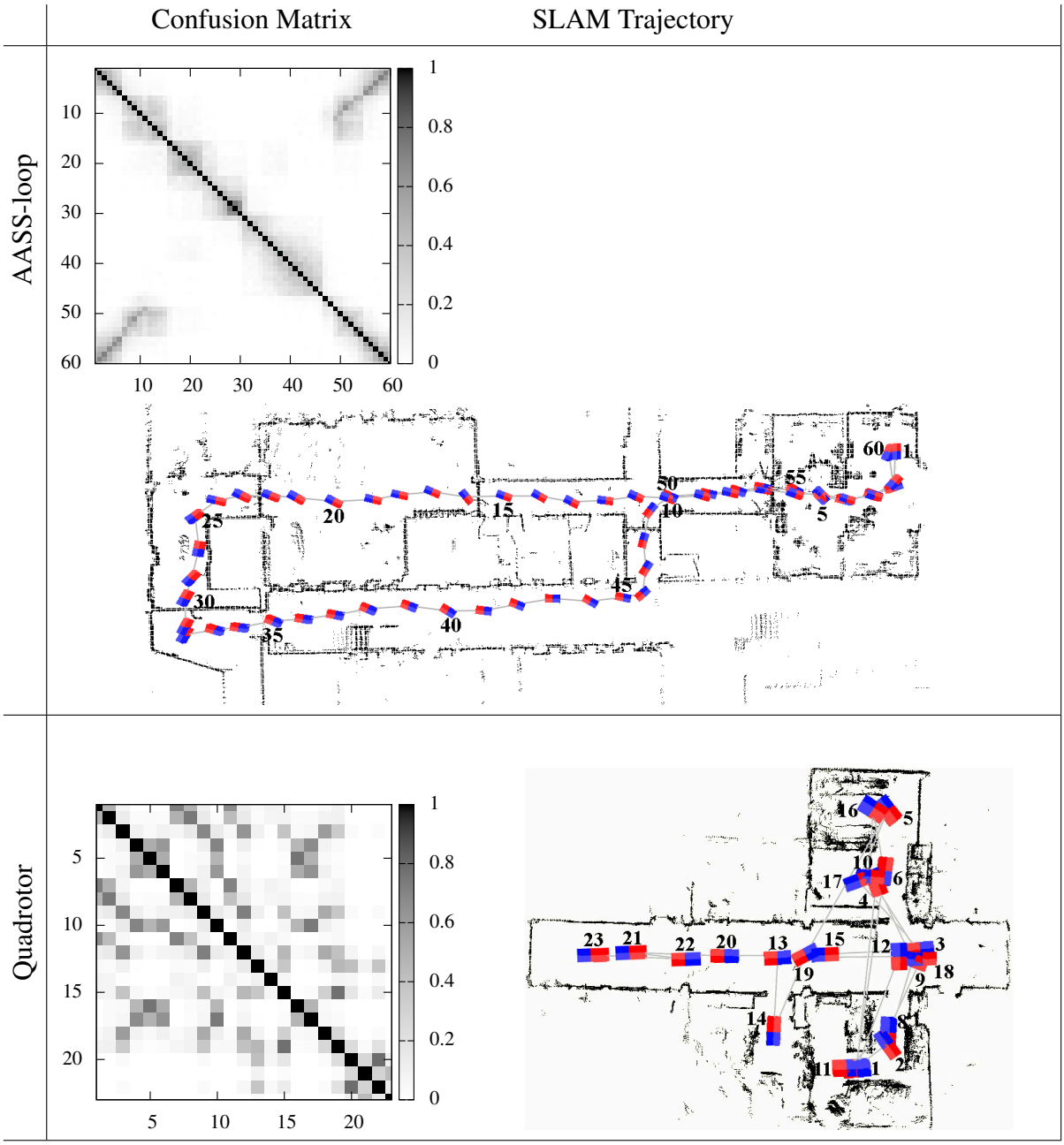
7.3.2 Evaluation and Comparison

We applied our method to all four datasets and determined the corresponding confusion matrices by calculating the score of the best transformation for all possible scan pairs. The results for the individual datasets are shown in the plots 7.5(b), 7.6(b), 7.7(b), and 7.8(b). The darker entries in the confusion matrix correspond to similar locations and the dark entries that are not around the main diagonal correspond to found loop closures.

To check if a found match is a false positive, we compared the computed ground truth transformation with the resulting estimated transformation and decide based on a specific error threshold. Note that most related work (Magnusson et al., 2009; Granström and Schön, 2010) does not calculate transformations and just checks the distance between scans to decide whether this is a valid match or not.

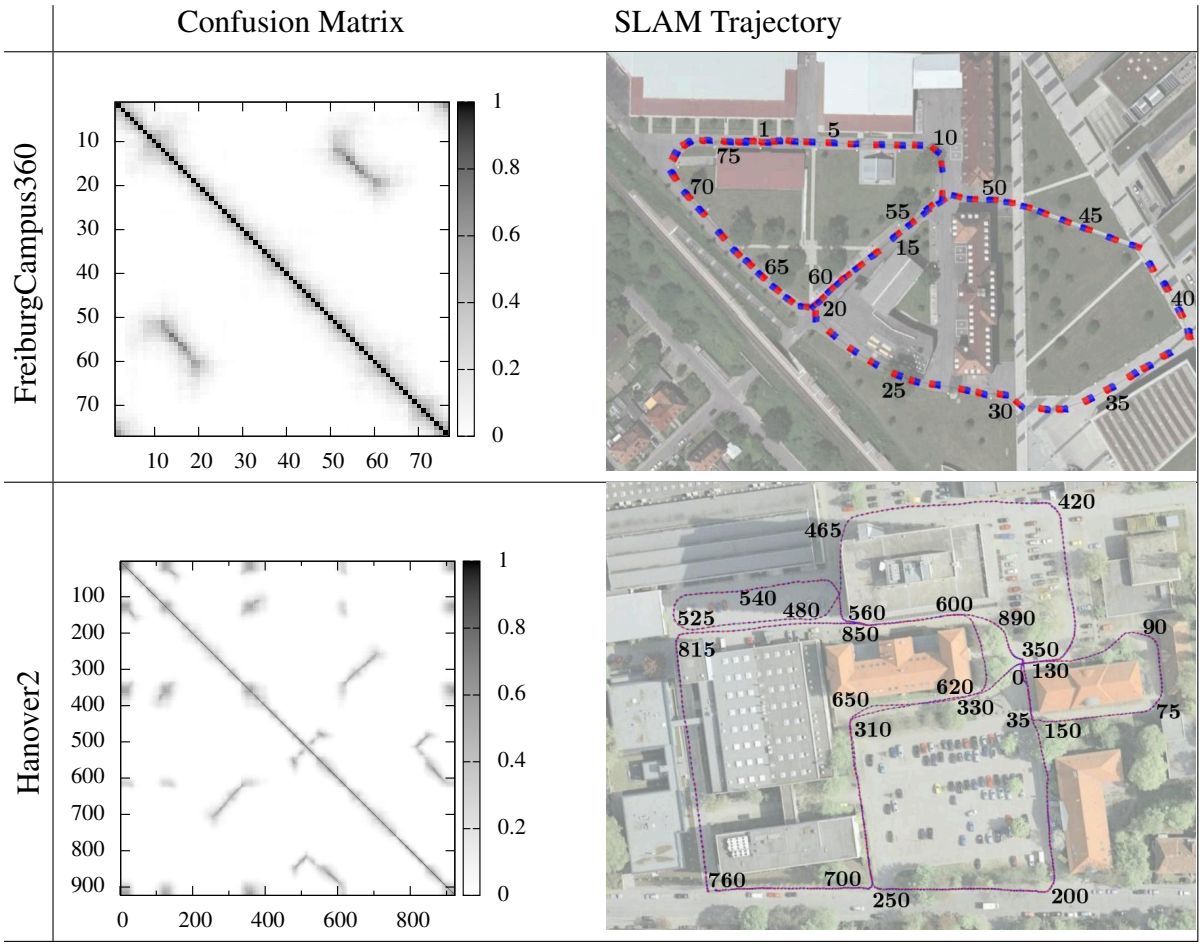
The resulting number of true positives, false negatives and the recall rates are plotted as a function of the distance between scans for each dataset shown in Figures 7.5(c), 7.6(c), 7.7(c), and 7.8(c). In all cases we used the same minimum acceptance threshold and did not find false positives.

Additionally we evaluated the dependence on the acceptance threshold. The plots in Figures 7.5(d), 7.6(d), 7.7(d), and 7.8(d) show the number of false positives and the corresponding



	stop & go	scans	points	far ranges	res.	traject. length [m]	aver. dist. [m]	mean range [m]	max. range [m]
AASS	yes	60	80873	n/a	0.7°	111.4	1.89	2.81	67.1
Quadrotor	no	23	171608	95515	1.0°	79.1	3.60	1.90	6.1

Figure 7.3: Top: Ground truth confusion matrices and SLAM trajectories of the indoor datasets. The SLAM trajectory is plotted together with a 2D grid map obtained by projecting wall points to 2D. The entries in the confusion matrices correspond to the overlap for each scan pair. White stands for no overlap and black for full overlap. The table gives an overview over the relevant quantities of the indoor datasets. **stop&go**: whether the robot was stationary while capturing scans, **scans**: number of scans, **points**: average number of points per scan, **far ranges**: average number of far range readings per scan, **res.**: angular resolution of range images, **traject. length**: length of trajectory, **aver. dist.**: average distance between consecutive scans, **mean range**: mean of the measured ranges, **max. range**: maximum measured range.



	stop & go	scans	points	far ranges	res.	traject. length [m]	aver. dist. [m]	mean range [m]	max. range [m]
Freiburg	yes	77	155562	56451	0.45°	723.0	9.51	8.90	50.0
Hanover	no	923	12959	3622	1.3°	1247.8	1.35	7.18	29.0

Figure 7.4: Top: Ground truth confusion matrices and SLAM trajectories of the outdoor datasets. The SLAM trajectory is overlaid on an aerial image taken from Google Earth. The entries in the confusion matrices correspond to the overlap for each scan pair. White stands for no overlap and black for full overlap. The table gives an overview over the relevant quantities of the outdoor datasets. **stop&go**: whether the robot was stationary while capturing scans, **scans**: number of scans, **points**: average number of points per scan, **far ranges**: average number of far range readings per scan, **res.**: angular resolution of range images, **traject. length**: length of trajectory, **aver. dist.**: average distance between consecutive scans, **mean range**: mean of the measured ranges, **max. range**: maximum measured range.

dataset	feature extraction [ms]	candidate matching [ms]	time per scan pair [ms]	BoW feature histogram
AASS-loop	881	585	10	894
Quadrotor	305	102	4	276
FreiburgCampus360_3D	1107	838	11	730
Hanover2	316	4132	4	246

Table 7.1: Average timings per scan for our method on the different datasets. All timings were measured on an Intel i7 with four cores.

recall rate per dataset. Note that the recall rate was calculated for a specific maximum distance between scans. For the AASS-loop plot in Figure 7.5(d) we used a maximum distance of 1.0 m to consider two scans a true positive, similar to the related work (Magnusson et al., 2009; Granström and Schön, 2010). The lowest acceptance threshold that did not result in false positives is 0.09. Above this threshold the recall rate was 0.938. The corresponding values for Figure 7.6(d) (Quadrotor) are 2.0 m / 0.25 / 0.75, for Figure 7.7(d) (FreiburgCampus360_3D) 10.0 m / 0.05 / 0.958, and for Figure 7.8(d) (Hanover2) 3.0 m / 0.19 / 0.925.

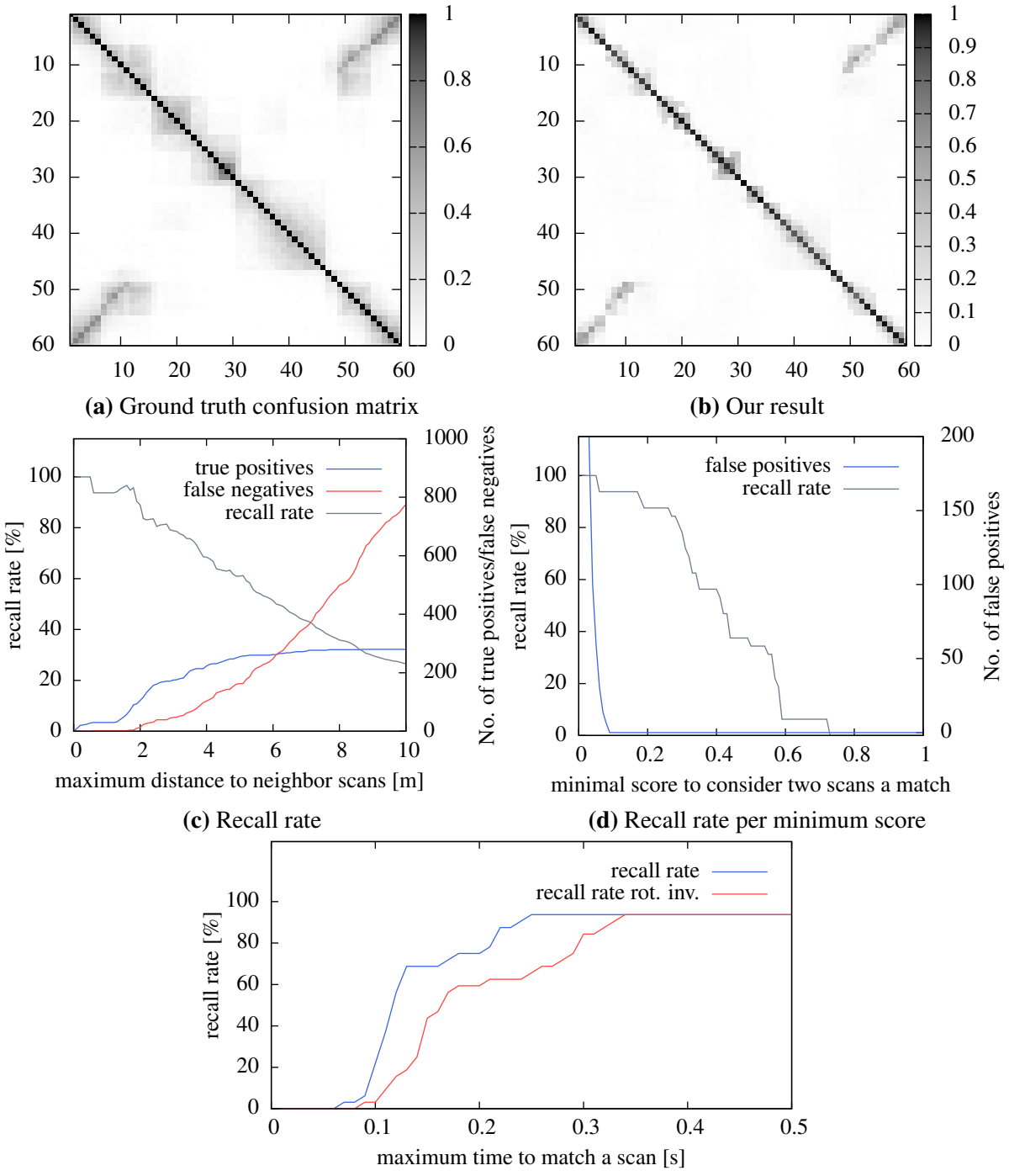
In the evaluation we did not consider the trivial cases of the diagonal elements of the confusion matrices, which correspond to matching a scan to itself with an identity transformation matrix. Granström and Schön (2010) did also not consider the full confusion matrix because they split the dataset into training and test set to apply boosting. In their experimental evaluation they report recall rates of 0.53 ± 0.14 (min 0, max 0.88) for AASS-loop and 0.63 ± 0.6 (min 0.28, max 0.76) for Hanover2.

Also Magnusson et al. (2009) did not consider all scan pairs for their evaluation, since they focus on SLAM and are only interested in the off-diagonal part of the matrix. Therefore, they did only check scan pairs that were at least 30 poses apart. In the experimental evaluation they report recall rates of 0.7 for AASS and 0.47 for Hanover2 at 100% precision. For comparison, in a similar setup our method had a recall rate of 1 with an acceptance threshold of 0.08 on the AASS dataset and a recall rate of 0.911 with an acceptance threshold of 0.19 on the Hanover2 dataset.

7.3.3 Timings and Influence of the Appearance Based Similarity

So far we did not restrict the time for search queries and the reported results were obtained by performing a full search. An overview of the average runtimes per dataset is shown in Table 7.1. The timings for feature extraction include the interest point selection, feature computation and validation point selection. The computation of candidate pairs and the evaluation of this pairs are part of the candidate matching timings. The average time for matching and evaluating a single scan pair is also provided. Furthermore, Table 7.1 lists the timings for the bag-of-words histogram computation for a single scan. Most of the time is spent for feature extraction, since the dictionary matching and histogram computation is rather cheap.

So far we did not analyze the time we can save if we use the similarity in appearance to sort our candidate set. The plots in figures 7.5(e), 7.6(e), 7.7(e), and 7.8(e) show the recall rates dependent on the allowed query time for the respective minimum acceptance threshold and maximum scan distance. Additionally, we added a comparison between the rotationally invariant and the regular version of the NARFs. As can be seen, the time needed to reach a comparable recall rate is higher for the rotational invariant version due to additional ambiguities



(a) Ground truth confusion matrix

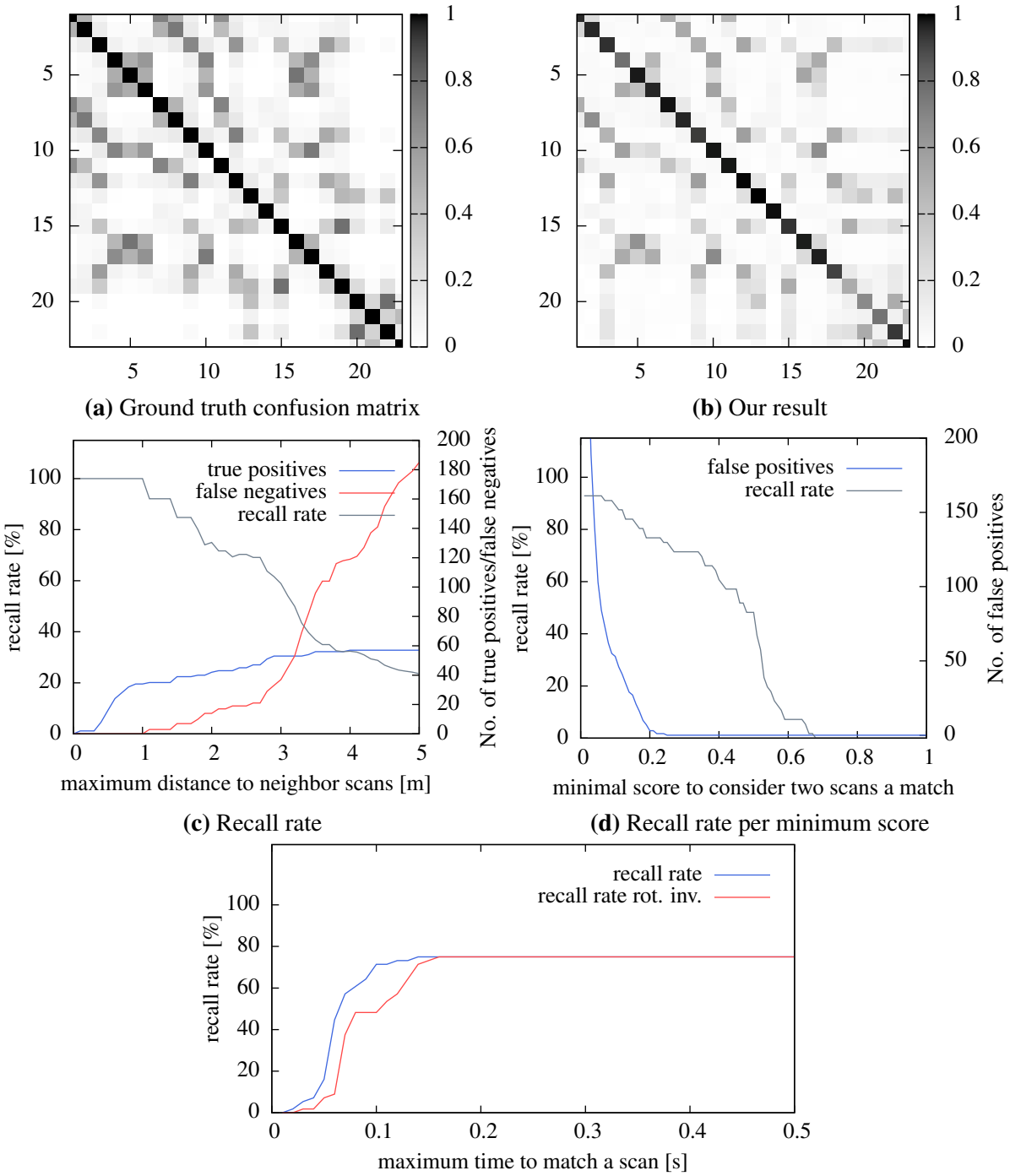
(b) Our result

(c) Recall rate

(d) Recall rate per minimum score

(e) Recall rate dependent on the maximum time

Figure 7.5: Results for the AASS-loop dataset. A comparison of **(a)** and **(b)** shows that our method detected all loop closures. Of course, the detection is more difficult the brighter a pixel is in **(a)**. The absolute number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to be considered as overlapping is shown in **(c)**. Our method did not return any false positives for the minimum acceptance threshold. The influence of different minimum scores on the number of false positives is evaluated in **(d)**. The recall rate is determined regarding a maximum distance of 1.0 m between the scans. The influence of the appearance detection with the bag-of-words is shown in **(e)**. In this plot we varied the maximum time for a search query to match a scan against the database. Furthermore we show the resulting recall rate for NARFs with and without the rotational invariance.



(a) Ground truth confusion matrix

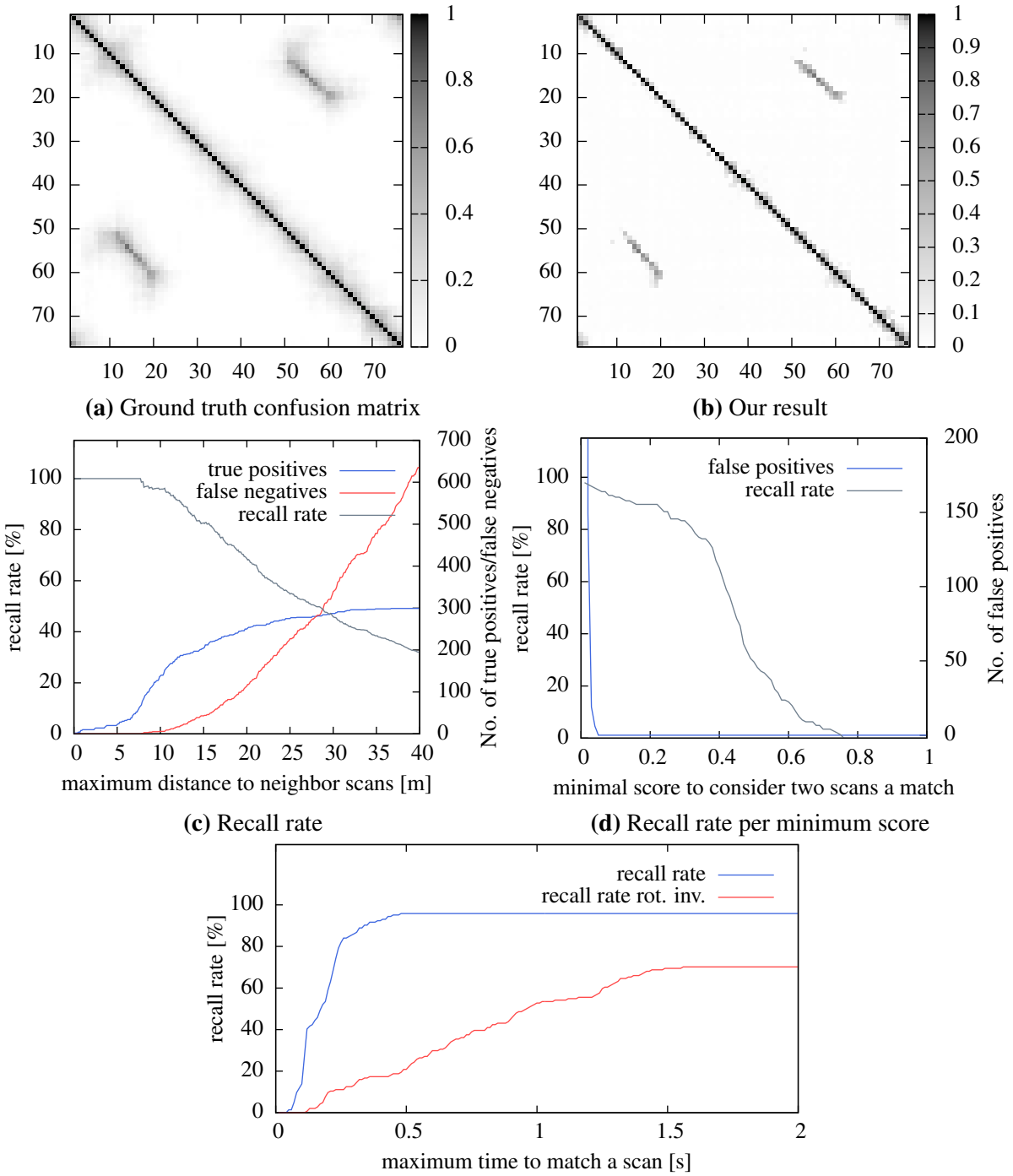
(b) Our result

(c) Recall rate

(d) Recall rate per minimum score

(e) Recall rate dependent on the maximum time

Figure 7.6: Results for the Quadrotor dataset. A comparison of **(a)** and **(b)** shows that our method detected all loop closures. Of course, the detection is more difficult the brighter a pixel is in **(a)**. The absolute number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to be considered as overlapping is shown in **(c)**. Our method did not return any false positives for the minimum acceptance threshold. The influence of different minimum scores on the number of false positives is evaluated in **(d)**. The recall rate is determined regarding a maximum distance of 2.0 m between the scans. The influence of the appearance detection with the bag-of-words is shown in **(e)**. In this plot we varied the maximum time for a search query to match a scan against the database. Furthermore we show the resulting recall rate for NARFs with and without the rotational invariance.



(a) Ground truth confusion matrix

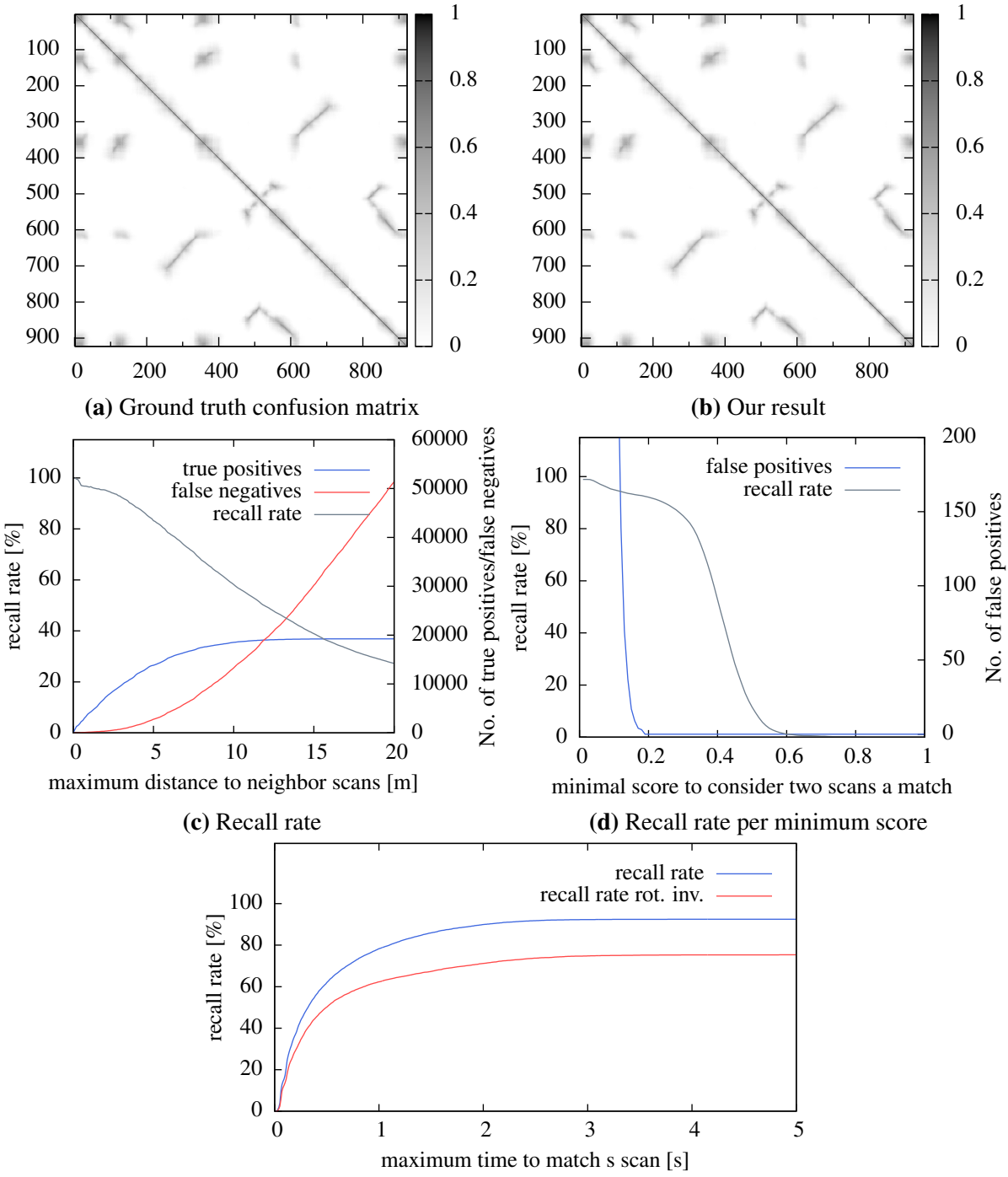
(b) Our result

(c) Recall rate

(d) Recall rate per minimum score

(e) Recall rate dependent on the maximum time

Figure 7.7: Results for the FreiburgCampus360 dataset. A comparison of **(a)** and **(b)** shows that our method detected all loop closures. Of course, the detection is more difficult the brighter a pixel is in **(a)**. The absolute number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to be considered as overlapping is shown in **(c)**. Our method did not return any false positives for the minimum acceptance threshold. The influence of different minimum scores on the number of false positives is evaluated in **(d)**. The recall rate is determined regarding a maximum distance of 10.0 m between the scans. The influence of the appearance detection with the bag-of-words is shown in **(e)**. In this plot we varied the maximum time for a search query to match a scan against the database. Furthermore we show the resulting recall rate for NARFs with and without the rotational invariance.



(a) Ground truth confusion matrix

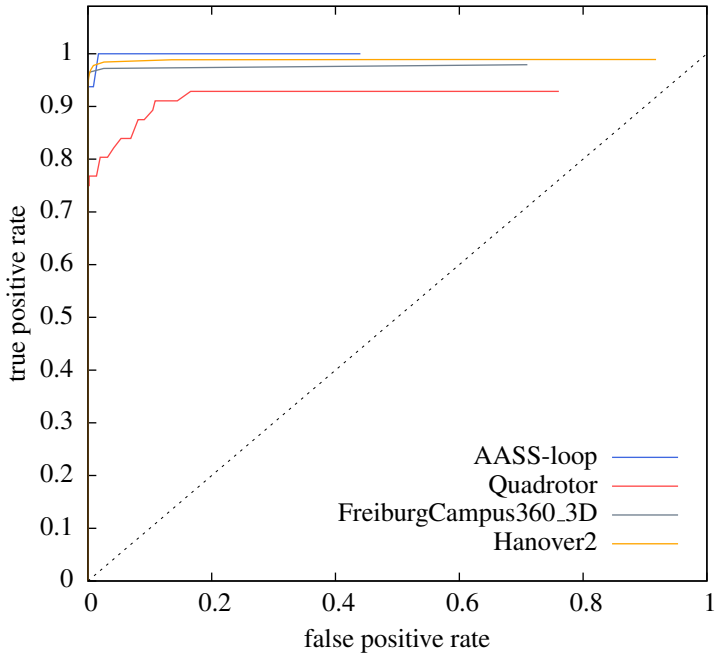
(b) Our result

(c) Recall rate

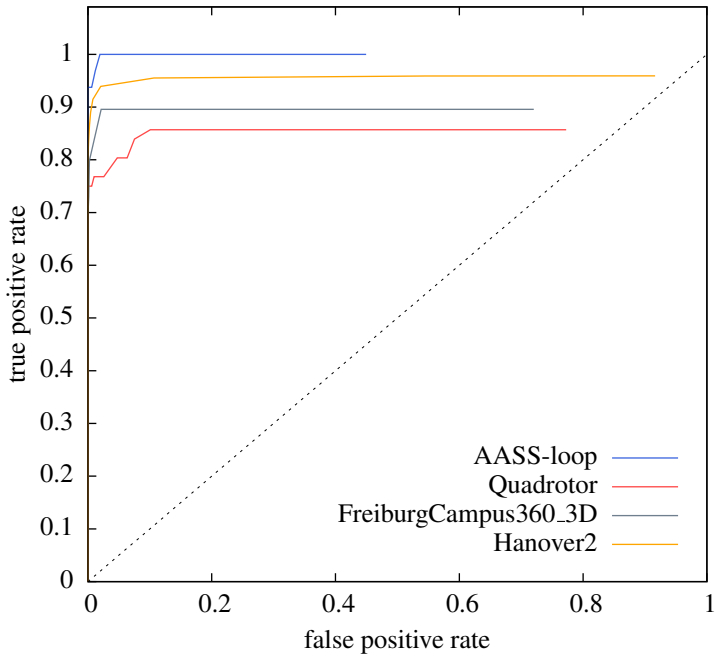
(d) Recall rate per minimum score

(e) Recall rate dependent on the maximum time

Figure 7.8: Results for the Hanover2 dataset. A comparison of **(a)** and **(b)** shows that our method detected all loop closures. Of course, the detection is more difficult the brighter a pixel is in **(a)**. The absolute number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to be considered as overlapping is shown in **(c)**. Our method did not return any false positives for the chosen minimum acceptance threshold. The influence of different minimum scores on the number of false positives is evaluated in **(d)**. The recall rate is determined regarding a maximum distance of 3.0 m between the scans. The influence of the appearance detection with the bag-of-words is shown in **(e)**. In this plot we varied the maximum time for a search query to match a scan against the database. Furthermore we show the resulting recall rate for NARFs with and without the rotational invariance.



(a) Non rotational invariant case



(b) Rotational invariant case

Figure 7.9: ROC (Relative Operating Characteristic) curves for our method on the four evaluated datasets. ROC curves plot true positives versus false positives and a perfect result would be located in the top left corner for 100% true positives and 0% false positives. As maximum distances between scans to be considered as true positive we used the similar values as described in Figures 7.5, 7.6, 7.7, and 7.8, 1.0 m for the AASS-loop dataset, 2.0 m for the Quadrotor dataset, 10.0 m for the Freiburg360_3D dataset, and 3.0 m for the Hanover2 dataset.

in the feature matching.

On the first three datasets the candidate matching on the full dataset is already close to the time needed to extract features for the bag-of-words histogram calculation on one scan. Therefore, the bag-of-words gives only a major speed up for the Hanover2 dataset, which consists of 923 scans. Our method reaches a recall rate around 80 with a restricted query time of one second. The feature extraction is also faster on this dataset since the scans have only relatively few points and the corresponding range image has a low resolution. The appearance based similarity enables our method to scale well on bigger datasets.

Our approach clearly outperforms the discussed state-of-the-art approaches in terms of the recall rate at cost of a higher computational complexity due to transformation estimation and geometric checking. Throughout our experiments we used a dictionary learned on the FreiburgCampus360_3D dataset. We achieved comparable results by learning the dictionary on one of the other datasets, non the less it might be possible that the results shown in Figure 7.7e are overconfident.

7.4 Related Work

Place recognition is a well-studied problem and has been addressed by a number of researchers in the past. Most approaches rely on camera data and use features like SIFT (Lowe, 2004) or SURFs (Bay et al., 2006) to find similar locations. One very successful approach in the context of mobile robots is the Feature Appearance Based MAPping algorithm (FABMAP) proposed by Cummins and Newman (2010). This approach uses SURF (Bay et al., 2006) appearances described with a bag-of-words to find similar places. The data was recorded with an omni-directional camera and it was shown to work reliably even on extremely large-scale datasets. We would like to refer the reader to this paper for a detailed discussion on both vision-based place recognition and bag-of-words approaches.

Vision data is known to be very descriptive and often unique. In case of 3D data features are less descriptive, but on the other side a geometric validation of matching candidate places is an advantage. An approach that is similar to ours regarding the utilization of point features to create candidate transformations is described in the PhD thesis of Huber (2002). His approach extracts Spin Images (Johnson and Hebert, 1999) from 3D scans and uses them to match each scan against a database. Huber reported 1.5 s as the time requirement to match one scan against another. Even considering the advances in computer hardware since 2002, our approach is substantially faster.

Li and Olson (2010) create visual images from LIDAR data, which enables them to use feature extraction methods from the vision sector to create a more universally usable point feature extractor. This feature extraction method is usable in 2D and 3D, although a 2D projection of the points is performed in the 3D case. Therefore, relative poses computed from feature correspondences will also just be 2D.

Several approaches have been designed especially for 2D range data. For, example, Bosse and Zlot (2008) presented a loop closure solution that builds local maps from consecutive 2D scans for which they compute histogram-based features. The correlation between these features is than used to match the local maps against each other. Tipaldi et al. (2010) perform place recognition on 2D range scans using the so-called FLIRT-features (Fast Laser Interest Region Transform). The features are used to find correspondences between points in the scans and transformations are extracted using RANSAC. Granström et al. (2009) proposed a machine learning approach to detect loops in 2D range scans. They extract a combination of rotation-invariant features from the scans and use a binary classifier based on boosting to recognize

previously visited locations.

Recently, Granström and Schön (2010) extended their system to 3D range scans. Their system only detects the existence of a loop closure and does not determine the relative transformation between scans. Magnusson et al. (2009) also proposed a system for place recognition based on 3D data. They utilize the Normal Distribution Transform as features to match scans. These features are global appearance descriptors, which describe the whole 3D scan instead of just small areas as it is the case for our features. While being very fast, their system does also not estimate relative poses and is therefore not able to perform geometric checking of the results. In Section 7.3, we compared our algorithm to these two methods. The results indicate that our approach yields substantially higher recall rates.

7.5 Conclusions

In this chapter, we presented a robust approach for recognizing places in 3D data. This is a key capability for a mobile robot to detect loop closures in 3D scans or to globally initialize a localization module. Our method estimates relative transformations, which are utilized for geometric validation of potential results. The presented appearance based ordering of the candidate set allows our method to scale well on larger datasets. Instead of a dictionary with Normal Aligned Radial Features we could also use a dictionary with surface primitives or sparse coded surface descriptions to find similar places. Additionally, we presented a detailed validation model that takes into account the visual capabilities of the sensor. We evaluated our approach on several publicly available datasets and outperform previous work in terms of the recall rate.

Chapter 8

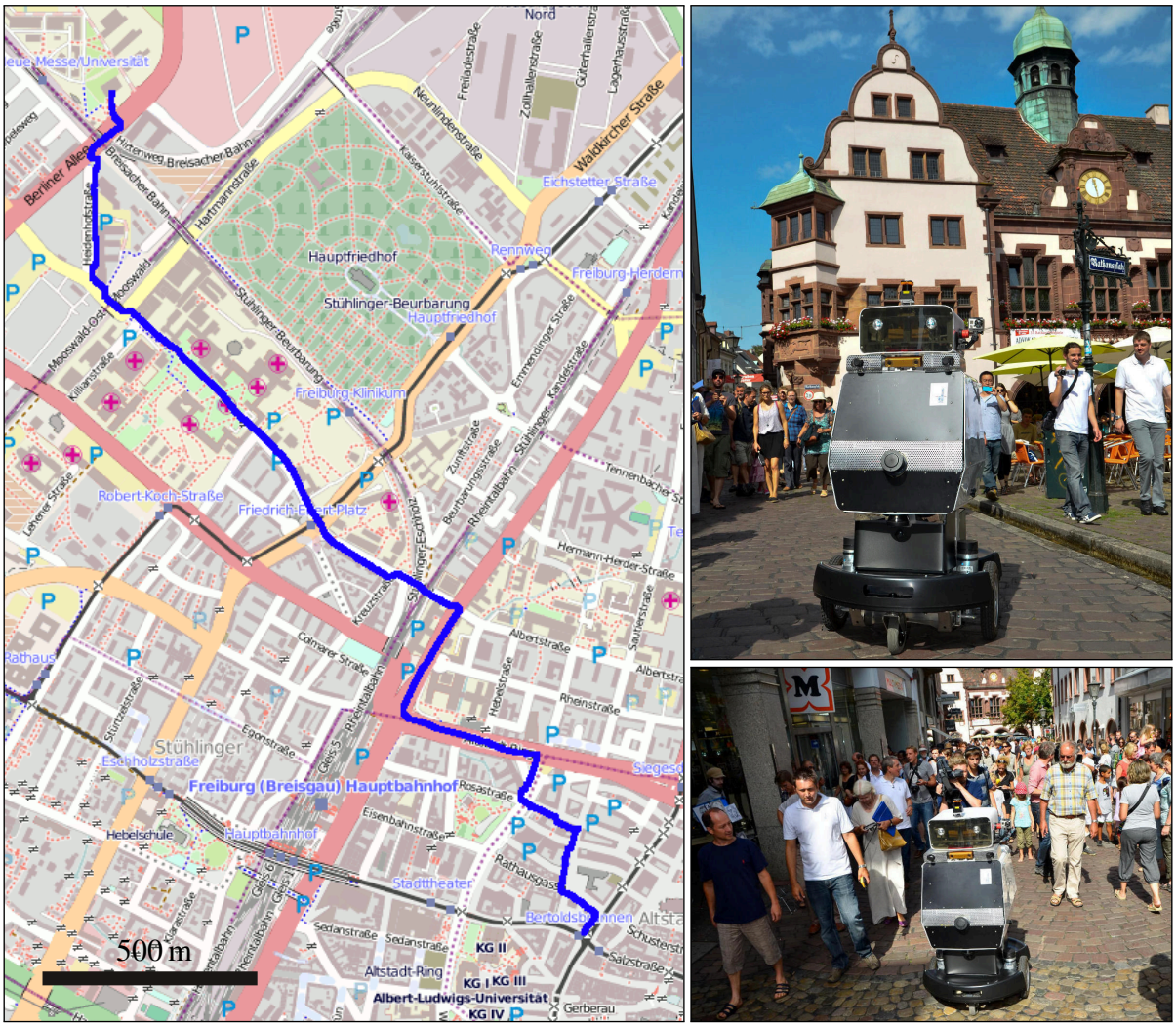
A Robotic Pedestrian Assistant

In the previous chapters, we presented methods to create highly accurate and compact models and to perform place recognition. All those techniques aim at enabling mobile robots to autonomously perform tasks by representing the environment, performing global localization or estimating loop-closure constraints for a SLAM problem. One fundamental challenge for mobile robots is safe, autonomous navigation. In this chapter, we present a navigation system for mobile robots designed to operate in crowded city environments and pedestrian zones. Urban areas introduce numerous challenges for autonomous robots, as they are highly complex and dynamic. We describe the different components of our navigation system including a SLAM module for dealing with huge maps of city centers, a planning component for inferring feasible paths taking also into account the traversability and type of terrain, a module for accurate localization in dynamic environments, and means for calibrating and monitoring the platform. Our navigation system has been implemented and tested in several large-scale field tests, in which a real robot autonomously navigated over several kilometers from our university campus to the city center of Freiburg. This also included a public demonstration, during which the robot was accompanied by several hundred people.

♦ ♦ ♦ ♦ ♦ ♦ ♦

Throughout the previous chapters, we presented several building blocks essential for mobile robots to perform environment modeling. One important application for such models is to enable a mobile robot to navigate through an environment. The majority of systems developed thus far, however, focuses on navigation in indoor environments, through rough outdoor terrain, or based on road usage. Only few systems have been designed for robot navigation in populated urban environments such as, for example, the autonomous city explorer (Bauer et al., 2009). Robots that are able to successfully navigate in urban environments and pedestrian zones have to cope with a series of challenges including complex three-dimensional settings and highly dynamic scenes paired with unreliable GPS information.

In this chapter, we describe a fully integrated system for autonomous navigation of a mobile robot in heavily crowded urban areas. As part of the system we employ a state-of-the-art graph based SLAM method together with a dedicated map data structure. This map data structure



(a) **(b)**

Figure 8.1: The trajectory taken by the robot during one of our large-scale urban navigation experiments is depicted as blue line in **(a)**. The visualized map is from OpenStreetMap (© OpenStreetMap contributors). In the city center the robot had to deal with a large number of people and cobble stone pavement as shown in **(b)**.

enables us to build maps at city scale and maintain only relevant or nearby information in the constrained memory of the robot. Another advantage of this dedicated map structure is that we can easily apply Sparse Surface Adjustment as described in Chapter 3 to optimize the accuracy of local maps instead of solving the optimization problem for a city scale map. Furthermore, we apply a variant of Monte-Carlo localization to estimate the pose of the robot in our map. Another crucial module of the system is the online terrain analysis to avoid potentially dangerous situations introduced by vegetated ground, were a robot could get stuck, dynamic obstacles, resulting in a possible collision, or negative obstacles which could result in a damaged robot. Additionally, we provide details on the integration of the modules and calibration of the sensors. Among others, we will present the result of a large-scale experiment, during which the robot Obelix traveled autonomously from our university campus to the city center of Freiburg during a busy day in August 2012. During that trial, the robot had to master a distance of over 3 km. The trajectory taken by the robot and two pictures taken during its experiment are depicted in Figure 8.1.

The aim of this chapter is to not only describe the relevant components but also to highlight the capabilities that can be achieved with a system like that. We motivate our design decisions and discuss critical aspects as well as limitations of the current setup. This chapter is structured as follows. First, we provide a detailed description of the robot platform in Section 8.1 and give an overview of the most important software components in Section 8.2. An extensive evaluation of the system is presented in Section 8.3. In Section 8.4 we discuss the results and the lessons learned. Afterwards, we discuss relevant related work in Section 8.5 and conclude the chapter with Section 8.6.

8.1 The Robotic Platform Obelix

The robot platform we used for collecting data and for evaluating the performance of our system is custom made and one of the outcomes of the EC-funded project EUROPA Project. The abbreviation EUROPA stands for European Robotic Pedestrian Assistant and the main purpose of this platform is navigation in indoor and outdoor environments, on sidewalks and in pedestrian zones. The robot is around 1.6 m tall, 0.9 m long, and 0.75 m wide. With this footprint it is able to access every place that is reachable for a wheel chair. It is equipped with a differential drive and two flexible caster wheels, one in the front and one in the back. Due to this configuration and its weight of approximately 100 kg, it is not possible, for the robot, to climb steps with a height of more than 3 cm. The robot is able to spin on the spot and its maximum velocity is 1 m/s.

Additionally, a touchscreen monitor is mounted in the back of the robot. The screen allows an experienced user to inspect the state of the navigation system. During our demonstrations, the screen provides information about the planned route of the robot as well as dialog windows to interact with users, for example, the robot displays a dialog to ask for permission before crossing a street.

In total we have four laser range finders mounted on the robot. Figure 8.2 (a) illustrates the scan plane of the two horizontally mounted SICK LMS 151. The LMS 151 have a 270° field of view. The forward-looking laser range finder uses only the centric 180° field of view to measure in the horizontal plane. The remaining beams are reflected by two mirrors and measure distances to the ground in front of the robot, as shown in Figure 8.2 (c). An additional SICK LMS 151 is mounted looking downward at 70° to observe the ground directly in front of the robot, as shown in Figure 8.2 (d). Figure 8.2 (b) illustrates the scan plane of the Hokuyo UTM-30LX mounted on a tilting servo on the head of the robot. This servo is actuated according to the measurements of an XSens IMU to align the UTM horizontally. The UTM-30LX is mainly used for mapping and localization, whereas the other laser range finders are used for traversability analysis. Furthermore, the robot is equipped with a Trimble GPS Pathfinder Pro and stereo cameras. The GPS measurements are used as prior information for the mapping task and to guide the global localization of the robot. The stereo cameras are only used to capture images for visualization and do not contribute to the navigation system.

8.2 System Overview

Our autonomous navigation system requires a map of the environment in which it should operate. For mapping an environment, the robot has to traverse it once controlled by a human. This can be done in a relatively small amount of time. We managed to record a 7.4 km long trajectory in around three hours. Since most structures in an urban environment do not change

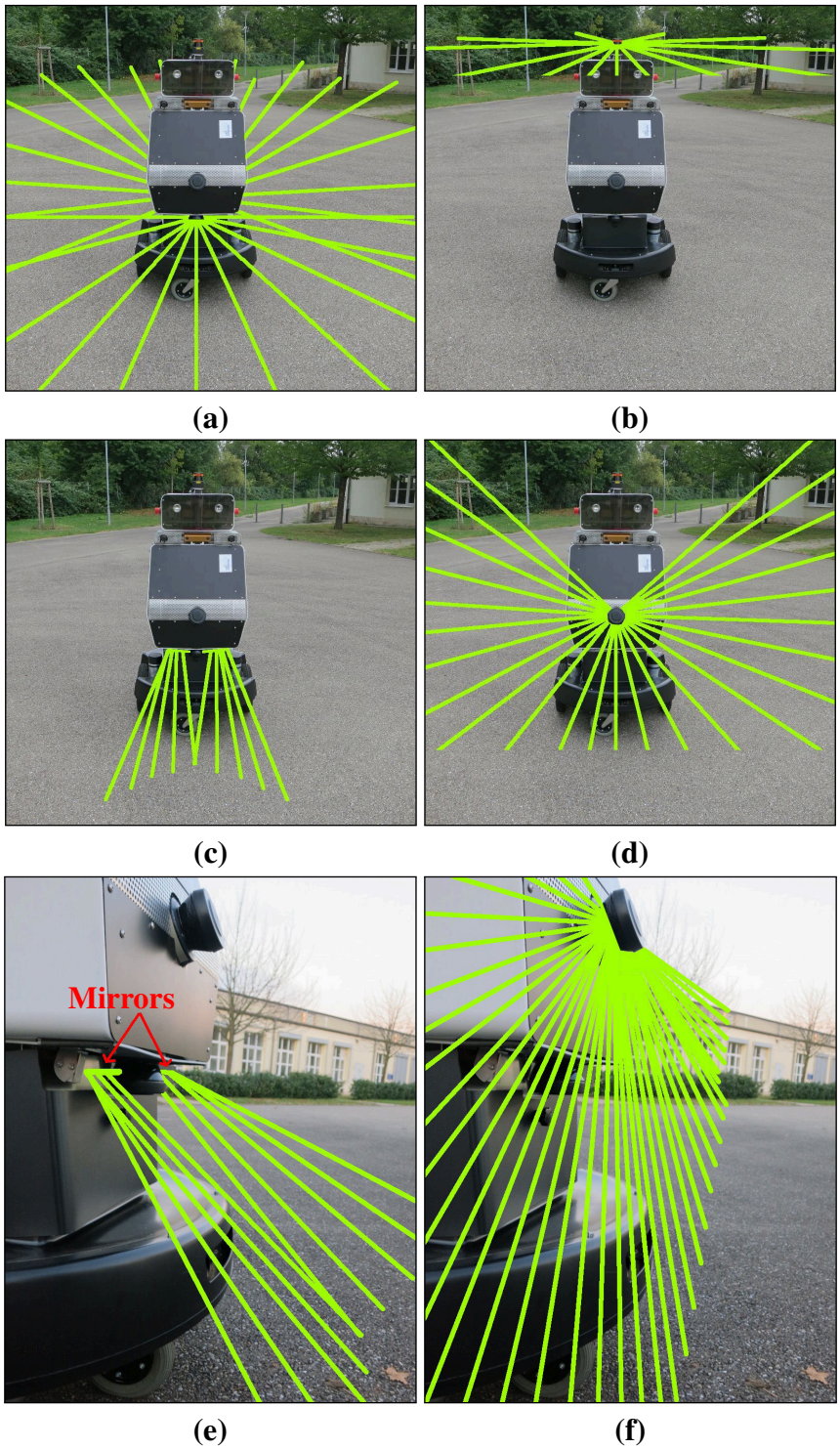


Figure 8.2: This figure illustrates the laser configuration of our robot. The fields of view of the individual laser range finders are visualized with green lines. Two SICK 270° laser range finders are mounted horizontally at the same height, one in the front and one in the back, as shown in **(a)**. Only the centric 180° of the front laser measure horizontally and the remaining beams are reflected onto the ground with two mirrors as shown in **(c)**&**(e)**. **(b)** shows the small Hokuyo range finder in the head of the robot. This scanner is mounted on a tilting servo and has also a 270° field of view. The third SICK range finder is mounted downward looking in the front of the robot to observe obstacles as illustrated in **(d)**&**(f)**. The laser scanners shown in **(a)** and **(b)** are mainly used for planning, localization and mapping, whereas the laser scanners illustrated in **(c)** and **(d)** are used for traversability analysis.

frequently, one has to do this only once. Our system is able to robustly handle small changes in the environment like displaced waste containers, billboards, or, shelves in front of shops. In the following, we present the key components of our autonomous navigation system. We will start with a description of the calibration of the four laser range finders, followed by a description of the SLAM module and a large-scale map data organization scheme. Furthermore, we will describe the algorithms used for localization, path planning, and obstacle detection. This system enables a mobile robot to perform large-scale navigation tasks in urban environments and highly populated city centers. The entire navigation system presented in this chapter runs on a standard i7 laptop with four cores at 2.3 GHz.

8.2.1 Calibration

As described in Section 8.1, our robot is equipped with multiple laser scanners. To enable the system to combine the measurements from the different sensors in a consistent manner, an accurate calibration for the positions of the sensors on the robot is required. To this end, in a first step, we extended our simultaneous calibration method (Kümmerle et al., 2012) to handle several laser range finders. The objective of the calibration is to estimate the positions $\mathbf{d} = (\mathbf{d}^{[1]}, \ldots, \mathbf{d}^{[K]})^\top$ of the range finders, where $K = 3$ for our robot.

Let $\mathbf{x}_i$ be the pose of the robot at time i. The error function $\mathbf{c}_{ij}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{d}^{[k]}, \mathbf{z}_{ij}^{[k]})$ measures how well the parameter blocks $\mathbf{x}_i$, $\mathbf{x}_j$, and $\mathbf{d}^{[k]}$ satisfy the measurement $\mathbf{z}_{ij}^{[k]}$, which is obtained as the scan-matching result of the range data acquired by the respective range finder at times i and j. If the three parameters perfectly satisfy the error function, then its value is $\mathbf{0}$. Here, we assume that the laser is mounted without inclination, which is the ideal condition. The error function $\mathbf{c}_{ij}(\cdot)$ has the following form:

$$\mathbf{c}_{ij}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{d}^{[k]}, \mathbf{z}_{ij}^{[k]}) = \Big((\mathbf{x}_j \oplus \mathbf{d}^{[k]}) \ominus (\mathbf{x}_i \oplus \mathbf{d}^{[k]}) \Big) \ominus \mathbf{z}_{ij}^{[k]}, \tag{8.1}$$

where $\oplus$ corresponds to the standard motion composition operator (Smith et al., 1990) and $\ominus$ denotes its inverse. We obtain the calibration result along with the positions of the robot by solving the following minimization:

$$(\mathbf{x}, \mathbf{d})^* = \operatorname*{argmin}_{\mathbf{x},\mathbf{d}} \sum_{k=1}^{K} \sum_{ij \in \mathcal{C}^{[k]}} \mathbf{c}_{ij}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{d}^{[k]}, \mathbf{z}_{ij}^{[k]})^\top {\Sigma_{ij}^{[k]}}^{-1} \mathbf{c}_{ij}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{d}^{[k]}, \mathbf{z}_{ij}^{[k]}), \tag{8.2}$$

where $\Sigma_{ij}^{[k]}$ represents the uncertainty of the scan matching result and $\mathcal{C}^{[k]}$ denotes the set of measurements that are available for a specific range finder. We solve Equation 8.2 using the Levenberg-Marquardt algorithm as it is implemented in the g^{2}o toolkit (Kümmerle et al., 2011).

To carry out the calibration procedure outlined above, we record a dataset in an indoor environment. This typically provides good results for a scan-matching process. The trajectory taken by the robot has to consist of translations as well as rotations, which we can guarantee easily by controlling the robot appropriately. Otherwise, we would not be able to recover the positions of the onboard sensors (Brookshire and Teller, 2011).

The method described above is not applicable for the downward facing laser and the mirrored beams, as can be seen in Figure 8.2(b) and (c), since scan matching cannot estimate the ego-motion of the robot by considering those measurements. However, they observe the same region of the environment when the robot is moving. We therefore developed a module that exploits this to calibrate the downward facing laser and the mirrors to obtain consistent 3D measurements while driving with the robot. The calibration procedure works as follows: First,

(a) **(b)**

Figure 8.3: (a) shows an example environment that we used for our calibration procedure. We created a narrow corridor with several objects on both sides, for the robot to drive through. These objects provide structure that is visible both in the downward facing laser scanner and the laser beams reflected by the mirrors. **(b)** shows the corresponding point clouds, in which the black points were measured by the downward facing scanner and the blue and red points are from the beams reflected by the left and the right mirror respectively. These point clouds were created with the already calibrated parameters.

the robot is driven through a small area with significant 3D structure that is visible in both, the downward facing laser and the mirrored beams. We typically used a straight trajectory of about 3 m. Figure 8.3(a) depicts an example environment that we used for the calibration procedure. After this data is collected, the point clouds of the different sources — the downward facing laser, the beams hitting the left mirror, and the beams hitting the right mirror — are accumulated given the odometry of the robot and the current estimate for the calibration parameters. Figure 8.3(b) visualizes the resulting point cloud. The user can manually adjust the parameters in a graphical user interface to provide a reasonable initial guess. Given this guess, the system starts a sampling based greedy method to improve the overlap of the different point clouds. That means the system changes the parameters of either one of the lasers or one of the mirrors, recalculates the corresponding point cloud and determines the overlap with the other clouds. If the overlap was improved, the calibration parameters are replaced with this solution or discarded otherwise. This process is then repeated. We found that this leads to a much more consistent calibration than what we achieved by manually tuning the parameters and takes less time.

8.2.2 Mapping

We use the graph-based SLAM formulation as described in Chapter 2 and estimate the maximum-likelihood (ML) configuration of a graph to compute an accurate map. For convenience, we briefly recall this formulation. We describe the poses of a robot with a vector $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)^\top$, in which each element corresponds to a pose at a certain time $t \in T$. An observation describes the motion of the robot between two poses at times i and j and is modeled with a mean $\mathbf{z}_{ij}$ and a covariance Σ_{ij}. Furthermore, we assume Gaussian noise. From the observations, we generate relations for our graph and seek to find the ML configuration. Therefore, we need an error function $\mathbf{e}_{ij}(\mathbf{x})$ which evaluates the difference between an observation $\mathbf{z}_{ij}$ and the estimated configuration of the nodes i and j. A definition of $\mathbf{e}_{ij}(\mathbf{x})$ can be found in Chapter 2 where the information matrix Ω_{ij} corresponds the inverse of the covariance Σ_{ij}. The function $\mathbf{e}_{ij}(\mathbf{x})$ is

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

Figure 8.4: GPS outliers can have a large impact on the consistency of the resulting model. GPS measurements and the trajectory of the robot are shown in **(a)**. If the SLAM estimate uses all prior information, including GPS outliers, the map could be corrupted by the erroneous GPS measurements, as can be seen in **(b)**. With the method described in this section we can reject prior information that is very likely to correspond to outliers. The map shown in **(c)** is more consistent, the walls are straight and there are less blurred artifacts in the map.

closely related to $\mathbf{c}_{ij}(\cdot)$ (see Section 8.2.1) but it directly employs the previously found calibration result as known parameters.

Since our robot is equipped with a GPS device, we can use GPS measurements as prior information to estimate a globally consistent map. To consider prior information $\hat{\mathbf{x}}_i$ for a node i, we need an additional error function $\mathbf{e}_i(\mathbf{x})$ with a covariance Σ_i. The differences between prior and current state are computed with the function $\mathbf{e}_i(\mathbf{x})$, which is defined as

$$\mathbf{e}_i^{gps}(\mathbf{x}) = (\mathbf{x}_i \ominus \hat{\mathbf{x}}_i)^\intercal {\Sigma_i}^{-1} (\mathbf{x}_i \ominus \hat{\mathbf{x}}_i). \tag{8.3}$$

We assume that all measurements are independent and we can compute the ML configuration of the graph as

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{ij \in \mathcal{C}} \mathbf{e}_{ij}(\mathbf{x}) + \sum_{i \in \mathcal{P}} \mathbf{e}_i^{gps}(\mathbf{x}), \tag{8.4}$$

where $\mathcal{C}$ corresponds to the set of relations and $\mathcal{P}$ corresponds to the priors. In Equation 8.4 we minimize the negative log-likelihood of the joint probability of all measurements. This is similar to computing the maximum-likelihood configuration given the measurements. More details can be found in the work of Dellaert and Kaess (2006) or Grisetti et al. (2010). Similar to the problems presented in Chapters 2 and 3, we can efficiently solve this minimization task with the g^{2}o framework (Kümmerle et al., 2011). So far we described a typical graph based SLAM back-end. Before we can solve the minimization problem, we need to construct the graph and a set of relations $\mathcal{C}$ between poses and the corresponding covariances in a laser-based front-end. The approach we follow is an extension of the method proposed by Olson (2008). We estimate the motion of the robot between successive poses with a correlative scan matcher. Loop closures are obtained by matching the current scan against a database of reference scans. To speedup this procedure we consider only poses that are in a three-sigma uncertainty ellipsoid around the current scan. To reject possibly wrong loop closures we perform spectral clustering to identify a maximally self-consistent set of loop closure hypotheses, similar to Olson (2009a).

Since our robot is equipped with a GPS sensor, we additionally collect a set of priors $\mathcal{P}$, to improve the accuracy of the map. Due to multi-path effects, GPS measurements may be corrupted and repeatedly measure a pose, which is off by several meters. Such outliers are hard

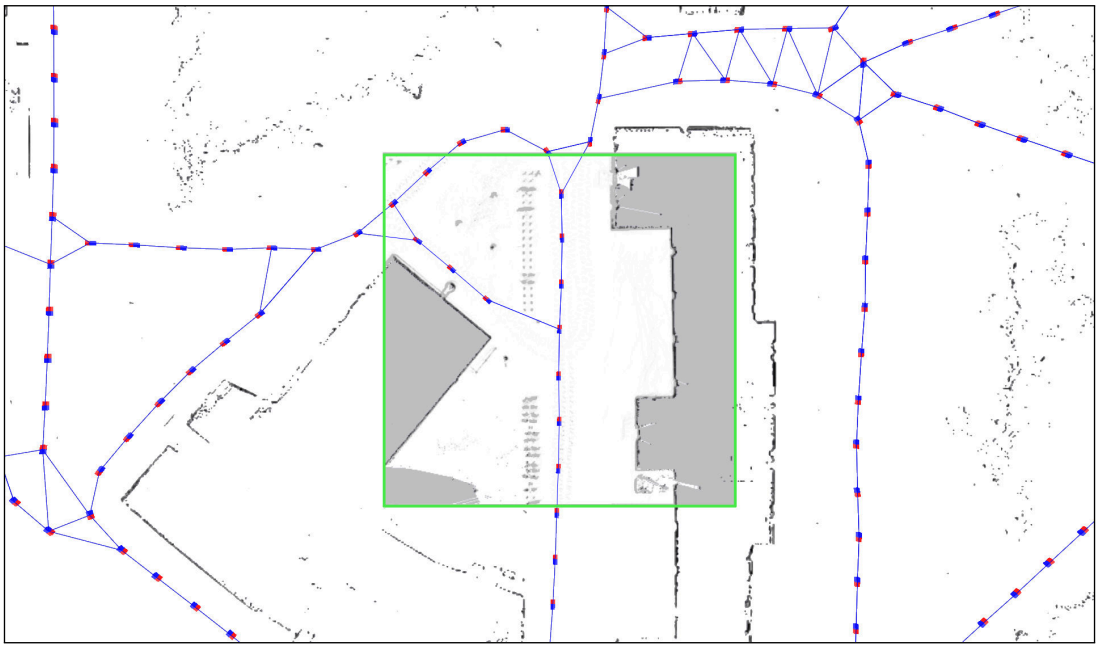


Figure 8.5: An example map of our campus. The graph nodes are colored red and blue and the edges are colored blue. The green marked region corresponds to a 40 m $\times$ 40 m local map. Since we load only small parts of the environment into the main memory, we use pre-computed low-resolution thumbnails for visualization of the local maps that are not available.

to detect by just looking at the GPS measurements. Therefore, we extend Equation 8.4 with a robust cost function (Pseudo Huber cost function (Hartley and Zisserman, 2004)) to restrict the influence of outliers. First, we solve Equation 8.4 and, once this is computed, we use the residual error of the priors to reject outliers. Therefore, we remove the top 2 % edges with the largest residual error. This is performed five times and leaves approximately 90 % of the prior information. In this way, we might reject some good GPS measurements but on the other side the impact of outliers can be prominent in practical applications as shown in Fig. 8.4. In general, the prior information helps to build more accurate maps (Kümmerle et al., 2011). Furthermore, the GPS priors provide a general coordinate frame, which can be used to transfer information between different maps or help to fuse several trajectories into one map.

8.2.3 Map Data Structure

We represent the 2D laser data with occupancy grid maps. Once we have a SLAM solution, we can render a map by integrating every scan along the trajectory at the known poses. One problem that becomes evident when dealing with city scale maps is the memory consumption. Consider a map with dimensions of 2 $\times$ 2 km and a resolution of 0.1 m, with 4 bytes per cell this leads to a requirement of 1.5 GB main memory. Instead of representing everything with one grid map, we use the topology of our graph solution and render only local maps at specific locations on the robot trajectory. Therefore, we select a subset out of the SLAM pose graph solution by considering only nodes that have a minimum distance of 5 m to all other connected nodes in the subset. Furthermore, we transfer the connectivity information of the pose graph to the constructed graph structure. Every node in the sparse graph can store grid maps. Recently, Konolige et al. (2011) presented a similar approach.

For rendering a local occupancy grid map, we consider only laser scans that have a certain distance on the resulting SLAM trajectory to a node in the map graph structure. To compute

this distance we apply Dijkstra's algorithm. This has the advantage that we only consider measurements observed in close proximity to a node. In this way, we can easily deal with variations in height or different levels and represent only structure that is visible around a node location. We select a set of nodes V_{map} from the SLAM pose graph to be part of a local map with the center at $\mathbf{x}_{base}$ as

$$V_{map} = \{\mathbf{x}_i \in \mathbf{x} \mid dijkstra(\mathbf{x}_i, \mathbf{x}_{base}) < \delta\}. \tag{8.5}$$

The distance reported by Dijkstra's algorithm $dijkstra(\mathbf{x}_i, \mathbf{x}_{base})$ has to be below the maximum distance δ to be considered in the rendering of a local map. Since hard drive space is rather cheap and to avoid unnecessary computations, we once render a local map for every node at the first access and store them on the disk. Throughout our experiments, we used a size of 40 m $\times$ 40 m for local maps. A reasonable map size depends on the typical range capabilities of the used laser range finder. Both the localization module and the path planning module described in the following are designed to work on this map data structure.

Another interesting aspect is that we can improve a local map before rendering by optimizing the nodes in V_{map} with sparse surface adjustment as described in Chapter 3. Instead of optimizing a several kilometer long trajectory, we can independently optimize the nodes and laser scans of each local map. Thus, we can compute a solution more efficiently and improve the accuracy of our localization module by reducing the uncertainty in the underlying map.

8.2.4 Localization

To estimate the pose $\mathbf{x}$ of the robot given a map, we maintain a probability distribution $p(\mathbf{x}_t \mid \mathbf{z}_{1:t}, \mathbf{u}_{0:t-1})$ of the location $\mathbf{x}_t$ of the robot at time t given all observations $\mathbf{z}_{1:t}$ and all control inputs $\mathbf{u}_{0:t-1}$.

Our implementation employs a sample-based approach that is commonly known as Monte Carlo localization (MCL) (Dellaert et al., 1999). MCL is a variant of particle filtering (Doucet et al., 2001) in which each particle corresponds to a possible robot pose and has an assigned weight $w^{[i]}$. In the prediction step, we sample a motion for every particle according to the prediction model $p(\mathbf{x}_t \mid \mathbf{u}_{t-1}, \mathbf{x}_{t-1})$. Based on the sensor model $p(\mathbf{z}_t \mid \mathbf{x}_t)$ each particle gets assigned a new weight within the correction step. To focus the finite number of particles in the regions of high likelihood, we need to re-sample the particles. A particle is drawn with a probability proportional to its weight. However, re-sampling may drop good particles. To avoid this, the decision when to re-sample is based on the number of effective particles N_{eff} (Grisetti et al., 2005; Liu, 1996). Our current implementation uses 1,000 particles.

A crucial question in the context of localization is the design of the observation model that calculates the likelihood $p(\mathbf{z} \mid \mathbf{x})$ of a sensor measurement $\mathbf{z}$ given the robot is located at the position $\mathbf{x}$. We employ the so-called endpoint model or likelihood fields (Thrun et al., 2005). Let z'_k be the k^{th} range measurement of $\mathbf{z}$ re-projected into the map given the robot pose $\mathbf{x}$. Assuming that the beams are independent and the noise is Gaussian, the endpoint model evaluates the likelihood $p(\mathbf{z} \mid \mathbf{x})$ as

$$p(\mathbf{z} \mid \mathbf{x}) \propto \prod_i \exp\left(-\frac{\|z'_i - d'_i\|^2}{2\sigma^2}\right), \tag{8.6}$$

where d'_i is the point in the map which is closest to z'_i. As described in Section 8.2.3, we use a graph-based model with locally attached submaps. Therefore, we have to determine a vertex $\mathbf{x}_{base}$ in the graph, in which we evaluate our sensor model $p(\mathbf{z} \mid \mathbf{x})$. In our current

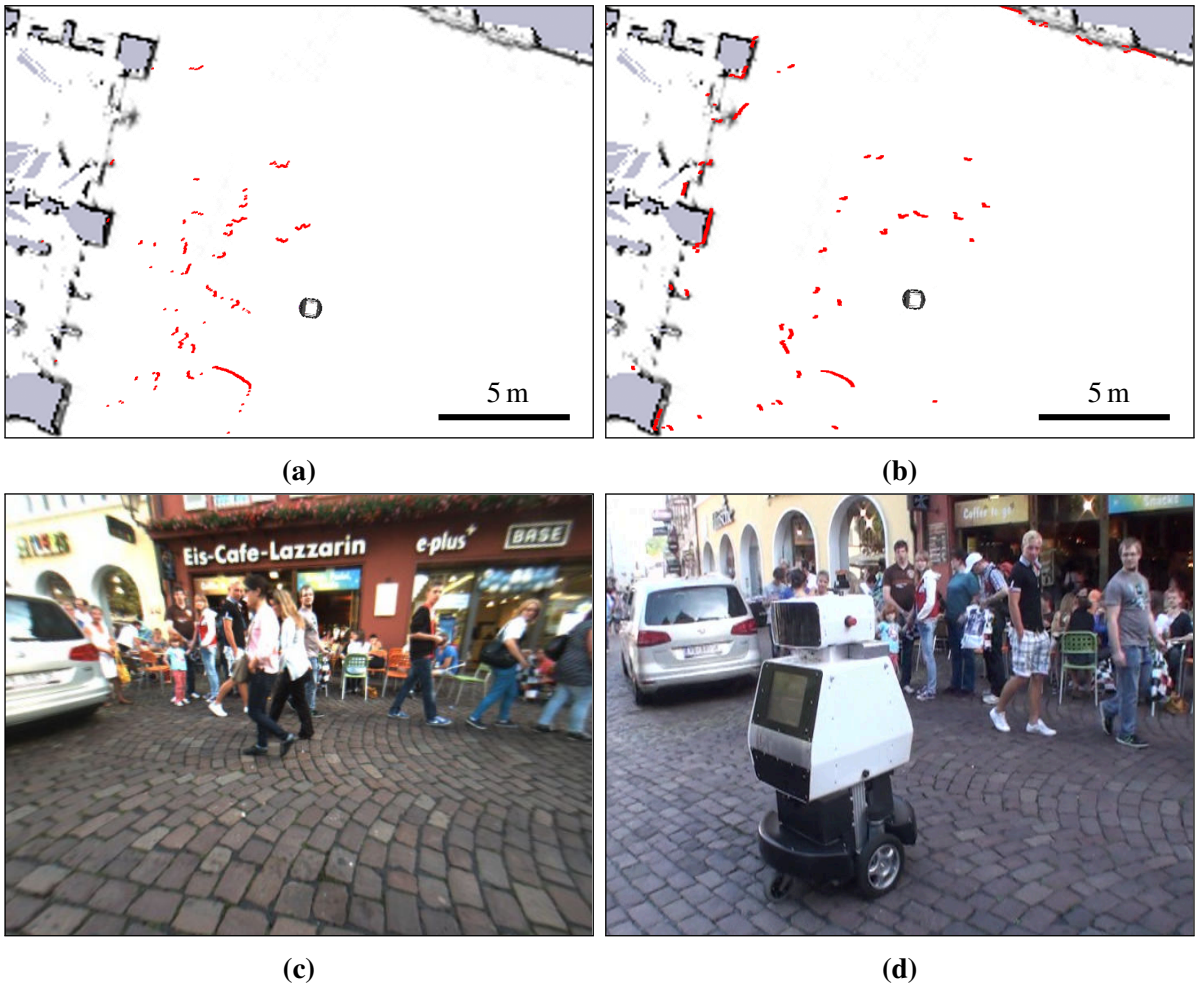


Figure 8.6: This figure shows a crowded scene in the city-center of Freiburg. The height of a horizontally mounted laser plays a major role for localizing a mobile robot. (a) shows a laser scan at a height of 0.55 m, whereas (b) shows a corresponding laser scan acquired at 1.55 m. As can be seen, the higher mounted laser better observes the static structure of our map. The camera view of the robot is shown in (c) and an image taken behind the robot is shown in (d).

implementation, we select the node $\mathbf{x}_{base}$ according to the minimum distance to a pose $\mathbf{x}$ and a visibility constraint that checks if $\mathbf{x}$ was observed in the local map of $\mathbf{x}_{base}$ as empty space and not as unknown. Without this visibility constraint we could encounter situations in which a mobile robot is indoors and drives close to an outer wall of a building and the closest vertex is outside of the building. In such a case, the robot observations and the map would have only few overlap. This could lead to a growing uncertainty of the pose estimate.

Another important design-related question for a robust localization in crowded urban environments is where to mount a laser scanner. Fig. 8.6 shows an exemplary scene in downtown Freiburg. The interesting aspect is the amount of measurements (shown as red points) that match to the map. Fig. 8.6(a) depicts the measurements of a laser mounted at a height of 0.55 m. In comparison, Fig. 8.6(b) visualizes a corresponding measurement of a laser mounted at a height of 1.55 m. The higher mounting position of the laser allows the robot to sense more of the static parts of the environment. This improves the localization since the robot is able to measure above small obstacles, e.g., chairs, shopping bags, bikes, and children.

A challenging problem is the pose initialization after starting the system also referred to as global localization. Since we have a GPS sensor on the robot and a GPS aligned map, we use

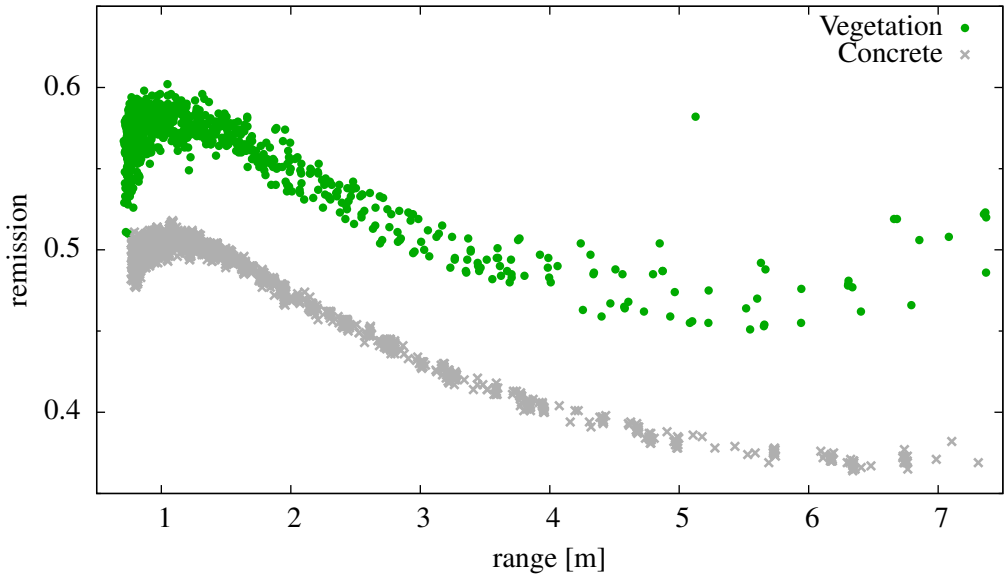


Figure 8.7: Range and remission data collected by the robot observing either a concrete surface or vegetation.

GPS measurements as initial guess to localize the robot. In that way, we avoid to distribute a massive amount of particles on a map that covers several kilometers and save computational resources. Furthermore, we exploit the compass of the Inertial Measurement Unit (IMU) to constrain the heading of the particles. Based on the GPS and IMU measurements, we sample a pose for each particle from a Gaussian distribution around the initial guess. As a standard deviation, we apply 5 m for the position and 10° for the heading. In case the automatic pose initialization fails because there is no GPS signal, e.g., if the robot is indoors, the user has the possibility to provide an initial guess manually.

8.2.5 Traversability Analysis

Since environments may change and the robot pose reported by the localization module might be off by a few centimeters, the robot needs to analyze the terrain for traversability online and cannot only rely on the traversability information stored in the map. The question whether an area is traversable by a robot depends on the capabilities of the platform. In case of the robot used throughout our experiments, we had to deal with a maximum permitted step size of 3 cm. Therefore, we have to detect obstacles that are lying on the ground and also negative obstacles, like gaps or holes in the ground, with a change in height of over 3 cm. Manhole covers and grids are a further challenge since a robot can usually traverse them but they are often classified as negative obstacles. In the following, we will describe the algorithm used to solve this task. Furthermore, we are interested in the dynamics of the surrounding scene. In heavily crowded city-centers we would like to treat moving people not just as static obstacles but also predict their motion. In a sense, we predict which space will be occupied in the near future. We do this only for a very small time horizon to help the planning module to react accordingly. Due to the weight of the robot we also have to classify vegetated areas to prevent the robot to get stuck in the soft ground. With laser range finders, it is hard to geometrically decide whether the ground in front of the robot is lawn or street and therefore traversable or not. Due to sensor noise and accuracy of the laser, a lawn might look as smooth as a street. In the following, we will briefly describe the traversability analysis sub-modules.

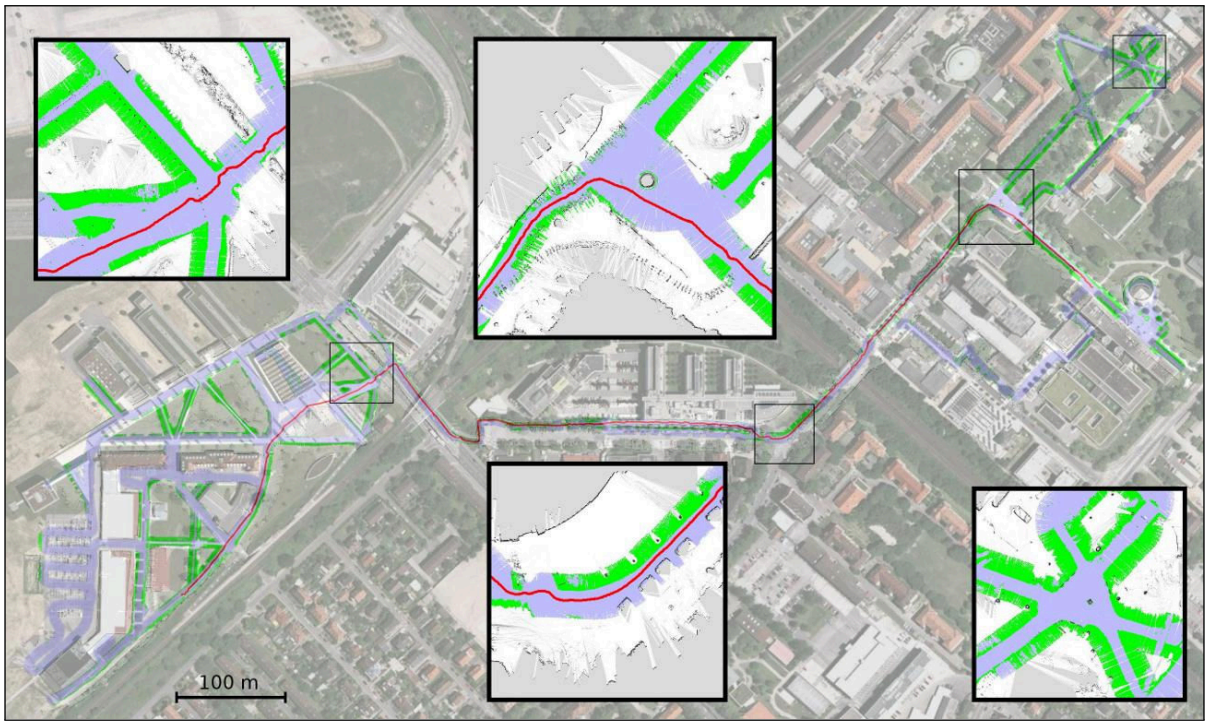


Figure 8.8: A map of the environment in which areas covered with vegetation are annotated. Areas that have been classified as vegetation are drawn in green, while concrete is drawn in blue. The trajectory drawn in red corresponds to a path that the robot travelled autonomously within one of the field tests.

Vegetation Detection

The detection of vegetated areas, such as lawn, with the available laser range finders cannot be reliably solved by just looking at the geometry. Many common laser range finders measure not just range but report also the amount of light that was reflected back to the sensor. These so called remission values depend on the range and the optical attributes of the measured surface in the near infrared spectrum of the laser beam. Living plants are typically very bright in infrared light and this fact can be used to reliably distinguish between vegetation and concrete (Wurm et al., 2009, 2012).

Another interesting aspect of the remission-based classification is that it also depends on the incidence angle of the beam on the measured surface. Since our robot is only equipped with a fixed downward looking laser, possible incidence angles are more constraint than in the case of an actuated laser scanner as used by Wurm et al. (2009). This makes the classification problem easier and enables us to solve it in an efficient manner with a non-linear function. In Fig. 8.7 we see a plot with data recorded by our robot. As can be seen the data for vegetation and concrete ground can be separated by a non-linear function. Compared to Wurm et al. (2009) the classification results are slightly worse with our method but still provide reasonable results, as can be seen in Figures 8.8 and 8.9. The main advantage of our method is that the computation is much faster. This allows us to do the classification online. Of course, we can also annotate a map by classifying the collected data. Fig. 8.8 shows an overlay of our annotated map with an aerial image. The vegetated areas are accurately classified and our robot is able to avoid potentially dangerous vegetated areas.

Tracking Dynamic Obstacles

We track moving objects in the 2D laser measurements of the robot with a blob tracker and do not focus on detecting pedestrians, bicyclists, or cars as classes but search for changes in small robot centered 2D grid maps. We build such a map in a time window of approximately 100 ms. To detect changes, we store several maps collected in the last second and compare the current map with the oldest available map. We mark all differences that are in the current map and not in the oldest map as dynamic.

Some of the regions marked as dynamic might relate to an area that was occluded in the previous map and did become visible because the robot or a dynamic obstacle moved. To avoid such results we perform a visibility check in all available maps for every obstacle marked as dynamic. If the line-of-sight was occluded with another obstacle, we drop the candidate. To group the dynamic candidates into a blob, we apply region growing. Given a blob, we search for a nearest neighbor in the blobs reported for the map that is closest in time to the current map in our history. Blob associations with a distance above a threshold are dropped. Furthermore, we compute the mean and bounding box for each blob. The velocity for each blob is computed based on the difference in the mean of the blob and its corresponding blob in the preceding map.

The presented method is rather simplistic and might report false positives from time to time but nonetheless proved to be very valuable for navigation in highly crowded areas, such as city-centers. Another important aspect is that the computational demands are rather moderate. Fig. 8.9 shows the movement predictions for four people walking behind the robot.

Detection of 3D Obstacles

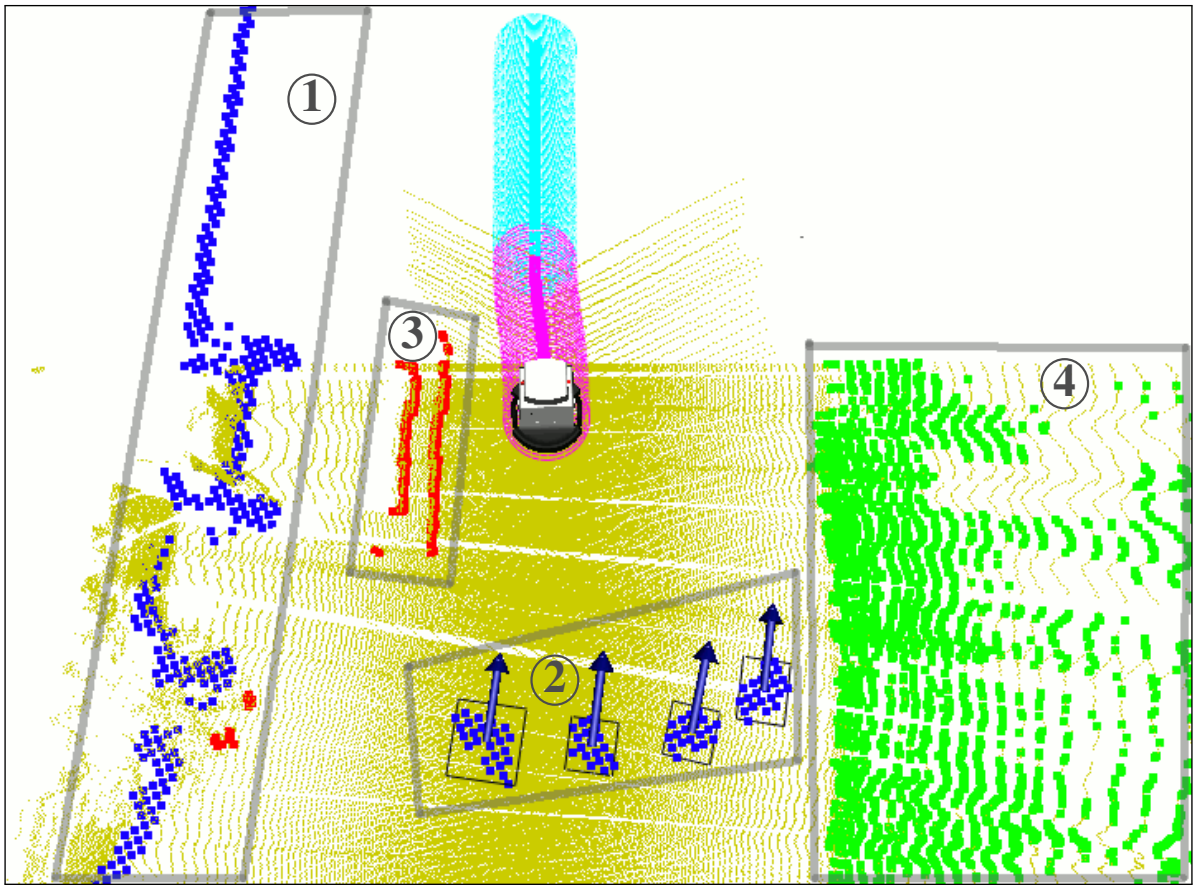
Unfortunately, it is not possible to observe all potentially dangerous obstacles in the horizontal 2D laser measurements. As already mentioned in Section 8.1 we equipped the robot with a downward looking laser and projected some of the horizontal beams to the ground in front of the robot. This allows the robot to collect 3D information while moving through an environment.

Similar to Pfaff et al. (2007), we analyze the incoming laser measurements per scan line, instead of collecting a point cloud while driving through an environment. This allows online processing and is also faster than reprocessing a collected point cloud. Since flat ground is typically observed as straight line in a scan, we look for larger discontinuities in every scan. To reduce the impact of sensor noise, we approximate small segments of a scan line with polynomials. In the next step, we check the gradient of each segment and mark all points that fall into a segment with a gradient larger than 10° or report a height difference of more than 3 cm as potential obstacle. Those thresholds directly correspond to the capabilities of the platform.

The described method has several advantages, it is efficient since it operates only on single 2D scans and we can avoid integrating data over a longer time period. In this way, we avoid errors in the point cloud, introduced by errors in the motion estimate or odometry of the robot. Another advantage is that the described method does not suffer from potential errors in the sensor calibration, as it could be the case for processing a point cloud fused out of several laser range finders. Efficiency is also important since we would like to have only a very small time delay between getting the data and the classification results in order to have a responsive system.

Of course, there are also disadvantages or error cases for the described method. Fig. 8.10 (a) illustrates problematic ground surfaces that will be classified as non-traversable. Manhole covers and gutters are usually traversable by the robot but do not have a smooth and regular surface. To deal with such cases we extended our method by accumulating points in a 2D elevation map (Bares et al., 1989; Hebert et al., 1989). Each cell in this grid stores the average height of the measured points that fall into that cell. To classify a point as belonging to a hole, we

(a)



(b)

Figure 8.9: An example scene is shown in **(a)**. In **(b)** we illustrate the information of the traversability analysis module as point cloud for the same scene. All blue points were observed by the horizontal laser range finders (① and ②). The four people walking behind the robot are shown in ② together with their estimated direction and speed. Red points correspond to reported obstacles observed in the downward looking laser that were not visible in the horizontal laser range finders, e.g., the steps shown in ③. The green points in the area shown in ④ correspond to the classified vegetation. All other points are colored yellow.

check if the height is substantially below the average elevation of the neighboring cells in the grid. Usually, this is the case for points that measure a small hole, while the neighborhood in the grid reports the ground surface. In this context, a hole is small with respect to the chosen grid size. This also works for gutters where only a few point fall below the ground surface. To avoid detections of false negative obstacles for such surfaces, we ignore all obstacles below a height of 10 cm in an area where we detected a small hole. This extension enables our method to reliably avoid false negative obstacle detections in areas that can be traversed by the robot, as can be seen in Figure 8.10.

Given the current sensor setup of our robot, it is not possible to reobserve 3D obstacles. Everything is observed once while driving by. Therefore, our method has to forget about old 3D obstacles after a time window to avoid to get stuck between previously observed obstacles that are no longer present. This time window has to be reasonably long to avoid oscillating behavior. Everything that is visible in the horizontal laser range finders can be reobserved at a high frequency. Therefore, we would like to ignore 3D obstacles that are close to a 2D obstacle, which is visible in the horizontal laser scanners. Especially in the case of dynamic obstacles, we can avoid to trap the robot with 3D obstacles of moving people. Figure 8.9 gives an overview of the different obstacle types classified by our traversability module. All blue points were observed by the horizontal laser range finders (① and ②). The four people walking behind the robot are shown in ② together with their estimated direction and speed as reported by our blob tracker. Red points correspond to reported 3D obstacles observed in the downward looking laser that were not visible in the horizontal laser range finders, e.g., the steps shown in ③. The green points shown in ④ correspond to the classified vegetation.

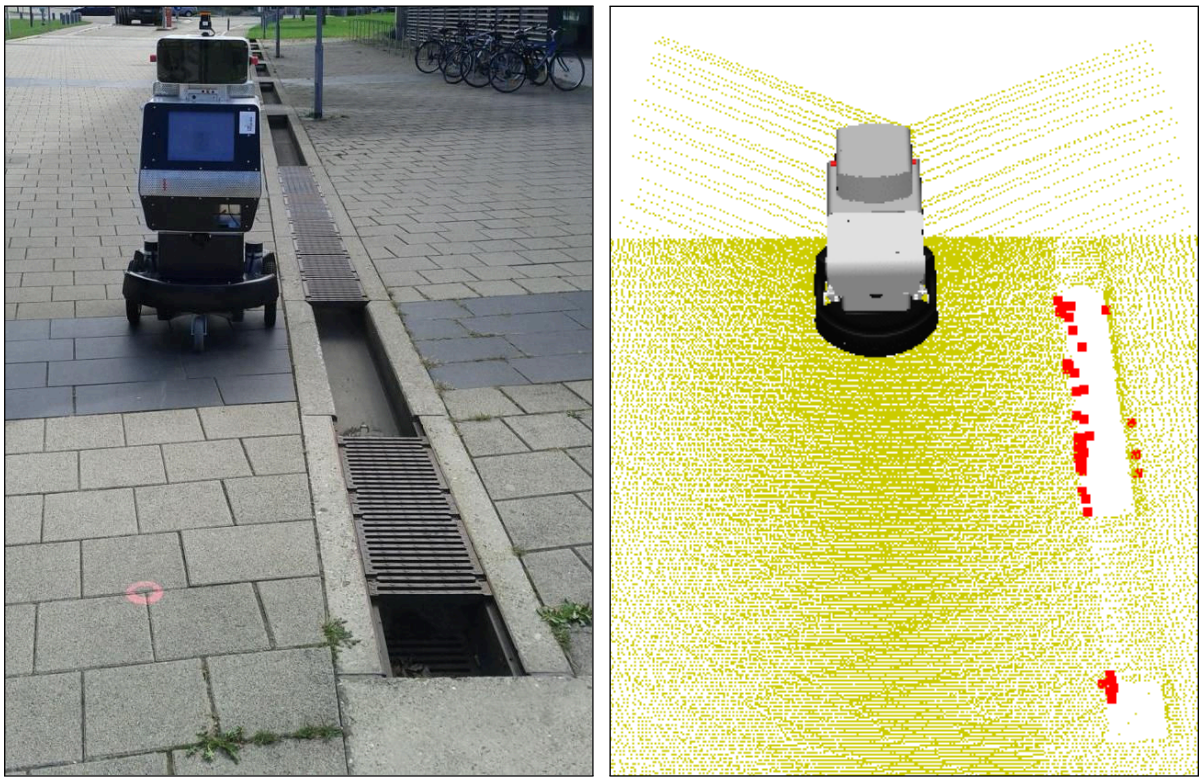
Vibration-Based Ground Evaluation

So far, our main concern was to decide whether an area is traversable versus non-traversable. Another important aspect is the smoothness of the ground plane itself. Many city centers have cobblestone pavement, especially in old European cities. Traveling with the maximum possible velocity of the platform results in vibration and might even damage the platform. Therefore, we control the maximum velocity according to the measurements of the gyroscopes of the IMU to avoid heavy shaking of the robot. The accelerometer measurements of the IMU tend to suffer from noise and seem to be less reliable for our purpose. We check the angular velocities for roll and pitch provided by the IMU in a 50 ms time frame. If the angular velocity for roll or pitch reports a value over a given threshold, we decrease the maximum permitted velocity in a stepwise manner. Since our laser range finders are not able to measure the smoothness of the ground surface accurately enough to decide how fast the robot can drive, we increase the maximum permitted velocity after a certain time again and repeat the described method. Figure 8.11 illustrates the behavior of our method over a 3.2 km long autonomous run. We selected examples for four different ground surfaces. As can be seen, the maximum permitted velocity is lowest for cobble stone pavement.

8.2.6 Path Planner

Planning a kilometer long trajectory in an efficient manner is an essential capability for a navigation system operating at city-scale. We approach this in a hierarchical fashion by considering three different levels of abstraction. On the highest level, we calculate a path on a topological graph that can be computed from our map data structure. One level below, we compute waypoints with Dijkstra's algorithm on the local maps attached to the nodes in the topological

(a)



(b)

Figure 8.10: Three examples of typical traversable structures that might be reported as non-traversable are shown in **(a)**. The output of our method for an example case is shown in **(b)**. The small canal is reliably detected as negative obstacle, whereas the gutters are classified as traversable.

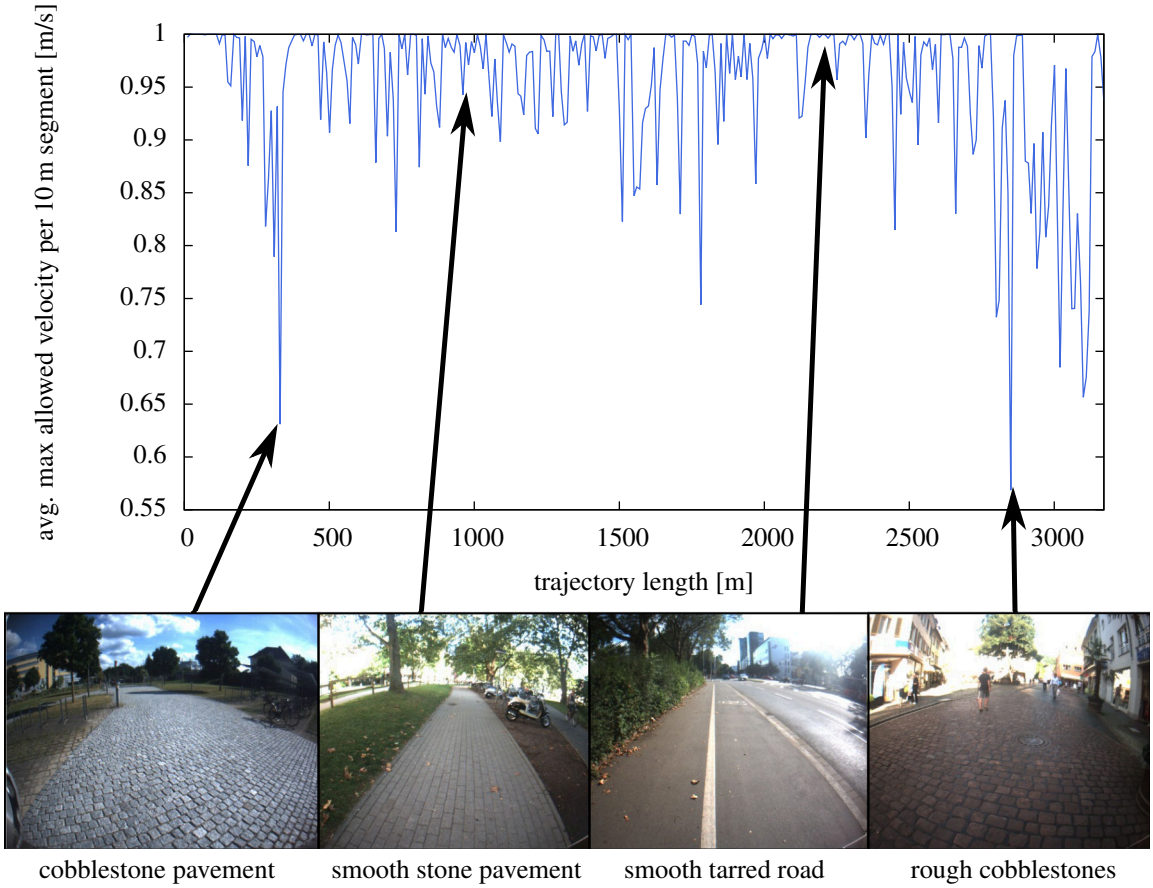
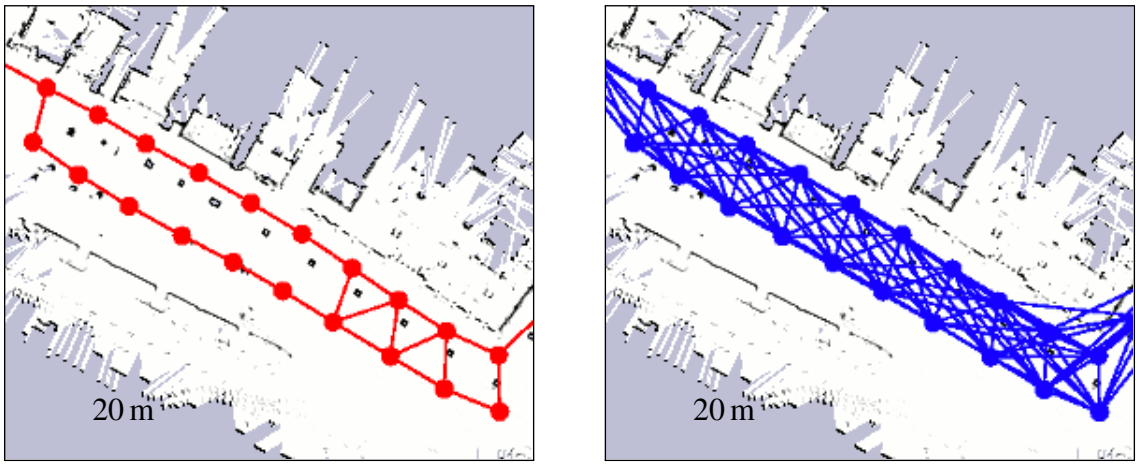


Figure 8.11: The plot at the top of this figure shows the maximum allowed velocity of the robot during a 3.2 km long autonomous run. This maximum velocity is determined based on the vibration measured while driving and therefore depends mostly on the ground surface. The pictures at the bottom show example images captured at the corresponding places of the trajectory. The left-most and right-most images are examples of rough surfaces that led to a reduced maximum velocity, whereas the two middle images show examples of smooth surfaces that allow a high maximum velocity.

graph. The basic motion commands to reach a waypoint are planned by a low-level planner as proposed by Rufli et al. (2009). Furthermore, the low-level planner uses the information provided by the traversability analysis module and especially the information about dynamic obstacles, as described in Section 8.2.5, to plan collision free paths. The planner predicts the motion of dynamic obstacles according to the velocity estimate of the tracker within the planning horizon. The intermediate level planner also uses the information of the traversability analysis module to avoid that a sub-goal for the low-level planner is not reachable. One disadvantage of this architecture is that the computed paths are not optimal. According to Konolige et al. (2011), who proposed a comparable planning approach, the computed paths are around 10 % longer compared to the optimum. Especially in highly populated city-centers the shortest path might often not be executable due to the dynamics of the people around the robot. Despite loss of optimality in path length, our approach can compute kilometer long paths efficiently.

The topology of the graph-based map structure reflects only found loop closures and not the true connectivity information of the environment. A SLAM algorithm needs a certain overlap between two scans to reliably detect loop closures, whereas the planner needs the information about which nodes are reachable in a local map, which are rendered from many scans. Therefore, our planner computes a new topology $\mathcal{T}$ for the full graph. Every node $\mathbf{x}_i$ in the graph has coordinates in the world frame and a 2D traversability grid map, which represents the collected



(a) **(b)**

Figure 8.12: Difference between a SLAM graph and a topology graph of the planner module. A part of a SLAM pose-graph is shown in **(a)** and the corresponding topology graph generated by the planner module is shown in **(b)**. Typically such a topology is denser connected since the connectivity is computed on local sub maps, whereas the SLAM pose-graph is computed on single scans.

knowledge about traversability of a cell. This information was computed by the traversability analysis module on the data acquired to compute the map. Furthermore, we store a cost value in each cell according to the cost of driving from the center at $\mathbf{x}_i$ to the corresponding cell. The costs are computed as weighted sum of the free space around the cell and the minimal travel distance to reach that cell. In that way, we increase the costs close to obstacles and penalize paths that are close to obstacles. Since costs are computed w.r.t the center node, we can pre-compute them with Dijkstra's algorithm. Furthermore, we also get the information out of the procedure if a cell is reachable from the center.

In our topology $\mathcal{T}$ we connect two nodes with an edge if we can reach a node in the traversability map of the other node. The cost of an edge is defined by the costs computed on the local maps for reaching the other node. The computed costs provide a more accurate description of an environment compared to just computing the straight-line distance. If a node cannot be reached from another node we insert an edge and set the cost to infinity. The costs for all edges can be pre-computed and reflect the true distances in the environment. Fig. 8.12 illustrates the difference between the spatial constraints $\mathcal{C}$ created by our SLAM module in (a) and the topology graph $\mathcal{T}$ constructed by our planner in (b). The topology graph of the planner is much denser connected than the map graph with the loop-closures generated by the SLAM system. The loop-closure constraints are generated on single laser scans, whereas an edge in the planner topology tells that a node is reachable in a local map, which typically consists of many scans.

From time to time it might happen that a route is blocked, for instance, due to a parking car or a novel construction site. In such cases, an autonomous navigating robot should detect a temporarily blocked route and find another path to the desired goal. The planning module handles such cases by identifying all edges in the topology that are not traversable given the data of the traversability analysis module. The cost of every blocked edge is then temporarily set to infinity. As a result, the high level planner will search for an alternative path to the goal location on the topology $\mathcal{T}$.

In the following, we will describe the interaction of the different levels to plan a path from the current pose of the robot to a goal location. First, we need to find the nodes in $\mathcal{T}$ that

correspond to the current pose and the goal location. We can identify those nodes by searching through the neighborhood of start and end poses and select the node with the smallest cost value for the corresponding location in its local map. Once we have this information, the high level planner performs an A* search on the topology $\mathcal{T}$. The result of this search is a good approximation of the A* search on a full grid map but is computed an order of magnitude faster. The result consists of a list of waypoints towards the goal. Since all points lie on the topology, a path that strictly follows these sub goals might be far away from the optimal solution. Therefore, the intermediate planner selects the farthest waypoint that is reachable in the current local map and applies Dijkstra's algorithm to plan a path to that waypoint. The intermediate planner gives the farthest away waypoint of the planned intermediate path to the low-level planner and the low-level planner computes the motion commands to execute this path. Both, the intermediate planner and the low-level planner are aware of the static obstacles provided by the online traversability analysis.

8.2.7 Watchdog

For a real-world system it proved to be useful to have a module that monitors the multitude of the other modules for errors and can restart them if something goes wrong. For this purpose, all our modules perform frequent consistency checks. For example, the hardware drivers check if they still receive data from the sensors as well as the platform and broadcasts status messages regularly. The watchdog module receives regular updates (typically once a second) from every module. If a module reports a failure or if its alive-messages time out, the watchdog module kills the process and restarts it. Modules that rely on a history, like the localization, store their current state regularly in a permanent manner, so that they can directly resume their work from a recent state when being restarted. Should the watchdog not be able to fix a problem by restarting a module, meaning it repeatedly fails after the restart, the watchdog will report the failure and, if needed, can enforce an emergency stop. The watchdog communicates with the operator over a graphical user interface, whose information is also available remotely, and plays sounds when problems appear.

This module does not have an explicit scientific background but serves as a handy solution for robots in the real world. It would be unnecessary in a system where every module and each of its third party dependencies work perfectly. But in practice, the watchdog module proved to be essential for long-term autonomous navigation, especially regarding the robust operation of certain hardware components. Since we logged whenever a module was reseted during our tests, we were able to carry out long-term experiments and to analyze the log-files afterwards to identify the faulty software parts. Instead of repeating the same experiment several times until everything works completely fine.

8.3 Evaluation

In this section, we describe a set of experiments in which we evaluated our system. The map used to carry out the experiments was obtained by driving the robot along a 7.4 km long trajectory. The map covers the area between the Faculty of Engineering of the University of Freiburg and the city center of Freiburg. Using this map, we carried out a series of navigation experiments. Besides several smaller tests, we performed six extensive navigation experiments during which the robot navigated from our campus to the city center and back. Figure 8.13 depicts an example trajectory with images taken along the trajectory. In these experiments, the robot traveled an overall distance of around 20 km and in three cases manual intervention was required.

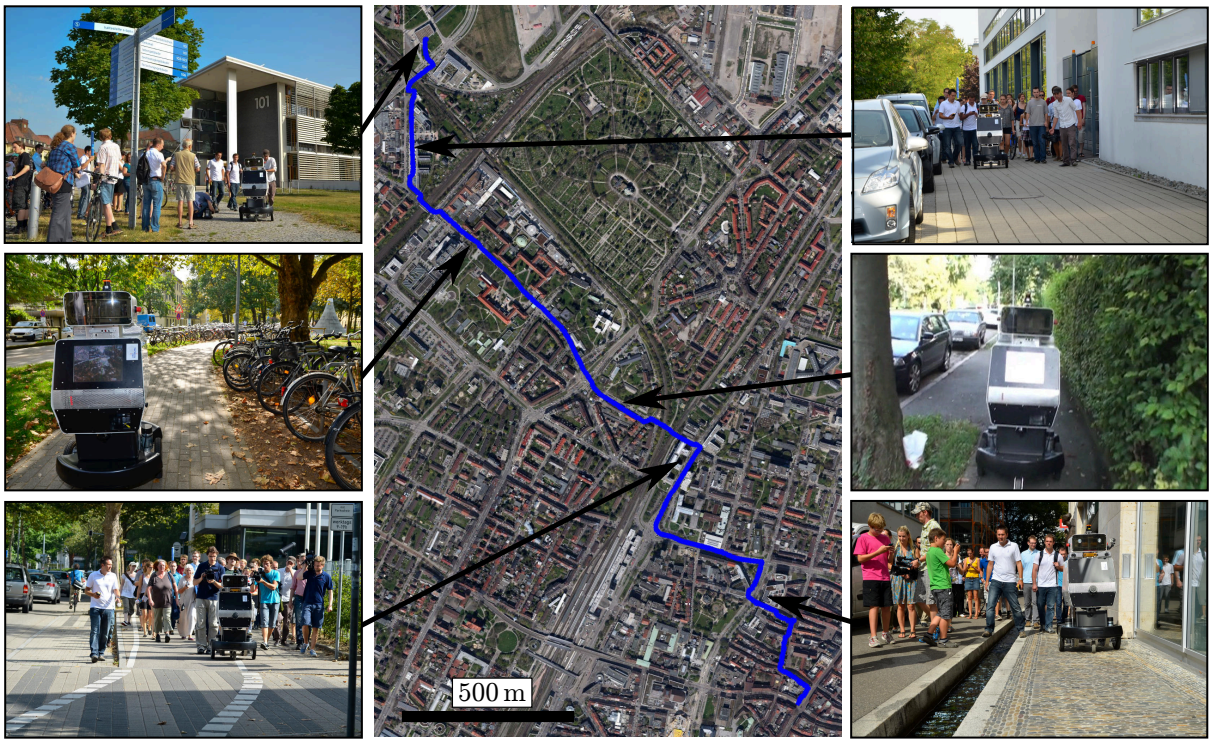


Figure 8.13: This figure shows an areal image (aerial image © Google with the trajectory taken by the robot and images of six example locations taken along the trajectory.

In addition to a localization failure discussed below in more detail, the robot once got stuck in front of a little bump and at another occasion was manually stopped because of an obstacle that we believed not being perceivable by the robot.

The final experiment was announced widely to give the public and the press the opportunity to see whether state-of-the-art robotic navigation technology can lead to a mobile robot that can navigate autonomously through an urban environment. This event attracted journalists from both TV and newspapers and led to a nationwide and international coverage in top-media. During all of our experiments, at least one human operator accompanied the robot. The task of the operator was to monitor the robot and to stop it by triggering the wireless emergency stop in case of a dangerous situation. Additionally, the robot was not allowed to cross streets without asking for permission. The operator had to confirm that it is safe to cross the street by pressing a button on the screen of the robot.

8.3.1 Localization

Whenever a robot navigates within an urban environment, the measurements obtained from the sensors of the robot are affected by the people surrounding the robot. As the localization algorithm is one of the core components of our system, we analyzed how the occlusions in the range data caused by people partially blocking the view of the robot affected the localization performance.

Fig. 8.14 depicts the fraction of valid range readings, i.e., readings smaller than the maximum range of the laser scanner, and the number of beams that match to the map for one of the large experiments mentioned above. Here, we regard a range reading as matching to the map if the distance between the measurement and the closest point in the map is below 0.2 m. The plot depicts several interesting aspects. A small fraction of valid beams indicates that the

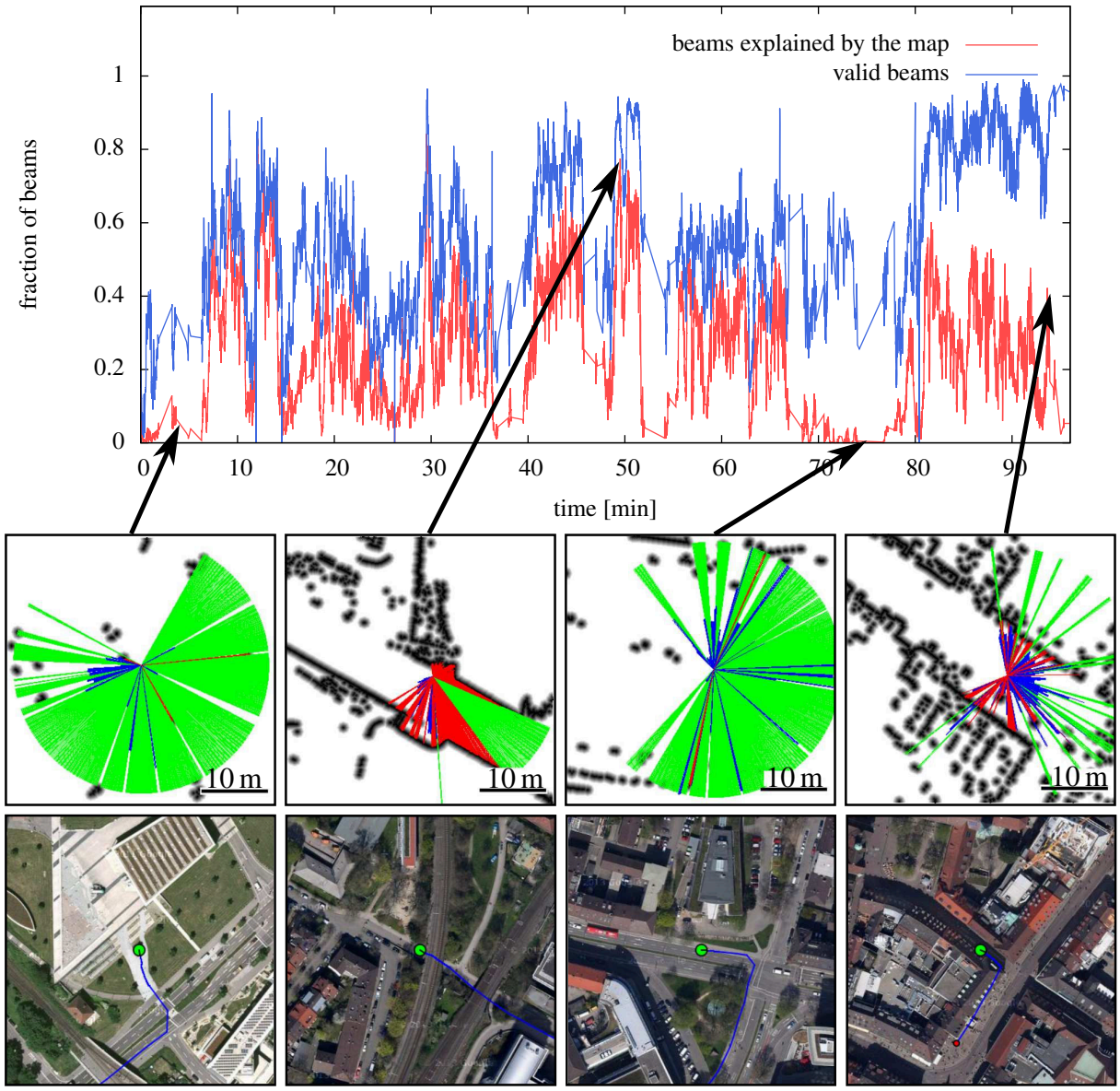
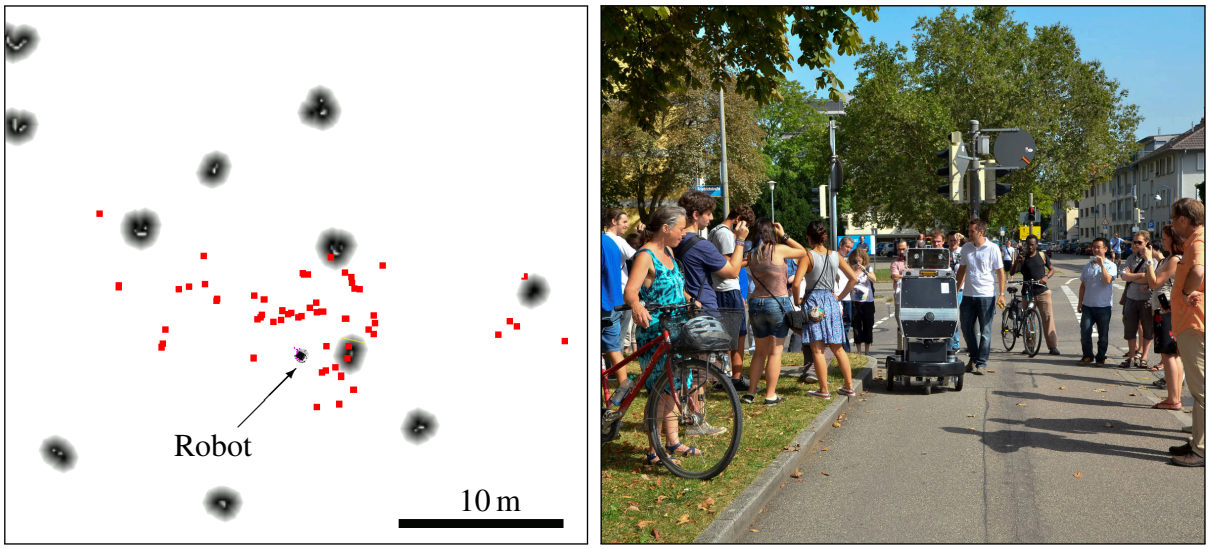


Figure 8.14: This plot shows the fraction of valid beams returned by the laser range finder and the fraction of beams that can be explained by the map of the environment. We regard a beam as valid if it reports a range reading shorter than the maximum range of the scanner. We classified a range reading as being explained by the map if the distance between the end point of the range reading and the closest occupied cell in the map is below 0.2 m. Furthermore, we show four different examples along the trajectory with colored beams on a distance map. Green beams correspond to maximum range readings. For a better visibility we cropped the beams after 19 m. Blue beams show valid range measurements that are not well explained by the map and red beams indicate measurements that can be explained by the map. The first example shows the robot in an open space where only a few trees are perceivable by the robot. In the second example the laser scan is well explained. The third example shows a situation a few minutes before the localization failure. Only a small fraction of the beams is explained by the map due to the open terrain and the presence of humans around the robot. The remaining observable structure is mainly parallel to the driving direction and does not constrain the position estimate in that direction. This led to the localization failure after around 78 minutes. The last example shows a densely crowded scene in the Freiburg downtown area. Due to the height of the laser, major parts of the surrounding buildings are visible and enable our robot to successfully localize itself (aerial images © Google).



(a) **(b)**

Figure 8.15: Background information for the localization failure during an autonomous run to Freiburg downtown. The 2D distance map is shown in **(a)**. As can be seen, there are only few localization features (shown in black) around — mostly tree trunks — and nearly all laser observations (red points) mismatch the provided model. Picture **(b)** shows that the robot is almost completely surrounded by people.

robot is navigating within open regions where only a small amount of structure is available to the robot for localizing itself. This can be seen in the first and third examples where most of the laser beams report maximum range (green). Furthermore, the difference between the number of valid beams and the number of beams that match to the map indicates that the view of the robot was partially blocked. In open areas with only few landmarks, it becomes critical if most of the visible landmarks are occluded for a longer time. In contrast, the second example shows a situation in which most of the laser scan is well explained and only few pedestrians are around the robot. The last example shows a densely crowded scene in the Freiburg downtown area. Due to the height of the laser, major parts of the surrounding buildings are visible and enable our robot to successfully localize itself.

In this experiment, the autonomous run was interrupted twice. In the first incident, the robot's wireless emergency stop button was pressed unintentionally, thereby being a human mistake. In the second case, a localization error occurred after around 78 min. As can be seen in Figure 8.14 between minutes 70 and 78 the robot traveled 200 m in an area with a very small amount of features while being surrounded by many people, as depicted in Figure 8.15. This mixture of very few relevant features in the map (shown in Figure 8.15(a)) and the fact that the robot was driving for an extensive distance while receiving mostly spurious measurements led to an error in the position estimate of around 2 m. This caused problems in negotiating a sidewalk after crossing the street. It made the robot stop and required us to manually re-localize the robot.

In other instances, sharing the same characteristics, for example, around minute 37 and around minute 52, the robot only drives substantially lower distances of 100 m and 50 m without meaningful sensor measurements. In both situations, the system is able to overcome the problem because it receives relevant information early enough again.

Furthermore, we analyzed a similar trajectory of the robot carried out during nighttime. At night, typically a smaller number of people are around and measurements are less frequently occluded. Hence, the offset between the number of valid beams and the number of beams

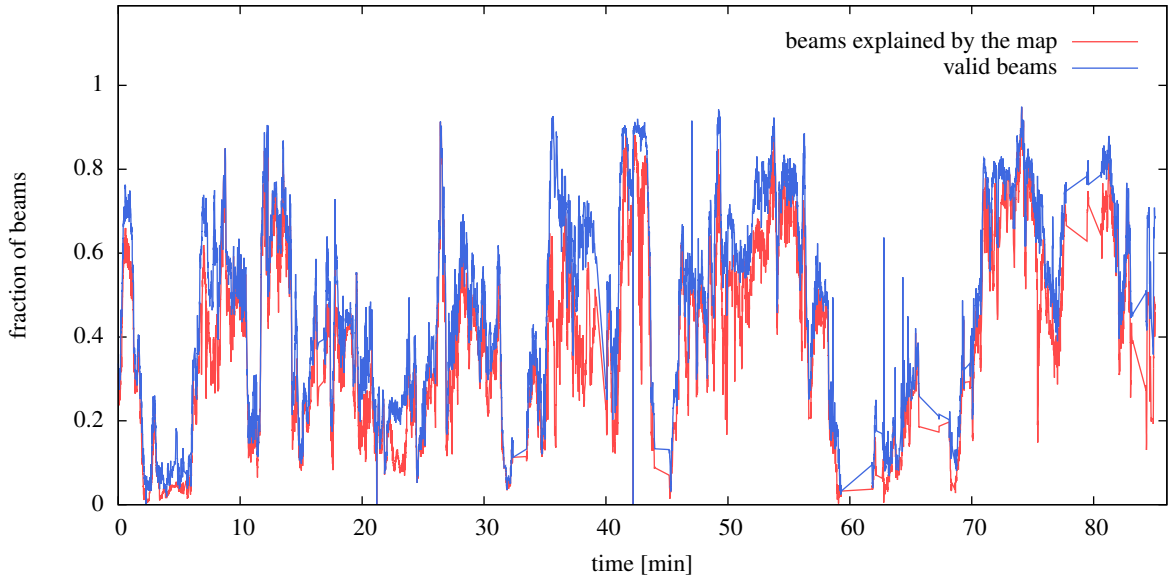


Figure 8.16: In this figure we present a similar plot to Fig. 8.14 for a dataset acquired during an autonomous run in the night. The environment is almost static and therefore most valid beams are well explained by the model. The robot traversed the same region as shown in the previous Figure around minute 60.

matching to the map is small all the time. In this experiment, the robot successfully reached its goal location without any problems and along a slightly different path of 3.5 km length. Visual inspection of the trajectory on the map revealed that the localization pose is much more accurate than the GPS solutions we had available. As far as we could observe, the trajectory was estimated correctly throughout the whole experiment.

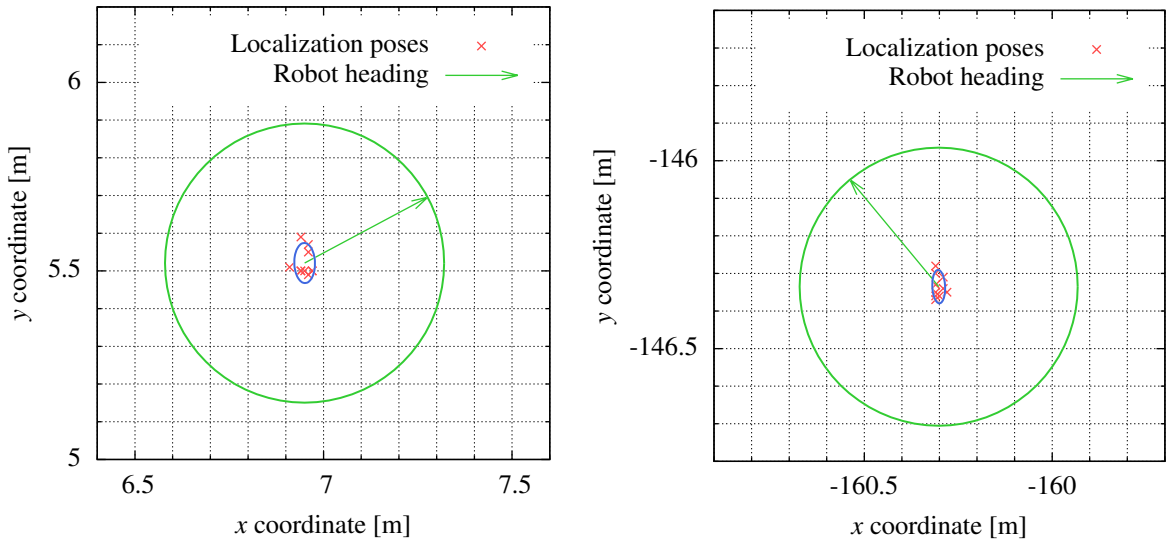
The global accuracy of the localization module is difficult to evaluate since city scale ground truth positions for a mobile robot are hard to acquire. Note that GPS in general is not accurate enough in urban canyons and without additional ground stations. Therefore, we decided to evaluate the localization accuracy locally with respect to two specific real-world positions. While the first position is nearby building 079 on our campus where the robot can observe a large portion of the building around, the second position on the open area in front of building 101 result in only a small portion of usable range measurements. We drove the robot ten times onto each marked position and evaluated the localization estimates. Figure 8.17 shows an exemplary marking on the ground and the robot standing on the marked position. After each visit of the goal location we drove at least 15 m before we approached the location for another pass.

Using the recorded estimates we computed the mean position and the corresponding standard deviation for the two locations on our campus. The standard deviation is an indicator for the localization accuracy but also suffers from small position errors since we cannot drive manually to a target location with millimeter accuracy. Figure 8.18 shows the collected localization estimates for both locations along with the 3σ confidence ellipse in blue and the robot footprint in green.

For the location next to building 079 the translational standard deviation was 0.035 m with a rotational standard deviation of 0.82°. For the other location near building 101 the standard deviations were 0.027 m for the translational error and 0.87° for rotational error, respectively. The resulting localization accuracy is more than sufficient to carry out the described navigation experiments. Some of the paths the robot regularly took during our experiments were only 20 cm wider than the width of the robot, thereby enforcing a minimum accuracy for the localization

(a) **(b)**

Figure 8.17: This figure shows an exemplary marking on the ground on an open space close to building 101 on our campus **(a)** and the robot standing on the marked position **(b)**. We were able to manually drive the robot on the marking with a precision of approximately 1.5 cm.



(a) **(b)**

Figure 8.18: In this figure, we show the localization estimates for several stops at the same location for two different places on our campus. The plot **(a)** illustrates the localization estimates for the reference pose near building 079 whereas **(b)** depicts the corresponding plot for the reference pose near building 101. The green circles show the approximate footprint of the robot and the 3σ confidence ellipse around the mean position are plotted in blue. The grid has a cell size of 10 cm.

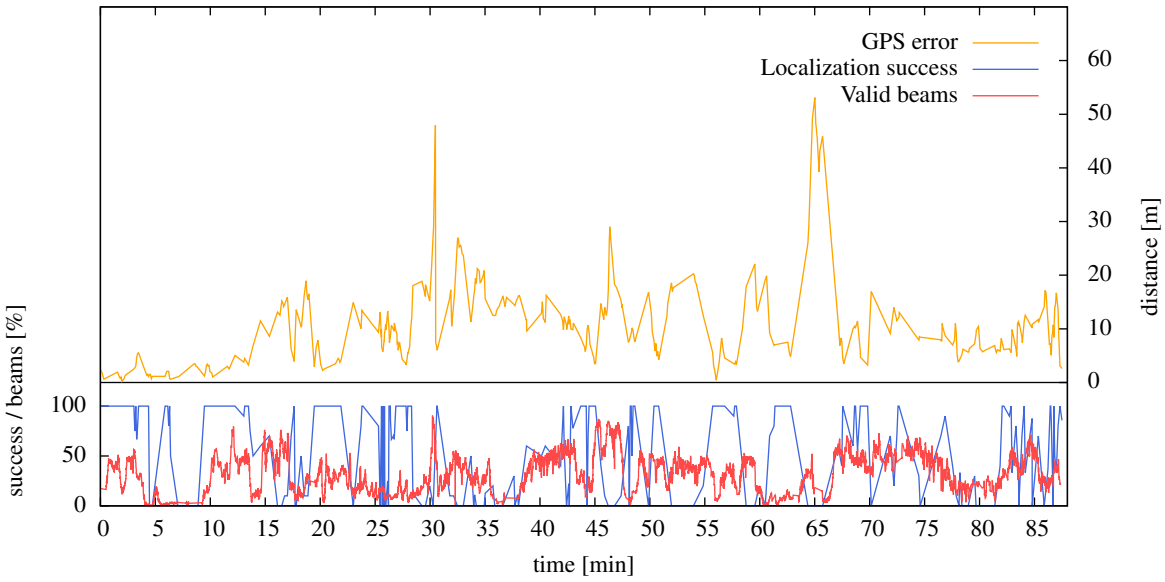
pose of 10 cm.

8.3.2 Global Localization

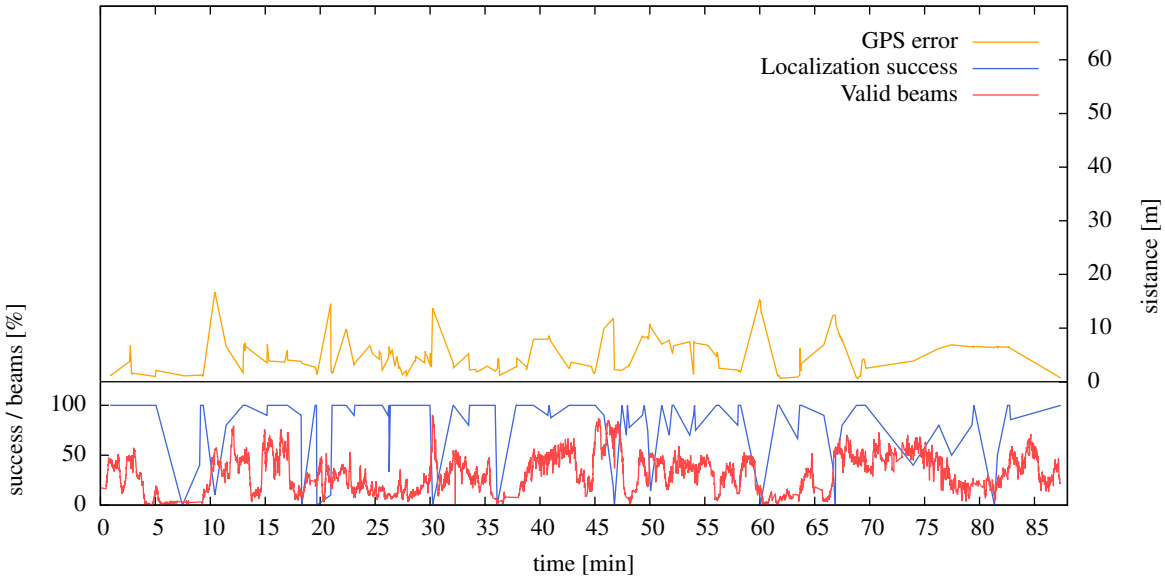
In this section, we evaluate the success rate of our GPS-based MCL initialization scheme. Therefore, we recorded one dataset in which we had two different GPS devices mounted on the robot. In addition to the Trimble GPS Pathfinder Pro GPS device, we mounted a low cost consumer GPS, i.e., a standard SiRF III, as baseline. For both devices, we selected GPS measurements every 15 s on the dataset when available. For every 15 s time window we initialized the MCL algorithm 10 times with the corresponding coordinates and let it run until either the filter converged or the time limit of 15 s was met. As ground truth, we considered the poses of the localization module running while acquiring the data and considered pose estimates closer than 1 m as success. The chosen ground truth is not perfect but a small difference to the ground truth indicates that the particle filter converged close to the previously estimated position of the robot. In order to gain insights in possible failure cases, we also evaluated the GPS accuracy with respect to the ground truth and the number of beams that match to the map on the ground truth trajectory. Throughout all our experiments, we used 1,000 particles and a standard deviation of 5 m for both GPS devices. Fig. 8.19 shows the resulting quantities for localization success, GPS error, and valid beams for both GPS devices. As can be seen in the upper plot, the pose estimates of the low-cost GPS (orange) can be up to 50 m away from the ground truth pose. In comparison the second GPS device results in errors below 20 m. The localization success per time slot is plotted in red and the number of beams that match to the map is plotted in blue. The overall localization success was 53.3 % for the low-cost GPS and 83.9 % for the highly accurate GPS device. Of course the accuracy of the GPS measurements has a major impact on the localization success but also the amount of information available at the true pose in the map. As can be seen in the lower plot in Fig. 8.19, between minutes 5 and 10 the GPS measurement is around 1 m close to the ground truth pose but the localization success is 0 %. This is mainly caused by the fact that there is not much laser-based information available, as can be seen in the red plot. Overall, a successful initialization for the localization algorithm highly depends on the accuracy of the GPS measurement as initial guess and the information provided by the laser range finder.

8.4 Discussion and Lessons Learned

The navigation system described in this paper has been implemented for and on the robot Obelix characterized in Section 8.1. The design of a platform typically has a substantial influence on the algorithms needed for accomplishing the desired task. Given the navigation task of Obelix, his structure definitely influenced the design of certain software components. For example, its almost circular footprint makes the planning of paths easier, as only a two-dimensional path needs to be computed (see Fig. 8.1). Additionally, the specific mounting of the range scanners, that resulted in the fact that three-dimensional structures could only be sensed when the robot moves, has an influence on collision avoidance routines. We are still convinced that these platform-specific design choices are not critical and that the mixture of components we realized is relevant for accomplishing this challenging navigation task and is sufficiently generic to be easily transferable to other robotic platforms, such as robotic wheelchairs or transportation vehicles in cities.



(a) SiRF III



(b) Trimble GPS Pathfinder Pro

Figure 8.19: Evaluation of the GPS accuracy versus the global localization success. The plot shown in **(a)** shows the results for initializing the localization algorithm based on the low-cost GPS device. As can be seen, the pose estimates of the low-cost GPS (orange) can be up to 50 m away from the true pose. The localization success per time slot is plotted in red and the number of beams that match to the map is plotted in blue. The results for the highly accurate GPS device are plotted in **(b)**. The maximum GPS measurement error for this device is below 20 m.

8.4.1 Lessons Learned

- Pedestrians, in particular children, in our experiments sometimes intentionally blocked the path of the robot. While this did not cause any collisions, the robot got stuck in such situations because it is not as agile as a human. According to our experience with such cases, a planning framework should actively negotiate a path with the humans around in highly crowded scenarios instead of solely focusing on collision free trajectories.

- Besides tools for automatic data logging and debugging, we found it handy to have the option to selectively de-activate various aspects of the autonomous system for testing and to easily change parameters on-the-fly. One typical example is the adjustment of the maximum allowed velocity during experiments. Our tools also allowed us to manually control the robot while inspecting the results of different aspects of the system, e.g., the tracking of dynamic objects and their impact on the planned path.

- In our system, the modules that have to fuse data over time operate in a local frame, which is given by the odometry of the robot. Clearly, this frame drifts over time but it is locally smooth, which cannot be guaranteed for position estimates in a global frame, such as the estimate of the localization algorithm. The usage of a local frame allowed us to estimate a consistent map that fuses multiple measurements (see also Moore et al. (2009)).

- Analyzing single range readings for obstacles and fusing these individually detected obstacles in a local frame proved to be very efficient. Furthermore, compared to analyzing point clouds, which are accumulated over time, it reduces the risk of detecting non-existing obstacles that may be caused by inaccurate calibration or imprecise position estimates of the robot.

- We found that specifying goal locations such as the stops of a guided tour as global coordinates (latitude/longitude) should be preferred compared to storing those location in local map coordinates. Since our mapping algorithm incorporates the GPS prior, we are able to project between map coordinates and the GPS frame. Without considering the GPS measurements, the coordinates in a map frame depend on the initial location of the robot. Consequently, the local map coordinates are subject to change while the GPS frame remains fixed.

- Throughout the project, we developed a vast set of visualization tools. These tools and the fact that we recorded all the data (sensor data and messages that got exchanged between the modules) allowed us to carefully analyze the behavior of the algorithms in practice.

- We furthermore realized that other aspects are rather challenging, as, for example, curly leafs on the ground look similar to little rocks in the range data. Whereas the robot can easily drive over leafs, rocks can actually have a substantial effect on the platform itself. This is a general dilemma which cannot be solved alone with perception algorithms. Therefore we decided to drive around all potential obstacles. In future systems this could be addressed by designing a robot platform that can measure forces on the bumpers and that is allowed to use a small force to overcome obstacles.

- In an early test of our system, we encountered a failure before the robot had to cross a traffic light. The robot was driving on a steep downward facing ramp leading from the sidewalk onto the street. This lead to laser beams hitting the street surface approximately 4 m in front of the robot that were wrongly classified as an obstacle. As a consequence,

Figure 8.20: Examples of dynamic 3D obstacles, which pose substantial challenges for the navigation system.

we revised our system to incorporate the attitude of the robot to identify measurements that are likely to be caused by the laser beams observing the ground.

8.4.2 Limitations of our Approach

- The most critical aspect of the entire navigation task was the crossing of roads or all situations in which the robot potentially had to interact with fast-driving cars. The two dimensional laser sensors do not allow to reliably sense cars approaching the robot in all situations. In our demonstration, the robot had to ask for permission to cross streets or other safety-relevant areas, which we marked manually in the map of the robot.

- The obstacle detection operates on rigidly mounted laser range finders (see Fig. 8.2). While this has the advantage that no moving parts are involved, the rigid mounting of the sensors allows the robot to observe objects such as curbs only at a range of approximately 1.5 m. When navigating in a previously unknown region, the robot has to frequently update its plan to move alongside a curb.

- Dynamic objects that are not visible in the horizontal range data, such as pets or other animals like pigeons or ducks (see Fig. 8.20) currently lead to a sub-optimal path execution, because they can only be observed in the close vicinity by the robot.

- The sensor setup has blind spots, for example, overhanging obstacles above a certain height in front of the robot are not visible. The only situation we encountered, in which this was critical was a bike handlebar in an unfortunate position. Most likely an update of the platform should incorporate revised sensor hardware.

8.5 Related Work

The problem of autonomous navigation in populated exhibition areas has been studied intensively in the past. Among the early systems are the robots RHINO (Burgard et al., 1998) and Minerva (Thrun et al., 1999), which operated as interactive mobile tour-guides in crowded museums. An extension of this tour-guide concept to interactive multi-robot systems is the RoboX system developed by Siegwart et al. (2003) for the Expo'02 Swiss National Exhibition and the TOURBOT system by Trahanias et al. (2005). Gross et al. (2009) installed a robot as a shopping assistant that provided way-finding assistance in home improvement stores. Although these systems are able to robustly navigate in heavily crowded environments, they assume that the robots operate in a relatively confined planar area and furthermore are restricted to two-dimensional representations of their environment.

Relatively few robotic systems have been developed for autonomous navigation in city centers. The concept closest to the one described in this chapter probably is the one of the Autonomous City Explorer developed by Bauer et al. (2009). In contrast to our system, which operates completely autonomously, the Autonomous City Explorer relies on interaction with humans to get the direction where to move next. The Autonomous City Explorer builds local maps and does not autonomously plan its path from its current position to the overall goal location. Other researchers developed robots that monitor pollutions in city environments and are designed to improve trash collection (Reggente et al., 2010). A further related approach has been developed in the context of the URUS project (Sanfeliu et al., 2010), which considered urban navigation but focused more on networking and collaborative actions as well as the integration with surveillance cameras and portable devices. Another result of the URUS project (Trulls et al., 2011) considers urban navigation in pedestrian-only environments. Compared to our city-scale navigation system their approach operates at a smaller scale in terms of the dimension of the map and the area covered while navigating autonomously. Due to the pedestrian-only scenario, it does not in particular address navigating on sidewalks and crossing streets. Additionally, the system developed by Morales et al. (2009) allows a robot to navigate over pedestrian walkways. In contrast to our approach, their system relies on a list of pre-defined waypoints, which the robot follows to reach the desired goal location.

Also, the problem of autonomous navigation with robotic cars has been studied intensively. For example, there has been the DARPA Grand Challenge Seetharaman et al. (2006) during which autonomous vehicles showed the ability to navigate successfully over large distances through desert areas (Cremean et al., 2006; Thrun et al., 2006; Urmson, 2005). During the later DARPA urban challenge, several car systems have been presented that are able to autonomously navigate through dynamic city street networks with complex car traffic scenarios and under consideration of road traffic navigation rules (Urmson et al., 2008; Montemerlo et al., 2008; Rauskolb et al., 2008). Such competitions and demonstrations like the European Land Robot Trial (ELROB) have the potential to accelerate robotics research and to make the general public aware of the current state of the art (Schneider and Wildermuth, 2011). Recently, commercial self-driving cars (Google Inc.) have been developed and legalized to perform autonomous navigation with cars. In contrast to such approaches, which focused on car navigation, the system described in this paper has been developed to enable mobile robots to perform autonomous navigation in densely populated urban environments with many types of dynamic objects like pedestrians, cyclists, or pets.

A long-term experiment to evaluate the robustness of an indoor navigation system was recently presented by Marder-Eppstein et al. (2010). Their system used a tilting a laser range finder for navigation and for obstacle detection. In contrast to this system, our approach also

has a component for tracking moving obstacles to explicitly deal with the dynamic objects in highly populated environments. It furthermore includes a terrain analysis component that is able to deal with a larger variety of terrain.

While the approaches discussed so far focus on ground-vehicles, navigation with flying robots can be realized by applying the same paradigms but require specific adaptations to the demands of flying robots (Bachrach et al., 2011; Grzonka et al., 2012). An essential element of navigation with both, ground-based and flying robots, is estimating the pose of the vehicle, for example, by localizing the robot against a map by means of a particle filter (Dellaert et al., 1999). Navigating in an environment that is as large as a city makes high demands on the mapping and localization algorithms. Newman et al. (2009) presented an approach that combines vision and laser data along with topological and metric representations to estimate maps of urban environments. Although the robot is navigating outdoors in such scenarios, it cannot rely on external systems such as GPS because their signal might be partially or completely blocked. Therefore, the robot has to estimate its pose with the data of the onboard sensors and a map (Georgiev and Allen, 2004). Furthermore, the robot may perform specific actions to resolve ambiguities and consequently converge faster to a single hypothesis (Fox et al., 1998; Murtra et al., 2008).

Instead of estimating a map, which is subsequently used for localizing the robot, the data recorded during following a route can be utilized more directly. Furgale and Barfoot (2010) presented a teach and repeat system based on stereo vision. In a first step, the robot is manually driven along a route and creates sub-maps by visual odometry. Within the second step, the retraversal of the learned route, a localization algorithm allows the robot to follow the taught route also in urban settings. Royer et al. (2005) suggested a similar system based on monocular vision. Compared to our system, both approaches rely on a previously taught path to reach a specific goal location.

8.6 Conclusions

In this chapter, we presented a robust navigation system for mobile robots dedicated to perform autonomous navigation tasks in urban environments at city scale. We use an extended SLAM approach that utilizes prior knowledge from GPS measurements in a robust fashion. To improve the localization accuracy, we applied SSA as described in Chapter 3 on the local maps. The described map data structure decouples this optimization problem into a set of local maps instead of optimizing a several kilometer long trajectory. As a result, we can apply SSA more efficiently and keep the number of variables in the optimization problem at a manageable level. Furthermore, the map structure improves the memory efficiency of the localization and planning modules. Planning time is also substantially reduced due to the topology built from the map structure. Another important module we presented in this chapter is the terrain analysis. This module is essential for a save navigation and to successfully detect obstacles, to classify vegetation, and to track dynamic obstacles.

We integrated the full system on the mobile robot Obelix and performed large-scale field tests. One test was publicly announced, leading to a very challenging and crowded test scenario. The robot managed to autonomously navigate a path of over three kilometers to the city-center of Freiburg. This public demonstration was a good showcase for what we can do with state-of-the-art algorithms integrated on a mobile robot.

Chapter 9

Conclusions

Accurate and compact 2D or 3D models of an environment, as well as accurate object models, are essential for a wide range of robotic applications. In order to build models with a mobile robot, we have to deal with several challenges, introduced by sensor noise and pose uncertainties. Furthermore, we have to consider a large search problem to find and encode similarities in range data. In this thesis, we presented innovative methods to improve the accuracy of SLAM results, to learn compact models of the environment based on similarities in shape and texture, and to perform place recognition and city-scale navigation with accurate large-scale environment models. With the presented approaches, we extend the state-of-the-art in environment modeling and 3D perception and bring them closer together.

SLAM is an important pre-requisite for our methods. To deal with remaining uncertainties in the pose estimates and with accumulated sensor noise of typical SLAM solutions, we construct a joined optimization problem and efficiently solve it. Since we allow range scans to deform, it is important to accurately model the sensor noise characteristics. Therefore, we proposed a sensor model for laser range finders, which depends on the range of a measurement and the incidence angle to the surface. Furthermore, we proposed a sensor model for RGBD cameras, which accounts for the error in the measured disparity, caused by the quantization in subpixels. With the described method, we are able to create highly accurate, smooth models. The only remaining drawback is the missing compactness of the models since all observed points are stored in the model.

As we have shown in this thesis, it is possible to use similarities of surfaces to reduce the amount of data we have to store. Especially in human made environments many structures have a similar shape. Therefore, we presented a method that describes surfaces based on a dictionary of sample surfaces. Each sample surface is represented as a small point cloud of a fixed volume. In this context, the main challenge is the learning of a representative dictionary. For learning a set of surface primitives, we proposed a greedy learning method that uses the Bayesian information criterion to tradeoff accuracy and compactness. With this technique we are able to represent environments with only few surface primitives. One advantage of this method is that we have only the point deviation σ as a parameter in the likelihood function and the number of dictionary entries is automatically computed based on the BIC.

The main disadvantage of the presented surface primitive models is that we cannot easily extend them with textured 3D data without loosing compactness. Due to texture information, we would need to store many different primitives for similar shapes with different textures. To avoid this problem, we proposed to split shape and texture into two different channels. As consequence, we had to discretize the local surface to do this in a meaningful fashion. Furthermore, the amount of possible variations in texture made it also necessary to use a more flexible coding

scheme for the dictionary learning. We extended K-SVD to better deal with occlusions and missing data to learn smaller and more expressive dictionaries. Also from a semantically point of view such models are interesting since we demonstrated that we can efficiently search for similar shapes in the sparse codes of the surface model.

So far, we did not consider that similarities in shape or texture can appear at an arbitrary scale. Therefore, we proposed a method that uses surface descriptions of different size and organized in a hierarchical fashion to represent an environment. In this way, we can represent similarities at different scales. The drawback is that we have to solve a challenging perception problem to decide at which scale a surface should be described. We proposed to approach this problem with a greedy method that uses the discretization error and the coverage of a surface description as heuristics to decide whether a surface should be described at a given level. The greedy selection starts with the largest allowed surface descriptions and represents the environment by stepwise reducing the surface description size. With the presented hierarchical organization scheme, our method learns even more compact models compared to SCSM at a comparable accuracy.

In place recognition, we would like to find the most similar location in a scan database given a query scan. This is a classic perception problem and again we are searching for 3D scans with similar surface configurations. The presented system reliably detects places it has in its database and furthermore estimates accurate transformations between the query scan and the best database scan. Furthermore, we can use the described dictionary learning techniques to speedup place recognition tasks and generate loop-closure constraints for graph based SLAM systems. In this way, our methods can also contribute to the underlying map estimation.

So far, we presented several essential building blocks to learn accurate and compact environment models. One important application for environment models is to successfully navigate with a mobile robot. Therefore, we developed a robust navigation system for the robot Obelix to operate in highly crowded city centers. The accuracy of GPS devices was not sufficient to successfully navigate on sidewalks in all cases. Therefore, we decided to build accurate large-scale environment models and localize the robot in the corresponding model. This has the further advantage that the robot can operate in GPS-denied environments, such as buildings or shopping centers. Due to the scale of the modeled environment, we developed a memory efficient topological representation with locally attached maps and maintained only a small part of the environment in the main memory of the robot. Furthermore, we developed a robust localization module working on the model structure that had to deal with the dynamics of highly crowded scenes as well as open areas with only few geometrical features. The presented navigation system worked reliably in extensive large-scale field tests, where the robot Obelix had to autonomously navigate over large distances.

As we have seen, building accurate and compact environment models is a difficult challenge for mobile robots. In this thesis, we presented approaches to improve the environment modeling capabilities of mobile robots by showing that:

- We can estimate very accurate models by considering sensor noise and pose uncertainties at the same time.
- We can deal with the massive amount of data in accurate 3D models by encoding similarities in shape and appearance.
- We use 3D perception techniques to provide a semantically richer representation.
- We encode similarities at different scales with hierarchical models.

- We can reliably detect places the robot has visited before based on 3D range data.
- We can build a fully integrated robotic navigation system to successfully navigate in large-scale urban environments.

All our algorithms were evaluated on real-world data and achieved very promising results. The underlying perception problems are hard to solve but we believe that the presented methods will enable mobile robots to build better models and therefore will provide better services for all humans and our society.

9.1 Future Work

Due to the nature of scientific work, there are still many challenges left and we would like to address some of them in the future. The joint optimization of sensor poses and surface points of dense RGBD streams or of large-scale models is hard to solve efficiently since we can barely maintain all sensor information in main memory. Therefore, we would like to efficiently decouple parts of the problem and solve them out of core. In the method described in Chapter 3, we only considered points and their normals and not the available color information. Accordingly, we would like to exploit the texture information to improve the data associations, leading to sharper textures in the resulting models.

Another very interesting idea is to enhance Sparse Coded Surface Models to deal with observable dynamics and update the model online. The sparse code descriptions of shape and texture could make it possible to track small changes over time. Besides dynamics, we could extend Sparsely Coded Surface Models with a time dependent component and compactly describe cyclic changes in shape or texture, introduced by seasons, with sparse codes. Furthermore, we could learn relations between textured surfaces and find surface subsets that correspond to entities like persons or other dynamic objects. Some other interesting extension is an online method to efficiently update a dictionary. Given we have a pre-computed dictionary and we encounter texture or shape that cannot be accurately described, it would be beneficial to efficiently enhance a given dictionary.

In the context of place recognition, a possible extension would be to find a place similar to the query scan directly in a Sparsely Coded Surface Model. As consequence, we could avoid to maintain a scan database in memory and just use the model to relate new observations to the previously observed data and utilize this information in a SLAM system.

List of Figures

Bibliography

M. Aharon, M. Elad, and A. Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Trans. on Signal Processing*, 54(11):4311, 2006.

V. Andersen, H. Aanæs, and J. A. Bærentzen. Surfel based geometry resonstruction. In *Theory and Practice of Computer Graphics*, pages 39–44, 2010.

A. Bachrach, S. Prentice, R. He, and N. Roy. RANGE - robust autonomous navigation in GPS-denied environments. *Journal on Field Robotics*, 28(5):644–666, September 2011.

J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. R. L. Whittaker. Ambler: An autonomous rover for planetary exploration. *IEEE Computer Society Press*, 22 (6):18–22, 1989.

A. Bauer, K. Klasing, G. Lidoris, Q. Mühlbauer, F. Rohrmüller, S. Sosnowski, T. Xu, K. Kühnlenz, D. Wollherr, and M. Buss. The autonomous city explorer: Towards natural human-robot interaction in urban environments. *International Journal of Social Robotics*, 1:127–140, 2009. ISSN 1875-4791.

H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2006.

O. Bengtsson and A.-J. Baerveldt. Robot localization based on scan-matching – setimating the covariance of for the icp algorithm. *Journal of Robotics & Autonomous Systems*, 44:29–40, 2003.

P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. ISSN 0162-8828.

P. Biber and W. Strasser. The normal distributions transform: A new approach to laser scan-matching. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2003.

L. Bo, X. Ren, and D. Fox. Unsupervised feature learning for RGB-D based object recognition. *Proc. of the Int. Symposium on Experimental Robotic (ISER)*, 2012.

M. Bosse and R. Zlot. Map matching and data association for large-scale two-dimensional laser scan-based slam. *Int. Journal of Robotics Research*, 27(6):667–691, 2008.

J. Brookshire and S. Teller. Automatic calibration of multiple coplanar sensors. In *Proc. of Robotics: Science and Systems (RSS)*, 2011.

W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1998.

W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardós. A comparison of SLAM algorithms based on a graph of relations. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

A. Censi. The achievable accuracy for range finder localization. *IEEE Trans. on Robotics*, 2009.

W. Chang and M. Zwicker. Global registration of dynamic range scans for articulated model reconstruction. *ACM Trans. Graph.*, 30:26:1–26:15, May 2011. ISSN 0730-0301.

Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1991.

L. B. Cremean, T. B. Foote, J. H. Gillula, G. H. Hines, D. Kogan, K. L. Kriechbaum, J. C. Lamb, J. Leibs, L. Lindzey, C. E. Rasmussen, A. D. Stewart, J. W. Burdick, and R. M. Murray. Alice: An information-rich autonomous vehicle for high-speed desert navigation: Field reports. *Journal on Field Robotics*, 23(9):777–810, September 2006. ISSN 0741-2223. doi: 10.1002/rob.v23:9.

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

M. Cummins and P. Newman. Appearance-only SLAM at large scale with FAB-MAP 2.0. *Int. Journal of Robotics Research*, Nov 2010. doi: 10.1177/0278364910385483.

B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. of the Conf. on Computer Graphics and Interactive Techniques*, pages 303–312. ACM, 1996.

P. de la Puente, D. Rodriguez-Losada, A. Valero, and F. Matia. 3D Feature Based Mapping Towards Mobile Robots' Enhanced Performance in Rescue Missions. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

F. Dellaert and M. Kaess. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *Int. Journal of Robotics Research*, 25(12):1181–1204, Dec 2006.

F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1999.

A. Doucet, N. de Freitas, and N. Gordan, editors. *Sequential Monte-Carlo Methods in Practice*. Springer Verlag, 2001.

M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Trans. on Image Processing*, 15(12):3736–3745, 2006.

EUROPA Project. The European robotic pedestrian assistant, 2009. http://europa.informatik.uni-freiburg.de.

R. Eustice, H. Singh, and J. Leonard. Exactly sparse delayed-state filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

N. Fairfield, G. A. Kantor, and D. Wettergreen. Real-time slam with octree evidence grids for exploration in underwater tunnels. *Journal on Field Robotics*, 2007.

U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4): 634–652, 1998.

S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, 24(3):544–552, 2005.

J. D. Foley, A. Van Dam, S. K. Feiner, J. F. Hughes, and R. L. Phillips. *Introduction to computer graphics*, volume 55. Addison-Wesley Reading, 1994.

D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Journal of Robotics & Autonomous Systems*, 25:195–207, 1998.

U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Trans. on Robotics*, 21(2):1–12, 2005.

P. Furgale and T. D. Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal on Field Robotics*, 27(5):534–560, 2010. ISSN 1556-4967.

A. Georgiev and P. Allen. Localization methods for a mobile robot in urban environments. *IEEE Trans. on Robotics*, 20(5):851–864, 2004. ISSN 1552-3098. doi: 10.1109/TRO.2004.829506.

Google Inc. Google self-driving car project, 2012. http://googleblog.blogspot.com.

K. Granström and T. Schön. Learning to close the loop from 3D point clouds. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

K. Granström, J. Callmer, F. Ramos, and J. Nieto. Learning to detect loop closure from range data. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based SLAM with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2443–2448, Barcelona, Spain, 2005.

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

G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based SLAM. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43, 2010. doi: 10.1109/MITS.2010.939925.

H.-M. Gross, H. Boehme, C. Schroeter, S. Mueller, A. Koenig, E. Einhorn, C. Martin, M. Merten, and A. Bley. TOOMAS: Interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

S. Grzonka, G. Grisetti, and W. Burgard. Towards a navigation system for autonomous indoor flying. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

S. Grzonka, G. Grisetti, and W. Burgard. A fully autonomous indoor quadrotor. *IEEE Trans. on Robotics*, 8(1):90–100, 2012.

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)*, pages 318–325, 1999.

R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

M. Hebert, C. Caillas, E. Krotkov, I. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 997–1002, 1989.

P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *Proc. of the Int. Symposium on Experimental Robotic (ISER)*, volume 20, pages 22–25, 2010.

P. Henry, D. Fox, A. Bhowmik, and R. Mongia. Patch volumes: Segmentation-based consistent mapping with RGB-D cameras. In *International Conference on 3D Vision (3DV)*, 2013.

Hokuyo Ltd. Scanning laser range finder utm-30lx/ln specification, 2012. http://www.hokuyo-aut.jp/02sensor/07scanner/download/products/utm-30lx/data/UTM-30LX_spec_en.pdf. Last accessed on 07.05.2014.

D. Huber. *Automatic Three-dimensional Modeling from Reality*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2002.

A. Hyvarinen, P. Hoyer, and E. Oja. Image denoising by sparse code shrinkage. *Intelligent Signal Processing*, pages 554–568, 2001.

A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(5):433–449, 1999. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/34.765655.

J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach. Real-time compression of point cloud streams. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Proc. of the Int. Symposium on Mixed and Augmented Reality (ISMAR)*, pages 225–234. IEEE, 2007.

K. Konolige and K. Chou. Markov localization using correlation. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, 1999.

K. Konolige and P. Mihelich. Kinect calibration tutorial, 2011. http://www.ros.org/wiki/kinect_calibration/technical. Last accessed on 07.05.2014.

K. Konolige, E. Marder-Eppstein, and B. Marthi. Navigation in hybrid metric-topological maps. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Autonomous Robots*, 27(4):387–407, 2009. doi: http://dx.doi.org/10.1007/s10514-009-9155-6.

R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g^2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large scale graph-based SLAM using aerial images as prior information. *Journal of Autonomous Robots*, 30(1):25–39, 2011. doi: 10.1007/s10514-010-9204-1.

R. Kümmerle, G. Grisetti, and W. Burgard. Simultaneous parameter calibration, localization, and mapping. *Advanced Robotics*, 26(17):2021–2041, 2012. doi: 10.1080/01691864.2012.728694.

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.

R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. Autonomous robot navigation in highly populated pedestrian zones. In *Journal on Field Robotics*, 2014. To appear.

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

H. Li, R. W. Sumner, and M. Pauly. Global correspondence optimization for non-rigid registration of depth scans. *Computer Graphics Forum (Proc. SGP'08)*, 27(5), July 2008.

Y. Li and E. Olson. A general purpose feature extractor for light detection and ranging data. *Sensors*, 10(11):10356–10375, 2010. ISSN 1424-8220. doi: 10.3390/s101110356.

B. Limketkai, L. Liao, and D. Fox. Relational object maps for mobile robots. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, volume 19, page 1471, 2005.

J. Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statist. Comput.*, 6:113–119, 1996.

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

F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2D range scans. In *Proc. of the IEEE Conf. on Comp. Vision and Pattern Recognition (CVPR)*, pages 935–938, 1994.

F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333–349, 1997. ISSN 0929-5593.

L. Ma, T. Whelan, E. Bondarev, P. H. N. de With, and J. McDonald. Planar simplification and texturing of dense point cloud maps. In *Proc. of the European Conf. on Mobile Robots (ECMR)*, Barcelona, Spain, September 2013.

M. Magnusson. Aass-loop dataset, 2007. http://kos.informatik.uni-osnabrueck.de/3Dscans. Last accessed 07.05.2014.

M. Magnusson, H. Andreasson, A. Nüchter, and A. J. Lilienthal. Automatic appearance-based loop detection from 3D laser data using the normal distributions transform. *Journal on Field Robotics*, 26(11–12):892–914, November 2009. ISSN 1556-4959 (Print), 1556-4967 (Online).

M. Magnusson, T. Duckett, and A. J. Lilienthal. 3D scan registration for autonomous mining vehicles. *Journal on Field Robotics*, 24(10):803–827, 2007. ISSN 1556-4959 (Print), 1556-4967 (Online).

E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey, and K. Konolige. The office marathon: Robust navigation in an indoor office environment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

M. Montemerlo, S. Thrun, 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.

M. Montemerlo et al. Junior: The stanford entry in the urban challenge. *Journal on Field Robotics*, 25(9):569–597, 2008.

D. Moore, A. S. Huang, M. Walter, E. Olson, L. Fletcher, J. Leonard, and S. Teller. Simultaneous local and global state estimation for robotic navigation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

Y. Morales, A. Carballo, E. Takeuchi, A. Aburadani, and T. Tsubouchi. Autonomous robot navigation in outdoor pedestrian walkways. *Journal on Field Robotics*, 26(8):609–635, 2009.

A. C. Murtra, J. M. M. Tur, and A. Sanfeliu. Action evaluation for mobile robot global localization in cooperative environments. *Journal of Robotics & Autonomous Systems*, 56(10):807 – 818, 2008. ISSN 0921-8890.

M. Nelson. LZW data compression. *Dr. Dobb's Journal*, 14(10):36, 1989.

R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proc. of the Int. Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

P. Newman, G. Sibley, M. Smith, M. Cummins, A. Harrison, C. Mei, I. Posner, R. Shade, D. Schroeter, L. Murphy, et al. Navigating, recognizing and describing urban spaces with vision and lasers. *Int. Journal of Robotics Research*, 28(11-12):1406–1433, 2009.

A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6d SLAM with approximate data association. In *Proc. of the Int. Conf. on Advanced Robotics (ICAR)*, pages 242–249, 2005.

B. Olshausen, D. Field, et al. Sparse coding with an overcomplete basis set: A strategy employed by vi? *Vision research*, 37(23):3311–3326, 1997.

E. Olson. *Robust and Efficient Robotic Mapping*. PhD thesis, MIT, 2008.

E. Olson, J. 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.

E. Olson. Recognizing places using spectrally clustered local matches. *Robotics and Autonomous Systems*, 57(12):1157–1172, December 2009a.

E. Olson and M. Kaess. Evaluating the performance of map optimization algorithms. In *RSS Workshop on Good Experimental Methodology in Robotics*, 2009.

E. B. Olson. Real-time correlative scan matching. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009b.

Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proc. of the 27th Annual Asilomar Conf. on Signals, Systems, and Computers*, pages 40–44. IEEE, 1993.

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

F. W. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher, et al. Caroline: An autonomously driving vehicle for urban environments. *Journal on Field Robotics*, 25(9):674–724, 2008.

M. Reggente, A. Mondini, G. Ferri, B. Mazzolai, A. Manzi, M. Gabelletti, P. Dario, and A. J. Lilienthal. The dustbot system: Using mobile robots to monitor pollution in pedestrian area. *Chemical Engineering Transactions*, 23:273–278, 2010.

E. Royer, J. Bom, M. Dhome, B. Thuilot, M. Lhuillier, and F. Marmoiton. Outdoor autonomous navigation using monocular vision. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1253–1258. IEEE, 2005.

R. Rubinstein, M. Zibulevsky, and M. Elad. Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. *CS Technion*, 2008.

R. Rubinstein, M. Zibulevsky, and M. Elad. Double sparsity: Learning sparse dictionaries for sparse signal approximation. *IEEE Trans. on Signal Processing*, 58(3):1553–1564, 2010.

M. Rufli, D. Ferguson, and R. Siegwart. Smooth path planning in constrained environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of 3D object models from partial views. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Range sensor based model construction by sparse surface adjustment. In *Proc. of the IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 2011a.

M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Highly accurate maximum likelihood laser mapping by jointly optimizing laser points and robot poses. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011b.

M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Range sensor based model construction by sparse surface adjustment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012a.

M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. *3D Environment Modeling Based on Surface Primitives*, pages 281–300. Springer Berlin/Heidelberg, 2012b.

M. Ruhnke, L. Bo, D. Fox, and W. Burgard. Compact RGBD surface models based on sparse coding. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2013.

M. Ruhnke, L. Bo, D. Fox, and W. Burgard. Hierarchical sparse coded surface models. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2014.

M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of compact 3D models based on the detection of recurrent structures. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2137–2142. IEEE, 2010.

R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3D recognition and pose using the viewpoint feature histogram. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

A. Sanfeliu, J. Andrade-Cetto, M. Barbosa, R. Bowden, J. Capitán, A. Corominas, A. Gilbert, J. Illingworth, L. Merino, J. M. Mirats, et al. Decentralized sensor fusion for ubiquitous networking robotics in urban areas. *Sensors*, 10(3):2274–2314, 2010.

R. Schnabel, P. Degener, and R. Klein. Completion and reconstruction with primitive shapes. *Computer Graphics Forum (Proc. of Eurographics)*, 28(2):503–512, 2009.

F. Schneider and D. Wildermuth. Results of the european land robot trial and their usability for benchmarking outdoor robot systems. *Towards Autonomous Robotic Systems*, pages 408–409, 2011.

G. Seetharaman, A. Lakhotia, and E. Blasch. Unmanned vehicles come of age: The darpa grand challenge. *Computer*, 39(12):26–29, 2006.

A. V. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

Sick AG. Lms2xx laser measurement systems, 2006. http://sicktoolbox.sourceforge.net/docs/sick-lms-technical-description.pdf. Last accessed on 07.05.2014.

Sick AG. Laser measurement sensors of the lms1xx product family, 2012a. https://www.mysick.com/saqqara/pdf.aspx?id=im0031331. Last accessed on 07.05.2014.

Sick AG. Lms5xx laser measurement technology, 2012b. https://www.mysick.com/saqqara/pdf.aspx?id=im0038166. Last accessed on 07.05.2014.

R. Siegwart et al. RoboX at Expo.02: A large-scale installation of personal robots. *Journal of Robotics & Autonomous Systems*, 42(3-4), 2003.

R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, 1990.

C. Stachniss, G. Grisetti, and W. Burgard. Information gain-based exploration using rao-blackwellized particle filters. In *Proc. of Robotics: Science and Systems (RSS)*, pages 65–72, Cambridge, MA, USA, 2005.

B. Steder. Dataset of 360° 3D scans of the Faculty of Engineering, University of Freiburg, Germany, 2009. http://ais.informatik.uni-freiburg.de/projects/datasets/fr360. Last accessed on 07.05.2014.

B. Steder, G. Grisetti, and W. Burgard. Robust place recognition for 3D range data based on point features. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010a.

B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010b.

B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard. Place recognition in 3D scans using a combination of bag of words and point feature based relative pose estimation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011a.

B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. Point feature extraction on 3D range scans taking into account object boundaries. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011b.

J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D slam systems. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A second generation mobile tour-guide robot. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1999.

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.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

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

G. D. Tipaldi, M. Braun, and K. O. Arras. Flirt: Interest regions for 2D range data with applications to robot navigation. In *Proc. of the Int. Symposium on Experimental Robotic (ISER)*, 2010.

P. Trahanias, W. Burgard, A. Argyros, D. Hähnel, H. Baltzakis, P. Pfaff, and C. Stachniss. TOURBOT and WebFAIR: Web-operated mobile robots for tele-presence in populated exhibitions. *IEEE Robotics & Automation Magazine*, 12(2):77–89, 2005.

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.

B. Triggs, P. Mclauchlan, H. Richard, and A. Fitzgibbon. Bundle adjustment – a modern synthesis, 2000.

E. Trulls, A. Corominas Murtra, J. Pérez-Ibarz, G. Ferrer, D. Vasquez, J. M. Mirats-Tur, and A. Sanfeliu. Autonomous navigation for mobile service robots in urban pedestrian environments. *Journal on Field Robotics*, 28(3):329–354, 2011.

C. Urmson. *Navigation Regimes for Off-Road Autonomy*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2005.

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.

T. Whelan, M. Kaess, J. Leonard, and J. McDonald. Deformation-based loop closure for large scale dense RGB-D SLAM. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

O. Wulf. Hanover2 dataset, 2007. http://kos.informatik.uni-osnabrueck.de/3Dscans. Last accessed on 07.05.2014.

K. M. 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.

K. M. Wurm, H. Kretzschmar, R. Kümmerle, C. Stachniss, and W. Burgard. Identifying vegetation from laser data in structured outdoor environments. *Robotics and Autonomous Systems*, 2012. ISSN 0921-8890. doi: 10.1016/j.robot.2012.10.003.

K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2, 2010.

J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Proc. of the IEEE Conf. on Comp. Vision and Pattern Recognition (CVPR)*, pages 1794–1801. IEEE, 2009.