Feature-Based 3D Perception for Mobile Robots

Bastian Steder

Technische Fakultät
Albert-Ludwigs-Universität Freiburg im Breisgau

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

April 2013

ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Dissertation zur Erlangung des Doktorgrades
der Technischen Fakultät der
Albert-Ludwigs-Universität Freiburg im Breisgau

Feature-Based 3D Perception for Mobile Robots

Bastian Steder

April 2013

Betreuer: Prof. Dr. Wolfram Burgard

Dekan der Technischen Fakultät: Prof. Dr. Yiannos Manoli

1. Gutachter: Prof. Dr. Wolfram Burgard
2. Gutachter: PD Dr. Cyrill Stachniss

Tag der Disputation: 15.04.2013

Abstract

Perception is one of the key topics in robotics research. It is about the processing of external sensor data and its interpretation. Strong perceptual abilities are a basic requirement for a robot working in an environment that was not specifically designed for the robot. Such a surrounding might be completely unknown or may change over time, so that a model cannot be provided to the robot a priori. Most people would only judge a robot to be truly intelligent if it perceives its environment, understands what is happening around it and acts accordingly. This is especially important in mobile robotics. A robot that moves through an environment and interacts with it has to know what is going on around it, where it is, where it can go, and where objects necessary for its task are located.

The topic of this thesis is the interpretation of low-level sensor information and its application in high-level tasks, specifically in the area of mobile robotics. We mainly focus on 3D perception, meaning the analysis and interpretation of 3D range scans. This kind of data provides accurate spatial information and is typically not dependent on the light conditions. Spatial information is especially important for navigation tasks, which, by nature, are elementary for mobile platforms. To solve different problems, we extract features from the sensor data, which can then be used efficiently to solve the task at hand. The term “feature” is very broad in this context and means that some useful information is derived from raw data, which is easier to interpret than the original input.

At first, we discuss the benefits of point feature extraction from 3D range scans, meaning the detection of interesting areas and an efficient description of the local data. Such point features are typically employed for similarity measures between chunks of data. We present an approach for point feature extraction and then build on it to create several systems that tackle highly relevant topics of modern robotics. These include the detection of known objects in a 3D scan, the unsupervised creation of object models from a collection of scans, the representation of an environment with a small number of surface primitives, and the ability to find the current position of a robot based on a single 3D scan and a database of already known places. All these problems require an algorithm that detects similar structures in homogeneous sensor data, i.e., to find corresponding areas between 3D range scans. In addition to this, we discuss a system where finding correspondences between heterogeneous data types is relevant. Specifically, we search for corresponding areas in 3D range data and visual images, to determine the position of a robot in an aerial image. Finally, we present a complete robotic system, designed to navigate as a pedestrian in an urban environment. Such a system is built up from a multitude of different modules, whereas high robustness requirements apply to each of them to ensure the reliability of the entire system. Our core contribution to this system is a module that analyzes the low-level sensor data and provides high-level information to the other modules, specifically, it performs a traversability analysis and obstacle detection in 2D and 3D laser data.

We present several innovative algorithms and techniques that advance the state of the art. We use them to build systems that enable us to address complex 3D perception problems, outperforming existing approaches. We evaluate our methods in challenging settings, focusing on realistic applications and using real world data.

Zusammenfassung

Eines der zentralen Themen der Robotik ist die Wahrnehmung. Die korrekte Interpretation roher Sensordaten ist oft der kritischste Aspekt, wenn man Roboter in nicht speziell für sie geschaffenen Umgebungen einsetzen will. Ausgeprägte Fähigkeiten in der Wahrnehmung sind unabdingbar für die Lösung komplexer Aufgaben in unbekannten oder sich verändernden Umgebungen. So würden die meisten Menschen einen Roboter nicht als intelligent einstufen, wenn er nicht wahrnehmen könnte, was um ihn herum passiert dementsprechend handelt und auf Veränderungen reagiert. Besonders für mobile Plattformen ist es wichtig zu verstehen, was um sie herum passiert, wo sie sind, wohin sie sich bewegen können und wo sich die Dinge befinden, die für die aktuelle Aufgabe relevant sind.

Der Titel dieser Arbeit lautet übersetzt „Merkmalsbasierte 3D-Wahrnehmung für mobile Roboter“. Zentrales Thema dieser Arbeit ist die Interpretation von rohen Sensordaten und deren Nutzung in komplexen Aufgaben, mit speziellem Fokus auf der mobilen Robotik. Wir werden uns dabei hauptsächlich auf 3D-Wahrnehmung konzentrieren, also auf die Analyse und Deutung von dreidimensionalen Scans, wie man sie beispielsweise von schwenkbaren Laserscannern oder Tiefenkameras erhält. Diese Daten haben den Vorteil, dass sie räumliche Strukturen akkurat beschreiben und typischerweise unabhängig von äußeren Einflüssen wie zum Beispiel der Beleuchtung sind. Das Wissen über die räumliche Struktur der Umgebung ist besonders wichtig für Navigationsaufgaben, die naturgemäß elementar für mobile Plattformen sind. Zur Lösung verschiedener Probleme werden wir Merkmalsextraktionsmethoden einsetzen. In diesem Zusammenhang bedeutet dies, dass wir aus den rohen Sensordaten nützliche Informationen ableiten, die einfacher zu verarbeiten sind als die ursprünglichen Daten. Diese extrahierten Merkmale können als Grundstein für die Lösung komplexer Wahrnehmungsprobleme genutzt werden.

Im Verlauf dieser Arbeit werden wir mehrere neue Algorithmen und innovative Techniken präsentieren, die den aktuellen Stand der Forschung im Bereich der automatisierten Wahrnehmung erweitern. Wir werden Systeme vorstellen, die es uns ermöglichen, komplizierte 3D-Wahrnehmungsprobleme anzugehen. Dabei legen wir einen hohen Wert auf realistische experimentelle Auswertungen in anspruchsvollen Situationen und auf echten Sensordaten.

Die vorliegende Arbeit ist wie folgt strukturiert. In Kapitel 1 werden wir den Inhalt dieser Arbeit zusammenfassen, unseren wissenschaftlichen Beitrag erläutern, resultierende quelleoffene Software vorstellen und unsere Veröffentlichungen auflisten. In Kapitel 2 werden wir grundlegendes Wissen erläutern, das für das Verständnis dieser Arbeit nötig ist. Die darauf folgenden Kapitel werden dann spezifische Wahrnehmungsprobleme im Bereich der mobilen Robotik behandeln.

In Kapitel 3 werden wir die Vorteile der Punktmerkmalsextraktion diskutieren. Dies behandelt die Bestimmung relevanter Punkte in den Eingangsdaten und der Berechnung eines kompakten Beschreibungsvektors für deren unmittelbare Umgebungen. Solche Merkmale werden typischerweise eingesetzt wenn man ein Ähnlichkeitsmaß für den Vergleich zweier Datenblöcke benötigt. Wir werden einen Punktmerkmalstyp einführen, den wir speziell für den Einsatz

auf 3D-Tiefendaten konzipiert haben.

Darauf folgt in den Kapiteln 4 bis 7 die Beschreibung verschiedener Systeme, die auf dieser Merkmalstechnologie aufbauen um verschiedene Wahrnehmungsprobleme zu lösen. Diese haben alle gemeinsam, dass sie eine Methode zum effizienten Finden ähnlicher 3D-Strukturen benötigen.

So präsentieren wir in Kapitel 4 ein Objekterkennungssystem. Es durchsucht 3D-Scans nach bekannten Objekten und versucht deren exakte Position zu bestimmen. Zu diesem Zweck werden die zuvor angesprochenen Punktmerkmale verwendet um ähnlich strukturierte Bereiche in dem 3D-Objektmodell und der aktuellen Szene zu finden. Basierend auf solchen potentiellen Korrespondenzen werden mögliche Objektpositionen berechnet. Diese werden dann anhand der Wahrscheinlichkeit bewertet, dass das Vorhandensein des Objektes an dieser Stelle zu den gemessenen Sensordaten führt. In Kapitel 5 erläutern wir einen Ansatz zum unüberwachten Lernen von Objektmodellen. Unser Verfahren erzeugt Punktwolkenmodelle von Objekten aus einer unzusammenhängenden Menge von 3D-Scans in denen verschiedene Perspektiven dieser Objekte vorkommen. Kapitel 6 behandelt eine Methode zur Umgebungsmodellierung basierend auf Oberflächenprimitiven. Unser System ist in der Lage, dreidimensionale Szenen basierend auf einer kleinen Sammlung (dem „Wörterbuch“) von Oberflächenteilen (den „Wörtern“) zu repräsentieren. Dies basiert auf der Erkenntnis, dass typische dreidimensionale Umgebungen aus vielen immer wieder vorkommenden Strukturen aufgebaut sind. Beispielsweise flache Flächen, Kanten, Ecken, Krümmungen, usw.. Unser System lernt eine Art Wörterbuch solcher Strukturen basierend auf einer vorgegebenen 3D Szene und benutzt es dann, um diese oder andere Szenen zu beschreiben. Hierzu wird für die verschiedenen Stellen in der Szene lediglich angegeben, welches Element aus dem Wörterbuch dort zu platzieren ist. Solche Beschreibungen sind nicht nur kompakter, sondern auch semantisch aussagekräftiger als die Rohdaten, da Ähnlichkeiten in den Strukturen explizit mitkodiert sind. In Kapitel 7 präsentieren wir ein System zur Ortserkennung. Dieses System vergleicht einen 3D-Scan mit einer Datenbank schon bekannter Scans. Es bestimmt dann, ob der Scan an einer schon bekannten Position aufgenommen wurde oder ob diese Umgebung bisher unbekannt ist. Im Falle einer bekannten Umgebung wird zusätzlich akkurat der Positionsunterschied zwischen den beiden Observationen bestimmt.

Bis zu diesem Punkt haben alle angesprochenen Systeme darauf aufgebaut, dass innerhalb homogener Daten nach ähnlichen Strukturen gesucht wird. Speziell wurden korrespondierende Bereiche in 3D-Scans gesucht. Im Unterschied hierzu präsentieren wir in Kapitel 8 ein System, das versucht Korrespondenzen zwischen heterogenen Datentypen zu finden. Speziell zwischen 3D-Tiefendaten und visuellen Bildern. Dieses Kapitel befasst sich mit einem Ansatz zur Kartierung unter Verwendung von Vorwissen in Form von Luftbildern. Das heißt wir werden eine Methode vorstellen, mit der die Position eines Roboters in einem Luftbild seiner Umgebung verfolgt werden kann. Die so bestimmten Positionen können dann verwendet werden, um aus den Sensordaten des Roboters akkurate und global konsistente Karten aufzubauen.

Kapitel 9 befasst sich mit einem kompletten robotischen System. Dieser Roboter dient der Fußgängerassistenz und ist dafür ausgelegt, autonom in stark belebten städtischen Umgebungen zu navigieren. Das System ist aus einer Vielzahl verschiedener Softwaremodule aufgebaut. Dabei bestehen hohe Ansprüche an die Robustheit jedes Einzelnen dieser Module, da nur so ein stabiles Gesamtsystem gewährleistet werden kann. Unser Hauptbeitrag zu diesem System ist ein Modul zur Befahrbarkeitsanalyse und Hinderniserkennung in Laserscannerdaten. Hierbei geht es darum neben bewegten Objekten auch Hindernisse zu erkennen, die die Roboterplattform bauartbedingt nicht befahren kann. Hierzu gehören unter anderem zu hohen Stufen und Grasflächen. Kapitel 10 wird die Ergebnisse dieser Dissertation abschließend zusammenfassen und außerdem mögliche Fortführungen unserer Arbeit thematisieren.

Acknowledgements

In the 21. Century, there once was a poor little computer scientist, unsure what his abilities should be used for and where his life would lead him. They were hard times of insecurity and fear. Would he have to create webpages for a living!? Or explain commercial software to n00bs!? But then he found a magical place, where he could work together with other ~~crazy nerdy~~ motivated people sharing his interests. There were amiable colleagues, interesting problems to work on, a super-fast Internet connection, high-end computers, and incredible freedom. He was allowed to sit in front of his keyboard all day, even being paid for it, and to travel the world to talk to like-minded people. And there were *robots*!!! Computers that you could take for a walk! So he lived happily ever after... Or, well, at least until the completion of this thesis...

I have to thank a lot of people, who made this a great time in my life and without whom this thesis would not have been possible.

First I would like to thank my adviser Wolfram Burgard for giving me this chance, for his advice, for his immense experience, for sometimes forcing me to broaden my horizon, and for giving us all extraordinary freedom in our work. But also for a relaxed atmosphere and 'ordering' us to leave work to go geo-caching from time to time.

I am immensely grateful to Giorgio Grisetti, for cajoling me into doing this Ph.D, and for his continuing support, especially in the beginning, when a paper was still something to write a grocery list on. He always had an open ear for problems, ideas, general discussions, or math questions and always gave great advice. The effort he puts into his students is admirable.

I want to thank Cyrill Stachniss for always being available for questions, for giving advice in scientific and administration issues, for his help with papers, his support in projects, and of course for agreeing to be a reviewer for this thesis.

Special thanks belongs to my office mates Rainer Kümmerle and Michael Ruhnke for a great atmosphere with a lot of fun, great discussions, willing answers to sometimes intellectually challenged questions, and in general for being good friends. But also for accepting a not too orderly office on my part (or should I call it an especially cluttered environment?) and a sometimes above-average sound intensity in the office.

Thank you to Slawomir Grzonka for always being helpful and supportive, for great discussions, enriching our office with his visits and just generally for being who he is. Axel Rottmann for always keeping the mood in the lab up and for being the first person to go to on a hunt for abstruse things. Christoph Sprunk for fun conversations and providing the sweets glass, thereby ensuring that everyone's blood sugar cannot drop to a dangerously low level. And generally a big thank you to all the current and former members of the AIS lab for creating a great working atmosphere and just being good people. Thanks a lot to Willow Garage for giving me the chance for a summer internship and making it an awesome time. And of course thanks to all the people involved in the projects I was working on.

Thanks to all my co-authors for providing their scientific expertise: Wolfram Burgard, Christian Dornhege, Giorgio Grisetti, Alexander Kleiner, Kurt Konolige, Rainer Kümmerle, Mark Van Loock, Axel Rottmann, Michael Ruhnke, Radu Bogdan Rusu, Cyrill Stachniss, Juan Domingo Tardós, Takashi Yamamoto, and Yoshiaki Asahara.

Above all, I want to thank my family and friends for always being there for me. You are just incredibly awesome!

Special thanks to Corinna for accepting a helpless nerd in her life and providing him with love and a normal, but not too normal, perceptive of things!

My deepest gratitude to Family K&H for a place where I always feel welcome and for giving me so much!

Thanks a million times to all the cool people from the Budokan Freiburg, who made my life so much richer!

And last but not least I have to thank all the people who helped me make this dissertation better by giving me feedback for earlier versions or providing general suggestions about the writing process: Wolfram Burgard, Giorgio Grisetti, Slawomir Grzonka, Corinna Guggemos, Henrik Kretzschmar, Rainer Kümmerle, Michael Ruhnke, and Cyrill Stachniss.

This work has partly been supported by the European Commission under contract numbers FP6-2005-IST-6-RAWSEEDS, FP6-IST-34120-muFly, FP7-231888-EUROPA, FP7-248258-First-MM, and ERC-AG-PE7-267686-LifeNav, the DFG under contract number SFB/TR-8, and the intern program of Willow Garage.

Contents

Chapter 1

Introduction

At the base of most complex tasks in mobile robotics lies a hard perception problem. The correct interpretation of raw sensor data is often a crucial part when one aims at applications in the real world. A robot must be able to understand its surrounding, in order to work in it and interact with it. Most people would only call a robot intelligent, if it is able to perceive what is happening around it. Perception is a prerequisite for the ability to take only suitable actions and to react to changes. Without appropriate sensors a robot is very restricted in what it can achieve and is only able to work at very specific tasks. In highly controlled environments, like they exists for automated assembly, external sensors might be omitted. If all required parts are always placed at the same locations, a robotic arm can perform tasks in a scripted open-loop fashion. But without such an environment that is specifically designed for the robot, it is indispensable to equip a robot with appropriate sensors and provide it with the means to understand their input. The topic of this thesis is therefore the interpretation of low-level sensor information and its application in high-level tasks.

The most popular sensors in robotics are arguably visual cameras and range sensors. Cameras provide rich information about the environment and have proven their potential in the scientific literature. In addition to that, human beings are able to solve a myriad of tasks based on vision. They have the advantage of being passive sensors with theoretically infinite range. Yet, visual information also has serious shortcomings. The data typically depends strongly on the illumination and spatial information can only be acquired indirectly because of the missing depth information. Range sensors, on the other hand, provide accurate spatial information about the environment. Their measurements are often independent of the illumination of the environment since they typically are active sensors. Laser scanners, for example, send out numerous pulses of infrared light in different directions and determine the distances to the closest obstacles on the beams, based on the time it takes the light to return to the receiver. While this means that such sensors have a very restricted range, typically less than $100\,\mathrm{m}$, much less prior knowledge is necessary to reason about the acquired data since it directly represents the geometry of the surrounding. This is an important property in navigation tasks, which, by nature, are elementary for mobile platforms. In this work, we mainly focus on 3D perception, meaning the analysis and interpretation of 3D range scans. We discuss several problems in the area of 3D perception and present solutions that are based on the extraction of features from the raw sensor data, which can then be used efficiently to solve the task at hand. “Feature” is a very broad term in this context. It means that we extract some compressed but useful information from the input data, which is easier to interpret than the raw measurements. We can then build upon these features to address complex 3D perception problems. During this thesis, we introduce several innovative algorithms and techniques that advance the state of the art in robotic perception. We elaborate systems that enable us to address complex 3D perception problems and focus on realistic experimental evaluations with real world data.

This thesis is organized as follows: In the remainder of this chapter, we summarize our scientific contributions and present open-source software resulting from our work. In Chapter 2, we introduce some basic knowledge that is elementary for the remaining content. The subsequent chapters then address specific perception problems in the context of mobile robotics.

In Chapter 3, we discuss the benefits of point feature extraction, meaning the detection of interesting areas (keypoint detection) and the description of such areas in a condensed fashion (descriptor calculation). Such features are used for similarity checks between chunks of data and are applicable to numerous purposes. We introduce an approach for point feature extraction from 3D range data, namely the Normal Aligned Radial Feature (NARF). See Figure 1.1(a) for a visual motivation.

The subsequent chapters (Chapter 4 to Chapter 7) present systems that build upon the NARF technology to solve different perceptual problems that all require an efficient way to find similar structures in 3D range scans. Finding such similarities is necessary to establish connections between parts of the scans. In Chapter 4, we describe an object recognition system that finds the positions of known objects in 3D scenes. Using point feature correspondences, the system determines similar areas between a scene and a point cloud model of an object. Based on the correspondences, our method calculates potential positions of the object in the scene. The candidate positions are then evaluated based on spatial verification. This means that our system compares the actual perception with what it expects to see and calculates a score based on how well that fits. See Figure 1.1(b) for a visual motivation. In Chapter 5, we introduce an approach for unsupervised model learning. We elaborate how we can build point cloud object models in an automated fashion. Our system only needs an unordered collection of scans that show different perspectives of one or multiple objects. See Figure 1.1(c) for a visual motivation. Chapter 6 presents a method for environment modeling based on surface primitives. Our system is able to represent 3D scenes based on a small dictionary of surface patches. This follows the observation that typical 3D environments consist of recurring structures, like flat surfaces, edges, corners, etc.. We learn a dictionary of such structures and use it to represent the scene, leading to a semantically rich representation, with low memory requirements. See Figure 1.1(d) for a visual motivation. In Chapter 7, we descuss a system for place recognition. This system matches a query scan against a database of scans and determines if the scan comes from a new location or one already in the database. In the latter case it also determines the relative pose between the scans. See Figure 1.1(e) for a visual motivation.

Up to this point, the mentioned approaches all require a method to detect similar structures in homogeneous sensor data, specifically to find corresponding areas between 3D range scans. In contrast to this, we describe a system that needs to find correspondences between heterogeneous data types, specifically between 3D range data and visual images, in Chapter 8. We present an approach for mapping under consideration of aerial images, meaning we discuss how one can track the position of a robot in an aerial image and use this information to build highly accurate, globally consistent maps. See Figure 1.1(f) for a visual motivation. In addition, we discuss how such maps can be used to evaluate SLAM approaches.

In Chapter 9, we introduce a complete robotic system in form of a pedestrian assistant robot, designed to navigate in highly populated urban environments. The system is built up from a multitude of different modules, to each of which apply high robustness requirements to ensure the robustness of the entire system. Our core contribution to this system is a module that analyzes the low-level sensor data and provides high-level information to the other modules, specifically, it performs a traversability analysis and obstacle detection in 2D and 3D laser data.

See Figure 1.1(g) for a visual motivation. We conclude this work in Chapter 10 and give an outlook on future work.

1.1 Scientific Contributions

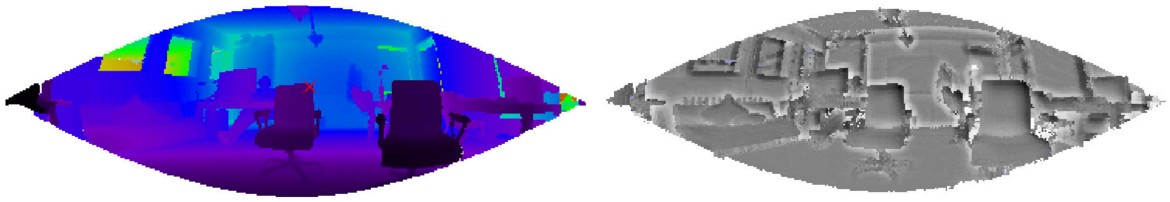
This thesis describes several contributions to the area of robotic perception based on 3D range data. In summary, we propose

- a novel point feature type for 3D range scans. The Normal Aligned Radial Feature (NARF) provides a method for keypoint extraction and descriptor calculation (Chapter 3);
- an approach for object recognition in 3D range scans, based on existing point cloud models of objects (Chapter 4);
- a new method to learn point cloud models of objects from an unordered collection of 3D range scans in an unsupervised fashion, whereas our main contribution is the detection and evaluation of overlapping areas, which is essential for building consistent models (Chapter 5);
- a system for 3D environment modeling based on a dictionary of surface primitives, which can be learned either on the same or on another collection of scans. Our main contribution here is the efficient detection of similar structures (Chapter 6);
- a technique for place recognition based on 3D range data, which calculates a score for a query scan and potential matches in the database, together with the relative transformation (Chapter 7);
- a SLAM algorithm, which incorporates aerial images as prior information, whereas our main contribution was in the matching of the sensor data against the aerial image. In this context we also present a metric to compare the performance of different SLAM approaches (Chapter 8);
- an existing, real world navigation system for a robotic pedestrian assistant, whereas our main contribution is in the traversability analysis and obstacle detection (Chapter 9).

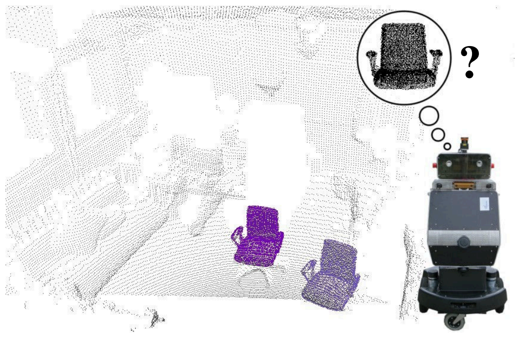
1.2 Contributions to Open-Source Software

We released several of the algorithms described in this thesis as open-source software. We believe that open-source software is a great possibility to enable realistic comparisons between different approaches and to speed up the research process since the wheel does not have to be invented again and again by different researchers. We invite others to use our software, to extend it, and to adapt it to their needs.

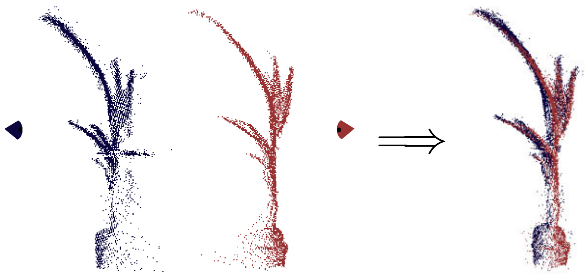
- Contributions to the Point Cloud Library (PCL)
  `http://www.pointclouds.org`
  We contributed different classes to this open source library. Mainly the following parts:
  - The range image class: We provide as class that implements range images (see Section 2.3.2) and implements specific functionalities for them.



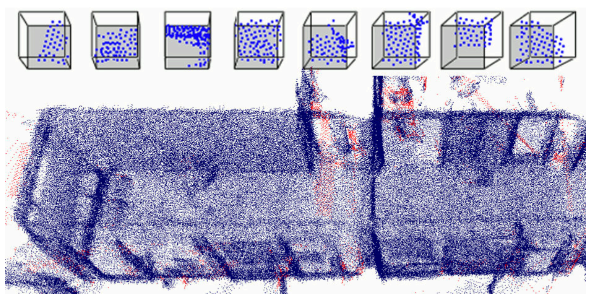
(a)



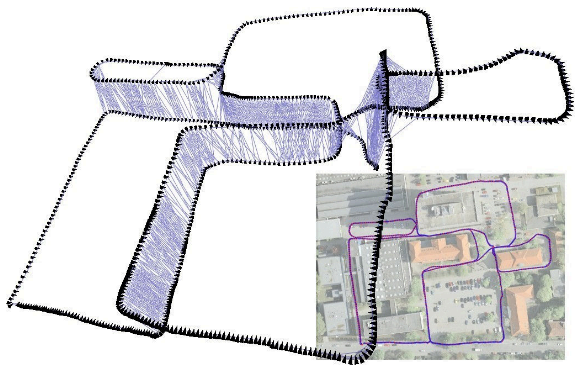
(b)



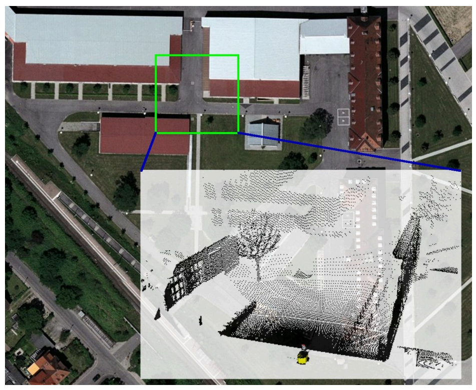
(c)



(d)



(e)



(f)

(g)

Figure 1.1: Motivational images for the different chapters of this thesis. **(a)**: On the left we marked the top right corner of a chair. On the right, the similarity values of all other points according to our feature descriptor are shown, from least similar (white) to most similar (black). Upper right rectangular corners are clearly distinguished. **(b)**: Object recognition. A robot looking for the model of a chair in a point cloud of an office scene. **(c)**: Unsupervised model learning. Two different views of a plant were found in different scans and automatically merged into a new model. **(d)**: Modeling with surface primitives. A point cloud of a corridor, built up from a collection of surface patches, whereas eight of these are shown on the top. **(e)**: Place recognition. The trajectory of a dataset of 3D scans on a Google Earth (©Google) image, overlain with loop closures found by our system. **(f)**: Mapping under consideration of aerial images. The robot matches its current 3D scan against the aerial image (©Google) to determine its position. **(g)**: Our pedestrian assistant robot moving in a crowded inner city area.

  - The range image border extractor: This class finds traversals from foreground to background in range images. See Section 3.1.
  - The NARF keypoint detector: This class extracts NARF keypoints from range images. See Section 3.2.
  - The NARF descriptor: This class calculates NARF descriptors for points in range images. See Section 3.3.

  This software was initially developed in collaboration with Radu B. Rusu, Kurt Konolige, and Wolfram Burgard. Since then, a number of PCL developers provided patches and bug fixes, for which we are grateful.

- SLAM evaluation toolkit
  `http://ais.informatik.uni-freiburg.de/slamevaluation`
  We created this tool to enable other researchers to evaluate the accuracy of their SLAM algorithms, given a set of manually verified relations. See Section 8.3.2 for additional details on the metric used for the evaluation. Next to the software library, we offer such relations for some frequently used public datasets on the website and also provide a number of complete datasets we captured ourselves. This framework is a joint effort with Rainer Kümmerle, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, Alexander Kleiner, and Wolfram Burgard.

1.3 Publications

The content of this thesis is based on papers published in international journals and conference proceedings and one patent. In the remainder of this section, these publications are listed chronologically.

Journal Articles:

- 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

- R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Journal of Autonomous Robots*, 27(4):387–407, 2009

Patents:

- Y. Asahara, T. Yamamoto, M. Van Loock, B. Steder, G. Grisetti, and W. Burgard. Object recognition method, object recognition apparatus, and autonomous mobile robot, 2012. (PATENT)

Conferences, Workshops, and Book Chapters:

- 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. Accepted for Publication

- M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. 3D environment modeling based on surface primitives. In *Towards Service Robots for Everyday Environments*, pages 281–300. Springer Berlin/Heidelberg, 2012

- 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)*, 2011

- 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)*, 2011

- 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)*, 2010

- 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)*, 2010 **Finalist Best Paper Award**

- 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)*, 2010

- B. Steder, G. Grisetti, M. Van Loock, and W. Burgard. Robust on-line model-based object detection from range images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009

- 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

- R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large scale graph-based SLAM using aerial images as prior information. In *Proc. of Robotics: Science and Systems (RSS)*, 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

This thesis does not report on the following publications, which were written during my time as a research assistant:

- Journal Articles

  - B. Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual SLAM for flying vehicles. *IEEE Transactions on Robotics*, 24(5):1088–1093, 2008

- Conferences and Workshops
  - B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, and W. Burgard. Estimating consistent elevation maps using down-looking cameras and inertial sensors. In *Proc. of the Workshop on Robotic Perception on the Int. Conf. on Computer Vision Theory and Applications*, 2008
  - B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, A. Rottmann, and W. Burgard. Learning maps in 3D using attitude and noisy vision sensors. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007
  - B. Steder, A. Rottmann, G. Grisetti, C. Stachniss, and W. Burgard. Autonomous navigation for small flying vehicles. In *Workshop on Micro Aerial Vehicles at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007

1.4 Collaborations

During the creation of this thesis, we collaborated with a number of other researchers. We will now summarize these collaborations for the individual chapters of this work.

- Chapter 3: The system presented in this chapter is the result of a summer internship at Willow Garage, Inc (CA, USA). It was created in collaboration with Radu B. Rusu, Kurt Konolige, and Wolfram Burgard. See [130, 131] for related publications.

- Chapter 4: An initial implementation of the system presented in this chapter was the result of a collaboration with Giorgio Grisetti, Mark Van Loock, and Wolfram Burgard. It was published in [5] and [127]. Yet, the system and results we present in this chapter are based on a new implementation, which only shares some basic concepts with the original one. These results, which are the outcome of a collaboration with Giorgio Grisetti, Radu B. Rusu, Kurt Konolige, and Wolfram Burgard, were previously unpublished.

- Chapter 5: The system presented in this chapter is the result of a collaboration with Michael Ruhnke, Giorgio Grisetti, and Wolfram Burgard. The work was done in the context of Michael Ruhnke's diploma thesis, which was co-supervised by the author of this PhD thesis. We present the whole system for completeness, yet our main contribution of Chapter 5 is described in Section 5.1.4. While the author of this PhD thesis took part in the creation of the whole system and supported the work, the remaining content of this chapter was mostly handled by others. See [104] for the related publication.

- Chapter 6: The system presented in this chapter is the result of a collaboration with Michael Ruhnke, Giorgio Grisetti, and Wolfram Burgard. While we present the whole system for completeness, only Section 6.3 can be claimed as work been done in the context of this thesis. The author of this thesis took part in the creation of the whole system and supported the work, yet the remaining content of this chapter was mostly handled by others. See [105, 106] for related publications.

- Chapter 7: The system presented in this chapter is the result of a collaboration with Giorgio Grisetti, Michael Ruhnke, Slawomir Grzonka, and Wolfram Burgard. See [123, 129] for related publications.

- Chapter 8: The system presented in this chapter is the result of a collaboration with Rainer Kümmerle, Christian Dornhege, Alexander Kleiner, Giorgio Grisetti, and Wolfram Burgard. While we present the whole system for completeness, only Section 8.1.3 and Section 8.1.4 can be claimed as work been done in the context of this thesis. The author of this thesis took part in the creation of the whole system, preparing and performing the experiments, and general support of the work, yet the remaining content of this chapter was mostly handled by others. See [17, 72, 73, 74] for related publications.

- Chapter 9: The system presented in this chapter is the result of a collaboration with Rainer Kümmerle, Michael Ruhnke, Cyrill Stachniss, and Wolfram Burgard. It describes work that was done in the in the context of an EU project (FP7-231888-EUROPA). While we present the whole navigation system for completeness, only Section 9.2.3 can be claimed as work been done in the context of this thesis. The author of this thesis took part in the creation of the whole system, preparing and performing the experiments, and general support of the work, yet the remaining content of this chapter was mostly handled by others. See [71] for the related publication.

Chapter 2

Basics and Notations

In this chapter, we present the notations and basic concepts used throughout the remainder of this work. We describe mathematical notations in Section 2.1, abbreviations in Section 2.2, some commonly used data structures in Section 2.3, and give details about SLAM in Section 2.4.

2.1 Mathematical Notations

In the remainder of this thesis we will use the following notations in mathematical equations:

Example(s)	**Type**	**Description of notation**
$x = 1.0, \delta = 0.3$ $\alpha = 12^\circ$	Scalar	Lower case latin and greek letters, with α, β, and γ reserved for angles
$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix}$	Vector	Bold lower case latin letters
$R = \begin{pmatrix} x_{11} & x_{12} & \cdots \\ x_{21} & x_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$	Matrix	Upper case latin and greek letters
$\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$	Set	Upper case calligraphic letters
$\mathfrak{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$ $\mathfrak{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots\}$	Set of sets	Fraktur symbols
$\lvert -1 \rvert = 1$	Absolute value	A scalar enclosed by $\lvert\ \rvert$
$\lvert\mathbf{x}\rvert = \left\lvert \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\rvert = 2$	Size	A set or a vector enclosed by $\lvert\ \rvert$
$\lvert\lvert\mathbf{x}\rvert\rvert_p = \left(\sum_{i=1}^{\lvert\mathbf{x}\rvert} \lvert x_i \rvert^p \right)^{\frac{1}{p}}$ $\lvert\lvert\mathbf{x}\rvert\rvert_1 = \sum_{i=1}^{\lvert\mathbf{x}\rvert} \lvert x_i \rvert$ $\lvert\lvert\mathbf{x}\rvert\rvert_2 = \sqrt{\sum_{i=1}^{\lvert\mathbf{x}\rvert} x_i^2}$	L^p-norm	A vector enclosed by $\lvert\lvert\ \rvert\rvert_p$, whereas $\lvert\lvert\ \rvert\rvert_1$ is the Manhattan norm and $\lvert\lvert\ \rvert\rvert_2$ is the Euclidean norm

2.2 Abbreviations

We will use the following abbreviations in this thesis:

Abbreviation	**Meaning**	**Brief description**
BIC	**B**ayesian **I**nformation **C**riterion	A decision criterion for model learning methods
BoW	**B**ag **o**f **W**ords	An approach that describes data compactly based on a dictionary
g^2o	**G**eneral **G**raph **O**ptimizer	An open source software for graph optimization problems
CAD	**C**omputer-**A**ided **D**esign	CAD programs are typically used to create 3D models
EUROPA	**EU**ropean **RO**botic **P**edestrian **A**ssistant	An EC-funded research project
GOODSAC	**GOOD SA**mple **C**onsensus	A sample consensus algorithm
GPS	**G**lobal **P**ositioning **S**ystem	A system for global localization based on satellite signals
ICP	**I**terative **C**losest **P**oint	A registration algorithm
IMU	**I**nertial **M**easurement **U**nit	A sensor measuring orientations
MCL	**M**onte **C**arlo **L**ocalization	A localization method based on a particle filter
NARF	**N**ormal **A**ligned **R**adial **F**eature	A 3D point feature type
NARVP	**N**ormal **A**ligned **R**ange **V**alue **P**atch	A 3D descriptor vector
PCA	**P**rincipal **C**omponent **A**nalysis	A method to find eigenvalues and eigenvectors for a matrix
PCL	The **P**oint **C**loud **L**ibrary	An open source software for point cloud processing
RANSAC	**RAN**dom **SA**mple **C**onsensus	A sample consensus algorithm
RBPF	**R**ao-**B**lackwellized **P**article **F**ilter	A filtering method applicable to SLAM
RGB-D	**R**ed, **G**reen, **B**lue, and **D**epth	RGB-D sensors capture color and range images
SIFT	**S**cale-**I**nvariant **F**eature **T**ransform	A 2D point feature type
SLAM	**S**imultanious **L**ocalization **A**nd **M**apping	A mapping procedure for mobile robots
SURF	**S**peeded **U**p **R**obust **F**eature	A 2D point feature type

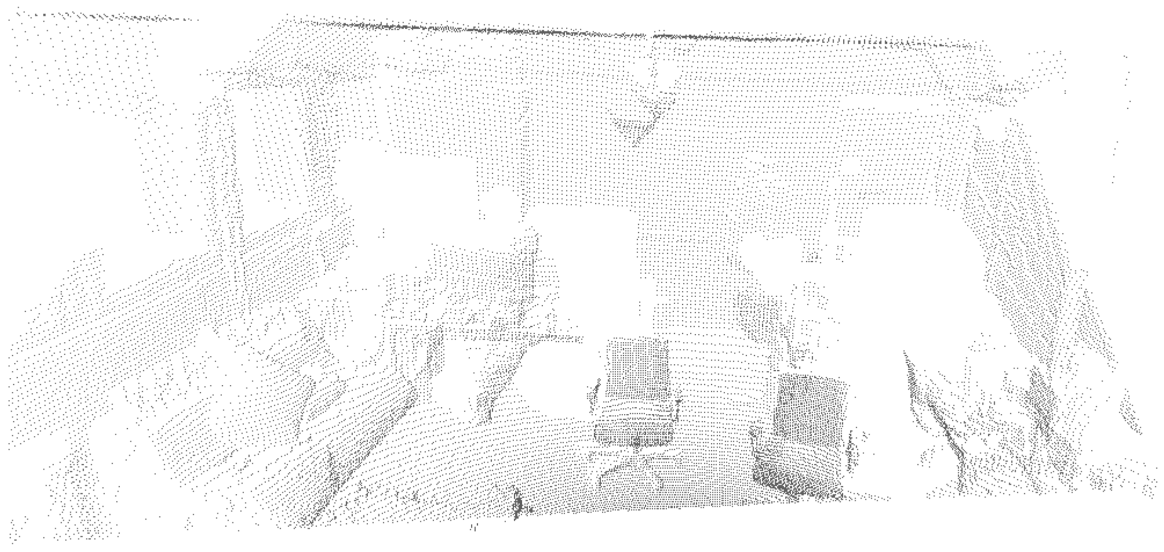


Figure 2.1: Visualization of an example point cloud showing an office scene.

2.3 Data Structures

The following data structures are frequently used in this work.

2.3.1 Point Clouds

Point clouds are a very common data structure to describe a set of unordered 3D points:

$$\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots\} \quad \text{with } \mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \forall\, \mathbf{p} \in \mathcal{P} \wedge p_x, p_y, p_z \in \mathbb{R} \tag{2.1}$$

See Figure 2.1 for a visualization of an example point cloud showing an office scene.

2.3.2 Range Images

Range images, also called *depth images*, are 2D representations of 3D scenes. Every pixel in a range image captures the distance to the closest obstacles as seen from one point in space, whereas the pixel position determines the direction of the beam. See Figure 2.2 for an example range image. There are different projection models to create a range image. They can be based on a projection plane, like normal visual camera images, but these lack the possibility to represent scenes that cover more than $180°$. Therefore, spherical projections are popular, like they are used for maps showing the whole globe. Here one important parameter of the range image is the angular resolution α, whereas $\alpha = 1°$ means that the angle between the beams represented by neighboring pixels is $1°$.

If a 3D scan is captured from one point in space, i.e., the sensor does not move while the 3D points are generated, a range image is a very good representation of the acquired data. It closely resembles the original sensor data and preserves not only information about the 3D end points, but also about the measured free space, maximum range readings, and unknown space. Every point in a range image also represents a 3D point, which is why range images can interchangeably be use as 2D images and 3D point clouds. Since computer vision is a very successful research area, it is reasonable to transfer concepts to the area of range-based

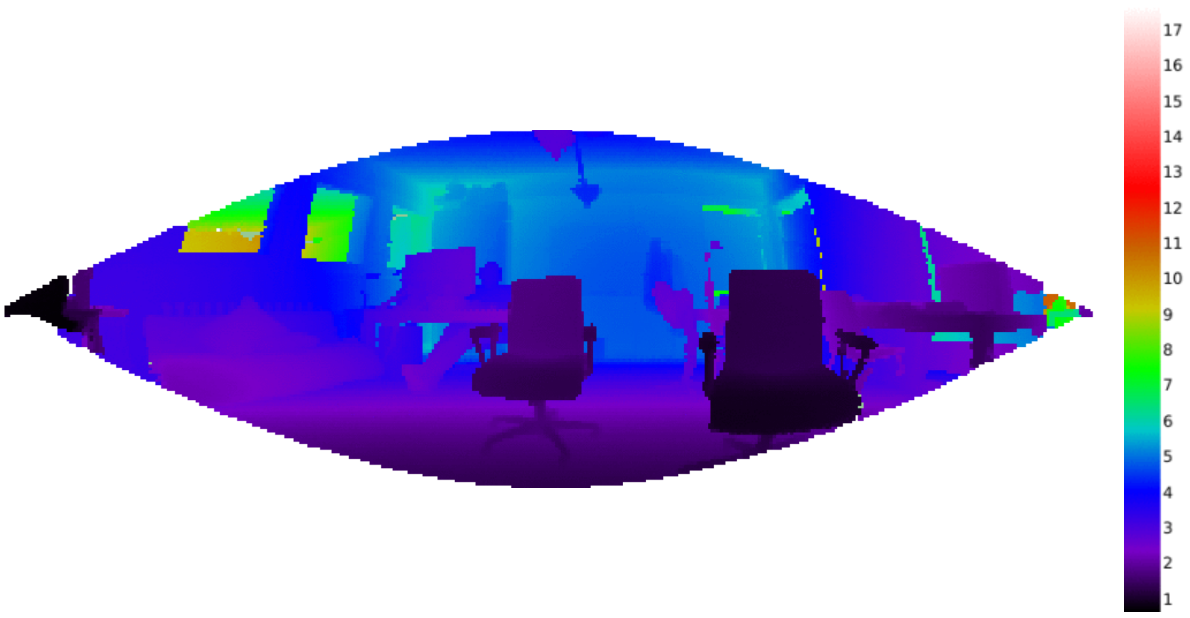


Figure 2.2: Visualization of an example range image representing the same scene as shown in Figure 2.1. The legend on the right shows which color corresponds to a specific measured range in meters. Every pixel represents a beam going from the known sensor position in a direction that is defined by the position of the pixel in the image. The range image in this example has an angular resolution of $\alpha = 0.5°$. The 3D endpoint can be calculated from the known beam and the measured range.

3D perception. Range images offer this possibility while still keeping the advantages of range scans. A range image can be calculated efficiently from a point cloud using a z-buffer [41], as long as the original sensor position is known.

An implementation of a range image class which was written in the context of this thesis is available under an open source license. See Section 1.2 for a list of contributions to open source software.

2.4 SLAM

SLAM means Simultaneous Localization and Mapping and describes one of the basic problems in robotics, namely how to use a mobile robot and its sensors to create a model of the environment. One can distinguish two sub-elements in this context, *mapping with known poses* and *localization*. Mapping with known poses addresses the problem, how the sensor information can be used to create a model of the environment under the assumption that the position of the robot is known for all time steps. Localization is the ability to determine the position of the robot given a model of the environment. Even if these sub-parts are solved, SLAM remains a hard problem since the two elements mutually depend on each other. To integrate new sensor data into the model, the robot needs to know its position, but to determine its position, it needs the model.

There exists a number of different approaches to address the SLAM problem. Approaches based on different variants of Kalman filters are very popular [39, 77, 120, 137] because they estimate a fully correlated posterior about landmark maps and robot poses. Unfortunately, they make strong assumptions about the motion model and the sensor noise. If these are violated, the filters tend to diverge [66, 144]. Particle filters [50, 90] are another possibility, where the probability distribution is approximated by a limited number of samples (particles). While such approaches can closely approximate arbitrary distributions for large numbers of particles, their runtime requirements grow linearly with this number which makes them computationally demanding in complex applications. In recent years, least square error minimization approaches [42, 51, 55, 70, 84, 96] have become very popular. These approaches try to optimize an error function for spatial constraints between robot poses to acquire a maximum likelihood trajectory. They have some relevance for the content of this work, mainly for Chapter 7, Chapter 8, and Chapter 9. We will therefore present their basic concepts in the following section.

2.4.1 Graph-based SLAM

One possibility to describe a SLAM problem is the so called *graph-based SLAM* formulation. Here, the individual robot poses of a trajectory are represented as nodes in a graph and the edges between the nodes represent spatial constraints between them. In general, these spatial constraints either represent incremental motions (odometry, scan matching, etc.) or detected *loop closures*. Loop closures are defined as multiple traversals of the same area and can be detected from overlapping observations. They are necessary to counter the error accumulation of the purely incremental constraints. Under this graph-based formulation, a solution to the SLAM problem is a configuration of the nodes which minimizes the error introduced by the constraints, thereby computing the configuration of the nodes that is most likely. Let $\mathbf{x}_i$ be node i in a graph, whereas each node contains the parameters to represent a robot pose. Let $\mathbf{z}_{ij}$ and Ω_{ij} be the mean and the information matrix of an observation, which describe a measurement of node j seen from node i. Let $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ be a function that computes the error between the current measurement $\mathbf{z}_{ij}$ and the expected measurement based on $\mathbf{x}_i$ and $\mathbf{x}_j$. We now define an error function for the complete graph configuration $\mathbf{x}$ as

$$\chi(\mathbf{x}) = \sum_{i,j} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^T \Omega_{ij} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}), \tag{2.2}$$

which leads to the following optimization problem:

$$\tilde{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x}} \chi(\mathbf{x}). \tag{2.3}$$

This means that we have to find a graph configuration $\tilde{\mathbf{x}}$ that minimizes the error of all the observations. This is equivalent to finding the maximum a-posteriori configuration given the measurements $\tilde{\mathbf{x}} = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{z})$ since $\chi(\mathbf{x})$ is actually the corresponding negative log-likelihood.

In this work, we solve such optimization problems with one of the publicly available software libraries that were designed for this purpose. Specifically we use the *General Graph Optimizer* (g^2o) [70].

Chapter 3

NARF: A Point Feature for 3D Range Data

The evaluation of similarity between different structures is highly relevant in many perception tasks since it is a prerequisite to recognize something as previously seen. In this chapter we address this topic by means of point feature extraction from 3D range data. We present the general concept of such features, their benefits, and how their application can typically be divided into keypoint extraction, descriptor calculation, and matching steps. Subsequently, we present a novel point feature developed throughout this work, namely the Normal Aligned Radial Feature (NARF). It operates on range images and has the special property that it explicitly considers the boundaries of objects as a source of information. We present experiments in which we analyze the individual components with respect to their repeatability and matching capabilities. Following this chapter, we will present several systems that build upon this point feature type to address hard perception problems.

* * * *

Many perception tasks in the area of robotics, like object recognition, place recognition, mapping, or localization, require the ability to find similar parts in different sets of sensor readings. A popular method is to extract *point features*, which describe a chunk of data in a form that can be used to efficiently perform comparisons between different regions. In 2D (e.g., photos) or 3D (e.g., range scans) perception, such features are usually *local* around a point. In this context this means that for a given point in the scene its immediate vicinity is used to determine the corresponding feature. We will refer to these kinds of features as *point features*. The extraction and usage of point features from the raw sensor data can typically be divided into several subtasks.

In the point feature extraction phase the system tries to find *keypoint*s/*interest point*s. This describes the identification of informative (interesting) positions in the data that are useful as point features. These are areas with well defined positions that allow a robust detection under noisy conditions or sensor viewpoint changes. This means that a keypoint should be found in the same place again, if the corresponding area is reobserved. This condition is also known as *repeatability*. The next step is the calculation of *descriptor*s/*description vector*s. These describe the area around a keypoint in form of a set of numbers $\mathbf{d} = \{d_1, \ldots, d_n\}$ with $d_i \in \mathbb{R}$. This description should encode the information in the vicinity of the point in a way that allows for easy similarity checks between different descriptors. Therefore, the description has to be as unique as possible, while still being robust to noise and changes in the sensor position. Last comes the matching phase. After the features have been extracted from the data, they can be used to find similar structures. For this purpose, the descriptors of different features can be compared in a pair-wise fashion. Typically the descriptors allow the *descriptor distance* to be

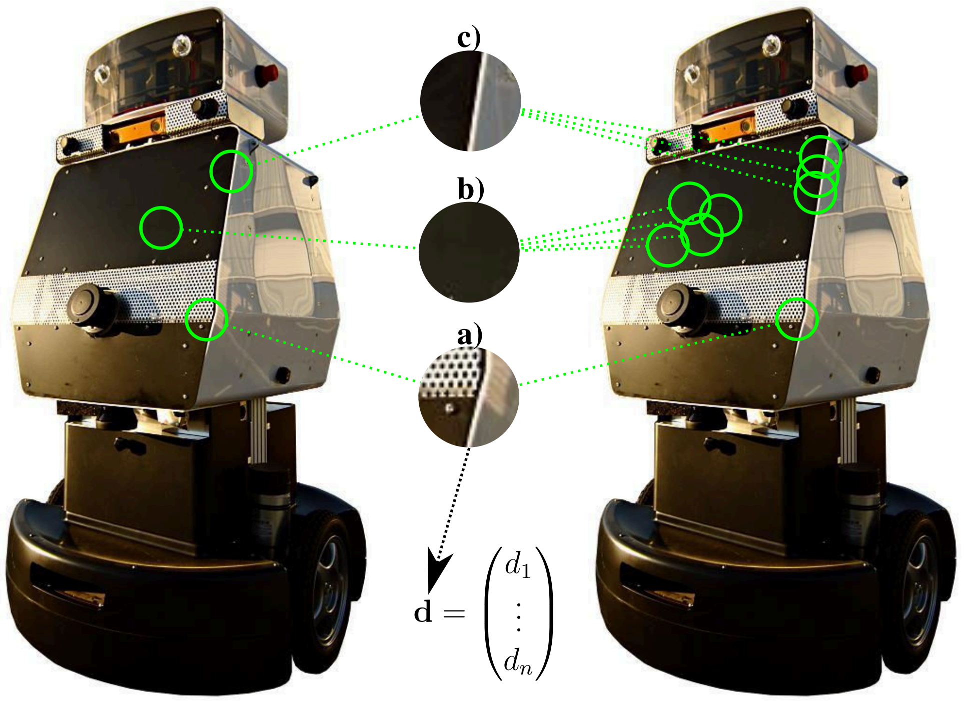

c)
b)
a)
$\mathbf{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix}$

Figure 3.1: Left: Example image with marked feature positions. **Middle**: Scaled up versions of the areas covered by the features. **Right**: Examples for positions, where the features could be found because of a similar structure. Feature **a)** has a clear match. The descriptor **d**, which can be used to determine such a match, is calculated from the image patch surrounding the keypoint in this two dimensional example case. The structures **b)** and **c)** are not unique in their immediate neighborhood and therefore have multiple candidate matches.

calculated by means of a simple norm (e.g., Euclidean Distance or Manhattan Distance). A low descriptor distance implies a high similarity.

Figure 3.1 gives a visual example for the feature extraction on a 2D camera image. Only the structure marked with a) is a good keypoint since its position is clearly defined in a local neighborhood. The structure marked with b) can be moved around arbitrarily on the dark surface and will still lead to very similar descriptors. The structure marked with c) has a line structure which leads to the situation that its position is clearly defined in one dimension, but not in the other one. Slight movements along the line will lead to very similar descriptors. The property describing this is called *self-similarity*. Feature a) is the only one with a low self similarity, because even slight changes in its position will lead to very different descriptors.

Important advantages of keypoints are that they substantially reduce the search space and the computation time required to find correspondences between two scenes. In addition, they focus the computation on areas that are more likely relevant for the matching process.

In 2D computer vision, many successful point features have been developed in the past. Some of the most popular approaches in this area are the SIFT (Scale Invariant Feature Transform) keypoint detector and descriptor [83], the SURF (Speeded Up Robust Features) keypoint detector and descriptor [9], the FAST (Features from Accelerated Segment Test) keypoint detector [102], and more recently the BRIEF (Binary Robust Independent Elementary Feature) descriptor [18] and the BRISK (Binary Robust Invariant Scalable Keypoints) keypoint detector and descriptor [79]. These and other methods have been used by researchers all over the world to detect known objects [83] or places [25] in images, create classifiers for object types [118] or scene types [24], implement visual SLAM systems [30], and many other tasks that require a similarity measure for images or image parts. Important properties for point feature types are often *rotational invariance* and *scale invariance*, meaning that correspondences should also be found if the observations differ regarding the orientation and size of the patches. Yet, there are also applications where these properties are not desirable and should therefore be optional.

Instead of 2D vision data, we focus on feature extraction from 3D range scans in this work. Compared to cameras, 3D sensors provide depth information and are often less sensitive to lighting conditions (e.g., laser sensors). In addition, scale information is directly available. Yet, geometry alone is often less expressive in terms of object uniqueness. Compared to vision, there has been surprisingly little research for keypoint extraction in raw 3D data in the past. We will focus on single range scans, as obtained with 3D laser range finders or camera based depth sensors, where the data is incomplete and dependent on a viewpoint. We chose range images as the way to represent the data since they reflect this situation and enable us to adapt ideas from computer vision. See Section 2.3.2 for more details on range images.

In the remainder of this chapter, we present the Normal Aligned Radial Feature (NARF), a novel keypoint extraction method combined with a feature descriptor for points in 3D range data. The NARF extraction procedure has been designed with two specific goals in mind:

- The selected points are supposed to be in positions where the surface is stable (to ensure a robust estimation of the normal) and where there are sufficient changes in the immediate vicinity to ensure the repeatability of the extraction procedure.

- Since we focus on partial views, we want to make use of object boundaries, meaning the outer shapes of objects seen from a certain perspective. The outer forms are often rather unique so that their explicit use in the keypoint extraction and the descriptor calculation can be expected to make the overall process more robust. For this purpose, we also present a method to extract those boundaries.

The implementation of our method is available under an open-source license. See Section 1.2 for a list of contributions to open source software.

The remainder of this chapter is organized as follows. We will first introduce a method to find object boundaries in range images in Section 3.1. Subsequently, we will describe the NARF keypoint extraction method in Section 3.2 and the NARF descriptor in Section 3.3. Following this, we will present experimental results in Section 3.4. We will discuss related work in Section 3.5 and conclude this chapter in Section 3.6.

3.1 Boundary Extraction on Range Images

One important requirement for our feature extraction procedure is the explicit handling of boundaries in the range data. Boundaries typically appear as non-continuous traversals from foreground to background. In this context there are mainly three different kinds of points that we are interested in detecting:

- *Object boundaries* are the outermost visible points still belonging to an object. These points are relevant for the feature extraction because they define the outer form of an object.

- *Shadow boundaries* are points in the background that adjoin occlusions. The detection of these points is important since the feature extraction should be prevented from using them as relevant structure. Keypoints on the shadow boundaries are not useful because their position changes with the perspective of the sensor.

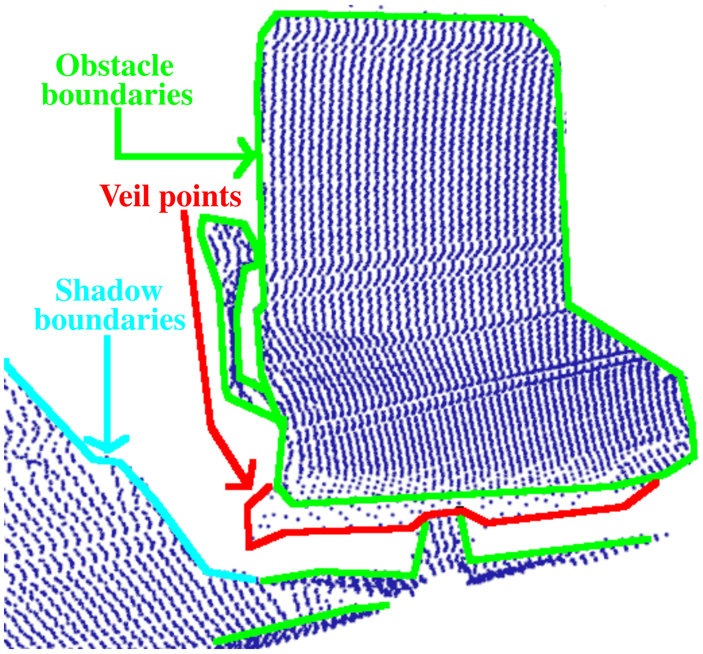


Figure 3.2: Visual example for the different kinds of boundary points in 3D range scans. Obstacle boundaries mark a traversal from foreground to background, whereas shadow boundaries mark a traversal from background to foreground. Veil points typically lie between the former two and are outliers since their range values are interpolated between foreground and background.

- *Veil points* are interpolated points between the obstacle boundaries and the shadow boundaries. Veil points are a typical phenomenon in 3D range data obtained by laser range scanners. Their correct classification is obviously is important since they should not be incorporated in the feature extraction process.

Figure 3.2 shows an example of the different types of boundary points described above.

3.1.1 Boundary Extraction Overview

There are different indicators that are useful for the detection of those boundaries in a range image, like acute impact angles or changes of the normals. In our practical experiments, we found that the most significant indicator, which is also very robust against noise and changes in resolution, is a change in the distance between neighboring points. We will use this feature to classify boundaries according to the following steps. For every image point consider its local neighborhood and

- employ a heuristic to find the typical 3D distance to neighboring points that are not across a boundary.
- use this information to calculate a score for how likely it is that this point is part of a boundary,
- identify the class to which this boundary point belongs, and
- perform non-maximum suppression to find the exact boundary position.

3.1.2 The Boundary Extraction Algorithm

Please note that since we are using range images, every point has a 2D position (the position of the pixel in the image) and a 3D position (the measured position in the world coordinate frame). The same applies for distances between points. We will use the term "2D distance" to refer to the distance between the pixels in the image and we will use the term "3D distance" to refer to the Euclidean distance between the 3D points.

First we analyze every image point and apply a heuristic to find out what 3D distance a certain point typically has to its 2D neighbors, which belong to the same surface. To detect this we use a concept similar to the idea of Bilateral Filtering [4]. For each point $\mathbf{p}_i$ in the range image we select all neighboring points $\{\mathbf{n_1}, \ldots, \mathbf{n}_{s^2}\}$ that lie in the square of size s, centered around $\mathbf{p}_i$. Then we calculate their 3D distances $\{d_0, \ldots, d_{s^2}\}$ to $\mathbf{p}_i$ and sort this set in increasing order to get $\{d'_0, \ldots, d'_{s^2}\}$. Assuming that at least a certain number c of the points lie on the same surface as $\mathbf{p}_i$, we select $\hat{d} = d'_c$ as a typical distance to $\mathbf{p}_i$-s neighbors, that do not include points beyond a boundary. In our implementation we selected $s = 5$ and $c = \left(\frac{(s+1)}{2}\right)^2 = 9$, which would be the highest c to still get a correct value for a point lying on the tip of a right angle corner. In the next step we calculate four scores for every image point, describing the probability of having a boundary on the top, left, right, or bottom. We will only explain the procedure for the direction to the right as the other three are carried out accordingly. Let $\mathbf{p}_{x,y}$ be the 3D point at the pixel position x, y in the image. We calculate the average 3D position of some of its neighbors on the right as

$$\mathbf{p}_{\text{right}} = \frac{1}{m_p} \sum_{i=1}^{m_p} \mathbf{p}_{x+i,y}, \tag{3.1}$$

where m_p is the number of points used for the calculation of the average (3 in our implementation). We take this average instead of just the neighbor $\mathbf{p}_{x+1,y}$ to account for noise and the possible existence of veil points. Next, we calculate the 3D distance $d_{\text{right}} = ||\mathbf{p}_{x,y} - \mathbf{p}_{\text{right}}||_2$ and a score based on the quotient of d_{right} and $\hat{d}$ as

$$s_{\text{right}} = \max\left(0, 1 - \frac{\hat{d}}{d_{\text{right}}}\right). \tag{3.2}$$

This gives us a value in $[0, 1)$, where high values indicate a substantial increase between the typical neighbor distance and the distance to the points on the right, indicating a probable boundary.

Many sensors return information about far ranges, meaning distances that are outside of their measuring capabilities. An integration of these far range pixels in the range image is useful because a traversal to far ranges typically marks a boundary. We therefore give such traversals a score of 1.0. Next, we apply a smoothing operation on the score values to achieve continuous boundaries and avoid disruptions originating from sensor noise.

To determine if a given point $\mathbf{p}_{x,y}$ belongs to the foreground or to the background, we have to check if its range value is lower or higher than the range of $\mathbf{p}_{\text{right}}$. A lower value indicates an obstacle boundary, a higher value indicates a shadow boundary. For all points $\mathbf{p}_{x,y}$ that are potential obstacle boundaries, we now search for a corresponding shadow boundary to the right, selecting the one with the highest score in a maximum 2D distance (3 pixels in our implementation). Depending on the score s_{shadow} of this potential shadow boundary we slightly decrease s_{right} according to

$$s'_{\text{right}} = \max(0.9, 1 - (1 - s_{\text{shadow}})^3)\ s_{\text{right}}. \tag{3.3}$$

We thereby reduce the score by up to 10% for small values of s_{boundary}, meaning areas where no clear shadow boundary is apparent.

In a last step, we check if s'_{right} is above a threshold (0.8 in our implementation) and if it is a maximum regarding $\mathbf{p}_{x-1,y}$ and $\mathbf{p}_{x+1,y}$. If this is the case, we mark $\mathbf{p}_{x,y}$ as an obstacle boundary, its counterpart from above as a shadow boundary, and all pixels in between as veil points. Figure 3.3(a) and (c) display an example of the output of this procedure. In this figure, the different kinds of boundary points are color coded.

Now that we have the information about the boundaries, we can use it in the keypoint extraction procedure, which will be explained in the next section.

3.2 The NARF Keypoint Detector

The detection of keypoints is an important step to reduce the search space for feature extraction and focus the attention on significant structures. We have the following requirements for our keypoint extraction procedure:

- It must take information about boundaries and the surface structure into account.
- It must select positions that can be reliably detected even if the object is observed from another perspective.
- The points must be on positions that provide stable areas for normal estimation or the descriptor calculation in general.

We will now first give an overview over our algorithm and how we approach the points above and then present the procedure in detail.

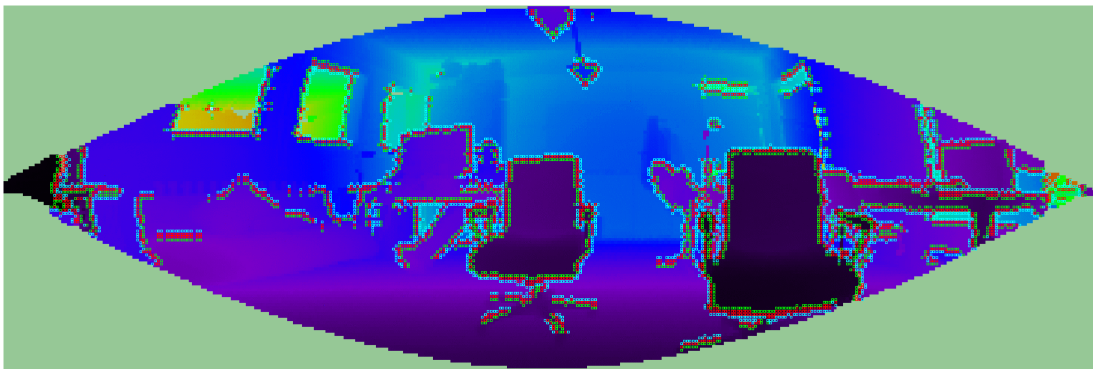
3.2.1 NARF Keypoint Detection Overview

Stable keypoints need significant changes of the surface in a local neighborhood to be robustly detected in the same place even if observed from different perspectives. This typically means that there are substantially different dominant directions of the surface changes in the area. To capture this, we

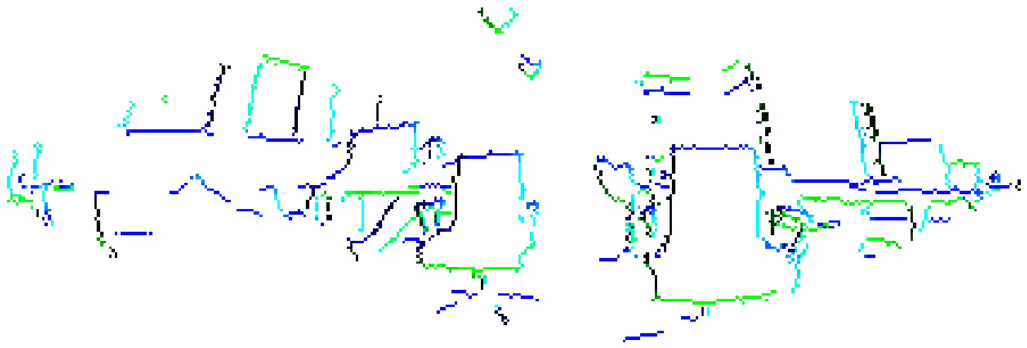
- consider the local neighborhood of every image point and calculate a score how much the surface changes at this position and determine a dominant direction for this change, also incorporating the information about boundaries,
- look at the dominant directions in the surrounding of each image point and calculate an interest value that represents both how much these directions differ from each other and how much the surface in the point itself changes (meaning how stable it is),
- perform smoothing on the interest values, and
- perform non-maximum suppression to find the final interest points.

The most important parameter of this process is the *support size* ρ, which is the diameter of the sphere around the keypoint, which includes all points whose dominant directions were used for the calculation of the interest value. This is the same value that will later be used to determine which points will be considered in the calculation of the descriptor. Choosing the value of ρ

(a)



(b)



(c)

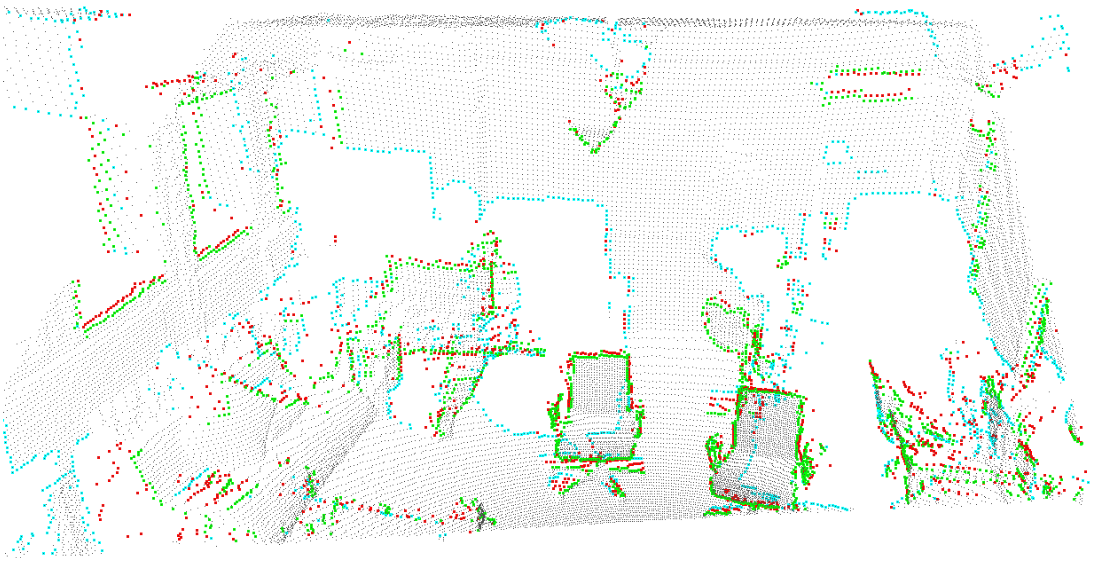


Figure 3.3: **(a):** Visualization of an example for the boundary extraction procedure overlain on a range image. Green points mark object boundaries, red points mark veil points and turquoise points mark shadow boundaries. **(b):** Visualization of the directions of the boundaries with blue to the top, turquoise to the right, green to the bottom and black to the left. **(c):** The same visualization as in (a), but showing the point cloud corresponding to the range image.

depends a lot on the size of the structures that we want to find. In general, the higher the value, the more points are used to calculate the feature, which therefore becomes more stable. But in the context of object recognition it should be smaller than the object itself to have some robustness against partial occlusion. We found in our experiments that 25% of the average object size is a reasonable value. For objects of very different sizes it might be necessary to use multiple scales.

3.2.2 The NARF Keypoint Detection Algorithm

We start by calculating the directions of the boundaries we extracted in the previous step. For each point we know if it has a boundary on its top, right, left, or bottom. Thereby every boundary pixel already encodes the direction of the boundary in steps of $45°$. Please note that the estimation of this quantity can be improved by averaging over multiple boundary pixels in a local neighborhood. Additionally, we use range images in spherical coordinates, which look distorted when visualized in 2D. If the estimation would be done directly in the range image space, this distortion would influence the calculation of 2D directions in the image itself. To prevent this, we perform the calculations on the 3D points corresponding to the pixels, using local normals. We estimate the normals using Principal Component Analysis (PCA) on a local 2D neighborhood of the points, where we disregard neighbors with 3D distances above $2\hat{d}$ (see Section 3.1.2).

Since we also want to consider the changes on the surfaces that are not related to boundaries, we calculate the principal curvature directions at each point, which gives us the principal direction and the magnitude λ (the largest eigenvalue) of the curvature. Every point in the image $\mathbf{p}_i$ gets an associated main direction $\mathbf{v}$, which is the boundary direction in every boundary point (see Figure 3.3(b)) and the principal direction in every other point. All of these directions get a weight w that is 1 for every boundary point and $1-(1-\lambda)^3$ for every other point (this expression scales the magnitude upwards, while keeping it in $[0,1)$). Figure 3.4(b) shows an example of the values of w for the range image shown in Figure 3.4(a).

Up to now, all the calculations were done on fixed 2D pixel radius surroundings. From now on, the actual 3D support size ρ will be used. As long as enough points are available inside of the sphere with diameter ρ, this will make the method invariant to resolution, viewing distance and non-uniform point distributions.

For every image point $\mathbf{p}$ we consider all its neighbors $\mathcal{N}=\{\mathbf{n}_1,\mathbf{n}_2,\ldots\}$ that are inside of the support size (meaning that the 3D distance to $\mathbf{p}$ is below $\frac{\rho}{2}$) and do not have a boundary in between. Each $\mathbf{n}\in\mathcal{N}$ has a main direction $\mathbf{v_n}$ and a weight $w_\mathbf{n}$. To get back to 2D direction vectors, which helps us reduce the influence of noise from the normal estimation, we project the directions onto a plane perpendicular to the viewing direction (the line from the sensor to $\mathbf{p}$). This leads to a one dimensional angle $\beta_\mathbf{n}$ for each $\mathbf{n}$.

Since two opposite directions do not define a unique position and since the principle curvature analysis does not provide a unique direction, we transform the angles in the following way:

$$\beta' = \begin{cases} 2\,(\beta-180°) & \text{for} \quad \beta > 90° \\ 2\,(\beta+180°) & \text{for} \quad \beta <= -90° \end{cases} \tag{3.4}$$

which is a value $\in(-180°,180°]$, but a difference of $180°$ of the resulting angles is actually just a difference of $90°$ in the original angles, which is where the wrap-around occurs now. We furthermore smooth all the weights and angles by applying a bounded Gaussian kernel.

We now define the interest value $i(\mathbf{p})$ of 3D point $\mathbf{p}$ as follows:

$$
\begin{aligned}
i_1(\mathbf{p}) &= \min_{\mathbf{n}\in\mathcal{N}}\left(1 - w_{\mathbf{n}}\ \max\left(0, 1 - \frac{10\ ||\mathbf{p}-\mathbf{n}||}{\rho}\right)\right) && (3.5)\\
f(\mathbf{n}) &= \sqrt{w_{\mathbf{n}}\ \left(1 - \left|\frac{2\ ||\mathbf{p}-\mathbf{n}||}{\rho} - \frac{1}{2}\right|\right)} && (3.6)\\
i_2(\mathbf{p}) &= \max_{\mathbf{n}_1,\mathbf{n}_2\in\mathcal{N}}(f(\mathbf{n}_1)\ f(\mathbf{n}_2)(1 - |\cos(\beta'_{\mathbf{n}_1} - \beta'_{\mathbf{n}_2})|)) && (3.7)\\
i(\mathbf{p}) &= i_1(\mathbf{p})\ i_2(\mathbf{p}) && (3.8)
\end{aligned}
$$

The Term i_1 scales the interest value i downwards, if $\mathbf{p}$ has neighboring points with high weights close by, which indicates strong surface changes. The term thereby satisfies our desired property to put keypoints only on locally stable surface positions. The term i_2 increases the interest value if there is a pair of neighbors with very different and strong main directions in the vicinity. After i is calculated in every image point, we perform an additional smoothing of the values over the image.

In a final step we now select all maxima of i above a threshold as keypoints. See Figure 3.4(c) and (d) for an example, where the values of i for two different values of ρ are visualized and the keypoints are marked. Note, how the keypoints are in the corners of the chairs for a small support size, whereas they move more to the middle for higher values.

We are now able to extract keypoints on a 3D scene. Our next goal is to create descriptors for these keypoints.

3.3 The NARF Descriptor

Feature descriptors characterize the area around a keypoint in a way that makes efficient comparison regarding similarity possible. Our goals in the development for the NARF descriptor were as follows:

1. The descriptor should captures the existence of occupied and free space, so that parts on the surface and also the outer shape of an object can be described.

2. The descriptor should be robust against noise on the keypoint position.

3. The descriptor should enable us to extract a unique local coordinate frame at the point.

To address issue 3, one has to eliminate six degrees of freedom, namely the 3D position and the 3D angle of the transformation between the original coordinate frame and the coordinate frame of the feature. If we choose the position of the point feature as the origin, only the three angles remain. The normal vector at the point can be used to eliminate two more degrees of freedom, which leaves the rotation around the normal to be determined. While many feature descriptors in 3D are invariant to the rotation around the normal (like Spin Images [65]), or even the complete 3D orientation [108], it is helpful to have the information about this orientation available for multiple reasons. For one, it might be desirable to be able to not use a unique orientation, e.g., if we only search for correspondences with a fixed patch orientation, as in the case of a wheeled robot searching for correspondences between its map and the environment. An invariance regarding the robot's roll might unnecessarily increase the size of the search space since the robot will operate at roll angle zero most of the time. On the other hand, in cases

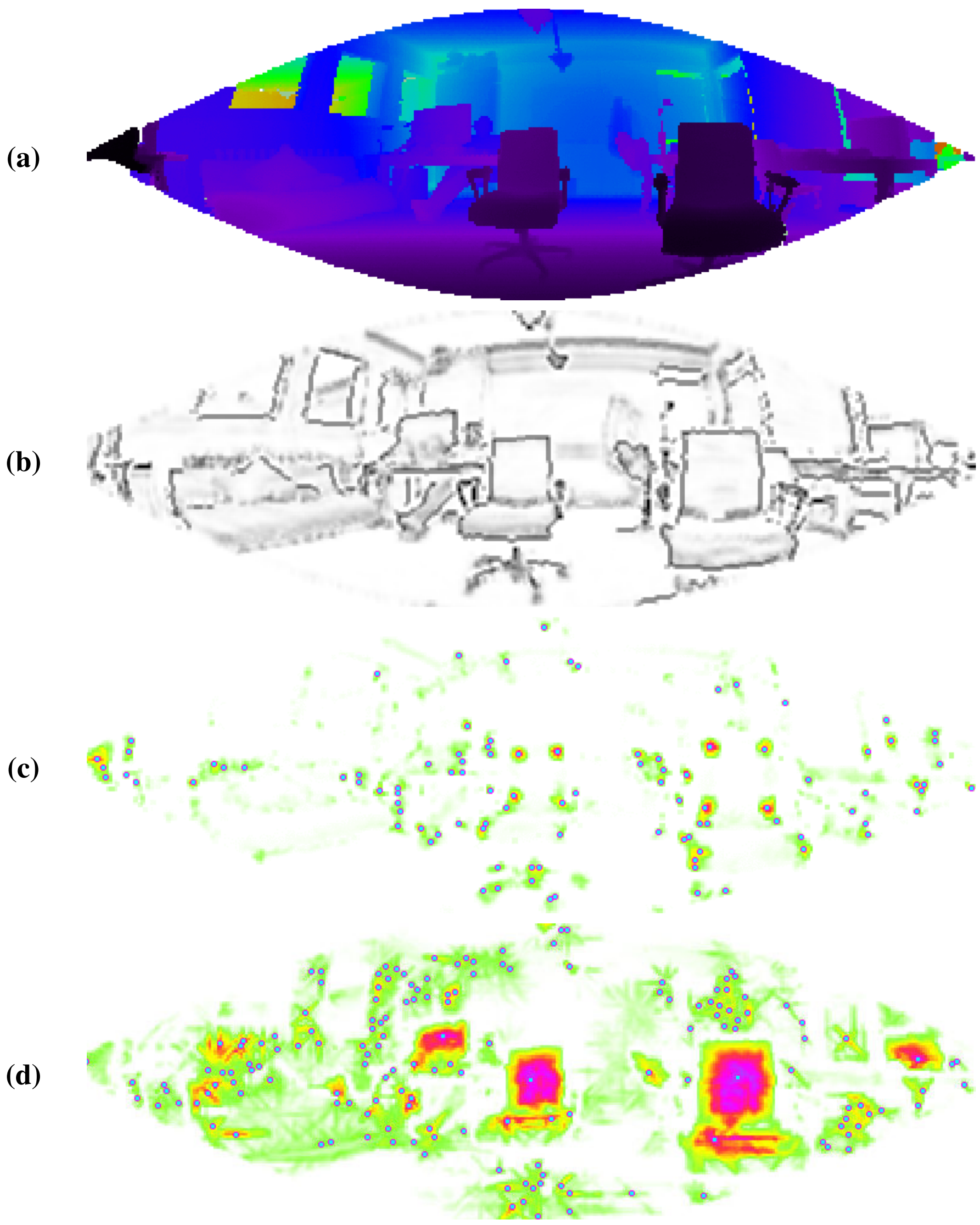


Figure 3.4: The keypoint extraction procedure. **(a)**: Range image of our example scene. **(b)**: Surface change scores according to boundaries and principle curvature. **(c)**: Interest values with marked keypoints for a support size of 20 cm. Note how the corners of the chairs are detected as keypoints at this scale. **(d)**: Interest values with marked keypoints for a support size of 1 m. Note how the whole surfaces of the backrests of the chairs contain one keypoint at this scale.

where the unique orientation is used, it enables additional filtering for consistent local coordinate frames between features. The NARF descriptor enables us to extract a unique orientation around the normal. The underlying idea is similar to what is done in SIFT [82] and SURF [9]. Yet, unlike its 2D siblings, this orientation together with the normal defines a complete 6DOF transformation at the position of the keypoint.

3.3.1 NARF Descriptor Calculation Overview

To compute the NARF descriptor in a keypoint, we perform the following steps. Please consider Figure 3.6(a,b) for a visual example of the process.

1. We calculate a normal aligned range value patch in the point, which is a small range image with the observer looking at the point along the normal and a size determined by the support size.
2. We overlay a star pattern onto this patch, where each beam corresponds to one value in the final descriptor. This value captures how much the pixels under the beam change.
3. We extract a unique orientation from the descriptor,
4. We shift the descriptor according to this value to make it invariant to the rotation.

Steps 3 and 4 are optional, if complete rotational invariance is not necessary, as explained above. In principal, the normal aligned range value patch (NARVP) mentioned in step 1 can also be used as a descriptor. Its advantage is that it actually preserves most of the information about the original 3D structure, apart from the resolution. Yet the procedure described in step 2 is supposed to increase the robustness of the descriptor regarding noise, especially regarding the keypoint position. We will show this advantage in the experimental evaluation (see Section 3.4).

3.3.2 The NARF Descriptor Algorithm

As mentioned above, the calculation of the NARF descriptor builds upon a normal aligned range value patch (NARVP) extracted from the feature position in the scan. Such a NARVP can be calculated by creating a local coordinate system with the origin in the keypoint position, the z-axis facing in the normal direction and y being oriented according to the upright vector in the world coordinate frame. We then transform all points within the support radius $\frac{\rho}{2}$ (see Section 3.2.1) into this coordinate frame. The resulting x and y coordinates define the cell of the descriptor in which a point falls, and the minimum over all z values is the value of a cell. A cell where no 3D points falls into gets the maximum value of $\frac{\rho}{2}$. See Figure 3.5 for a visualization of the process.

The normal vector at the feature point is calculated using PCA on all points that will be used to calculate the descriptor to maximize its stability. The size of the image patch should be high enough to keep enough descriptive structure, but low enough to not surpass the typical resolution of the scan. We chose a size of 10×10 pixels in our experiments. To prevent problems in areas where the resolution of the original scan is low, interpolation between cells or usage of ray tracing is necessary. In the next step we put a Gaussian blur onto the patch. Then we project a star shaped pattern onto it (see Figure 3.6(b)). Every beam from the star pattern corresponds to one value in the descriptor $\mathbf{d}$. We chose $|\mathbf{d}| = 36$ for our experiments, which means 10° between consecutive beams. For each beam we select the set of cells $\{c_0, \ldots, c_m\}$ that the beam crosses, with c_0 being the middle of the patch and the rest ordered according to

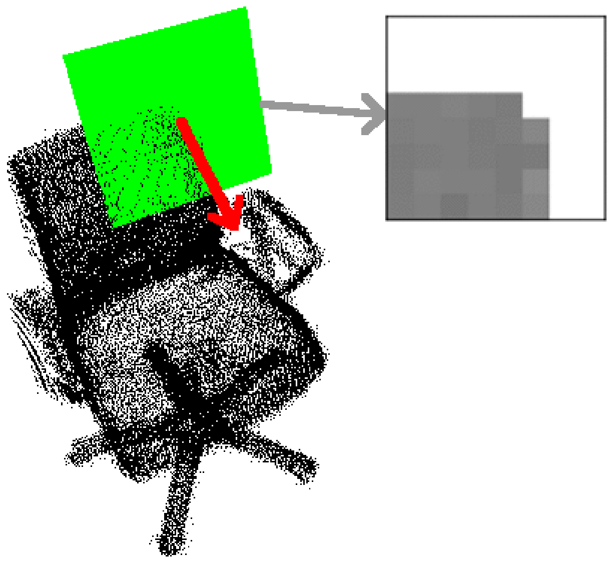


Figure 3.5: Here we visualize, how the Normal Aligned Range Value Patch (NARVP) is calculated. The green patch is centered around the keypoint and oriented according to the normal vector (the red arrow) at the point. The cells of the NARVP, which can be seen on the top right, receive values according to the minimum distance of the points that fall into them.

the distance to c_0. We define the value d_i of the descriptor cell corresponding to the i-th beam as

$$w(c_j) = 2 - \frac{2\,||c_j - c_0||}{\rho} \tag{3.9}$$

$$d'_i = \frac{\sum_{j=0}^{m-1} \left(w(c_j)\,(c_{j+1} - c_j)\right)}{\sum_{j=0}^{m-1} w(c_j)} \tag{3.10}$$

$$d_i = \frac{\text{atan2}\left(d'_i,\, \frac{\rho}{2}\right)}{180^\circ}, \tag{3.11}$$

where $w(c_j)$ is a distance-based weighting factor that weights the middle of the patch with 2 and decreases to 1 towards the outer edges of the patch. The basic intuition for d'_i isthe following. The closer to the center a change in the surface is, and the stronger the change is, the more the beam value will deviate from 0. The step from d'_i to d_i is for normalization purposes and scales every cell to [-0.5, 0.5]. Please consider Figure 3.6(a) and (b). At the bottom of (b) the final descriptor is visualized and the arrows mark corresponding beams. Beams that lie on a flat surface have low values, whereas beams going over the boundary have high values.

Up to now the descriptor was not invariant to the rotation around the normal. We now try to find one or more unique orientations for the descriptor. For this purpose we discretize the possible 360° into a number of bins and create a histogram. The value for a histogram cell corresponding to angle β is

$$h(\beta) = \frac{1}{2} + \frac{1}{|\mathbf{d}|}\sum_{i=1}^{|\mathbf{d}|} d_i \left(1 - \frac{|\beta - \gamma_i|}{180^\circ}\right)^2, \tag{3.12}$$

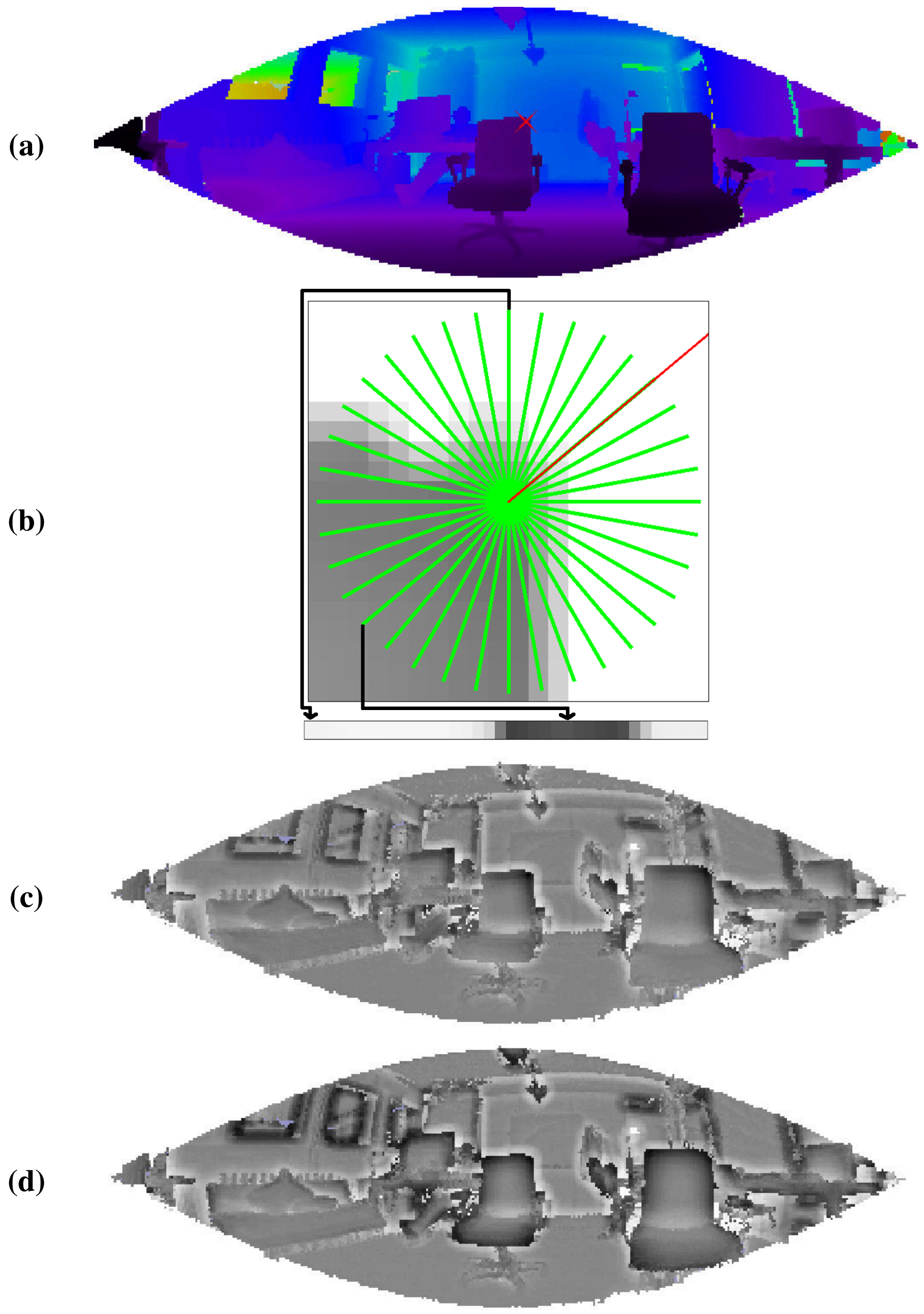


Figure 3.6: **(a)**: An example range image. The red cross marks the position where we extracted the NARF descriptor. **(b)**: Visualization how the descriptor is calculated. The background image behind the star pattern shows a normal aligned range value patch (NARVP) of the top right corner of chair. The actual descriptor is visualized below. Each of the 36 cells of the descriptor corresponds to one of the green beams visualized in the patch, with two of the correspondences marked with arrows. The top most beam corresponds to the first descriptor value and then revolves clock-wise. The additional red beam pointing to the top right shows the extracted dominant orientation. The intuition is that the more the structure below the beam changes, the higher the corresponding descriptor value. **(c)**: The descriptor distances to every other point in the scene (the brighter the higher the value). Note that mainly top right rectangular corners get low values. **(d)**: Same as (c), but for the rotationally invariant version. In this case, all rectangular corners get low values.

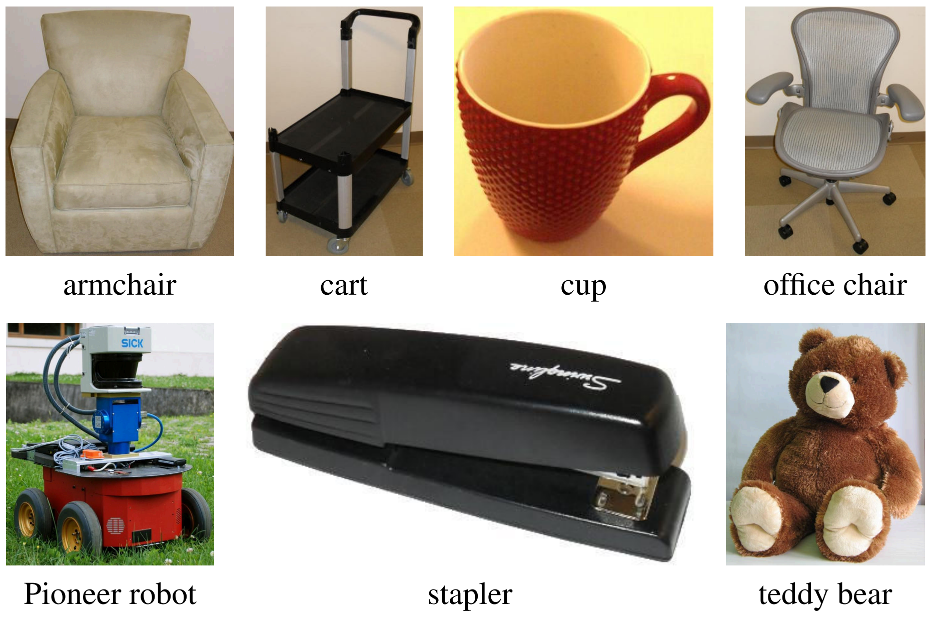


Figure 3.7: Objects used for the experiments. For each of these objects we obtained a complete point cloud model.

where γ_i is the angle corresponding to the i-th descriptor cell. We select the histogram bin with the maximum as the dominant orientation of the patch. If there is another cell with a value above 80% of the maximum, we create a second feature with this orientation. We can now shift the descriptor to create the rotationally invariant version.

The resulting descriptors can now easily be compared using standard distance functions. We chose the Manhattan distance divided by the number of cells in the descriptor, which normalizes the distance to values in $[0, 1]$. This means the so called *descriptor distance* of two NARF descriptors $\mathbf{d}$ and $\mathbf{d}'$ (with $|\mathbf{d}| = |\mathbf{d}'|$) is defined as:

$$\delta(\mathbf{d}, \mathbf{d}') = \frac{||\mathbf{d} - \mathbf{d}'||_1}{|\mathbf{d}|} = \frac{1}{|\mathbf{d}|} \sum_{i=0}^{|\mathbf{d}|} |d_i - d'_i|. \tag{3.13}$$

Figure 3.6(c) visualizes all the descriptor distances to the selected point in the range image without the rotational invariance. Figure 3.6(d) shows the same for the version with rotational invariance. While the first clearly distinguishes upper right corners from other structures, the latter distinguishes all right angle corners from the other structures, without focus on a specific alignment.

3.4 Experiments

We will now present our experimental evaluation of the NARFs, in which we analyze the keypoint stability and the matching performance of the descriptors.

3.4.1 Keypoint Stability

In a first experiment, we analyze how stable the position of the keypoints is relative to changes in scale (distance to the object) and viewing angle. For this purpose we selected seven object models in the form of complete point clouds. These models represent very different kinds of objects and include furniture-sized objects and tabletop objects. See Figure 3.7 for a list of the models. To be able to evaluate objects of very different sizes together, we scaled all the models to a bounding sphere with a diameter of 1.0 m. We used $\rho = 0.25\,\mathrm{m}$ for the support size, which is large enough to cover most of the significant structure of the models, but low enough to account for partial occlusions. For each of these objects we simulated 50 noiseless views from different angles and distances around the object. Then we simulated 100 different views with additional noise on the point positions. See Figure 3.8(c) for example views with marked keypoint positions. We then compared the keypoint positions on the first set of views with each view of the other set.

We calculate the repeatability of the keypoint extraction according to the method proposed by Unnikrishnan [145], which is a 3D extension of a commonly used method in the computer vision literature [88]. For every keypoint $\mathbf{k}_1$ in one view of the object, we search for the closest keypoint $\mathbf{k}_2$ in another view and calculate the ratio of the intersection between the two spheres with radius $\frac{1}{2}\rho$ around them as

$$\text{repeatability} = 1 - \frac{3}{4}\frac{||\mathbf{k}_1 - \mathbf{k}_2||_2}{\frac{1}{2}\rho} + \frac{1}{16}\left(\frac{||\mathbf{k}_1 - \mathbf{k}_2||_2}{\frac{1}{2}\rho}\right)^3. \tag{3.14}$$

For the calculation of the repeatability we only take unoccluded points into account, meaning we check for every point if it is influenced by self occlusion regarding the current pair of views, and reject it if this is the case.

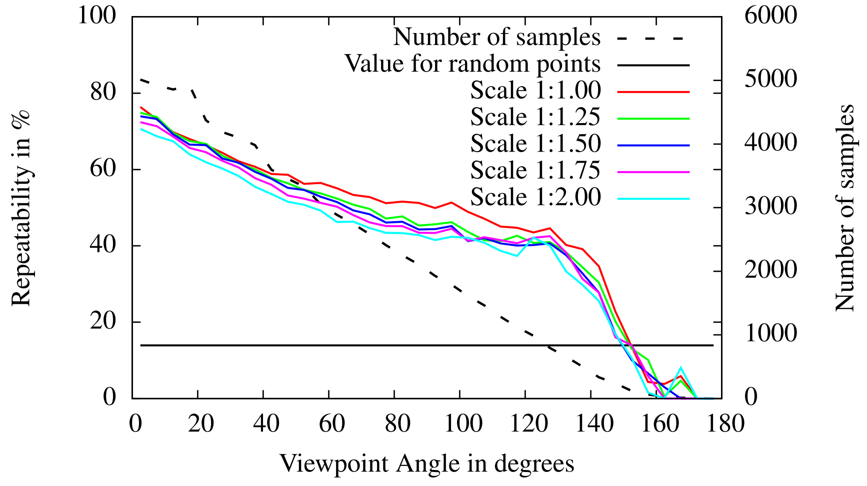
Figure 3.8(a) shows the result of the cross comparison of the positions of the keypoints of all the objects. The scores are shown dependent on the angular viewpoint change and the difference in observing distance (leading to a difference in scale in the range images). While a high change in scale obviously has a negative influence on the keypoint stability (see the different plots for different scales), this influence seems minor compared to angular changes, which is why we will omit changes in scale from now on. The dashed black line shows how many samples were available for the calculation of the averages. This value naturally decreases with the angle because increasingly fewer points are actually visible from both perspectives. The solid black line close to the bottom represents the repeatability value for a random point on the object surface, thereby defining a minimum value below which the results are not meaningful anymore. This plot shows that the keypoint extraction is relatively stable over a wide area of viewpoint changes. For changes below $20°$ the keypoints share about 70% of their support size on average and about 55% for changes below $60°$.

Figure 3.8(b) shows the same analysis, but for every model individually. The cup shows an unusual behavior compared to the rest. This results from the fact that it is rotationally symmetric over a wide range, making the keypoint position much more dependent on the viewpoint. The increase in value between $100°$ and $160°$ is caused by keypoints on the left boundary of the cup again being extracted when seen from the other side on the right boundary. The cart, the chair and the pioneer robot models have higher values for the dashed lines. This mainly indicates that a higher number of keypoints was extracted on these models, leading to the situation that random points on the surface are closer to the keypoint positions on average.

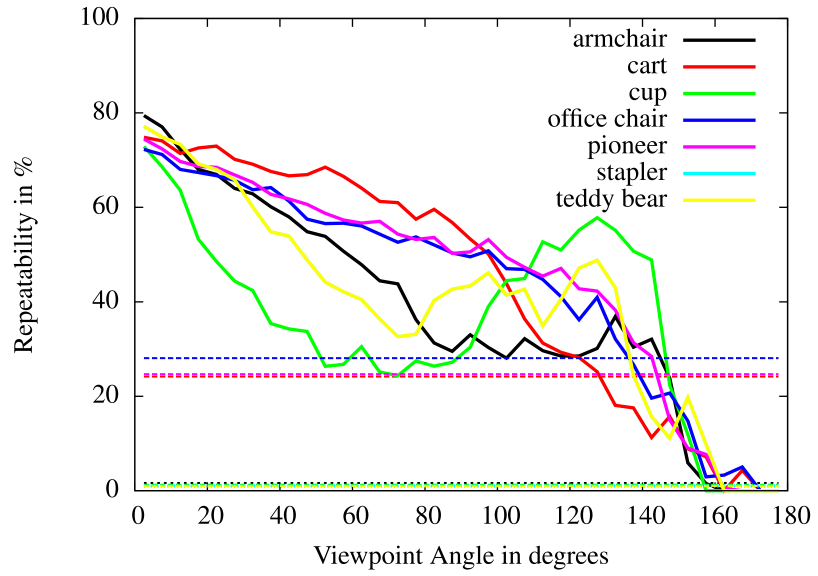
3.4.2 Matching Capability

In another experiment we tested the capability of the NARF-descriptors to match features from an object model to the features in a scene containing the model. For this purpose we collected 10 scenes with a 3D laser range finder in a typical cluttered office environment and 10 scenes with a stereo camera system of tabletop scenes. We artificially added our models, including

(a)



(b)



(c)

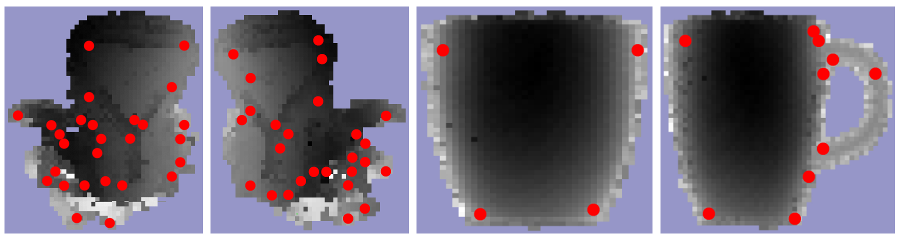


Figure 3.8: (**a**): This graph shows how repeatable the keypoint detection is regarding differences in the viewpoint angle and differences in the scale between two views of the object (the five solid plots). A scale of 1:1 means the object was seen at the same distance, 1:2 means at double the distance. The dashed line shows how many samples were available for averaging. The number goes down with increased angle since less and less pairs of points are actually visible from both perspectives. The solid black line follows from the average distance of a random point on the models surface to the next keypoint, giving a minimum value for the repeatability, below which the value is meaningless. (**b**): Same as (a), but per object model. The dashed lines with constant value correspond to the solid black line in the plot in (a). (**c**): Examples of the simulated views used to create (a) showing two views of the backside of an office chair and two views of a cup. The keypoints extracted on the views are marked in the images.

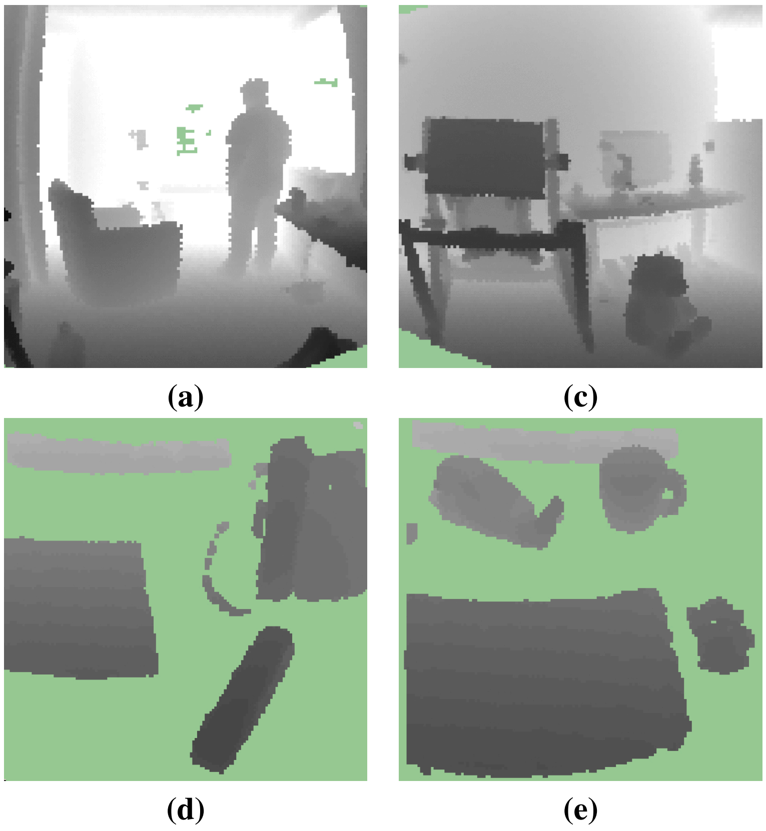


Figure 3.9: Examples for the created scenes. (**a**): Office scene with armchair. (**b**): Office scene with teddy bear. (**c**): Tabletop scene with stapler. (**d**): Tabletop scene with cup.

noise on the 3D point positions, to these scenes - the armchair, cart, chair, robot, and the teddy bear to the office scenes and the stapler and the cup to the tabletop scenes, thereby creating 70 scenes with one known object in each. In this process, the objects could appear in arbitrary yaw and x,y positions, while the height was restricted to the floor/tabletop and roll and pitch were in their natural orientations. The latter were chosen to be able to test our descriptor also without the rotational invariance. For the tabletop scenes, the table plane itself was removed, as it is often done in tabletop object recognition. Figure 3.9 shows some examples of the created scenes.

To match the objects against the scenes, we sampled poses from our models that differed from the ground truth poses between $0°$ and $50°$.

The resulting numbers of true positives and false positives are summarized in Figure 3.10(a) as ROC (Relative Operating Characteristic) curves, which plot the true positives rate against the false positives rate. In this context that means that the bottom left represents a maximum descriptor distance of 0.0 and for the top right a maximum descriptor distance of 1.0. The general interpretation of these curves is that an approach is better, the higher the curve is above the diagonal. Please note the logarithmic scales, which highlight the area with a low number of false positives. The absolute number of false positives is much higher than the absolute number of true positives, which makes areas with a high ratio of false positives less useful. The thicker plots mark areas, where the number of true positives to false positives is lower than 1:10. The plot labeled as 'NARFs with all points' is for our NARF feature descriptors extracted at every image point, without using the keypoints. The plot labeled as 'NARFs with keypoints' shows the performance of the NARF features extracted only at the keypoint positions. The plot labeled as 'NARFs rot. inv. with keypoints' is for the rotationally invariant version of the NARF descriptor. The plot labeled as 'NARVPs with keypoints' is using the Normal Aligned Range Value Patches directly as the descriptor, by just considering them as one-dimensional vectors of range values.

Those curves show that the keypoints are a definite improvement compared to random point positions. The rotationally variant version of the NARF descriptors naturally performs better than the rotationally variant version, because of the reduces search space. Yet, both the rotationally invariant version and the rotationally variant version of the NARF descriptor outperform the NARVP descriptors, which confirms our choice to do the additional computation (see Section 3.3.1).

To evaluate if the system can be used for object recognition, we used the extracted feature matches to actually calculate object poses. Since every feature encodes a local 6DOF coordinate frame, one feature match is enough to calculate an object position and orientation. We calculated a pose for every match with a descriptor distance below 0.05. Figure 3.10(b) shows the average number of correct poses versus false poses for one object and one scene. An object pose was classified as correct if its error compared to the true pose was below 0.3 times the object radius in translation and below $15°$ in rotation. For angular differences below $20°$ there are typically 2 correct poses versus 10 false poses. It would be up to a spatial verification step to reject those false positives. We will address this step in Chapter 4.

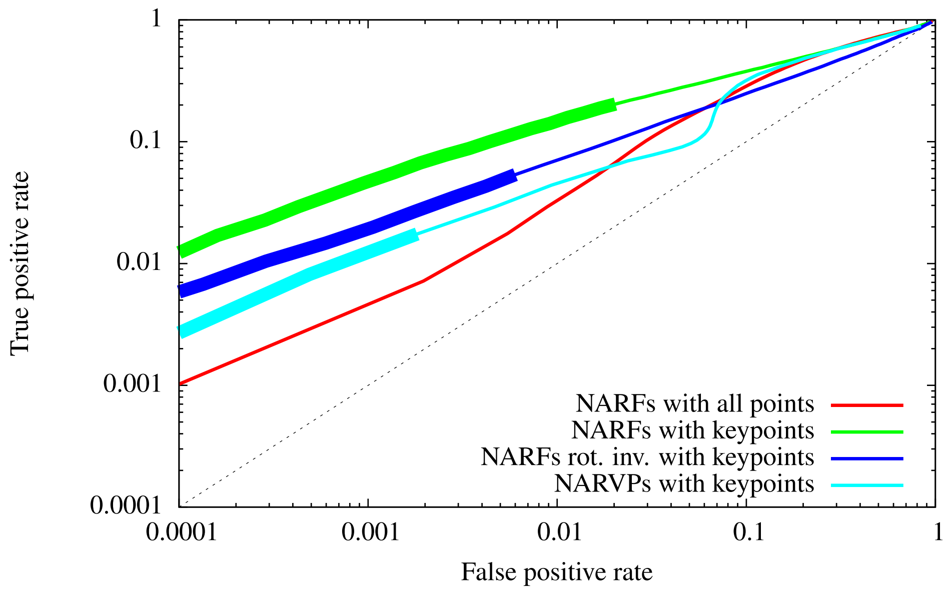
Figure 3.10(c) shows the true poses per used object model. The armchair performs best since it mostly consists of large rectangles that do not change much with moderate changes in the viewpoint. The cup model performs worst, which is due to the low number of keypoints on the model and the symmetrical structure. Yet, the position of the cup, disregarding the orientation, can still be found, as the plot labeled *cup wrong rot* shows.

We also tested at which position in the set of matches (ordered by descriptor distance) the first correct object pose typically occurs. Table 3.1 gives an overview per object depending on the viewpoint angle difference. The first number in the table tells the average position in the set of matches (ordered by descriptor distance) where the first correct object pose occurs. The second is the rate with which a correct position was found. As can be seen, if we disregard the cup, which is a special case as explained above, for viewpoint changes up to $15°$, the first correct pose can typically be found within 3 trials in the 80% where a correct pose could be found. For the rotationally invariant version of NARF, this is slightly worse. For viewpoint changes up to $15°$, the first correct pose can typically be found within 10 trials in the 70% where a correct pose could be found.

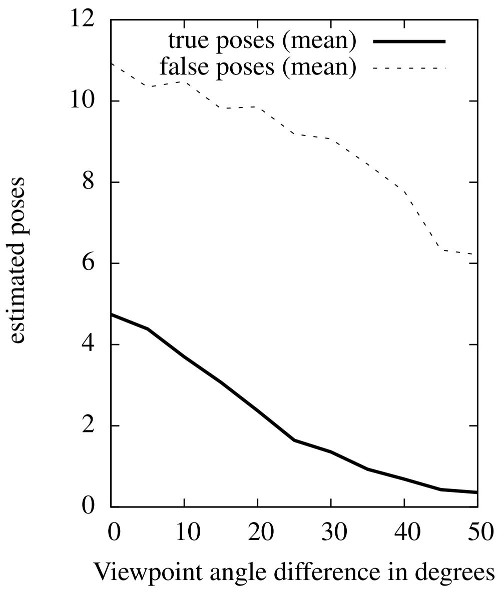
3.4.3 Runtime

The point clouds for the office scenes consist of 11,500 points on average. We used an angular resolution of $\alpha = 0.4°$ for the range image creation, which took 18.1 ms on average to compute. The average runtime for the boundary detection was 22.9 ms, for the interest point extraction 27 ms, and for the feature descriptor calculation with 104 features on average 2.9 ms.

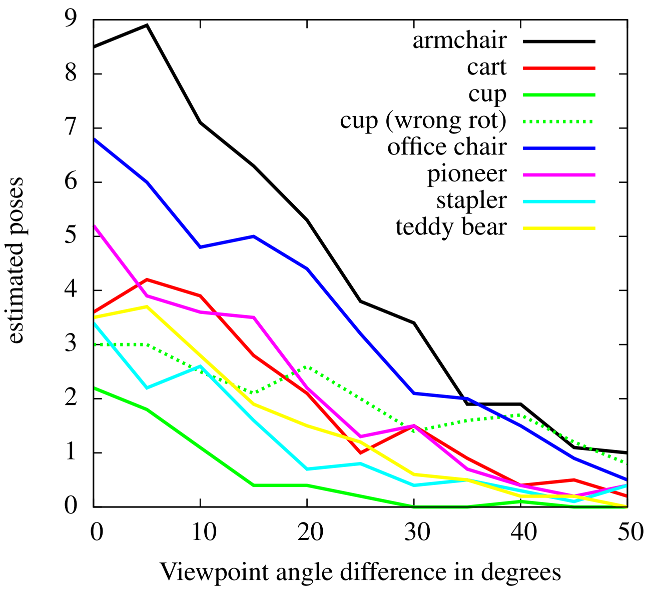
The point clouds for the tabletop scenes had 88,395 points on average. We used an angular resolution of $\alpha = 0.2°$ for the range image creation, which took 26.4 ms on average to compute. The average runtime for the boundary detection was 5.41 ms, for the interest point extraction 6.9 ms, and for the feature descriptor calculation with 48 features on average 1 ms.



(a)



(b)



(c)

Figure 3.10: (**a**): This graph shows ROC curves for feature matches where the threshold for the descriptor distance increases from the bottom left to the top right. Please note that the axes are logarithmic. The higher a line in this graph, the better is the performance. The parts of the plots that are printed thicker mark the areas, where the ratio between true positives and false positives, in absolute numbers, is better than 1:10. (**b**): The average number of true/false poses extracted from the feature matches for all objects. (**c**): The average number of true poses extracted from the feature matches per object. To account for the symmetry of the cup, the plot labeled as 'cup (wrong rot)' gives the number of poses that are correct regarding (x, y, z)-position, but not necessarily regarding orientation.

viewpoint angle	armchair	cart	cup	office chair	pioneer robot	stapler	teddy bear
0°	1.6, 100%	1.2, 90%	1.3, 100%	1.2, 90%	1.9, 100%	1.2, 90%	2.1, 0.8
5°	1.1, 100%	2.6, 80%	1.5, 80%	1.6, 80%	1.6, 70%	2.9, 90%	1.2, 100%
10°	1.6, 100%	1.3, 90%	2, 50%	1.7, 90%	1.2, 80%	2.8, 80%	2, 90%
15°	1.8, 100%	2.4, 80%	1, 30%	1.1, 80%	2.9, 80%	4, 80%	2.9, 80%
20°	1.1, 100%	3.3, 90%	1.5, 20%	1.2, 90%	4.2, 60%	2.6, 50%	4.6, 70%
25°	1.8, 90%	3.5, 40%	2, 20%	3.3, 60%	1.5, 60%	3.8, 60%	2.3, 60%
30°	3.4, 90%	6.8, 60%	−, 0%	2, 60%	5, 60%	4, 30%	1, 30%
35°	2, 80%	4.8, 60%	−, 0%	2.2, 40%	6.4, 50%	3, 40%	3, 30%
40°	2.4, 70%	2.3, 30%	6, 10%	3, 30%	2.7, 30%	3, 20%	1, 10%
45°	3.6, 70%	5.7, 30%	−, 0%	1.5, 20%	5.5, 20%	8, 10%	1, 10%
50°	5, 70%	5.5, 20%	−, 0%	2.5, 20%	2.3, 30%	2.7, 30%	−, 0%

viewpoint angle	armchair	cart	cup	office chair	pioneer robot	stapler	teddy bear
0°	2.9, 90%	6.4, 80%	4.6, 70%	1.1, 80%	5.6, 90%	3, 90%	8, 70%
5°	5.1, 90%	2, 60%	1.4, 70%	2.1, 70%	7.3, 70%	3.7, 70%	3.1, 80%
10°	3.1, 90%	2.8, 80%	3.7, 70%	4.2, 80%	7, 80%	4.6, 70%	2.7, 70%
15°	2.8, 90%	6.9, 70%	1, 10%	2.9, 70%	15, 70%	6.2, 60%	22, 70%
20°	3.4, 90%	12, 60%	1, 10%	12, 80%	15, 60%	23, 50%	21, 40%
25°	5.2, 90%	14, 20%	12, 20%	5.2, 40%	5.7, 30%	3.6, 50%	15, 50%
30°	12, 80%	13, 50%	−, 0%	8, 30%	12, 40%	11, 30%	15, 30%
35°	7.3, 70%	18, 30%	35, 10%	18, 30%	27, 50%	4.5, 20%	10, 20%
40°	5.2, 60%	14, 20%	−, 0%	8.5, 20%	55, 30%	1, 10%	14, 20%
45°	9.4, 50%	7.3, 30%	−, 0%	4, 20%	36, 20%	−, 0%	5, 10%
50°	13, 40%	4, 10%	−, 0%	19, 10%	4, 10%	4, 20%	−, 0

Table 3.1: Position in the set of matches where the first correct object pose typically occurs and the recall rate for the individual objects. The table on **top** shows the results for the NARF version without rotational invariance and the table on **bottom** for the version with rotational invariance.

3.5 Related Work

Two of the most popular systems for extracting keypoints and creating stable descriptors in the area of 2D computer vision are SIFT (Scale Invariant Feature Transform) [82] and SURF (Speeded Up Robust Features) [9]. The keypoint detection and the descriptors are based on local gradients and a unique orientation for the image patch is extracted to achieve rotational invariance. Our approach operates on 3D data instead of monocular camera images. Many of the general concepts of SIFT and SURF, such as the usage of gradients and the extraction of a unique orientation, are useful concepts to be transfered to feature extraction approaches for 3D range scans.

There are also approaches that use features originally designed for vision tasks on special types of images that are calculated from 3D range data. Zhuang *et al.* [154] extract SIFT features on *Bearing Angle Images* [114], which were extracted from range images of the 3D scenes. Li and Olson [81] also create visual images from LIDAR data. They use images from the 2D visualization of laser range data from the birds eye perspective. Their method is usable in 2D and 3D, although a 2D projection of the points is performed in the 3D case. Approaches like these make it possible to use well established feature extraction methods from the vision sector, However, since those features have not been specifically designed for this purpose, they can not make use of range specific information like the known scale and also are more susceptible to viewpoint changes.

One of the most popular descriptors for 3D data is the *Spin Image* presented by Johnson [65], which is a 2D representation of the surface surrounding a 3D point and is computed for every point in the scene. A variant of Spin Images, *Spherical Spin Images* [107], improves the comparison by applying nearest neighbor search using the linear correlation coefficient as the equivalence classes of Spin Images. To efficiently perform the comparison of features, the authors furthermore compress the descriptor. Spin Images do not explicitly take empty space (e.g., beyond object boundaries) into account. For example, for a square plane the Spin Images for points in the center and the corners would be identical, while the feature described in this chapter is able to discriminate between such points.

Li and Gusko [80] presented a keypoint detector and descriptor, which they use for surface alignment. They use a scale-space representation of the input data and use the locations of level difference extrema as keypoints. The descriptors are based on the normal field near the surface. Compared to us, this approach focuses mainly on high quality 3D scans with little noise.

An object detection approach based on silhouettes extracted from range images was presented by Stiene *et al.* [132]. The proposed features are based on a fast Eigen-CSS method and a supervised learning algorithm. This is similar to our work in the sense that the authors also try to make explicit use of boundary information. By restricting the system to boundaries only, however, valuable information regarding the structure of the objects is not considered. Additionally, the extraction of a single descriptor for the complete silhouette makes the system less robust to occlusions.

Huang et al. [63] developed a system for automatic reassembly of broken 3D solids. For this purpose the authors also extract boundaries on the 3D structures, in this case to detect sets of faces. The method detects cycles of edges based on surface curvature. The detected surface parts are then matched to find corresponding fragment parts. In our application the boundary detection finds changes from foreground to background and uses this outer shape in the keypoint detection and matching, compared to dividing the 3D structure into faces that are then matched individually.

Many approaches compute descriptors exhaustively in every data point or use simple sampling methods [65, 43, 86], thereby introducing an unnecessary overhead. Gelfand *et al.* [46] present an approach to global registration using one-dimensional Integral Volume Descriptors (IVD) estimated at certain keypoints in the data. These keypoints are extracted using a self similarity approach in the IVD space, meaning the descriptor of a point is compared to the descriptors of its neighbors to determine areas where there is a significant change. While this method for keypoint extraction explicitly takes the descriptor into account, it becomes impractical for more complex descriptors, which are more expensive to extract. Unnikrishnan [145] presents a keypoint extraction method with automatic scale detection in unorganized 3D point clouds. This approach, however, does not consider any viewpoint related information and does not attempt to place keypoints in stable positions. Rusu *et al.* [111] present a method for selecting keypoints based on their *persistence* in growing point neighborhoods. Given a Point Feature Histogram space [110], multiple descriptors are estimated for several different radii, and only the ones similar between one radius step and the next are kept. The method described in this chapter is by several orders of magnitude faster in the estimation of the keypoints.

Recently, mainly thanks to the new availability of cheap 3D sensors like the Microsoft Kinect or the Asus Xtion Pro Live, the area of feature extraction on 3D range data has gotten more focus in the community, resulting in a few new developments. The *SHOT* (Signature of Histograms of OrienTations) descriptor [139] uses a set of local histograms based on normals that are then grouped together to form the descriptor. The descriptor is made viewpoint invariant using an unique local 3D reference frame. Zhong [153] presented the *ISS* (Intrinsic Shape Signature), which provides a simple keypoint extraction method and a descriptor calculation method. The keypoint extraction is based on an analysis of the eigenvalues of the point scatter matrix of the spherical neighborhood of a point. This measures the variation of the surface in the area and typically returns a high number of keypoints. The descriptor calculation is also based on a local 3D reference frame extracted from the structure and is a histogram of the weighted 3D occupation in its support.

Bo *et al.* [13] use a method based on sparse coding for object recognition and classification with an RGB-D sensor. Sparse coding is based on the idea that surfaces are described using a linear combination of dictionary entries. The dictionary consists of a number of previously learned surface attributes. Compared to NARFs this approach does not purely rely on depth information, but also incorporates vision information.

Behley *et al.* [10] evaluated the performance of different histogram descriptors regarding the classification (ground, vegetation, etc.) of points in urban 3D datasets, collected with different laser based sensors. They also proposed their own descriptor in this context, a spectral histogram. This descriptor is similar to the SHOT descriptor [139] regarding the construction and performed favorably in the classification scenario.

3.6 Conclusions

In this chapter, we presented a novel approach to feature extraction in range images, namely the Normal Aligned Radial Feature (NARF). These point features are generated at locations where the surface is mostly stable but changes significantly in the immediate vicinity. The approach makes explicit use of boundary information, thereby encoding information about the outer shape of an object, which is often very descriptive. In practical experiments, we analyzed the performance of the keypoints and descriptors regarding repeatability and matching capability. We showed that the keypoints are highly repeatable even for substantial viewpoint differences and that the descriptors enable a robust matching of similar structures. We integrated the NARF keypoint extraction and the descriptor calculation into the Point Cloud Library (PCL) [109] as open source, to enable other researchers to use it. In the following chapters we will present several approaches that build upon the results of this chapter to address complex 3D perception problems in the area of robotics, like object recognition, environment modeling, and place recognition. These systems could not have provided such high quality results without a strong underlying feature type.

Chapter 4

Object Recognition using 3D Range Data

Object recognition is the process to detect known objects and estimate their poses in a perceived scene. The ability to robustly perform object recognition is especially relevant in the area of service robotics. Consider a robot that is supposed to perform pick-and-place tasks. It has to identify the object in question and provide an accurate pose estimate for the subsequent grasping task. In this chapter we present an approach that, given a point cloud model of an object, uses point correspondences found with NARFs to determine possible poses in the scene where the object could be present. We introduce a novel algorithm that makes this calculation of candidate object poses from a set of feature matches possible in a very efficient manner. Next, our system performs spatial verification tests that score these candidate poses. We present a novel method that makes a robust evaluation like this possible, by analyzing how well the actual sensor measurements fit the assumption that the object is present at a certain position. In practical experiments we show that our method is able to recognize objects reliably, even in substantially cluttered scenes. In addition we demonstrate that our system can use a partial view of an object as a model, if a complete point cloud model is not available. Since all evaluated object poses rely on the feature matching process, the experiments presented in this chapter also extend the insights we got about the NARFs in Chapter 3, proving their reliability and enhancing the understanding of their properties.

* * * *

The ability of a robot to perform high level tasks like tidying up a room, fetching something for the user, or assembling a product from an unorganized set of parts requires many different non-trivial abilities. One of those is the capability to find objects that are relevant for the current task. For example, to execute commands like “please bring a chair” or “please wait at the table”, the robot needs to identify the existence and the 6DOF pose of the corresponding object in the scene.

In computer vision, object recognition has been a popular research topic for a long time. It was, e.g., the first application of the popular SIFT algorithm [82]. Compared to the approaches from computer vision, we focus on the recognition of objects in 3D range data, based on object models in the form of 3D point clouds. The high quality spatial information in 3D scans is advantageous for determining highly accurate object poses. Additionally, 3D sensors often have a wider field of view than cameras, enabling us to find objects in a larger area without an active search. Yet, 3D structures are typically much less distinctive than textures in vision. Object recognition in 3D scans must therefore be especially robust regarding high numbers of false

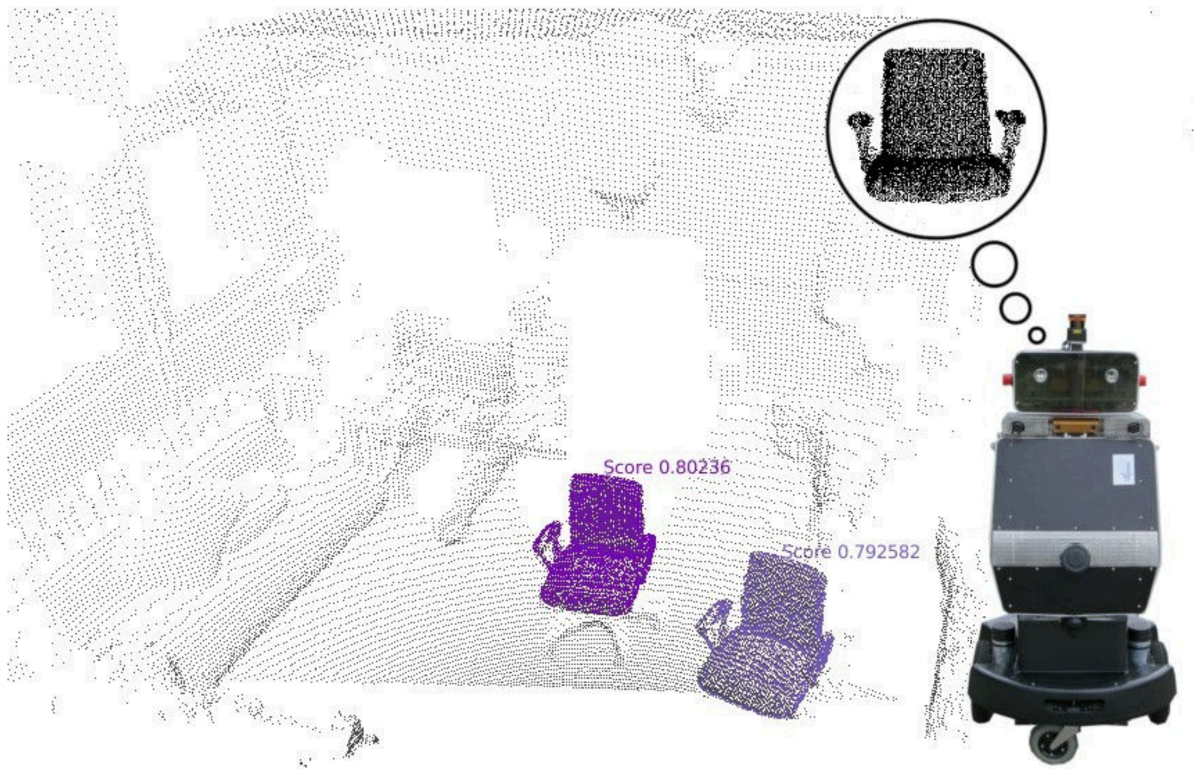


Figure 4.1: Motivational image for the object recognition system, showing a robot searching for the model of a chair in a point cloud of an office scene. Our system found two of these chairs with a certainty given by the score printed above them, whereas the score is between 0 (uncertain) and 1 (certain).

feature matches. For this purpose we will present a new algorithm to calculate potential object poses from a set of feature matches and will introduce two novel methods for the evaluation of such candidate poses based on spatial verification. This means we evaluate how well the actual data fits the predicted measurements assuming a correct pose estimate.

The calculation of the pose estimates and the evaluation step are mostly independent. In principle, one could sample from the 6DOF search space for every potential object and then verify those candidates. Since this is intractable we use the point features we presented in Chapter 3. We compare NARFs extracted from the model with NARFs extracted from the scene and calculate candidate object poses from the matches. Figure 4.1 shows an example of a scene we already used in Chapter 3. Here the two chairs in the foreground were correctly identified by our method.

The remainder of this chapter is organized as follows: we will first discuss the structure of our object database in Section 4.1. After this, we will explain how feature matches are determined between the object database and the current 3D scan in Section 4.2. Following this, we will discuss, how possible object poses can be calculated from sets of matching features in Section 4.3. In Section 4.4 we will explain how we evaluate these potential object poses to reject false positives. Section 4.5 briefly explains, how the approach handles multiple objects in the database. After this we present our experimental evaluation in Section 4.6. We will finish this chapter with a discussion about related work in Section 4.7 and a conclusion in Section 4.8.

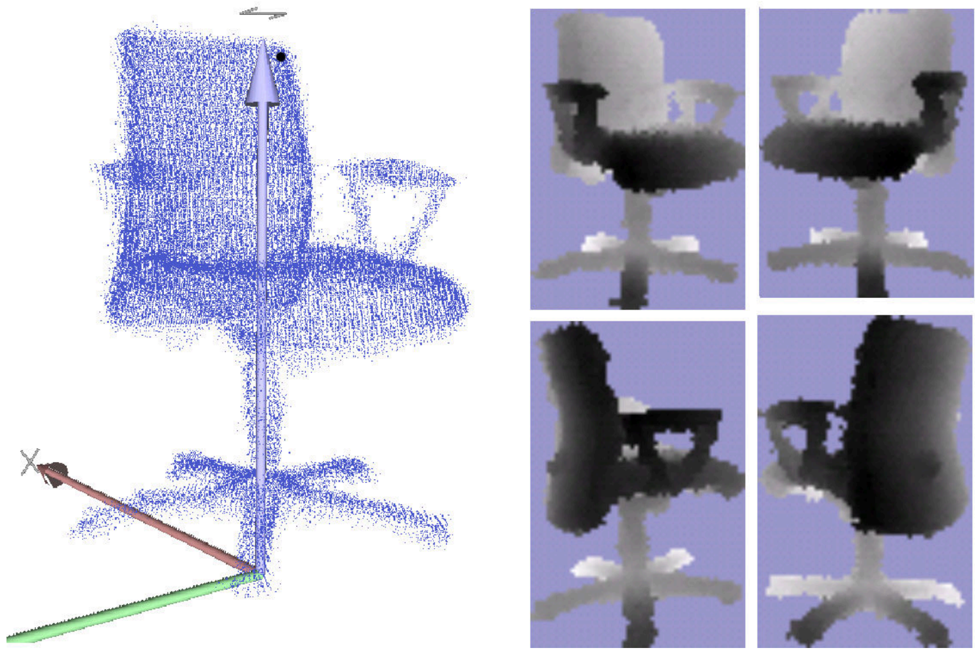


Figure 4.2: Point cloud model of a chair and four range images sampled from different perspectives around the model.

4.1 The Object Database

Our object database consists of a set of object models. For each of these models our system expects either a complete point cloud or a range scan from a single perspective. In the latter case the original sensor position must also be known and a correct detection is only possible if the object is seen from the same side. For a single view, we calculate a range image (see Section 2.3.2) from the perspective defined by the given viewpoint. For the complete point cloud model on the other hand, we sample different view points around the object and calculate range images for these perspectives. Figure 4.2 shows an example for this procedure. For each of the sampled views we then extract NARFs and store them in the model structure.

4.2 Feature Matching

The scene can be provided to our system either directly as a range image, or in form of a point cloud with a given view point. The latter is then used to compute a range image (see Section 2.3.2). From the scene range image we then extract NARFs (see Chapter 3) and compare them to the features extracted from the model views.

Let $\mathcal{N}^{\mathcal{S}} = \{\mathbf{d}_1^{\mathcal{S}}, \mathbf{d}_2^{\mathcal{S}}, \ldots, \mathbf{d}_{|\mathcal{N}^{\mathcal{S}}|}^{\mathcal{S}}\}$ be the set of NARF descriptors for the current scene and let $\mathcal{N}^{\mathcal{M}} = \{\mathbf{d}_1^{\mathcal{M}}, \mathbf{d}_2^{\mathcal{M}}, \ldots, \mathbf{d}_{|\mathcal{N}^{\mathcal{M}}|}^{\mathcal{M}}\}$ be the set of NARF descriptors for the model. For each descriptor $\mathbf{d}^{\mathcal{S}} \in \mathcal{N}^{\mathcal{S}}$ in the current scene and each descriptor $\mathbf{d}^{\mathcal{M}} \in \mathcal{N}^{\mathcal{M}}$ in the model, we compute the descriptor distance $\delta(\mathbf{d}^{\mathcal{S}}, \mathbf{d}^{\mathcal{M}})$ according to Equation 3.13. Based on this, we determine the set of potential correspondences $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{|\mathcal{C}|}\}$, which are defined as all feature index pairs between the scene and the model where the descriptor distance is below a given threshold $\delta_{\max}$:

$$\mathcal{C} = \{\mathbf{c} \quad | \quad \mathbf{c} = (i,j)^T \text{ with } \delta(\mathbf{d}_i^{\mathcal{S}}, \mathbf{d}_j^{\mathcal{M}}) \leq \delta_{\max} \ \wedge \ \mathbf{d}_i^{\mathcal{S}} \in \mathcal{N}^{\mathcal{S}} \ \wedge \ \mathbf{d}_j^{\mathcal{M}} \in \mathcal{N}^{\mathcal{M}}\} \tag{4.1}$$

To efficiently perform this comparison process, we store all feature description vectors in a *k*d-tree. To find the neighbors in the feature space, we apply the best-bins-first technique proposed by Lowe [82] to handle the high dimensionality of the vectors.

In the following we will use the feature correspondences $\mathcal{C}$ to calculate candidate transformations.

4.3 Candidate Poses from Feature Matches

Each NARF encodes its own local coordinate frame. Therefore, a single feature correspondence is enough to determine the potential 6DOF pose of the object in the scene as the difference between the feature coordinate frames. Another possibility is to use several correspondences to calculate a candidate pose. If we use only the point positions and the normals, then two feature matches are enough. If we use only the point positions alone, then we need to know at least three correct feature correspondences [60]. While using an increasing number of correspondences reduces the influence of noise on the estimate, there are also significant disadvantages: the search space size increases exponentially with the number of used correspondences and if the object model provides only a small number of keypoints or if it is partially occluded in the scene, there may only exist a small number of correct feature matches for a certain instance in the scene. To exploit the advantages of the different sized sets of correspondences, we add candidate poses from single correspondences, from correspondence pairs, and from correspondence triples. Yet we only consider a fixed number $n_{\mathcal{C}}$ of the best candidate poses for each of these possibilities. To decide which sets of the correspondences are used to determine candidate poses, we sort the set of correspondences $\mathcal{C}$ according to the respective descriptor distances to get the ordered set of correspondences $\mathcal{C}' = \{\mathbf{c}'_1, \mathbf{c}'_2, \dots, \mathbf{c}'_{|\mathcal{C}|}\}$. For single correspondences we just select the first $n_{\mathcal{C}}$ elements from $\mathcal{C}'$ to calculate candidate poses. For two or three correspondences we work through the sets in the order defined by the loop structures in Algorithm 1. In this way, we obtain the best correspondences (according to the descriptor distance) in an order that prevents the search from getting stuck if correspondences with low descriptor distances are actually false matches. We discard or accept the correspondence sets based on some simple heuristics. For example, we check if the metric distances between the three feature points in the scene are similar to the distances of the feature points in the model and we enforce a minimum distance between them. Additionally, we do not use triples of features that lie on a line. Correspondence sets that pass these quick tests are then used to generate candidate object poses, until $n_{\mathcal{C}}$ such poses were created for pairs and triples of correspondences respectively.

In the next step, the elements of the resulting set of candidate object poses $\mathcal{O}$ are evaluated regarding how well they fit the actual observation. Our method to perform this evaluation will be discussed in Section 4.4.

Please note that the idea behind this procedure, to select a small set of correspondences to calculate a transformation and then verify it in a second step, is borrowed from sample consensus algorithms like RANSAC [40]. Or in this case rather GOODSAC [87], where the samples are not selected randomly but according to a quality measure (the descriptor distance in our case). Yet, we do not actually use a sample consensus method since or spatial verification method (see Section 4.4) does not rely on the original feature matches to ensure that we can also find an object with even only one matching feature.

Algorithm 1 Nested loops that determine the order in which pairs or triples of correspondences are selected to find candidate object poses.

Selection scheme for pairs of correspondences:

```
FOR ($j = 2$; $j \leq |C|$; $j = j + 1$)
  FOR ($i = 1$; $i < j$; $i = i + 1$)
    $pair = (c_i, c_j)$
    $\ldots$
```

This generates the sequence (c'_1, c'_2), (c'_1, c'_3), (c'_2, c'_3), (c'_1, c'_4), (c'_2, c'_4), (c'_3, c'_4), $\ldots$

Selection scheme for triples of correspondences:

```
FOR ($k = 3$; $k \leq |C|$; $k = k + 1$)
  FOR ($j = 2$; $j < k$; $j = j + 1$)
    FOR ($i = 1$; $i < j$; $i = i + 1$)
      $triple = (c_i, c_j, c_k)$
      $\ldots$
```

This generates the sequence (c'_1, c'_2, c'_3), (c'_1, c'_2, c'_4), (c'_1, c'_3, c'_4), (c'_2, c'_3, c'_4), (c'_1, c'_2, c'_5), $\ldots$

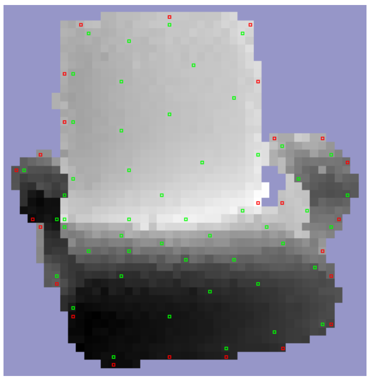


Figure 4.3: This image shows one of the sampled views of a point cloud model of a chair with the validation points marked in color. The green points are on the regular surface of the chair, whereas the red points are on the boundary.

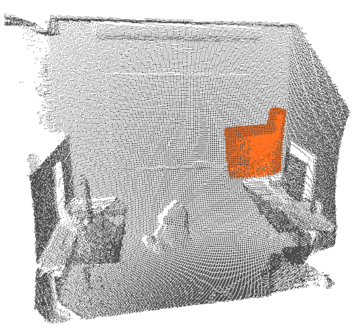
4.4 Spatial Verification

Our spatial verification procedure is divided in two independent steps. At first a fast approximate method to reduce the number of potential object poses in a runtime efficient way and second a slower more accurate method to further reduce the number of false positives.

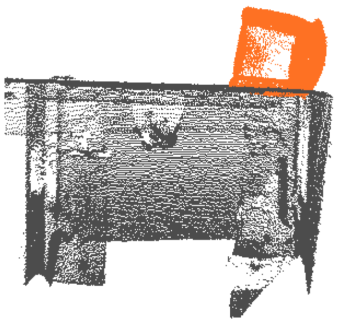
4.4.1 Validation Points

The first method is based on a restricted number of surface points that are compared between scene and model regarding the candidate transformation in question. We will call these points *validation points*. The procedure works as follows:

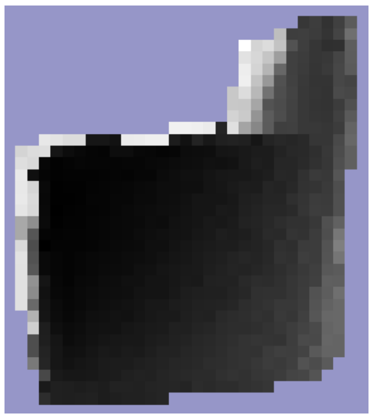
As already done for the feature extraction procedure in Section 4.1, we sample different views of the object to simulate different perspectives from which the object could have been seen. For each of these views we select a fixed number (50 in our implementation) of uniformly



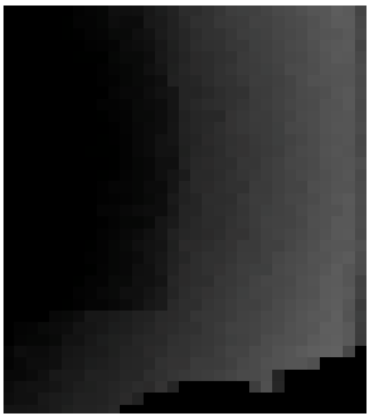
(a)



(b)



(c)



(d)

Figure 4.4: Example why considering boundaries is important **(a)** and **(b)** show different perspectives of an example scene with a falsely detected armchair model in a wall. **(c)** shows how the perspective of the object should look like and **(d)** shows what the part in the actual scene looks like. Considering only the range measurements on the surface of the object would lead to a high score, since the flat part fits well onto the wall. An analysis of the outer shape on the other hand would reveal this as a false positive.

distributed 3D points from the surface of the object and a fixed number (25 in our implementation) of uniformly distributed 3D points from the boundary of the object. The boundary is defined as those pixels in the range image of the model that have at least one unoccupied pixel as a neighbor. We call these points the validation points, whereas we distinguish between the surface validation points $\mathcal{V}_{\text{Surface}}$ and the boundary validation points $\mathcal{V}_{\text{Boundary}}$. These points need to be extracted only once during the model creation. This process therefore does not influence the runtime of the object detection process. See Figure 4.3 for an example view of a model of a chair with marked positions of the extracted validation points. We distinguish between these two types of points to increae the robustness of the evaluation. While it is of course important that the surface structure matches, the outer shape of the object is also a strong feature and should therefore be considered in the evaluation. If we only consider the surface, a flat object might be matched into a wall, where the surface fits perfectly, but the outer shape is not recognizable anymore. Figure 4.4 shows an example for this situation.

Please note that the set of simulated views does not have to be identical with the one used for feature extraction. This is because the size of the set for the feature extraction influences the runtime of the object search since more features have to be matched. Therefore we try to keep this set small, meaning we sample in larger angular steps. We use $36°$ in our implementation, meaning we have a maximum error of $18°$ between the sampled view and the actual observation. It is up to the feature descriptor, to have enough view point invariance to enable matching under these conditions. We will analyze this property of the NARF descriptor in Section 4.6.1. For the validation points on the other hand, the number of views does not influence the runtime of the matching procedure, which is why we use a much denser sampling of $10°$ here.

Let $T \in \mathcal{O}$ be the current candidate transformation we want to evaluate. We consider the validation points $\mathcal{V} = \{\mathbf{p}_1^{\mathcal{V}}, \ldots, \mathbf{p}_{|\mathcal{V}|}^{\mathcal{V}}\}$, which are sampled from the range image whose viewpoint is most similar to the one defined by T. Please note that we currently do not distinguish between $\mathcal{V}_{\text{Surface}}$ and $\mathcal{V}_{\text{Boundary}}$, since they are mostly treated the same. We will indicate the differences below. We map the selected validation points into the scene using T and receive the set of transformed validation points $\mathcal{V}' = \{\mathbf{p}_1^{\mathcal{V}'}, \ldots, \mathbf{p}_{|\mathcal{V}|}^{\mathcal{V}'}\}$. We then project each $\mathbf{p}^{\mathcal{V}'} \in \mathcal{V}'$ into the scene range image and consider a local pixel neighborhood around it (with a radius of two pixels in our implementation), which contains the points $\mathcal{S}_{\mathcal{V}'} = \{\mathbf{p}_1^{\mathcal{S}_{\mathcal{V}'}}, \ldots, \mathbf{p}_{|\mathcal{S}_{\mathcal{V}'}|}^{\mathcal{S}_{\mathcal{V}'}}\}$.

Here appears the only difference between the treatment of the surface validation points $\mathcal{V}_{\text{Surface}}$ and the boundary validation points $\mathcal{V}_{\text{Boundary}}$. For the surface validation points, $\mathcal{S}_{\mathcal{V}'}$ really contains all points from the local 2D neighborhood in the range image. For the boundary validation points, we only consider those points from the local neighborhood for $\mathcal{S}_{\mathcal{V}'}$ that are also boundary points. We use the method introduced in Section 3.1.2 to detect these boundary points. Please note that the boundary classification was already performed during the NARF extraction phase. Therefore its usage in the spatial evaluation does not influence the runtime.

If the candidate transformation is correct, then the range values of the original points in the range image of the scene and the range values of the transformed validation points should be the same. Let $r_{\mathcal{S}}(\mathbf{p})$ be the range value of a point $\mathbf{p}$ in the range image of the scene. Based on this we calculate the following score $s_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'})$ and weight $w_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'})$ for a single transformed validation point $\mathbf{p}^{\mathcal{V}'} \in \mathcal{V}'$.

$$s_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'}) = \max_{\mathbf{p}^{\mathcal{S}_{\mathcal{V}'}} \in \mathcal{S}_{\mathcal{V}'}} \begin{cases} |r(\mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}) - r(\mathbf{p}^{\mathcal{V}'})| < \varepsilon_r & : \; s'_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'}, \mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}) \\ \text{else} & : \; 0 \end{cases}, \tag{4.2}$$

$$\text{whereas} \quad s'_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'}, \mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}) = \left(1 - \frac{1}{4}\frac{d_{\text{2D}}(\mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}, \mathbf{p}^{\mathcal{V}'})}{d_{\text{2D}}^{\max}}\right) \frac{|r(\mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}) - r(\mathbf{p}^{\mathcal{V}'})|}{\varepsilon_r} \tag{4.3}$$

$$w_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'}) = \begin{cases} r(\mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}) - r(\mathbf{p}^{\mathcal{V}'}) > \varepsilon_r \;\; \forall \;\; \mathbf{p}^{\mathcal{S}_{\mathcal{V}'}} \in \mathcal{S}_{\mathcal{V}'} & : \; w_{\text{seeThrough}} \\ \text{else} & : \; 1 \end{cases} \tag{4.4}$$

Here, ε_r is the maximum error between the range values, below which the validation point can still be considered to be at the right place. If the range difference between the two points is within the error bound, i.e., $|r(\mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}) - r(\mathbf{p}^{\mathcal{V}'})| < \varepsilon_r$, then the point pair receives a positive score $s'_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'}, \mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}) \in (0, 1]$, which is the relative range error, weighted by the 2D distance between the two pixel positions in the range image $d_{\text{2D}}(\mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}, \mathbf{p}^{\mathcal{V}'})$. If the pixel position is identical, d_{2D} is 0 and therefore the left factor in Equation 4.3 is 1. If d_{2D} reaches the maximum allowed 2D distance $d_{\text{2D}}^{\max}$ ($2\sqrt{2}$ in our implementation), then the left factor in Equation 4.3 is 0.75. If the difference in range between the points is above ε_r, the score for the validation point $s_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'})$ is 0. Yet, there are differences for the weight $w_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'})$ of the validation point. Normally, this weight is 1. But if all neighboring points in the scene have a substantially higher range, i.e., $r(\mathbf{p}^{\mathcal{S}_{\mathcal{V}'}}) - r(\mathbf{p}^{\mathcal{V}'}) > \varepsilon_r \; \forall \; \mathbf{p}^{\mathcal{S}_{\mathcal{V}'}} \in \mathcal{S}_{\mathcal{V}'}$, then we increase $w_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'})$ to $w_{\text{seeThrough}}$ (5 in our implementation) since this situation means the sensor actually measured a point behind the object, effectively looking through it, which is a strong indicator for a wrong pose estimate.

The score for the complete set of validation points has a value between 0.0 and 1.0 and is defined as

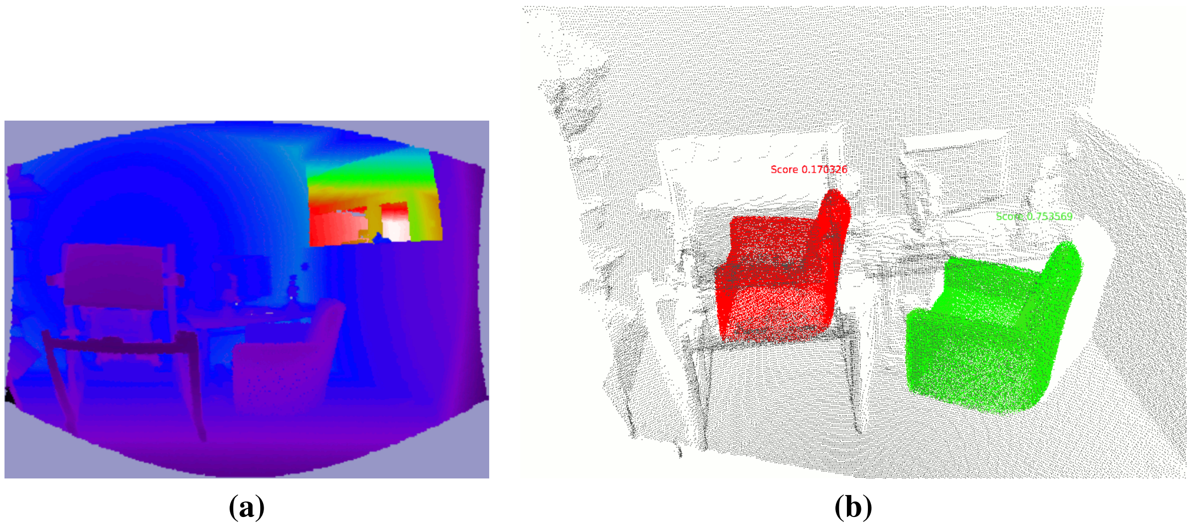


Figure 4.5: (**a**): Range image of an example scene. (**b**): The scene as a point cloud with two found armchairs. The red one is a false positive, whereas the green one is a true positive.

$$\tilde{s}_{\mathcal{V}}(\mathcal{V}') = \frac{\sum\limits_{\mathbf{p}^{\mathcal{V}'} \in \mathcal{V}'} w_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'})\ s_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'})}{\sum\limits_{\mathbf{p}^{\mathcal{V}'} \in \mathcal{V}'} w_{\mathcal{V}}(\mathbf{p}^{\mathcal{V}'})}, \tag{4.5}$$

whereas we combine the scores for the surface validation points and the boundary validation points to

$$\hat{s}_{\mathcal{V}}(\mathcal{V}') = \tilde{s}_{\mathcal{V}}(\mathcal{V}'_{\text{Surface}})\ \tilde{s}_{\mathcal{V}}(\mathcal{V}'_{\text{Boundary}}). \tag{4.6}$$

This score is already a good indicator for the quality of an object candidate and can be used as an initial filter. We can reject poses with a low validation point score. In addition, we typically receive multiple instances of an object at very similar locations. This is caused by different sets of correspondences, which all originate from the same object in the scene. Therefore, we prune the solutions by keeping only the ones with the highest score for all objects that are found in a local area. The validation point method has the advantage of being very fast for a reasonable number of validation points. Yet since we only sampled possible perspectives, this is an approximate method. We will perform an additional, more expensive verification for the remaining object poses.

4.4.2 View Simulation

In the last step we presented a method to score candidate poses based on a set of precomputed points. In this section we present a method to evaluate a candidate position based on a simulated view of the object. For this purpose we calculate a low-resolution range image of the model given the virtual camera position described by the candidate pose. If the match is valid, the result should be similar to the corresponding part of the range image of the scene.

Figure 4.5 shows an example scene with two detected instances of a model of an armchair, whereas one of these detections is a false positive and one is a true positive. Figure 4.6(a) shows the simulated view of the object for the true positive and Figure 4.6(b) shows the corresponding part of the scene. One can see, that these two images look very similar. Figure 4.6(c) shows a combination of the other two images, in form of a range image of the scene with the object point cloud projected into it. There are only very minor visual differences between Figure 4.6(b) and Figure 4.6(c). Figure 4.7(a),(b), and (c) show the corresponding images for the false positive, where the armchair was wrongly matched with a part of a table. Visually it is obvious that this is not a correct match.

We now calculate a score based on these images. This process is mostly equivalent to the validation point scoring described in Section 4.4.1. The main difference is that we use all valid pixels from the object view for the comparison instead of the validation points. In addition, we introduce a new error term based on the gradients of the range image pixels (see Figure 4.6(d)-(i) and Figure 4.7(d)-(i)). For every point we now not only evaluate if the measured range value fits the expected range value (see Equation 4.3), but we also compare, in an equivalent fashion, if the gradients in x and y direction between the original perception and the simulated version are similar. In our example this means we compare Figure 4.6(e) with Figure 4.6(f) and Figure 4.6(h) with Figure 4.6(i). Equivalently for Figure 4.7. This enforces an even stricter likeness of the surface.

Figure 4.6(j) and Figure 4.7(j) show the individual scores each pixel received, whereas the true positive received an overall score of 0.75 and the false positive received an overall score of 0.17.

This step is computationally more expensive than the validation point scoring because we have to compute new range images from the model and perform several costly operations on them. However, we found that this leads to more reliable scores. We reduce the influence of the higher computational needs by first filtering out results with low validation point scores. The view simulation is then typically only performed on a small number of potential object poses.

4.5 Searching for Multiple Objects

In the previous section, we described how to detect one type of object in a scene. The approach can be straightforwardly extended to detect multiple models. In this case, we store all the features of all the models in the *k*d-tree and save from which view of of a certain model model it was extracted. Once the potential correspondences are found, we determine the possible models from the associated range images. We finally execute the previously described procedure once for every candidate model. In the best case this makes the object search complexity logarithmic in the number of object models. The average and worst case however are linear.

4.6 Experimental Evaluation

We will now present experiments to evaluate our approach. In these experiments we will compare poses our system calculated with previously determined ground truth poses. In this evaluation, we consider an object pose as correct, if no point on the model is further than one tenth of the diameter of the model away from its true position. The runtime results provided in the experiments were achieved on an Intel I7 quad core machine.

4.6.1 Single View Models

In a first experiment we evaluate a situation, where no complete point cloud model is available. As discussed in Section 4.1, our system can also use a single view of an object as the model. To test how well the object recognition works in this case, we captured 80 3D scans of an office

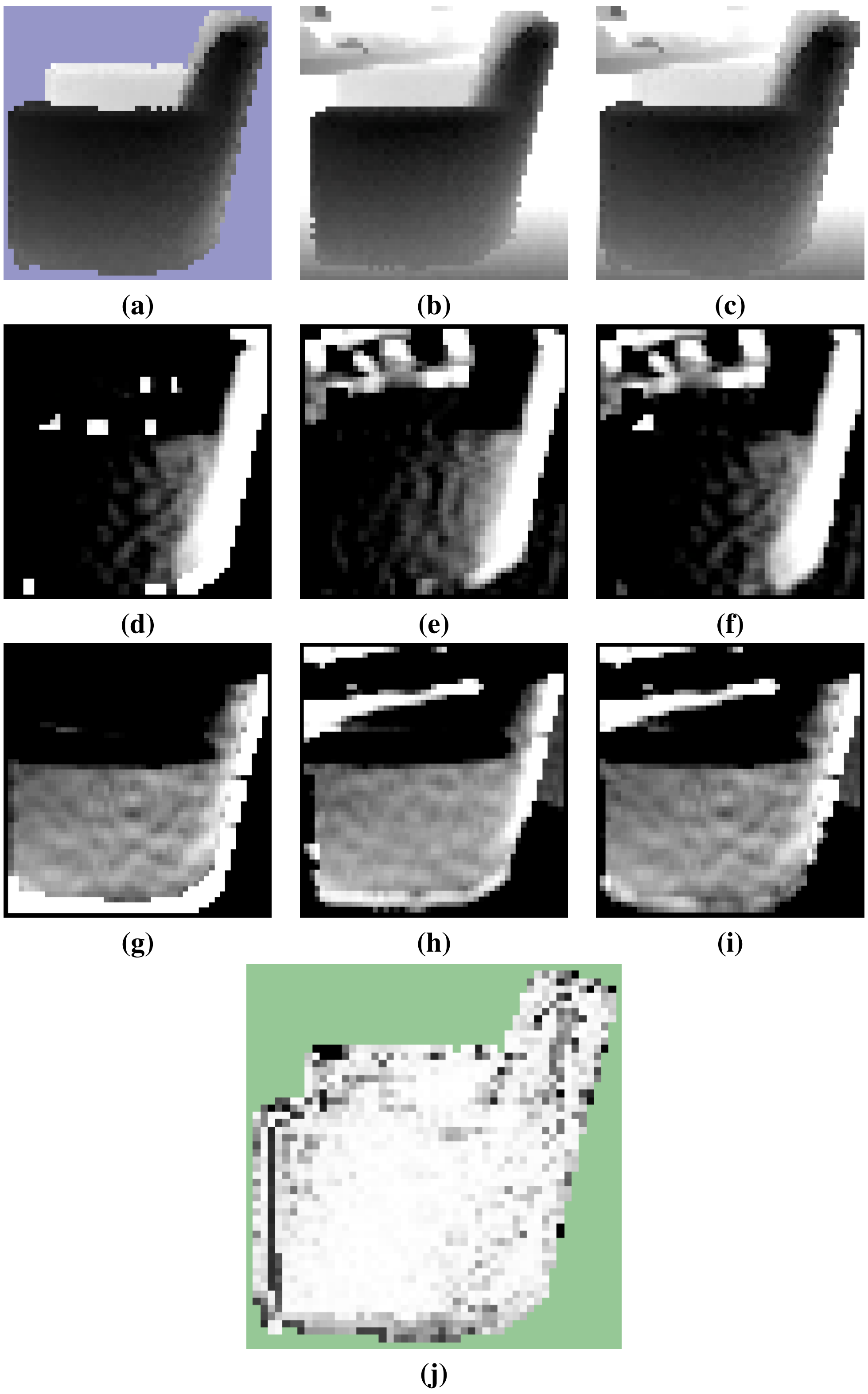


Figure 4.6: View simulation example for the true positive shown in Figure 4.5. **(a)**: The simulated view of the object. **(b)**: The corresponding part of the scene. **(c)**: The part of the scene with the object simulated inside. **(d)**: The gradients of (a) in x-direction. **(e)**: The gradients of (b) in x-direction. **(f)**: The gradients of (c) in x-direction. **(g)**: The gradients of (a) in y-direction. **(h)**: The gradients of (b) in y-direction. **(i)**: The gradients of (c) in y-direction. **(j)**: The scores for every pixel from black (score 0) to white (score 1). Irrelevant pixels are marked green.

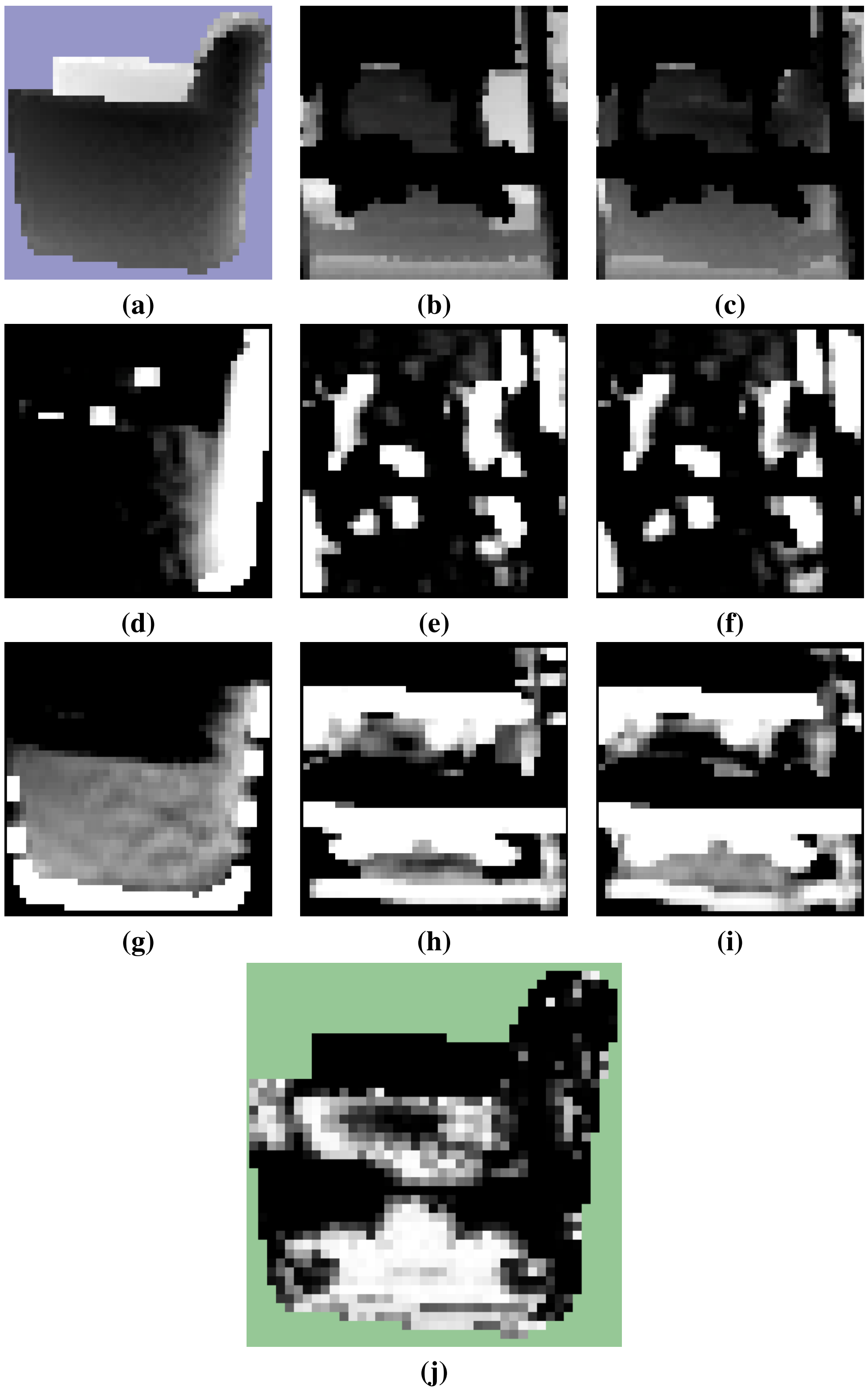


Figure 4.7: View simulation example for the false positive shown in Figure 4.5. **(a)**: The simulated view of the object. **(b)**: The corresponding part of the scene. **(c)**: The part of the scene with the object simulated inside. **(d)**: The gradients of (a) in x-direction. **(e)**: The gradients of (b) in x-direction. **(f)**: The gradients of (c) in x-direction. **(g)**: The gradients of (a) in y-direction. **(h)**: The gradients of (b) in y-direction. **(i)**: The gradients of (c) in y-direction. **(j)**: The scores for every pixel from black (score 0) to white (score 1). Irrelevant pixels are marked green.

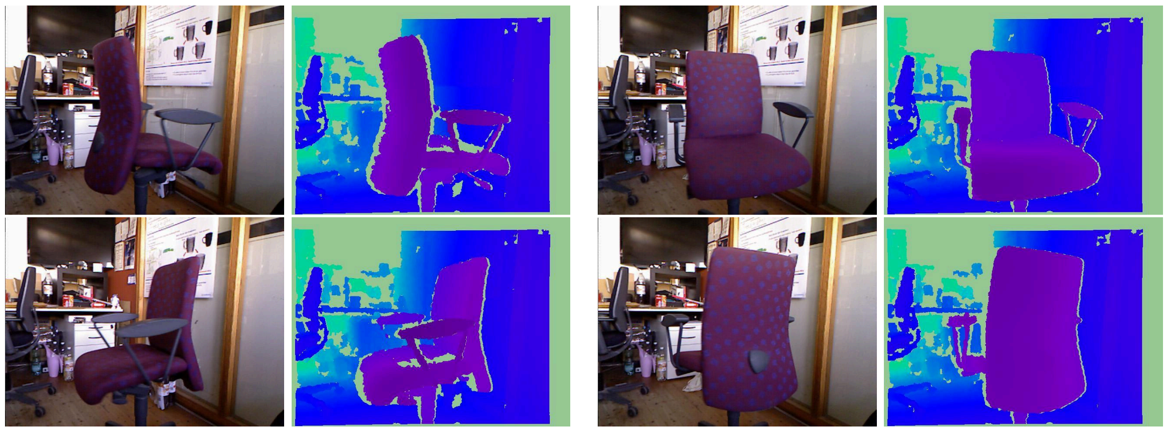


Figure 4.8: Example images from a dataset of a rotating chair, both as color images and range images.

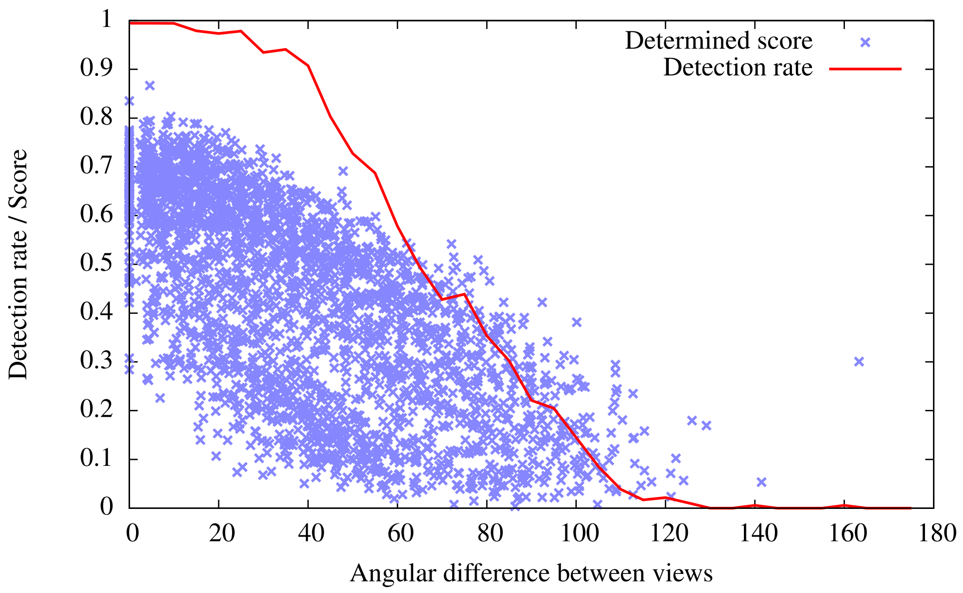


Figure 4.9: This plot shows experimental results regarding the detection of an object, in this case an office chair, for which we do not have a complete point cloud model, but a single view of the object instead. We show the detection rate for different angular differences in the perspective and also the determined scores for the successfully detected object poses.

chair from equally distributed perspectives around it. We used a Microsoft Kinect for this purpose. See Figure 4.8 for example images. We manually labeled the true positions of the object in the scans to acquire close-to-ground-truth data. We then used each of the 80 perspectives as a single-view object model and tried to detect the chair in the entire collection of scenes. The result is summarized in Figure 4.9. The plot shows the rate with which the correct object position was found, depending on the angular difference between the perspectives. One can see that the detection rate is above 90% for differences below $40°$. For higher angular differences it drops quickly. The plot also shows the determined validation point scores (see Section 4.4.1) for the individual detection processes where a correct pose could be found. Typical validation point scores for small angular differences are between 0.3 and 0.8. The lower values typically occur for unfavorable perspectives of the chair, meaning side views where the backrest is visible at a steep angle. Please note that we did not perform the view simulation step (see Section 4.4.2) for this experiment since this process explicitly requires a complete model.

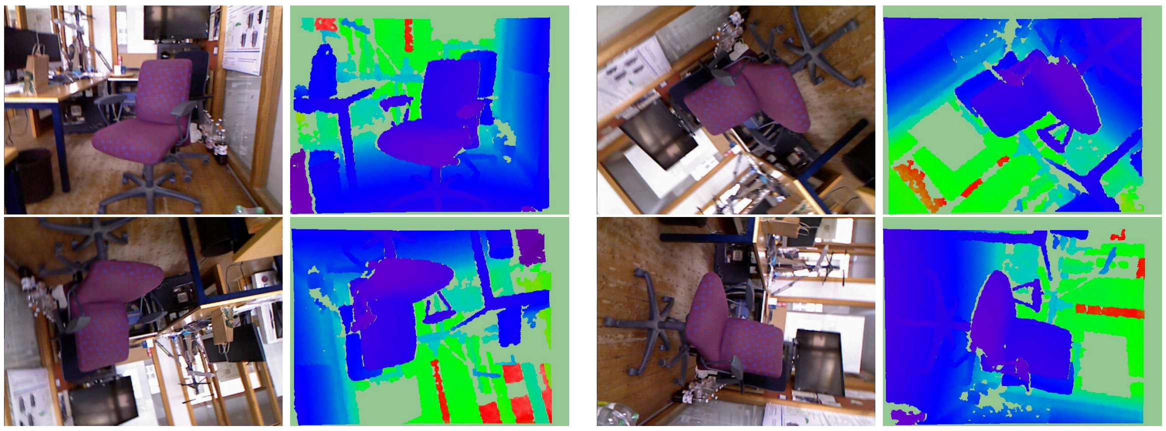


Figure 4.10: Example images from a dataset of a chair, captured with a rotating depth camera, both as color images and range images.

4.6.2 Rotational Invariance

In a second experiment we evaluated the option for rotational invariance for the NARF descriptors (see Section 3.3). The NARF descriptor calculation has the option to normalize the descriptor regarding the rotation around the normal, thereby making it rotationally invariant. If this option is not used, the descriptors will not match, e.g., for a top right rectangular corner and a top left rectangular corner. This can be advantageous in situations where objects have a typical rotation in which they occur. A chair, e.g, typically stands upright on its feet. When this is the case, reducing the search space simplifies the problem. To compare the two options against each other, we captured a dataset of the same chair as used in Section 4.6.1. This time we did not rotate the chair, but the camera 360° around its center axis. See Figure 4.10 for example images. We captured 235 3D scans in this situation, again using a Microsoft Kinect and again manually labeling the true positions of the chair in the scans. We then searched for the model of the chair with and without the option for rotational invariance. Figure 4.11 summarizes the result. The plots show the score of the correctly found object poses for angular differences between the model and the instance in the scene. A score of 0 means that the pose was not in the set of candidate poses. The NARF version with and without rotational invariance are shown in different colors. While the rotationally invariant version was able to find a pose for all angular differences, the version without rotational invariance could only reliably find poses for small angular differences (up to 25°). Yet, the spatial verification step for the rotationally invariant version took 300 ms on average per scene, whereas the version without rotational invariance took only 80 ms. This is caused by the increased size of the search space if the rotation around the normal is also a free parameter. The decision to use the NARFs with or without rotational invariance should therefore depend on the actual application.

4.6.3 Performance for Office Environments

In the next experiment, we tested the performance of our approach in an office environment. Specifically, we applied our method to the dataset already presented in Section 3.4.2, where we used it to evaluate the performance of our NARF keypoints and descriptors. This dataset includes seven point cloud models of very different kinds of objects. Five furniture sized objects (an armchair, a cart, an office chair, a Pioneer robot, and a teddy bear) and two tabletop objects (a cup and a stapler). See Figure 3.7 for photos of the used objects. In addition to the models, the dataset includes 10 scenes captured with a 3D laser range finder in a typical cluttered office

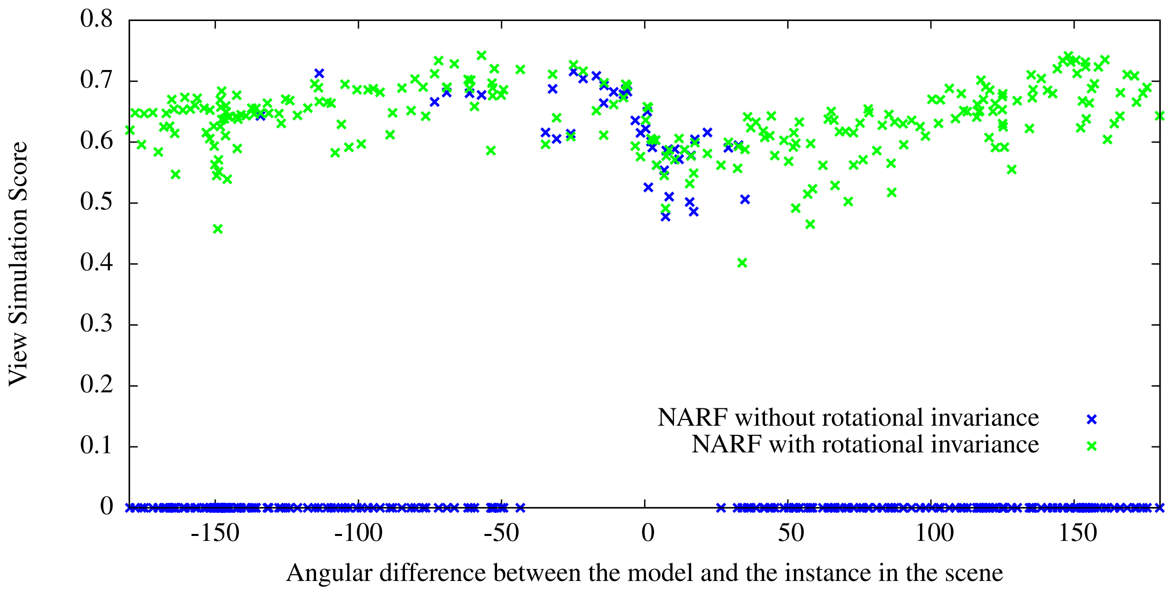


Figure 4.11: This plot shows experimental results regarding the detection of an object, in this case an office chair, with the sensor rotating around its center axis. The dots mark the scores the system determined for the correctly found object positions. A score of 0 means the object was not found. The two colors distinguish the NARF versions with and without rotational invariance. Wile the rotationally invariant version is able to find the correct pose for all rotations, the other version is only able to find poses with low angular differences between the model and the instance in the scene.

environment and 10 tabletop scenes captured with a stereo camera system. By artificially adding the models to the scenes we created 70 scenes with one known object in each. See Figure 3.9 for examples of these scenes. In Section 3.4.2 we already used this dataset in an object recognition context, yet we only analyzed the performance of the keypoints and descriptors, without the spatial verification steps that we introduced in this chapter in Section 4.4. We used our object recognition system to detect the five furniture sized objects in the 50 office scenes and the 2 tabletop objects in the 20 tabletop scenes. Since the true object positions are known, we are able to determine for each found object position if it is a true positive or a false positive. In addition, we can also determine situations, where the position of the object was correctly determined regarding the translation, but with a wrong orientation. Figure 4.12 shows two plots that summarize the outcome. We plot the validation point score (Figure 4.12(a)) and the view simulation score (Figure 4.12(b)) for every found object position, indicating which were true positives, true positions with wrong orientations, and false positives. It can be seen that both methods distinguish well between false positives (low scores) and true positives (high scores). As expected, the view simulation gives the better results, yet the runtime requirements for this method are much higher (see below). As already discussed in Section 3.4.2, the cup is a very symmetrical object and only the handle makes it possible to clearly determine its orientation. This property is visible in the plots, since there is an overlap in the score values of the cup between the true positives and the true positions with wrong orientations. These plots indicate that simple thresholding on the score values enables us to distinguish true object positions from wrong ones. This is even better visible in Figure 4.13, where we plot the recall rates for the different objects, depending on different acceptance thresholds for the view simulation score values. We also plot the number of false positives found for each acceptance threshold. Here we see that nearly all of the objects were found in the scenes and got scores above 0.6. Only the cart was missed once, meaning that the correct pose was not in the set of potential poses determined by the feature matches. Using an acceptance threshold of 0.65 would produce recall rates of 90% and above for most of the objects, while not returning any false positives. Please note that we did not consider the poses with correct 3D positions but wrong orientations in this plot.

The runtime requirements for this experiment were as follow. For the office scenes with furniture sized objects we used an angular resolution of 0.5° to create range images. The range image creation took 15 ms, the feature extraction 80 ms, and the feature matching 15 ms on average. Spatial verification using only validation points took 250 ms, using only view simulation 7,500 ms, and using a combination (first validation point scoring and then view simulation scoring for all results above 0.3) 350 ms on average. This confirms our decision to use this combination, since it combines the better scoring results of the view simulation with the faster runtime of the validation point scoring. For the tabletop scenes we used an angular resolution of 0.25°. The range image creation took 12 ms, the feature extraction 10 ms, the feature matching 1 ms, spatial verification with only validation points 60 ms, spatial verification with only view simulation 3,500 ms, and spatial verification using a combination 230 ms.

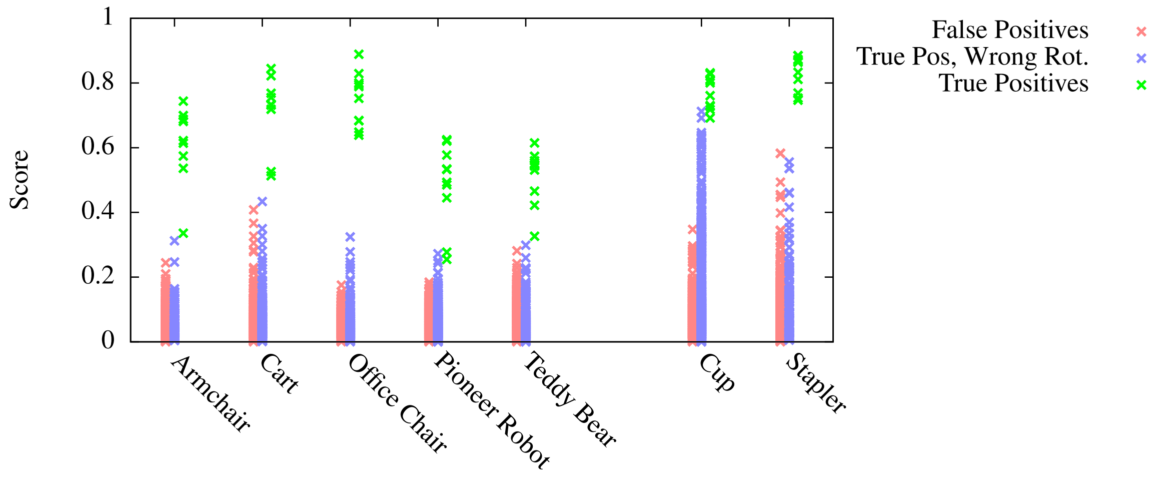
4.7 Related Work

Detecting objects based on point feature matches is an established concept in computer vision. E.g., the popular SIFT algorithm proposed by Lowe [82] was originally designed for an object recognition task. In this approach, SIFT correspondences between a query image and a database of reference images are combined into clusters if they agree on a possible object pose. The probability to be a true match is computed based on how well the matched features fit and on the number of probable false matches.

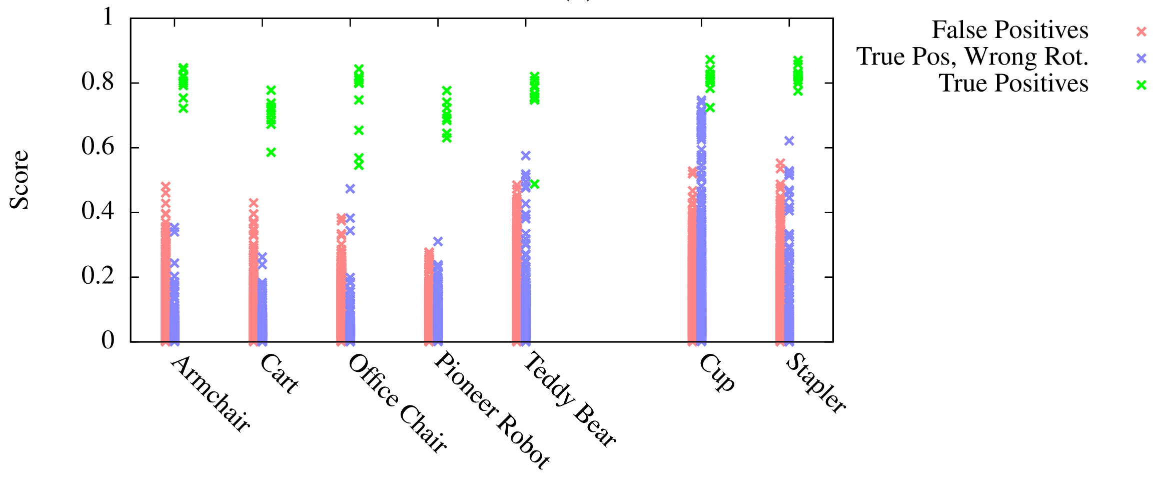
Our approach operates on 3D range data. While spatial reasoning is much more convenient in this kind of data, it is typically less descriptive than visual textures. Johnson *et al.* [65] originally proposed the popular *spin-images* in an object recognition context using 3D scans. The approach computes a spin-image for every point in the model and every point in the scene and then groups correspondences together to calculate plausible transformations, which are then verified based on the overlap. Triebel *et al.* [142] used spin-images as features for an associative Markov network. The parameters of this network are learned from a manually labeled training dataset of objects and are subsequently used to find object labels for each 3D point. This means the approach does not identify actual instances of an object but determines the most likely class to which individual points belong.

Stiene *et al.* [132] introduced an approach for object detection based on silhouettes extracted from range images. They use a feature type that is based on a fast Eigen-CSS method and a supervised learning algorithm. The approach is similar to ours in the sense that it works on range images and explicitly considers the outer shape of objects. But being restricted to the complete contour of the objects makes this approach less resistant to partial occlusion.

Blum *et al.* [12] described an object recognition approach relying on feature descriptors that are learned from RGB-D data in an unsupervised fashion, whereas they used SURF for the extraction of keypoints. The object recognition process itself is performed using a Support Vector Machine that was trained based on the feature descriptors appearing on a labeled dataset. With this, the approach is able to determine the probable existence of an object in a certain area of the input image. The authors then propose a method to determine the 6DoF position of the object based on feature correspondences and refinement with ICP. Yet the authors do not present a method for spatial verification of the found object poses.



(a)



(b)

Figure 4.12: Experimental results for object recognition in an office environment. The plots show the scores for all evaluated object poses for the individual objects. The marked points are color coded, distinguishing between false positives, true positives and correctly found 3D positions that have wrong orientations. **(a)** shows the results for the validation point scoring and **(b)** shows the results for the view simulation scoring. For both methods the highest scores are exclusively true positives, whereas the view simulation divides the classes better, as expected.

Rusu *et al.* [112] presented a 3D feature descriptor that encodes geometry and viewpoint information. The authors use it to simultaneously recognize objects and determine their pose. In this approach, a complete object view is described with a single feature descriptor. The system allows a very efficient retrieval of similar objects for a query from a database, but relies on a good initial segmentation of the object, which is not always possible in cluttered scenes. In addition, using a single descriptor for the whole object makes the system less robust regarding partially occluded objects.

4.8 Conclusions

In this chapter we presented an object recognition system that successfully employs the NARFs we introduced in Chapter 3. The point feature correspondences are used to calculate potential object poses, thereby reducing the size of the search space substantially compared to exhaustive

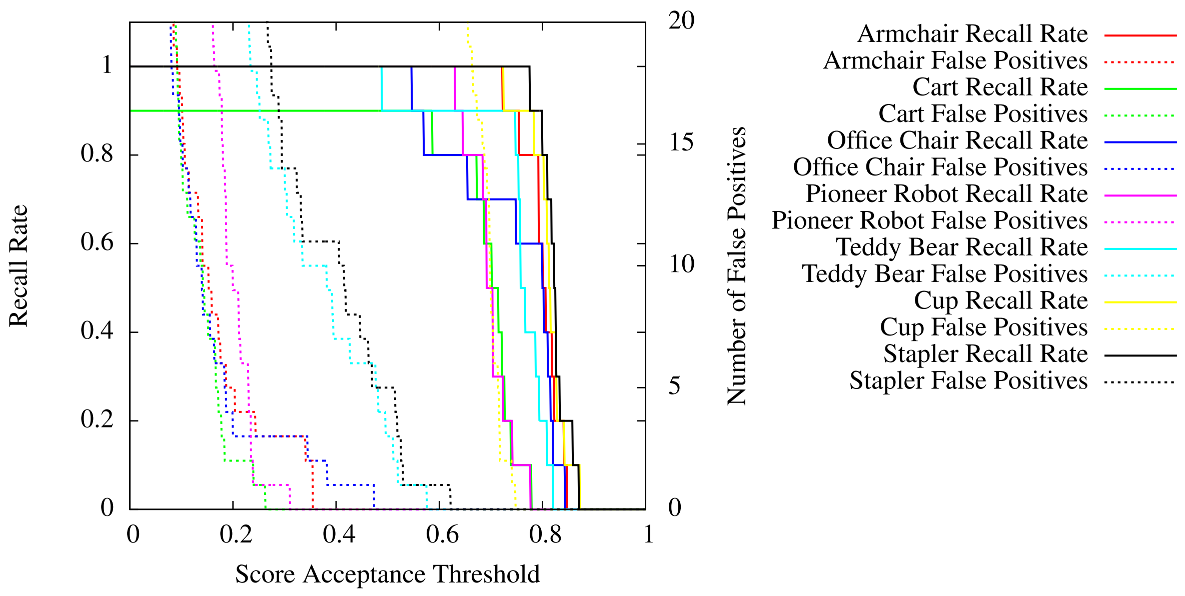


Figure 4.13: Experimental results for object recognition in an office environment. The plot shows the recall rates and the number of false positives for the individual objects, plotted against the acceptance threshold for the view simulation score. For acceptance thresholds slightly above 0.625, there are no false positives and recall rates of 80%-100%.

search. To calculate candidate object poses from the feature matches we introduced a novel algorithm for the selection of sets of correspondences that makes use of the descriptor distances. It returns clusters of correspondences that are most likely to belong to a true detection. The candidate object poses are evaluated based on two novel spatial verification methods. One very fast method based on a limited number of precomputed points, which is used to remove a large number of the false positives in advance. Second a computationally more demanding method, which simulates the expected view and compares it with the actual measurements. We presented experiments using real data from 3D laser range scanners and depth cameras. They show the capabilities of our system to recognize objects robustly, even in the presence of substantial clutter and for arbitrary 6DOF positions of the objects in the scenes. In addition, we demonstrated that even if no complete object model is available, a single view of the object can be used to detect it reliably, for viewpoint changes of up to 30°.

Chapter 5

Unsupervised Learning of 3D Object Models from Partial Views

The creation of 3D object models is highly relevant in robotics since such models are a prerequisite for basic tasks like object recognition. Often such models are created manually in a CAD program or by scanning objects in specific setups, e.g., on a turntable. This process is often tedious regarding the involved amount of human labor. This chapter presents an approach to build object model point clouds from a set of 3D range scans in an unsupervised fashion. The system searches for partial views potentially belonging to the same object and merges them into an accurate 3D representation of the object. We use methods already applied for object recognition tasks in the previous chapter to find overlapping areas and perform the alignment. The approach performs spectral clustering on a similarity matrix of partial object views to decide which of them should be merged into one object model.

*   *   *   *

In Chapter 4 we presented an approach for model-based object recognition. The models for this process were taken from public databases or created manually by capturing different views of an object and merging them. In this chapter we present an alternative to this model creation procedure, which learns object models in an unsupervised fashion. It applies methods from the object recognition system presented in Chapter 4 to find and align overlapping areas of partial views, belonging to the same object. These partial views in a sequence of 3D range scans do not necessarily have to belong to the same instance of an object and can be captured in different places at different times. Multiple instances of an object visible in a single scan are also utilizable. See Figure 5.1 for an example application where models from a wooden chair, a garbage bin, and an office chair are created from three, two, and one partial views visible in a single scan respectively.

Our approach first determines potential partial objects from the input point clouds. These are then iteratively merged into more complete point clouds of the objects until our system can not find enough similarity between the remaining models anymore.

The remainder of this chapter is organized as follows: In Section 5.1 we will discuss, how our system works, first as a rough overview in Section 5.1.1, then in more detail starting from Section 5.1.2. We present our experimental evaluation in Section 5.2. Afterwards we will discuss related work in Section 5.3, followed by our conclusion in Section 5.4.

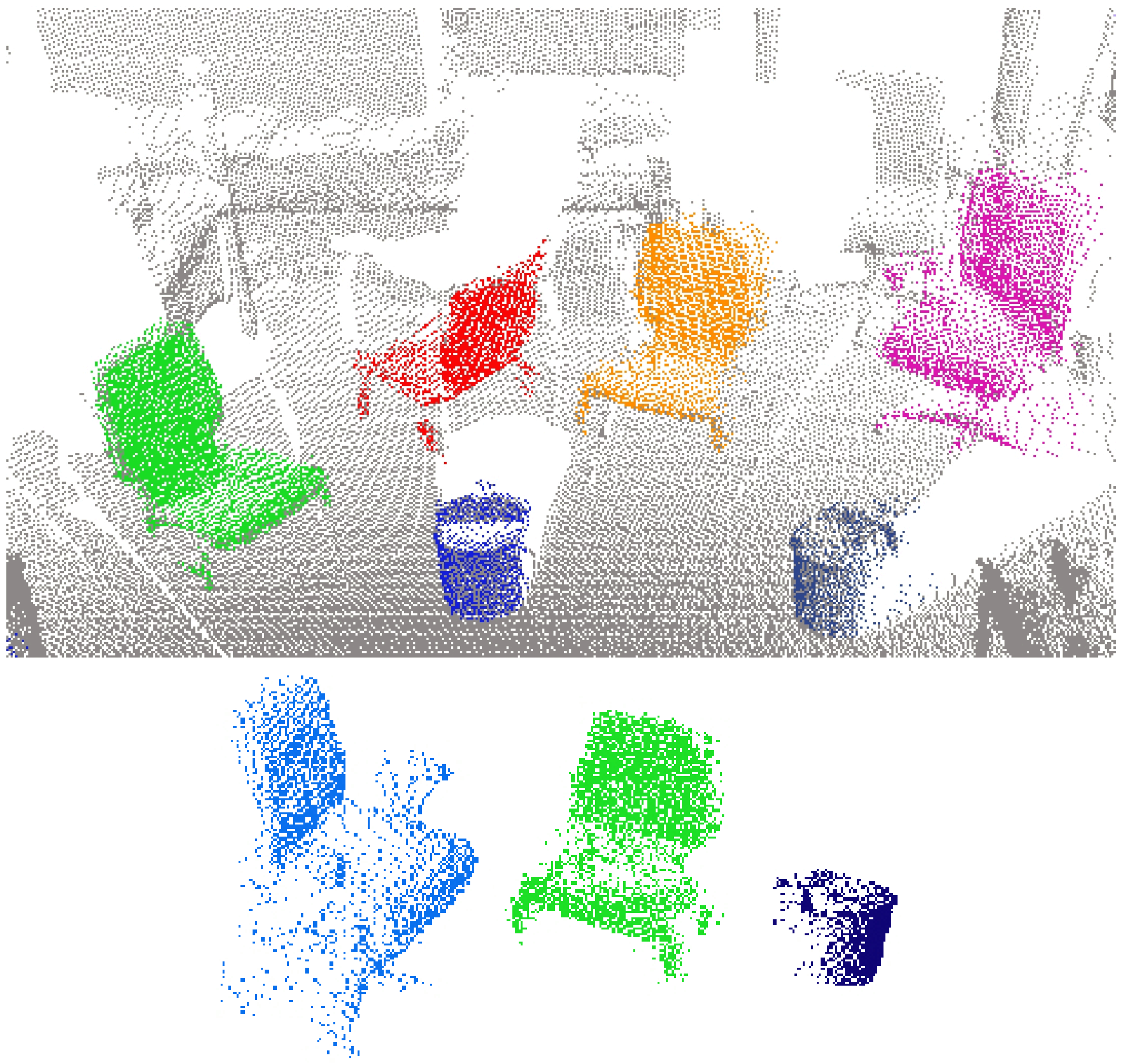


Figure 5.1: Motivational example for our model learning system. The top image shows a single 3D scan with multiple objects in the foreground: three instances of a wooden chair, two instances of a garbage bin, and one instance of an office chair. Our system merged the different views of the objects that were present multiple times and created more complete models. Those are shown in the bottom image. (Source: [104])

5.1 Unsupervised Object Model Learning

We will now describe our approach to learn 3D point cloud object models from partial views, as they are typically observed by 3D laser scanners or range cameras. We begin with an overview of the system and will then describe the individual parts in detail.

5.1.1 Overview

For a given sequence of 3D range scans our method performs the following steps to create models of the contained objects from them:

1. We first remove background elements like the floor, the walls, and the ceiling using a plane extraction method.

2. We then perform spatial clustering on the remaining points in the scans to find sets of connected points.

3. The clusters above a certain size are then used as an initial set of partial object models.

4. These partial models are then merged in an iterative fashion:

    - First we determine a crosswise similarity measure between all of the partial models, which we specifically designed to work with partial views. We also determine a relative transformation between all model pairs, which can be used to merge the point clouds into a common coordinate frame according to the best found fit.
    - Then we use spectral clustering on the resulting similarity matrix to decide which models should be merged into a new one, which then replaces the original partial models.
    - If at least two models were merged into a new one, we repeat step 4. Else we can stop the process since no two models were similar enough anymore.

We will now discuss the individual steps in more detail.

5.1.2 Background Extraction

The first step for our system is to remove parts of the environment from the 3D scans that are not relevant for our application. For this purpose we remove points belonging to large horizontal or vertical planes from the scans. These structures typically represent the floor, the ceiling, or walls, which all are not interesting for our goal to find partial object models. They could also interfere with the clustering step we will discuss in Section 5.1.3.

The plane extraction procedure we use employs a region growing approach, which iteratively adds points to a plane if it fits well to it and then updates the plane estimation accordingly, until no fitting point can be found anymore in a local neighborhood. In our experiments we used $0.25\,\mathrm{m}^2$ as the minimum size of a plane to be removed. If this value is to small, we might risk to remove parts actually belonging to an object we are interested in. See Figure 5.2(a) and (b) for an example of this procedure.

5.1.3 Spatial Clustering

After we removed the background from the 3D scans, we perform a spatial clustering step to identify individual objects in the scene. Like for the plane extraction, we do this using a region growing approach. We first select a random point from the scene and add it to the current cluster. Then we iteratively add points to this cluster that are spatially close to at least one point already in the cluster. This is done until no close by point can be found anymore. We can now add this new cluster to our set of clusters and remove the contained points from the 3D scan. All point clusters above a predefined size now form the initial set of partial objects, which we will try to merge in the next step.

Please note that the removal of the floor, which was done in the last step, is crucial for this procedure since most of the objects are typically connected by the floor, which would then lead to one big cluster, preventing us from identifying individual objects in the scene.

See Figure 5.2(c) for an exemplary outcome of this procedure.

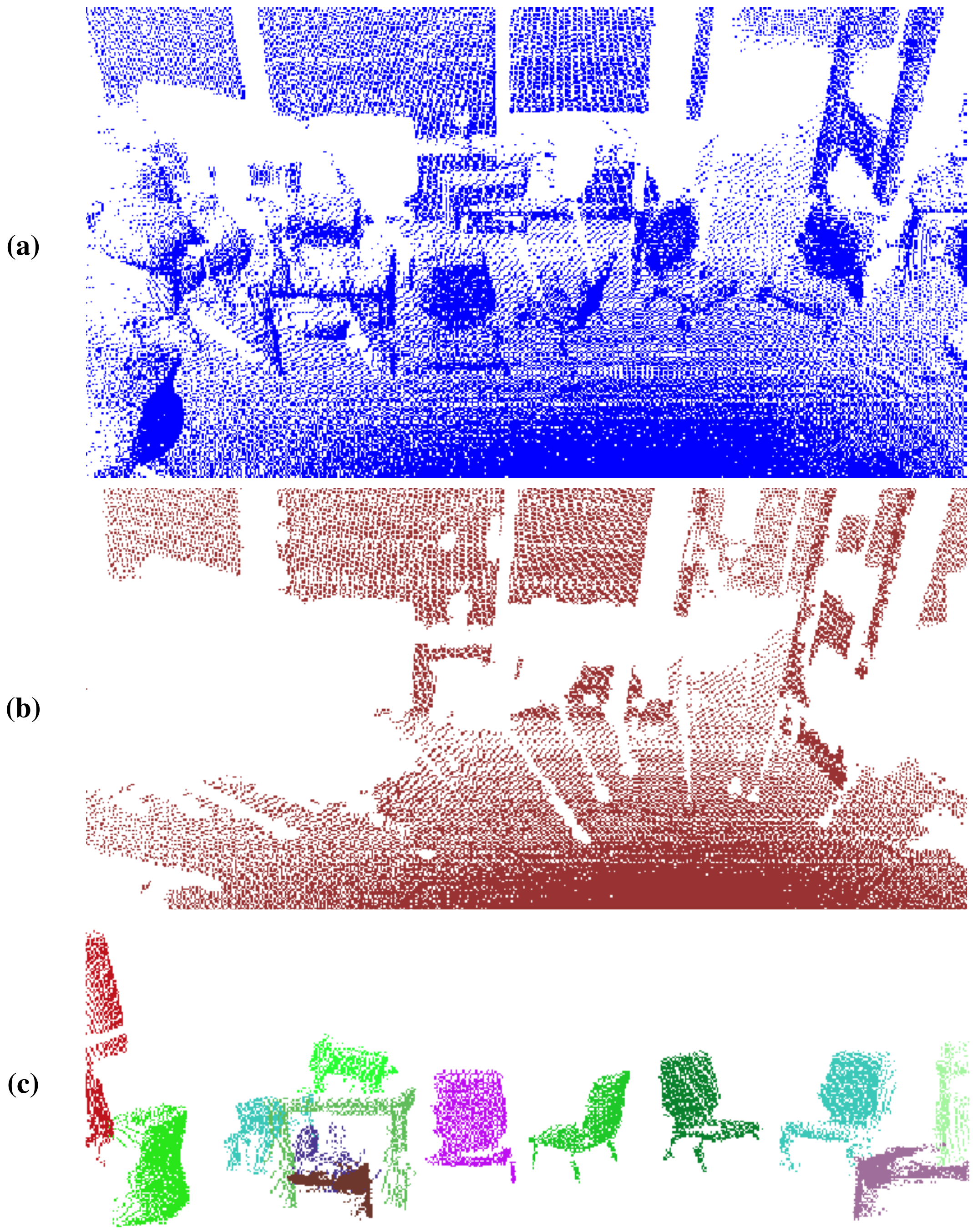


Figure 5.2: **(a)**: Example point cloud of an office. **(b)**: Found large horizontal and vertical planes. **(c)**: The structures remaining when the planes are removed. The different colors represent the found clusters. (Source: [104])

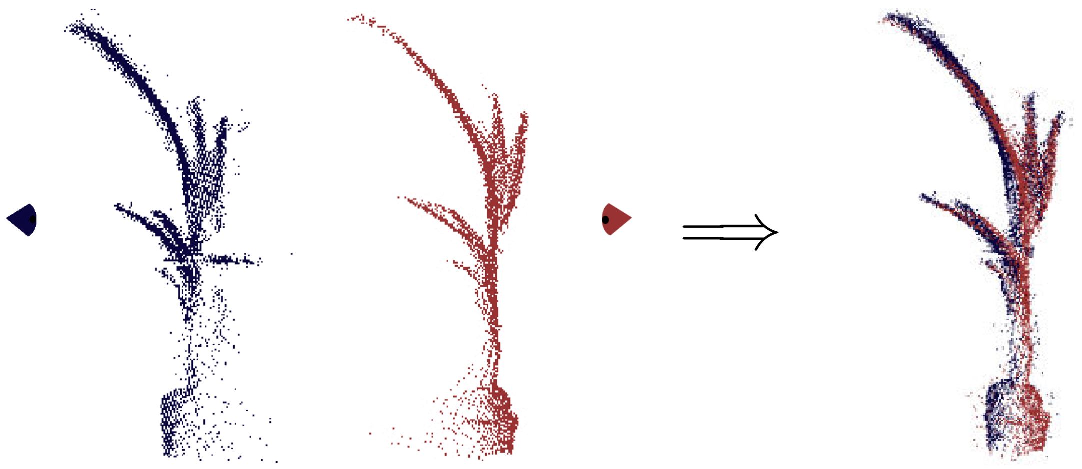


Figure 5.3: This figure shows an example how two partial models of a plant are merged into one. The triangles mark the original viewpoints from which the scans were perceived. (Source: [104])

5.1.4 Determining Partial Similarity and Relative Poses

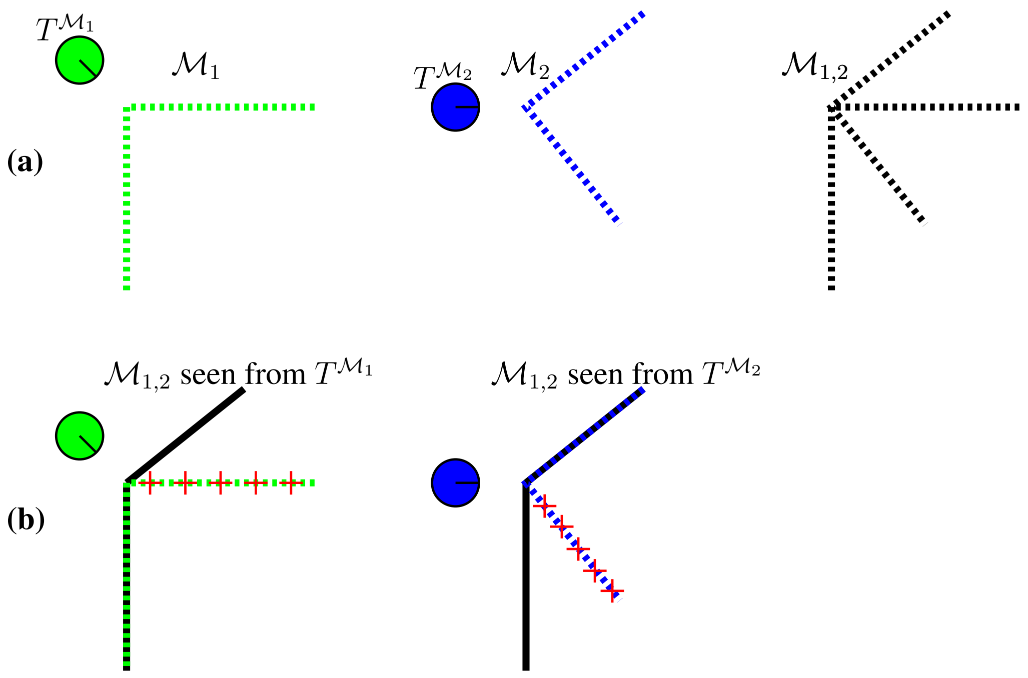
We now have a set of partial object models $\mathfrak{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_{|\mathfrak{M}|}\}$. Every object model contains a set of point clouds $\mathfrak{P}^{\mathcal{M}} = \{\mathcal{P}_1^{\mathcal{M}}, \ldots, \mathcal{P}_{|\mathfrak{P}^{\mathcal{M}}|}^{\mathcal{M}}\}$ and a set of viewpoints $\mathcal{T}^{\mathcal{M}} = \{T_1^{\mathcal{M}}, \ldots, T_{|\mathfrak{P}^{\mathcal{M}}|}^{\mathcal{M}}\}$, meaning the poses from which these point clouds were originally observed. Let $\hat{\mathcal{P}}^{\mathcal{M}}$ be the merged point cloud incorporating all original view point clouds:

$$\hat{\mathcal{P}}^{\mathcal{M}} = \bigcup_{i \in \{1,\ldots,|\mathfrak{P}^{\mathcal{M}}|\}} \{\hat{\mathbf{x}} \mid \hat{\mathbf{x}} = T_i^{\mathcal{M}} \cdot \mathbf{x} \quad \forall \mathbf{x} \in \mathcal{P}_i^{\mathcal{M}}\}. \tag{5.1}$$

It is now our goal to find similar regions in pairs of models and to determine potential alignments between them. For this purpose we employ a similar method as the one we proposed for object recognition in Chapter 4. Let us consider a pair of models $\mathcal{M}_i$ and $\mathcal{M}_j$. As we did in the object recognition system (see Section 4.1) we now sample different views of the models, meaning we calculate range images from different perspectives (see Section 4.1), making sure that the original viewpoints $\mathcal{T}^{\mathcal{M}}$ are also used. Then we calculate point features on the different views (we used a support size of 30 cm in our experiments), match the features from $\mathcal{M}_i$ against those from $\mathcal{M}_j$ and use the found matches to calculate relative transformations between the two models. Since we only need a small number of matches, with a minimum of one, we need only a partial overlap of the two models to find a correspondence between them (see Section 4.3). Afterwards, to reduce the error of these transformations, we apply an additional ICP step. Before this step, the typical error for true relative transformations is about $10\,\text{cm}$ in translation and about 2° in orientation. After the ICP step it typically decreases to $3\,\text{cm}$ in translation and about 0.5° in orientation.

Figure 5.3 shows an example for our model merging procedure. It shows the point clouds of a plant that is visible from two different perspectives in two different scans, which are then merged into a new, more complete model.

We now need to calculate a score for each of the candidate transformations. We do this based on view simulations regarding the original partial views and the merged model. This process is similar to what we did in Section 4.4.2 to score candidate poses in the object recognition system. The main difference is, that we now try to find inconsistencies in multiple observations of the same structure, whereas we only evaluated one perspective of a complete model in the object recognition system. Our scoring method for the potential relative poses between models



$T^{\mathcal{M}_1}$
$\mathcal{M}_1$
(a)
$T^{\mathcal{M}_2}$
$\mathcal{M}_2$
$\mathcal{M}_{1,2}$
$\mathcal{M}_{1,2}$ seen from $T^{\mathcal{M}_1}$
$\mathcal{M}_{1,2}$ seen from $T^{\mathcal{M}_2}$
(b)

Figure 5.4: Visualization of the idea behind the scoring function. **(a)** shows the two models $\mathcal{M}_1$ and $\mathcal{M}_2$, which only include one point cloud from one view each. The perspective from which these were originally seen, $T^{\mathcal{M}_1}$ and $T^{\mathcal{M}_2}$ are also shown. The rightmost structure shows the combined model $\mathcal{M}_{1,2}$ resulting from a wrong alignment. **(b)** shows view simulations of $\mathcal{M}_{1,2}$ from the perspectives defined by $T^{\mathcal{M}_1}$ and $T^{\mathcal{M}_2}$, overlaid by the corresponding original view. Assuming that the merged model is correct, the points marked by red crosses should not have been visible in the original view. Therefore this match will get a low score.

works as follows:

Let us assume that we are evaluating the potential relative pose $T_{i,j}$ between the models $\mathcal{M}_i$ and $\mathcal{M}_j$, which leads to the combined model $\mathcal{M}_{i,j}$. We now calculate range images for all of the original perspectives of the source models, namely the ones defined by $\mathcal{T}^{\mathcal{M}_i}$ and $\mathcal{T}^{\mathcal{M}_j}$, for the point cloud of the merged model $\mathcal{M}_{i,j}$. We then compare these range images with their corresponding original observations. These range image views will most likely differ also for correct relative poses, but inconsistencies show themselves as areas where we get larger ranges for the original view than for the merged model. This indicates that areas were visible that should have been occluded if the merged model were correct. Figure 5.4 visualizes this situation in a two-dimensional example.

To compute the score for one of the views in a merged model, we consider all the range image pixels that were visible in the original view. The merged model will typically have more valid pixels, but since we want to evaluate how well the original observation fits the model, we can only consider pixels that are visible in both, the original and the simulated view. For each of these we calculate the difference in range Δr. Based on a maximum error of the difference in range $e_{\max}$, we define the score of this particular pixel as

$$s = 1 - \frac{\min(|\Delta r|, e_{\max})}{e_{\max}} \quad , \tag{5.2}$$

which means that $s \in [0, 1]$. The score for the complete view is the average over all valid pixels. The score for the complete model is the average over all the scores of the individual views. To reduce the runtime requirements, we first perform this scoring step at a low resolution

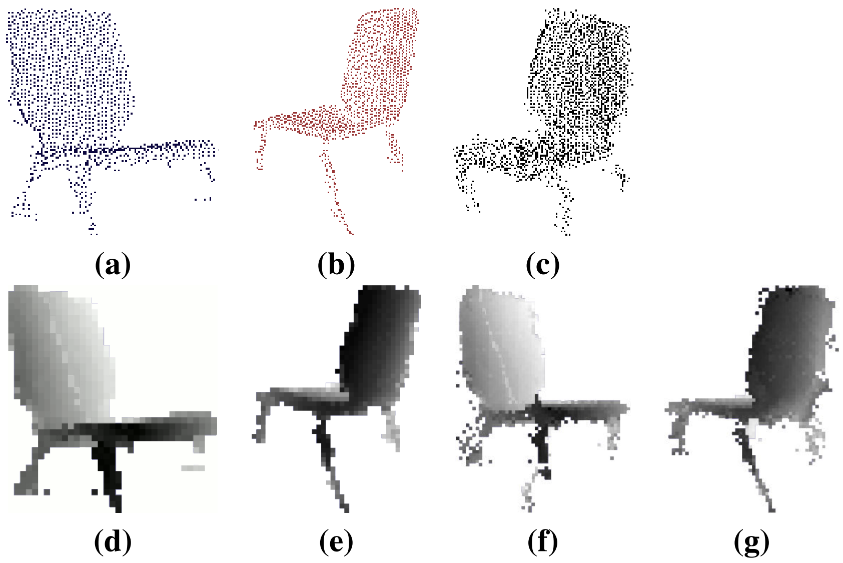


Figure 5.5: Example for the views used in the scoring process. **(a)** and **(b)** show the original views of a wooden chair. **(c)** shows the merged model according to a correctly found transformation between the two partial models. **(d)** and **(e)** show the range images corresponding to (a) and (b). **(f)** and **(g)** show the range images corresponding to the merged model from (c), seen from the same perspectives as (d) and (e). For the calculation of the score of the model shown in (c), the range image from (f) will be compared with (d) and the range image from (g) with (e). (Source: [104])

($1°$ per pixel) and only repeat it at a higher resolution ($0.3°$ per pixel) if the score in the first step is above a minimum value. Figure 5.5 shows an example for the scoring step regarding the model of a wooden chair.

The relative poses calculated from the feature matches typically return multiple solutions for the same pair of models. We calculate the score for all the solutions, but keep only the one with the highest score as the best relative transformation between the models. Based on these best relatives transformations we build a similarity matrix for all our models, where the $|\mathfrak{M}| \times |\mathfrak{M}|$ elements are the scores of the best relative transformations of the corresponding model pairs. See Figure 5.6 for an example of such a matrix. Please note that we define a minimum score for each similarity value in the matrix, below which we set the entry to zero (0.7 in our current implementation).

5.1.5 Merging of Models

We now have to decide, which models are to be joined during one iteration of our algorithm. For this purpose we employ the single cluster graph partitioning algorithm [97] on the similarity matrix. This method tries to find the single best cluster of well connected nodes in a graph by maximizing the average consensus. In our context the partial models are the nodes in a fully connected graph, whereas the found similarities provide weights for the edges. The method boils down to finding the dominant eigenvector of the similarity matrix, which can then be discretized into an indicator vector where each element is either 1, if the corresponding model is part of the main cluster, or 0 if it is not. If the best found cluster according to this method includes at least two models, we merge them into one new model and replace the original models with the new one. Then we go back to the calculation of the similarity matrix for this new set of models, thereby starting the next iteration. Please note that only the entries affected by the removed and added models need to be recomputed in the next step. We will now present

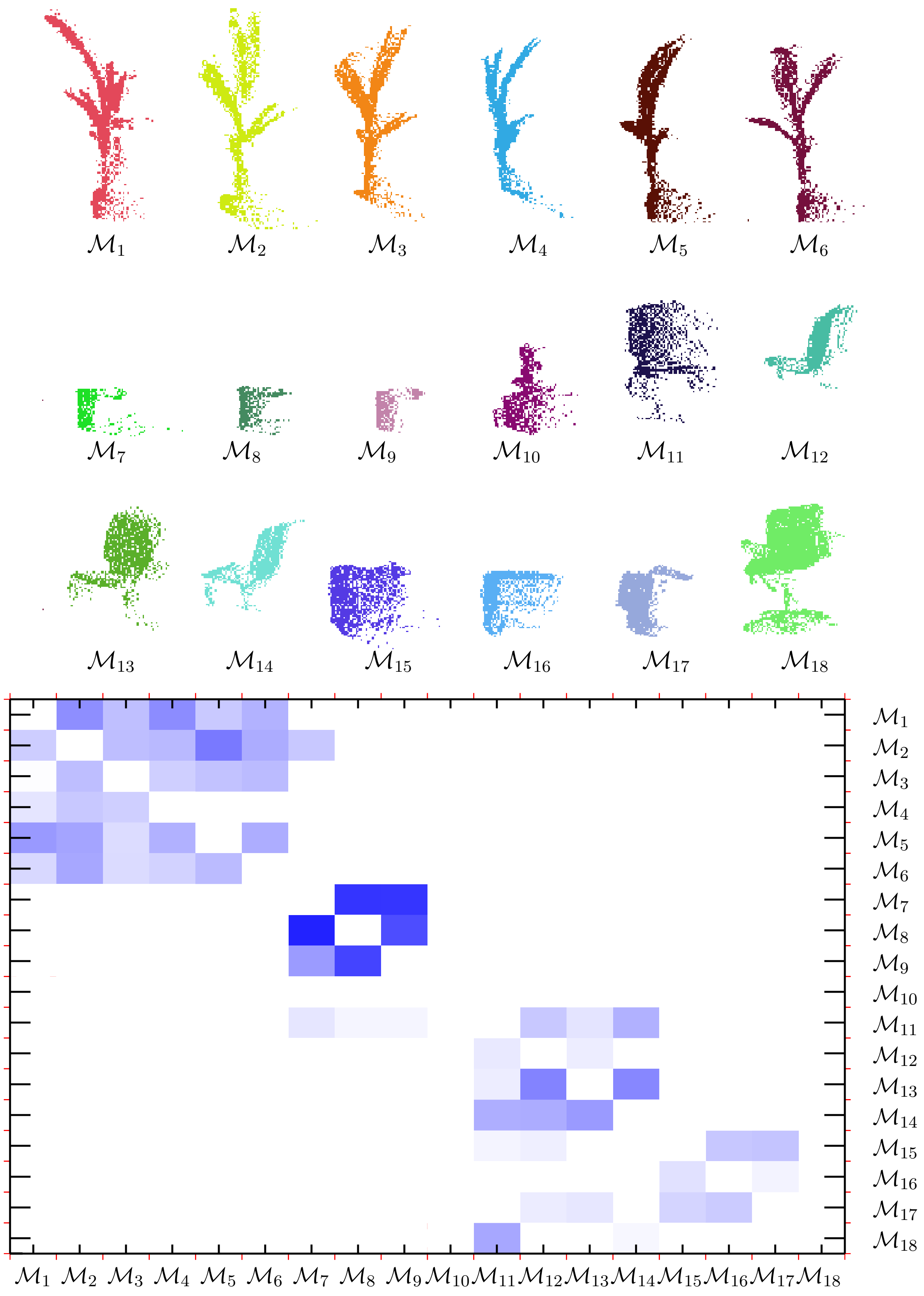


Figure 5.6: Individual model views that were extracted from the list of scans and the similarity matrix. The darker the value in the matrix, the higher the score value for the best found relative transformation between the two partial models. (Source: [104])

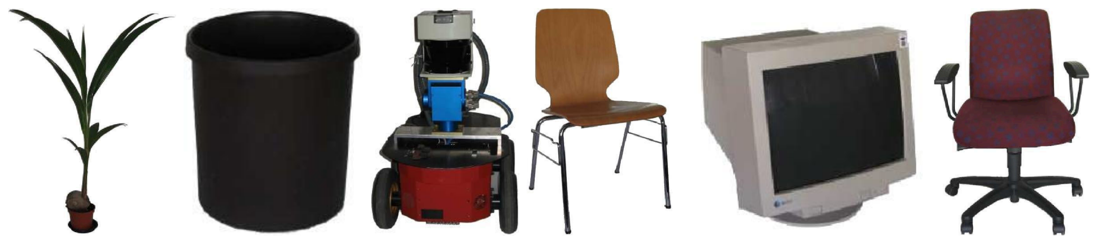


Figure 5.7: The used objects for the experiment. A plant, a garbage bin, a small pioneer robot, a wooden chair, a monitor, and an office chair. (Courtesy of Michael Ruhnke)

the experimental evaluation of our approach.

5.2 Experimental Evaluation

To evaluate our approach, we captured datasets of 3D scans using a tilting SICK laser scanner, from which we can extract range images of up to 0.3° per pixel. We will now present two experiments. In the first one we will show how the set of models evolves during the iterative procedure. In the second one we will analyze how the number of available viewpoints influences the result.

5.2.1 Development of the Model Set

For the first experiment we acquired seven 3D scans in a cluttered office environment. After background subtraction and clustering, our system extracted an initial object set containing 18 partial models. Figure 5.6 shows this initial set of partial models with the corresponding similarity matrix. Please note that we do not calculate the similarity of an object to itself, which is why the diagonal in the matrix is empty. For better visibility we ordered the partial views according to the object they belong to in this visualization. $\{\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4, \mathcal{M}_5, \mathcal{M}_6\}$ are partial views of a plant, $\{\mathcal{M}_7, \mathcal{M}_8, \mathcal{M}_9\}$ of a garbage bin, $\mathcal{M}_{10}$ of a small pioneer robot, $\{\mathcal{M}_{11}, \mathcal{M}_{12}, \mathcal{M}_{13}, \mathcal{M}_{14}\}$ of a wooden chair, $\{\mathcal{M}_{15}, \mathcal{M}_{16}, \mathcal{M}_{17}\}$ of a monitor, and $\mathcal{M}_{18}$ of an office chair. See Figure 5.7 for photos of the used objects.

The blocks in the matrix already show that our similarity measure properly separates the different objects, although there are some pairs that falsely received high similarity values, like $\mathcal{M}_{11}$ and $\mathcal{M}_{18}$. Yet, as we will see, this does not necessarily lead to wrong object merges.

Figure 5.8 shows how the set of models evolved after two iterations. Some of the models of the plant, $\mathcal{M}_2$ and $\mathcal{M}_8$, are already correctly merged. The garbage bin models $\mathcal{M}_7$, $\mathcal{M}_8$, and $\mathcal{M}_9$ were also correctly merged. Yet, since the garbage bin is completely symmetrical, our system was only able to merge them by maximizing the surface overlap and was not able to reconstruct the complete round structure. This is a general problem of a lack of structure and could only be solved with additional data about the captured scans.

The final model set is shown in Figure 5.9. It took 18 minutes on a 1.7 GHz Pentium 4 processor until our system converged to this state. There are no falsely merged models in the final set and our system only failed to merge M_{15} with $M_{16:17}$, due to a very limited overlap. Additional views could enable our system to also merge these observations correctly. This behavior will be demonstrated in the next experiment.

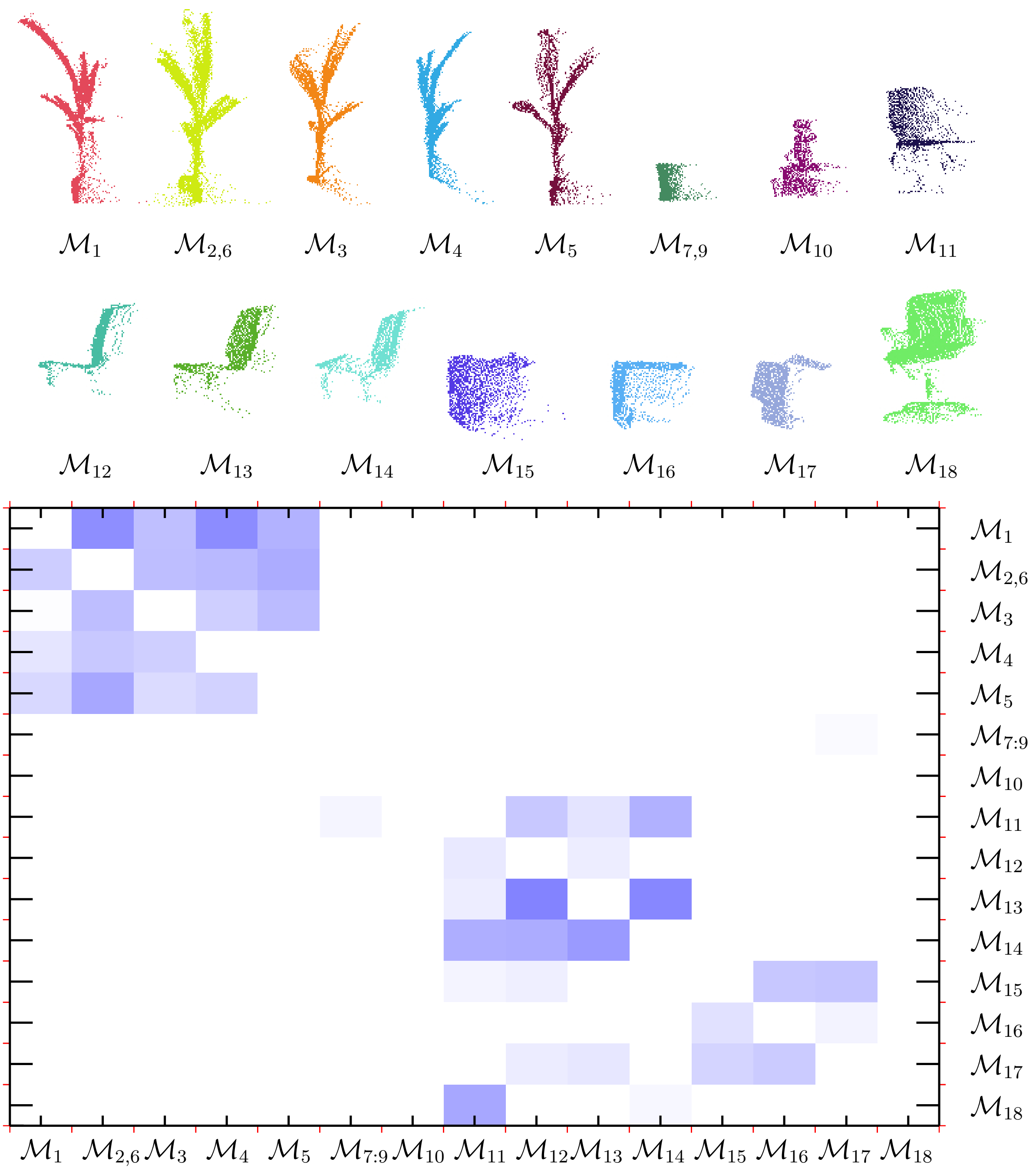


Figure 5.8: Individual model views and similarity matrix after two iterations of our model merging procedure. Two views from the plant and three from the garbage bin are already correctly merged. (Source: [104])

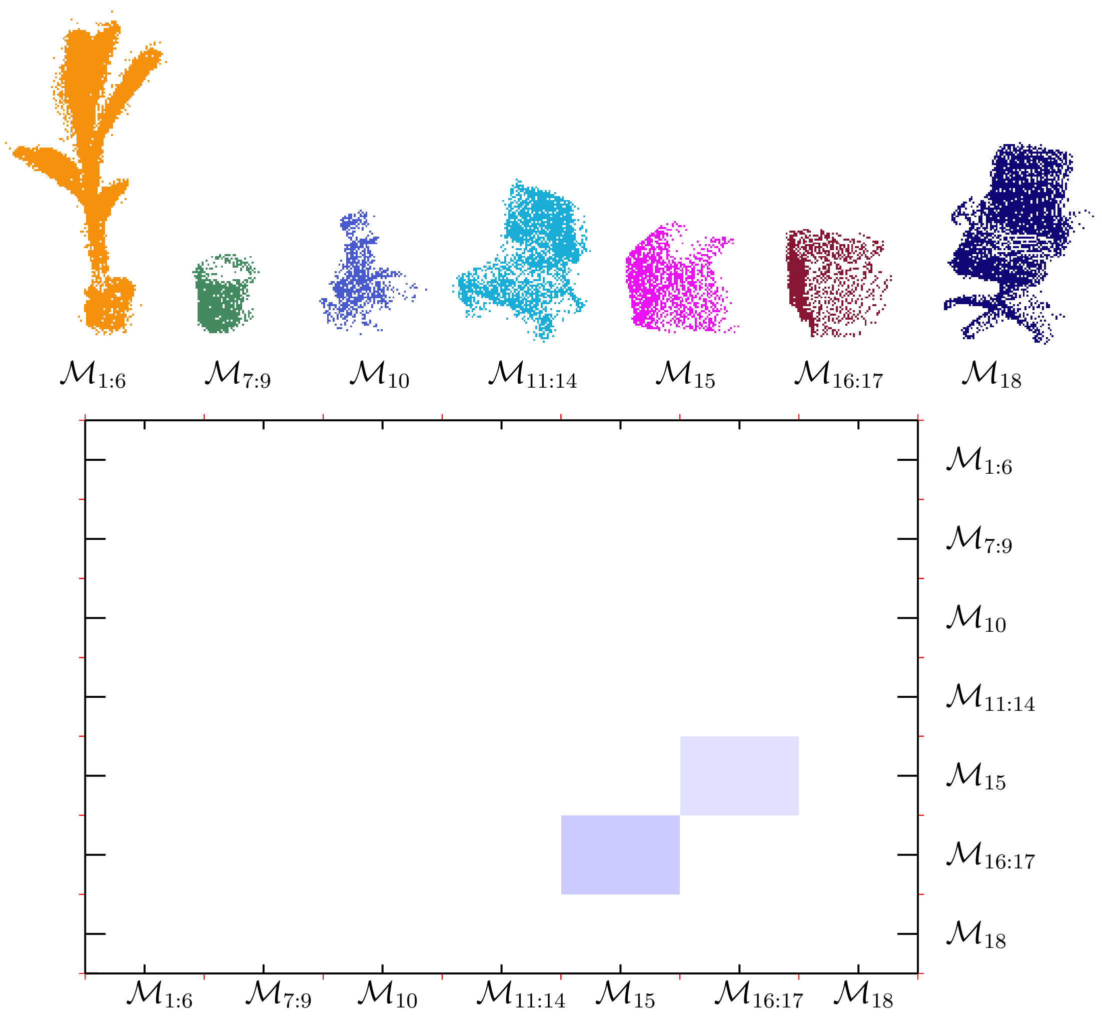


Figure 5.9: Final models and the corresponding similarity matrix. There are no falsely merged models and our system only failed to merge the remaining two models of the monitor, due to the limited overlap. (Source: [104])

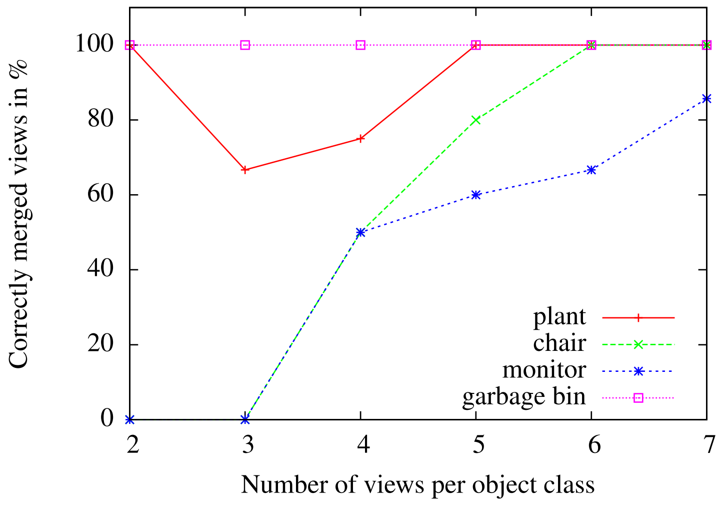


Figure 5.10: This plot shows how the number of available views for a certain model influences the merging capability. As expected, the more views are available for the system, the better the matching.

5.2.2 Influence of the Number of Views

In the second experiment we analyzed, how the number of available views for an object influences the merging results. For this purpose we collected seven views of a garbage bin, a monitor, a plant, and a chair, with evenly distributed perspectives around the objects. We used these sets to test the merging capabilities for all set sizes between two and seven.

The plot in Figure 5.10 summarizes the outcome of this experiment. It shows which percentage of the sets could be successfully matched depending on the set size. The different colors are for the different objects. It shows the expected behavior, meaning that more views simplify the merging process. The needed number of views depends strongly on the structure of the object and how much the different views are influenced by self-occlusion. E.g., the views of the garbage bin do not differ much with changed perspectives and the plant provides a lot of unique structure to ease the matching process. Therefore these two models need very few views to reach high success rates. On the other hand, the monitor and the chair are strongly influenced by self-occlusion, which is why more views are needed here to create complete models.

5.3 Related Work

The creation of object models is a highly relevant topic in perceptual tasks since they are a vital input for systems that need to find instances of these objects or try to learn classifiers for certain types of objects. Being able to learn objects in an unsupervised fashion from unconnected data can save a lot of manual work and thereby ease the creation of large model databases.

In computer vision several methods for object discovery, meaning the search for reoccurring structures in unlabeled visual images, have been proposed in the past [149, 118, 133, 67]. A useful comparison of such methods was published by Tuytelaars *et al.* [143].

In the context of 3D range based perception most attempts to discover objects do not actually learn a complete 3D representation of objects but aim at finding classifiers to group the found structures into different types. E.g., Endres *et al.* [37] presented an approach for unsupervised discovery of object classes in 3D range scans. They use a variant of Spin Images to represent surfaces and model object classes as distributions of these features. They apply Latent Dirichlet Allocation [11] to cluster 3D objects regarding shape similarity. The learned models can later on be used to detect these object classes in new data. This system aims at object categories rather than specific objects and does not create 3D models, but a collection of shape properties encoded in feature descriptors.

Other approaches use changes over time to find and model non-stationary objects. E.g., Anguelov *et al.* [3] learn 2D models of non-stationary objects from multiple instances of complete 2D maps of an environment, captured at different times, between which the objects moved. The objects are found through differencing and represented in a hierarchical fashion. Herbst *et al.* [58] follow a similar goal, but using RGB-D (color and depth) sensors. They build colored 3D maps of an environment at different times and detect segments that moved in between. These segments are merged into object models using spectral clustering. By using RGB-D data they can use shape and texture to distinguish different objects. Compared to these two approaches, we can also detect static objects and do not rely on a pre-built map, but on unconnected 3D scans, each captured from only one perspective.

Shin *et al.* [116] also presented a system for unsupervised object discovery and classification. Like we do, they first perform background extraction, but based on a saliency measure. Then they find potential object parts using a clustering approach that considers metric distance and similarity regarding extracted features. These object parts then form objects in a hierarchical fashion. One advantage of their method is that the object classification step is able to split clusters again, thereby being robust against mistakes that happened in the clustering step. The extracted object types can then be used to classify parts of the environment. While this method works well to create and find object classes, it does not provide the means to merge different views into a single point cloud model.

Moosmann *et al.* [92] focused on object class discovery in 3D outdoor datasets (trees, cars, buildings, etc.). Their method does not rely on a perfect segmentation of the scans. Instead they propose a hierarchical segmentation, where each level in the hierarchy represents different segmentation parameters. A feature descriptor is calculated for every node, enabling similarity checks between them. This similarity measure is then used to create and assign object classes. This method is only able to detect structures that occur frequently and also aims at providing a general classifier for object classes instead of creating specific point cloud models.

5.4 Conclusions

In this chapter we presented a novel method for unsupervised object model learning from an unconnected collection of 3D range scans. The method employs point features to address this hard 3D perception problem. Our system finds potential objects using a combination of background subtraction and spatial clustering on the individual scans. These partial object views are then merged into more complete models based on a similarity measure that is derived from the methods we used for object recognition in Chapter 4. We presented experiments using real data from a 3D laser range scanner that showed that our system can robustly detect and merge object instances from cluttered scenes while being able to reliably distinguish different objects that should not be merged.

Chapter 6

Using a Dictionary of Surface Primitives for 3D Environment Modeling

The way how a robot models a 3D environment can have a strong influence on its performance in different tasks. Most representations, like databases of individual scans, point clouds, or 3D grids have high memory requirements since they contain much redundant information. In this chapter we introduce a new approach to represent 3D environments. It is based on the insight that typical human made environments contain many repetitive structures. We therefore developed a method that describes a 3D scene based on a limited number of surface patches. Our system learns a dictionary of surface primitives from a collection of scans. This dictionary can then be used to model 3D scenes by means of indices to the primitives and positions, where they occur. Our system determines a model for a given dataset under consideration of the Bayesian information criterion, which weighs the model complexity against the accuracy of the reconstruction. We show that even a small dictionary can be used to accurately describe typical 3D scenes in a condensed form. In addition, this representation has the benefit of being semantically richer than the raw sensor data since it directly encodes similarities in the environment.

* * * *

In the previous chapters we discussed methods to find similar structures in 3D scenes. First in the context of basic feature matching, then to find known object models in 3D scenes and then to create such object models by merging several overlapping observations into one representation. Finding similar structures can also be used to simplify the description of a scene. If there are several identical objects in a 3D environment, we only have to describe the object once and then save the information, where the individual instances occur. This makes it possible that we do not have to provide a detailed description of the same structure over and over again in our representation of the environment. If we only consider large objects for the description, this may only have a small impact on the overall amount of data since identical objects only occur irregularly, like, e.g., chairs, tables or closets in an office environment. But at a smaller scale, 3D structures are very repetitive. There are many patches of flat surfaces, rectangular corners, straight edges, etc. in human-made environments. To detect those occurrences would enable us to describe them all with a small number of *surface primitive* respectively. This brings us back to the problem of finding similar structures in 3D data.

We will present a system that, given a dataset of 3D scenes, learns a *dictionary* of *words*. Each of these words contains one of the aforementioned surface primitives, describing a small part of a 3D scene in the form of a point cloud. This dictionary can then be used to describe a 3D scene, by assigning the most similar surface primitive to each individual part. These *instances* of the words only need the information which surface primitive is to be used, typically in form of a simple index, and the 6DOF position in the scene, where it occurs.

It is a common concept in the area of data compression to find identical data blocks and represent them with a single dictionary word. This leads to a compression because the links to the dictionary typically take less space than the word itself. A popular example for a lossless compression algorithm that is based on this principle is the LZW-algorithm [93]. This method can be used for arbitrary data streams since the dictionary is created during runtime and encoded in the compressed file. In the approach presented in this chapter, we are not focused on lossless compression of arbitrary data, but on the potentially lossy compression of 3D range data captured with noisy sensors. In vision, there exists an abundance of lossy and lossless image compression methods, e.g., the famous JPEG standard [148]. Yet in the area of 3D measurements such methods are not yet very common. Our approach aims at closing this gap and replaces parts of the scene by surface patches that are not identical, but similar. Finding these similarities in a naive way is computationally very demanding, which is why we will make use of the feature extraction methods presented in Chapter 3. The NARF descriptor is used to perform the necessary similarity comparisons. We also adapted the NARF keypoint extraction method to fit the problem of placing word instances in a scene.

Besides reducing the amount of data necessary to describe a 3D scene, this representation also provides more semantic context since it explicitly describes which structures in the scene are similar to each other.

The basic principle of our method is to build the dictionary and reconstruction of a scene together and optimize it in an iterative fashion by minimizing the Bayesian information criterion (BIC). This provides a trade-off between the complexity of the model and its accuracy while needing very few user defined parameters. This means that the size of the dictionary is determined by the algorithm, based on the actual need of the current scene.

See Figure 6.1 for a visual example of a scene that was described with a dictionary containing 114 words. The points in blue mark areas which can be accurately described by the words from the dictionary. Red points mark areas were the reconstruction error was too high. The top left shows an example for one word that was chosen by the algorithm to be part of the dictionary. The boxes show different perspectives for the point cloud stored in the word.

We tested this approach on large real world datasets and our experiments suggest that even a small dictionary (below 100 words) can be used for accurate reconstructions of typical indoor 3D scenes.

The remainder of this chapter is structured as follows: in Section 6.1 we will give a formal definition of the problem at hand. In Section 6.2 we discuss our method to evaluate a model together with a dictionary. Section 6.3 is about heuristic methods we employ to reduce the search space. In Section 6.4 we present our iterative method to learn a dictionary and a model for a given scene. Our experimental evaluation is contained in Section 6.5. Finally, we discuss related work in Section 6.6 and will conclude the chapter in Section 6.7.

6.1 Formal Problem Definition

The input for our system is a dataset of 3D scans with known sensor poses. The contained 3D measurements can be merged into a single point cloud $\mathcal{S} = \{\mathrm{s}_1, \mathrm{s}_2, \dots\}$. It is our goal to construct a dictionary $\mathcal{D} = \{\mathrm{w}_1, \mathrm{w}_2, \dots\}$, where each word $\mathbf{w}$ contains a small point cloud

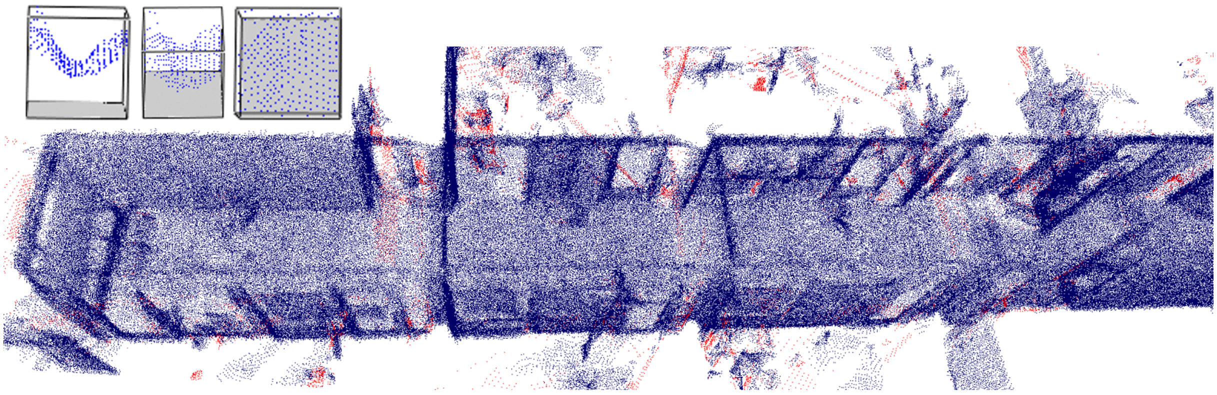


Figure 6.1: This figure shows a 3D scene of a corridor which was reconstructed from a model with a dictionary of size 114. The dark blue points show areas that were accurately described by the model. Points from the original point cloud that are not covered well by the dictionary are shown in red. The boxes in the top left visualize an exemplary word that is part of the dictionary for this scene. The point cloud from the word is shown in three different perspectives.

$\mathbf{w}^{\mathcal{P}}$. In our specific implementation, these point clouds contained in the words are subsets of the original point cloud $\mathcal{S}$ from a fixed size neighborhood (diameter ρ) around a point $\mathbf{p} \in \mathcal{S}$:

$$\mathbf{w}^{\mathcal{P}} = \left\{ \mathbf{s} \;\middle|\; \mathbf{s} \in \mathcal{S},\; ||\mathbf{s} - \mathbf{p}|| \leq \frac{\rho}{2} \right\}. \tag{6.1}$$

A given 3D scene can now be approximated as a list of instances, where these words occur. We will call this a model $\mathcal{M}(\mathcal{D}, \mathcal{S}) = \{\mathbf{m}_1, \mathbf{m}_2, \ldots\}$ for dictionary $\mathcal{D}$ and scene $\mathcal{S}$. Each instance $\mathbf{m} \in \mathcal{M}$ is a tuple $\mathbf{m} = \langle \text{id}, T \rangle$ where id is the index of a word in the dictionary and T describes a 3D transformation. E.g., $\mathbf{m}_j = \langle i, T_j \rangle$ would mean that the points from $\mathbf{w}_i^{\mathcal{P}}$, which is the point cloud contained in word $\mathbf{w}_i$, have to be transformed using T_j, to be in the right positions to form a part of the reconstructed point cloud. Therefore, the reconstructed scene $\mathcal{R}$ is defined as the union of all those transformed point clouds:

$$\mathcal{R}(\mathcal{M}, \mathcal{D}) = \bigcup_{j=1}^{|\mathcal{M}|} T_j \; \mathbf{w}_{\text{id}_j}^{\mathcal{P}}. \tag{6.2}$$

Since the point clouds from the different word instances might be overlapping, there will be areas that are unnecessarily dense. Therefore our specific implementation includes a sparsification step that reduces the resolution in those areas to a pre-defined value, meaning that we enforce a minimum distance between the points in $\mathcal{R}$.

6.2 Model Evaluation

The quality of the reconstruction clearly depends strongly on the available words in the dictionary. With a growing number of words in the dictionary, the chances to find a good match at a certain position in the 3D scene also increases. Yet, this also increases the memory footprint of the dictionary. An exaggerated dictionary can easily need more memory than the scene for which we want to use it. We need to find a balance between the size of the dictionary and the accuracy of the reconstruction. The Bayesian Information Criterion (BIC) is designed to capture this tradeoff. The BIC is a general tool for model selection problems. It is typically defined as

$$\text{BIC} = -2\ln(l) + k\ln(N). \tag{6.3}$$

Here l is the likelihood of the current model, k is the number of free parameters in the model and N is the number of data points in the original observation. For the problem at hand, this leads to

$$\begin{aligned}\text{BIC}(\mathcal{S},\mathcal{M},\mathcal{D}) &= -2\ln(l(\mathcal{S},\mathcal{R}(\mathcal{M},\mathcal{D}))) + \\ &\left(7|\mathcal{M}| + 3\sum_{i=1}^{|\mathcal{D}|}|\mathbf{w}_i^{\mathcal{P}}|\right)\ln(3|\mathcal{S}|).\end{aligned} \tag{6.4}$$

The likelihood depends on the original point cloud and the reconstructed point cloud since it captures the error between the two. It will be defined below. The number of free parameters in the dictionary is the size of the model, multiplied by seven because every instance contains one index and one 6DOF transformation, plus the complete number of points in the word point clouds, multiplied by three since the points are three-dimensional. The last term is the number of points in the original point cloud, multiplied by three. It is our goal to find a model that minimizes the BIC, thereby being maximally consistent with the input data, while having a limited complexity.

Under the assumption that the points in $\mathcal{S}$ are independent of each other, i.e., we assume that the error of each point in the cloud can be evaluated individually, we can define the log-likelihood as follows:

$$\ln(l(\mathcal{S},\mathcal{R})) = \ln\left(\prod_{i=1}^{|\mathcal{S}|} p(\mathbf{s}_i \mid \mathcal{R})\right) \tag{6.5}$$

$$= \sum_{i=1}^{|\mathcal{S}|} \ln(p(\mathbf{s}_i \mid \mathcal{R})). \tag{6.6}$$

This means that we can analyze how well the single points are represented in the reconstructed scene and sum up their log-likelihoods. The calculation of the probability for the individual points requires the introduction of a data association between $\mathbf{s}_i$ and the points from the reconstructed scene $\mathcal{R} = \{\mathbf{r}_1, \mathbf{r}_2, \dots\}$:

$$p(\mathbf{s}_i \mid \mathcal{R}) = \sum_{j=1}^{|\mathcal{R}|} p(\mathbf{s}_i \mid \mathbf{r}_j, a_{i,j}) p(a_{i,j}). \tag{6.7}$$

Here, $a_{i,j}$ defines that there is a data association between $\mathbf{s}_i \in \mathcal{S}$ and $\mathbf{r}_j \in \mathcal{R}$.

Unfortunately, we lack any knowledge about the data associations. Therefore, we assume a uniform distribution of the prior $p(a_{i,j})$, leading to

$$p(\mathbf{s}_i \mid \mathcal{R}) = \frac{1}{|\mathcal{R}|}\sum_{j=1}^{|\mathcal{R}|} p(\mathbf{s}_i \mid \mathbf{r}_j, a_{\mathbf{s}_i,\mathbf{r}_j}). \tag{6.8}$$

Given a data association $a_{i,j}$, we model the probability for the point pair as a spherical Gaussian:

$$p(\mathbf{s}_i \mid \mathbf{r}_j, a_{i,j}) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}\|\mathbf{s}_i - \mathbf{r}_j\|^2). \tag{6.9}$$

The standard deviation σ should be dependent on the resolution of the source point cloud. Since we aim for scenes captured with real sensors, like laser scanners or 3D cameras, the resolution differs with the distance from the sensor. Assuming an angular resolution of α, we compute $\sigma_{\mathbf{s}_i}$ as follows:

$$\sigma_{\mathbf{s}_i} = \omega \ \tan\left(\frac{\alpha}{2}\right) \ \text{range}(\mathbf{s}_i), \tag{6.10}$$

whereas range denotes the distance between the point $\mathbf{s}_i$ and the original sensor position.

In our implementation, we truncate the Gaussian distribution at 3σ. This reduces the runtime since we only have to consider the points in a local neighborhood in $\mathcal{R}$ instead of the whole point cloud.

Up to now we described how to evaluate the BIC of a given model $\mathcal{M}(\mathcal{D}, \mathcal{S})$. We will now start to explain, how our system constructs a dictionary and a model from a given scene.

6.3 Heuristic Search Space Reduction

As defined in Section 6.1, a model $\mathcal{M}(\mathcal{D}, \mathcal{S})$ for dictionary $\mathcal{D}$ and scene $\mathcal{S}$ contains tuples $\{\mathbf{m}_1, \mathbf{m}_2, \dots\}$, where every tuple $\mathbf{m} = \langle \text{id}, T \rangle$ contains the index to a word and its position in the scene. Finding the optimal dictionary and model for a given scene, meaning the combination with the smallest BIC, is hard as the search space is huge. In theory, every combination of arbitrarily shaped words has to be checked for every possible combination of positions in the scene. Since we cannot check the complete search space, we have to rely on heuristics to find a good, even if not optimal, solution. The main heuristics used to reduce the search space for our system are:

- We use words of a fixed size and shape, namely spheres with a pre-defined diameter. We will refer to this diameter as the *support size* ρ of the words. This is in accordance with what we called the size of the area covered by a 3D point feature descriptor in Chapter 3.
- We determine the position of the instances in the scene only once at the beginning and do not change them afterwards, i.e., we have fixed positions for the instances, even if the word they refer to is variable. We define the following requirements regarding the placement of the instances:
  - The scene should be completely covered according to the support size ρ of our words but without an excessive overlap.
  - The same pose should be found for every instance of a complex recurring structure in order to minimize the number of words in the dictionary and to facilitate the feature matching process.

To achieve the above requirements for the placement of the instances, we use a combination of a keypoint extraction method and uniform sampling on the scene. The keypoint extraction is supposed to find points in the scene that have stable positions like corners or points with partially stable positions, e.g., on straight edges. The uniform sampling is supposed to cover areas where the surface does not change in any significant way, like flat surfaces.

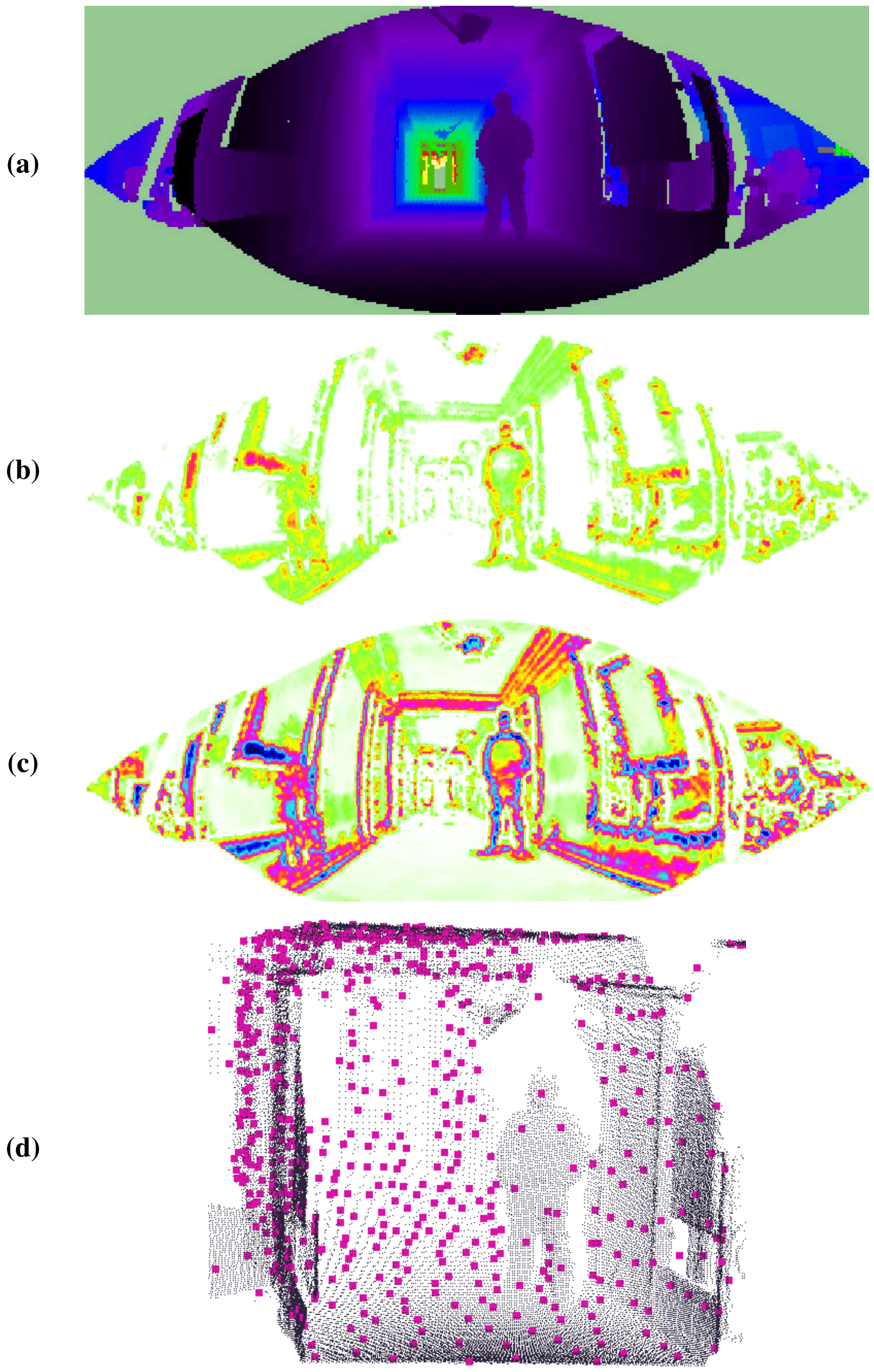


Figure 6.2: This figure visualizes the procedure used for deciding where to place the words in the scene. **(a)** A range image of the original scene, showing a person standing in a corridor. **(b)** The scores for the original NARF keypoint extraction. **(c)** Scores for our adapted version, that also gives high scores to areas close to straight edges. **(d)** The points extracted from these scores marked in the original point cloud.

For this purpose we adapted the NARF keypoint extraction method presented in Section 3.2. The original NARF keypoint detection was created to return points where the surface is stable (to ensure a robust estimation of the normal), but where there are sufficient changes in the immediate vicinity. Based on this criteria a score (the interest value) is calculated for every point in a range image (see Figure 6.2(a,b) for an example) and the keypoints are found using non-maximum suppression. To adapt this method for the current application, we developed an new scoring function that, while still giving high scores for the original criteria, also favors points on straight edges. For this purpose we added a term that increased the interest value for points close to high curvature values (see Figure 6.2(c)). In addition, to ensure coverage of the complete scene, we exchanged the non-maximum suppression for the selection of keypoints with the following procedure:

1. find the point with the highest interest value
2. add this point to the list of selected points
3. set all scores in a certain radius around the point to a negative value
4. repeat until no point with a score of at least zero is left

Thereby we first add all points on complex structure, which typically have a precise position, then the points on straight edges, where the positions are well defined in one axis, and in the end the remaining points on uniform structures (see Figure 6.2(d)). The radius mentioned above is chosen according to the support size ρ of our words since a big enough overlap has to be guaranteed.

In the next step, the NARF descriptor (see Section 3.3) is calculated for all selected points. These descriptors will enable us to perform fast similarity checks between the instances and the words in the dictionary. Figure 6.3 gives a visual example for these similarity checks. The dark areas in the individual images are suited to be represented by only one word since they have a very similar surface structure. Additionally, as explained in Section 3.3) this process also provides a unique 6DOF coordinate frame that is calculated from the local structure, thereby providing the means to calculate the 6D transformations needed for every instance in our model.

With this done, we already determined the size of the model $|\mathcal{M}|$ and the 3D positions of the instances in the scene. Please note, that for this case the calculation of the BIC from Equation 6.4 can be simplified to

$$\begin{aligned} \mathrm{BIC}'(\mathcal{S},\mathcal{M},\mathcal{D}) &= -2\ \ln(l(\mathcal{S},\mathcal{R}(\mathcal{M},\mathcal{D}))) + \\ & \left(3\sum_{i=1}^{|\mathcal{D}|}|\mathbf{w}_i^{\mathcal{P}}|\right)\ \ln(3\ |\mathcal{S}|) \end{aligned} \tag{6.11}$$

since the term $7|\mathcal{M}|\ln(3\ |\mathcal{S}|)$ from the original equation is a constant and will only influence the absolute number of the BIC, but not the minimization process.

We have a local point cloud for every instance and a descriptor for every local point cloud. The main information missing for a complete model are the indices of the dictionary words to be used for the instances. If a dictionary $\mathcal{D}$ is given, where every word also includes a descriptor for its contained point cloud, $\mathcal{M}(\mathcal{D},\mathcal{S})$ can be calculated by assigning the most similar word to every instance. This is done by finding the word with the minimum descriptor distance between the NARF descriptor associated with the word and the NARF descriptor associated with the

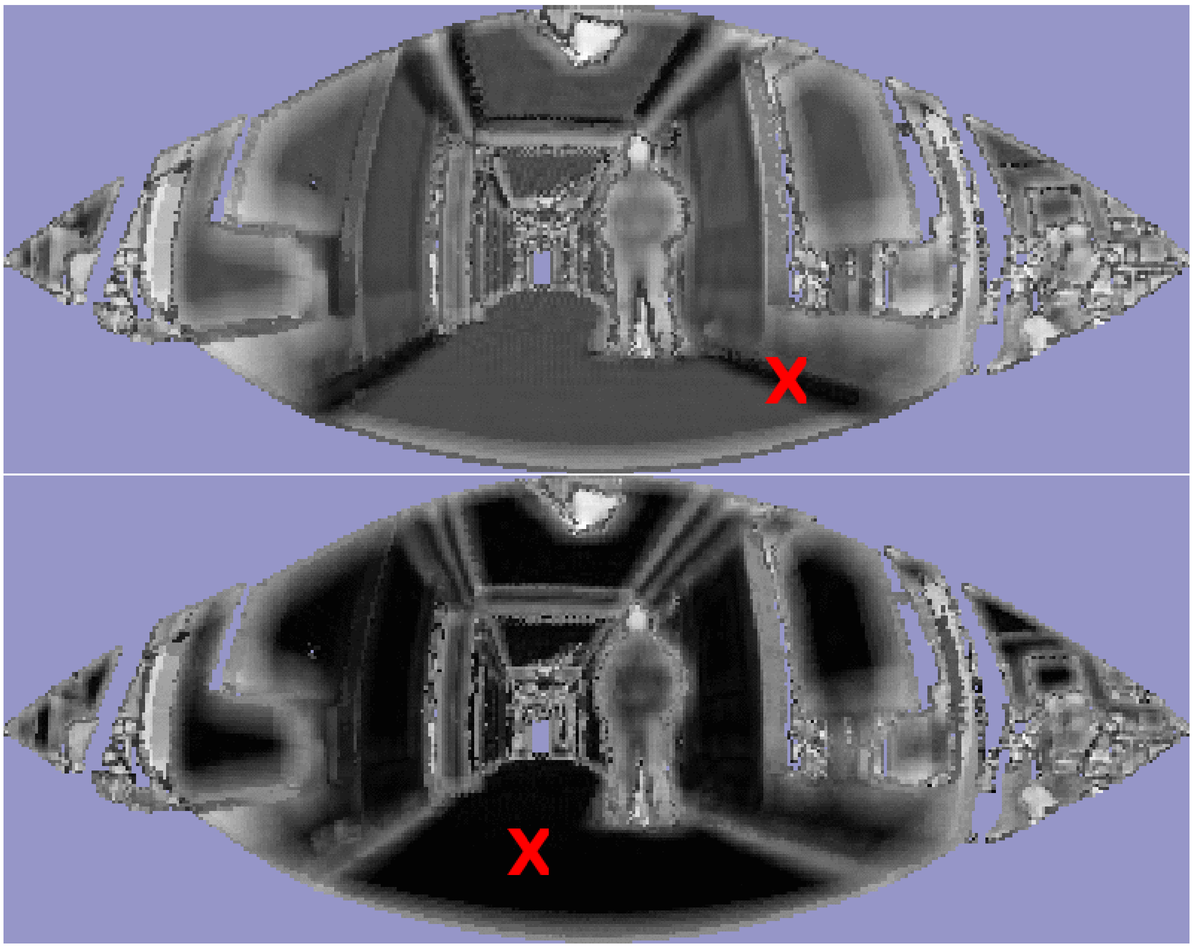


Figure 6.3: This figure visualizes how the features can be used to determine the similarity between two regions in the 3D data. It shows the same scene as in Figure 6.2(a). The feature at the position marked with a cross is compared with every other position in the scene and the descriptor distances are visualized. Dark means high similarity and bright means low similarity. In these examples, a feature describing a 90° angle between two flat surfaces (top) and a completely flat surface (bottom) are visualized.

instance. The transformation T for an instance $\mathbf{m} = \langle \mathrm{id}, T \rangle$ can be calculated based on the transformations associated with the NARFs as $T = T_a^{-1} \cdot T_s$, where T_a is the transformation associated with the NARF from the dictionary word and T_s is the transformation associated with the NARF from the current instance.

We will now explain the procedure to iteratively find a good dictionary and model regarding the corresponding BIC value.

6.4 Dictionary Refinement

Based on what we discussed in Section 6.3, we can construct a temporary dictionary $\mathcal{D}_{\mathcal{S}}$, that contains all words occurring in the scene $\mathcal{S}$ at the keypoint positions, whereas each word is associated with the local point cloud, the extracted transformation and the NARF descriptor. This dictionary could be used to represent the original scene perfectly since every partial surface from the scene is available in it. Yet, the dictionary is very large, which would lead to a very high BIC value.

Instead, we try to build a good dictionary and model by iteratively changing the dictionary and keeping the new version if it has a smaller BIC value than the last version. The concrete implementation works as follows:

1. Start with an empty dictionary $\mathcal{D}$.
2. Randomly perform one of the following actions:
   - Insert action: Choose a random word from $\mathcal{D}_\mathcal{S} \setminus \mathcal{D}$ and add it to $\mathcal{D}$.
   - Delete action (if $|\mathcal{D}| \geq 2$): Remove a random word from $\mathcal{D}$.
   - Replace action (if $|\mathcal{D}| \geq 1$): Remove a random word from $\mathcal{D}$ and add a random word from $\mathcal{D}_\mathcal{S} \setminus \mathcal{D}$ in its place.
3. Create the model $\mathcal{M}(\mathcal{D}, \mathcal{S})$ as explained in Section 6.3. Please note that the assignment of words to the instances can be calculated rather efficiently by caching the results of the last iteration and only performing updates regarding the changes caused by the last action taken in step 2.
4. Calculate the reconstructed scene from the current model and dictionary $\mathcal{R}(\mathcal{M}, \mathcal{D})$ as described by Equation 6.2.
5. Calculate BIC$(\mathcal{S}, \mathcal{M}, \mathcal{D})$ as described in Section 6.2.
6. If the BIC did improve, we accept the new dictionary. Otherwise we backtrack to the former dictionary. If a maximum number of iterations is reached we stop the procedure. Else we go back to step 2.

6.5 Experimental Evaluation

We will now present some experiments to evaluate how our approach performs in different settings. We will analyze the size of the dictionary $|\mathcal{D}|$, the size of the model $|\mathcal{M}|$, the number of inliers (points from the original scene that are correctly represented in the reconstructed point cloud), the number of outliers (points contained in the reconstructed point cloud that do not exists in the original data), and how much memory is saved compared to the original point cloud representation. For the latter we use a standard binary encoding. Every 3D point uses 3 floating point values (12 Bytes), every 6DOF transformation uses 6 floating point values (24 Bytes), and a word index uses one integer (12 Bytes). The inliers and outliers are determined regarding a maximum distance to the next correct point of $\sigma_{\mathbf{s}_i}$ (see Section 6.2).

Our algorithm ends the iterative process, when the BIC did not decrease for 100 iterations.

6.5.1 Influence of the Standard Deviation Parameter

Our method has mainly two adjustable parameters: the support size ρ and ω, which is used to calculate the standard deviation σ (see Equation 6.10) for a certain measurement range. The standard deviation σ determines how accurate the model has to be because it restricts the maximum error between point positions in the likelihood computation. To analyze the concrete influence of different values of σ on the model creation process, we performed a small scale experiment with the point cloud of a chair visualized in Figure 6.4(a). We performed the model creation process for $\sigma \in [0.1\,\text{mm}, 20\,\text{mm}]$ and $\rho = 20\,\text{cm}$. Figure 6.4(b) shows how the different values influence the dictionary size. As expected, the dictionary size decreases with increasing values of σ since errors in the approximation receive a lower weight in the likelihood evaluation and therefore less words are needed to describe the point cloud. Figure 6.4(d) shows the reconstructed point cloud for a dictionary of size 34, whereas Figure 6.4(e) shows the reconstructed point cloud for a dictionary of size 12. As expected, the cloud in (e) does not represent the original point cloud as well as the one shown in (d). The backrest is flat, while the original is slightly curved and the fine structure of the feet of the chair is also not well represented. Figure 6.4(c) shows the evolution of the BIC and the dictionary size during the iterative refinement for the model from (d). While the BIC stabilizes after about 300 iterations, the dictionary size needs slightly above 400 to get to a nearly constant level. In the beginning every inserted word has a high chance to be a better approximation of the structure somewhere in the point cloud and therefore justifies the increase in the dictionary size. Later on, mainly replacements of words still have a chance to lower the BIC, leading to a much slower progress.

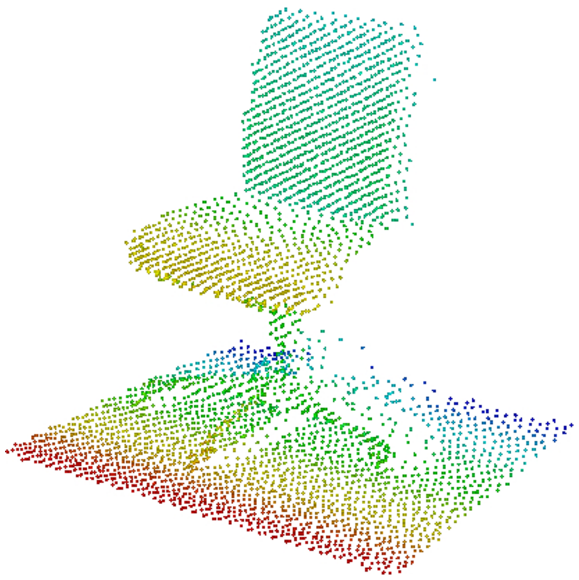
For the remaining experiments we will use $\omega = 0.5$ to calculate the range and resolution dependent values for σ, which is a good tradeoff between accuracy and size.

6.5.2 Small Indoor Scene

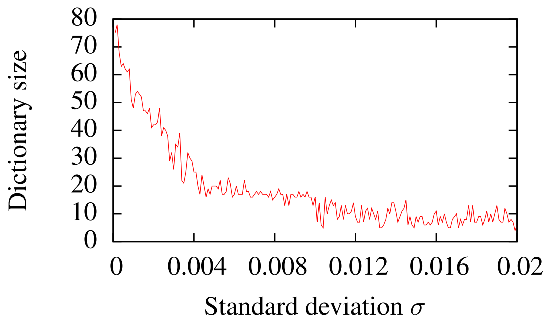
We tested our approach on a small scale indoor scene of a person standing in a corridor (see Figure 6.5(a)). The final dictionary after 3,048 iterations of our algorithm contained only 68 words with a support size of $\rho = 20\,\text{cm}$. The evolution of the BIC for an increasing dictionary size is depicted in Figure 6.5(b). Figure 6.5(c) shows the reconstructed point cloud for a dictionary with only 10 words, whereas Figure 6.5(d) shows the reconstructed point cloud for the final dictionary with 68 words. While flat structures are already well represented with 10 words, round structures are much better approximated with the larger dictionary. Please note that the visual difference between the original scene and the reconstructed scenes mainly results from the different resolution since the reconstructed point cloud contains more points in low density regions of the original cloud. The final reconstructed point cloud covers 88.3% of the scene with 4.2% of outliers, whereas the largest errors occur in the low density regions of the scan far away from the sensor. The resulting size of the binary model file is 199.2 kB. Compared to the original point cloud of size 1,208.9 kB, this means a compression down to 16.5%. Additionally, we tested the influence of a standard compression algorithm, namely gzip [33]. Using gzip on the original point cloud led to a file of size 1,089.9 kB, which is 90.1% of the original file size. Using gzip on the binary files of our model led to a file of size 160.2 kB, which is 80.0% of the original file size. In general, the compression that can be achieved with gzip is roughly between 80% and 90% of the original file size for both, the original point clouds and our models. If we compress both representations with gzip, the relative size difference does only change slightly. For the current dataset it is 14.7% instead of the original 16.5%. Table 6.1 summarizes the results of the experiments.

6.5.3 Dictionary Transfer

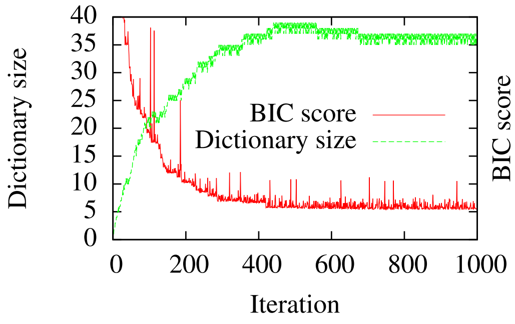
Subsequently we wanted to evaluate, if the dictionary of one scene can be used to reconstruct another similar scene. We used the dictionary from the last experiment (Section 6.5.2) to create a model for the scene shown in Figure 6.6(a), using the method explained in Section 6.4. The resulting model includes 5,263 word instances and the reconstructed point cloud has 89% inliers and 12% outliers (see Figure 6.6(b)). This result is comparable to the experiment above, with mainly an increased outlier rate. Yet the scene is still well represented, leading to the conviction that our approach provides an accurate reconstructions even when the dictionary was learned for another scene.



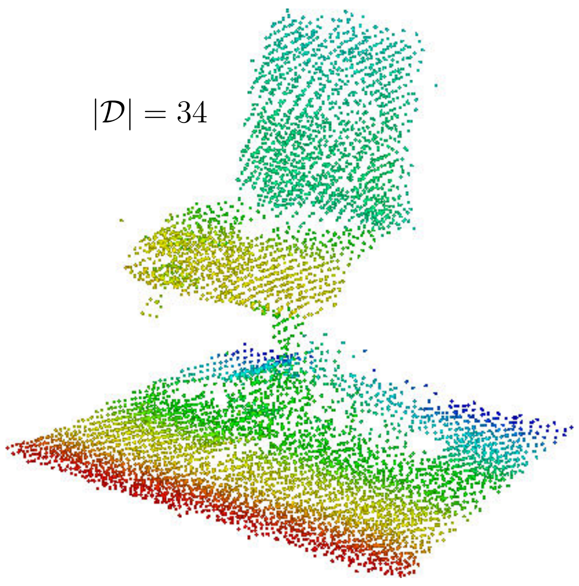
(a)



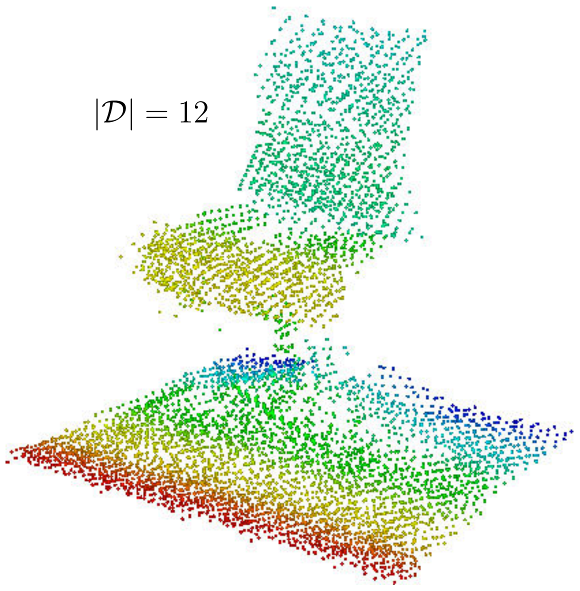
(b)



(c)

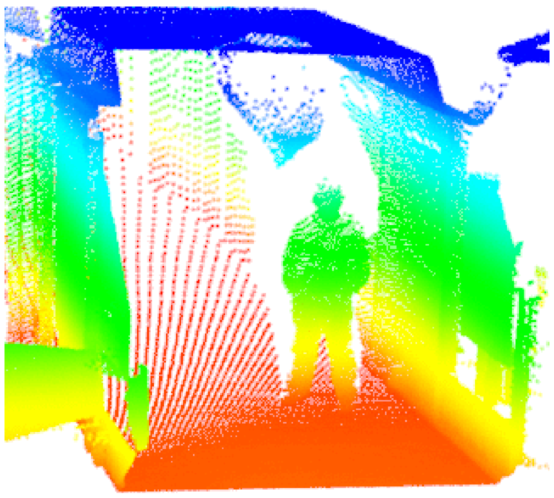


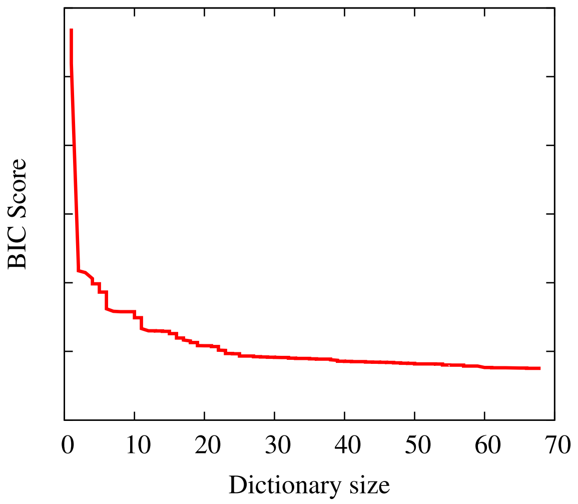
(d)



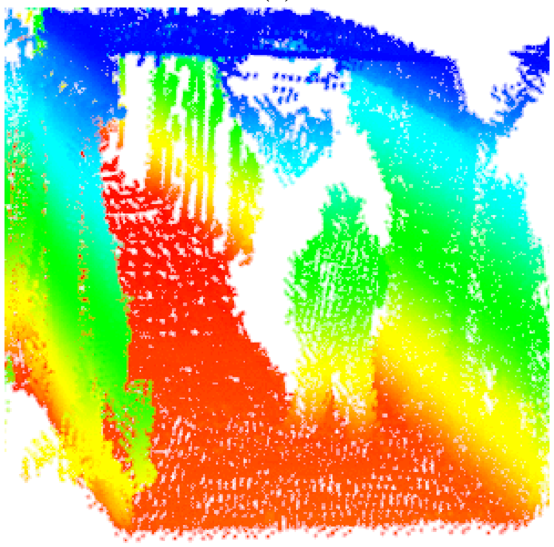
(e)

Figure 6.4: **(a)**: a point cloud of a chair. **(b)**: the evolution of the dictionary size for increasing values of σ. **(c)**: the development of the BIC and dictionary size during the iterative dictionary refinement for $\sigma = 3.2$ mm and $|\mathcal{D}| = 34$ (see (d)). **(d)** Reconstructed point cloud for $\sigma = 3.2$ mm, leading to $|\mathcal{D}| = 34$. **(e)** Reconstructed point cloud for $\sigma = 14.4$ mm, leading to $|\mathcal{D}| = 12$. (Source: [106])


(a)



(b)


(c)

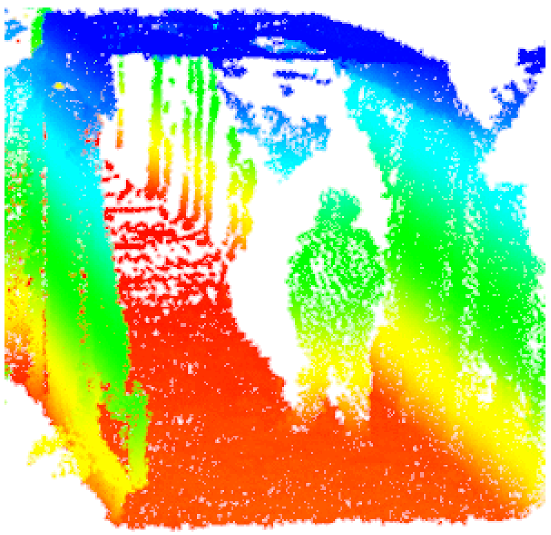

(d)

Figure 6.5: This figure shows the results of an experiment for a 3D scene of a corridor with a person standing in it. **(a)** shows the original point cloud, color coded according to the z-axis for better visibility. **(b)** shows the evolution of the BIC during the iterative refinement. **(c)** shows the point cloud reconstructed with an early dictionary containing only 10 words. **(d)** shows the reconstructed point cloud using the final representation with 68 words in the dictionary. The planar corridor is already well explained with a dictionary of size 10. Yet, round regions (e.g., on the person) or sparse structures (like the lamp above the head of the person) are much better represented with an increasing number of words.

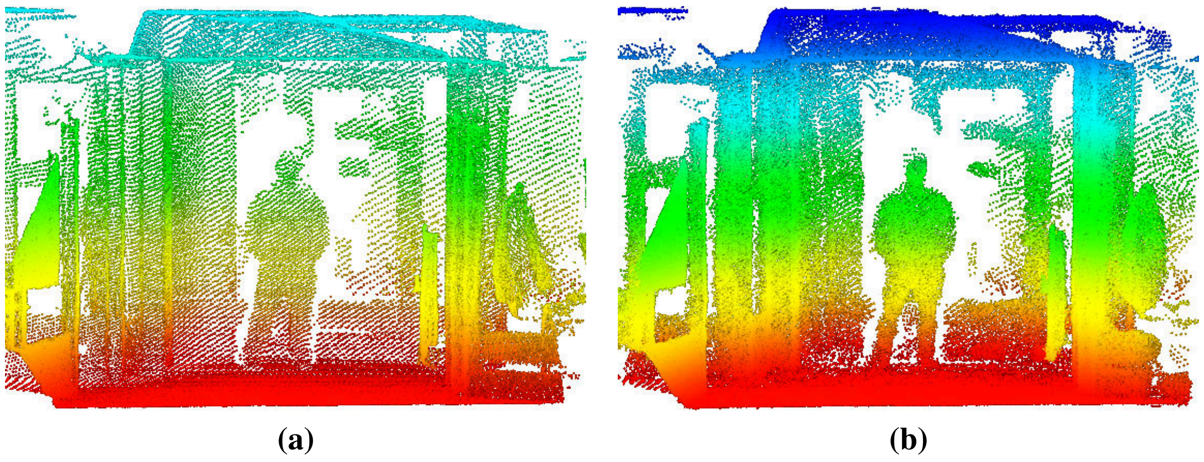


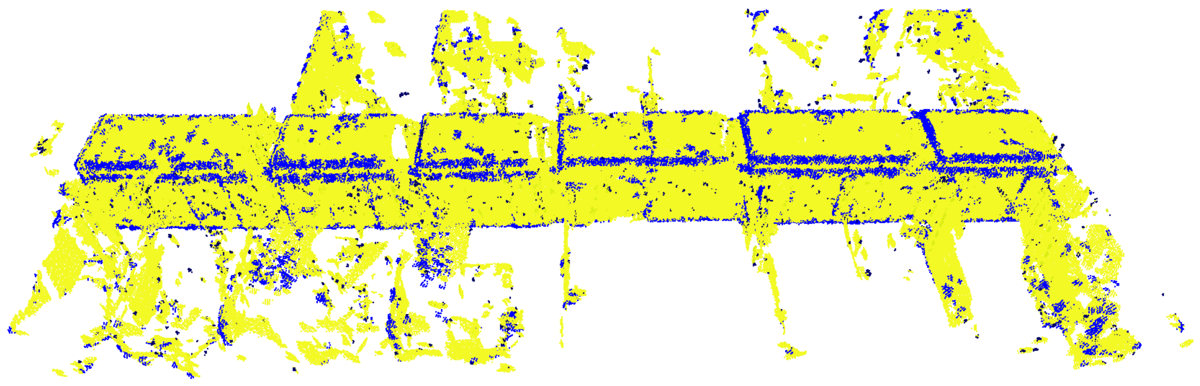
Figure 6.6: **(a)**: Another point cloud of a scene of a corridor with a person. **(b)**: A reconstructed point cloud using the dictionary originally created for the scene in Figure 6.5(a). (Source: [106])

6.5.4 Large Scale Indoor Dataset

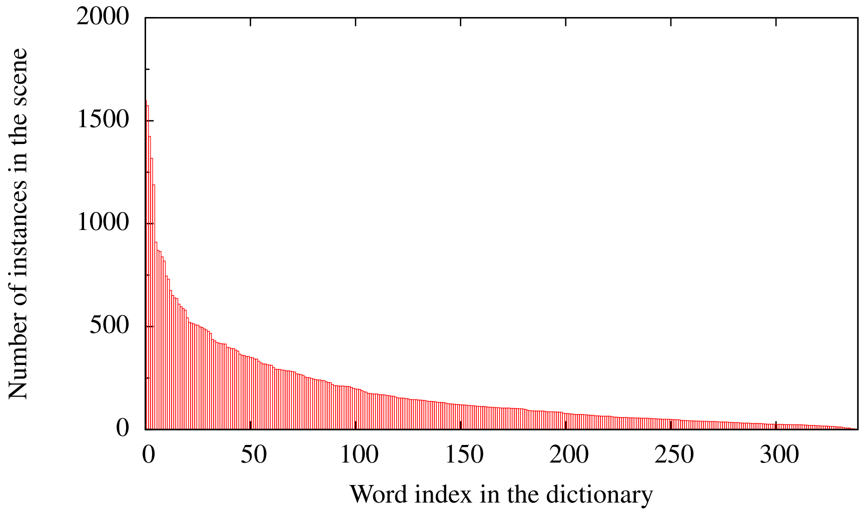
Up to now we showed that our approach can be used to accurately describe small scale 3D scenes with a substantially lower memory footprint than the original scene. It is reasonable to assume that the performance will improve for larger scenes, where the repetitiveness of the environment can be exploited even better. To test this assumption, we captured a dataset of 31 3D range scans in the corridor of an office building and used our approach to build a model of the resulting combined point cloud. The resulting reconstructed 3D map is visualized in Figure 6.7(a). The final dictionary included 339 words. The most frequently used word in the model is a mostly planar patch, which is not surprising because there are many flat surfaces in this environment. The occurrences of this word are colored yellow. The remaining instances are colored blue. The plot in Figure 6.7(b) shows how often the different words from the dictionary occur in the model. The number of occurrences decreases quickly, which confirms the assumption that a small dictionary can already lead to an accurate description of the environment. The model and dictionary together require only 5% of the memory, the original point cloud needed, with 98.7% inliers and only 0.3% outliers. This confirms our assumption that the approach returns even better results for a larger dataset. Figure 6.7(c) and (d) show a close-up view from inside of the corridor, for the original point cloud and the reconstructed version respectively. The structures in the corridor are well represented in the reconstruction. The main visual difference is that the individual scans are visible in the original point cloud, because of the varying density, whereas the density in the reconstructed version is more uniform. Additional results for this experiment can be found in Table 6.1, which summarizes our experiments.

6.5.5 Outdoor Dataset

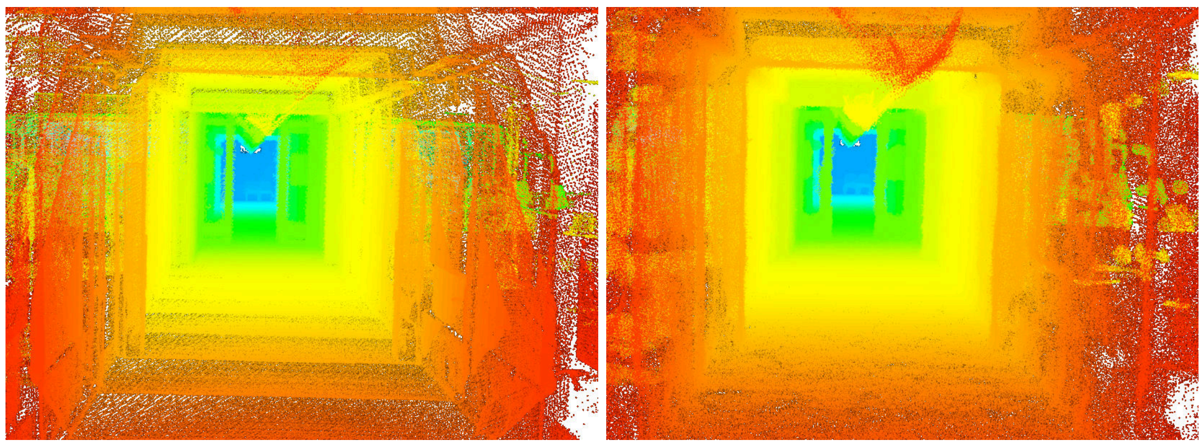
In another experiment we investigated the ability of our system to model a large scale outdoor dataset. For this purpose we collected 3D scans on our campus of the Technical Faculty of the University of Freiburg, containing roads, buildings, trees, and vegetation. We made this dataset publicly available [27]. The reconstructed point cloud is shown in Figure 6.8. For a support size of $\rho = 0.5\,\mathrm{m}$ our system learned a dictionary of 159. The reconstruction quality is very high with 99.5% inliers and only 1.2% outliers. The compression for this case is down to 17.8%. This is higher than what we achieved on the large indoor dataset, since the outdoor data includes more complex structures like trees and natural vegetation. Yet, also in this case



(a)



(b)



(c) **(d)**

Figure 6.7: Experimental results for a dataset of a corridor. **(a)** shows the reconstructed point cloud of the map generated from a corridor dataset. The most common word, which is mostly planar, is colored in yellow, the rest in blue. **(b)** shows how often the different words from the dictionary occur in the model. **(c)** shows a close up view of the inside of the corridor from the original point cloud. **(d)** shows the corresponding reconstructed point cloud. (Source: [106])

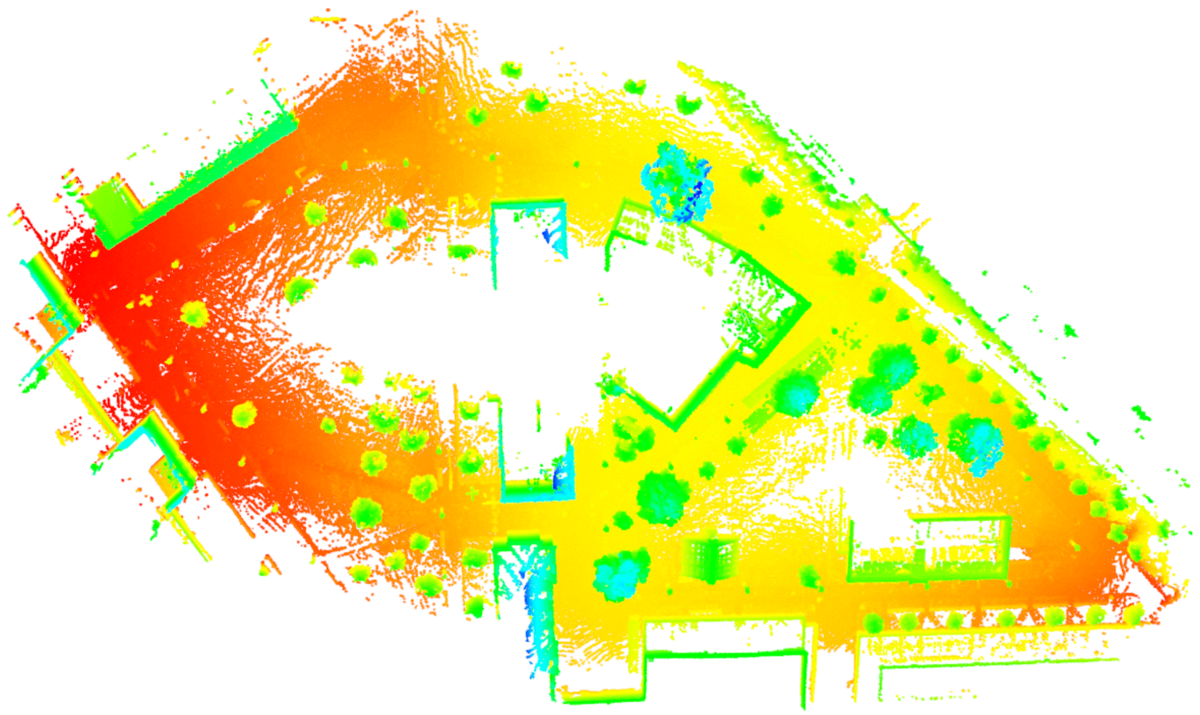


Figure 6.8: Top view of a point cloud reconstructed from a 3D outdoor dataset. The color differs with the height of the points (the z-axis)

our approach was able to accurately reconstruct a complex environment with a small number of surface primitives. Our system took 22.3h to learn the model for this dataset. The high runtime is caused by the large scale of this dataset, containing a large number of instances. As for the other experiments, Table 6.1 summarizes the results of this experiment.

Our results suggest that typical real world scenes can be accurately described by a small number of reoccurring words. The runtime requirements of our dictionary creation method are currently too high to consider them for online applications. Yet, a previously learned dictionary could be used for such applications.

6.6 Related Work

The compression of data is an important to save costs for storage and bandwidth. There are general lossless data compression methods like the LZW-algorithm [93] or methods specifically designed for a certain kind of data, like the JPEG [148] standard for visual images, both for lossless and lossy compression. There is also a lot of activity in the field of compression of mesh-based 3D models [1].

Spinello *et al.* [121] presented an unsupervised approach to discover repetitive object patterns in single camera images, which is based on a voting scheme regarding image descriptors. These patterns can then be used for model prediction and completion or for model compression. This is very similar to our concept but relies on camera images, not on 3D range data.

There are many different ways to represent 3D data with different advantages and disadvantages. Range Images and point clouds (see Section 2.3) are frequently used in this work. They are representations that are close to the original sensor data, but have high memory requirements. *Voxel grids* are another possible structure to represent 3D data. Voxel stands for volumetric pixel and those grids basically divide the 3D space in equidistant cells where each

Dataset	Single scene (see Figure 6.5)	Corridor (see Figure 6.7)	360° outdoor (see Figure 6.8)
dictionary size $\lvert\mathcal{D}\rvert$	68	339	159
support size ρ	0.2 m	0.3 m	0.5 m
word occurrences $\lvert\mathcal{M}\rvert$	6,599	62,792	873,464
memory req. original point cloud without and with gzip	1.21 MB 1.09 MB	37.77 MB 33.99 MB	137.47 MB 122.11 MB
memory requirement our model without and with gzip	0.20 MB 0.16 MB	1.85 MB 1.67 MB	24.47 MB 21.25 MB
relative memory requirement without and with gzip	16.5% 14.7%	4.9% 4.9%	17.8% 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 6.1: Summary of experimental results. The experiments were performed on an Intel I7 quad core machine.

cell stores information about the covered space, e.g., if it is empty or occupied, either in a probabilistic or a boolean fashion. While queries regarding the values of cells are very efficient, namely $O(1)$, the memory requirements are cubic regarding the size of the environment one wants to represent.

Octree-based structures [152] represent the world as hierarchical three-dimensional grids and are popular for 3D mapping approaches because they efficiently represent filled and empty space and can also be used in a probabilistic fashion. Yet they are actually better suited to represent continuous volumes than surfaces, as they are measured by typical range sensors.

Another popular method to represent 3D environments are *surface maps* [7, 57, 75, 98], also called or *height maps* or *elevation maps*. These maps are two-dimensional grids, where every cell stores a height value. These maps are efficient regarding their memory footprint, as long as the resolution is low. Yet they cannot represent arbitrary 3D structures, mainly regarding multiple layers in one cell. To overcome this restriction, Triebel *et al.* developed the *multi-level surface (MLS) maps* [141] where every cell can save information about multiple layers. Unfortunately, the memory requirements still depend on the size of the environment and the resolution of the grid, independent of the actually contained 3D structures.

Puente *et al.* [31] uses planar patches to build approximate 3D maps. While plane extraction can be performed efficiently and flat structures are often dominant in man-made environments, this approach is very restricted in what it can represent.

6.7 Conclusions

In this chapter, we presented a novel method to model 3D environments by means of surface primitives, which are used to describe reoccurring structures. We thereby minimize redundant information storage, which leads to a substantial reduction of the memory requirement. We use the Bayesian Information Criterion (BIC) as a decision criterion in the creation of the dictionary and the model to weight the memory requirements against the accuracy of the reconstruction. We presented experiments that suggest that even a small dictionary learned from the original data can be used to accurately describe complex scenes. The use of our NARF descriptors and an adapted version of the NARF keypoint extraction made an efficient implementation of similarity checks between dictionary words and areas in the scene possible. Our method to represent 3D data does not only have lower memory requirements, it is also very convenient for further processing in approaches that require frequent similarity tests. Since information about similar structures in the environment is explicitly encoded in our representation, the need for preprocessing would be substantially reduced.

Chapter 7

Place Recognition based on 3D Range Scans

Place recognition is one of the fundamental tasks in mobile robotics. It is the ability to recognize previously seen parts of the environment based on the current observation and a database of previous observations. The wide range of applications of place recognition includes localization (determine the initial pose), SLAM (detect loop closures), and change detection in dynamic environments. In the past, relatively little work has been carried out to address this problem using 3D range data and the majority of approaches focuses on detecting similar structures without estimating relative poses. In this chapter, we present an algorithm based on 3D range data that is able to reliably detect previously seen parts of the environment and at the same time calculates an accurate transformation between the corresponding scan pairs. We present an extensive set of experiments using publicly available datasets in which we compare our system to other state-of-the-art approaches.

*   *   *   *

Place recognition describes the ability to detect that a robot revisited an already known area. It is a crucial part in key navigation tasks including localization and SLAM. The majority of state-of-the-art place recognition techniques have been developed for vision or two dimensional range data. Relatively few approaches work on three-dimensional laser range scans and can efficiently calculate the similarity and the relative transformation between two scans.

In the previous chapters we presented different systems, where finding correspondences between small parts of 3D scenes played an important role. The scale of the structures our approaches were matching decreased steadily, from complete objects to partial objects, to surface patches. In this chapter we want to match structures on a larger scale, basically determining if two complete scans show the same scene.

The contribution of this chapter is a novel approach to place recognition using 3D range data. Our approach transforms a given 3D range scan into a range image and uses a combination of a Bag-of-Words (BoW) approach and a point-feature-based estimation of relative poses that are then individually scored. Our approach provides high recall rates, outperforming other state-of-the-art approaches, and calculates accurate relative poses between corresponding scans. Figure 7.1 shows an example application. It visualizes how the calculated relative transformations between scans can be used as edges (loop closures) in a pose graph. This enables us to apply our approach as a front-end for a graph-based SLAM system (see Section 2.4.1). We tested our approach on different kinds of platforms: ground robots and flying vehicles. We used

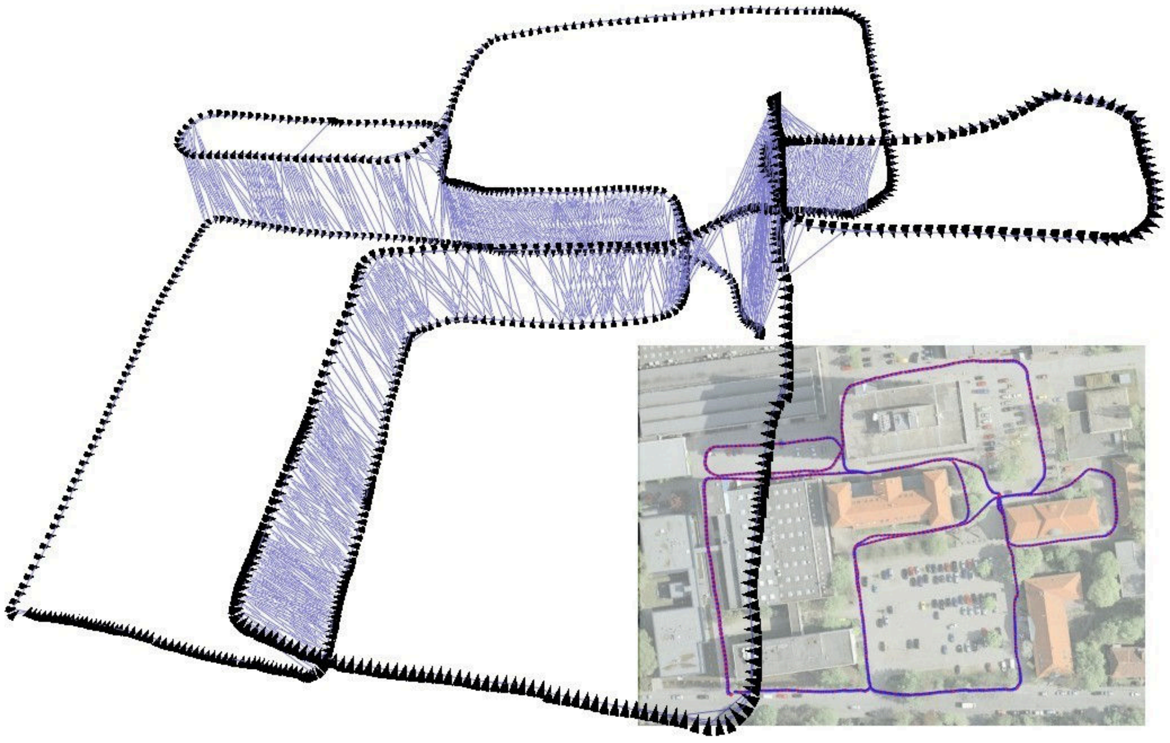


Figure 7.1: Scan correspondences found by our place recognition system for the Hanover2 dataset (see Section 7.2.1 for details on the dataset). The image shows the graph of the trajectory (black nodes) and the found loop closures between the scans (blue/gray lines). The z-axis of the trajectory represents the scan index to make the loop closures better visible. The image in the bottom right shows an aerial image from Google Earth (©Google), overlain with the trajectory of the dataset.

a combination of publicly available datasets and datasets we captured with our own robots. For the sake of repeatability we made all those datasets publicly available.

The remainder of this chapter is organized as follows: We will present our system in Section 7.1, whereas we first give an overview over the algorithm in Section 7.1.1, followed by more details on the individual components. Our experimental evaluation will be discussed in Section 7.2, followed by related work in Section 7.3. Section 7.4 concludes this chapter.

7.1 Our Approach

We will now present our place recognition system. First in a rough overview, which presents the idea of our algorithm, followed by a more detailed explanation.

7.1.1 Overview

Given a database of 3D scans and a scan as input query, our algorithm returns a set of scans which are potential matches with the input. Additionally, it calculates a transformation and a score for every returned scan, reflecting how certain the system is that the scans actually match.

More formally, let $\mathcal{Z}$ denote the database of 3D range measurements and $\mathbf{z}_{\text{query}}$ a query scan. The goal of our approach is to calculate $\mathcal{B}(\mathbf{z}_{\text{query}}) = (\langle \mathbf{z}_1, T_1, s_1 \rangle, \ldots, \langle \mathbf{z}_n, T_n, s_n \rangle)$, which is a set of candidate matches. Here, $\mathbf{z}_i \in \mathcal{Z}, i \in \{1, \ldots, n\}$ are the potential measurement candidates from the database which are similar to the current query $\mathbf{z}_{\text{query}}$, T_i denotes the estimated

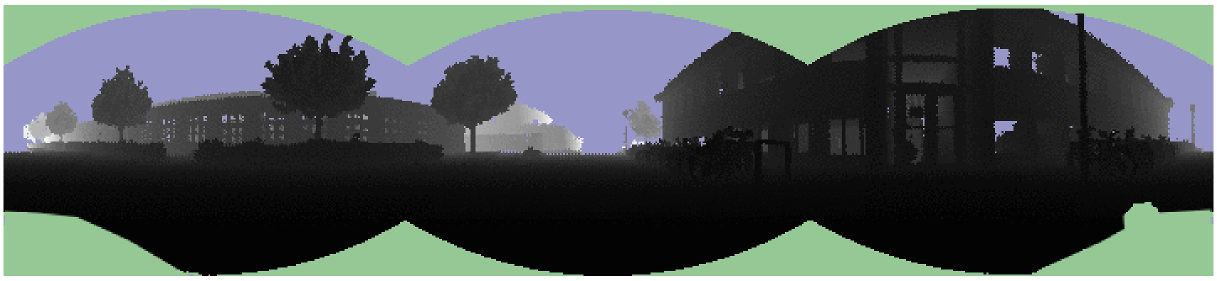


Figure 7.2: Example range image from the FreiburgCampus360_3D dataset (see Section 7.2.1 for details on the dataset). The image pixel positions represent the spherical coordinates of the points. The gray values represent the measured ranges. Blue points are maximum range readings and green points are unknown space.

transformation from $\mathbf{z}_{\text{query}}$ to $\mathbf{z}_i$, and s_i is a score reflecting the confidence about the match. Our algorithm for calculating $\mathcal{B}(\mathbf{z}_{\text{query}})$ mainly consists of the following four steps.

1. Given a database of 3D range measurements $\mathcal{Z}'$ (training set), calculate a set of features from the 3D scans and build a dictionary for a Bag-of-Words approach.

2. Use the Bag-of-Words approach to get an initial similarity measure for all scans in the database $\mathcal{Z}$ with respect to the query scan $\mathbf{z}_{\text{query}}$. This measure is then used to order the database scans from most similar to least similar. Let the resulting ordered set be $\bar{\mathcal{Z}}(\mathbf{z}_{\text{query}}) = \langle \bar{\mathbf{z}}_1, \dots, \bar{\mathbf{z}}_{|\mathcal{Z}|} \rangle$.

3. For each pair $\langle \mathbf{z}_{\text{query}}, \bar{\mathbf{z}}_k \rangle, \bar{\mathbf{z}}_k \in \bar{\mathcal{Z}}(\mathbf{z}_{\text{query}})$, starting with $k = 1$, calculate a set of possible transformations between $\mathbf{z}_{\text{query}}$ and $\bar{\mathbf{z}}_k$ by matching point features of the corresponding scans. Note that this set of features is not the same as the one used for the Bag-of-words approach since the parameters for the feature extraction differ.

4. Score each of the possible transformations and get the transformation T_k with the highest score s_k. If this score is above an acceptance threshold then $\langle \bar{\mathbf{z}}_k, T_k, s_k \rangle$ is a candidate for a recognized place, i.e., it is added to $\mathcal{B}(\mathbf{z}_{\text{query}})$.

The last two steps are repeated until a timeout occurs or $k = |\mathcal{Z}|$. Note that if there are no time constraints, the first two steps can be skipped so that all scans in $\mathcal{Z}$ are checked.

Although we work with a database of 3D range scans, we do not use this data directly. We rather represent each three-dimensional range scan by its dual, namely a range image (see Figure 7.2). If the 3D scan is captured from one point in space, i.e., the sensor does not move while the 3D points are generated, the range image contains the same information as the scan. The advantage of the range image is that it allows us to model unknown areas as well as maximum range readings more efficiently. See Section 2.3.2 for more details on range images. In the following sections, we will describe the individual components of our approach in more detail.

7.1.2 Feature Extraction

Our approach uses the NARFs (Normal Aligned Radial Features) that we presented in Chapter 3. These point features are used to build a dictionary for the Bag-of-Words approach and also to find corresponding regions between two 3D measurements. Since each NARF encodes a full 3D transformation, a single feature correspondence between two scans is already sufficient to calculate a relative transformation.

There are three parameters needed for the extraction of NARFs. First, the size of the feature descriptor, second the maximum number of calculated features, and finally the support size, which is the size of the area around the feature point that is used to calculate the descriptor. We chose 36 as the descriptor size. For the Bag-of-Words approach, a high number of features describing small parts of the environment is most useful. Therefore we extract 2000 features with a support size of $1/10$ of the average range in the database $\mathcal{Z}$. However, when matching a new query $\mathbf{z}_{\text{query}}$ against $\mathcal{Z}$, a smaller number of more distinctive features is needed. Here, we extract 200 features with a support size of $1/5$ of the average range in $\mathcal{Z}$. Intuitively, a small support size makes the features susceptible to noise and less distinctive, whereas a large support size makes them more expensive to compute and less robust to partial occlusion and missing data. However, we found the values above to provide reasonable trade-offs between those properties. The descriptors can be compared using standard norms like the Manhattan distance. The resulting measure (the *descriptor distance*) describes the similarity between the described regions. Here, a high value reflects a low similarity. Furthermore, NARFs can either be used in a rotationally invariant version or without invariance regarding the rotation around the normal. For example, in the rotationally variant case the features distinguish between a top right corner and a top left corner of a square, whereas they do not in the rotationally invariant case. This is a useful distinction for place recognition purposes because wheeled robots capturing 3D scans often move with very little change in their roll and pitch angle. Accordingly, they do not need the rotational invariance around the normal vector for the features. The same is the case if the robot is equipped with an IMU. This can reduce the computational complexity of the problem since the feature matching is more robust with one degree of freedom less.

7.1.3 The Bag of Words Approach

We employ a Bag-of-Words approach as a fast initial method to pre-order the scans according to their similarity to the given query scan $\mathbf{z}_{\text{query}}$. Bag-of-Words approaches are based on the idea that similar structures in an environment will create similar distributions of features. The goal is to obtain a general representation for those feature distributions. We want to encode each scan in terms of a small set of words (the *dictionary*). To learn this set, we use a training database $\mathcal{Z}'$ of 3D scans and calculate 2000 NARFs for each scan. This leads to $|\mathcal{Z}'|\ 2000$ feature descriptors (each of size 36). We then apply k-means clustering to obtain a total of 200 clusters. Our dictionary is now made of 200 words, each being the averaged descriptor of its cluster. We found that this size provides a reasonable trade-off between being able to generalize and being descriptive enough. Given this dictionary, we can now express each scan $\mathbf{z}_i \in \mathcal{Z}$ in terms of the words of the dictionary by selecting the closest word for every feature descriptor (regarding the Euclidean distance). For each $\mathbf{z}_i$, we obtain a histogram H_i having 200 bins. The number in each bin reflects how often the corresponding word is present in $\mathbf{z}_i$. Given the histogram of the query scan H_{query} (obtained in the same way), we calculate $\|H_{\text{query}} - H_i\|_2$ as the distance between the histograms. This distance is then used as an initial similarity measure to create the ordered set $\bar{\mathcal{Z}}(\mathbf{z}_{\text{query}})$, as described in Section 7.1.1.

7.1.4 Determining Candidate Transformations

Each NARF encodes a full 3D transformation. Therefore, the knowledge about a single feature correspondence between two scans enables us to retrieve all six degrees of freedom of the relative transformation between them by calculating the difference between the two encoded transformations. To obtain the candidate transformations for each scan pair, we order the feature pairs according to increasing descriptor distance (see Section 7.1.2) and evaluate the transformations in this order. In other words, we calculate a score for each of these transformations (see Section 7.1.5). In our experiments we stop after a maximum number of 2000 evaluated transformations when using the rotationally variant version of the NARFs. In the rotationally invariant case we evaluate up to 5000 transformations due to the increased search space size introduced by the additional degree of freedom.

7.1.5 Scoring of Candidate Transformations

The result of the feature matching is a list of relative poses $\bar{T}_k = \{\bar{T}_{k_1}, \ldots, \bar{T}_{k_n}\}$ for the candidate pair $\langle \mathbf{z}_{\text{query}}, \bar{\mathbf{z}}_k \rangle, \bar{\mathbf{z}}_k \in \bar{\mathcal{Z}}(\mathbf{z}_{\text{query}})$. Our goal is to evaluate those candidate transformations and calculate a score/likelihood for each $\bar{T} \in \bar{T}_k$ reflecting the confidence of the transformation given a model of our sensor. Recall that we use 3D range data, i.e., each measurement $\mathbf{z}$ is a set of 3D points. This enables us to evaluate the candidate transformation $\bar{T}$ on a point-by-point basis by assuming that the points are mutually independent.

The method for the evaluation of candidate poses we describe below is based on the same basic principles as the method we presented for the same purpose in the context of object recognition in Section 4.4.1. Yet, we had to adapt the scoring method to the matching of two individual range scans instead of the matching of a known point cloud object model against a range scan.

Let $\mathcal{V}$ be the set of *validation points* from the query scan $\mathbf{z}_{\text{query}}$. This set $\mathcal{V}$ could contain all points from $\mathbf{z}_{\text{query}}$ but we will only use a representative subset of $\mathbf{z}_{\text{query}}$ as described in Section 7.1.6. Given a candidate transformation $\bar{T} \in \bar{T}_k$, we first transform each $\mathbf{p} \in \mathcal{V}$ in the reference frame of $\mathbf{z}_{\text{query}}$ into a point $\mathbf{p}'$ in the reference frame of $\bar{\mathbf{z}}_k$. Since we represent our scans as range images, we can calculate the pixel position $\mathbf{p}'(x, y)$ in the range image of $\bar{\mathbf{z}}_k$ in which the point $\mathbf{p}'$ would fall into as well as the range value r' the point should have. Let $\mathbf{p}_k(x, y) \in \bar{\mathbf{z}}_k$ be the point that is already at this pixel position (in the range image of $\bar{\mathbf{z}}_k$) having the range value $r_k(x, y)$. For each $\mathbf{p} \in \mathcal{V}$ we will now calculate a score $s_{\bar{T}}(\mathbf{p}) \in [0, 1]$ and a weight $w_{\bar{T}}(\mathbf{p}) > 0$ reflecting how good the prediction r' is explained by the observation $r_k(x, y)$. The point scores will then be used to calculate the overall likelihood $s(\bar{T})$ for the transformation $\bar{T}$ as

$$s(\bar{T}) = \frac{\sum\limits_{\forall \mathbf{p} \in \mathcal{V}} w_{\bar{T}}(\mathbf{p})\; s_{\bar{T}}(\mathbf{p})}{\sum\limits_{\forall \mathbf{p} \in \mathcal{V}} w_{\bar{T}}(\mathbf{p})}. \tag{7.1}$$

Let $\Delta r = r_k(x, y) - r'$ be the difference between the observed and the predicted range. To evaluate Δr, we have to consider the model of the sensor. In the case of a laser scanner, a pulse of light moves from the origin of the sensor along a line to the measured point. Each range image pixel represents one such beam. There are several different cases that need to be considered regarding the interpretation of Δr:

1. The observation is within a confidence interval $\Delta r_{\max} > 0$ of the prediction, i.e., $|\Delta r| < \Delta r_{\max}$. In other words, what we expected to see mostly fits with what we measured. In this case, we calculate the score as $s_{\bar{T}}(\mathbf{p}) = 1 - \frac{|\Delta r|}{\Delta r_{\max}}$ and weight it by $w_{\bar{T}}(\mathbf{p}) = 1$, which represents a triangular distribution. While a Gaussian would be a more realistic representation, we chose a triangular distribution because it is less expensive to compute. All the other cases will receive a score $s_{\bar{T}}(\mathbf{p}) = 0$. Thus, the associated weight $w_{\bar{T}}(\mathbf{p})$ reflects the confidence about how *wrong* the transformation $\bar{T}$ is.

2. The observed range is larger than the predicted one, i.e., $\Delta r > \Delta r_{\max}$. This means that we actually observed something behind $\mathbf{p}'$ and basically looked through it. This could be the evidence for a dynamic or partially transparent obstacle, but in general it is a strong indicator for a wrong transformation. We therefore penalize the overall likelihood by a high weight $w_T(\mathbf{p}) = w_{\text{seeThrough}} \geq 1$. We used $w_{\text{seeThrough}} = 25$ in our experiments.

3. The observed range is smaller than the predicted range, i.e., $\Delta r < -\Delta r_{\max}$. In this case, there are two more situations to distinguish:

   (a) $\bar{T}^{-1} \cdot \mathbf{p}_k(x, y)$ exists in $\mathbf{z}_{\text{query}}$. This means that we could not see $\mathbf{p}'$ in $\bar{\mathbf{z}}_k$ because of an already known obstacle. In this case we give a low weight $w_{\bar{T}}(\mathbf{p}) = w_{\text{knownObstacle}} \leq 1$ in order to enable us to receive relatively high scores even if the overlap between scans is low. We used $w_{\text{knownObstacle}} = 0.5$ in our experiments.

   (b) $\bar{T}^{-1} \cdot \mathbf{p}_k(x, y)$ does not exist in $\mathbf{z}_{\text{query}}$. This could be evidence for a formerly unseen or dynamic obstacle, but in general it is a strong indicator for a wrong transformation. Similar to case 2, we penalize this by a high weight $w_{\bar{T}}(\mathbf{p}) = w_{\text{unknownObstacle}} \geq 1$. We used $w_{\text{unknownObstacle}} = 15$ in our experiments.

4. $\mathbf{p}_k(x, y)$ is an unobserved point in the range image of $\bar{\mathbf{z}}_k$. This means that $\mathbf{p}'$ could not be observed because it is outside of the scan. We treat this the same as 3a.

5. $r_k(x, y)$ is a far range reading (i.e., exceeding the max range of the sensor) in the range image of $\bar{\mathbf{z}}_k$. There are two more situations to distinguish, for which we need to consider the original range r of $\mathbf{p}$ in $\mathbf{z}_{\text{query}}$:

   (a) The point should actually be closer to the sensor in $\bar{\mathbf{z}}_k$, i.e., $r' \leq r$. In this case it is improbable that $\mathbf{p}'$ is out of range and therefore we treat this the same as case 2.

   (b) The point moved further away from the sensor in $\bar{\mathbf{z}}_k$, i.e., $r' > r$. In this case it is possible that $\mathbf{p}$ moved out of range and we give a medium high weight $w_{\bar{T}}(\mathbf{p}) = w_{\text{farRange}} \geq 1$. We used $w_{\text{farRange}} = 5$ in our experiments.

To avoid that slight errors in the estimate of a correct transformation lead to a very small score, e.g., if the point lies on an obstacle boundary and we hit the much further away neighbor instead, we actually consider not only $\mathbf{p}_k(x, y)$ as a correspondence for $\mathbf{p}'$, but also its neighbors in a small pixel radius $e \in \mathbb{N}$ (3 in our experiments) around it and select the point with the least negative influence on the complete score.

Please note that while most of the above rules are valid for all major range sensing devices (3D laser scanners, stereo cameras, ToF-cameras, cameras with texture projectors, ...), the used parameter values are tuned for laser scanners and especially the handling of unknown points (case 4) and far ranges (case 5) can differ significantly for other devices.

The values we chose are strict regarding any indication that our transformation might be false. This is because in general, false negatives are far less hurtful (considering, e.g., a SLAM system) than false positives. To allow more robustness, e.g., to dynamic obstacles, one can reduce the weights $w_{\text{seeThrough}}$, $w_{\text{unknownObstacle}}$, and $w_T(\mathbf{p}) = w_{\text{farRange}}$.

7.1.6 The Set of Validation Points

Up to now we did not say, how the set of validation points $\mathcal{V}$, from which we select $\mathbf{p}$, is obtained. In principle it could contain all the points from $\mathbf{z}_{\text{query}}$. However, this would lead to a high number of points to be tested and thus would be computationally expensive. We therefore use only a subset of $\mathbf{z}_{\text{query}}$. A random subset of a fixed size could be used, but it is better to select points that have some significance in the scene, or two scans could get a high score, just because the floor or a big wall is well aligned. Furthermore, the points should be evenly distributed over the scan in 3D space to be invariant regarding the non-uniform resolution of 3D scans. To achieve this, we use the set of keypoints $\bar{\mathcal{V}}$ (i.e., the points where the NARFs were extracted) that we calculated in the feature extraction process as a base to create the set of validation points. We add a random point from $\bar{\mathcal{V}}$ to $\mathcal{V}$ and then iteratively add the point $\bar{\mathbf{p}}_i \in \bar{\mathcal{V}}$ that has the highest 3D distance to all points already in $\mathcal{V}$, until a maximum size is reached (200 points in our current implementation). This has the interesting property that each ordered subset $\langle \mathbf{p}_0, \ldots, \mathbf{p}_j \rangle$ of the ordered set $\mathcal{V} = \langle \mathbf{p}_0, \ldots, \mathbf{p}_{|\mathcal{V}|} \rangle$ is a subsampled version of $\bar{\mathcal{V}}$ with mostly equidistant points for every j. This also means that one can stop the calculation of $s(\bar{T})$ (see Eq. 7.1) before handling each point in $\mathcal{V}$ if the score is already to low after a certain minimum of handled points (30 points in our experiments) since this subset already represents the whole set quite well. The score $s(\bar{T})$ for the transformation $\bar{T}$ is not necessarily the same as for $\bar{T}^{-1}$ (by switching the role of $\mathbf{z}_{\text{query}}$ with $\bar{\mathbf{z}}_k$). We therefore adapt the scoring to

$$s'(\bar{T}) = \min(s(\bar{T}), s(\bar{T}^{-1})), \tag{7.2}$$

as the score for the pair $\langle \mathbf{z}_{\text{query}}, \bar{\mathbf{z}}_k \rangle$ with transformation $\bar{T}$.

7.1.7 Self-Similarity

There are scans that qualify only poorly for the pose estimation process because of a high self-similarity, e.g., corridors with very few distinctive structure. For areas like these it is hard to determine a correct relative transformation since different positions in the vicinity (e.g., along the corridor) lead to very similar observations. To prevent false positives (false transformations getting a high score) in those areas, we calculate a self-similarity value for every scan. We do this by matching the scan $\mathbf{z}$ against itself, using the procedure described above and consider only transformations that are not close to the identity matrix. We call the highest score in this set $\mathit{self}(\mathbf{z})$ and consider it as a measure for self similarity because it represents the definition of self similarity above: the position in the scan itself cannot be clearly distinguished because close by positions lead to similar measurements. We now use the self similarity value $\mathit{self}(\mathbf{z})$ to adapt the scoring and obtain the final score for a transformation between $\mathbf{z}_{\text{query}}$ and $\bar{\mathbf{z}}_k$ as follows:

$$s^*(\bar{T}) = (1 - (\mathit{self}(\mathbf{z}_{\text{query}}) + \mathit{self}(\bar{\mathbf{z}}_k))/2)\ s'(\bar{T}). \tag{7.3}$$

Recall that we perform the steps described so far for each candidate transformation $\bar{T} \in \bar{T}_k$. If the best score out of all candidates is above a threshold, $\bar{\mathbf{z}}_k$ represents a potential loop closure, i.e.,

$$\mathcal{B}(\mathbf{z}_{\text{query}}) := \mathcal{B}(\mathbf{z}_{\text{query}}) \cup \langle \bar{\mathbf{z}}_k, T_k, s_k \rangle, \tag{7.4}$$

with $T_k = \operatorname{argmax}_{\bar{T} \in \bar{T}_k} s^*(\bar{T})$ and $s_k = s^*(T_k)$.

7.1.8 Implementation Details

We perform some additional steps to improve the results. After an initial scoring of the candidate transformations for a scan pair $\langle \mathbf{z}_{\text{query}}, \bar{\mathbf{z}}_k \rangle$ we first remove transformations with a very low score. We then cluster the transformations and identify those describing very similar relative poses, keeping only the best ones in the candidate list. Next, we perform ICP to improve the transformation estimate, using only the set of validation points to speed up this step. Finally, we update the scores given the corrected transformations and return the transformation associated with the highest score as the result.

7.2 Experimental Evaluation

We will now present the real world experiments carried out to evaluate our approach.

7.2.1 Datasets

We used four publicly available datasets of 3D scans for our experiments, namely two outdoor datasets and two indoor datasets. In the following we will give an overview over these datasets and their specific challenges.

The following datasets were used in our experiments:

- For the first indoor dataset we chose **AASS-loop**[1] [26]. This dataset was also used in the related work [48, 85], which makes a comparison easier. Its main challenge is that it contains some highly ambiguous areas in long corridors.

- For the second indoor dataset we captured 3D scans with a flying **Quadrotor** robot, equipped with a 2D laser scanner [54, 53]. This dataset [29] is challenging because of a higher noise level and the existence of highly similar scans from different poses in a corridor environment.

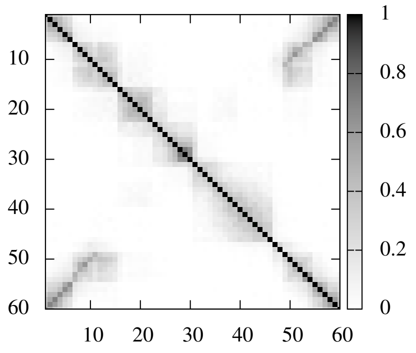
- For the first outdoor dataset we chose **FreiburgCampus360_3D** [27]. It contains high resolution 360° scans and its main challenge is the large distance between consecutive scans, stressing the system's ability for translational invariance.

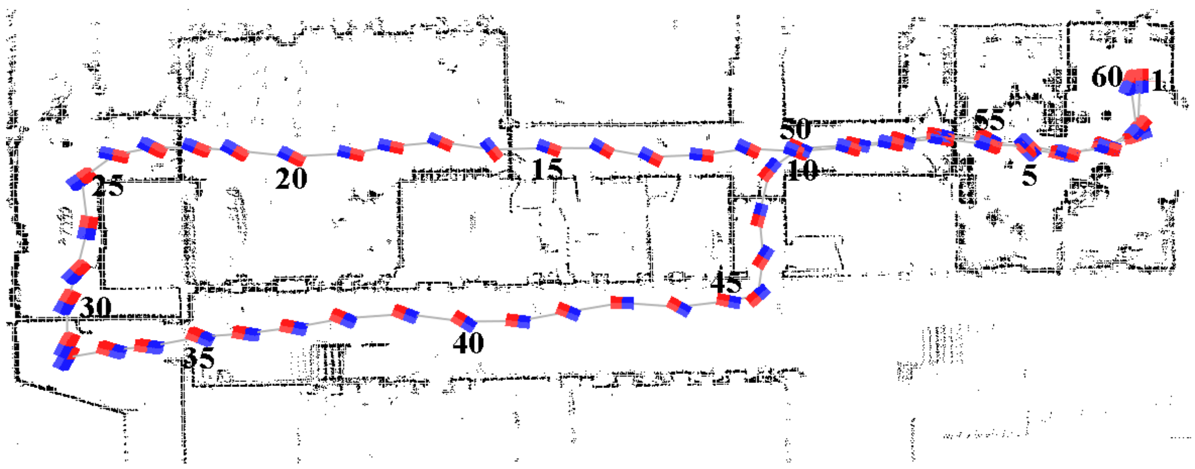
- For the second outdoor dataset we chose **Hanover2**[2] [28]. This dataset was also used in related work [48, 85], which makes a comparison easier. This is a challenging dataset because it contains a high number of very sparse scans and the robot traverses different areas with very similar structure.

All datasets apart from the Quadrotor dataset were recorded with 2D laser scanners mounted on pan/tilt units. We acquired SLAM trajectories using the provided odometry and manually verified scan matching as edges in a graph-based SLAM system (see Section 2.4.1), which we solved using g^2o [70]. These trajectories were used to evaluate false/true positives and false/true negatives in our system. In the Quadrotor dataset the helicopter occasionally captured a 3D scan by flying downwards and upwards again while hovering around the same spot. The trajectory was estimated using the navigation system of the quadrotor as described by Grzonka *et al.* [54, 53]. Please refer to Figure 7.3 and Figure 7.4 for more information about the datasets.

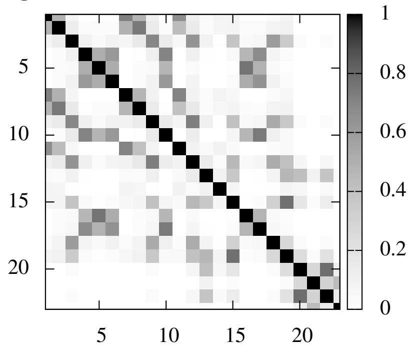
[1]Courtesy of Martin Magnusson, AASS, Örebro University, Sweden
[2]Courtesy of Oliver Wulf, Leibniz University, Germany

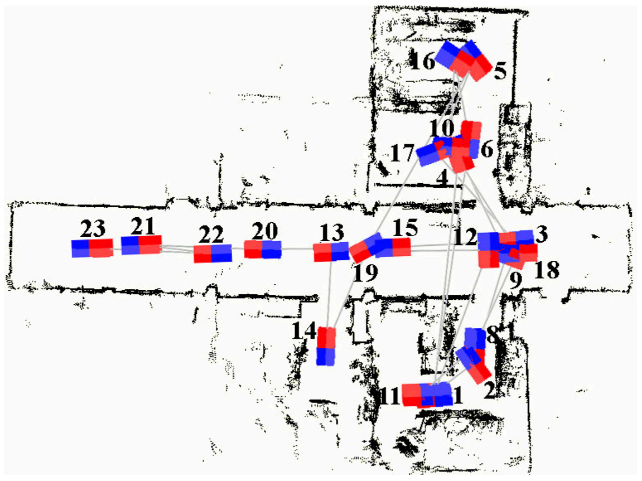
AASS-loop





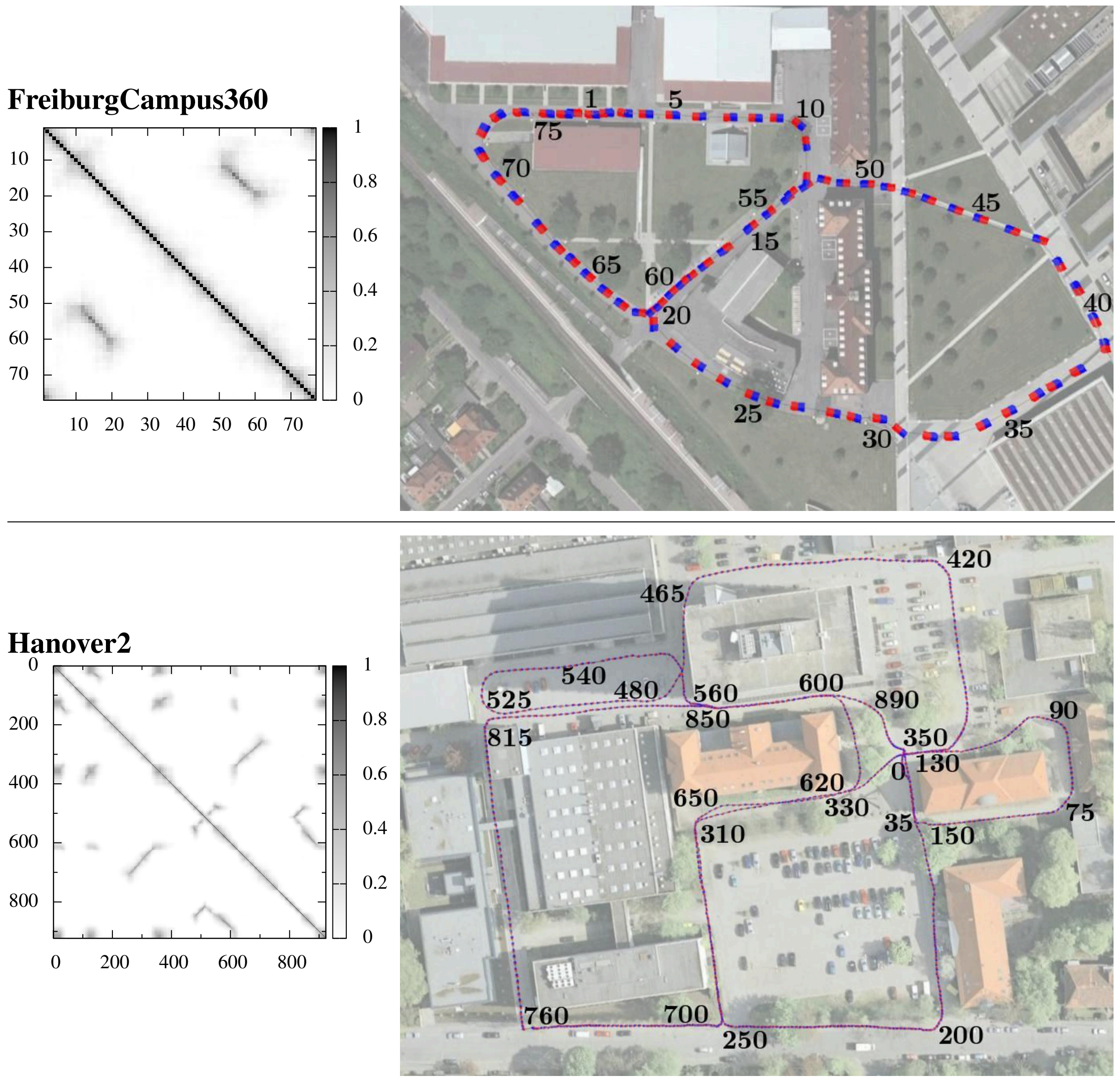
Quadrotor





	stop and go	# scans	# points	#far ranges	res. α	traj. length	dist.	mean range	max range
AASS-loop	yes	60	80873	n/a	0.7°	111.4 m	1.89 m	2.81 m	67.1 m
Quadrotor	no	23	171608	95515	1.0°	79.1 m	3.6 m	1.9 m	6.1 m

Figure 7.3: Top: SLAM trajectories and ground truth confusion matrices for the used indoor datasets. The trajectory is plotted on a 2D projected laser map. The gray values in the confusion matrices represent the amount of overlap between the scans given the true relative poses. **Bottom:** Overview over the properties of the indoor datasets: **stop and go**=scans captured in stop and go fashion, **#scans**=number of scans, **#points**=average number of points per scan, **#far ranges**=average number of far range readings per scan, **res.** α=usable angular resolution for range images, **traj. length**=trajectory length, **dist.**=average distance between consecutive scans, **mean range**=average measured range value, **max range**=maximum range value



	stop and go	# scans	# points	#far ranges	res. α	traj. length	dist.	mean range	max range
Freiburg	yes	77	155562	56451	0.45°	723 m	9.51 m	8.9 m	50 m
Hanover	no	923	12959	3622	1.3°	1247.8 m	1.35 m	7.18 m	29 m

Figure 7.4: Top: SLAM trajectories and ground truth confusion matrices for the used outdoor datasets. The trajectories are overlaid on Google Earth aerial images. The gray values in the confusion matrices represent the amount of overlap between the scans given the true relative poses. **Bottom:** Overview over the properties of the outdoor datasets: **stop and go**=scans captured in stop and go fashion, **#scans**=number of scans, **#points**=average number of points per scan, **#far ranges**=average number of far range readings per scan, **res.** α=usable angular resolution for range images, **traj. length**=trajectory length, **dist.**=average distance between consecutive scans, **mean range**=average measured range value, **max range**=maximum range value

7.2.2 Performance Analysis

We calculated the confusion matrices for the datasets by matching each scan with every scan in the database and returning the score of the best found transformation. See Figures 7.5, 7.6, 7.7, and 7.8 for the different datasets. In these figures, (a) shows the respective ground truth confusion matrix and (b) the output of our system. The dark areas that are not close to the main diagonal mark loop closures. Here the system was able to match scans from different points in time where the robot visited a previously visited area (see also Figure 7.3 and Figure 7.4). Visual comparison of the two matrices for each dataset already gives the impression that our result is close to the ground truth version in most areas. To perform a more objective evaluation, we evaluate each match and check if it is a false positive. We do this by comparing the ground truth transformations between the scans with our found transformations and check if the difference exceeds an error value.
Figures 7.5, 7.6, 7.7, and 7.8 (c) give an overview over the number of true positives and false negatives and the resulting recall rate as a function of the distance between scans, using the minimum acceptance threshold for which no false positive was found. These plots show that our approach is highly invariant to translation because recall rates of above 70% could be achieved for all datasets for distances between the scans that were below the average measured range in the dataset.

Figures 7.5, 7.6, 7.7, and 7.8 (d) plot the number of false positives as a function of the acceptance threshold. The recall rate for a manually set maximum distance between scans is also shown. For the AASS-loop dataset we used $1.0\,\mathrm{m}$ as the distance to consider two scans a match. This is the same as was used in related work [85, 49]. The minimum acceptance threshold for which we received no false positive is 0.09. Above this value we have a recall rate of 0.938. For the Quadrotor dataset we used $2.0\,\mathrm{m}$ as the distance to consider two scans a match. We chose this value as the upwards rounded average range measurement in the dataset. The minimum acceptance threshold for which we received no false positive is 0.25. Above this value we have a recall rate of 0.75. For the FreiburgCampus360_3D dataset we used $10.0\,\mathrm{m}$ as the distance to consider two scans a match. We chose this value as the upwards rounded average range measurement in the dataset. The minimum acceptance threshold for which we received no false positive is 0.05. Above this value we have a recall rate of 0.958. For the Hanover2 dataset we used $3.0\,\mathrm{m}$ as the distance to consider two scans a match. This is the same as was used in related work [85, 49]. The minimum acceptance threshold for which we received no false positive is 0.19. Above this value we have a recall rate of 0.925. Please note that our evaluations do not include the diagonal elements of the confusion matrices, meaning we did not match scans against themselves.

Granström and Schön [49] used a machine learning algorithm based on boosting. Therefore they had to split their dataset into learning and test sets for the cross validation and could not evaluate the complete confusion matrix at once. They reported rates of 0.53 ± 0.14 (min 0, max 0.88) for the AASS-loop dataset, where we achieved 0.938, and 0.63 ± 0.6 (min 0.28, max 0.76) for the Hanover2 dataset, where we achieved 0.925. Magnusson *et al.* [85] evaluated their system in a SLAM scenario, where only scans that are at least 30 scans apart are evaluated. In this scenario they got 0.7 as the recall rate for AASS and 0.47 for Hanover2, respectively at 100% precision. With the same setting we got 1 (0.08 acceptance threshold) for AASS and 0.911 (0.19 acceptance threshold) for Hanover2. For an acceptance threshold of above 0.25, our system does not return any false positives for all the tested datasets. Considering this as the acceptance threshold for all datasets, we achieve an overall recall rate of at least 75%.

Figure 7.9 summarizes the results for the different datasets in one plot. It shows the ROC (Relative Operating Characteristic) curves for the datasets, which plot the true positives rate against the false positives rate. In our context that means that the bottom left represents an acceptance threshold of 1.0 for the minimum score to consider two scans a match and 0.0 for the top right. The general idea of these curves is that an approach is better, the higher the curve is above the diagonal. Below the diagonal would be worse than random assigments. Please note that the data used to create this plot does not include the values for a threshold of 0.0 (it actually starts at 0.01), which is why the curves do not reach the top right. The performance on the Quadrotor dataset is the lowest in this plot. The reason for this is that this is the dataset with the highest noise in the 3D scans.

7.2.3 Timings and Influence of the BoW approach

The values given so far are the results we receive, when we do not restrict the time requirements of our system.

For the AASS-loop dataset it takes us 881 ms to extract interest points, features and validation points per scan and 585 ms to match a scan against the database, meaning 10 ms for each scan pair. The equivalent values for the Quadrotor dataset are 305 ms, 102 ms, and 4 ms, for the FreiburgCampus360_3D dataset 1107 ms, 838 ms, and 11 ms, and for the Hanover2 dataset 316 ms, 4132 ms, and 4 ms respectively. All experiments were performed using an Intel I7 quad-core PC.

When using the Bag-of-Words approach, there is an additional overhead for the creation of the histograms (including feature extraction), which is 894 ms for AASS-loop, 276 ms for Quadrotor, 730 ms for FreiburgCampus360_360, and 246 ms for Hanover2.

Using the Bag-of-Words pre-ordering of the potential corresponding scans, we can define a timeout for the database query. Please refer to Figures 7.5, 7.6, 7.7, and 7.8 (e) for an overview, how the recall rates (for the respective minimum acceptance threshold and maximum scan distance) evolve for increasing timeout values. It can be seen that the additional overhead for the histogram calculation is only justifiable for the biggest dataset, namely Hanover2. Here, a recall rate of close to 80% can already be reached after one second per database query. In the same plots there is also a comparison between the rotationally invariant and non-invariant version of the NARFs. It can be seen that the additional degree of freedom introduced by the rotational invariance increases the typical runtime to achieve a certain recall rate and that the maximum achievable recall rate is lowered. But overall, the recall rates are still above the values of the other state-of-the-art systems in the related work.

Please note that we used the Freiburg dataset to learn the dictionary for the Bag-of-Words approach. Therefore the result from Figure 7.7(e) might be overconfident.

7.3 Related Work

In the past, the problem of place recognition has been addressed by several researchers and approaches for different types of sensors have been developed. Cameras are often the first choice. Compared to 3D data, vision features are typically very descriptive and unique. However, spacial verification is naturally easier in 3D data. One very successful approach using vision is the Feature Appearance Based MAPping algorithm (FABMAP) proposed by Cummins and Newman [25]. This algorithm uses a Bag-of-Words approach based on SURFs [9] extracted from omni-directional camera images and was shown to work reliably even on extremely large-scale datasets. Paul and Newman [100] presented an extended version called FAB-MAP 3D, which also incorporates 3D information into the framework. They report a substantial improvement

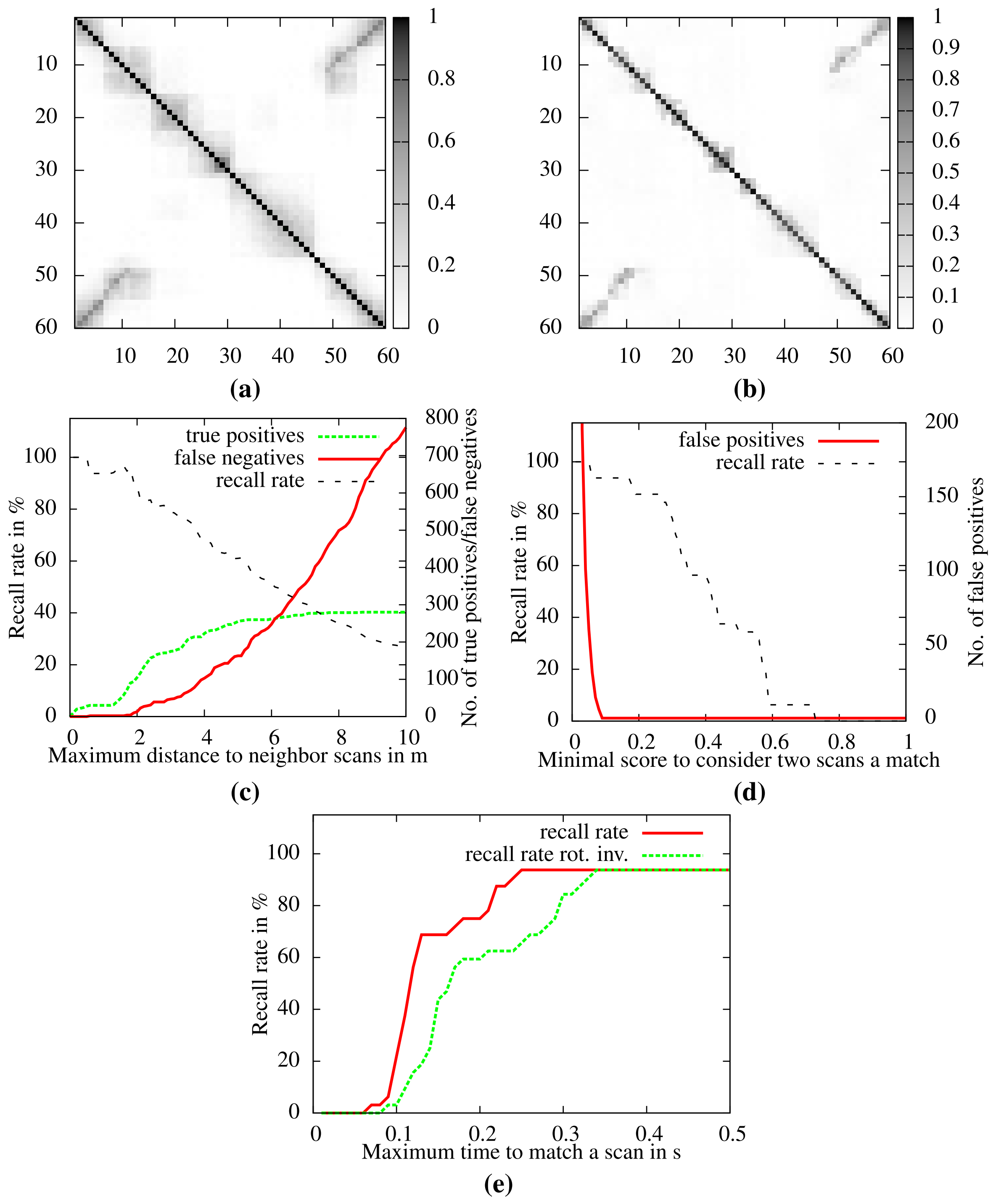
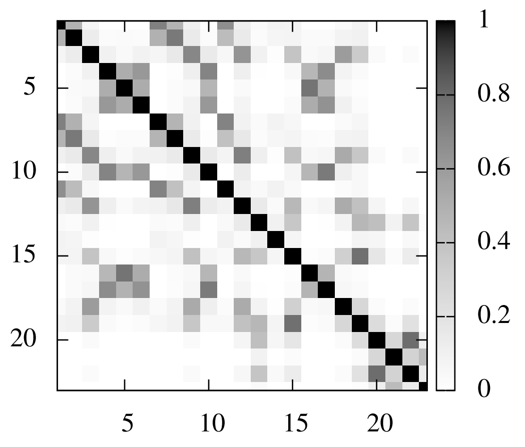
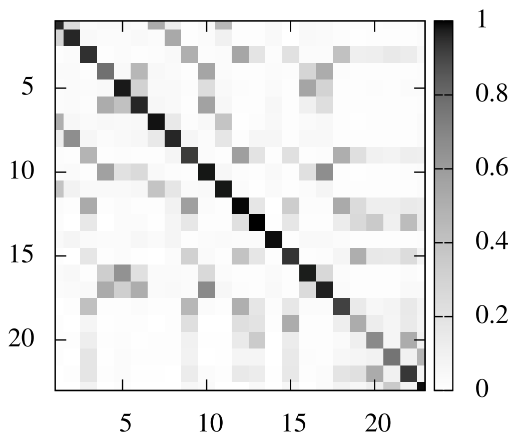


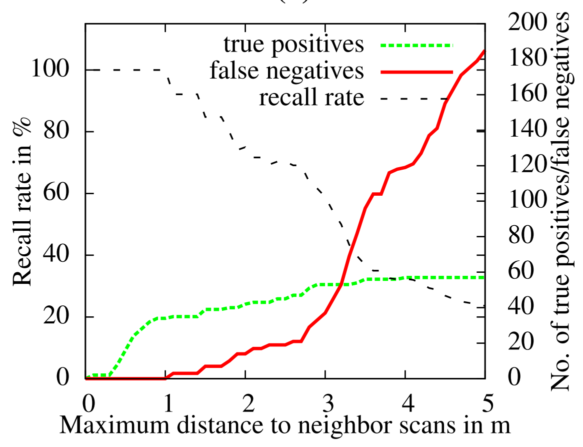
Figure 7.5: Results for the AASS-loop dataset **(a)**: Ground truth confusion matrix of the dataset. The gray values represent the overlap of the scans given the true relative pose. **(b)**: Confusion matrix created by our system. **(c)**: The number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to consider them overlapping. Respectively for the minimum acceptance threshold that did not return any false positives. **(d)**: Number of false positives and the recall rate for different minimum scores. The recall rate is determined regarding a maximum distance of 1.0 m between the scans. **(e)**: The recall rate dependent on the maximum time the system has to match a scan against the database, using the BoW approach. The two plots represent the recall rate with and without the rotational invariance in the NARFs.



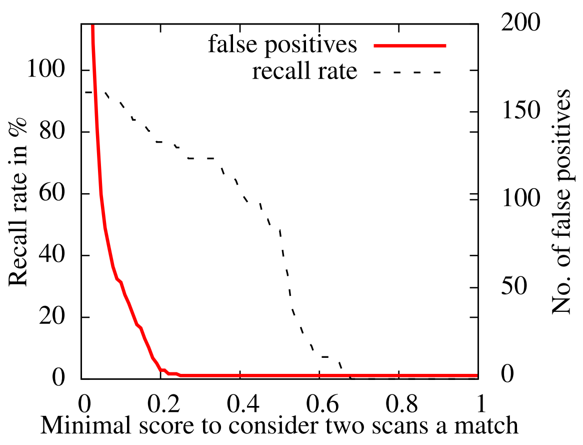
(a)



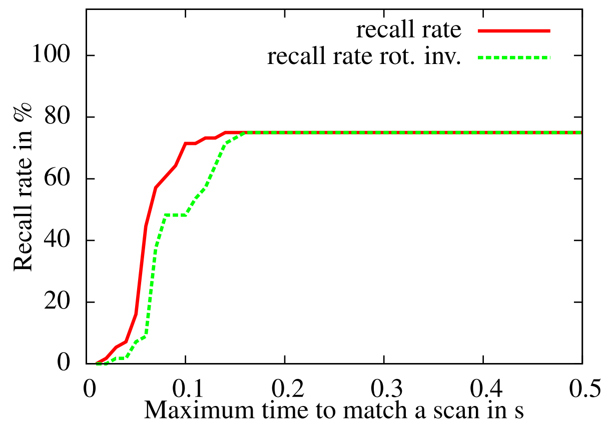
(b)



(c)



(d)



(e)

Figure 7.6: Results for the Quadrotor dataset. (**a**): Ground truth confusion matrix of the dataset. The gray values represent the overlap of the scans given the true relative pose. (**b**): Confusion matrix created by our system. (**c**): The number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to consider them overlapping. Respectively for the minimum acceptance threshold that did not return any false positives. (**d**): Number of false positives and the recall rate for different minimum scores. The recall rate is determined regarding a maximum distance of 2.0 m between the scans. (**e**): The recall rate dependent on the maximum time the system has to match a scan against the database, using the Bag-of-Words approach. The two graphs represent the recall rate with and without the rotational invariance in the NARFs.

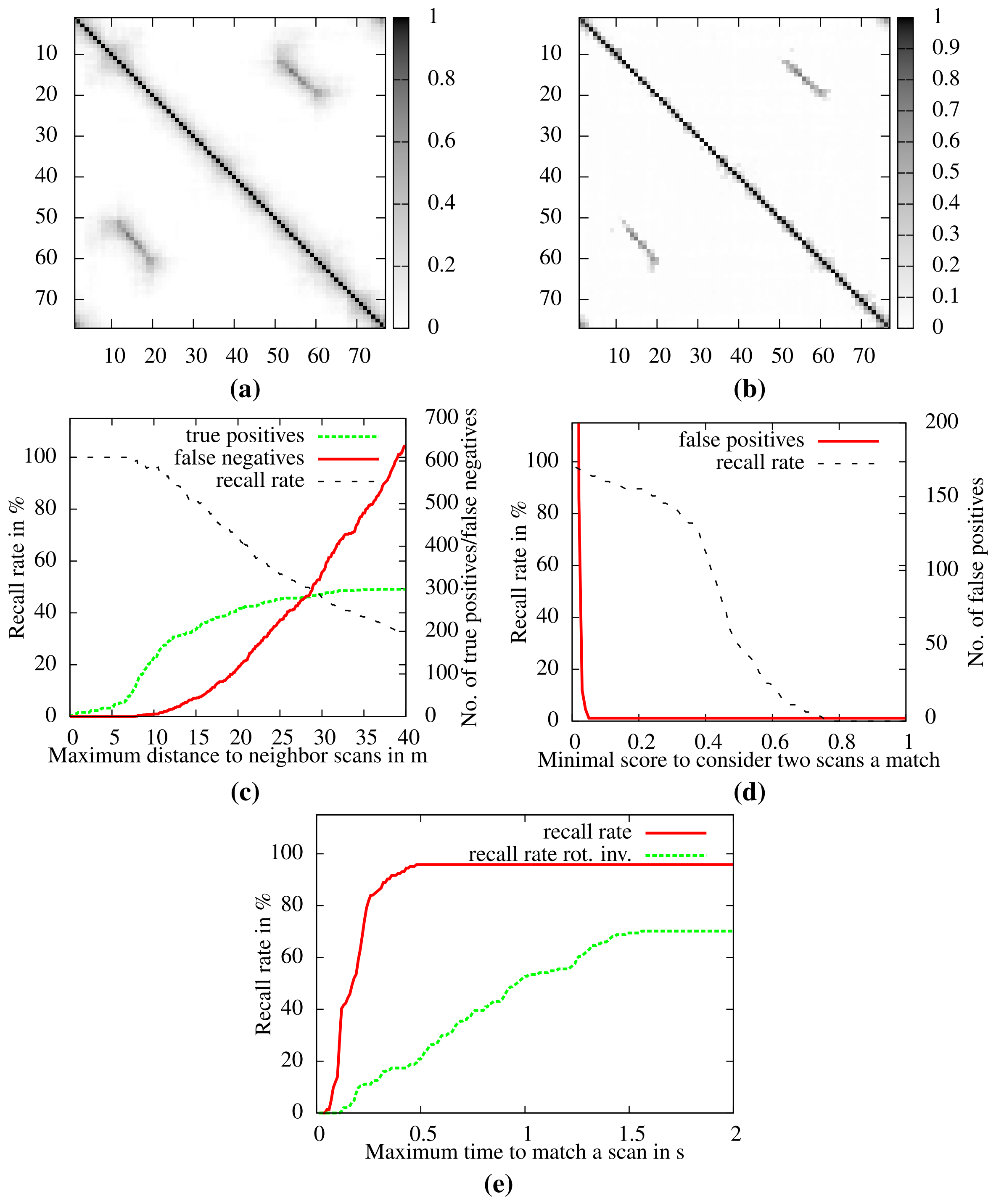


Figure 7.7: Results for the FreiburgCampus360_3D dataset. (**a**): Ground truth confusion matrix of the dataset. The gray values represent the overlap of the scans given the true relative pose. (**b**): Confusion matrix created by our system. (**c**): The number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to consider them overlapping. Respectively for the minimum acceptance threshold that did not return any false positives. (**d**): Number of false positives and the recall rate for different minimum scores. The recall rate is determined regarding a maximum distance of 10.0 m between the scans. (**e**): The recall rate dependent on the maximum time the system has to match a scan against the database, using the Bag-of-Words approach. The two graphs represent the recall rate with and without the rotational invariance in the NARFs.

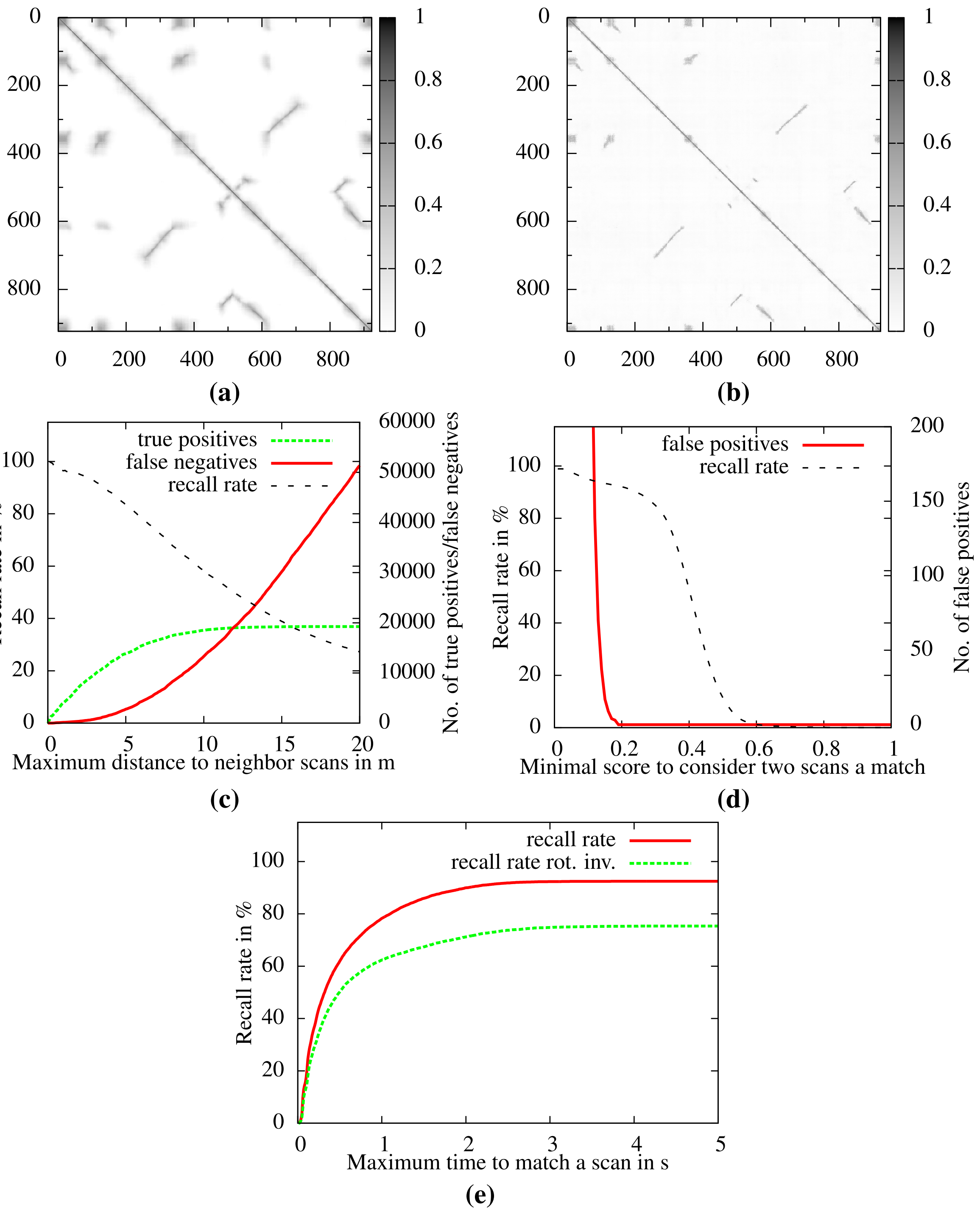
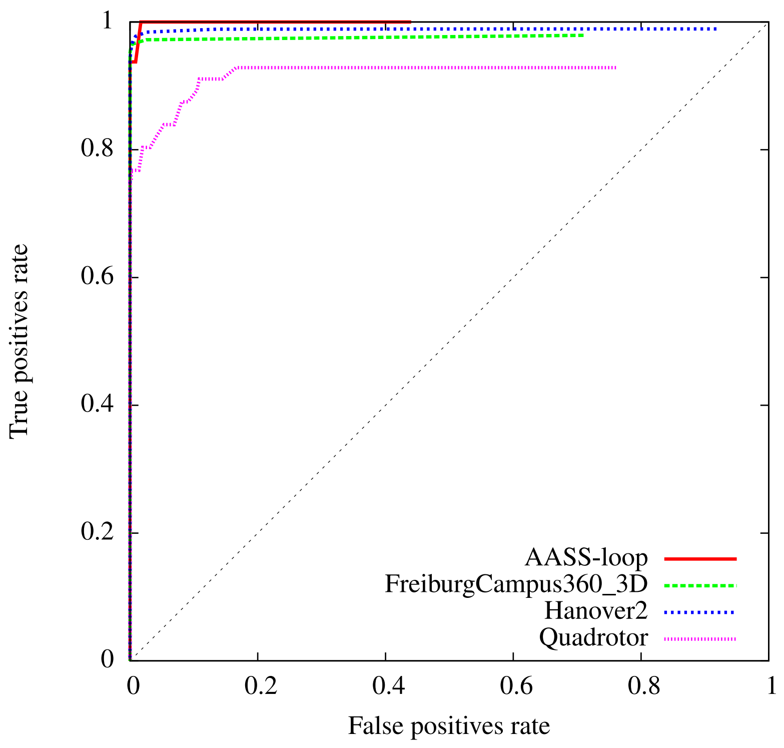
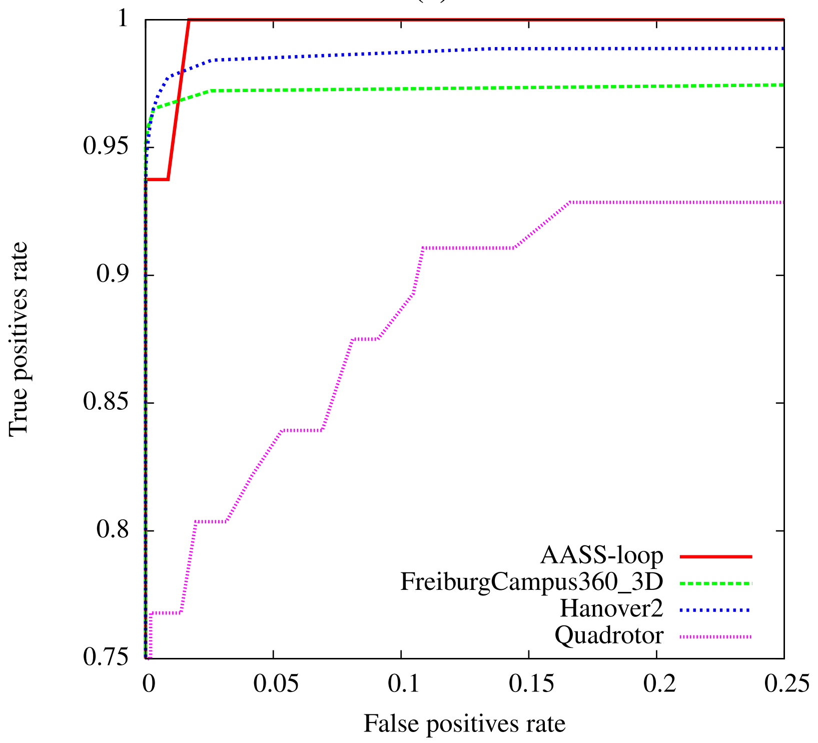


Figure 7.8: Results for the Hanover2 dataset. (**a**): Ground truth confusion matrix of the dataset. The gray values represent the overlap of the scans given the true relative pose. (**b**): Confusion matrix created by our system. (**c**): The number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to consider them overlapping. Respectively for the minimum acceptance threshold that did not return any false positives. (**d**): Number of false positives and the recall rate for different minimum scores. The recall rate is determined regarding a maximum distance of 3.0 m between the scans. (**e**): The recall rate dependent on the maximum time the system has to match a scan against the database, using the Bag-of-Words approach. The two graphs represent the recall rate with and without the rotational invariance in the NARFs.



(a)



(b)

Figure 7.9: **(a)** shows the ROC (Relative Operating Characteristic) curves for the individual datasets. **(b)** is a zoomed in version of the relevant area from the top plot. These curves plot the true positives rate against the false positives rate. The general interpretation is that a result is better, the closer the plot is to the top left corner (close to 1 on the y-axis and close to 0 on the x-axis). The data to create this specific plot does not include the values for an acceptance threshold of 0.0, which is why the curves do not reach the top right. True positives are considered for maximum distances between scan of 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. This is equivalent to the plots in Figures 7.5(d), 7.6(d), 7.7(d), and 7.8(d).

regarding precision-recall, compared to the vision-only approach. The idea to use a BoW approach with visual words was first introduced by Sivic and Zisserman [119] in an approach called Video Google, which implements a system to query for a specific object in a video stream. Next to FABMAP there are a number of successful approaches that later-on employed this method for vision-based place recognition [22, 45, 101].

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 [64]. His approach extracts Spin Images [65] 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 [81] 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 [15] 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.* [138] 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.* [48] 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.

Granström *et al.* [49] also 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.* [85] 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. In Section 7.2.2, we compared our algorithm to these two methods. The results indicate that, while being slower, our approach yields substantially higher recall rates.

7.4 Conclusions

In this chapter we presented a robust approach to 3D place recognition. Our approach provides very high recall rates and also computes accurate relative pose estimates between the involved 3D range scans. We presented a novel sensor model that enables us to efficiently evaluate relative poses between scans and introduced a method to evaluate self similarity to further reduce the risk for false positives. We evaluated our approach on freely available real world datasets and compared the performance of our system with that of other state-of-the-art approaches. While our computational requirements are higher than those, our recall rates on the shared datasets are substantially better. We have shown that our proposed point feature type, the NARF, is also suitable for this kind of problem by providing an efficient way to find candidate transformations between the scans, thereby freeing us from the need to search through a high dimensional parameter space.

Chapter 8

Using Aerial Images as a Prior for Mapping Approaches

Creating consistent maps of the environment is a very important task in mobile robotics. Most mapping approaches do not make use of prior knowledge about the environment they are mapping, thereby discarding valuable sources of information that might improve their results. One of these sources of prior information are aerial image, which provide a globally consistent structure of an environment. In this chapter we present a system that enables a mobile robot operating in an outdoor environment and equipped with a 3D range scanner or a stereo camera to localize itself in aerial images. This makes it possible to get highly accurate global position information, surpassing what is possible with GPS, only using onboard sensors and publicly available image material. This makes the creation of high quality maps within a global reference frame possible. Our system extracts basic features (line structures) from the sensor data and the aerial images and uses them to identify corresponding areas within a particle filter framework. The pose estimates acquired in this fashion are then used in a graph-based SLAM system. Our experimental evaluation on large real world datasets from mixed in- and outdoor environments shows that our approach outperforms state-of-the-art SLAM approaches and GPS regarding global and local consistency. These highly accurate results can then be used as a reference to evaluate the outcome of conventional SLAM approaches. We propose an error metric for this purpose and present an exemplary experiment for such an evaluation.

* * * *

In this chapter, we present a novel approach to enable a mobile robot to localize itself in an aerial image, using either 3D range scans or stereo camera images as sensory input. Such a system is beneficial for many outdoor robotics applications, where very exact and consistent trajectories, going beyond what is possible with GPS, are needed.

In the previous chapters, we mainly focused on 3D point feature extraction, meaning the description of an area around a point in a scan by a descriptor vector and the decision where to extract these, and systems that built upon such a technology. All the handled problems have in common that we need to find similar areas in homogeneous data parts, specifically to relocate a structure from one 3D scan in another one. In this chapter, we try to find correspondences between two very different types of sensor data, namely 3D range scans and 2D visual images.

This boils down to extracting some kind of feature in both data types which can then be used to find correspondences between them.

The trajectories calculated with our approach can be used to create highly accurate maps of the environment, which are also registered in a global reference frame. This is an important property to enable a robot to reach a given GPS coordinate with a minimal error, as required, e.g., for rescue or surveillance tasks. Most modern SLAM approaches work well regarding the creation of consistent maps. Yet, since they typically only use local sensor measurements, accumulated errors interfere with the global consistency and are not necessarily avoidable through the integration of loop closures. This is not critical in most applications because local consistency is typically more important [61], but as mentioned above, some tasks require global consistency and an alignment to a global coordinate frame. GPS measurements can help in this context, but unfortunately they can suffer from substantial positioning errors. Aerial images can be considered to be prior information about the visited area since they can function as an already existing map. Like a human navigating with a visual map, a robot can try to find correspondences between the aerial images and its own measurements. Poses extracted in this way can then be used as priors on the nodes in a graph-based SLAM system (see Section 2.4.1) to improve the original SLAM solution and making loop closures less essential.

The goal of our system is the ability to use aerial images as a prior information for a robot mapping an environment, yet falling back gracefully to a standard SLAM solution in cases where the prior is not available, e.g., when the robot enters a building or if the environment does not provide enough unique structure to be found in both the sensor data and the aerial image. We performed experiments on large-scale datasets in mixed in- and outdoor environments and we compare our method with state-of-the-art SLAM systems and GPS.

Figure 8.1 shows a motivating example of three different maps acquired by a mobile robot driving in a mixed in- and outdoor scenario. The images show a SLAM solution that is locally consistent but does not align properly with the aerial image, a map that is only based on positions the robot was able to match against the aerial image, therefore being aligned properly with the building but only showing the structure visible from the outdoor area, and a map based on our combined approach, showing the complete mapped area properly aligned with the aerial image.

Our method produces highly accurate trajectory estimates, which offer both strong global and local consistency. We therefore propose that such trajectories and the resulting maps can be used to evaluate conventional SLAM approaches that do not make use of prior information and we present an error metric that can be used for such evaluations, as we will demonstrate for a large-scale outdoor dataset.

The remainder of this chapter is organized as follows: We will first present our system for SLAM using aerial images as prior information in Section 8.1, whereas Section 8.1.1 gives a general overview, which is followed by more detailed explanations about the individual components. Afterwards we present our experimental evaluation in Section 8.2. Following this we propose a metric to use the trajectories acquired by our system to evaluate other SLAM approaches in Section 8.3, whereas an exemplary evaluation of three SLAM methods on a large-scale outdoor dataset is given in Section 8.3.2. This is followed by a discussion of related work in Section 8.4 and our conclusion in Section 8.5.

8.1 Our Approach

We will now describe our system to determine highly accurate SLAM trajectories in a global reference frame based on 3D sensors and aerial images. We will first give a quick overview how our system works, followed by a more detailed description of the involved algorithms.

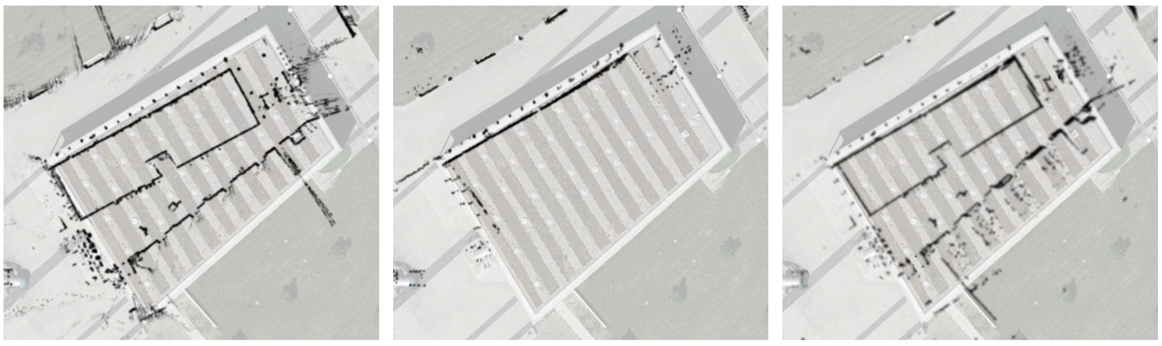

(a) (b) (c)

Figure 8.1: This image motivates the benefit of our approach. A mobile robot was driving in a mixed indoor and outdoor environment, capturing a map. **(a)** shows a bird's eye perspective of the laser map from a state-of-the-art graph-based SLAM system, overlain onto an aerial image. While the map is locally consistent, it does not align properly with the image. **(b)** shows the solution if only positions are used, where the robot could match its observations with the aerial image, meaning those acquired outdoors. This map aligns well with the aerial image but lacks all the indoor structures. **(c)** shows our combined approach, where all the perceived measurements are integrated and are properly aligned with the aerial image. (Source: [73])

8.1.1 Overview

Our approach finds global poses using a *Monte Carlo Localization* (MCL) approach, which tracks the position of the robot in an aerial image. MCL is an implementation of a particle filter, where every particle represents a potential robot pose. To assign a weight to a particle, the system extracts features on the sensor data and on the aerial image. These features are then used to determine how well an observation from a certain particle pose fits onto the aerial image. The features we use are based on edges in the 3D scans and the visual images since sudden depth traversals in 3D structures typically also create edges visible in camera images. Using 3D data also allows us to determine the parts of the sensor data that should be visible in the perspective of the aerial image.

The poses found by this procedure are then used in a graph-based SLAM system as global anchors for the nodes in the graph, thereby enabling the system to reduce accumulated errors and provide a global reference frame.

We will now first present our MCL system using aerial images and afterwards we will discuss how this is integrated into a graph-based SLAM system.

8.1.2 Monte Carlo Localization in Aerial Images

It is our goal to localize a mobile robot in a given aerial image. Such images are freely available from different sources on the Internet. We chose a *particle filtering* approach [36], more specifically MCL [32], to track the position of the robot in the aerial image in a robust manner.

MCL is based on the idea, that possible robot poses are sampled. Each of this samples, or particles, represents a potential robot position. In each time step these particles are moved based on the motion model of the robot (in the *prediction* step) and weighted according to its sensor measurements (in the *correction* step). Such a cloud of particles is a possibility to approximate a probability density $p(\mathbf{x}_t \mid \mathbf{z}_{1:t}, \mathbf{u}_{0:t-1})$ representing the location $\mathbf{x}_t$ of the robot at time t given all observations $\mathbf{z}_{1:t}$ and all control inputs $\mathbf{u}_{0:t-1}$. Using the Markov assumption we can express this recursively as

$$p(\mathbf{x}_t \mid \mathbf{z}_{1:t}, \mathbf{u}_{0:t-1}) = \alpha p(\mathbf{z}_t \mid \mathbf{x}_t) \int p(\mathbf{x}_t \mid \mathbf{u}_{t-1}, \mathbf{x}_{t-1}) p(\mathbf{x}_{t-1} \mid \mathbf{z}_{1:t-1}, \mathbf{u}_{0:t-2})\, d\mathbf{x}_{t-1}, \tag{8.1}$$

which conforms to the Bayesian filtering scheme. The purpose of the term α is to ensure that $p(\mathbf{x}_t \mid \mathbf{z}_{1:t}, \mathbf{u}_{0:t-1})$ sums up to 1 over all $\mathbf{x}_t$. The term $p(\mathbf{x}_t \mid \mathbf{u}_{t-1}, \mathbf{x}_{t-1})$ is called the *prediction model* and represents the motion model of the robot which encodes the probability that the robot ends up in $\mathbf{x_t}$ if it executes the motion command $\mathbf{u_{t-1}}$ while being in $\mathbf{x_{t-1}}$. The term $p(\mathbf{z}_t \mid \mathbf{x}_t)$ is called the *sensor model*, because it encodes the probability that the sensors of the robot return a certain measurement $\mathbf{z_t}$ given the state $\mathbf{x_t}$.

In MCL the current state is approximated using the above mentioned particle cloud, whereas each particle has an associated weight. In the prediction step at time t the particles are moved according to the motion model, meaning that a new pose is sampled according to $p(\mathbf{x}_t \mid \mathbf{u}_{t-1}, \mathbf{x}_{t-1})$. We use a standard motion model as proposed in [32]. In the correction step, the particle weights are updated according to the sensor model $p(\mathbf{z}_t \mid \mathbf{x}_t)$, meaning how well the current observations correspond to the map according to the respective particle pose. The larger the number of particles, the closer this method becomes to implementing the true probability distribution. To be able to handle real problems with a limited number of particles we have to regularly resample the particles to remove particles with a low weight and add particles to areas with a high probability. Yet, if this step is performed to often, the filter might lose correct poses and diverge. We therefore follow the method proposed by Doucet *et al.* [36], where resampling is only performed when the number of effective particles n_{eff} is low, according to the following definition:

$$n_{\text{eff}} = \frac{1}{\sum\limits_{i=1}^{n} w_i^2}, \tag{8.2}$$

where n is the number of particles and w_i is the normalized (i.e., $\sum_{i=1}^{n} w_i = 1$) weight of the i-th particle. In our implementation we use $n = 1,000$ particles and resample when n_{eff} drops below 500.

For our current application, the interesting part is how we define the sensor model to evaluate a potential position of the robot on the aerial image. We will now present our approach regarding this for two different sensors. First for a 3D laser range finder and second for a stereo camera.

8.1.3 Sensor Model for 3D Laser Range Scans

It is now our goal to find a possibility to match 3D laser range scans against aerial images. For this purpose we want to extract a type of feature from the aerial image and from the sensor data that can be used in the particle filter to evaluate how well the positions match. Since the aerial image is only two-dimensional, we to select points from the 3D scan that are likely to result in structures that can be identified and matched in the image. We therefore transform both, the scan and the image, into a set of 2D points, which can than be compared by a simple distance measure.

We use the Canny edge extraction procedure [19] on the aerial image, with the intention that a height gap in the world will also result in a change in intensity in the aerial image, leading to a visible edge. In an urban environment such edges are typically generated by boundaries of

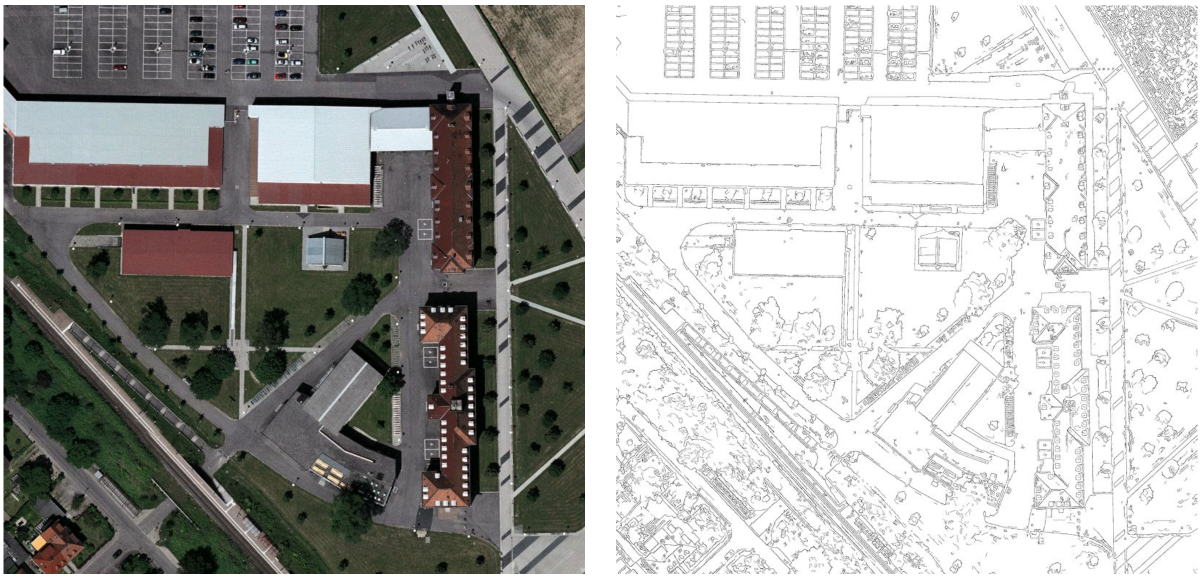


Figure 8.2: Example image for the Canny edge extraction on an aerial image **(a)**: Google Earth image of the Freiburg campus. **(b)**: The corresponding Canny image. Note that the structure of the buildings and the vertical elements is clearly visible despite of the considerable clutter.

roofs, trees, or fences. The edge extraction procedure also returns a lot of false positives that do not represent any actual 3D structure, like street markings, grass boundaries, shadows, and other flat markings. These aspects will have to be considered by the sensor model. Figure 8.2 shows an aerial image and the extracted Canny image. Früh and Zakhor [44] already proposed this method to match 2D laser scans against an aerial image. Yet, the usage of 3D data is much less error-prone, as we will explain below.

We now have to transform the 3D scan into a set of 2D points that can be compared to the Canny image. Since the aerial image is captured from above in a bird's eye perspective, it is reasonable to simulate a similar perspective on the 3D scan. From this perspective we select all points that are visible and project them onto the ground plane. To perform this operation we compute the z-buffer [41] of a scan from a bird's eye perspective, thereby discarding points which could not have been visible in the aerial image. By simulating this view, we handle situations like overhanging roofs, where the house wall is occluded and therefore is not visible in the aerial image in a more sophisticated way than just considering 2D scans which only see the wall or all the 3D scan points, which would cause more false correspondences. An implementation purely based on a 2D scanner (like the approach proposed by Früh and Zakhor [44]) would not account for occlusions due to overhanging objects. Trees are another example, where this makes a substantial difference. In a 2D view we would only sense the trunk, whereas the whole treetop is visible in the aerial image.

The regions of the z-buffer that are likely to be visible in the Canny image are the ones that correspond to relevant depth changes. In our implementation we consider variations in height of 0.5 m and above as possible positions of edges that could also be visible in the aerial image. We construct a 2D scan by considering the 2D projection of the points in these regions. This procedure is illustrated by the sequence of images in Figure 8.3.

We are now able to match the positions of these variations relative to the robot against the Canny edges of the aerial image in a point-by-point fashion. This is similar to the methods commonly used to match 2D laser scans against a 2D occupancy grid map. In this case we use

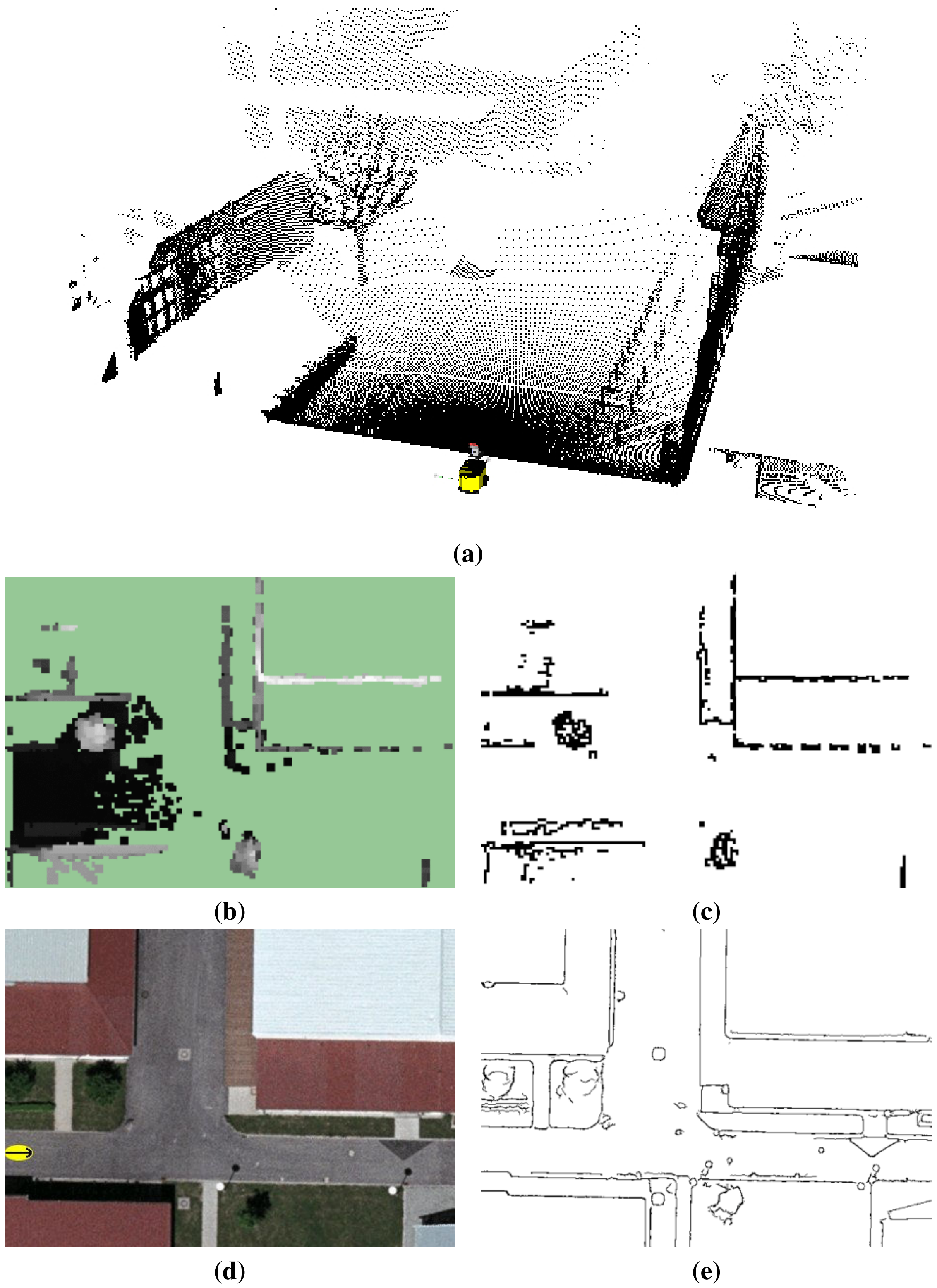


Figure 8.3: Visualization of the procedure to extract a 2D structure from the 3D scan, which can be matched against the canny image extracted from the aerial image. **(a)**: A 3D scan represented as a point cloud. **(b)**: The 3D scene from (a) seen from the top. Gray values represent the maximum height per cell. The darker a pixel, the lower the height. The green area was not visible in the 3D scan. **(c)**: The positions extracted from (d), where a high variation in height occurred. **(d)**: The aerial image of the corresponding area (Source: Google Earth). The position of the robot is marked in yellow. **(e)**: The Canny edges extracted from the aerial image. Please note the clear visual correspondence between (c) and (e).

the so called *endpoint model* or *likelihood fields method* [135] to match our observations from the 3D scan to the map from the aerial image in our sensor model. Let $\mathbf{p}_i^{\mathbf{z}}$ be the i-th point in the set of 2D points extracted from the 3D scan $\mathbf{z}$ (i.e., the points visible in Figure 8.3(c)) and let $\mathbf{p}_{\mathbf{x},i}^{\mathbf{z}}$ be the corresponding point projected onto the aerial image map according to the pose $\mathbf{x}$ of the robot. The endpoint model computes the likelihood of the 3D scan based on the 2D distance between $\mathbf{p}_{\mathbf{x},i}^{\mathbf{z}}$ and the point $\mathbf{p}_i^{\mathbf{m}}$ in the map that is closest to $\mathbf{p}_{\mathbf{x},i}^{\mathbf{z}}$, as a value proportional to the probability derived from a Gaussian and the assumption that the individual points are independent:

$$p(\mathbf{z} \mid \mathbf{x}) = \prod_i e^{-\frac{1}{2}\left(\frac{||\mathbf{p}_{\mathbf{x},i}^{\mathbf{z}} - \mathbf{p}_i^{\mathbf{m}}||_2}{\sigma}\right)^2} \tag{8.3}$$

In this equation, σ is the assumed standard deviation for the Gaussian error distribution. Please note that we skipped the time indices in this equation for better readability.

The endpoint model has some limitations in this context. Visually cluttered areas can create situations where random correspondences in the Canny edges from the clutter lead to overconfidence regarding the position estimate. Additionally, systematic errors can be caused by wrong line correspondences, e.g., when lines resulting from shadows in the aerial image are confused with the lines of a nearby wall. Yet, we could not find evidence that this leads to substantial errors when one applies position tracking and as long as the robot does not traverse such areas for an extended time period. The main motivation to use the endpoint model in this context, next to its high efficiency, is that it only considers structures that are close to the sensor measurements, thereby ignoring the Canny edges that originate from flat structures not visible in the 3D data. Of course our method strongly depends on the quality of the aerial images. Perspective distortions might introduce a systematic error. Yet, in the datasets we used for our experiments, we did not encounter any major problems resulting from such situations. There are also cases where there are no correct correspondences between the aerial image and the sensor data possible because the robot is indoors or moving under overhanging structures. To detect these situations, we use the data from the 3D scanner. In the existence of 3D measurement above the robot, we do not perform the correction step in the particle filter because we assume that the area the robot is sensing is not visible in the aerial image. While a more profound solution is clearly possible, this simple heuristic proved to be reliable enough for our purposes.

We have now presented our method to localize a robot in an aerial image using a 3D laser scanner. We will now introduce a similar method using a stereo camera instead.

8.1.4 Sensor Model for Stereo Cameras

Using 3D scanners for localization purposes has great advantages because of the large field of view and the accurate spatial information they provide. Yet, to match them against visual images has the disadvantage of missing some clues from flat structures that are not visible in the 3D data. We therefore present a method for using a stereo camera instead.

We utilize the color information from the stereo images to enable the robot to take advantage of flat structures. This includes street markings or boundaries between different ground surfaces. We do this as follows: In a first step we process the stereo image to obtain 3D information. Then we apply an adapted version of the Canny edge detector to the camera image since the same edges that are visible in the aerial image might also be visible in the stereo camera image. Since the aerial image is an orthogonal view, we discard features that are not obtained from the ground plane. In the last step we project the 3D points on the ground plane to obtain a 2D set of points which can be used in the same fashion as the 2D points extracted from the 3D laser scans in Section 8.1.3.

The aerial image and the camera images of the robot differ substantially regarding viewpoint and resolution. This can lead to situations, in which the robot detects structures on the ground that are not visible in the aerial image. Consider, for example, Figure 8.4(a) which shows a stone pattern in the stereo camera image. Such a pattern typically leads to edges in the Canny image (see Figure 8.4(d)). Typically, such fine structures are not visible in the aerial image (see Figure 8.4(c)) due to the much lower resolution and might disturb the matching process. To reject these fine structures, one can increase the acceptance threshold of the edge extraction (see Figure 8.4(e)), but this also removes true positives from the detected edges, i.e., lines that are also visible in the aerial image. Ideally, one would like to remove the false positives resulting from fine structures near the robot, while keeping true positives that are farther away. This is not possible by adjusting the threshold alone. By exploiting the 3D information extracted from the stereo images, we can relate depth measurement for an image pixel with the structure size. The idea is that fine edges that are far away result from large structures visible in the aerial image, but fine edges near the robot represent small structures. Given the distance, we can adapt the level of blur in different regions of the image. Regions closer to the robot will become more blurred than regions that are farther away. In particular, we process each camera image by adding a distance-dependent blur to each pixel, whereas we choose the kernel size for the blurring inversely proportional to the 2D distance between the measured 3D point and the robot (see Figure 8.4(f)). This dynamic blur can be implemented efficiently using box filters in combination with integral images [9]. In this way, we take the low resolution of the aerial image into account and reduce the influence of fine structures that are likely to be not visible in the aerial image. Please note that the same procedure could be applied for a 3D scanner in combination with a calibrated mono camera.

Up to now, we explained, how we match either 3D scans our stereo camera images against an aerial image in an MCL, to acquire global position estimates for our robot. We will now explain, how these positions are integrated into a graph-based SLAM system as prior information.

8.1.5 Graph-based SLAM using Prior Information

Our approach uses the graph-based SLAM formulation, which we discussed in Section 2.4.1. In this formulation, every node in the graph represents a robot pose and the edges represent spatial constraints between the nodes. These are relative transformations between the nodes, often extracted from different sensor observations (odometry, scan matching, shared landmark observations, etc.). In addition to these relative measurements, we now want to add global position measurements coming from found correspondences between aerial images and the current sensor measurements of the robot. These are added as new global links to the graph (see the yellow nodes in Figure 8.5) and introduce a new error term into Equation 2.2 to define a new negative log-likelihood as follows:

$$\chi'(\mathbf{x}) \quad = \quad \chi(\mathbf{x}) + \sum_i \mathbf{e}(\mathbf{x}_i, \hat{\mathbf{x}}_i)^T \Omega_i \mathbf{e}(\mathbf{x}_i, \hat{\mathbf{x}}_i), \tag{8.4}$$

where $\mathbf{x}_i$ is the position of node i in the graph and $\hat{\mathbf{x}}_i$ is the prior for this node, meaning the global position as determined by our approach for localization in aerial images. The term $\mathbf{e}(\mathbf{x}_i, \hat{\mathbf{x}}_i)$ describes the error between these two poses and Ω_i is the corresponding information matrix, which we estimate based on the particle distribution during the localization procedure. $\chi(\mathbf{x})$ represents the negative log-likelihood for the normal edges between the nodes, not coming from a global link (see Equation 2.2). As in the standard graph-based SLAM formulation

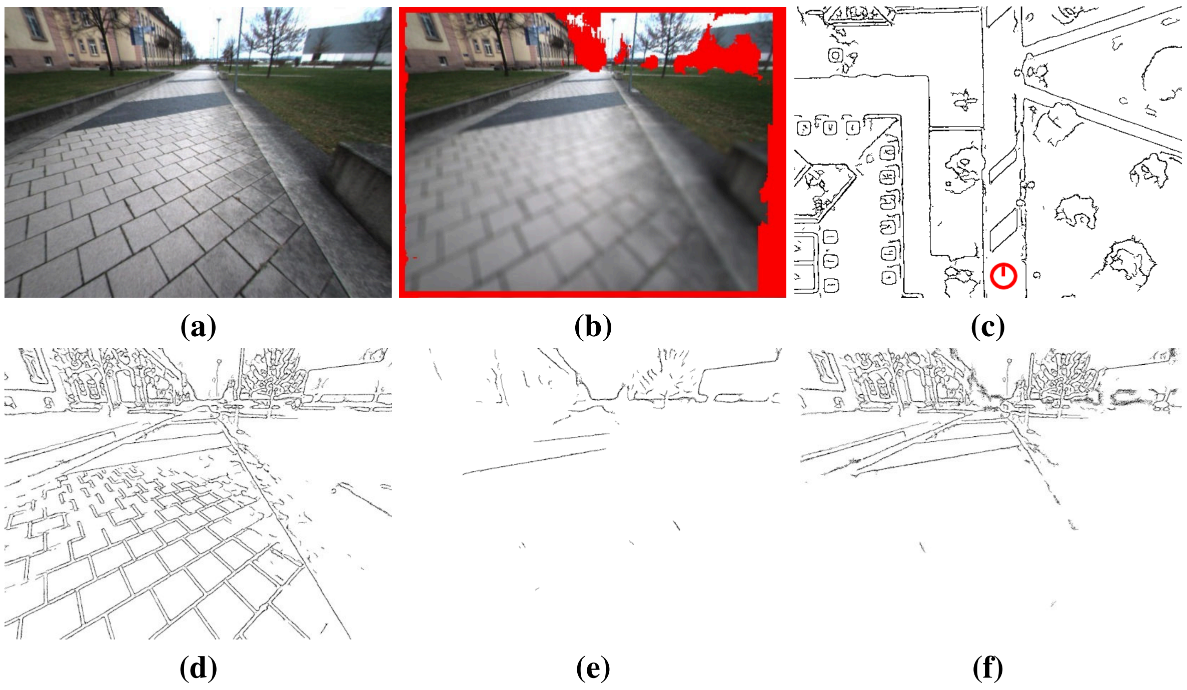


Figure 8.4: Calculation of corresponding points between an aerial image and a stereo camera image. **(a)**: The original camera image (the left image of the stereo pair). **(b)**: A blurred version of (a), where the strength of the blur is inversely proportional to the distance. Areas where no 3D information was available are marked in red. **(c)**: The corresponding part in the Canny aerial image (Source of the original image: Google Earth) with the robot position marked in red. **(d)**: Standard Canny on the original camera image using low edge detection thresholds. **(e)**: Standard Canny on the original camera image using edge detection thresholds that are high enough that the pattern of the ground directly in front of the robot is not recognized as edges. **(f)**: Standard Canny on the dynamically blurred image using the same edge detection thresholds as in (d). Most of the important edges, i.e., those on the ground that are also visible in the aerial image, were extracted correctly, whereas the pattern on the ground, which was not visible in the low resolution aerial image, is correctly discarded.

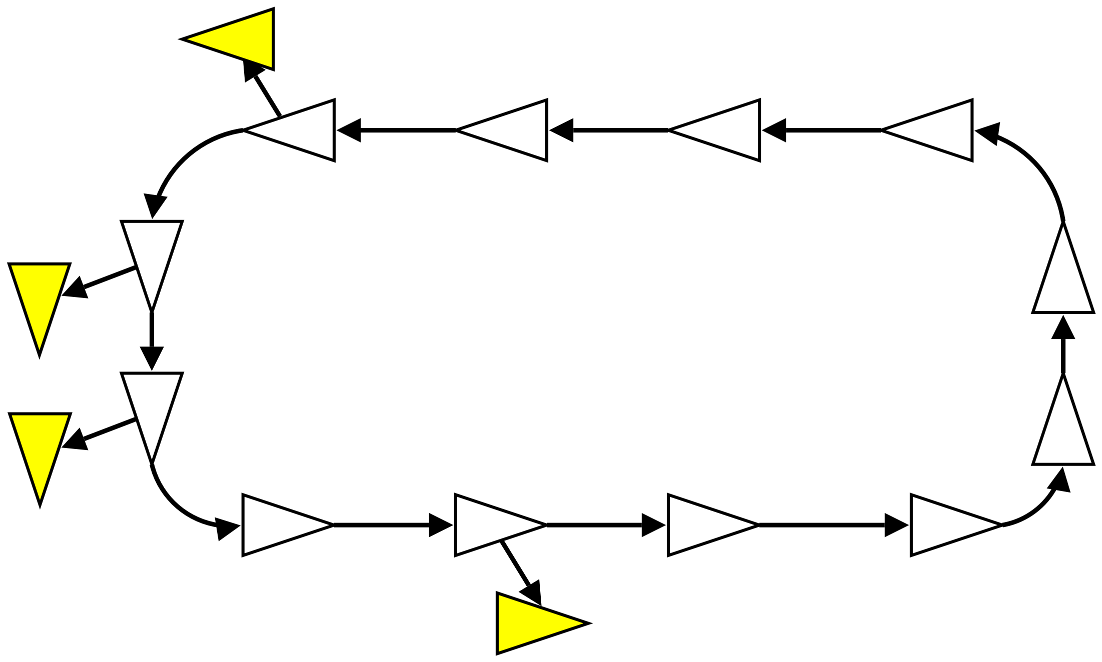


Figure 8.5: Graph for our graph-based SLAM system, where the triangles represent robot poses. The links between the white triangles represent relative position estimates (odometry, scan matching, found loop closures, etc.). The links to the yellow triangles represent the global position measurements. (Source: [73])

(see Equation 2.3) we now search for the configuration of the nodes that maximizes this log-likelihood:

$$\tilde{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x}} \chi'(\mathbf{x}). \tag{8.5}$$

We solve this optimization problem using the g^{2}o-framework [70]. This gives us a solution for all the poses in the robot trajectory, which minimizes the error for all relative constraints between the poses and also for the additional global constraints. This anchors the trajectory to the coordinate frame defined by the global links, which, in our case, is the latitude/longitude frame. From this trajectory we can compute a globally consistent map, as long as the MCL approach provided correct positions. An important property of this approach is that it does not require loop closures.

8.2 Experiments

To evaluate our approach we collected different datasets using a MobileRobots Powerbot platfrom, equipped with a SICK LMS laser range finder mounted on an Amtec wrist unit. The platform continuously acquired 3D scans, while driving with a maximum velocity of $0.35\,\frac{\mathrm{m}}{\mathrm{s}}$, by tilting the laser up and down. During each 3D scan the robot moved up to $2\,\mathrm{m}$. We used the odometry to account for the distortion caused by the movement of the platform. As a stereo camera we employed a Point Grey Bumblebee2 and used the provided software library to calculate 3D points from the stereo pair. See Figure 8.6 for an image of the robot. Although the robot is equipped with many other sensors, we only used the devices mentioned above for our experiments. Table 8.1 summarizes the system parameters we used all our experiments.

To initialize the particle filter localization in the aerial images we used randomly sampled poses around the respective GPS estimate.

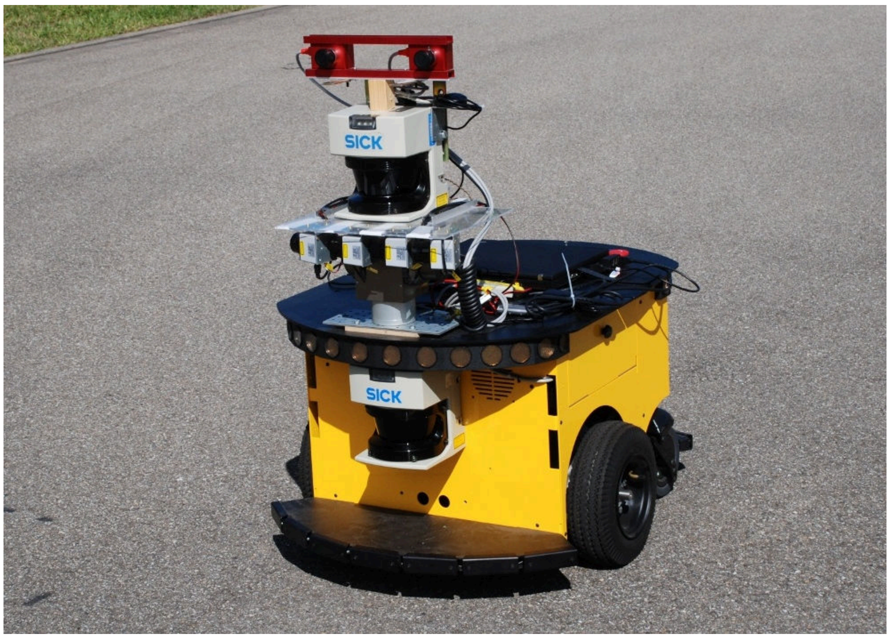


Figure 8.6: The robot we used for our experiments. The top SICK laser range finder mounted on a pan-tilt-unit was used to acquire the needed 3D scans. (Source: [73])

Number of particles	1,000
Minimum driving distance between particle filter updates	2.0 m
Grid resolution	0.15 m
Standard deviation σ from Equation 8.3	2.0 m
Minimum height variation threshold for the top view edge extraction on 3D scans	0.5 m

Table 8.1: Parameters used in all our experiments.

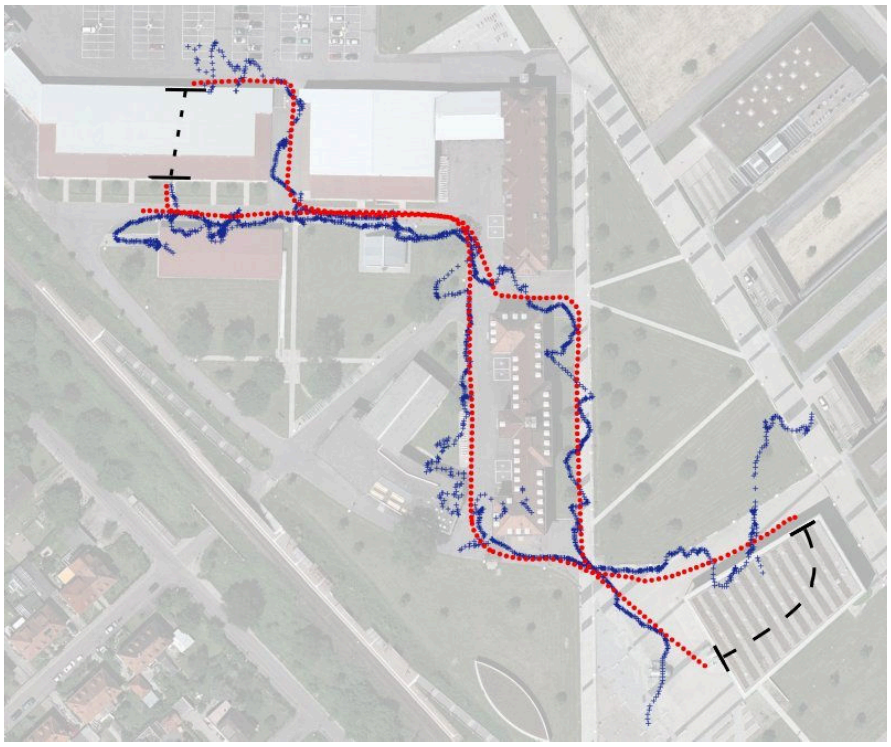


Figure 8.7: This image shows a comparison between the trajectories acquired with our approach (red) and a standard GPS device (blue). The trajectories are overlain on the aerial image (Source: Google Earth) that was also used for our localization approach. The dashed black lines mark areas where the robot was indoors and therefore could not track the position. (Source: [73])

8.2.1 Comparison to GPS

In a first experiment we wanted to show how accurate our localization in aerial images is compared to a standard consumer-level GPS device. For this purpose we captured a log with a $890\,\mathrm{m}$ long trajectory on the campus of the Faculty of Engineering, University of Freiburg. The robot was entering and leaving buildings, collecting 445 3D scans. Figure 8.7 shows a visual comparison between the GPS estimates and the poses acquired with our MCL-based localization on an aerial image. The higher error of the GPS-based approach is clearly visible. In addition, our approach also provides an estimate for the orientation of the robot, which is not possible for the individual GPS measurements.

8.2.2 Comparison of 3D Laser and Stereo Camera

In another experiment we wanted to evaluate how the localization in the aerial image using a 3D laser scanner compares to using a stereo camera. While the 3D laser scanner provides more accurate spatial information and has a larger field of view, the camera can also detect flat structures like road markings, or changes in the ground materials. We collected another dataset on our campus, with a trajectory of $680\,\mathrm{m}$ length. Apart from 3D scans the robot now also collected stereo camera images, whereas the camera was mounted $1.2\,\mathrm{m}$ above the ground, looking $30°$ downwards, to observe a large area of the ground surface. We ran our MCL-based approach for both sets of sensor data using the sensor models described in Section 8.1.3 and Section 8.1.4. To make the comparison more balanced, we used the same sampling frequency for both sensors, meaning we dropped frames from the stereo camera until we had the same number of observations as for the 3D scanner. Figure 8.8 shows the trajectory estimates of the

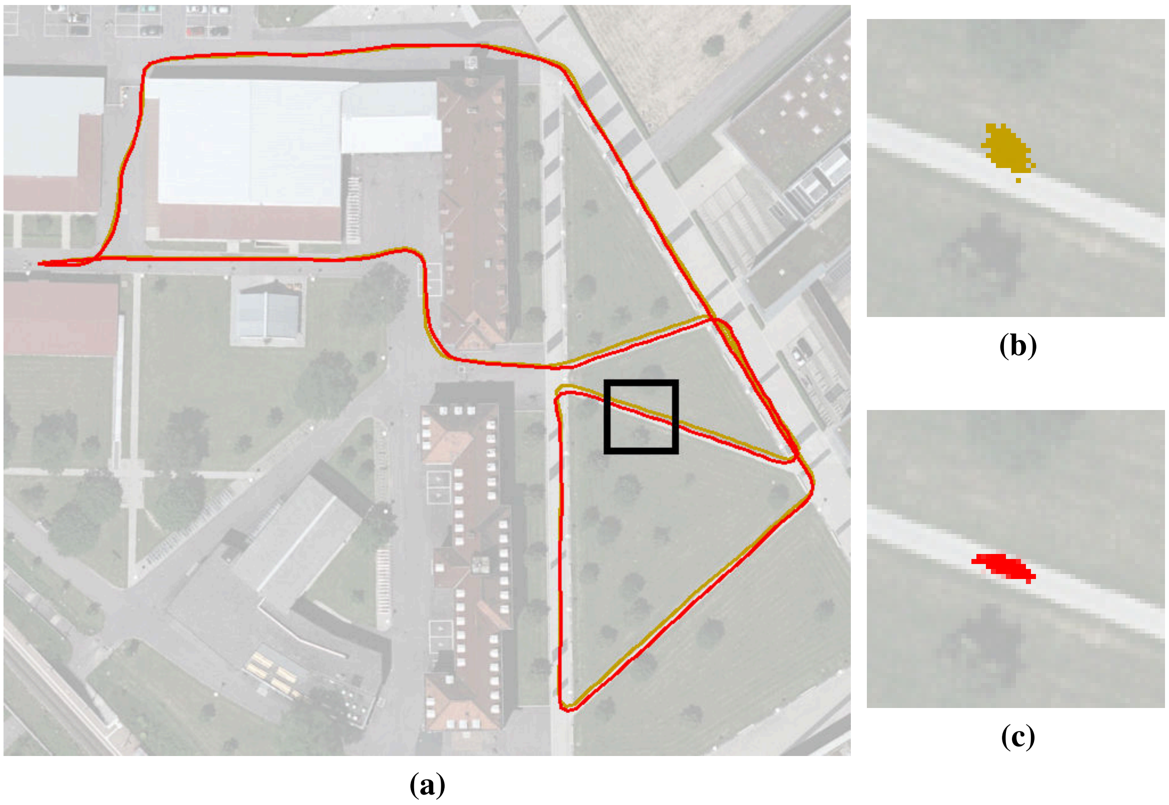


Figure 8.8: **(a)**: A comparison between localization in an aerial image (Source: Google Earth) using a 3D scanner or a stereo camera. The trajectory based on the 3D laser scanner is drawn in brown, whereas the trajectory calculated based on the stereo camera is drawn in red. **(b)**: A zoomed-in version of the area marked by a black box in (a). The particle cloud of the localization run using the 3D laser scanner is shown for a time step when the robot was on the shown footpath. It can be seen, that the particle cloud is slightly of on the grass, whereas the robot drove on the footpath. This area provides nearly no 3D structure helping in the matching against the aerial image. **(c)**: Same as (b), but showing the equivalent particle cloud for the localization run using the stereo camera. The grass boundary is visible in the camera image and can therefore be used in the localization procedure, thereby enabling the particle filter to correctly stay in the footpath, even if there is a higher uncertainty along the footpath. (Source: [73])

two approaches and highlights an area where the stereo camera had a clear advantage. In this area there is nearly no usable 3D structure, because it is mostly flat ground. But the traversal from concrete on the footpath to grass is clearly visible in the aerial image and the stereo camera image, leading to a particle cloud that stays on the footpath instead of drifting to the side. In the other areas the trajectories are nearly identical.

8.2.3 Global Map Consistency and Comparison against Standard SLAM

Our next goal was to evaluate the global consistency of maps created with our SLAM approach as discussed in Section 8.1.5 and to compare it to a state-of-the-art SLAM system not using the prior information from the aerial images. For this purpose we implemented a SLAM system similar to the one proposed by Olson [95]. In our comparison we will either add global links from the aerial images to the graph or leave them out, whereas the latter leads to the standard SLAM solution.

We recorded two datasets in different environments, on our campus and in a residential

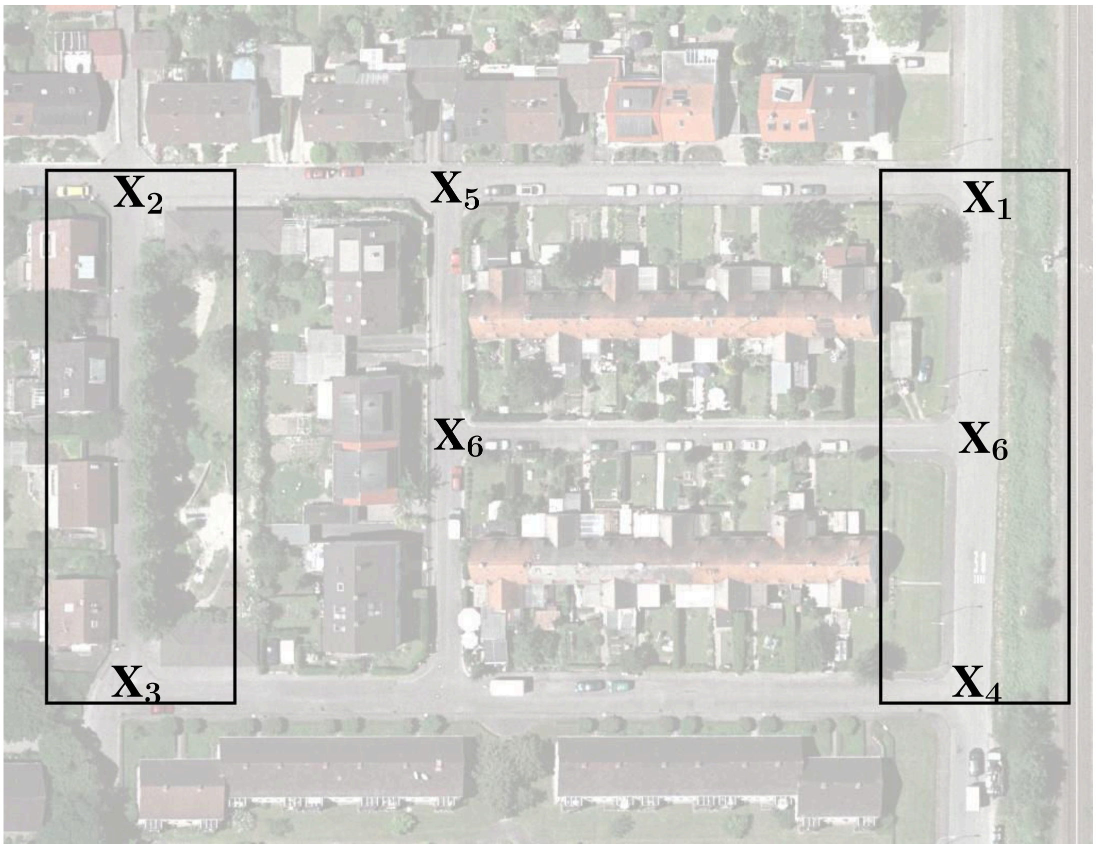


Figure 8.9: This figure shows an aerial image of a residential area (Source: Google Earth) in Freiburg. The seven points marked with X were used for the evaluation. The black boxes mark areas that are challenging for the localization. In the left area the localization using stereo vision fails. Within the area marked on the right the localization using 3D laser data is inaccurate. Only a combination of both sensors made an accurate localization for the whole environment possible. (Source: [73])

area. These two have very different structures. The campus area contains only a few large buildings compared to the residential area, which consists of several small houses with front gardens surrounded by fences or hedges, with cars parked on the street. See Figure 8.2(a) and Figure 8.9 for the aerial images used in our experiments.

For the experiment on our campus we drove the robot five times on trajectories very similar to the one already shown in Figure 8.7. For each of these five runs we created trajectories for the standard SLAM approach and for our approach extending the graph with global links from the aerial image. Figure 8.10 shows the results for one of these runs with some zoomed-in areas that highlight the differences. Our approach leads to maps that align much better with the aerial image.

To perform a more objective evaluation of the resulting maps, we selected six easily noticeable points in the map, namely corners of buildings. Figure 8.11 shows one of the created maps with the marked six points.

We then compared their distances in the maps (five different maps for our approach and five for standard SLAM) with the ground-truth distances. We determined these ground-truth distances using the so-called *Automatisierte Liegenschaftskarte*, which we obtained from the German land registry office. These highly accurate maps contain the outer walls of all buildings. Figure 8.12 shows the resulting error statistics for the distances between the points, showing the superiority of our approach.

We performed an equivalent experiment in the residential area already mentioned above

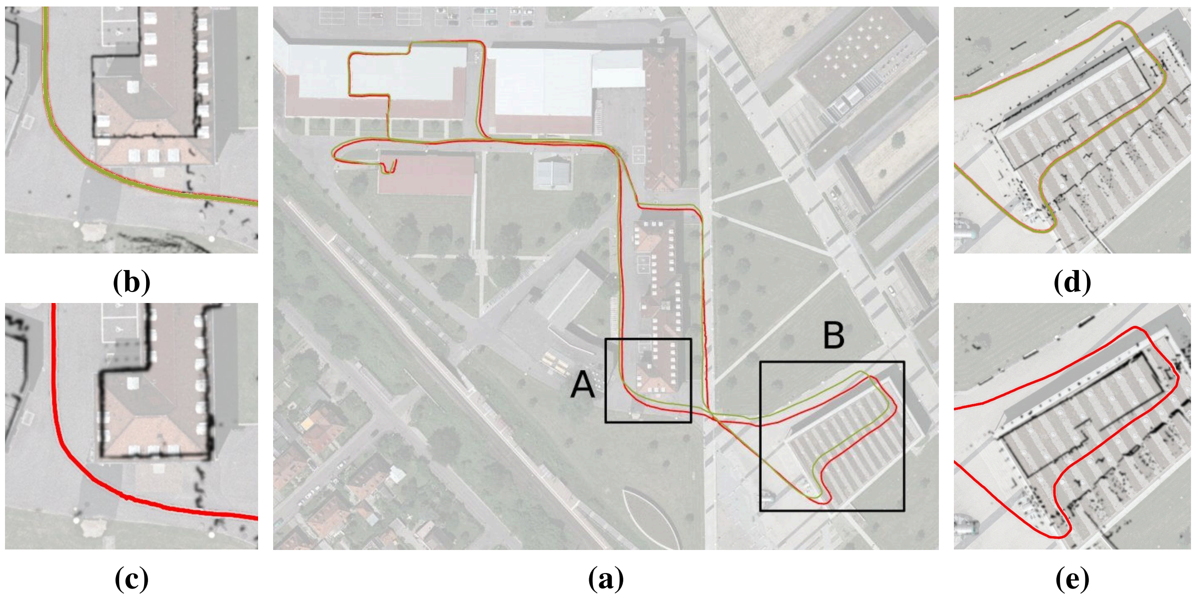


Figure 8.10: Comparison of our system to a standard SLAM approach in a complex indoor/outdoor scenario. **(a)** shows the aerial image (Source: Google Earth) of our campus, overlain with the trajectory estimated by the SLAM approach in brown and the trajectory generated by our approach in red. **(b)**&**(d)** show zoomed in versions of the areas marked A and B in (a), overlain with the standard SLAM trajectory and the resulting laser map. There is a clear misalignment between the building walls in the map and in the aerial image. **(c)**&**(e)** show the same images for our approach. Here the alignment of the laser map and the aerial image is much better. (Source: [73])

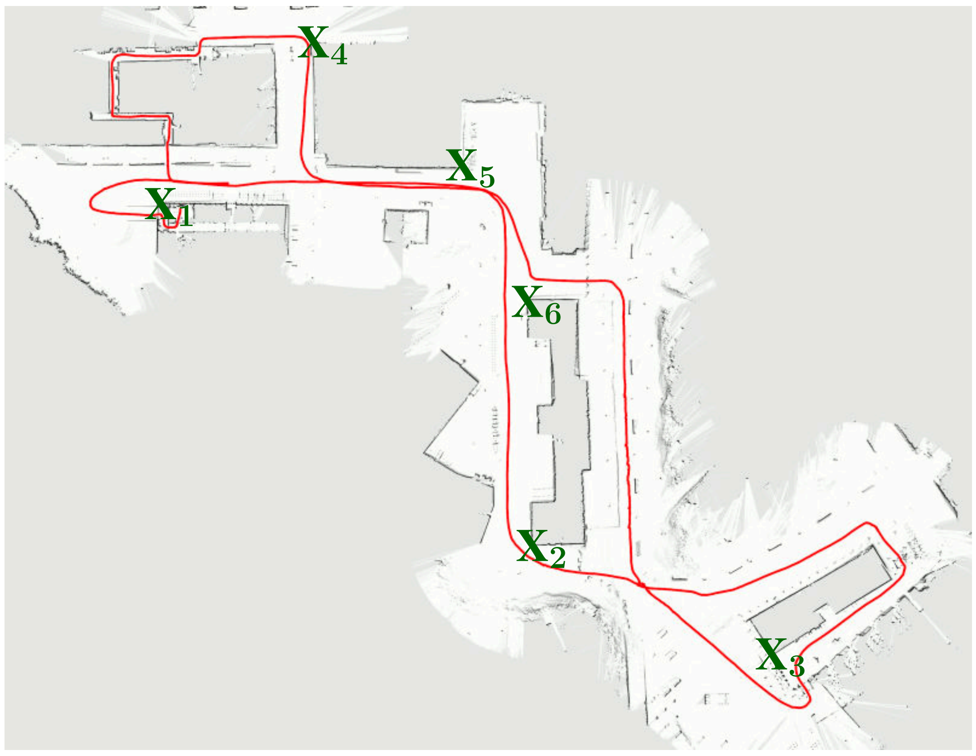


Figure 8.11: An example map with the estimated trajectory marked in red and the used points for the evaluation (building corners) in green. (Source: [73])

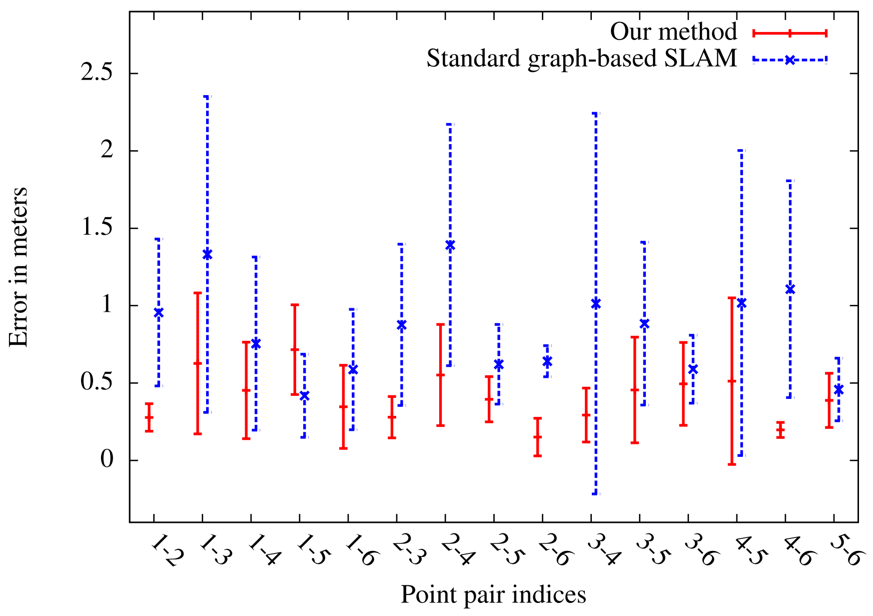


Figure 8.12: Error bars (95% confidence interval) for the estimated distances between the six points used for the evaluation of the map consistency in the campus dataset. (Source: [73])

(see Figure 8.9). We again drove five very similar trajectories of approximately 710 m length in this area. We recorded these different runs at different times and on several days. This means that parts of the environment changed in between, e.g., the position of shadows and cars were different for the individual runs. This environment is less structured than our campus. Especially the parts of the environment which are marked in Figure 8.9 are challenging for the localization. The area marked on the right is dominated by vegetation along a railway embankment, resulting in cluttered 3D range measurements. In this area, our approach using only 3D laser data is unable to accurately localize the robot. In the area marked on the left, the street is partially occluded due to overhanging trees. Due to this the localization using stereo vision data is unable to robustly localize the vehicle. However, in combination, the two sensors enable our approach to localize the robot also in these two areas. The vision data provides useful information about the road boundaries that are not observed by the 3D laser close to the railway whereas the 3D laser measures the trees and building structures in the other problematic region. We calculated maps for each run, both with our approach and with the standard graph-based SLAM approach. For our ground truth evaluation we selected seven points in the environment (see the points marked with X in Figure 8.9), where the robot stopped each time during the runs. This enabled us to identify the same points in the resulting maps. Unfortunately, we do not have a ground truth map of this environment available, as we did on our campus. For this reason we determined the ground truth poses of the seven positions using a highly accurate GPS receiver which achieves a sub-meter accuracy by accumulating data for several minutes. Figure 8.13 provides the error plots for the distances between the individual points. Again, our approach (mean error 0.85 m) outperforms the standard SLAM system (mean error 1.3 m).

As expected, SLAM using the aerial images as prior information leads to smaller errors in the map with lower variances for point positions than a standard state-of-the-art SLAM approach. While the error for the standard approach could be reduced by introducing a higher number of loop closures, our approach has the advantage that it does not require such special care while capturing a dataset.

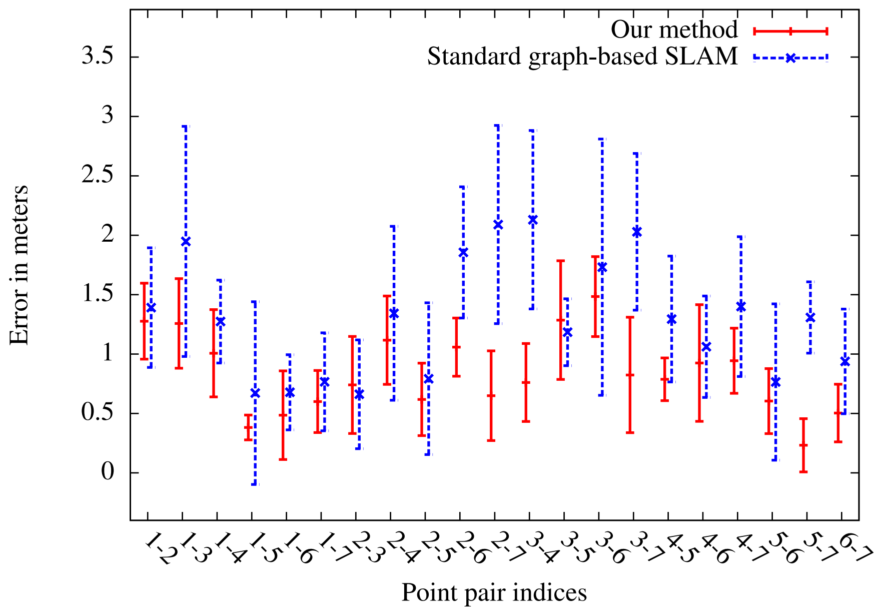


Figure 8.13: Error bars (95% confidence interval) for the estimated distances between the seven points used for the evaluation of the map consistency in the residential area dataset. (Source: [73])

8.2.4 Local Consistency

Up to now, we mainly tested the global accuracy that can be achieved with our approach. The next experiment is designed to check if we also acquire a higher accuracy, compared to standard SLAM, for smaller scale areas in a map. For this purpose we had a closer look at the outer walls of the buildings on our campus regarding their real appearance and there representation in the acquired maps. In this context, walls that are straight in the real world should also be straight in the maps. We used this property in an experiment, where we analyzed an approximately 70 m long building on our campus, visible in the dataset, which we acquired for the experiment in Section 8.2.3. To measure the local accuracy of our approach and the standard SLAM approach, we approximated the first part of the wall as a straight line and extended it to the other end of the building. Then we analyzed how far the other corner is away from this approximated line. Figure 8.14 shows a typical result. The average error between the line and the corner for the standard graph-based SLAM approach is 0.5 m and only 0.2 m for our approach. This leads to the conclusion, that our approach not only improves the global consistency of the maps, but also its local properties.

With this set of experiments we showed that our method using aerial images as prior information for SLAM makes it possible to create maps that are highly accurate regarding global and local properties. In the next section we will discuss, how such maps can be used to evaluate other SLAM approaches.

8.3 Using Our Maps for SLAM Evaluation

We have presented a system that enable us to create highly accurate maps using additional information from public databases and showed that it outperforms traditional SLAM approaches. We therefore reasoned that the trajectories we estimate with our approach can be used to evaluate other SLAM approaches that do not have the advantage of prior information. For this purpose we first have to define an error metric to compare different trajectories.

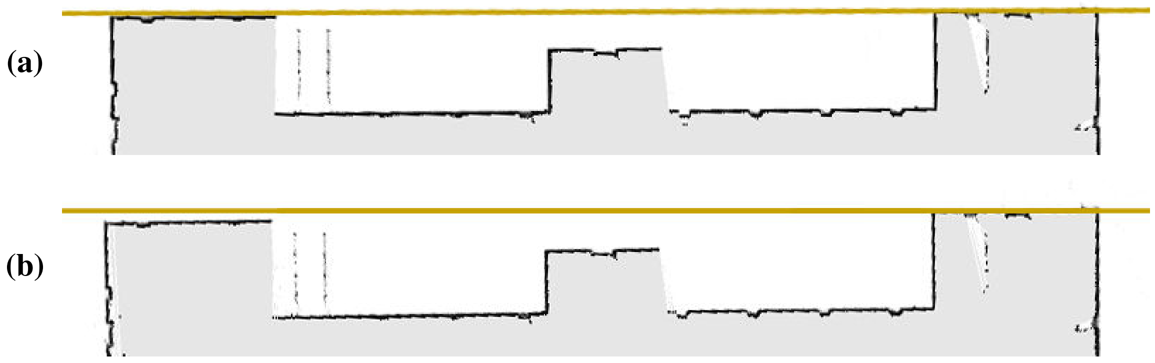


Figure 8.14: This figure shows close-up views of the outer wall of a building. **(a)** shows the result of our approach and **(b)** the result of a standard SLAM approach. The building should be straight, but is actually slightly bend in both views. Yet the error for our approach is much lower, as indicated by the straight brown lines. (Source: [73])

8.3.1 The Error Metric

Let $\mathbf{x}_{1:t}$ be the poses of the robot estimated by a SLAM algorithm from time step 1 to t. Let $\mathbf{x}^*_{1:t}$ be the poses from our approach. A straightforward error metric could be defined as

$$\varepsilon(\mathbf{x}_{1:t}, \mathbf{x}^*_{1:t}) = \sum_{i=1}^{t} \varepsilon(\mathbf{x}_i, \mathbf{x}^*_i)^2, \tag{8.6}$$

where $\varepsilon(\mathbf{x}_i, \mathbf{x}^*_i)$ describes the error between these two poses. We will discuss this in more detail below.

Unfortunately, this error metric is suboptimal for most applications since it only evaluates the global position of the nodes. Let us consider a one-dimensional example, where a robot moves along a straight line and makes a slight motion estimation error in the first step. Every following pose will be shifted by this error and the above error value would increase linearly with the length of the trajectory. Even though the trajectory is mostly correct, it will receive a high error value. On the other hand, if we invert the trajectory, so that the slight mistake occurs at the end, it will receive a low error value. See Figure 8.15 for an illustration. This is a suboptimal behavior of the above error metric because correct global poses are only relevant in some application, whereas most only need local consistency of the derived maps, as we already discussed in the beginning of this chapter.

Based on this we propose an error metric that is derived from the idea of graph-based SLAM and evaluates the deformation energy needed to transfer the estimate into the ground truth. This means that our metric is based on relative displacement between robot poses. Instead of comparing $\mathbf{x}_i$ to $\mathbf{x}^*_i$ in the global reference frame, we compare $\mathbf{d}_{ij} = (\mathbf{x}_i \ominus \mathbf{x}_j)$ with $\mathbf{d}^*_{ij} = (\mathbf{x}^*_i \ominus \mathbf{x}^*_j)$, whereas $\ominus$ denotes the inverse of the standard motion composition operator, meaning it gives the relative displacement between the two input poses.

We can now define an error metric based on a set of index tuples denoting the links between nodes used for the evaluation $\mathcal{E} = \{< i_1, j_1 >, < i_2, j_2 >, \ldots\}$:

$$\varepsilon(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{<i,j>\in\mathcal{E}} \varepsilon(\mathbf{d}_{i,j}, \mathbf{d}^*_{i,j})^2. \tag{8.7}$$

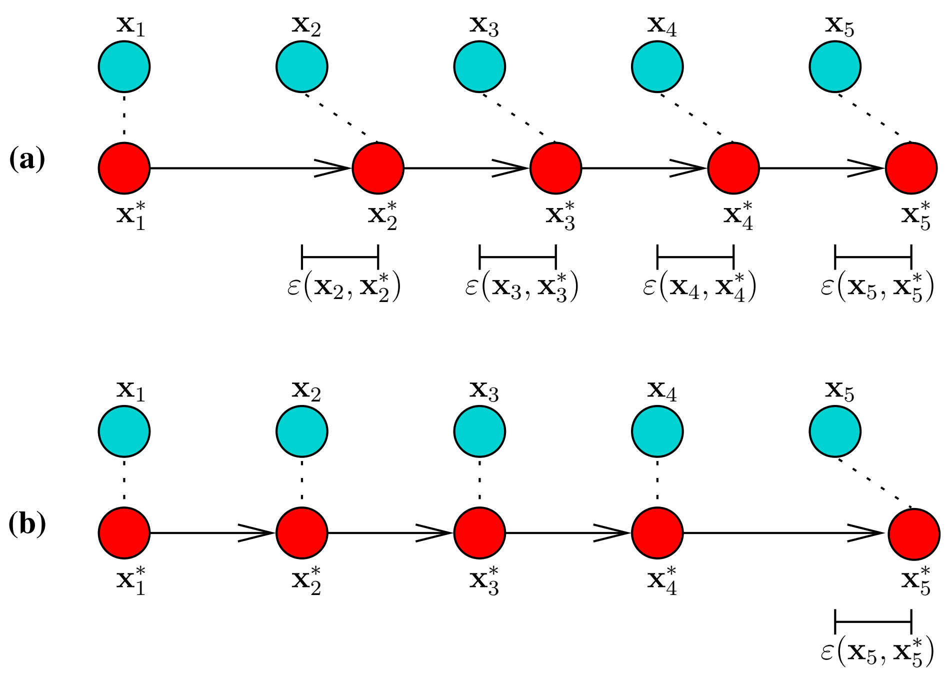

$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$
(a)
$\mathbf{x}_1^*$
$\mathbf{x}_2^*$
$\mathbf{x}_3^*$
$\mathbf{x}_4^*$
$\mathbf{x}_5^*$
$\varepsilon(\mathbf{x}_2, \mathbf{x}_2^*)$
$\varepsilon(\mathbf{x}_3, \mathbf{x}_3^*)$
$\varepsilon(\mathbf{x}_4, \mathbf{x}_4^*)$
$\varepsilon(\mathbf{x}_5, \mathbf{x}_5^*)$
(b)
$\varepsilon(\mathbf{x}_5, \mathbf{x}_5^*)$

Figure 8.15: This figure shows an example in which the metric in Equation 8.6 is suboptimal. The blue circles show the reference positions of the robot $\mathbf{x}_i^*$ while the red circles show the estimated positions of the robot $\mathbf{x}_i$. The non-zero errors $\varepsilon(\mathbf{x}_i, \mathbf{x}_i^*)$ are marked as an interval below the nodes. **(a)** shows a situation, where an error occurs early in the trajectory. The sum of the squared errors will be high, even though the SLAM system made only a small mistake for the pose in time step 2. **(b)** shows a situation, where an error occurs late in the trajectory. The sum of the squared errors will be low, even though the error for the pose in time step 5 is identical with the one made in (a) in time step 2. (Source: [74])

Up to now we did not give any details about the calculation of an error between two poses $\varepsilon(\mathbf{x}_1, \mathbf{x}_2)$. If we consider $\mathbf{x}_1$ and $\mathbf{x}_2$ to be six-dimensional vectors containing the 3D position and the 3D orientation, $\varepsilon(\mathbf{x}_1, \mathbf{x}_2)$ could be the sum of an error term for the translation $\varepsilon_{\text{Trans}}(\mathbf{x}_1, \mathbf{x}_2)$ and an error term for the rotation $\varepsilon_{\text{Rot}}(\mathbf{x}_1, \mathbf{x}_2)$, whereas the terms already include weighting factors. Yet, such a weighting between translation and rotation is non-trivial. It depends on the scale of the problem and for which application it is intended. We therefore propose to actually leave the two error terms without combining them into one:

$$\varepsilon_{\text{Trans}}(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{<i,j>\in\mathcal{E}} \varepsilon_{\text{Trans}}(\mathbf{d}_{i,j}, \mathbf{d}_{i,j}^*)^2 \quad \text{and} \tag{8.8}$$

$$\varepsilon_{\text{Rot}}(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{<i,j>\in\mathcal{E}} \varepsilon_{\text{Rot}}(\mathbf{d}_{i,j}, \mathbf{d}_{i,j}^*)^2. \tag{8.9}$$

Alternatively one can also use the sum of absolute errors instead of the sum of squared errors:

$$\varepsilon'_{\text{Trans}}(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{<i,j>\in\mathcal{E}} |\varepsilon_{\text{Trans}}(\mathbf{d}_{i,j}, \mathbf{d}_{i,j}^*)| \text{ and} \tag{8.10}$$

$$\varepsilon'_{\text{Rot}}(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{<i,j>\in\mathcal{E}} |\varepsilon_{\text{Rot}}(\mathbf{d}_{i,j}, \mathbf{d}_{i,j}^*)|. \tag{8.11}$$

It remains the question, which relative displacements are used for our error metric, meaning which tuples are included in $\mathcal{E}$. This depends on the properties of the SLAM algorithm

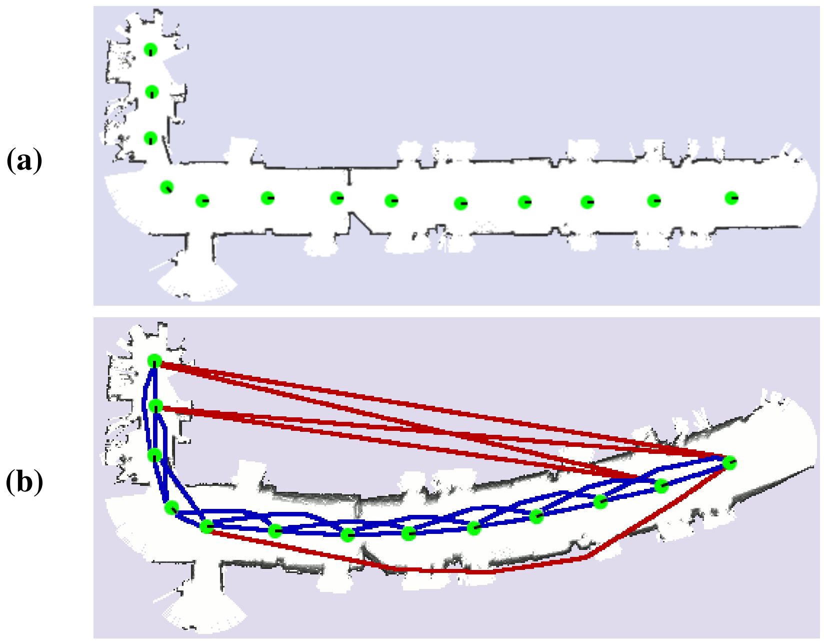


Figure 8.16: Example to demonstrate why global consistency is not always necessary. **(a)** A 2D laser map of a corridor created using the ground truth robot poses marked in green. **(b)** A map corresponding to (a), but created by a suboptimal SLAM approach. The trajectory and therefore the corridor in the map is bent. But a robot with a 2D laser scanner moving in the corridor could still use this map to localize and for navigation purposes since it is locally consistent. We marked some exemplary links between nodes, which could be used for evaluation purposes. Local links in blue and global links in red. Using only the blue links as $\mathcal{E}$ in our error metric would lead to a low error, which would reflect that the map is locally consistent. Adding also the red links to $\mathcal{E}$ would lead to a higher error, reflecting that the map is globally inconsistent.

one wants to highlight. For example, a robot generating blueprints of buildings should aim for global consistency to reflect the geometry of a building as accurately as possible. In contrast to that, a robot performing navigation tasks requires a map that can be used to robustly localize itself and to compute valid trajectories to a goal location. For this the map has only to be locally consistent, so that observations can be matched to corresponding parts of the map. Figure 8.16 gives a visual example for such a situation. It shows a bent corridor that, while being globally inconsistent, still could be used for localization and navigation purposes since it is locally consistent. It is now up to the user to put the focus on local consistency, by adding only relative displacements between nearby poses based on visibility, or to enforce global consistency by adding relative displacements of far away poses.

We will now present an experiment where we used our metric and the results of our SLAM method using aerial images to evaluate other SLAM approaches.

8.3.2 Exemplary Evaluation of SLAM Systems

We performed an experiment that uses this metric and a large scale dataset for which we created a globally consistent map with our approach using aerial images as a prior. The dataset was captured in the Freiburg University Hospital area and the created map and the used aerial images can be seen in Figure 8.17(a). This dataset consists of 2D and 3D laser range data obtained with one statically mounted SICK scanner and one mounted on a pan-tilt unit. The robot was steered through a park area that contains a lot of vegetation, surrounded by buildings. The robot was mainly steered along a bike lane with cobble stone pavement. The area is around $500\,\mathrm{m}$ by $250\,\mathrm{m}$ in size.

We selected three popular mapping techniques to be tested on this dataset:

- 2D Scan matching: 2D scans are incrementally matched against their immediate neighbors in time. This method is typically locally stable, but drifts over time because it does not incorporate loop closures. We specifically use the approach presented by Censi [20].
- A Rao-Blackwellized particle filter-based approach (RBPF): A particle filter where every particle builds its own 2D occupancy grid map. We use GMapping, as presented by Grisetti *et al.* [50].
- A graph-based SLAM system, as discussed in Section 2.4.1. We used the same system as for the comparison in Section 8.2.3, which follows the approach presented by Olson [95].

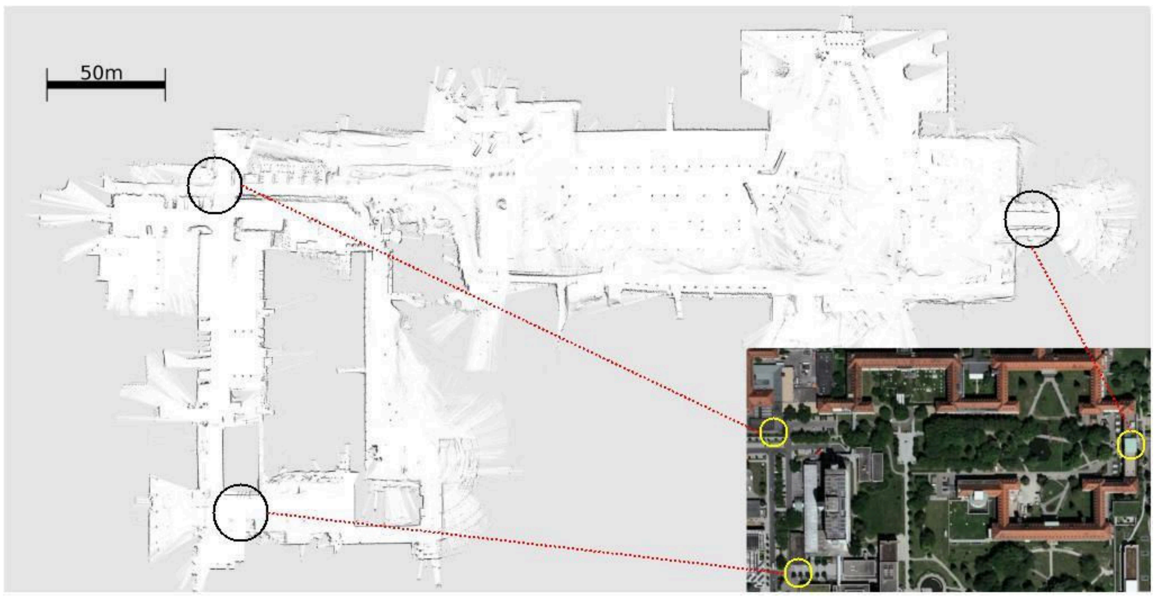
Figure 8.17(b), (c), and (d) depict the three mapping results. All maps are bent compared to the result using aerial images, which shows that the buildings in the upper right part should be at a right angle to the building in the lower left part. Visual inspection leads to the assumption that graph mapping outperforms the RBPF and scan matching. The RBPF was not able to close the large loop and therefore performed similar to scan matching. In most parts of the map, the local structure of the scan matcher and RBPF are comparable to the one of graph mapping. Significant differences can be observed in the areas where the robot reentered a known area (loop closures) since the data is not overlaid correctly.

We created two alternative sets of relations $\mathcal{E}$. One using only local relations based on the sensor range and one where the relations are generated for pairs of randomly sampled poses. This set should be used if global consistency is desired. The results for our error metric are presented in Table 8.2. Graph-based mapping clearly outperforms the other two approaches regarding local translational consistency. This is in accordance with what visual inspection on the map suggests (see Figure 8.17). Regarding the rotational error and the global consistency, the values of the three approaches do not differ much. The lack of global consistency of all maps is also apparent from the visual impression of the maps because they all are bent, which is caused by the lack of frequent loop closures in the dataset.

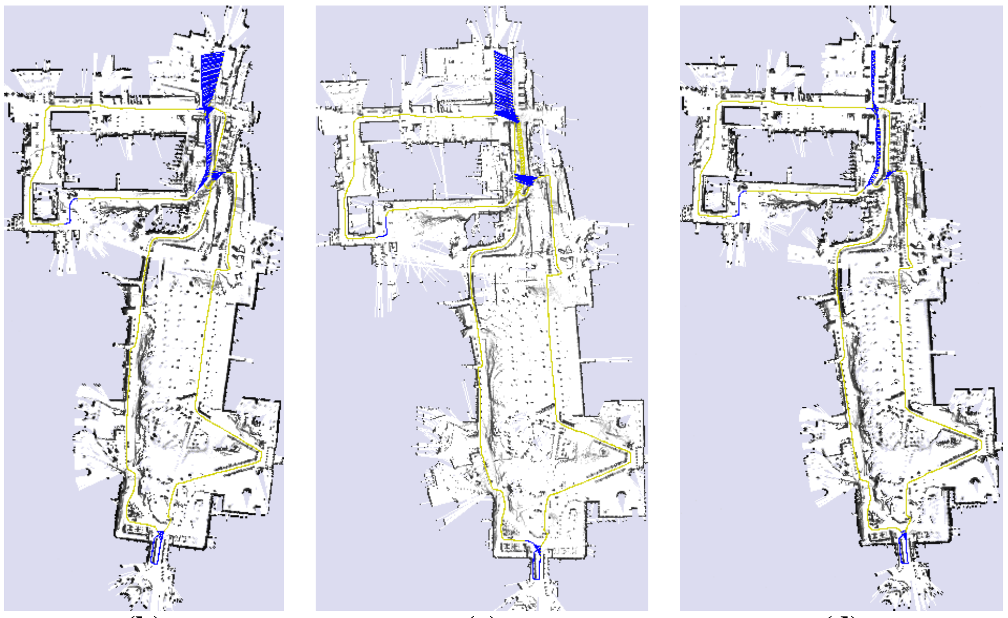
In summary we can say that our error metric reflects the impression one gets by visually inspecting the maps and that our SLAM approach using aerial images provides high quality trajectory and maps, which can be used to evaluate the quality of SLAM approaches lacking this additional information.

8.4 Related Work

While there exists a multitude of SLAM approaches in the literature (see Section 2.4), most approaches do not take any prior knowledge about the environment into account. Yet, there are some approaches that use prior knowledge for localization purposes. Korah and Rasmussen [69] find road structures in aerial images. The found road positions are then used to improve GPS paths using a particle filter, which increases the weight of particles according to how close they are to the found roads. Lee *et al.* [76] use road graphs from a given map for SLAM. They constrain the probabilistic model to place the vehicle on the roads and achieve better results than traditional FastSLAM. Leung *et al.* [78] presented an approach similar to ours in the sense that they used a particle filter to localize a monocular vision system in an aerial image based on extracted edges. Yet, their localization accuracy (several meters) is much lower than



(a)



(b) (c) (d)

Figure 8.17: **(a)**: This figure shows a map we acquired in the Freiburg University Hospital area and the used aerial image (Source: Google Earth). We marked some corresponding areas in the image and the map. **(b)**: The mapping result for the scan matching approach. **(c)**: The mapping result for the particle filter approach. **(d)**: The mapping result for the graph-based SLAM approach. The trajectories are shown in yellow. Local relations with a high error are connected by blue links. (Source: [74])

	Scan Matching	RBPF (50 particles)	Graph-based Mapping
Local relations			
Squ. trans. error (Equation 8.8) in m^2	2.79 ± 18.19	7.367 ± 38.496	0.053 ± 0.272
Abs. trans error (Equation 8.10) in m	0.434 ± 1.615	0.637 ± 2.638	0.143 ± 0.180
Maximum abs. trans. error in m	15.584	15.343	2.385
Squ. rot. error (Equation 8.9) in degree^2	10.9 ± 50.4	7.1 ± 42.2	5.5 ± 46.2
Abs. rot. error (Equation 8.11) in degree	1.3 ± 3.0	1.3 ± 2.3	0.9 ± 2.2
Maximum abs. rot. error in degree	27.4	28.0	29.6
Global relations			
Squ. trans. errors (Equation 8.8) in m^2	305.4 ± 518.9	288.8 ± 626.3	276.1 ± 516.5
Abs. trans. errors (Equation 8.10) in m	13.0 ± 11.6	12.3 ± 11.7	11.6 ± 11.9
Maximum abs. trans. error in m	70.9	65.1	66.1
Squ. rot. error (Equation 8.9) in degree^2	66.1 ± 101.4	64.6 ± 144.2	77.2 ± 154.8
Abs. rot. error (Equation 8.11) in degree	6.3 ± 5.2	5.5 ± 5.9	6.3 ± 6.2
Maximum abs. rot. error in degree	27.3	35.1	38.6

Table 8.2: Results for our SLAM evaluation metric for the three mapping methods and either local or global relations. The error terms are divided into translational and rotational elements. For each combination the average squared error, the average absolute error and the maximum error for the relations is given. (Source: [74])

what we achieve with either 3D laser scans or a stereo camera system. Früh and Zakhor [44] described an approach similar to ours, but using 2D laser scans. Edges are extracted from the aerial images and used as a 2D map for localizing the robot. As the authors mention in their paper and as we already discussed in Section 8.1.3, structures that differ substantially in a 2D scanner and a top-down view, like trees or overhanging rooftops, can have a negative influence on the localization results. These situations are better handled in our system using a 3D scanner. Additionally, our system is not limited to operate in areas where the prior is available, which enables our system to operate in mixed indoor/outdoor scenarios. When no prior is available, our algorithm behaves like the standard graph-based SLAM approach, which does not utilize any prior. Dogruer *et al.* [35] segment aerial images into different regions (buildings, roads, forests, etc.) and use MCL to localize a robot equipped with a 2D laser scanner within the segmented map. This approach strongly depends on the color distribution of the aerial images to segment objects that are then used in a virtual map to be matched against the 2D laser scan. Ding *et al.* [34] extract 2D corners from aerial images and match them with 2D corners extracted from 3D laser maps. Corresponding corners are found in a multi-stage process and are then used to estimate the position of the 3D scan in the aerial image, which is then used to color the 3D map. Chen *et al.* [21] build 3D maps consisting of individual point clouds, which are connected by spatial constraints. The authors then define a vector field on an aerial image, which simulates forces that pull the point clouds in directions that maximize the overlay of extracted edges. They then use an energy minimization technique to merge the prior information from aerial images and mapping procedure. Parsley and Julier [99] presented a method to incorporate a heterogeneous prior map into an Extended Kalman Filter for SLAM, which introduces an error bound when the robot moves in open-loop. Montemerlo and Thrun [89] presented an approach to introduce a prior into a SLAM system, but using GPS instead of positions based on localization in aerial images. Due to the large noise that regularly affects GPS measurements, this prior can lead to larger errors in the estimation of the trajectory and therefore the resulting maps.

Lu and Milios [84] originally introduced the concept of graph-based SLAM, thereby laying the foundation for both our SLAM system using a prior and the evaluation metric for SLAM systems we proposed. Kümmerle *et al.* [70] created the *General Graph Optimizer* (g^2o), which provides a convenient possibility to solve arbitrary graph-based optimization problems, like the one we presented in Section 8.1.5.

Benchmarking of systems from specific datasets has reached a mature level in the vision community. There is a large supply of freely accessible data bases on the Internet, often together with proposals for performance measures to be used for comparisons. The topics include, for example, image annotation [140], range image segmentation [59], and stereo vision correspondence algorithms [115]. These image databases provide ground truth data [115, 140], tools for generating ground truth [140] and computing the scoring metric [115], and an online ranking of results from different methods [115] for direct comparison. Compared to this, equivalent offers for mapping approaches in robotics are rare. While there are some well-known web sites providing datasets [14, 62] and algorithms [122] for mapping, they often do not provide ground truth data or recommendations on how to compare different maps in a meaningful way. Amigoni *et al.* [2] provided some suggestions how mapping benchmarks should be approached. They, e.g., stress the usage of publicly available datasets and that system parameters should be clearly stated and explained. For a comparison with a ground truth map they suggest to use the Hausdorff metric, which, unfortunately, is very susceptible to outliers. Wulf *et al.* [150] proposed an MCL in an artificial map, e.g., created from data obtained from a land registration office, whereas the matching is manually checked by a human. The authors plot the error for each scan over time both as the distance in translation and in the angle. In addition the standard deviation and maximum error for the trajectory is provided. This might overrate small errors in the beginning of the trajectory since it mainly checks global consistency, as discussed in Section 8.3.1. Balaguer *et al.* [6] propose to use the USARSim robot simulator to test SLAM approaches in a virtual environment. While they show that SLAM approaches behave similarly in simulated environments and the real word, they did not propose a quantitative measure for the evaluation and just used visual inspection.

8.5 Conclusions

In this chapter, we presented an approach to incorporate prior information in form of aerial images into a graph-based SLAM approach. This enables us to calculate highly accurate trajectories, which are both globally and locally consistent. Our method uses line-based feature extraction methods to define a sensor model for 3D scans and stereo vision data, which enables our system to identify corresponding areas in an aerial image. Compared to the previous chapters in this thesis, where point features were employed to detect corresponding areas in homogeneous data (3D scans), we now required a method to detect similar structures in heterogeneous data, namely between 3D scans and visual images, as well as between visual images with substantially different perspectives (frontal stereo cameras and top-down aerial images).

We tested our method in numerous experiments, including datasets from mixed indoor and outdoor environments, leading to the conclusion that this system can produce highly accurate maps and outperforms state-of-the-art SLAM systems that do not use prior information. This led us to the idea that the outcome of our method could be used to evaluate such SLAM systems, for which we provided an evaluation metric, which is derived from the idea of graph-based SLAM error functions. We demonstrated the usage of this metric for three exemplary SLAM systems, which we evaluated on a large-scale outdoor dataset.

In the following chapter we will present a complete robotic system, which also uses components developed in the context of this thesis. This robot, which is able to autonomously navigate in pedestrian areas, was presented to the press in a large-scale demonstration, which required all components to work robustly over extended time periods.

Chapter 9

Safe Autonomous Navigation in Crowded Pedestrian Areas

Autonomous navigation is a very challenging problem, especially in populated environments that are not specifically designed for a robot. While there has been much progress in robot navigation, most systems are restricted to indoor environments, unpopulated outdoor environments, or road usage with cars. Urban pedestrian areas are highly complex and dynamic, thereby creating numerous challenges for an autonomous robot. In this chapter we present a robotic system, which is able to navigate autonomously in populated urban areas. It is designed to operate in mixed in- and outdoor environments and to safely navigate from one point in a city to another. We presented this system to a large audience, including nation-wide German television, in a large-scale demonstration, which required all components to work robustly over an extended time period. Specifically, in 2012 our robot autonomously navigated from our University Campus over a 3.3 km long route to the city center of Freiburg, while being accompanied by a large group of onlookers and members of the press. In this chapter we will describe the different components we had to implement to make such a system possible. This includes a mapping system for large environments that do not fit into the main memory, and a module for planning and robust localization working on these maps. Our main focus will be on the analysis of raw 2D and 3D laser data, mainly in context of a traversability analysis dealing with 3D obstacles and vegetation, as well as dynamic 2D obstacles.

* * * *

Up to now we presented a number of systems, which address different perception problems in the area of robotics. We will now make the step to a physical robot solving an ambitious task. The step from the creation of software modules designed to address isolated problems to a complete robotic system designed to work in real world environments is hard. A large number of modules have to be integrated to work together on a specific hardware. The robustness of each individual module is vital since each of them can cause the complete system to fail at its task. During the creation of this thesis, the author was part of a team working on such a system, with the goal to create a pedestrian assistant robot. This was in the context of a project funded by the European Commission, namely EUROPA [38], which is an acronym for the EUropean RObotic Pedestrian Assistant. The robot resulting from this project is able to travel in crowded environments, interpreting its various sensor inputs in a meaningful way, while communicating with human users in different fashions and providing useful services to them. These services include tour guide scenarios as well as fetch and delivery tasks. In this project the involved members of the University of Freiburg were mainly concerned with the navigation system, which is the topic of this chapter. We will present the complete system, whereas we put special focus on the analysis of raw sensor data, mainly 2D and 3D laser measurements, resulting in high level features that are then passed to the other modules of the navigation system.

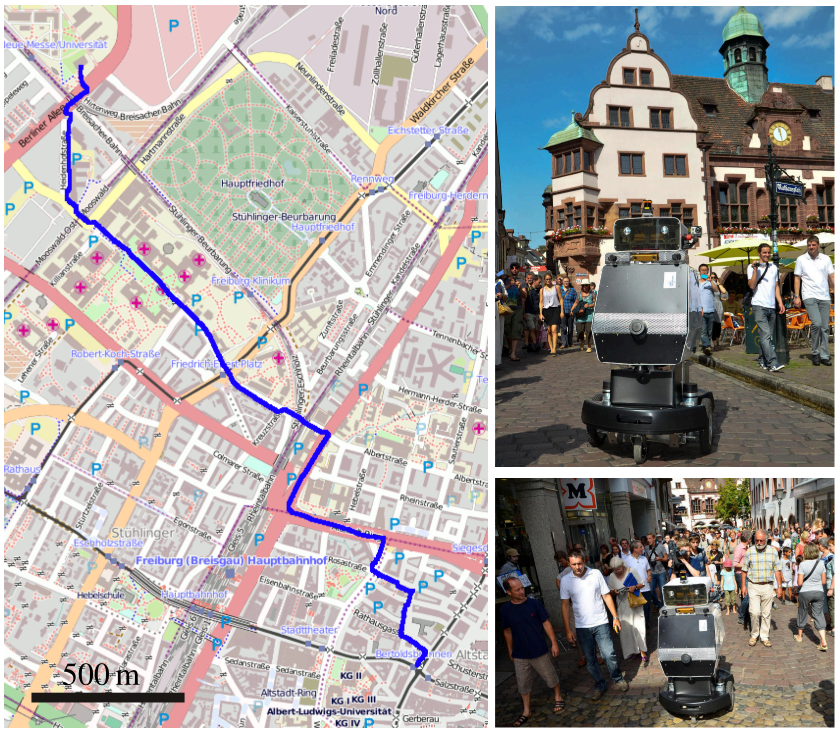
There has been substantial progress in robot navigation in recent years. Most systems are restricted to indoor environments [16, 134] or unpopulated outdoor environments, like deserts [23, 136, 146]. Road usage with autonomous cars has also shown a lot of progress [47, 91, 147]. Yet, there has been a surprisingly low number of systems that operate in populated pedestrian areas (e.g., [8]). Such systems have to cope with challenges like complex three-dimensional settings and highly dynamic scenes paired with unreliable GPS information.

In this chapter we will present our robot, which we nicknamed *Obelix*. It is designed to navigate in urban pedestrian areas like city centers. Next to the hardware components we will describe several software modules, which together form the complete navigation system. We will present a mapping system designed for large environments, storing the maps in a hierarchical fashion since the complete map might not fit into the main memory, a path planner and a localization module working on the aforementioned maps, and a module for the traversability analysis, which detects 3D obstacles and vegetation, as well as dynamic 2D obstacles. We will motivate our design decisions, critical aspects, as well as limitations of the current setup. We will then describe a large scale experiment where the robot autonomously navigates from our University Campus over a 3.3 km long route to the city center of Freiburg, accompanied by a large group of onlookers and members of the press. We believe that the completion of such a hard task shows the robustness of our complete system, thereby giving credibility to every individual module. Figure 9.1 shows the driven trajectory on a map and two pictures captured during the run.

The remainder of this chapter is organized as follows: After presenting the used robot platform and its sensors in Section 9.1, we explain the individual software modules in Section 9.2. This is followed by a discussion of current restrictions of the system in Section 9.3 and the experimental evaluation in Section 9.4. We discuss related work in Section 9.5. Section 9.6 concludes the chapter.

9.1 The Robot Platform

The robot platform was developed and build during the project by an industrial partner. It uses a differential drive and can move up to 1 $\frac{\text{m}}{\text{s}}$. It is equipped with flexibly mounted caster wheels in the front and the back, enabling it to climb steps of 3 cm and lower. Whereas this is sufficient to negotiate a lowered pedestrian sidewalk, it can not go up and down normal curbs, meaning that it has to avoid such larger steps. Its footprint is 0.9 m $\times$ 0.75 m and it is around 1.5 m tall. The main sensor input for our navigation system comes from laser range scanners. Two SICK LMS 151 are mounted horizontally in the front and in the back of the robot. These laser scanners have a 270° field of view, yet the view of the front laser is restricted to 180°. The remaining 90° are reflected by mirrors to observe the ground in front of the robot. Another SICK LMS 151 laser scanner is mounted in the front, tilted 70° downwards. An additional Hokuyo UTM-30LX laser scanner is mounted on a servo motor on the top of the head of the robot. The robot is also equipped with an XSENS Inertial Measurement Unit (IMU). The servo on the Hokuyo is used to always level the scanner horizontally, according to the pitch angle the IMU measures. This means that the scanner remains horizontal, even if the robot drives up or down a slope. This facilitates the mapping and localization procedure. Figure 9.2 shows the



(a) **(b)**

Figure 9.1: **(a)** The 3.3 km long trajectory our robot chose to go from our University Campus to the city center of Freiburg, overlain onto an OpenStreetMap* image **(b)** Two pictures of the robot taken during the experiment. (Source: [71])
*©OpenStreetMap contributors: http://www.openstreetmap.org/copyright

setup of the mounted laser scanners and depicts what they observe. Next to the laser scanners and the IMU, the platform is equipped with a high-end GPS sensor (a Trimple Pathfinder Pro) and a Bumblebee stereo camera. Please note that the latter is not used in the navigation system. We only use the images for visualization purposes.

9.2 The Navigation System

Our system relies on several different modules which work together to form the navigation system. To enable the different modules to communicate, we use a network-based inter-process communication tool (MOOS [94]). Using this, the different modules can exchange messages during runtime. Next to hardware drivers, user interfaces and other software providing basic functionalities, the relevant modules are:

- The mapping system: Our navigation system relies on maps of the environment to autonomously travel from one place to another. While this might seem like a drawback, the mapping procedure is actually quite efficient and one only has to drive the robot manually through the environment in question once to create a map. It took us, e.g., around three hours to map a 7.4 km long trajectory. Small modifications to the environment, like billboards or shelfs placed in front of shops, which are not contained in the map, can be handled by our system robustly.

- The localization system: While driving autonomously, the map is used by the localization system to match the current laser measurements against it to track the current position.

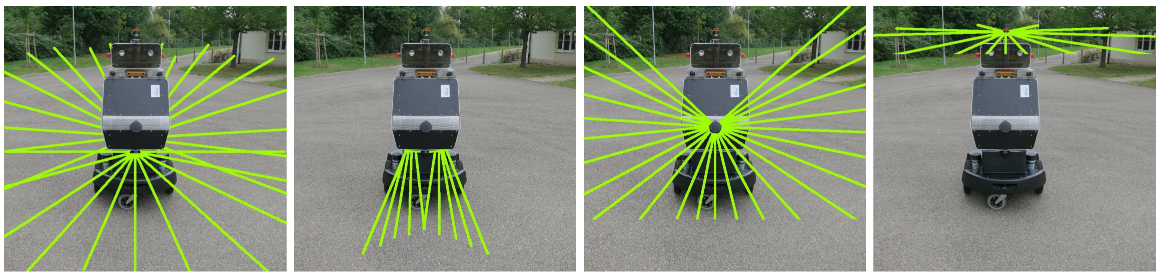


Figure 9.2: The laser sensor setup of our robot. The scan patterns of different lasers are visualized by the green lines. **(a)**: The robot is equipped with two horizontal SICK lasers, one 270° scanner to the back and one 270° scanner to the front, whereas only the middle 180° beams are used horizontally. The remaining beams are reflected onto the ground by mirrors, building a cone in front of the robot, as visible in **(b)**. **(c)** shows a downwards facing 270° SICK scanner in the front and **(d)** shows a small 270° Hokuyo scanner on a tilting servo on the head of the robot. The beams shown in **(b)** and **(c)** are mainly used for obstacle detection, whereas the beams shown in **(a)** and **(d)** are mostly used for localization and mapping purposes.

- The traversability analysis: The laser measurements are used to find obstacles that the robot can not drive over. This includes positive and negative obstacle, vegetation, and dynamic obstacles. This software is running both during mapping and during autonomous navigation. Thereby, stationary obstacles are integrated into the map, but they are also detected online (apart from vegetation), to be robust against changes in the environment.
- The path planner: The plan to reach a given goal point is computed based on the map and recently seen obstacles.

The entire navigation system runs on one standard quad core i7 laptop operating at 2.3 GHz. We will now describe the individual components in detail.

9.2.1 The Mapping Module

Our mapping software is based on a graph-based SLAM system (see Section 2.4.1) and is very similar to the one we described in Section 8.1.5. The constraints are generated based on the horizontal laser scanners of the robot. As proposed by Olson [95], we use a correlative scan-matcher for the incremental motion estimation and find loop-closures by matching the current scan against all former scans that are within a 3σ uncertainty ellipsoid, whereas false positives are filtered by spectral clustering. Compared to the system we presented in Chapter 8, we use GPS for the prior instead of aerial images. We decided to use the GPS measurements to decrease the complexity of our mapping system and because our navigation system does not require strict global consistency. As discussed in Section 8.3.1, locally consistent maps are more important for normal navigation tasks. In addition, our platform is equipped with a very accurate high-end GPS device, which only returns position estimates when the uncertainty is low. While this results in sparse measurements, it is enough to register our maps in a global coordinate frame and bound the accumulated error. In addition, we filter inaccurate GPS measurements using an outlier rejection method based on a robust cost function, namely the Pseudo Huber cost function [56]. First, the network is optimized using all GPS links. After optimization the $2\,\%$ of the GPS measurements with the largest residual are removed. This process is repeated five times, thereby removing about 10% of the GPS measurements. While this might also remove correct position estimates, we found that the results lead to much more accurate maps. Next to the increased map quality, using the GPS also registers the map in a global frame, which makes merging of maps and reasoning about positions in the map much more convenient.

After obtaining the trajectory from our graph-based SLAM approach, we perform mapping with known poses, meaning we build a map by accumulating the projected laser data into a 2D occupancy grid. Unfortunately, these maps can become huge regarding their memory requirements. We therefore do not keep the complete map in memory, but load sub-maps from the hard drive. For this purpose we use the graph structure in a way similar to what Konolige *et al.* [68] proposed. Every node in the map graph carries a sub-map, and whenever the robot travels from the vicinity of one node to another, it updates the current local map. We currently use a map size of $40\,\mathrm{m} \times 40\,\mathrm{m}$. The straight-forward way to create the submaps would be to just integrate all laser measurements that fall into the area into the map. This would be sufficient if our trajectory would be perfectly consistent. While the integration of GPS measurements bounds the global error, there might still be slight misalignments between close-by traversals, if no loop-closure was detected. This can happen, if there is limited visibility between the two poses, e.g., if there is a wall between them. In this case it is better not to merge the observations of the two different perspectives. The decision which observations are integrated into the local maps is therefore based on a distance determined with Dijkstra's algorithm on the graph. If a loop closure was found within the SLAM system, this will create a shortcut within the graph, whereas a missing loop closure leads to a higher distance. This prevents the integration of measurements, which might be inconsistent because of missing loop closures, thereby creating maps with a higher local consistency than the naive approach.

Each node in the map graph actually contains two different grid maps. One, as explained above, containing all the 2D laser measurements. These maps are used by the localization module (see Section 9.2.2). In addition, each node contains a traversability map, which incorporates all static 2D laser obstacles plus all detected non-dynamic obstacles from the traversability analysis (see Section 9.2.3). These maps are used for path planning (see Section 9.2.4).

9.2.2 The Localization Module

Our localizer is based on Monte-Carlo-Localization, similar to what we already presented in Section 8.1.2. Instead of an aerial image we use the local map corresponding to the closest visible node. The visibility constraint is relevant because the closest node might be on the other side of a wall, which could mean that the map does not have a substantial overlap with the current observations. Each of the 1,000 particles we use in our implementation is weighted according to how well the current laser measurements fit into the occupancy grid map, whereas the endpoint model [135] is used for this evaluation.

9.2.3 Traversability Analysis

The correct identification of obstacles is a critical component for autonomous navigation with a mobile robot. Given our platform, we need to identify obstacles having a height just above 3 cm. Such obstacles are commonly described as positive obstacles, as they stick out of the ground surface the robot travels upon. In contrast to that, negative obstacles are holes or downwards steps deeper than 3 cm. Such obstacles also need to be avoided by the robot. In the following, we describe the module which detects positive and negative obstacles while at the same time allowing the robot to drive over manhole covers and grids, which might normally be falsely classified as negative obstacles. Furthermore, while navigating in urban areas, the robot may

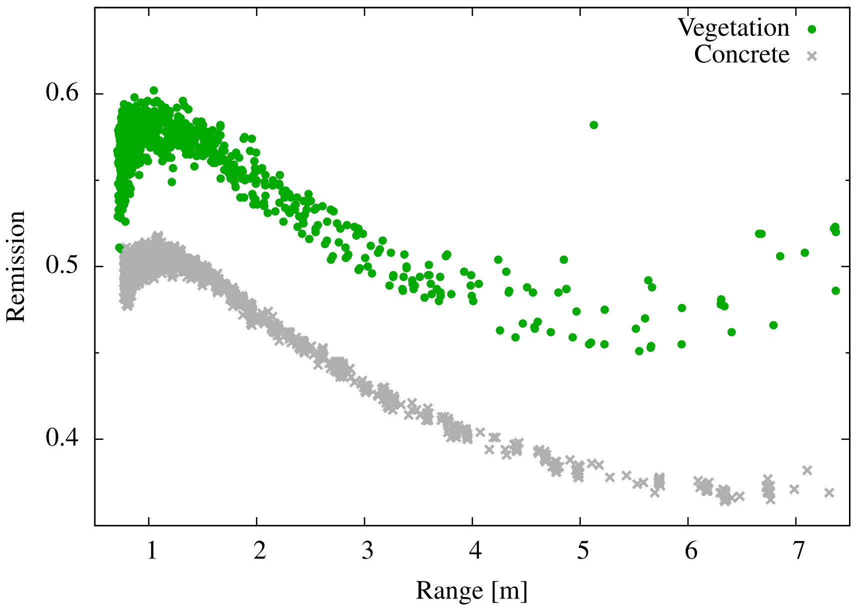


Figure 9.3: Range and remission values observed by our robot for either concrete or grass surfaces. The two plots appear to be easily separable by a non-linear function. (Source: [71])

encounter other undesirable surfaces, such as lawn. Here, considering only the range data is not sufficient, as the surface can appear to be smooth. Since our platform cannot safely traverse grass areas, where it might easily get stuck due to the small caster wheels, we also have to identify such areas to allow the robot to avoid them and thus to reduce the risk of getting stuck while trying to reach a desired location.

Vegetation Detection

In our implementation, we detect flat vegetation, such as grass, which cannot be reliably identified using only range measurements, by considering the remission values returned by the laser scanner along with the range [151]. We exploit the fact that living plants show a different characteristic with respect to the reflected intensity than the concrete surface found on streets. Unlike Wurm *et al.* [151], we do not use a tilting laser scanner for the vegetation detection, but a fixed downwards facing scanner (see Figure 9.2(c)). This simplifies the terrain classification process since the incidence angle of the laser beam hitting the presumably flat ground correlates with the measured range. Figure 9.3 shows the difference in remission values for concrete and grass. It shows that the two classes are easily separable using a non-linear function. We therefore fit a function to the lower boundary of the remission values belonging to vegetation to reduce the risk of false positive detections. This method is very efficient regarding runtime and proved to be sufficient to identify areas of vegetation, which should be avoided by our robot.

Tracking Dynamic 2D Obstacles

To detect moving obstacles in the vicinity of the robot, like pedestrians or bicyclists, we employ a blob tracker based on the 2D range scanner data. We first accumulate the 2D laser readings in a 2D grid map for a short time window (about 100 ms in our current implementation). In addition to that, we keep a history of these maps for a larger amount of time (about 1 s). To find the dynamic elements in the map, we compare the current map with the oldest in the history and mark the obstacles that only appear in the newer map as dynamic. Then, we filter out those obstacles that appear to be dynamic but that were occluded in the older map and are therefore probably false positives. In the next step we cluster the dynamic obstacles into blobs using a region growing approach. Then, we find corresponding blobs in the preceding map using a nearest neighbor approach (rejecting neighbors above a predefined distance). Based on the mean positions of the corresponding blobs we estimate velocities and bounding boxes that are aligned to the movement direction.

While this method is relatively simple (and occasionally creates false positives and sometimes wrongly merges multiple moving objects into one), it proved to be highly effective for the city navigation task. It can be calculated very efficiently and provides a sufficient movement prediction for avoidance purposes.

Detection of 3D Obstacles

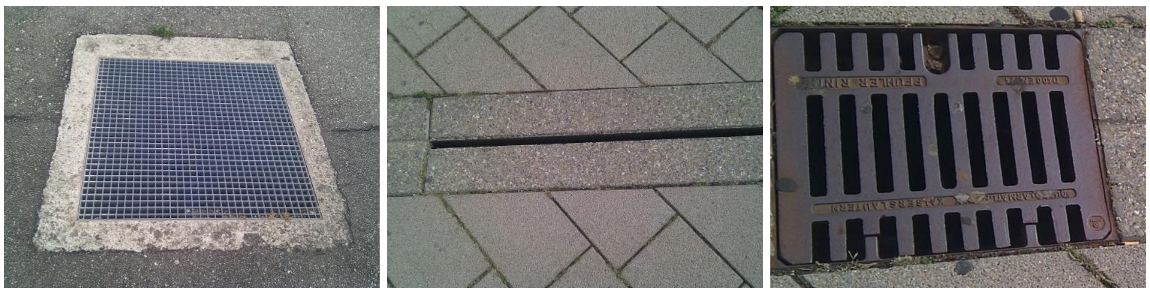
Unfortunately, not all obstacles that might block the path of the robot are visible in the horizontal laser scans. For this reason, we implemented a module that analyzes the scan lines captured by the downwards facing laser and the mirrored laser beams in front of the robot (see Figure 9.2(b) and (c)). These lasers provide 3D information about the environment when the robot is moving.

In a first step, we perform a filtering on the raw scans to get rid of some false measurements. This especially targets the spurious points that are typically returned at the boundary of objects in the form of interpolated point positions between the foreground and the background. These points might create false obstacles. To detect them, we check for sudden variations in depth which are noticeable as very small viewing angles from one point in the 2D scan to its immediate neighbors. Those boundary areas are erased from the scans before performing the obstacle detection procedure.

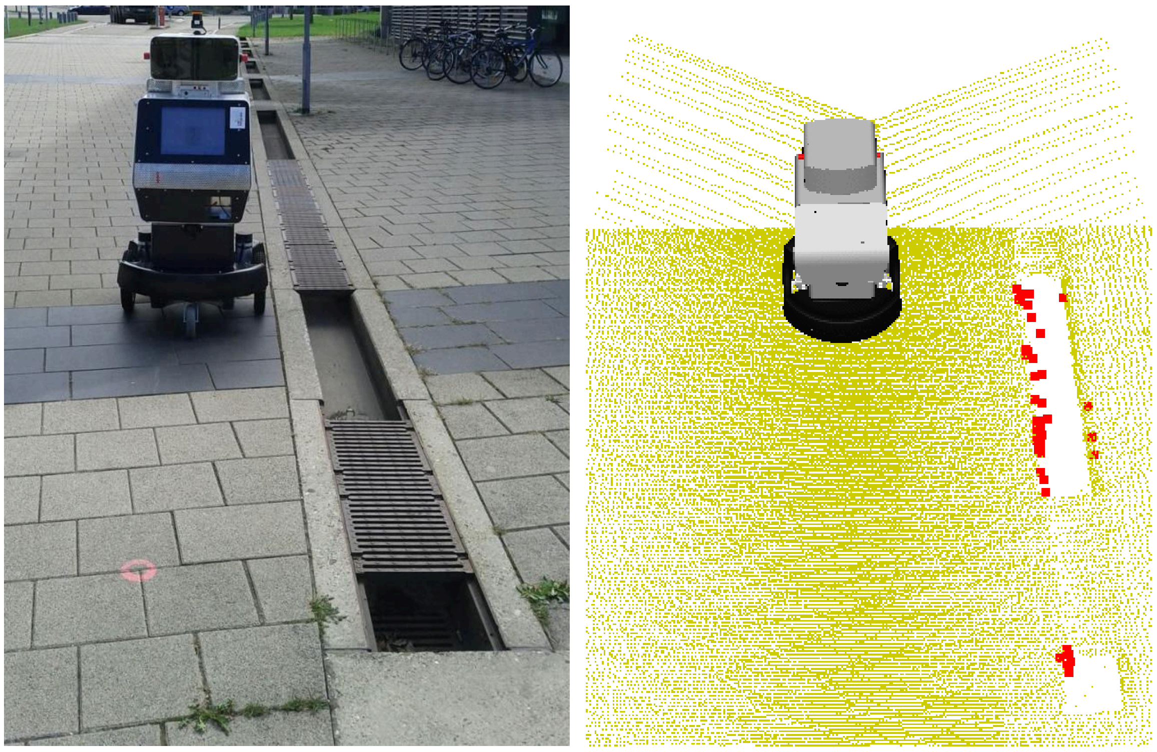
The main part of the obstacle detection process is done by analyzing only single scan lines, instead of the point cloud which is accumulated during driving. To decide if points in these scan lines measure flat ground or an obstacle the robot cannot traverse, we analyze how straight the scan lines are and if there are significant discontinuities in there because a flat ground would lead to straight scan lines. To be robust to noise, we use polynomial approximations for small segments of the scan lines and analyze their incline. Every point that lies in a segment which has an incline above a maximum value ($10\,^{\circ}$) and a height difference above a maximum allowed step height (3 cm) is reported as a potential obstacle. This heuristic proved to be very effective and has the advantage of being very efficient for single 2D scans, without the need of integration over a longer period of time. It also does not require information about position changes of the robot, which would be a source of considerable noise. In addition to that, there are no strong requirements regarding the external calibration parameters of the lasers.

Unfortunately, there are rare cases where this procedure fails. The main source of failures are false positives on manhole covers or gutters in the ground. Example images can be seen in Figure 9.4(a). Since some laser beams go through the structure and some do not, they appear to be negative obstacles. We implemented a heuristic to detect those cases by identifying areas with small holes. For this purpose, we extended the method described above and build a height map from the accumulated scan points while driving. For every scan point, we check if it lies substantially below the estimated height in both directions. This indicates a small hole. Obstacles close to such holes are ignored, if they are below a certain height (10 cm). This approach proved to provide the desired reliability for different settings in which the naive approach would have reported non-traversable negative obstacles. See Figure 9.4(b) for an example.

For every positive obstacle detected by the approach above, we check if this obstacle also has a corresponding obstacle in its vicinity in the 2D scans from the horizontal lasers. If not,



(a)



(b)

Figure 9.4: (**a**): Traversable structures that might be detected as negative obstacles by a naive method, because some laser beams can go through them. (**b**): Example case for the obstacle detection module. While the small canals on the robot's right side are classified as negative obstacles, the gutters are identified as traversable even though there are laser measurements going through the holes.

the corresponding 3D position is reported as a 3D obstacle. If yes, it is considered to belong to the 2D obstacle and only stored for a short amount of time. The reason for this is that our sensor setup does typically not allow us to reobserve a 3D obstacle in regular intervals since it is just seen once while driving by. Therefore, we have to keep the 3D obstacles in the map for an indefinite amount of time. On the other hand, obstacles observed in the 2D scanners can be reobserved and therefore do not have to be kept for a long time. This procedure prevents most dynamic objects (those that are also visible in 2D) from trapping the robot because it does not notice their disappearance. An example for all the different obstacle types we detect can be seen in Figure 9.5.

Vibration Based Ground Evaluation

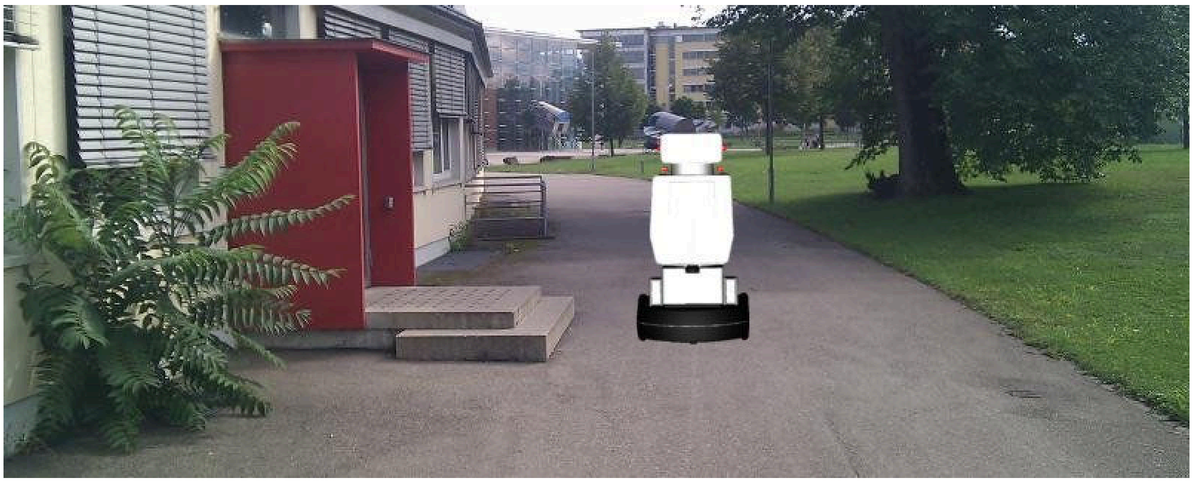
While the approach described above allows the robot to identify objects that need to be avoided, the ground surface itself needs to be taken into account when driving autonomously. Cobble stone pavement, which can typically be found in the centers of old European cities, leads to a substantial vibration and shaking of the platform. Hence, we consider the measurements provided by the IMU to control the speed of the platform based on the current vibration. If the vibration exceeds a certain limit, the maximum allowed velocity of the platform is gradually decreased. As the accuracy of the laser sensors is not sufficient to classify the smoothness of the surface, the robot has no means to identify whether the surface allows driving fast again without inducing vibrations. Hence, we greedily increase the maximum velocity again after a short delay.

9.2.4 The Path Planning Module

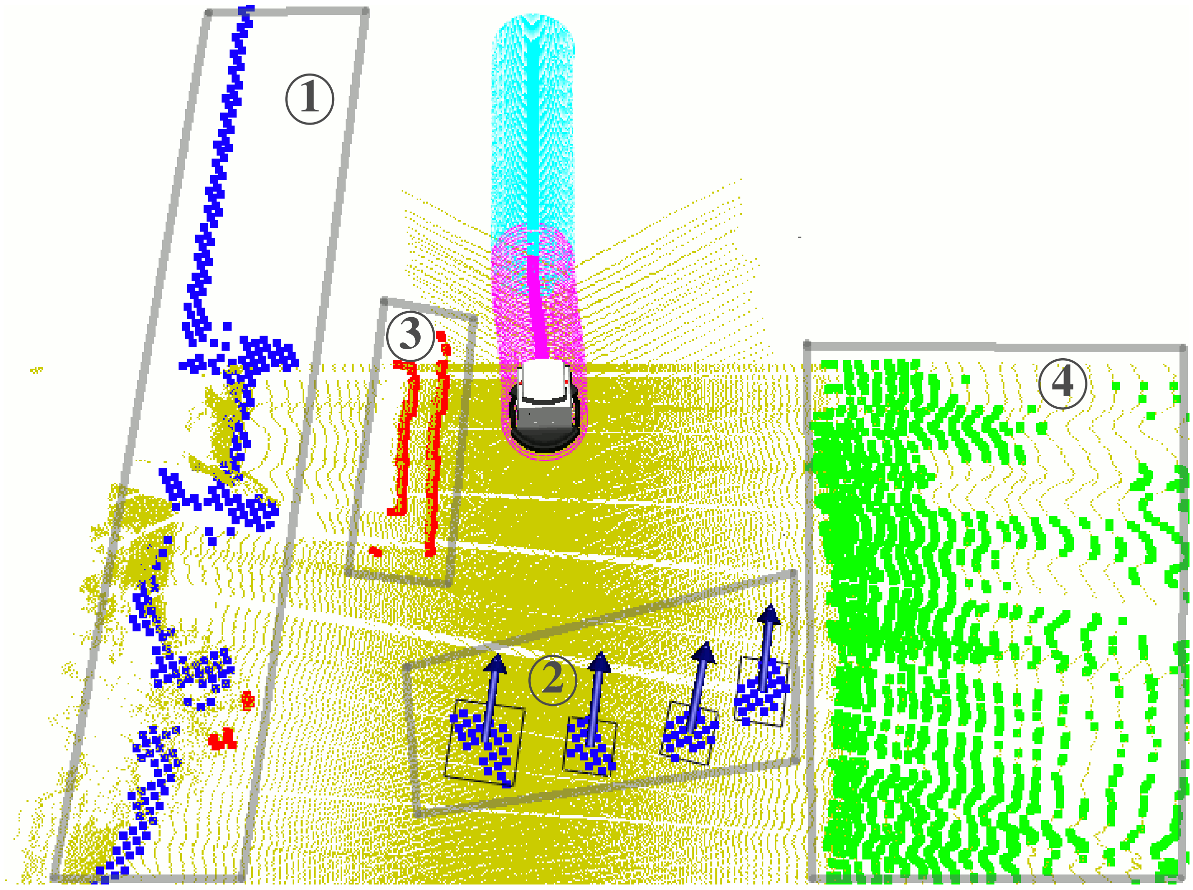
The path planning module works in three layers. At the highest layer the path in planned on a graph structure, whereas the nodes are identical with the nodes of the map graph. The edges are determined based on a reachability condition. For this purpose the system performs a pre-proccesing step once. It evaluates for every pair of nodes in the graph, if the robot can drive directly from one to the other according to the underlying occupancy map. If one node is reachable from the other node, an edge connects the two nodes, whereas the edge is assigned a cost proportional to the length of the trajectory connecting the two nodes. In addition, every cell in the map of a node receives a cost proportional to the length of the shortest path from the cell to the middle of the map (the node position), whereas unreachable positions receive a cost of infinity. The global planning can be performed efficiently on the graph using this information in an A^*-planner. The outcome is a list of nodes in the graph, which mark the way to reach the goal. The second planning layer calculates a trajectory inside of the current local traversability map (which contains the traversability information from the map plus newly detected obstacles) using Dijkstra's algorithm. If the goal point lies inside of the map, it calculates the shortest path there. If it lies outside of the map, the next node position in the global plan is the goal. This trajectory is used to create goal points for the lowest planning layer, which actually calculates the motion commands. This low-level planner was developed by Rufli *et al.* [103].

Using this hierarchy, we loose the optimality of the computed paths, but Konolige *et al.* [68] reported that the resulting paths are typically only 10 % longer, while the runtime requirements are several orders of magnitude lower.

Since the global planner works only on the map structure, the robot might get stuck when an unforeseen obstacles completely blocks its path. This means it cannot find an alternative route inside of the current local map, e.g., because of a new construction site or if cars block the



(a)



(b)

Figure 9.5: **(a)**: This photo shows an example scene. **(b)**: This image visualizes the different kinds of detected obstacles in the scene shown in (a), whereas four people are walking behind the robot. Blue points mark obstacles that are visible in the horizontal 2D laser scanners (areas ① and ②). Red points mark 3D obstacles that are visible in the downwards facing laser beams, but not in the 2D laser beams (mainly area ③). Green points mark the detected vegetation/grass (area ④). The black boxes with the arrows mark detected dynamic obstacles (area ②). The remaining yellow dots visualize the accumulated point cloud from the laser measurements.

walkway. To solve this issue, our system identifies the edges in the planning graph that are not traversable. These edges are then temporarily removed from the graph and a global replanning is triggered, thereby forcing the system to take an alternative route.

9.3 System Restrictions

There are some unsolved issues or restrictions when our robot is traveling autonomously. One of these is how the crossing of streets is handled. Appropriately dealing with such situations would require far-sighted sensors, such as radar or high frequency 3D laser scanners. The current setup does not allow a robust detection of fast moving objects. Our laser scanners have a maximum range of 50 m, but the actually measurable range depends on the surface in question. Shiny or dark objects are typically only visible at much shorter distances. A black car might be seen at distances below 20 m, which would mean that the robot would detect it too late and even then would only have very few measurement points to reason about. In addition, the car might be outside of the area the 2D scanners measure because of occlusions or because the robot is pitched slightly upwards or downwards. Crossings with pedestrian traffic lights are also not completely safe, even if a robust software to detect the state of the traffic light is available. For example, police cars or ambulances in action might still try to cross and trying to detect such situations would introduce new sources for errors. In our demonstration, we therefore enforced that the robot stops in front of every crossing and asks for a confirmation (pressing a button on the touch screen) before it crosses the street. The areas in question were marked in the robot's map in advance.

Also problematic are deformable objects in the environment, like leafs on the ground or twigs from shrubbery protruding into the walkway. While these objects are actually safe to drive over/through, the robot does not know this and treats them like every other obstacle. This means that he has to avoid them, which might block a path completely in the worst case.

Another restriction coming from the current sensor setup is that small dynamic obstacles below the horizontal lasers are typically not reobserved, meaning that a position on the ground is only seen once, while the platform is moving. Should an obstacle block the path there, the robot will try to move around it. But it will not notice if the obstacle actually moved away. Figure 9.6 shows such a situation, where birds were moving in front of the robot. To detect such situations, a 3D scanner, which repeatedly observes the ground in front of the robot, would be needed.

9.4 Experimental Evaluation

We will now describe a series of large scale experiments, where our robot traveled between our campus (the Faculty of Engineering of the University of Freiburg) and the inner city of Freiburg (2.3 km linear distance) multiple times. The map for this set of experiments was created in advance, by steering our robot manually along a 7.4 km trajectory to collect the data about the environment.

Overall, the robot mastered the way six times, driving an overall distance of around 20 km. During this time, there were only three failure cases:

- We stopped the robot in front of an overhanging obstacle, which we believed to be in a blind spot of the sensor setup.

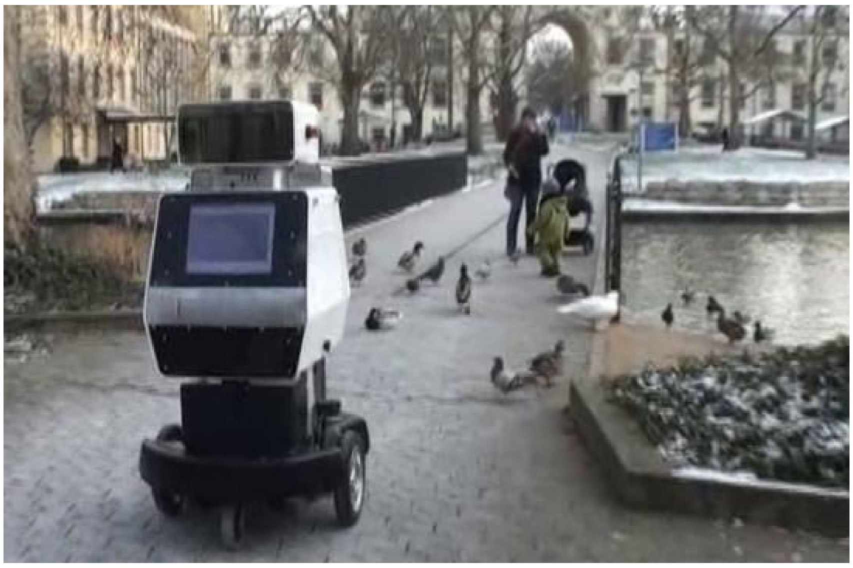


Figure 9.6: Birds moving in front of the robot. The ducks and pigeons are to small to be visible in the horizontal scanners. While the laser beams observing the ground will notice them and cause the robot to stop, it can not observe when they move away. This can create fake obstacles, which might block the robot's path completely. (Source: [71])

- The robot got stuck on a small bump in the road. Normally, the platform can traverse steps of up to 3 cm, but this requires a certain momentum. In this situation the robot stopped shortly before trying to climb the bump, which lead to the situation that the motors were not strong enough to master the climb. Since the software did not detect this situation, we had to move the robot back half a meter manually and restart.
- The localization failed once and had to be re-initialized. More details on this below.

For the last of these runs we invited the press and actually received a very strong echo from newspapers, radio, and television leading to a nationwide coverage in top-media and also international attention. During the event, a group of about 50 people (members of the press, University personal, and onlookers) followed the robot along its 3.3 km long path to the highly populated inner city. See Figure 9.1 for the driven trajectory and photos of the run. During this time, the robot stopped twice. Once the wireless emergency stop button was pressed by mistake, which was a human error, not a failure of the system. The second time, as already mentioned above, the localization system lost the correct pose and had to be relocalized manually. This was actually caused by the large audience. In an area with very few features (a large open area with only a few trees) usable for the localization system, people blocked the robot's view to the available landmarks (see Figure 9.7). When the accumulated position error was between one and two meters, the robot believed not to be on the sidewalk anymore and stopped moving. We had to manually reinitialize the localization to resolve this problem.

9.5 Related Work

Autonomous navigation in different crowded environments has been successfully addressed in the past. The robots RHINO [16] and Minerva [134] were used as interactive museum guides. This concept was extended to the RoboX multi-robot system by Siegwart *et al.* [117] for the Expo'02 Swiss National Exhibition. Gross *et al.* [52] presented a robot, which functioned as

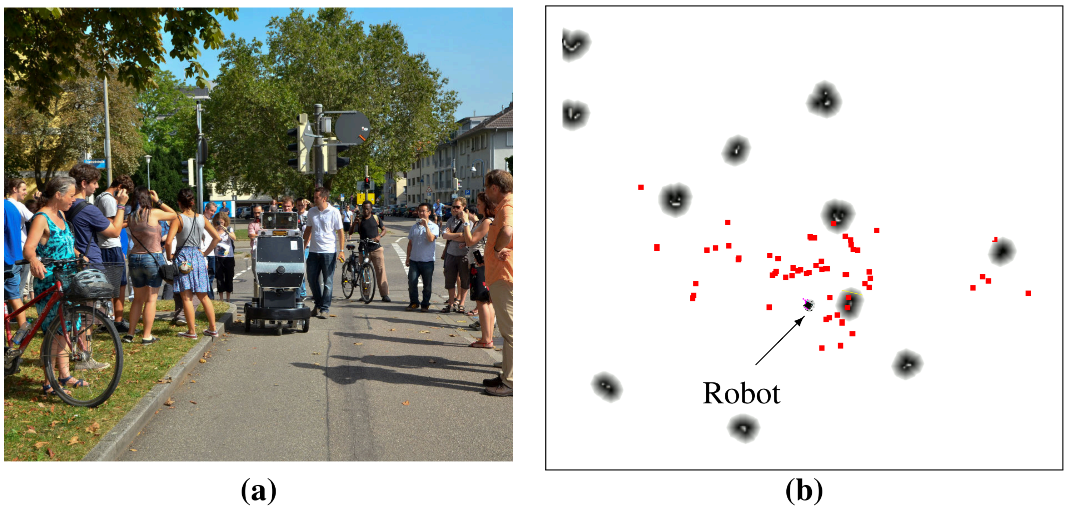


Figure 9.7: This figure shows the situation, when the localization system failed. **(a)** shows a photo of the situation. The robot is surrounded by onlookers. **(b)** shows the 2D distance map (gray) with which our system matches the current laser measurements (red) to determine its position. As can be seen, there are very few landmarks available for the localization, mostly stems of trees. Yet, nearly all laser observations fall onto the surrounding people, causing a mismatch with the model. (Source: [71])

a shopping assistant, helping customers to find their way in home improvement stores. All these systems were able to operate in highly populated environments. Yet, they are restricted to structured 2D indoor environments.

Recently, there have been many noticeable successes in the area of autonomous cars. During the DARPA Grand Challenge, autonomous cars moved over large distances through desert areas [23, 136, 146]. Later, in the DARPA urban challenge, several systems proved to be able to autonomously navigate in dynamic city street networks with complex traffic scenarios, even considering road traffic rules [91, 147]. Later, Google presented its fleet of self-driving cars [47], which are actually used in public traffic. In 2012 these cars already traveled multiple hundred thousand kilometers autonomously, without causing any accidents. They even achieved the legalization of autonomous cars in parts of the USA.

Compared to the approaches presented above, we focus on navigation in urban areas designed for pedestrians. Relatively few robotic systems have been developed for such environments. The most similar to our system was probably developed by Bauer *et al.* [8]. The Munich City Explorer operates fully autonomously and without a prior map. It interacts with pedestrians to get hints about the direction where it has to go. During this process it builds local maps and explores the environment online, without a global plan. Another related approach has been developed in the context of the URUS project [113]. It is concerned with urban navigation, but the focus is more on networking and collaborative actions as well as the integration with surveillance cameras and portable devices.

9.6 Conclusions

In this chapter, we presented a complete robotic system, designed to navigate through an urban environment like a pedestrian. Different modules, handling mapping, localization, planning, and traversability analysis, work together to allow our robot to successfully travel long distances in highly populated inner city areas. We performed extensive experiments, in which we repeatedly navigated along an over three kilometers long route in Freiburg, thereby showing the robustness of our system. These experiments received a lot of interest from the press, enabling us to spread the word about our research also to a non-expert audience.

In the context of this thesis, we discussed another perception problem in mobile robotics: the extraction of high level information (traversable versus non-traversable ground) from low level 3D and 2D laser data. This information is then used by the other modules to create traversability maps and plan save paths. We demonstrated that this information enabled our robot to safely navigate in urban environments, even with its restricted sensor setup, which only acquires 3D information while driving.

While the immediate target applications for our system are tasks like a robotic tour guide or fetch and deliver functions, the implemented navigation system could also provide help for wheelchair users. On a basic level, the traversability maps built with a system like ours could be used to support wheelchair users in choosing a path because they include information about the maximum step height to be traversed on a certain path. Going further, an automated wheelchair could partially support the user while driving or even navigate completely autonomously. We believe that a robustly working system like the one we presented in this chapter is a useful proof of concept, showing that this specific combination of existing technologies is able to solve a hard task and exceeds the current state of the art.

Chapter 10

Conclusions and Outlook

This work addresses a number of highly relevant techniques for advanced 3D perception in robotics. We presented innovative approaches for point feature extraction, object recognition, object and environment modeling, place recognition, localization in aerial images, and traversability analysis. All these techniques have in common that they help a mobile robot to understand its environment. This includes information like where it is, where it can go, what is around it, and how its surrounding is structured. We extended the state of the art in these areas and created useful tools to approach the above mentioned 3D perception problems.

First, we introduced the Normal Aligned Radial Feature (NARF), providing a novel keypoint extraction method together with a new type of descriptor for 3D range scans. We then presented several approaches that build upon this point feature type to solve hard perception problems. We began with an object recognition system, which uses NARFs to find similar structures between object models and 3D scenes to determine potential object positions. These positions are then evaluated based on spatial verification, meaning we described methods to evaluate how well an assumed object pose fits the actual sensor measurements.

The point cloud object models needed for our object recognition system were initially created manually by driving around an object with a mobile platform and merging several scans. To make the modeling process more convenient, we developed an approach for unsupervised object modeling. This system takes an unordered set of 3D range scans and identifies potential partial objects. It then searches for similar structures that might belong to the same object type, but seen from slightly different perspectives. The process to find potential overlapping object parts in the data is similar to our object recognition method, but adapted for incomplete data. If enough views of an object are available in the data, our system is able to create complete models without any manual user input.

The insight that typical human-made environments contain many repetitive structures, e.g., from identical objects in the environment, led us to the idea to detect such reoccuring elements. We then only need to describe each unique structure once and store where it appears. This enables us to reduce the complexity of 3D scenes substantially. We implemented a system that builds a dictionary of surface patches from a 3D environment and can express a scene by means of these patches. We were able to show that even a small dictionary can be used to accurately describe typical 3D scenes in a condensed form, with the additional benefit that this form of representation is semantically richer since it directly encodes similarities in the environment.

Up to this point, the scale of the structures our approaches were matching decreased steadily, from complete objects to partial objects, to surface patches. Yet, there are are also areas in robotics, where the detection of similarities between larges structures is relevant. In place recognition approaches, whole scans are matched against each other to determine if a robot already visited a certain area. This is useful for global localization purposes but also for loop closure detection in SLAM approaches. We presented such a system that can match individual 3D scans against a database of scans and reliably detects if the current observation fits one of the known places. In addition, our system calculates highly accurate relative poses between the corresponding scans.

The found links between 3D scans can be used in a SLAM system to create high quality 3D maps. However, if loop closures are infrequent, there will still be accumulated errors in the trajectory, which lead to global inconsistencies in the map. To get around this problem, we considered a source of information, which is nowadays freely available for most urban areas, namely aerial imagery. We developed a method to localize a robot in an aerial image, using 3D scans or stereo camera data. This information is then used as a prior in a mapping approach, leading to maps with increased local and global consistency.

All these approaches are mostly independent of the actual robotic platform used, apart from basic requirements regarding the sensor setup. But in practice, there are certain perceptual problems that depend strongly on the employed platform and approaches to solve them have to be adapted to the specific robot. This is especially true for the question where a robot can go in an environment. We introduced Obelix, a robotic pedestrian assistant, able to navigate in crowded urban environments. We described the complete navigation system, whereas our focus was on the traversability analysis. Such a complete system has high demands on the robustness of the individual components since every one of them can cause the robot to fail at its tasks. We showed in extensive experiments that Obelix is able to travel autonomously over large distances, avoiding positive and negative obstacles, lawn, and dynamic obstacles.

To summarize, in this work we provided the means to enable a mobile robot to accurately model its environment regarding spatial structure and reoccuring elements, determine its position by different means, recognize objects relevant for its tasks, and evaluate which areas it can safely traverse.

All the presented systems address hard perception problems and we evaluated them all in realistic scenarios and on real world data, receiving high quality results. Despite the encouraging outcomes, we of course plan to continue our work and further improve the capabilities of our systems, as discussed below.

Future Work

We focused mostly on 3D range data in this work. A promising next step is the integration of visual information, meaning either intensity and color information from cameras, or the remission information that laser scanners provide. While we used the latter for vegetation detection, this information could also be used to match textures on otherwise featureless (regarding the 3D composition) surfaces to match or distinguish structures. Especially given the recent advent of cheap RGB-D sensors, like the Microsoft Kinect or the Asus Xtion Pro Live, combining visual information and 3D structure to solve perception problems will be a worthwhile research topic.

Another useful extension to our current research would be a multi-scale extension of the NARFs. At the moment, the support size of the features, meaning the size of the area used for the keypoint detection and the descriptor calculation, is a parameter defined by the user. Yet, there are situations, where a fixed support size is suboptimal. For the object recognition, e.g., the selection of this size should depend on the size of the object, which of course complicates scenarios where we want to search for several objects of very different sizes. Simply performing the search at different scales would be computationally expensive. Therefore, finding a method to determine the support size based on the structure or encoding information for multiple scales into each feature would be worth investigation. This would also be advantageous for our environment modeling approach based on a dictionary of surface primitives. Currently, these primitives have a fixed size. Differently sized or scalable patches might enable us to produce even more compact representations. In the same context, the positions of the word instances in the scene are currently at fixed positions. It would be interesting to investigate if one can change instance positions in the process to acquire better models, without a substantial increase in runtime.

There is, of course, always room for improvements. Robotic perception is a complex and versatile area of research without clear limits regarding what is possible and what is not. Yet, with the content of this work, we hope to have provided some useful insights and tools to facilitate the way to more intelligent and more capable robots.

List of Figures

List of Tables

Bibliography

[1] P. Alliez and C. Gotsman. Recent advances in compression of 3D meshes. In *In Advances in Multiresolution for Geometric Modelling*, pages 3–26. Springer-Verlag, 2003.

[2] F. Amigoni, S. Gasparini, and M. Gini. Good experimental methodologies for robotic mapping: A proposal. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[3] D. Anguelov, R. Biswas, D. Koller, B. Limketkai, S. Sanner, and S. Thrun. Learning hierarchical object maps of non-stationary environments with mobile robots. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 10–17, 2002.

[4] A. Ansar, A. Castano, and L. Matthies. Enhanced Real-time Stereo Using Bilateral Filtering. In *Int. Symp. on 3D Data Processing, Visualization and Transmission (3DPVT)*. IEEE Computer Society, 2004.

[5] Y. Asahara, T. Yamamoto, M. Van Loock, B. Steder, G. Grisetti, and W. Burgard. Object recognition method, object recognition apparatus, and autonomous mobile robot, 2012. (PATENT).

[6] B. Balaguer, S. Carpin, and S. Balakirsky. Towards quantitative comparisons of robot algorithms: Experiences with SLAM in simulation and real world systems. In *IROS Workshop on Performance Evaluation and Benchmarking for Intelligent Robots*, 2007.

[7] 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.

[8] 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. *Int. Journal of Social Robotics*, 1:127–140, 2009.

[9] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2006.

[10] J. Behley, V. Steinhage, and A. Cremers. Performance of histogram descriptors for the classification of 3D laser range data in urban environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[12] M. Blum, J. T. Springenberg, J. Wülfing, and M. Riedmiller. A learned feature descriptor for object recognition in RGB-D data. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[13] L. Bo, X. Ren, and D. Fox. Unsupervised Feature Learning for RGB-D Based Object Recognition. In *Proc. of the Int. Symp. on Experimental Robotics (ISER)*, 2012.

[14] A. Bonarini, W. Burgard, G. Fontana, M. Matteucci, D. G. Sorrenti, and J. D. Tardos. Rawseeds a project on SLAM benchmarking. In *Proc. of the Workshop on Benchmarks in Robotics Research (IROS)*, 2006.

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

[16] 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 Conf. on Artificial Intelligence (AAAI)*, 1998.

[17] 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.

[18] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 34(7):1281–1298, 2012.

[19] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 8(6):679–698, 1986.

[20] A. Censi. Scan matching in a probabilistic framework. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2291–2296, 2006.

[21] C. Chen and H. Wang. Large-scale loop-closing by fusing range data and aerial image. *Int. Journal of Robotics and Automation (IJRA)*, 22(2):160–169, 2007.

[22] O. Chum, A. Mikulik, M. Perdoch, and J. Matas. Total recall ii: Query expansion revisited. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[23] L. Cremean, T. Foote, J. Gillula, G. Hines, D. Kogan, K. Kriechbaum, J. Lamb, J. Leibs, L. Lindzey, C. Rasmussen, A. Stewart, J. Burdick, and R. Murray. Alice: An information-rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics (JFR)*, 2006.

[24] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[25] M. Cummins and P. Newman. Appearance-only SLAM at large scale with FAB-MAP 2.0. *Int. Journal of Robotics Research (IJRR)*, 2010.

[26] Dataset: AASS-loop. http://kos.informatik.uni-osnabrueck.de/3Dscans.

[27] Dataset: Freiburg360 - 360° 3D scans of the Faculty of Engineering, University of Freiburg, Germany. http://ais.informatik.uni-freiburg.de/projects/datasets/fr360.

[28] Dataset: Hanover2. http://kos.informatik.uni-osnabrueck.de/3Dscans.

[29] Dataset: QuadrotorFreiburg3D - 3D scans of building 79 of the Faculty of Engineering, University of Freiburg, Germany, captured by a flying quadrotor robot. http://ais.informatik.uni-freiburg.de/projects/datasets/quadrotor079.

[30] A. Davison, I. Reid, , N. Molton, and O. Stasse. MonoSLAM: Real time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 29(6), 2007.

[31] 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.

[32] 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)*, 1998.

[33] P. Deutsch. GZIP file format specification version 4.3. RFC 1952 (Informational), 1996.

[34] M. Ding, K. Lyngbaek, and A. Zakhor. Automatic registration of aerial imagery with untextured 3D lidar models. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[35] C. U. Dogruer, K. A. Bugra, and M. Dolen. Global urban localization of outdoor mobile robots using satellite images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

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

[37] F. Endres, C. Plagemann, C. Stachniss, and W. Burgard. Unsupervised discovery of object classes from range data using latent dirichlet allocation. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[38] The European robotic pedestrian assistant. http://europa.informatik.uni-freiburg.de, 2009.

[39] R. Eustice, H. Singh, and J. Leonard. Exactly Sparse Delayed-State Filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2428–2435, 2005.

[40] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM (CACM)*, 24(6):381–395, 1981.

[41] J. D. Foley, A. Van Dam, K. Feiner, J. Hughes, and P. R.L. *Introduction to Computer Graphics*. Addison-Wesley, 1993.

[42] U. Frese, P. Larsson, and T. Duckett. A Multilevel Relaxation Algorithm for Simultaneous Localisation and Mapping. In *IEEE Transactions on Robotics (T-RO)*, volume 21, pages 1–12, 2005.

[43] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik. Recognizing objects in range data using regional point descriptors. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2004.

[44] C. Früh and A. Zakhor. An automated method for large-scale, ground-based city model acquisition. *Int. Journal on Computer Vision (ICCV)*, 60:5–24, 2004.

[45] D. Galvez-López and J. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics (T-RO)*, 28(5):1188 –1197, 2012.

[46] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann. Robust global registration. In *Proc. of the third Eurographics Symp. on Geometry processing*, page 197, 2005.

[47] Google self-driving car project. http://googleblog.blogspot.de/2012/08/the-self-driving-car-logs-more-miles-on.html, 2012. Accessed: 13.02.2013.

[48] 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.

[49] 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.

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

[51] G. Grisetti, C. Stachniss, and W. Burgard. Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, 10(3):428–439, 2009.

[52] 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.

[53] 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.

[54] S. Grzonka, G. Grisetti, and W. Burgard. A Fully Autonomous Indoor Quadrotor. *IEEE Transactions on Robotics (T-RO)*, 8(1):90–100, 2012.

[55] J. Gutmann and K. Konolige. Incremental Mapping of Large Cyclic Environments. In *Proc. of the IEEE Int. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, 1999.

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

[57] 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.

[58] E. Herbst, X. Ren, and D. Fox. Rgb-d object discovery via multi-scene analysis. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[59] A. Hoover, G. Jean-Baptiste, X. Jiang, P. J. Flynn, H. Bunke, D. B. Goldgof, K. K. Bowyer, D. W. Eggert, A. W. Fitzgibbon, and R. B. Fisher. An experimental comparison of range image segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 18(7):673–689, 1996.

[60] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America. A*, 4(4):629–642, 1987.

[61] A. Howard. Multi-robot mapping using manifold representations. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 4198–4203, 2004.

[62] A. Howard and N. Roy. Radish: The robotics data set repository, standard data sets for the robotics community, 2003. http://radish.sourceforge.net/.

[63] Q. Huang, S. Flöry, N. Gelfand, M. Hofer, and H. Pottmann. Reassembling fractured objects by geometric matching. *ACM Transactions on Graphics (TOG)*, 25(3):569–578, 2006.

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

[65] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 21(5):433–449, 1999.

[66] S. Julier, J. Uhlmann, and H. Durrant-Whyte. A new Approach for Filtering Nonlinear Systems. In *Proc. of the American Control Conf. (ACC)*, pages 1628–1632, 1995.

[67] G. Kim, C. Faloutsos, and M. Hebert. Unsupervised modeling of object categories using link analysis techniques. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[68] 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.

[69] T. Korah and C. Rasmussen. Probabilistic contour extraction with model-switching for vehicle localization. In *IEEE Intelligent Vehicles Symp. (IVS)*, pages 710–715, 2004.

[70] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A General Framework for Graph Optimization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[71] 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. Accepted for Publication.

[72] R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large scale graph-based SLAM using aerial images as prior information. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[73] 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.

[74] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Journal of Autonomous Robots*, 27(4):387–407, 2009.

[75] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains: Functions and integration. *Int. Journal of Robotics Research (IJRR)*, 21(10-11):917–942, 2002.

[76] K. W. Lee, S. Wijesoma, and J. I. n. Guzmán. A constrained slam approach to robust and accurate localisation of autonomous ground vehicles. *Robot. Auton. Syst.*, 55(7):527–540, 2007.

[77] J. Leonard and H. Durrant-Whyte. Mobile Robot Localization by Tracking Geometric Beacons. In *IEEE Transactions on Robotics and Automation (T-RA)*, volume 7, pages 376–382, 1991.

[78] K. Y. K. Leung, C. M. Clark, and J. P. Huissoon. Localization in urban environments by matching ground level video images with an aerial image. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

[79] S. Leutenegger, M. Chli, and R. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, pages 2548 –2555, 2011.

[80] X. Li and I. Guskov. Multi-scale features for approximate alignment of point-based surfaces. In *Proc. of the third Eurographics Symp. on Geometry processing (SGP)*, 2005.

[81] Y. Li and E. Olson. A general purpose feature extractor for light detection and ranging data. *Sensors*, 10(11):10356–10375, 2010.

[82] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 1999.

[83] D. G. Lowe. Distinctive image features from scale invariant keypoints. *Int. Journal on Computer Vision (ICCV)*, 2004.

[84] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. In *Journal of Autonomous Robots*, volume 4, pages 333–349, 1997.

[85] 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 of Field Robotics*, 26(11–12):892–914, November 2009.

[86] B. Matei, Y. Shan, H. Sawhney, Y. Tan, R. Kumar, D. Huber, and M. Hebert. Rapid object indexing using locality sensitive hashing and joint 3D-signature space estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 28(1):1111 – 1126, 2006.

[87] E. Michaelsen, W. von Hansen, M. Kirchhof, J. Meidow, and U. Stilla. Estimating the essential matrix: GOODSAC versus RANSAC. In *Photogrammetric Computer Vision*, 2006.

[88] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *Int. J. Comput. Vision*, 65(1-2):43–72, 2005.

[89] M. Montemerlo and S. Thrun. Large-scale robotic 3-D mapping of urban structures. In *Proc. of the Int. Symp. on Experimental Robotics (ISER)*, pages 141–150, 2004.

[90] 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. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1151–1156, 2003.

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

[92] F. Moosmann and M. Sauerland. Unsupervised discovery of object classes in 3D outdoor scenarios. In *Proc. of the Workshop on Challenges and Opportunities in Robot Perception (ICCV)*, pages 1038 –1044, 2011.

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

[94] P. Newman et al. The MOOS - Cross Platform Software for Robotics Research. http://www.robots.ox.ac.uk/ mobile/MOOS/wiki/pmwiki.php.

[95] E. Olson. *Robust and Efficient Robotic Mapping*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2008.

[96] 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)*, pages 2262–2269, 2006.

[97] E. Olson, M. Walter, S. Teller, and J. Leonard. Single-cluster spectral graph partitioning for robotics applications. In *Proc. of Robotics: Science and Systems (RSS)*, 2005.

[98] C. Parra, R. Murrieta-Cid, M. Devy, and M. Briot. 3-D modelling and robot localization from visual and range data in natural scenes. In *1st Int. Conf. on Computer Vision Systems (ICVS)*, number 1542 in LNCS, 1999.

[99] M. Parsley and S. J. Julier. Slam with a heterogeneous prior map. In *Proc. of the Conf. on Systems Engineering for Autonomous Systems (SEAS)*, 2009.

[100] R. Paul and P. Newman. FAB-MAP 3D: Topological mapping with spatial and visual appearance. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[101] J. Philbin, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[102] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *In European Conf. on Computer Vision*, pages 430–443, 2006.

[103] 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.

[104] 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.

[105] 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)*, 2010.

[106] M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. 3D environment modeling based on surface primitives. In *Towards Service Robots for Everyday Environments*, pages 281–300. Springer Berlin/Heidelberg, 2012.

[107] S. Ruiz-Correa, L. G. Shapiro, and M. Meila. A new signature-based method for efficient 3-D object recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 769–776, 2001.

[108] R. B. Rusu, N. Blodow, and M. Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

[109] 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.

[110] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz. Learning Informative Point Classes for the Acquisition of Object Model Maps. In *Proc. of the 10th Int. Conf. on Control, Automation, Robotics and Vision (ICARCV)*, 2008.

[111] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz. Persistent Point Feature Histograms for 3D Point Clouds. In *Proc. of the 10th Int. Conf. on Intelligent Autonomous Systems (IAS-10)*, 2008.

[112] 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.

[113] A. Sanfeliu. URUS project: Communication systems. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009. Workshop on Network Robots Systems.

[114] D. Scaramuzza, A. Harati, and R. Siegwart. Extrinsic self calibration of a camera and a 3D laser range finder from natural scenes. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4164 –4169, 2007.

[115] D. Scharstein and R. Szeliski. Middlebury stereo vision page, 2002. http://www. middlebury. edu/stereo.

[116] J. Shin, R. Triebel, and R. Siegwart. Unsupervised 3D object discovery and categorization for mobile robots. In *Proc. of the 15th Int. Symp. on Robotics Research (ISRR)*, 2011.

[117] R. Siegwart, K. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, R. Philippsen, R. Piguet, G. Ramel, G. Terrien, and N. Tomatis. RoboX at Expo.02: A large-scale installation of personal robots. *Journal of Robotics & Autonomous Systems*, 42(3-4), 2003.

[118] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering objects and their location in images. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, volume 1, pages 370 – 377 Vol. 1, 2005.

[119] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 2003.

[120] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial realtionships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.

[121] L. Spinello, R. Triebel, D. Vasquez, K. O. Arras, and R. Siegwart. Exploiting repetitive object patterns for model compression and completion. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2010.

[122] C. Stachniss, U. Frese, and G. Grisetti. OpenSLAM.org – give your algorithm to the community. http://www.openslam.org, 2007.

[123] 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)*, 2010.

[124] B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, and W. Burgard. Estimating consistent elevation maps using down-looking cameras and inertial sensors. In *Proc. of the Workshop on Robotic Perception on the Int. Conf. on Computer Vision Theory and Applications*, 2008.

[125] B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, A. Rottmann, and W. Burgard. Learning maps in 3D using attitude and noisy vision sensors. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[126] B. Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual SLAM for flying vehicles. *IEEE Transactions on Robotics*, 24(5):1088–1093, 2008.

[127] B. Steder, G. Grisetti, M. Van Loock, and W. Burgard. Robust on-line model-based object detection from range images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[128] B. Steder, A. Rottmann, G. Grisetti, C. Stachniss, and W. Burgard. Autonomous navigation for small flying vehicles. In *Workshop on Micro Aerial Vehicles at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[129] 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)*, 2011.

[130] 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)*, 2010.

[131] 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)*, 2011.

[132] S. Stiene, K. Lingemann, A. Nüchter, and J. Hertzberg. Contour-based object detection in range images. In *Proc. of the Third Int. Symp. on 3D Data Processing, Visualization, and Transmission*, pages 168–175, 2006.

[133] J. Tang and P. H. Lewis. Non-negative matrix factorisation for object class discovery and image auto-annotation. In *Proc. of the Int. Conf. on Content-based image and video retrieval (CIVR)*, CIVR '08, pages 105–112. ACM, 2008.

[134] 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.

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

[136] S. Thrun and colleagues. Winning the darpa grand challenge. *Journal of Field Robotics (JFR)*, 2006.

[137] 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 (IJRR)*, 23(7/8):693–716, 2004.

[138] 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. Symp. on Experimental Robotics (ISER)*, 2010.

[139] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2010.

[140] A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: the open annotation tool, 2007. http://labelme.csail.mit.edu/.

[141] 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.

[142] R. Triebel, R. Schmidt, O. M. Mozos, and W. Burgard. Instance-based AMN classification for improved object recognition in 2D and 3D laser raneg data. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2007.

[143] T. Tuytelaars, C. H. Lampert, M. B. Blaschko, and W. Buntine. Unsupervised object discovery: A comparison. *Int. Journal on Computer Vision (ICCV)*, 88(2):284–302, 2010.

[144] J. Uhlmann. *Dynamic Map Building and Localization: New Theoretical Foundations*. PhD thesis, University of Oxford, 1995.

[145] R. Unnikrishnan. *Statistical Approaches to Multi-Scale Point Cloud Processing*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2008. Chapter 4.

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

[147] C. Urmson et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics (JFR)*, 25(8):425–466, 2008.

[148] G. K. Wallace. The JPEG still picture compression standard. *Commun. ACM*, 34(4):30–44, 1991.

[149] M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 101–108, 2000.

[150] O. Wulf, A. Nüchter, J. Hertzberg, and B. Wagner. Benchmarking urban six-degree-of-freedom simultaneous localization and mapping. *Journal of Field Robotics*, 25(3):148–163, 2008.

[151] 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.

[152] K. M. 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 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation (ICRA)*, 2010.

[153] Y. Zhong. Intrinsic shape signatures: A shape descriptor for 3D object recognition. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th Int. Conf. on*, pages 689 –696, 2009.

[154] Y. Zhuang, N. Jiang, H. Hu, and F. Yan. 3-D-laser-based scene measurement and place recognition for mobile robots in dynamic indoor environments. *IEEE Trans. on Instrumentation and Measurement*, 62(2):438–450, 2013.