UNIVERSITÄT BONN

Inaugural-Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Landwirtschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn
Institut für Geodäsie und Geoinformation

# Robot Mapping and Navigation in Real-World Environments


von

Igor Bogoslavskyi

aus

Kiew, Ukraine




Bonn 2018

**Referent:**
Prof. Dr. Cyrill Stachniss, Friedrich-Wilhelms-Universität Bonn

**1. Korreferent:**
Prof. Dr. Giorgio Grisetti, La Sapienza University of Rome

**2. Korreferent:**
Prof. Dr. Wolf-Dieter Schuh, Friedrich-Wilhelms-Universität Bonn

Tag der mündlichen Prüfung: 19. November 2018

Erscheinungsjahr: 2018

Angefertigt mit Genehmigung der Landwirtschaftlichen Fakultät der Universität Bonn

# Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenenen Hilfsmittel angefertigt habe; die aus fremdem Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

| | |
|---|---|
| Ort, Datum | (Unterschrift) |

# Zusammenfassung

Roboter können unterschiedlichste Aufgaben ausführen, z.B. in Such- und Rettungsszenarien eingesetzt werden, Waren ausliefern oder Personen transportieren. Roboter, die in der realen Welt eingesetzt werden, müssen viele Herausforderungen auf dem Weg zur Vollendung ihrer Mission meistern. Zentrale Fähigkeiten, die für den Betrieb solcher Roboter erforderlich sind, sind Kartierung, Lokalisierung und Navigation. Die robuste Lösung dieser Aufgaben ist eine nicht-triviale Aufgabe, da unter anderem die Komponenten typischerweise voneinander abhängig sind. So muss beispielsweise ein Roboter gleichzeitig eine Karte aufbauen, sich darin lokalisieren, die Umgebung bzgl. möglichen Kollisionen analysieren und einen geeigneten Weg planen, um eine unbekannte Umgebung effizient zu erkunden.

Die Lösungen dieser Aufgaben hängen meist von den verwendeten Sensoren und von der Art der Einsatzumgebung ab. Eine RGB-Kamera kann zum Beispiel in einer Außenszene zum Berechnen einer visuellen Odometrie oder zur Erkennung dynamischer Objekte verwendet werden. Im Gegensatz dazu ist sie weniger nützlich in Umgebungen, die nicht genug Licht für den Betrieb von Kameras zur Verfügung stellen. Des Weiteren sollte die Software, die das Verhalten des Roboters steuert, alle Daten der verschiedenen Sensoren verarbeiten und integrieren. Dies führt oft zu technischen Systemen, die nur mit einem bestimmten Robotertyp und einem bestimmten Satz von Sensoren funktionieren. In dieser Doktorarbeit fokussieren wir uns auf Systeme und implementieren Methoden für Roboternavigationssysteme, die nahtlos mit verschiedenen Sensoren arbeiten können, sowohl im Innen- als auch im Außenbereich. Speziell mit der kürzlichen Entwicklung neuer distanzmessender RGBD und LiDAR Sensoren sehen wir die Möglichkeit Systeme zu bauen, die sowohl im Innen- als auch im Außenbereich robust arbeiten können und erweitern, damit die Einsatzgebiete von mobilen Robotern.

Die in dieser Arbeit vorgestellten Techniken zielen darauf ab, sowohl mit RGBD als auch mit LiDAR Sensoren – ohne Anpassungen für einzelne Sensormodelle – Methoden für Navigation und Szeneninterpretation in statischen sowie dynamischen Umgebungen zu realisieren. Für statische Umgebungen präsentieren wir eine Reihe von Ansätzen, welche die Kernkomponenten einer typischen

Roboter-Navigationspipeline adressieren. Ein Fokus ist die Erstellung einer konsistenten Karte der Umgebung mittels Punktwolkenregistrieung. Zu diesem Zweck präsentieren wir eine neue Methode zur photometrischen Punktwolkenregistrieung, die RGBD und LiDAR Sensoren in identischer Art und Weise behandelt und in der Lage ist Punktwolken genau und in Echtzeit zu registrieren, d.h. mit der Frequenz des Sensors. Unsere Methode dient als Baustein für den weiteren Navigationsprozess. Zusätzlich zu diesem Verfahren präsentieren wir eine Methode für Traversierbarkeitsanalyse des aktuell beobachteten Geländes. Eine Gefahrenquelle beim Navigieren von schwer zugänglichen oder komplexen Orten ist die Tatsache, dass der Roboter beim Erstellen einer konsistenten Umgebungskarte scheitern kann. Dies hat typischerweise dramatische Auswirkungen auf die Fähigkeit eines autonomen Roboters erfolgreich sein Ziel anzusteuern. Daher ist es wichtig, dass Roboter eine solche Situation erkennen und mit dieser umgehen kann, beispielsweise sicher zum Startpunkt seiner Mission zurückzufahren. Um diese Herausforderung anzugehen, haben wir eine Methode zur Analyse der Qualität der Karte, die der Roboter gebaut hat, entwickelt und können den Roboter sicher zum Ausgangspunkt der Mission zurückbringen, auch wenn die Umgebungskarte sich in einem inkonsistenten Zustand befindet.

Szenen in dynamischen und statischen Umgebungen unterscheiden sich für einen Roboter erheblich von einander. In einer dynamischen Einstellung können sich Objekte bewegen und daher ist die Schätzung der statischen Traversierbarkeit nicht ausreichend. Mit den entwickelten Ansätzen dieser Arbeit zielen wir darauf ab, einzelne Objekte zu identifizieren und sie virtuell zu verfolgen. Wir begegnen diesen Herausforderungen mit einer Methode zum Clustering einer Szene, welche mit einem LiDAR Scanner abgenommen wurde. Diese benötigt nur einen einzigen Parameter, der beschreibt, wann zwei Cluster ähnliche Objekte repräsentieren. Dieser Verfahren kann mit hoher Frequenz ausgeführt werden und die Tracking-Leistung unterstützen.

Alle in dieser Arbeit vorgestellten Methoden sind in der Lage Roboter mit Echtzeitsteuerung im Betrieb zu unterstützen. Sie basieren auf RGBD oder LiDAR Sensoren und wurden auf realen Robotern in reale Umgebungen und auf Basis verschiedener Datensätze getestet. Alle Ansätze waren in Konferenzpapieren und Zeitschriftenartikeln mit Peer-Review-Verfahren veröffentlicht. Darüber hinaus wurden die meisten der vorgestellten Beiträge als Open Source Software der Öffentlichkeit zur Verfügung gestellt.

# Abstract

Robots can perform various tasks, such as mapping hazardous sites, taking part in search-and-rescue scenarios, or delivering goods and people. Robots operating in the real world face many challenges on the way to the completion of their mission. Essential capabilities required for the operation of such robots are mapping, localization and navigation. Solving all of these tasks robustly presents a substantial difficulty as these components are usually interconnected, i.e., a robot that starts without any knowledge about the environment must simultaneously build a map, localize itself in it, analyze the surroundings and plan a path to efficiently explore an unknown environment. In addition to the interconnections between these tasks, they highly depend on the sensors used by the robot and on the type of the environment in which the robot operates. For example, an RGB camera can be used in an outdoor scene for computing visual odometry, or to detect dynamic objects but becomes less useful in an environment that does not have enough light for cameras to operate. The software that controls the behavior of the robot must seamlessly process all the data coming from different sensors. This often leads to systems that are tailored to a particular robot and a particular set of sensors. In this thesis, we challenge this concept by developing and implementing methods for a typical robot navigation pipeline that can work with different types of the sensors seamlessly both, in indoor and outdoor environments. With the emergence of new range-sensing RGBD and LiDAR sensors, there is an opportunity to build a single system that can operate robustly both in indoor and outdoor environments equally well and, thus, extends the application areas of mobile robots.

The techniques presented in this thesis aim to be used with both RGBD and LiDAR sensors without adaptations for individual sensor models by using range image representation and aim to provide methods for navigation and scene interpretation in both static and dynamic environments. For a static world, we present a number of approaches that address the core components of a typical robot navigation pipeline. At the core of building a consistent map of the environment using a mobile robot lies point cloud matching. To this end, we present a method for photometric point cloud matching that treats RGBD and LiDAR sen-

sors in a uniform fashion and is able to accurately register point clouds at the frame rate of the sensor. This method serves as a building block for the further mapping pipeline. In addition to the matching algorithm, we present a method for traversability analysis of the currently observed terrain in order to guide an autonomous robot to the safe parts of the surrounding environment. A source of danger when navigating difficult to access sites is the fact that the robot may fail in building a correct map of the environment. This dramatically impacts the ability of an autonomous robot to navigate towards its goal in a robust way, thus, it is important for the robot to be able to detect these situations and to find its way home not relying on any kind of map. To address this challenge, we present a method for analyzing the quality of the map that the robot has built to date, and safely returning the robot to the starting point in case the map is found to be in an inconsistent state.

The scenes in dynamic environments are vastly different from the ones experienced in static ones. In a dynamic setting, objects can be moving, thus making static traversability estimates not enough. With the approaches developed in this thesis, we aim at identifying distinct objects and tracking them to aid navigation and scene understanding. We target these challenges by providing a method for clustering a scene taken with a LiDAR scanner and a measure that can be used to determine if two clustered objects are similar that can aid the tracking performance.

All methods presented in this thesis are capable of supporting real-time robot operation, rely on RGBD or LiDAR sensors and have been tested on real robots in real-world environments and on real-world datasets. All approaches have been published in peer-reviewed conference papers and journal articles. In addition to that, most of the presented contributions have been released publicly as open source software.

# Acknowledgements

Writing a Ph.D. thesis is an interesting and a unique endeavor. I feel very lucky to be surrounded by some outstanding people, without whom this work would have never even been started. I would like to thank them all in this chapter.

First and foremost, I want to thank my advisor and a good friend, Cyrill Stachniss for his continuous guidance not only in the scientific world, but also in being a living example of how to live a life to its fullest. Many of the milestones that I have achieved in the last six years would be utterly impossible if I would not have met him. I will never forget the all-nighters we pulled before the conference deadlines, good times later at these conferences, as well as the invaluable lessons on how to give a presentation, or on writing a paper. I am extremely thankful for these and a huge number of other things!

I also want to thank my family for their continuous support and advice throughout the years. It is them that I must thank for all the opportunities available to me now, for guiding and pushing me through my life. No words can express how much their support means to me. I want to thank them for presenting me all these possibilities that I am able to enjoy now.

I would like to express my gratitude to all the people, who have become my close friends throughout these years. I feel privileged to be surrounded by these great and very different people, who support and motivate me in my career and life. The time spent with Lorenzo Nardi, Nived Chebrolu, Olga Vysotska, Andres Milioto, Emanuele Palazzolo, Philipp Lottes, Jens Behley, and Fabrizio Nenci is utterly unforgettable, and I thank them all for sharing the ups and downs of the life in the lab and beyond. In addition to my colleges, I want to thank my old childhood friends Yurij Liapin, Dmytro Lyganov, Iurii Perga, and Michael Danilevich for their constant support in anything that I have ever started. Their belief keeps me going in the hardest times and I can always rely on them, as much as I can rely on myself. Also, I would like to thank Maxim Tatarchenko for the awesome discussions about everything, both scientific and not, and for being a friend I can always expect to understand my point of view, and to provide a valuable and well-weighted feedback.

In addition to the people mentioned above, there have been an enormous amount of people, who have collaborated with me, and who I have learnt a

great deal from, during my time as a Ph.D. student. I'd like to thank people in the labs of Wolfram Burgard and Giorgio Grisetti for being my "second scientific family", for all the fun times at the conferences, and when meeting around the world. Pratik Agarwal, Mladen Mazuran, Nicola Abdo, Diego Tipaldi, Christoph Sprunk, Philipp Ruchti, Tim Caselitz, Jacopo Serafin, Taigo Bonanni, Bartolomeo Della Corte, and Dominik Schlegel, I want to thank all of you for all the fun times we shared, wherever it was. I also want to thank Wolfram Burgard for the great times I have spent while being a student in his lab, and Giorgio Grisetti for invaluable lessons on programming and scientific rigor. I have also collaborated with some other students: Jacopo Serafin, Daniel Perea Ström, and Mladen Mazuran have all been extremely pleasant to work with, and I greatly appreciate their help in all our collaborative efforts.

My thanks also go to Birgit Klein, without whom my life would become hell whenever I would need to encounter any administrative matters.

I would also like to thank people, who have published open source datasets, that allowed the methods presented in this thesis to be evaluated against the other ones. My thanks to Frank Moosmann for open sourcing his SLAM datasets and to Andreas Geiger for the great effort of providing the KITTI datasets. My thanks also go to Christian Kerl, Jürgen Sturm, and Daniel Cremers for open sourcing the dense visual odometry method.

Lastly, I would love to thank Olga for sharing this roller-coaster life with me. In all the adventures and moves throughout these years, she uniquely combines the impossible: being my support, motivation, and challenge at the same time. I cannot wait to see what awaits us next!

The work presented in this thesis is partially supported by the European Commission through the ROVINA project, contract number FP7-600890-ROVINA. The financial support of the EC through the ROVINA project is gratefully acknowledged, and I would like to also thank our project officer Albert Gauthier for his support throughout and after the completion of ROVINA.

# Contents

# List of Figures

# List of Tables



# List of Algorithms

# Chapter 1

# Introduction

## 1.1 General software for robot operation

There are many tasks that are mundane, repetitive, or dangerous that humans do on a regular basis. Typical examples of such tasks range from surveying a particular potentially dangerous area to delivering goods and people. Mobile robots have the potential to aid or even substitute humans in these tasks. In order to carry out such tasks, the robots have to solve a number of challenges such as dealing with complex surroundings, performing mapping, localization, and navigation in static and dynamic environments.

Diverse environments put tight constraints on the methods and sensors that can be used by the robots. For example, while a robot that navigates in an urban environment can rely on RGB cameras for detecting objects around it, these cameras have less value in an underground site where there is no light apart from the one that the robot carries itself. To circumvent this problem, sensors like Microsoft Kinect or Asus Xtion can be used as they provide 3D data in the absence of light. However, using these sensors in an outdoor environment is complicated as they provide much less information when their infrared emitter is overwhelmed by the light from the sun. Because of these and other similar constraints, the robots are usually designed with a particular environment in mind and rely on different sensors.

Having different sensors onboard leads to additional complexity in designing the software that can work with the data coming from these sensors. A typical approach is to design custom algorithms that make use of specific sensors. This, however, leads to low code reusability forcing the methods to be reimplemented for every new robot configuration. To avoid these issues, it might be beneficial to invest more time into the design of the software to make it more sensor agnostic, i.e., making the same software work well with multiple types of sensors.

Figure 1.1: Robots targeted in this thesis. *Left:* ROVINA robot in the Roman catacombs of Priscilla. *Middle:* Clearpath Robotics Husky robot with custom sensor setup at Bonn University campus. *Right* AnnieWAY self driving car, image courtesy of [43].

In addition to the challenges presented above, robots navigating in real-world environments must make precise estimations about the surrounding world fast. Furthermore, these computations must be performed on an onboard processing unit, and as the amount of power the robot can carry with it is limited, expensive computations drain it quickly. The robots, thus, face a complex trade-off between how many computations they must make to understand the surroundings, build a map, and localize themselves in it, how much they must move to gather new knowledge about the environment, and how much power these actions take. Therefore, one of the cornerstones of developing robust algorithms for robots operating in the real world is efficiency. Robots require information at framerate or even faster and, at the same time, the quality of this information cannot be compromised, as any mistake potentially has a high cost.

In this thesis, we focus on developing robust and efficient algorithms that aid different parts of the robot operation pipeline. We present contributions to mapping, perception and navigation stacks of a robot navigating in real-world environments. The data that we work with come from various sensors, and we are able to process it at framerate on a mobile robot platform. All the methods presented in this thesis have been published at peer-reviewed international conferences and journals, and some of them have been made available as open source software.

This thesis is organized as follows. In Part I, we focus on the challenges associated with navigating in static environments, and present our solutions to typical problems that arise in these environments like incremental pose matching, traversability analysis, dealing with broken maps and navigation with or without a valid map. In Part II, we show how these methods can be adapted and extended to work in the environments with dynamic objects.

## 1.2 Focus on robots operating in real world

Our work is partially motivated by a project for autonomous exploration and digital preservation of hard-to-access archaeological sites such as catacombs. Catacombs are old Roman burying places used between the 2nd and 5th century in Italy. Even today, they are partially unexplored due to the high risk of entering them. First, most sites are unstable and can collapse. Second, most of the (non-ventilated) catacombs yield a high concentration of radioactive radon gas so that humans are only allowed to stay in these sites for 15 min-30 min to prevent serious health issues. Thus, robots are an excellent tool for the exploration, mapping, and digital preservation of such sites. To achieve that, the robots have to operate and explore the space in a completely autonomous fashion. As part of a joint effort in the ROVINA project, we have developed a robot depicted on the left of Figure 1.1, that is able to navigate and map catacombs of Priscilla in Rome.

Even though catacombs are challenging environments, they have one trait that simplifies robot operation: they are static. The static assumption holds for most underground environments but rarely for urban ones. Therefore, we have implemented a number of extensions to the methods required for static environment navigation to aid the robot in dealing with the changing environments. The presence of dynamic obstacles in an environment has a significant impact on the reasoning behind the actions of the robot. When targeting dynamic environments, we target mobile robots such as one depicted in the middle of Figure 1.1, navigating the campus at the University of Bonn, as well as robots tailored for autonomous driving, such as the AnnieWAY car from the University of Karlsruhe shown on the right of Figure 1.1.

We focus on designing algorithms that work at least at the frame rate of the sensors mounted on real robots with constrained computational power while being generic and applicable to different indoor and outdoor environments. While a single Ph.D. thesis is, probably, not enough to completely solve perception, mapping, localization, and navigation of mobile robots in static and dynamic environments, we focus on providing the implementations for a number of crucial parts of these tasks. The contributions of this thesis are solutions to different aspects of the robot perception and navigation tasks that use a depth sensor as the source of information about the surrounding world. Depth sensors have been popularized by the introduction of the Microsoft Kinect sensor in 2010 and the consequent introduction of the Asus Xtion in 2011, as well as the Velodyne PUCK LiDAR in 2014. These sensors made it possible to obtain 3D information about the environment on a cheap mobile robot. We treat the data coming from Microsoft Kinect, Asus Xtion and LiDARs uniformly and represent it in the form of range images, as this representation allows for efficient processing of the sensor data. For more information on range images, we refer the reader to Section 2.2.

We envision that a robot that carries various sensors such as Microsoft Kinect, Asus Xtion or a LiDAR navigating an environment autonomously, must address a number of challenges in order to carry out its mission safely. We assume that the robot starts without any knowledge about the environment, and its goal is to explore this environment fully. To do this, the robot must be able to localize itself in an unknown environment. A typical approach to solving this problem is simultaneous localization and mapping (SLAM) as the robot must build its map and localize itself in it. SLAM is a relatively well-studied problem [119] and relies on a number of techniques. In this thesis, we only consider the graph-based variant of SLAM, where all the poses where the robot takes measurements are organized into a pose-graph representation that allows for integrating information from different sensor measurements taken at various positions, and optimizing the resulting graph based on this. We refer the reader to a tutorial by Grisetti *et al.* [47] for more details on pose-graph construction and typical optimization techniques. Building such a graph usually consists of incrementally matching the point clouds while moving in an environment adding the edges with relative constraints to the graph, detecting so-called loop closure edges to avoid incremental error accumulation and an optimization algorithm that is able to optimize this graph. In this thesis, we specifically address the point cloud matching by implementing our own generic point cloud matching algorithm, which takes advantage of the information provided by multiple cues available from the sensor data by optimizing the photometric error. To support incremental matching, this component must run at the frame rate of the sensor and work reliably utilizing all available information, such as color, depth and surface normals.

Having a consistent map of the currently observed environment allows the robot to plan paths through it in order to discover more about it. While this is not the main contribution of this thesis, we have also created a method that allows exploring an unknown environment efficiently. However, being able to *plan* a path is not enough to guarantee safe navigation in an unknown environment along this path. The robot must know, which parts of the surrounding environment are traversable and if the map that it has built so far is reliable for navigation. We contribute solutions to both of these challenges. We analyze the traversability of the currently seen environment, and integrate it into a single consistent map while continuously monitoring the consistency of this resulting map. If, at any point in time, a map inconsistency is detected, the robot must be able to recover itself not relying on this inconsistent map. Such a method for robust robot recovery is also one of the main contributions of this thesis.

A robot equipped with the functionality described above is capable to autonomously explore a static environment, i.e., the one that does not change with time. This assumption holds for caves, catacombs, warehouses, and some search-

and-rescue scenarios, but is not valid for outdoor navigation in typical urban scenarios. To account for this, we extend the set of methods described above by further methods for scene analysis to detect objects that might be moving in the scene. In particular, we developed a method that clusters the data observed by the robot into meaningful objects in an unsupervised fashion. These objects can then be tracked with any tracking algorithm to find out if they are dynamic or static. To this end, we also implemented a robust measure that provides the information if two objects match each other using the notion of the shape of the objects. While these contributions do not *solve* navigation in dynamic environments, we believe they are an important cornerstone to implementing such algorithms.

## 1.3 Main contributions

This thesis presents novel solutions to several relevant problems in the context of mobile robots operating with depth sensors. It provides contributions to multiple aspects of robot perception and navigation in the real world under computational constraints stemming from the fact that all these algorithms are capable of running on board of a mobile robot platform. This section provides a short summary of the central achievements and methods that contribute to the state of the art in robotics.

The first contribution presented in this work is in the context of point cloud matching. Our method for photometric point cloud matching [26] pushes the state of the art forward by incorporating additional modalities into the standard photometric matching, presented by the dense visual odometry [67] (DVO) approach. Our method allows for matching point clouds originating from various sensors, such as Microsoft Kinect or LiDARs at the frame rate of the used sensor. It builds upon DVO but allows using additional cues to aid the matching procedure. Running it on real-world data shows that we can maintain a very good matching performance under various conditions using different sensors while using the exact same code base, which we have also made available as an open source library. The method is presented in-depth in Section 3.1.

The second contribution of this thesis targets the scene analysis that can be performed on the range data perceived by the robot at real time. It is a method for performing scene traversability analysis [14]. This information is crucial for a robot when navigating in complex environments. This method has been developed as part of the EC-funded ROVINA project, where we have deployed a robot to explore the Roman catacombs for the tasks of cultural heritage preservation. The method analyzes if the surroundings are traversable by the robot, which is the prerequisite for collision-free, autonomous navigation. The method

is strictly geometrical, running at the frame rate of the sensor on computationally constrained hardware using input data from cheap and rather noisy sensors. The software developed for this method has formed the basis for navigation in real underground sites in Rome and has been used by multiple components running on the robot such as the navigation and exploration stacks, proving it to be a viable method for analyzing traversability of static environments. This method is presented in-depth in Section 4.1.

The third contribution of this thesis builds upon the same input data and targets the navigation stack of the robot operating in an unknown environment. We have developed systems that allow the robot to navigate the environment autonomously, but the main contribution of this part of this work is a method that focuses on detecting if the map constructed by the robot driving through a static environment is in an inconsistent state, and on handling the safe return of the robot to the starting point by unwinding the recorded odometry trajectory [10, 96]. This method continuously retracts over the odometry data taken on the way up to the place where the mapping system failed, and corrects its position using point cloud matching. The main contribution of this method is not as much in a single state-of-the-art method, but in a combination of the above mentioned methods into a single system that has been successfully used within ROVINA project on a real robot platform in real underground environments. It is presented in Section 4.3 of this thesis.

The fourth contribution of this work is a robust and efficient clustering algorithm for point clouds that uses a novel cluster separation criterion defined on range images [11, 13]. The performance of this algorithm is guided by a single bounded threshold value and has performance as good as the classical Euclidean clustering while being orders of magnitude faster. It has been tested on real world data acquired with various Velodyne LiDARs and delivers state-of-the-art performance much faster than the frame rate of the sensor that provides data to it. More details on this method can be found in Section 5.1.

Lastly, our fifth contribution is a method to analyze the quality of the alignment between the objects clustered with a clustering algorithm in a scenario where the objects have to be tracked [12]. We again make use of the range image data representation and use free space surrounding the objects as an additional criterion for providing a single score for quantifying the quality of a match of pairs of objects. For more details on this method, please refer to Section 5.2.

Overall this thesis presents five contributions, each of which target an important part of the robot mapping, navigation, or scene analysis pipeline. All methods presented here are tested on real-world data and on real robots navigating challenging environments. These methods share the fact that they work on range data and on their range image representation. Most of the contributions

presented here have been published as open source software, either as part of the ROVINA open source software suite or standalone.

## 1.4 Publications

Parts of this thesis have been published in the following peer-reviewed conference and journal articles:

- I. Bogoslavskyi, O. Vysotska, J. Serafin, G. Grisetti, and C. Stachniss. Efficient traversability analysis for mobile robots using the kinect sensor. In *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2013
- I. Bogoslavskyi and C. Stachniss. Fast range image-based segmentation of sparse 3d laser scans for online operation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016
- I. Bogoslavskyi, M. Mazuran, and C. Stachniss. Robust Homing for Autonomous Robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2016
- D. Perea-Ström, I. Bogoslavskyi, and C. Stachniss. Robust exploration and homing for autonomous robots. In *Journal on Robotics and Autonomous Systems (RAS)*, volume 90, pages 125–135, 2017
- I. Bogoslavskyi and C. Stachniss. Analyzing the quality of matched 3d point clouds of objects. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017
- I. Bogoslavskyi and C. Stachniss. Efficient online segmentation for sparse 3d laser scans. In *Photogrammetrie – Fernerkundung – Geoinformation (PFG)*, volume 85, pages 41–52, 2017
- B. Della Corte*, I. Bogoslavskyi*, C. Stachniss, and G. Grisetti. A general framework for flexible multi-cue photometric point cloud registration. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2018. * authors have contributed equally

## 1.5 Collaborations

Some work included in the publications presented above has been done in collaboration with other people. The work "Efficient Traversability Analysis for Mobile Robots using the Kinect Sensor" [14] has been carried out at the University of Freiburg along with Olga Vysotska and Jacopo Serafin under the supervision of Giorgio Grisetti and Cyrill Stachniss. Olga has been helping in implementing the traversability algorithm, while Jacopo has provided his help in initial adaptation of the algorithm to the robot platform that was in the initial stage of development in Rome at the time.

The work "Robust Homing for Autonomous Robots" [10] has been carried out in collaboration with Mladen Mazuran. Mladen has provided his advice on adapting the method from his earlier work [80] to a new sensor.

We further collaborated with Daniel Perea-Ström on the paper "Robust exploration and homing for autonomous robots" [96]. This paper focuses on implementing a robust and fail-safe system for robot exploration that is able not only to use the geometry of the environment to explore the environment efficiently, but is also able to detect when the underlying map of the environment is broken and in this case returns the robot safely to the initial position. In this work, Daniel was mostly focusing on exploration, while my focus has been the robust homing and integration of the components into a single system.

Finally, our last collaboration has been carried out along with Bartolomeo Della Corte, who is a student at the lab of Giorgio Grisetti. Bartolomeo and I have put equal efforts into developing the proposed algorithm and therefore share the first authorship of the paper.

# Chapter 2

# Basic techniques

## 2.1 Least Squares

Multiple problems in robotics can be addressed by local minimization of squared errors. This type of minimization problems maps directly to the least squares formulation. The idea behind the least squares method is that if we can define a function that consists of a sum of squares of values that we have some control over, we can use efficient iterative techniques to find such a configuration that minimizes the resulting sum. It is used in many robotics applications, e.g., using iterative closest point algorithms for point cloud registration or in various graph optimization techniques for simultaneous localization and mapping. In this section we will outline the general least squares formulation and method as well as the typical assumptions made in order to solve the underlying non-linear least squares problems efficiently.

### 2.1.1 Least squares formulation

Let $\mathbf{x} = [x_1, x_2, \dots]^\top \in \mathbb{R}^n$ be a vector of values $x_i$. Then we can talk about minimizing the function $F(\mathbf{x})$ that is defined as a sum of squares of functions $f_i$ applied to $\mathbf{x}$:

$$F(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^{m} (f_i(\mathbf{x}))^2, \tag{2.1}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \dots, m$ are given functions, and $m \geq n$. We want to find a local minimizer $\mathbf{x}^*$ for the function $F(\mathbf{x})$:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} F(\mathbf{x}). \tag{2.2}$$

We assume that the cost function $F$ is differentiable and so smooth that the following Taylor expansion holds:

$$F(\mathbf{x}+\mathbf{h}) = F(\mathbf{x}) + \mathbf{h}^\top \mathbf{g} + \frac{1}{2}\mathbf{h}^\top \mathbf{H}\mathbf{h} + O(\|\mathbf{h}\|^3), \tag{2.3}$$

where $\mathbf{g}$ is the *gradient*:

$$\mathbf{g} \equiv \mathbf{F}'(\mathbf{x}) = \begin{bmatrix} \frac{\partial F}{\partial x_1}(\mathbf{x}) \\ \dots \\ \frac{\partial F}{\partial x_n}(\mathbf{x}) \end{bmatrix}, \tag{2.4}$$

and $\mathbf{H}$ is the *Hessian*:

$$\mathbf{H} \equiv \mathbf{F}''(\mathbf{x}) = \left[ \frac{\partial^2 F}{\partial x_i \partial x_j}(\mathbf{x}) \right]. \tag{2.5}$$

### 2.1.2 Necessary and sufficient conditions for a local minimizer

If $\mathbf{x}^*$ is a local minimizer, then:

$$\mathbf{g}^* \equiv \mathbf{F}'(\mathbf{x}^*) = 0. \tag{2.6}$$

This condition is a necessary condition, but not a sufficient one, as such a solution can also be a saddle point, i.e., a point which is a local minimum in one direction and a local maximum in another direction. To determine if a given stationary point $\mathbf{x}_s$ is a local minimizer we can use the second order term in the Taylor series:

$$F(\mathbf{x}_s+\mathbf{h}) = F(\mathbf{x}_s) + \frac{1}{2}\mathbf{h}^\top \mathbf{H}_s \mathbf{h} + O(\|\mathbf{h}\|^3), \tag{2.7}$$

where $\mathbf{H}_s = \mathbf{F}''(\mathbf{x}_s)$. From the fact that $F(\mathbf{x}_s+\mathbf{h})$ must be bigger than $F(\mathbf{x}_s)$ for any $\mathbf{h}$ we can formulate a sufficient condition for a local minimizer: if $\mathbf{x}_s$ is a stationary point and $\mathbf{F}''(\mathbf{x}_s)$ is positive definite then $\mathbf{x}_s$ is a local minimizer.

### 2.1.3 Descent methods

It is usually impossible to find the minimum of $F(\mathbf{x})$ directly, therefore a variety of iterative methods are used in order to descent into the minimum. We can search for the solution $\mathbf{x}^*$ by iteratively moving in the direction towards a local minimizer. The direction towards the local minimizer has to be determined at each iteration. Roughly speaking, we can describe the procedure as one consisting of the following steps:

1. Search for the direction of descent $\mathbf{h}$.
2. Make a step of some size $\alpha\mathbf{h}$ in that direction.
3. Repeat the procedure until convergence.

By making a step we mean evaluating the function $F$ for the argument $\mathbf{x} + \alpha\mathbf{h}$. From the Taylor expansion in Equation (2.3):

$$F(\mathbf{x} + \alpha\mathbf{h}) \simeq F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}). \tag{2.8}$$

We say, that $\mathbf{h}$ is a descent direction if $F(\mathbf{x} + \alpha\mathbf{h})$ is a decreasing function of $\alpha$ at $\alpha = 0$.

#### 2.1.3.1 Steepest descent

When performing the optimization procedure we can pick both $\mathbf{h}$ and $\alpha$. Usually we want to minimize the number of steps we need to make in order to reach the minimum of the function $F(\mathbf{x})$. We can say that we perform a step $\alpha\mathbf{h}$, where $\alpha$ is positive. Then, using Equation (2.8), we can write the relative gain in function value:

$$\begin{aligned}
\lim_{\alpha\to 0} \frac{F(\mathbf{x}) - F(\mathbf{x} + \alpha\mathbf{h})}{\alpha \|\mathbf{h}\|} &= \lim_{\alpha\to 0} \frac{F(\mathbf{x}) - (F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}))}{\alpha \|\mathbf{h}\|} \\
&= \lim_{\alpha\to 0} \frac{-\alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x})}{\alpha \|\mathbf{h}\|} \\
&= -\frac{1}{\|\mathbf{h}\|}\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) \\
&\overset{\text{dot product}}{=} -\|\mathbf{F}'(\mathbf{x})\| \cos\theta,
\end{aligned} \tag{2.9}$$

where, $\theta$ is an angle between $\mathbf{h}$ and $\mathbf{F}'(\mathbf{x})$. Therefore, we see that the biggest change in function is reached if $\theta = \pi$, i.e., $\mathbf{h} = -\mathbf{F}'(\mathbf{x})$. This choice of $\alpha$ and $\mathbf{h}$ is called "steepest descent" but it comes at a cost. The final convergence of this method is slow as the derivative $\mathbf{F}'(\mathbf{x})$ approaches zero at the final stages of the optimization. To mitigate this behavior other methods, like the Newton's method, are commonly used.

#### 2.1.3.2 Newton's method

To have good convergence in the final stage we must pick the step $\mathbf{h}$ better than when using the steepest descent method. We can derive the Newton's method from the fact that $\mathbf{x}^*$ is a stationary point, i.e., $\mathbf{F}'(\mathbf{x}^*) = 0$. This is a non-linear system of equation. To solve it we consider its Taylor expansion:

$$\begin{aligned}
\mathbf{F}'(\mathbf{x} + \mathbf{h}) &= \mathbf{F}'(\mathbf{x}) + \mathbf{F}''(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2) \\
&\simeq \mathbf{F}'(\mathbf{x}) + \mathbf{F}''(\mathbf{x})\mathbf{h} \\
&= \mathbf{F}'(\mathbf{x}) + \mathbf{H}\mathbf{h}.
\end{aligned} \tag{2.10}$$

Setting the term $\mathbf{F}'(\mathbf{x} + \mathbf{h})$ to zero following the logic described above we search for the best direction vector $\mathbf{h}$ as a solution to

$$\mathbf{H}\mathbf{h} = -\mathbf{F}'(\mathbf{x}). \tag{2.11}$$

Solving this equation allows us to find the current best direction vector $\mathbf{h}$ and iterate by

$$\mathbf{x} \rightarrow \mathbf{x} + \mathbf{h}. \tag{2.12}$$

### 2.1.4 Non-linear least squares

In the case when the functions $f_i$ are non-linear, more efficient methods are usually used in order to find the minimum of $F(\mathbf{x})$. These methods avoid computing the second derivatives of function $F(\mathbf{x})$. More formally, we want to find $\mathbf{x}^* = \text{argmin}_{\mathbf{x}} F(\mathbf{x})$, where

$$F(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{x}))^2 = \frac{1}{2}\mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}). \tag{2.13}$$

Provided, that $\mathbf{f}$ has continuous second partial derivatives, we can write its Taylor expansion:

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) = \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2), \tag{2.14}$$

where $\mathbf{J}$ is the Jacobian. Making use of Equation (2.1), the partial derivatives of $F$ are:

$$\begin{aligned} \frac{\partial F}{\partial x_j}(\mathbf{x}) &= \frac{\partial(\frac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{x}))^2)}{\partial x_j} \\ &= \sum_{i=1}^{m} f_i(\mathbf{x})\frac{\partial f_i}{\partial x_j}(\mathbf{x}). \end{aligned} \tag{2.15}$$

Thus the gradient is

$$\mathbf{F}'(\mathbf{x}) = \mathbf{J}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}). \tag{2.16}$$

Following the same logic, we can also estimate the Hessian:

$$\mathbf{F}''(\mathbf{x}) = \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x})\mathbf{f}_i{}''(\mathbf{x}). \tag{2.17}$$

Such Hessian can be complicated to compute as it involves the second derivatives of functions $f_i$. In order to avoid computing them, approximations are used that allow simplifying these equations. The most commonly used method to avoid computing the Hessian is the Gauss-Newton method presented in the section below.

#### 2.1.4.1 Gauss-Newton method

The Gauss-Newton method approximates the functions $\mathbf{f}$ by linearizing $\mathbf{f}$ in the neighborhood of $\mathbf{x}$, i.e., for small $\|\mathbf{h}\|$, the Taylor expansion of $\mathbf{f}$ is:

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) \simeq \boldsymbol{l}(\mathbf{h}) \equiv \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{h} = \mathbf{f} + \mathbf{J}\mathbf{h}. \tag{2.18}$$

The same operation can be applied to the sum of all functions $\mathbf{f}$, i.e, to the $F$ function:

$$\begin{aligned} F(\mathbf{x}+\mathbf{h}) \simeq L(\mathbf{h}) &\equiv \frac{1}{2}\boldsymbol{l}(\mathbf{h})^\top \boldsymbol{l}(\mathbf{h}) \\ &= \frac{1}{2}\mathbf{f}^\top\mathbf{f} + \mathbf{h}^\top\mathbf{J}^\top\mathbf{f} + \frac{1}{2}\mathbf{h}^\top\mathbf{J}^\top\mathbf{J}\mathbf{h} \\ &= F(\mathbf{x}) + \mathbf{h}^\top\mathbf{J}^\top\mathbf{f} + \frac{1}{2}\mathbf{h}^\top\mathbf{J}^\top\mathbf{J}\mathbf{h}. \end{aligned} \tag{2.19}$$

From this it is easy to derive that the gradient and the Hessian of $L$ are:

$$\mathbf{L}'(\mathbf{h}) = \mathbf{J}^\top\mathbf{f} + \mathbf{J}^\top\mathbf{J}\mathbf{h} \tag{2.20}$$

$$\mathbf{L}''(\mathbf{h}) = \mathbf{J}^\top\mathbf{J}. \tag{2.21}$$

Now we can find a unique minimizer by solving for $\mathbf{h}$:

$$(\mathbf{J}^\top\mathbf{J})\mathbf{h} = -\mathbf{J}^\top\mathbf{f}. \tag{2.22}$$

And the typical next step is then: $\mathbf{x} \to \mathbf{x} + \alpha\mathbf{h}$. The classical Gauss-Newton method uses $\alpha = 1$.

#### 2.1.4.2 Levenberg-Marquardt method

Levenberg and Marquardt suggested to use a *damped* Gauss-Newton method, where the step $\mathbf{h}$ is defined by solving the damped version of Equation (2.22):

$$(\mathbf{J}^\top\mathbf{J} + \alpha\mathbf{I})\mathbf{h} = -\mathbf{J}^\top\mathbf{f}, \tag{2.23}$$

where $\mathbf{I}$ is an identity matrix.

For all $\alpha > 0$ the coefficient matrix is positive definite which ensures that the step $\mathbf{h}$, picked in such a way, is always taken towards the descent direction, which is not the case for Gauss-Newton algorithm, e.g., in the case of "overshooting" the correct solution. For large values of $\alpha$ the step is inverse proportional to $\alpha$ itself and is oriented in the steepest descent direction. This is good if the current iteration is far from the local optimum. If, however, $\alpha$ is very small the method converges to a standard Gauss-Newton method, which ensures good final convergence. Usually it is a good idea to start with larger values of $\alpha$ related to the Jacobian:

$$\alpha_o = \tau \cdot \max_i\{a_{ii}^{(0)}\}, \tag{2.24}$$

where $a_{ii}^{(0)}$ are the elements of matrix $\mathbf{A}_0 = \mathbf{J}(\mathbf{x}_0)^\top\mathbf{J}(\mathbf{x}_0)$, and $\tau$ is the parameter chosen by the user. The values of $\alpha$ can be reduced with iterations depending on the gain ratio

$$\varrho = \frac{F(\mathbf{x}) - F(\mathbf{x}+\mathbf{h})}{L(\mathbf{0}) - L(\mathbf{h})}. \tag{2.25}$$

This way, at the start of the minimization, we use the steepest gradient descent, and towards the end of the minimization procedure we switch to the Gauss-Newton method. Overall, using Levenberg-Marquardt algorithm in place of Gauss-Newton method results in a more robust solution less sensitive to a bad initial guess.

### 2.1.5 Use cases in robotics

Non-linear least-squares techniques have multiple typical use cases in robotics. In this thesis they are used for point cloud registration and pose graph optimization. Most of the time the observations are correlated and we need to extend the formulation to account for this and the measurement uncertainty. In this case, functions $f_i(\mathbf{x})$ represent errors between the estimated constraint and the measured one. We will denote these errors as

$$e_i(\mathbf{x}, z_i) = h_i(\mathbf{x}) - z_i, \tag{2.26}$$

where, $h_i(\mathbf{x})$ is a function that maps $\mathbf{x}$ to a predicted measurement, while $z_i$ is the actual observation.

If the additional information on the reliability of these measurements is available, we can incorporate it into the squared error terms. We can therefore rewrite Equation (2.13) for the error term defined in Equation (2.26) in a matrix form as

$$E(\mathbf{x}, \mathbf{z}) = \frac{1}{2}\mathbf{e}(\mathbf{x}, \mathbf{z})^\top \mathbf{\Lambda}\ \mathbf{e}(\mathbf{x}, \mathbf{z}), \tag{2.27}$$

where $\mathbf{\Lambda}$ is the so-called information matrix. In this thesis, we assume our measurements to be uncorrelated so the information matrix only has diagonal values that depict our certainty in a particular sensor. This matrix propagates naturally through all the equations presented above, resulting in Equation (2.22) to have the form

$$(\mathbf{J}^\top \mathbf{\Lambda}\ \mathbf{J})\mathbf{h} = -\mathbf{J}^\top \mathbf{\Lambda}\ \mathbf{e}(\mathbf{x}, \mathbf{z}). \tag{2.28}$$

### 2.1.6 Relation to adjustment theory

The least-squares formulation presented above, maps to well-known adjustment models [38, 88]. In this theory we have a vector of observations $\mathbf{L}$, a vector of initial values $\mathbf{X}^0$ and a vector of adjustments $\hat{\mathbf{x}}$. Combining these vectors yields a vector of balanced parameters $\hat{\mathbf{X}}$:

$$\hat{\mathbf{X}} = \mathbf{X}^0 + \hat{\mathbf{x}}. \tag{2.29}$$

We can further define a vector of the observation estimates $\mathbf{L}^0$ that depends on the current state $\mathbf{X}^0$ and some observation function

$$\mathbf{L}^0 = \mathbf{f}(\mathbf{X}^0), \tag{2.30}$$

which leads to the observations to be formally defined as

$$\mathbf{L} = \mathbf{L}^0 + \mathbf{l}, \tag{2.31}$$

where, $\mathbf{l}$ is the shortened observation vector

$$\mathbf{l} = \mathbf{L} - \mathbf{L}^0. \tag{2.32}$$

It is common to think of a vector $\mathbf{v}$ as a vector of improvements that, combined with the shortened observation vector $\mathbf{l}$, allows to define a functional model in the following matrix form:

$$\mathbf{l} + \mathbf{v} = \mathbf{A}\hat{\mathbf{x}}, \tag{2.33}$$

where $\mathbf{A}$ is the matrix of partial derivatives that maps to the Jacobian $\mathbf{J}$ in the previous sections. The goal of our optimization procedure is to minimize the needed improvements $\mathbf{v}$ by minimizing the expression $\mathbf{v}^\top\mathbf{v}$, from Equation (2.33). This results in the final equation

$$\mathbf{A}^\top\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^\top\mathbf{l}, \tag{2.34}$$

that we can solve with respect to $\hat{\mathbf{x}}$ to minimize the needed improvements.

To account for the uncertainty of the measurements, we use the Gauss-Markov model. The covariance matrix $\mathbf{\Sigma}_{ll}$ of this model is defined through a variance $\sigma_o^2$ that represents an absolute precision with which we can measure a particular modality, and the cofactor matrix $\mathbf{Q}_{ll}$, such that

$$\mathbf{\Sigma}_{ll} = \sigma_o^2\mathbf{Q}_{ll}. \tag{2.35}$$

Here, the inverse of the cofactor matrix corresponds to the information matrix presented in the previous sections

$$\mathbf{Q}_{ll}^{-1} = \mathbf{\Lambda}. \tag{2.36}$$

In adjustment models, this matrix is usually called the weighting matrix $\mathbf{P}$ of the observations vector. This matrix consists of diagonal elements, each of which is a variance of a particular modality $\sigma_o^2/\sigma_i^2$ and can be homogenized as

$$\mathbf{\Lambda} = \mathbf{P} = \mathbf{P}^{\frac{1}{2}}\mathbf{P}^{\frac{1}{2}}. \tag{2.37}$$

Multiplying the Equation (2.33) by the matrix $\mathbf{P}^{\frac{1}{2}}$ from both sides, and minimizing the value of $\mathbf{v}^\top\mathbf{v}$ using the resulting equation, allows us to rewrite the Equation (2.34) in the following form:

$$\mathbf{A}^\top\mathbf{P}\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^\top\mathbf{P}\mathbf{l}, \tag{2.38}$$

which directly maps onto Equation (2.28). Therefore, without the loss of generality, we can follow the notation presented in Section 2.1.4 in the remainder of this thesis for all problems that require a least-squares solution.


Figure 2.1: *Left:* Microsoft Kinect (image courtesy of Microsoft), *Right:* Asus Xtion (image courtesy of Asus)

## 2.2 Depth and range images

In our work we heavily rely on range images. These images can be obtained in two ways. They can be provided by a sensor directly or can be generated as a projection of a 3D point cloud. There is also a way to generate a depth image from a stereo RGB image pair but this is not considered in this thesis.

### 2.2.1 Sensors that produce range data

Typical sensors that provide a range image out of the box are Microsoft Kinect and Asus Xtion as shown in Figure 2.1. These sensors generate depth data by emitting an infrared pattern and perceiving the distortion of this pattern.

Example images that are produced by such sensors can be seen in Figure 2.2. Every pixel of a range image contains a distance to the object in the real world. Because the image has rigid pixel structure and a depth value for each pixel, it allows to reconstruct 3D representation of the environment measured by the sensor.

A slightly less common case is to reconstruct a range image from a 3D scene or from a LiDAR sensor. A spherical or a cylindrical projection can be used in order to project the 3D points onto an image plane to generate a (virtual) range image. In this thesis, we use Velodyne and Ocular LiDARs shown in Figure 2.3. The main method for generating a range image from these LiDARs is to fix the angular resolution of the horizontal and the vertical viewport, thus mapping

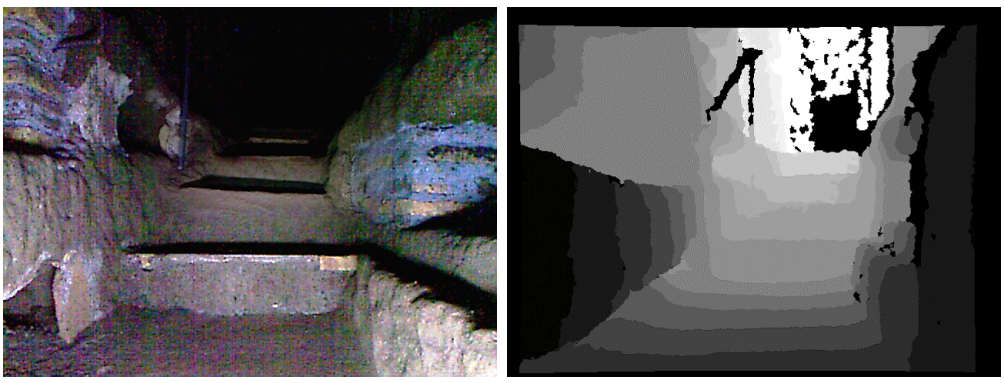

Figure 2.2: Data from a Microsoft Kinect sensor. *Left:* RGB, *Right:* Depth


Figure 2.3: *Left:* 16-beam Velodyne PUCK LiDAR (image courtesy of Velodyne), *Middle:* 64-beam Velodyne HDL-64E LiDAR (image courtesy of Velodyne) *Right:* Ocular Robotics RobotEye RE05 3D LiDAR (image courtesy of Ocular Robotics).

range readings to discrete pixels of a range image. Using the angular resolution of the laser beams provided by the sensor manufacturer, we can minimize beam overlapping, thus generating a nearly perfect mapping from beams to a range image.

This results in images that look as presented in Figure 2.4. Note, that the range images generated from a Velodyne sensor span full view of 360 degrees around the sensor and wrap on the border, i.e., an object on the right border of an image can also appear on the left of the image. In this particular example the wall that goes out of the image on the left continues on the right.

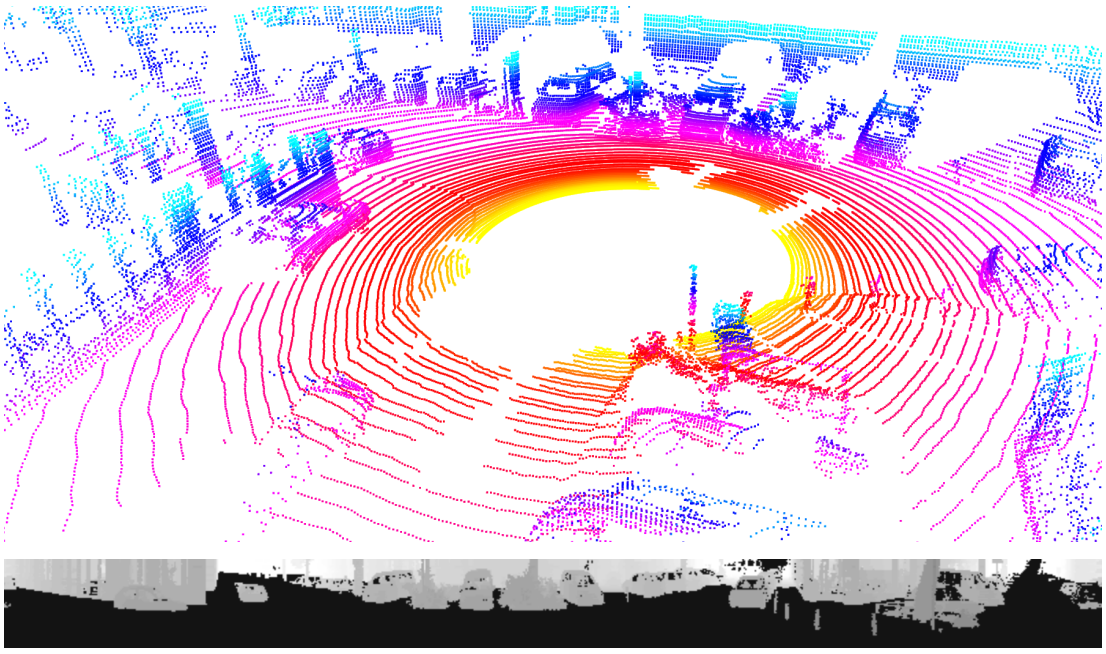

Figure 2.4: *Top:* 3D representation of a single LiDAR scan, *Bottom:* range image constructed from the same data. The part of the image that corresponds to the floor is removed for clarity.

### 2.2.2 Efficient data representation

Range images are useful to perform efficient computations on 3D data. Note, that usually range images are considered to be 2.5D data representation, as the 3D points that fall into the same pixel under a projection operation, have to be discretized into the pixel coordinates, and each pixel has a single range value stored in it. However, in the case of Velodyne LiDARs, we can guarantee that the range image contains all data from the 3D scene as we have control over how these data are generated. The Velodyne sensor rotates a vertical array of 16, 32, 64 or 128 beams to produce the full spherical scene representation. The angular resolution of the placement of the beams and the speed of rotation are given as parameters of the sensor, thus, we can tailor the size of the range image to match the number of beams up to a negligible error in the laser firing timings.

Range images are a useful representation of the 3D data as they allow for an extremely efficient neighbor search with $\mathcal{O}(1)$ complexity. We are making use of this property in multiple parts of this thesis. First, we propose a generalized framework for photometric matching of point clouds. The proposed method runs at the frame rate of the sensor partially due to the fact that we can avoid searching for neighbors of each point in 3D, which is a typical step in point cloud registration methods. This simplification is only possible because we are using the image representation of the 3D data. Additionally we make use of range images in the clustering section of this thesis, where we process the data received from a Velodyne sensor to find clusters of objects of interest in the vicinity of the robot. We achieve performance comparable to a standard Euclidean clustering while having the runtime performance orders of magnitude faster. This speed up comes from the fact that we analyze the neighbors of a point on a grid defined by the range image. As the neighbor search runs in constant time, it allows us to process all data coming from the sensor at up to 667 Hz. Lastly, we extend our clustering method by adding a fast quality assessment method to detect if the clouds of two objects match. We also make extensive use of range images in this part of this work. Range image representation is again responsible for quick execution of our algorithm on data from a real world dataset, and for quick estimation of the free space around the object.

### 2.2.3 Fast normal computation

Range image representation allows for fast and effective normal direction computation. Classically, to compute a normal vector of a point, a plane is fitted to the 3D points within some neighborhood of the query point. The normal is then computed as a direction of the normal vector to this plane. This mathematically sound approach, however, comes at a cost as it requires searching for neighbors for

every point. This algorithm, at best, has an asymptotic complexity of $\mathcal{O}(n \log n)$, where $n$ is the number of the points in the whole point cloud. When working on 3D data and the number of points is high, the runtime of algorithms that rely on searching for neighbors in 3D is doomed to deteriorate. As the mobile robots are usually computationally constrained, it is a desirable feature to avoid the potentially costly operations.

The range image representation outlined in the previous section is strictly structured as opposed to the unordered 3D point cloud representation. The points that lie close in the real world get projected to the nearby pixels in the range image representation. This allows us to perform the neighbor search in the image plane. Because the image has a rigid grid structure, neighborhood search has the complexity of $\mathcal{O}(1)$ which has a desirable property of being independent from the number of points in the point cloud.

Our approaches exploit information about the surface normals of the perceived environment. Thus, the first operation we perform is to compute a normal vector for each 3D point using the depth image. For a point $\mathbf{p}$, we can compute its normal by considering a Gaussian distribution with parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, which models the distribution of the neighboring points of $\mathbf{p}$. The eigenvector of $\boldsymbol{\Sigma}$, which corresponds to the smallest of the three eigenvalues, provides the direction of the normal.

We apply a fast method for normal extraction that exploits the structure of the input similar to Holzer *et al.* [61], and take advantage of the underlying hardware. We can rewrite the computation of the mean and the covariance matrix as:

$$\mu = \frac{1}{N} \underbrace{\sum_{i} \mathbf{p}_i}_{\mathbf{P}} = \frac{1}{N}\mathbf{P} \tag{2.39}$$

$$\Sigma = \frac{1}{N} \underbrace{\sum_{i} \mathbf{p}_i \mathbf{p}_i^\top}_{\mathbf{S}} - \boldsymbol{\mu}\boldsymbol{\mu}^\top = \frac{1}{N}\mathbf{S} - \boldsymbol{\mu}\boldsymbol{\mu}^\top, \tag{2.40}$$

where $N$ is the number of neighbors of the point $\mathbf{p}$. If we know the terms $\mathbf{P}$ and $\mathbf{S}$, we can compute the covariance matrix and thus the normals in constant time. By exploiting the fact that we are using a depth image as the input, we can realize the computation of $\mathbf{P}$ and $\mathbf{S}$ in constant time too through the use of integral images.

The integral image $\mathcal{I}(i, j) \rightarrow (\mathbf{P}_{ij}, \mathbf{S}_{ij})$ maps the pixel coordinates $(i, j)$ to the tuple $(\mathbf{P}_{ij}, \mathbf{S}_{ij})$ such that

$$\mathbf{P}_{ij} = \sum_{k=1}^{i} \sum_{l=1}^{j} \mathbf{p}_{kl} \qquad \mathbf{S}_{ij} = \sum_{k=1}^{i} \sum_{l=1}^{j} \mathbf{p}_{kl} \mathbf{p}_{kl}^\top, \tag{2.41}$$

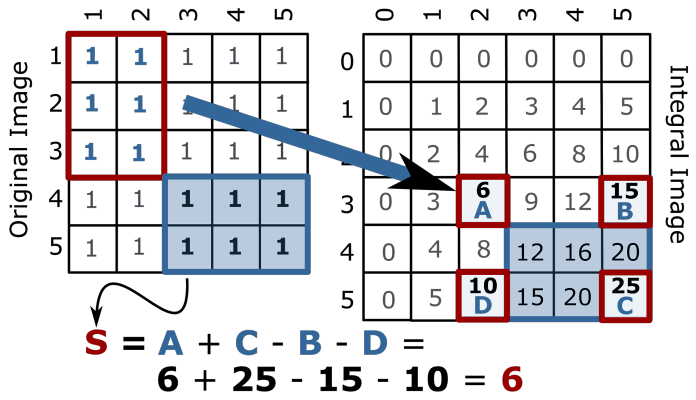


Figure 2.5: *Left:* an image filled with integer pixel values. *Right:* an integral image computed from the image on the left. The value stored in the pixel labeled $A$ on the right is the sum of all pixels in a red rectangle on the left. The sum of all elements in a blue rectangle from the original image is computed by the provided formula.

where $\mathbf{p}_{kl}$ corresponds to the 3D point, which is observed by the pixel $(k, l)$. Thus, this integral image is a 2D array where the position $(i, j)$ contains the sum $\mathbf{P}_{ij}$ and the squared sum $\mathbf{S}_{ij}$ of all other points visible in the rectangular area between $(1, 1)$ and $(i, j)$. Exploiting the fact that the terms $\mathbf{S}$ and $\mathbf{P}$ support addition and subtraction, we can determine $\mathbf{S}$ and $\mathbf{P}$ for any rectangular region $(i_1, j_1)$ to $(i_2, j_2)$ in the integral image by

$$\mathcal{I}(i_1, j_1, i_2, j_2) = \mathcal{I}(i_2, j_2) + \mathcal{I}(i_1, j_1) - \mathcal{I}(i_1, j_2) - \mathcal{I}(i_2, j_1). \tag{2.42}$$

By processing the depth image from the top left to the lower right corner, each computation exploits the result of the previous step leading to a constant time computation of $\mathbf{P}_{ij}$ and $\mathbf{S}_{ij}$ for each pixel. Figure 2.5 depicts an integral image computed from a toy example image filled with integer values.

Note that the computation of the normals through the integral image is an approximation. The reason is that only rectangular image regions can be queried with Equation (2.42) and the queries are performed in image coordinates and not in world coordinates. The appropriate size of the region, however, can easily be determined given the distance of the query point to the camera.

The implementation of the algorithm described above can furthermore exploit the SSE extensions of Intel CPUs by operating on packed structures of four floats. The points are stored as homogeneous vectors $\mathbf{p} = [x\ y\ z\ 1]^\top$. The terms $\mathbf{P}$ and $\mathbf{S}$ are stored as a 4-dimensional vector and as a $4 \times 4$ matrix respectively. Adding points to an accumulator $\mathbf{P}$ using this representation, results in a homogeneous component that contains the number of points. This has the effect of removing conditional instructions and of performing the evaluation almost entirely in the SSE subsystem of the CPU. This provides a speed up of a factor of 3 compared to an implementation of the same algorithm without SSE instructions.

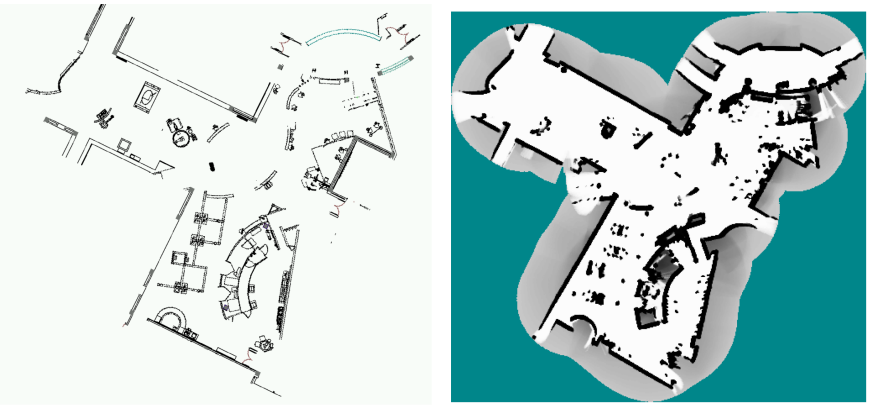

Figure 2.6: This image illustrates an occupancy grid map of an environment. *Left:* a CAD rendering of a floor plan, *Right:* an occupancy grid map of the same environment build with a 2D LiDAR. Cyan shows parts of the map that have never been updated, and the occupancy probability in each cell scales from occupied (black) to free (white). Image courtesy of Thrun *et al.* [128].

## 2.3 Grid maps

There are different ways to represent a map of an environment, for example, maps can be represented as either feature maps or grid maps. Each of the representations has its own advantages and best use cases.

Feature maps store the features detected in the environment and match these features for localization and map updates. A typical source of features range from lines and corners for pure 3D features to other invariant features detected from images in case a camera is used in adjacency, or instead of a 3D sensor.

Feature maps are the standard for bundle adjustment and are useful for localization, but are generally less useful for constructing and viewing the map in a format that is easy to perceive by a human. In addition to that, the features from which the map is constructed must be computed while a robot navigates in an environment, which can take time especially in a computationally constrained system. Therefore, in this thesis, we focus explicitly on the grid maps to represent the environment. Grid maps are a popular representation of an environment used in both static and dynamic environments. A particular variant of grid maps are the so-called occupancy grid maps. These maps were originally constructed in 2D, but can also be trivially extended to 3D world. In this thesis we mostly will use the 2D version of occupancy grids.

Occupancy grid maps store a single value per cell. This value represents the probability of the particular cell to be occupied by an obstacle. The map then gets updated with the new incoming measurements. An example of such a map can be found in Figure 2.6. In this map shown on the right of the figure, white stands for free space, black for occupied and cyan depicts parts of the map that

have never been observed. In this section we will present the original formulation of the occupancy grid maps as presented by Moravec and Elfes [84]. We will then refer to this section from the other parts of this thesis that either use or extend this formulation.

In this thesis we use the occupancy grid maps in their classical formulation for navigation and exploration tasks described in Section 4.2 as well as for checking the consistency of the map constructed by the robot described in Section 4.3. In addition to the classical formulation of occupancy used within the cells of the grid maps, we extend these maps to hold additional information like traversability. We refer the reader to Section 4.1 for more details on our traversability analysis method.

Occupancy grid maps are used to create a grid map given known poses and sensor measurements in those poses. The main idea is to categorize the environment into the empty space and into that occupied by obstacles. Grid maps store a single value $p(c)$ for each cell $c$ which denotes the probability of this particular cell to be occupied by an obstacle. The algorithm developed by Moravec and Elfes takes into account a sequence of observations $z_{1:t}$ obtained by the robot at poses $x_{1:t}$. The subscripted notation $x_t$ denotes that the robot had the pose $x_t$ at time $t$ and notation $x_{1:t}$ denotes a sequence of such poses. It seeks to maximize the occupancy probability for the grid map. A single most important assumption that Moravec and Elfes rely on, is that the map cells are independent. This is not entirely true, and in reality it is likely more probable for cells neighboring occupied cells to also be occupied, but this assumption is necessary in order to be able to compute the posterior of the occupancy probability, as can be seen in the remainder of this section. The assumption of independence allows us to formulate the probability of the map $m$ to be given by the product over all probabilities of individual cells

$$p(m) = \prod_{c \in m} p(c). \tag{2.43}$$

This formulation allows us to focus on estimating the occupancy probability of the individual cells $c \in m$ given the measurements $z_{1:t}$ performed by the robot at poses $x_{1:t}$. This probability is hard to measure directly, but it can be rewritten making use of the Bayes' rule

$$p(c \mid x_{1:t}, z_{1:t}) \;=\; \frac{p(z_t \mid c, x_{1:t}, z_{1:t-1})\; p(c \mid x_{1:t}, z_{1:t-1})}{p(z_t \mid x_{1:t}, z_{1:t-1})}. \tag{2.44}$$

Assuming that $z_t$ is independent from $x_{1:t-1}$ and $z_{1:t-1}$ leads to

$$p(c \mid x_{1:t}, z_{1:t}) \;=\; \frac{p(z_t \mid c, x_t)\; p(c \mid x_{1:t}, z_{1:t-1})}{p(z_t \mid x_{1:t}, z_{1:t-1})}. \tag{2.45}$$

We apply Bayes' rule for the term $p(z_t \mid c, x_t)$ in Equation (2.45) and obtain

$$p(z_t \mid c, x_t) = \frac{p(c \mid x_t, z_t)\ p(z_t \mid x_t)}{p(c \mid x_t)}. \tag{2.46}$$

We can combine Equation (2.45) and Equation (2.46). In addition to that we can assume assume that $x_t$ does not carry any information about $c$ if there is no observation $z_t$. We can safely make this assumption as we can only gain new information about the environment if we make a measurement in a given position. Just residing in a specific part of an environment making no observation adds no information about the state of the map. Following this assumption, we can write

$$p(c \mid x_{1:t}, z_{1:t}) = \frac{p(c \mid x_t, z_t)\ p(z_t \mid x_t)\ p(c \mid x_{1:t-1}, z_{1:t-1})}{p(c)\ p(z_t \mid x_{1:t}, z_{1:t-1})}. \tag{2.47}$$

Each cell of the environment is assumed to be in only one of two possible states: free or occupied. We therefore make use of the logical negation and estimate the probability of cell $c$ being in an opposite state to the one estimated in Equation (2.47)

$$p(\neg c \mid x_{1:t}, z_{1:t}) = \frac{p(\neg c \mid x_t, z_t)\ p(z_t \mid x_t)\ p(\neg c \mid x_{1:t-1}, z_{1:t-1})}{p(\neg c)\ p(z_t \mid x_{1:t}, z_{1:t-1})}. \tag{2.48}$$

At this point we can make use of a trick and divide Equation (2.47) by Equation (2.48) and we obtain

$$\frac{p(c \mid x_{1:t}, z_{1:t})}{p(\neg c \mid x_{1:t}, z_{1:t})} = \frac{p(c \mid x_t, z_t)\ p(\neg c)\ p(z_t \mid x_{1:t}, z_{1:t-1})}{p(\neg c \mid x_t, z_t)\ p(c)\ p(z_t \mid x_{1:t}, z_{1:t-1})}. \tag{2.49}$$

Finally, we use the fact that $p(\neg c) = 1 - p(c)$ which yields

$$\frac{p(c \mid x_{1:t}, z_{1:t})}{1 - p(c \mid x_{1:t}, z_{1:t})} = \frac{p(c \mid x_t, z_t)}{1 - p(c \mid x_t, z_t)} \cdot \frac{1 - p(c)}{p(c)} \cdot \frac{p(c \mid x_{1:t-1}, z_{1:t-1})}{1 - p(c \mid x_{1:t-1}, z_{1:t-1})}. \tag{2.50}$$

Given all of the above equations, we can specify the full *occupancy update formula* as follows:

$$p(c \mid x_{1:t}, z_{1:t}) = \left[1 + \frac{1 - p(c \mid x_t, z_t)}{p(c \mid x_t, z_t)} \cdot \frac{p(c)}{1 - p(c)} \cdot \frac{1 - p(c \mid x_{1:t-1}, z_{1:t-1})}{p(c \mid x_{1:t-1}, z_{1:t-1})}\right]^{-1}. \tag{2.51}$$

Equation (2.51) tells us how to update our belief $p(c \mid x_{1:t}, z_{1:t})$ about the occupancy probability of a grid cell given sensory input. In practice, one often assumes that the occupancy prior is 0.5 for all cells, so that $\frac{p(c)}{1-p(c)}$ can be removed from the equation.

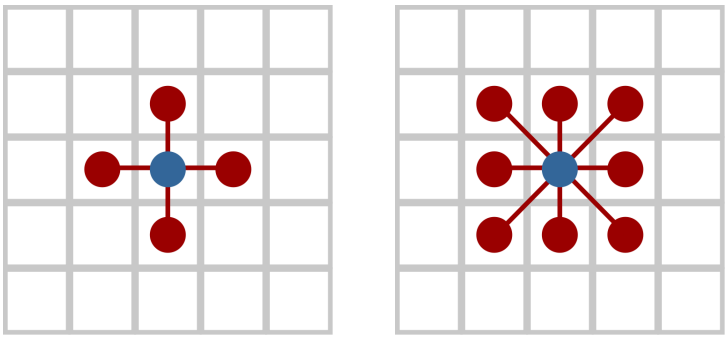

Figure 2.7: Neighborhoods defined on a grid. *Left:* 4-neighborhood (N4), *Right:* 8-neighborhood (N8).

In this section, we will skip the computation of the occupancy probability $p(c \mid x_t, z_t)$ of a grid cell given a *single* observation $z_t$ and the corresponding pose $x_t$ of the robot. This quantity depends on the sensor of the robot and has to be defined manually for each type of sensor. For more information on grid maps we refer the reader to the original version of the Moravec and Elfes paper [84]. In Section 4.1 we specify a modification of the standard occupancy grid estimation technique to store traversability information of the environment.

## 2.4 Graph algorithms on a grid

When using grid maps as presented in Section 2.3 or range images from Section 2.2 it is often helpful to define a graph over the cells of the grid. In case of grid maps these graphs can be used for robot navigation while in case of range images these graphs can be used in various image processing techniques.

The vertices of such graphs are the cells of the grid and the edges are defined based on the neighborhood that is of interest for a particular problem. In this thesis we will be mostly using 4-neighborhoods and 8-neighborhoods shown in Figure 2.7. Defining a graph over a given grid allows to move onto a more abstract layer of reasoning about the problem by substituting problems like searching for the closest path between two cells or detecting connected components in an image with well-studied problems on the graphs.

In this thesis, we use two different grid representations. One is used for planning a route of the robot and is defined over a traversability grid, presented in detail in Section 4.1, while the other is defined on the pixels of the image and is used for clustering the scene into distinct objects by finding connected components on a specifically defined image-based grid. The clustering approach is presented in detail in Section 5.1.

In this section we introduce the basic algorithms that are used in this thesis. We base all path planning capabilities of our system on the A* algorithm

**Algorithm 1** Breadth-First Graph Traversal

```
1: procedure BFS(N_0)
2:     Queue: Q ← N_0
3:     Open set: O ← {N_0}
4:     Visited set: V ← ∅
5:     Parent map: P ← ∅
6:     while Queue Q is not empty do
7:         N_current ← Q.top()
8:         V ← [V, N_current]
9:         for N_neighbor ∈ neighborhood(N_current) do
10:            if N_neighbor ∉ V and N_neighbor ∉ O then
11:                O ← [O, N_neighbor]
12:                Q.push(N_neighbor)
13:                P[N_neighbor] ← N_current
14:        Q.pop()
15:    return V, P
16: procedure GetPath(P, N_0, N_target)
17:    Path ← ∅
18:    N_current ← N_target
19:    while N_current ≠ N_0 do
20:        Path ← [Path, N_current]
21:        N_current ← P[N_current]
22:    return Path
```

as presented below in Section 2.4.3 and our algorithm for searching connected components on the breadth-first search algorithm outlined in Section 2.4.1.

### 2.4.1 Breadth-first search

A typical approach to traversing an unweighted graph is the breadth-first search (BFS) algorithm. This algorithm is most commonly used to either find a shortest path between two nodes on an unweighted graph or to efficiently traverse connected components of the graph. Algorithm 1 shows an approximate sequence of actions to perform the breadth-first search on an arbitrary unweighted graph. It can be briefly summarized by the following sequence of actions. The algorithm makes use of a queue $Q$, i.e., a first in first out (FIFO) data structure to store the nodes. It starts in an initial node $N_0$ being the only node in the queue $Q$. Firstly, the current node is removed from the queue and stored in the visited set, then, depending on the connectivity of the graph, the neighbors of the current node are added to the queue. The same approach is then repeated for the next node in the

queue until it becomes empty. This detects a single connected component of the graph given the connectivity rules between the grid cells. If the closest path is needed, in addition to the visited set we must maintain the "parent" from which each node is reached for the first time. This way when the target node has been added to the queue we can stop the execution of the algorithm, and backtrack the shortest path from the initial node to the target node.

### 2.4.2 Dijkstra's algorithm

The breadth-first search algorithm, as described in Section 2.4.1, works well for unweighted graph, however, there is no way to determine the shortest path using the presented formulation of BFS for a weighted graph. In the path planning tasks that we deal with in this thesis, we usually have a graph defined over an occupancy or traversability grid. We take traversability of the environment into account in order to find the most suitable path for the robot to take. The traversability information can be seen as weights added to a 2D grid representing our environment. The most basic solution to searching the shortest path from the root to the goal in a weighted graph is the Dijkstra's algorithm. It can be seen as an extension of the BFS algorithm. The original implementation [29] focused on the shortest path between two nodes in the graph, while later extensions have focused on producing a shortest-path tree.

The performance of this algorithm is very similar to the BFS and is outlined in Algorithm 2. The main difference is in the data structure that is being used at the core of the algorithm to store the nodes of the graph prior to their opening. Instead of using a simple FIFO queue as in BFS, Dijkstra's algorithm can be implemented using a priority queue. The priority queue is a data structure that stores the elements in a sorted manner, guaranteeing that an element with the best score is the first one to be popped from the queue. This allows to seamlessly incorporate the distance to a particular cell or additional traversability information into the existing BFS algorithm. Instead of simply putting a new node into the queue, the cell is now put into the priority queue with the corresponding weight being the distance to this node. This guarantees that the queue is being emptied with respect to the distance to the nodes that are stored within it, guaranteeing that the first time we visit a particular node we reach it with the shortest path. Therefore, Dijkstra's algorithm has been a default choice for searching the shortest path on the graph since the 1960's until the introduction of the A* algorithm presented in the next section.

**Algorithm 2** Dijkstra's Graph Traversal

```
1: procedure DIJKSTRA(N_0)
2:     Priority queue: Q ← {N_0, 0}
3:     Visited set: V ← ∅
4:     Open set: O ← {N_0}
5:     Parent map: P ← ∅
6:     while Queue Q is not empty do
7:         N_current, d_current ← Q.top()
8:         V ← [V, N_current]
9:         for N_neighbor ∈ neighborhood(N_current) do
10:            if N_neighbor ∈ O then
11:                d^O_neighbor ← distance to a vertex in the open set O
12:            else
13:                d^O_neighbor ← ∞
14:            if N_neighbor ∉ V and d^O_neighbor > d_current + d_neighbor then
15:                O ← [O, N_neighbor]
16:                Q.push(N_neighbor, d_current + d_neighbor)
17:                P[N_neighbor] ← N_current
18:            P[N_neighbor] ← N_current
19:        Q.pop()
20:    return V, P
```

### 2.4.3 A* algorithm

Dijkstra's algorithms outlined in Section 2.4.2 finds the shortest path in a weighed graph, but does so in a suboptimal way. When uncovering the nodes formed from grid cells, Dijkstra's algorithm tends to expand the cells in a circular fashion as can be seen on the left side of Figure 2.8. There are multiple states that had to be evaluated in order to find the correct shortest path, most of them in the direction opposite to the direction of the goal node. It is therefore preferable to allow guiding the search in a particular direction. For this purpose, Hart *et al.* [54] presented the A* algorithm in 1968. A* has quickly become a popular algorithm for path planning on a graph. This algorithm is an extension of the Dijkstra's path planning algorithm. It builds upon that algorithm by introducing a heuristic that allows to guide the search in a particular direction. This algorithm uses the same method and data structures as the Dijkstra's algorithm but does the weighting of the nodes stored in the priority queue in a different way.

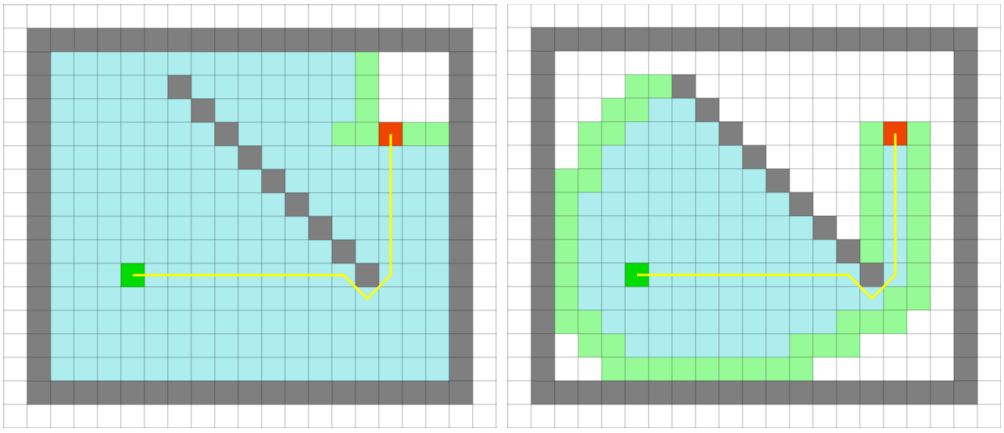

Figure 2.8: A comparison between the performance of two algorithms: Dijkstra's on the left and A* on the right. Both algorithms search for a path from a green cell to the red one. The dark cells represent obstacles, white stand for free space, light blue cells are the nodes "visited" by each algorithm in order to find the shortest path, the light green ones represent the open set and the yellow line depicts the found shortest path. Images courtesy of Xu [139].

The A* algorithm makes use of the heuristic function and the weight of each node upon adding it to the priority queue is modified with the heuristic function

$$f(n) = g(n) + h(n), \tag{2.52}$$

where $n$ is the node being added to the priority queue, $g(n)$ is the cost of the path from the start node to node $n$ and $h(n)$ is the heuristic that estimates the cost of the potential cheapest path from $n$ to the goal. For the algorithm to find the actual shortest path, the heuristic function must be admissible, i.e. it should never overestimate the actual cost to get to the goal node. A typical heuristic function used for graphs defined over a grid is the Euclidean distance between the two nodes. The A* algorithm is the same as Dijkstra's with the difference of using Equation (2.52) in lines 14 and 16 of Algorithm 2. A small illustration of the difference in rate of expansion between the A* and Dijkstra's algorithm can be found in Figure 2.8. The A* algorithm on the right visits fewer vertices on the grid while delivering the same shortest path.

## 2.5 Coordinate frames

When talking about sensors placed on the robot that moves through the world, it is of crucial importance to explicitly define all coordinate frames. In this thesis, we mainly use three different coordinate frames: sensor, robot, and world reference frames.

The sensor coordinate frame depends on the sensor being used in a particular domain. In this thesis, we mostly focus on two types of sensors: RGBD cameras and LiDARs. These sensors usually use different coordinate systems. Figure 2.9 showcases the two different coordinate systems that we use throughout this thesis. All the RGBD cameras use the coordinate system depicted on the left with $z$ axis pointing forward from the sensor, $x$ axis pointing right and $y$ axis — down from the sensor, while all the LiDARs use the one shown on the right with $x$ axis pointing forward, $z$ axis pointing up and $y$ axis pointing to the left. These coordinate systems differ solely in orientation.

The typical robot coordinate frame is defined in the point on the floor level below the center of mass of the robot. This coordinate frame is very similar to the one used for LiDARs, shown on the right of Figure 2.9, i.e., the $x$ axis points to the front of the robot, $y$ axis to the left and $z$ axis points up, and differs from the LiDAR one only in translation.

When we talk about the world coordinate frame we are not talking about the global geo-referenced frame in this thesis, but rather about an arbitrary general frame to which we transform all the local measurements the robot makes during the time of its operation. Without loss of generality, we can set the world frame to be the frame of the first robot pose, i.e., the one when the robot starts its operation. We will refer to the world frame by this definition in this thesis.

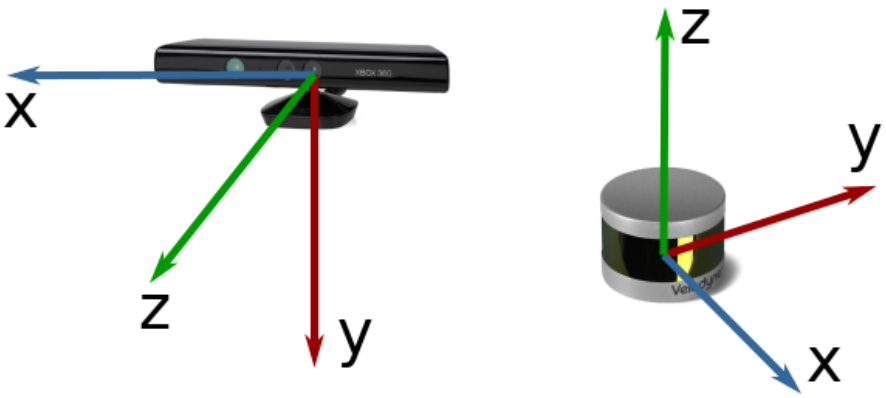


Figure 2.9: The sensor coordinate systems we use in this thesis. *Left:* the classical computer vision coordinate system with z axis pointing forward from the sensor. *Right:* a coordinate system that we use for LiDARs and mapping.

# Part I

# Static environments

# Chapter 3

# Registration and pose estimation

Mobile robots navigating in an unknown environment need to analyze their surroundings as well as to estimate a consistent map of this environment in order to navigate safely. A classical approach to estimating a consistent static map is to perform graph-based simultaneous localization and mapping (SLAM). Here, every pose of the robot is a node on a graph and every edge between these poses holds information about the relative movement as well as the uncertainty of this movement. Additionally, when the robot observes a place where it has already been, it can add a loop closing edge to the graph. These edges add inconsistencies into the pose-graph as, usually, robot motion estimate is not as precise as the robot observation model. Whenever these edges are added to the pose graph, the graph can be optimized on-the-fly using any convex optimization technique outlined briefly in Section 2.1.

SLAM is a well studied topic in the robotics community and is not the main contribution of this thesis. For a comprehensive analysis of available methods, we refer the reader to a chapter on SLAM by Stachniss *et al.* [119]. In this chapter any graph-based SLAM approach that provides a consistent map of the environment, and can use the edges coming from point cloud matching, can be used with equal success.

Our interest lies in the first step of the graph construction, i.e., pose estimation through sequential point cloud registration. In order to build the initial graph the robot must have a way of estimating a relative motion from one pose to another. The default source of such information is the odometry information provided by the tracks or wheels. One downside of using odometry, however, is that it lacks precision, especially when driving over slippery terrain, and when the robot uses tracks.

Thus, the task of registering the incoming sensor data, such as images or point clouds, is an important building block for most autonomous systems. This functionality is also of key importance for estimating the relative motion of a

robot through incremental matching, often called visual odometry or laser-based odometry, depending on the used sensing modality. In this work, we seek to develop an approach that performs equally well in different environments and under different circumstances, i.e., ideally it should work both underground and above the ground level, in a broad daylight with direct sunlight, and with no light at all. This requirement eliminates the possibility of using the RGB sensor as the main sensor for the robot. We focus on using depth and range sensors, such as RGBD cameras and LiDARs in this work. These sensors are all capable of producing point clouds and range images that can be registered with generic algorithms, thus fulfilling the requirement of being usable under different lighting conditions.

This chapter focuses on estimating the pose of the robot incrementally through sequential point cloud matching. Our method is designed to be sensor-agnostic and utilizes the 2.5D range image data representation to achieve the state-of-the-art photometric matching quality for RGBD cameras, as well as for LiDARs, while being able to register the incoming data at the frame rate of the sensor.

## 3.1 Incremental point cloud matching

### 3.1.1 Multi-cue photometric point cloud matching

In this section of this thesis, we investigate the problem of registering data from typical robotic sensors such as a Microsoft Kinect camera, a 3D LiDAR such as a Velodyne laser scanner, or similar, in a general way without requiring special, sensor-specific adaptations. More concretely, our goal is to provide a general methodology to finding the transformation that maximizes the overlap between two measurements taken from the same scene exploiting as much sensor information as is being provided by the hardware.

To this extent, ICP is a popular strategy for registering point clouds. It proceeds by iteratively alternating two steps: data association and transformation estimation. Data association computes pairs of corresponding points in the two clouds, while transformation estimation calculates an isometry that, applied to one of the two clouds, minimizes the distance between corresponding points. The weakness of ICP lies in the correspondence search as this step usually relies on expensive 3D neighborhood search, heuristics, and may introduce biases or gross errors.

Over time, effective variants of ICP that exploit the structure of the scene have been proposed [107, 109]. Using the structure either by relying on a point-to-plane or a plane to plane metric has been shown to improve the performance of the algorithm, especially in indoor/structured environments. Kerl *et al.* [67]

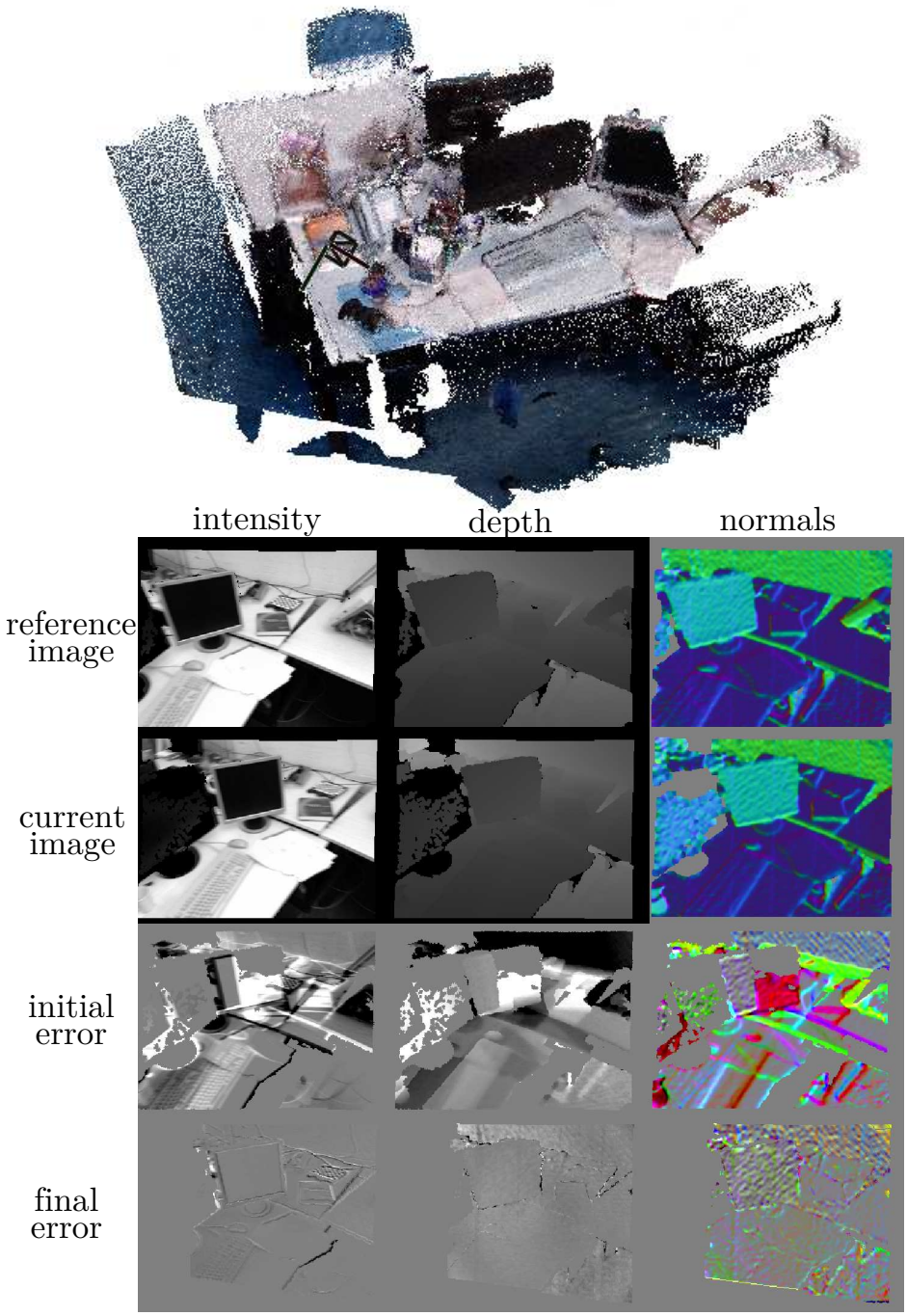


Figure 3.1: Our approach implements an effective multi-cue registration without requiring features nor an explicit data association between 3D points. *Top:* A 3D scene reconstructed using point clouds registered with our method. *Bottom:* A grid of images showing the intensity, depth and normals input channels for reference and current images in the first two rows of the image grid, and showing the initial per-channel error as well as final error after registration visulized in color in the bottom rows.

recently proposed dense visual odometry (DVO), an approach to register RGBD images by minimizing the photometric distance. The idea is to find the location of a camera within a scene such that the image captured at that location is as close as possible to the measured RGBD image. By exploiting the image gradients, DVO does not require explicit data association, and it is able to achieve good accuracy given a good initial guess. The main shortcoming of DVO is that it is restricted to the use of depth/intensity image pairs. In its original formulation DVO does not naturally incorporate additional structural cues such as normals or curvature. The use of these cues has been shown to substantially enlarge the basin of convergence and reduce the number of iterations needed to find a solution.

The main contribution of this section is a general methodology to photometric sensor data registration that works on cues such as color, depth, and normal information in a unified way and does not require an explicit data association between features, 3D points, or surfaces. The approach is inspired by DVO [67] as it does not require any feature extraction, and operates directly on the image or image-like data obtained from a sensor such as the Microsoft Kinect or a 3D LiDAR. A key property of our approach is an easy-to-extend, mathematically sound framework for registration that does not need to make an explicit data association between sensor readings and the 3D model. In contrast, it solves the registration problem as a minimization in the color, depth, and normal data exploiting projections of the sensor data. It can be seen as a generalization of DVO, with added ability to handle arbitrary cues and multiple sensing modalities in a flexible way. An example of this registration is illustrated in Figure 3.1.

In sum, we make the following key claims: our approach (i) is a general methodology for photometric registration that works transparently with different sensor cues and avoids an explicit point-to-point or point-to-surface data association, (ii) can accurately register typical sensor cues such as RGBD or LiDAR data exploiting the color, depth, and normal information, (iii) robustly computes the transformation between view points under realistic disturbances of the initial guess, and (iv) can be executed fast enough to allow for online processing of the sensor data without sensor-specific optimizations. These four claims are backed up through the experimental evaluation on real-world data.

We also provide an open source C++ implementation that follows the descriptions in this paper, available here:

`https://gitlab.com/srrg-software/srrg_mpr`

### 3.1.2 Our approach to point cloud registration

Our approach seeks to register either two sensor readings with respect to each other or a sensor reading against a 3D model. The sensor observations are assumed to have an image-like representation $\mathfrak{I}$, such as an image from a regular camera, a range image from a depth camera or a 3D LiDAR, or a similar type of observation. Such a 2D measurement can be seen as an image, where each pixel in the image plane contains one or more channels, i.e., $\mathfrak{I} = \{\mathfrak{I}^c\}$ with the channel index $c$. Examples of such channels are light intensity, depth information, or surface normals.

We aim at registering the current observation to a model, for which 3D information is available in the form of a point cloud. This model can be a given 3D model, or a point cloud estimated from the previous observation(s). We refer to it as the model cloud $\mathcal{M} = \{\mathbf{p}\}$. In addition to the 3D coordinates, each point $\mathbf{p} \in \mathcal{M}$ can also store multiple cues such as light intensity or a surface normal.

The following subsections describe the key ingredients of our approach Section 3.1.2.1 presents the overall error minimization formulation that uses three functions, which are sensor and/or cue-specific and *must be implemented by the user* when adding a new cue or a different sensor, everything else is handled by our framework. The outline of the components and sections where these components are described can also be found in Figure 3.2. The following parts of the pipeline need to be implemented by the user of the library:

- Cue mapping function $\text{map}^c(\dots)$ that describes if and how a cue is transformed through a coordinate transformation. See Section 3.1.2.2 for details.
- Projection model $\text{proj}(\dots)$ of the sensor, e.g. the pinhole camera model, see Section 3.1.2.3 for details.
- The Jacobians corresponding to the projection and mapping functions. See Section 3.1.2.5 for details.

#### 3.1.2.1 Photometric error minimization

As in photometric error minimization approaches, our method seeks to iteratively minimize the pixel-wise difference between the current image $\mathfrak{I}$ and the predicted image $\hat{\mathfrak{I}}(\mathcal{M}, \mathbf{X})$. Here, $\hat{\mathfrak{I}}(\mathcal{M}, \mathbf{X})$ is a multi-channel image obtained by projecting the model $\mathcal{M}$ onto a virtual camera located at such a position in the world, that a transformation matrix $\mathbf{X}$ transforms the points of $\mathcal{M}$ from the world into the

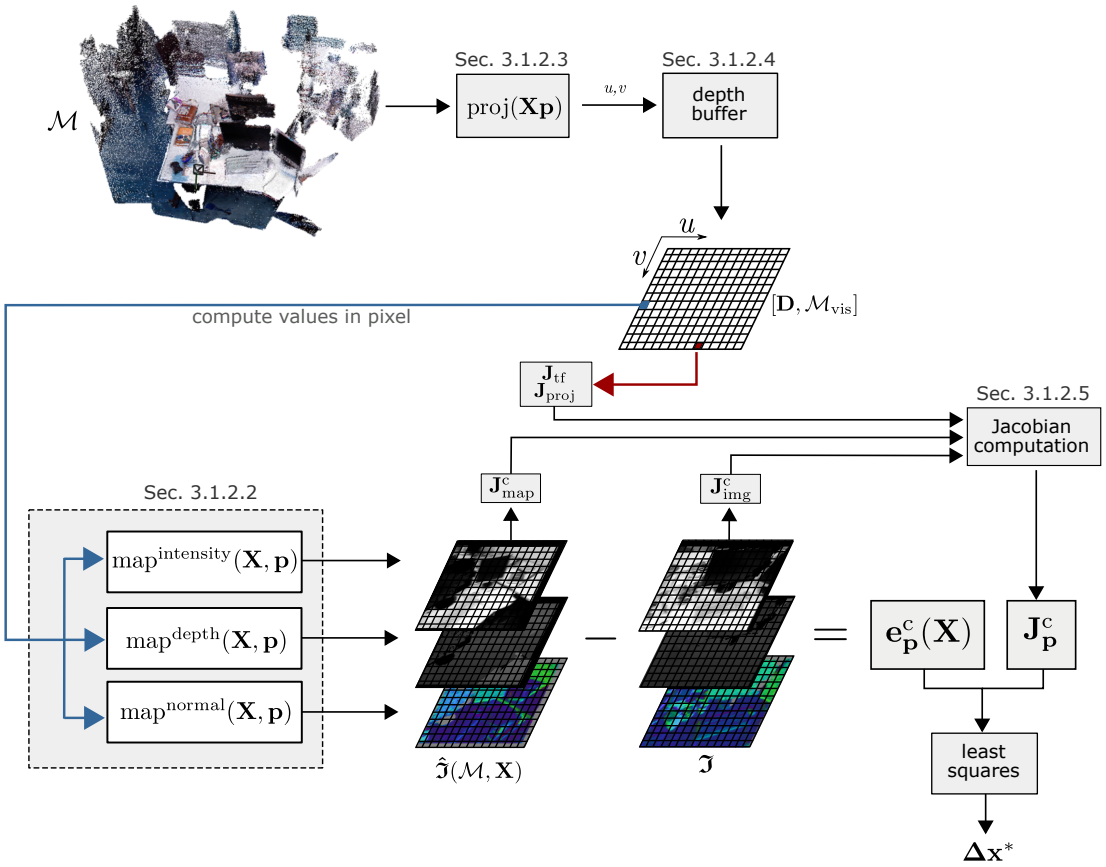


Figure 3.2: Key ingredients of our framework and links to the corresponding subsections. We start by projecting a model $\mathcal{M}$ onto its virtual observation $\hat{\mathfrak{I}}(\mathcal{M}, \mathbf{X})$ from a sensor at position defined by the transformation $\mathbf{X}$. This projection is generated by applying the projection $\text{proj}(\dots)$ and mapping $\text{map}^{c}(\dots)$ functions to each point $\mathbf{p}$ of the model. We then define the error between this virtual observation and a real one $\mathfrak{I}$, which allows to formulate the minimization procedure of the resuting photometric error in terms of least-squares error minimization.

local camera coordinate system. More formally, our method seeks to minimize

$$\mathbf{X}^{*} = \underset{\mathbf{X}}{\text{argmin}} \sum_{u,v,c} \| \underbrace{\hat{\mathfrak{I}}_{u,v}^{c}(\mathcal{M}, \mathbf{X}) - \mathfrak{I}_{u,v}^{c}}_{\mathbf{e}_{u,v}^{c}(\mathcal{M}, \mathbf{X})} \|_{\mathbf{\Lambda}^{c}}^{2} \tag{3.1}$$

$$= \underset{\mathbf{X}}{\text{argmin}} \sum_{u,v,c} \mathbf{e}_{u,v}^{c}(\mathcal{M}, \mathbf{X})^{\top} \, \mathbf{\Lambda}^{c} \, \mathbf{e}_{u,v}^{c}(\mathcal{M}, \mathbf{X}), \tag{3.2}$$

where $\mathbf{e}_{u,v}^{c}(\mathcal{M}, \mathbf{X})$ denotes an error at a pixel location $(u \; v)^{\top}$ between the predicted value $\hat{\mathfrak{I}}_{u,v}^{c}(\mathcal{M}, \mathbf{X})$ and the measured one $\mathfrak{I}_{u,v}^{c}$ for a particular channel index $c$. The matrix $\mathbf{\Lambda} = \text{diag}(\{\mathbf{\Lambda}^{c}\})$ is a block diagonal information matrix used to weight the different channels of the image.

Let $\mathcal{M}_{vis}$ be the subset of points from the model $\mathcal{M}$ that are visible from the image plane of image $\mathfrak{I}$. In our current implementation, we construct this set using the depth buffer as explained in details in Section 3.1.2.4. Given $\mathcal{M}_{vis}$, we

can rewrite the sum in Equation (3.1) using the points:

$$\mathbf{X}^* \;=\; \underset{\mathbf{X}}{\operatorname{argmin}} \sum_{\mathrm{c}\in\mathcal{C},\mathbf{p}\in\mathcal{M}_{\text{vis}}} \left\|\mathbf{e}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X})^\top\right\|^2_{\mathbf{\Lambda}^{\mathrm{c}}}. \tag{3.3}$$

Each point-wise error term $\mathbf{e}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X})^\top$ is the difference between a predicted and a measured channel evaluated at the pixel where the point $\mathbf{p}$ projects onto given the model of the sensor. We expand the point-wise error as follows:

$$\mathbf{e}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X}) \;=\; \mathrm{map}^{\mathrm{c}}(\mathbf{X},\mathbf{p}) - \mathfrak{I}^{\mathrm{c}}_{\mathrm{proj}(\mathbf{X}\,\mathbf{p})}. \tag{3.4}$$

The term $\mathrm{proj}(\mathbf{X}\,\mathbf{p}) = (u\ v)^\top$ is a function that computes the image coordinates obtained by projecting the point $\mathbf{p}$ onto a camera located at $\mathbf{P}_{\mathbf{X}}$. The function $\mathrm{map}^{\mathrm{c}}(\mathbf{X},\mathbf{p})$ computes the *value* of channel $\hat{\mathfrak{I}}^{\mathrm{c}}(\mathcal{M},\mathbf{X})$ evaluated at the pixel $\mathrm{proj}(\mathbf{X}\,\mathbf{p})$. During a first read, that may sound confusing as the default cue light intensity is not affected by the transformation and are simply copied from the information in the point $\mathbf{p}$. Other cues, however, such as normals, or depth values are viewpoint-dependent, and therefore change depending on the current camera pose.

Our approach minimizes Equation (3.3) by using a regularized least squares optimization procedure, described in Section 2.1.4. Combining Equation (3.3) with Equation (3.4) and adding a per-point regularization weight $w_{\mathbf{p}}$, we can rewrite Equation (3.1) in terms of points as

$$\mathbf{X}^* = \underset{\mathbf{X}}{\operatorname{argmin}} \sum_{\mathbf{p}\in\mathcal{M}_{\text{vis}}} w_{\mathbf{p}} \sum_{\mathrm{c}\in\mathcal{C}} \left\|\mathrm{map}^{\mathrm{c}}(\mathbf{X},\mathbf{p}) - \mathfrak{I}^{\mathrm{c}}_{\mathrm{proj}(\mathbf{X}\,\mathbf{p})}\right\|^2_{\mathbf{\Lambda}^{\mathrm{c}}}, \tag{3.5}$$

where the regularization weight $w_{\mathbf{p}}$ decreases with the magnitude of the channel errors $\mathbf{e}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X})$ and is used to reject outliers. The minimization is performed using a local perturbation:

$$\boldsymbol{\Delta x} = (\underbrace{\Delta t_x, \Delta t_y, \Delta t_z}_{\boldsymbol{\Delta t}}, \underbrace{\Delta\alpha_x, \Delta\alpha_y, \Delta\alpha_z}_{\boldsymbol{\Delta\alpha}})^\top, \tag{3.6}$$

consisting of a 3D translation vector $\boldsymbol{\Delta t}$ and three Euler angles $\boldsymbol{\Delta\alpha}$. The vector $\boldsymbol{\Delta x}$ is a minimal representation for the transformation matrix $\mathbf{X}$ and Equation (3.3) can be reduced to a quadratic problem in $\boldsymbol{\Delta x}$ by computing the Taylor expansion of the error function around a null perturbation as follows:

$$\mathbf{e}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X} \boxplus \boldsymbol{\Delta x}) \;\simeq\; \mathbf{e}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X}) + \left.\frac{\partial \mathbf{e}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X}\boxplus\mathbf{x})}{\partial \mathbf{X}}\right|_{\mathbf{x}=\mathbf{0}} \boldsymbol{\Delta x} \tag{3.7}$$

$$= \underbrace{\mathbf{e}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X})}_{\breve{\mathbf{e}}^{\mathrm{c}}_{\mathbf{p}}} + \underbrace{\mathbf{J}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X})}_{\breve{\mathbf{J}}^{\mathrm{c}}_{\mathbf{p}}} \boldsymbol{\Delta x} \;=\; \breve{\mathbf{e}}^{\mathrm{c}}_{\mathbf{p}} + \breve{\mathbf{J}}^{\mathrm{c}}_{\mathbf{p}} \boldsymbol{\Delta x}. \tag{3.8}$$

The $\boxplus$ operator is the transform composition defined as

$$\mathbf{X} \boxplus \boldsymbol{\Delta x} = \underbrace{\begin{bmatrix} \boldsymbol{\Delta R} & \boldsymbol{\Delta t} \\ \mathbf{0} & 1 \end{bmatrix}}_{\boldsymbol{\Delta x}} \underbrace{\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}}_{\mathbf{X}}, \tag{3.9}$$

with $\boldsymbol{\Delta R}$ obtained by chaining rotations along the three axes:

$$\boldsymbol{\Delta R} = \mathbf{R}_x(\Delta\alpha_x)\mathbf{R}_y(\Delta\alpha_y)\mathbf{R}_z(\Delta\alpha_z). \tag{3.10}$$

Thus, the quadratic approximation of Equation (3.5) becomes

$$\boldsymbol{\Delta x}^* = \underset{\boldsymbol{\Delta x}}{\operatorname{argmin}} \sum_{\mathbf{p}\in\mathcal{M}_{\text{vis}}} w_{\mathbf{p}} \sum_{\mathrm{c}\in\mathcal{C}} \|\breve{\mathbf{e}}^{\mathrm{c}}_{\mathbf{p}} + \breve{\mathbf{J}}^{\mathrm{c}}_{\mathbf{p}} \boldsymbol{\Delta x}\|^2_{\boldsymbol{\Lambda}^{\mathrm{c}}}, \tag{3.11}$$

and $\boldsymbol{\Delta x}^*$ can be found by solving a linear system of the form $\mathbf{H}\boldsymbol{\Delta x}^* = \mathbf{b}$ with the term $\mathbf{H}$ and $\mathbf{b}$ given by

$$\mathbf{H} = \sum_{\mathbf{p}\in\mathcal{M}_{\text{vis}}} w_{\mathbf{p}} \sum_{\mathrm{c}\in\mathcal{C}} \breve{\mathbf{J}}^{\mathrm{c}\top}_{\mathbf{p}} \boldsymbol{\Lambda}^{\mathrm{c}} \breve{\mathbf{J}}^{\mathrm{c}}_{\mathbf{p}} \tag{3.12}$$

$$\mathbf{b} = \sum_{\mathbf{p}\in\mathcal{M}_{\text{vis}}} w_{\mathbf{p}} \sum_{\mathrm{c}\in\mathcal{C}} \breve{\mathbf{J}}^{\mathrm{c}\top}_{\mathbf{p}} \boldsymbol{\Lambda}^{\mathrm{c}} \breve{\mathbf{e}}^{\mathrm{c}}_{\mathbf{p}}. \tag{3.13}$$

To limit the magnitude of the perturbation between iterations, thus enforcing a smoother convergence, we solve a damped linear system of the form

$$(\mathbf{H} + \lambda\mathbf{I})\,\boldsymbol{\Delta x}^* = \mathbf{b}. \tag{3.14}$$

Solving Equation (3.14) yields a perturbation $\boldsymbol{\Delta x}^*$ that minimizes the quadratic problem. A new solution $\mathbf{X}^*$ is found by applying $\boldsymbol{\Delta x}^*$ to the previous solution $\mathbf{X}$ as follows:

$$\mathbf{X}^* = \mathbf{X} \boxplus \boldsymbol{\Delta x}^*. \tag{3.15}$$

This subsection provided the overall minimization approach and in the remainder of this section, we describe the parts of our approach in detail. These details are the cue-specific mapping function and the projection model. Furthermore, we explain how to use the depth buffer to construct $\mathcal{M}_{\text{vis}}$ and finally discuss the structure of the Jacobians and a pyramidal approach to the optimization.

#### 3.1.2.2 Cue mapping functions

In contrast to traditional ICP approaches but similar to DVO [67], our method does not use an explicit point-to-point data association procedure. In addition to that, by abstracting the cues into channels of the image, our method can be extended to deal with an arbitrary number of cues and can benefit from all the available information. In our current implementation, we consider the following

cues: *intensity*, *depth*, *range* and *normals*. Note that additional cues or further sensor information can be added easily without changing the framework.

In this subsection, we provide the mapping functions $\text{map}^{\text{c}}(\dots)$ for the used cues intensity, depth, range, and normals that depend on $\mathbf{X}$. We will use $\mathbf{R}$ and $\mathbf{t}$ as defined in Equation (3.9) as the rotation matrix and translation vector of $\mathbf{X}$.

**Intensity:** The intensity is not a geometric property of the point and thus it is not affected by the transformation defined through $\mathbf{X}$, therefore the intensity value of a point $\mathbf{i_p}$ is considered invariant under the map function, i.e.,

$$\text{map}^{\text{intensity}}(\mathbf{X}, \mathbf{p}) = \mathbf{i_p}. \tag{3.16}$$

**Depth:** The depth cue of a point is the $z$-component of the point transformed by $\mathbf{X}$, i.e.,

$$\text{map}^{\text{depth}}(\mathbf{X}, \mathbf{p}) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} (\mathbf{R}\mathbf{p} + \mathbf{t}). \tag{3.17}$$

**Range:** The range cue of a point is the norm of the point transformed by $\mathbf{X}$, i.e.,

$$\text{map}^{\text{range}}(\mathbf{X}, \mathbf{p}) = \|\mathbf{R}\mathbf{p} + \mathbf{t}\|. \tag{3.18}$$

**Normals:** The normal cue of a point $\mathbf{p}$ is the normal vector specified by $\mathbf{n_p}$ at the point rotated (but not translated) by $\mathbf{X}$, i.e.,

$$\text{map}^{\text{normal}}(\mathbf{X}, \mathbf{p}) = \mathbf{R}\,\mathbf{n_p}. \tag{3.19}$$

#### 3.1.2.3 Projection models

The projection function maps a 3D point from the model cloud onto a coordinate in a 2D image. In our implementation, we provide two projective models, the *pinhole* and the *spherical* model. The pinhole model better captures the characteristics of imaging sensors such as RGBD cameras, while the spherical model is entailed to 3D LiDARs. In this sub-section, we describe these two models but note that the framework easily extends to other types of projection functions, such as the cylindrical model. Only a single function needs to be overridden.

**Pinhole model:** Let $\mathbf{K}$ be the camera matrix. Then, the pinhole projection of a point $\mathbf{p}$ is computed as

$$\text{proj}^{\text{pinhole}}(\mathbf{p}) = \pi(\mathbf{K}\,\mathbf{p}) \tag{3.20}$$

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

$$\pi(\mathbf{v}) = \frac{1}{v_z} \begin{bmatrix} v_x \\ v_y \end{bmatrix}, \tag{3.22}$$

with the intrinsic camera parameters for the focal length $f_x$, $f_y$ and the principle point $c_x$, $c_y$. The function $\pi(\mathbf{v})$ is the homogeneous normalization.

**Spherical model:** Let $\mathbf{K}$ be a camera matrix in the form of Equation (3.21), where $f_x$ and $f_y$ specify respectively the resolution of azimuth and elevation and $c_x$ and $c_y$ their offset in pixels. Then, the spherical projection of a point is given by

$$\text{proj}^{\text{spherical}}(\mathbf{p}) = \mathbf{K} \begin{bmatrix} \text{atan2}(\mathbf{p}_y, \mathbf{p}_x) \\ \text{atan2}\left(\mathbf{p}_z, \sqrt{\mathbf{p}_x^2 + \mathbf{p}_y^2}\right) \\ 1 \end{bmatrix} \tag{3.23}$$

#### 3.1.2.4 Computing visible points using depth buffers

As stated in the beginning of Section 3.1.2, Equation (3.1) and Equation (3.3) are equivalent if there are no occlusions or self-occlusions. This condition can be satisfied by removing all points from the model $\mathcal{M}$ that would be occluded after applying the projection. At each iteration, the estimated position $\mathbf{X}$ of the cloud with respect to the sensor changes, thus before computing the projection, we need to transform the model according to $\mathbf{X}$. Subsequently, we project each transformed point onto an image and for each pixel in the image, we preserve only the point that is the closest one to the observer according to the chosen projective model. The outcome of the overall procedure is a subset of the transformed points in the model that are visible from the origin.

The reduced set of non-occluded points can be effectively computed as follows. Let $\mathbf{D}$ be a 2D array of the size of the image, each cell $(u\ v)^\top$ of $\mathbf{D}$ contains a depth or range value referred to as $\mathbf{D}(u, v)$ and a model point $\hat{\mathbf{p}}$ that generated this depth or range reading. First, all depths $\mathbf{D}(u, v)$ are initialized with $\infty$. Then, we iterate over all points $\mathbf{p} \in \mathcal{M}$ and perform the following computations:

- Let $\mathbf{p}' = \mathbf{R}\mathbf{p} + \mathbf{t}$ be the transformed point and let $(u, v) = \text{proj}(\mathbf{p}')$ be the image coordinates of the point after the projection.

- If using the pinhole projective model, let $d' = \mathbf{p}'_z$ be the $z$ component of the transformed point. If using the spherical model, let $d' = \|\mathbf{p}'\|$ be its norm.

- We compare the range value $\mathbf{D}(u, v)$ previously stored in $\mathbf{D}$ with the range computed for the current point $d'$. If the latter is smaller than the former, we replace $\mathbf{D}(u, v)$ with $d'$ and $\hat{\mathbf{p}}$ with $\mathbf{p}'$.

At the end of the procedure, $\mathbf{D}$ contains all points of the model visible from the origin, i.e., those that are not occluded. These points form the $\mathcal{M}_{\text{vis}}$ cloud.

#### 3.1.2.5 Structure of the Jacobian

We want to highlight the modular structure of the Jacobian $\mathbf{J}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X})$, which is key for an efficient computation. By applying the chain rule to the right summand of Equation (3.7), and using the definition of the error function from Equation (3.4), we obtain the following form for the Jacobian $\mathbf{J}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X})$:

$$\begin{aligned}\mathbf{J}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X}) \;=\; & \overbrace{\left.\frac{\partial \mathrm{map}^{\mathrm{c}}(\mathbf{X} \boxplus \mathbf{x}, \mathbf{p})}{\partial \mathbf{x}}\right|_{\mathbf{x}=0}}^{\mathbf{J}^{\mathrm{c}}_{\mathrm{map}}} \\ & - \underbrace{\left.\frac{\partial \mathfrak{I}^{\mathrm{c}}_{u,v}}{\partial u, v}\right|_{u,v=\mathrm{proj}(\mathbf{X}\mathbf{p})}}_{\mathbf{J}^{\mathrm{c}}_{\mathrm{img}}} \underbrace{\left.\frac{\partial \mathrm{proj}(\breve{\mathbf{p}})}{\partial \breve{\mathbf{p}}}\right|_{\breve{\mathbf{p}}=\mathbf{X}\mathbf{p}}}_{\mathbf{J}_{\mathrm{proj}}} \underbrace{\left.\frac{\partial (\mathbf{X} \boxplus \mathbf{x})\,\mathbf{p}}{\partial \mathbf{x}}\right|_{\mathbf{x}=0}}_{\mathbf{J}_{\mathrm{tf}}}.\end{aligned} \tag{3.24}$$

Thus, the Jacobian can be compactly written as:

$$\mathbf{J}^{\mathrm{c}}_{\mathbf{p}}(\mathbf{X}) = \mathbf{J}^{\mathrm{c}}_{\mathrm{map}} - \mathbf{J}^{\mathrm{c}}_{\mathrm{img}}\,\mathbf{J}_{\mathrm{proj}}\,\mathbf{J}_{\mathrm{tf}} \tag{3.25}$$

Note that the multiplicative nature of Equation (3.25) in this formulation allows us to easily compute the overall Jacobian from its individual components $\mathbf{J}^{\mathrm{c}}_{\mathrm{map}}$, $\mathbf{J}^{\mathrm{c}}_{\mathrm{img}}$, $\mathbf{J}_{\mathrm{proj}}$ and $\mathbf{J}_{\mathrm{tf}}$. In particular, $\mathbf{J}_{\mathrm{tf}}$ does not depend on the channel, nor on the projective model. Similarly, $\mathbf{J}_{\mathrm{proj}}$ depends only on the projective model. The image Jacobian $\mathbf{J}^{\mathrm{c}}_{\mathrm{img}}$ can be computed directly from the image through a convolution for obtaining image gradients. Note that to increase the precision, we recommend to compute $\mathbf{J}^{\mathrm{c}}_{\mathrm{img}}$ with subpixel precision through bilinear interpolation during the optimization. Only the Jacobian $\mathbf{J}^{\mathrm{c}}_{\mathrm{img}}$ of the function $\mathrm{map}(\cdot)$ depends on the specific cue.

For completeness we provide more detail on each part of the Jacobian in Equation (3.25) and specify all terms used in this equation.

**3.1.2.5.1 Jacobian of transformation** We are using the $\mathtt{v2t}(\cdot)$ operator to denote the conversion between $\mathbf{X}$ and $\boldsymbol{\Delta x}$ as defined in Equation (3.6). By applying the $\mathtt{v2t}(\cdot)$ operator, we obtain the standard Jacobian:

$$\mathbf{J}_{\mathrm{tf}} = \left.\frac{\partial \mathtt{v2t}(\mathbf{x})\,\breve{\mathbf{p}}}{\partial \mathbf{x}}\right|_{\mathbf{x}=0} = \begin{bmatrix}\mathbf{I} & -\breve{\mathbf{p}}_{\times}\end{bmatrix} \tag{3.26}$$

where $\mathbf{I}$ is a $3 \times 3$ identity matrix and $\breve{\mathbf{p}}_{\times}$ denotes the skew-symmetric matrix formed from $\breve{\mathbf{p}}$.

**3.1.2.5.2 Jacobian of the mapping function** The mapping function Jacobian $\mathbf{J}^{\mathrm{c}}_{\mathrm{map}}$ differs for each of the considered channels c. In this work, we use

the following channels: *intensity*, *depth*, *range* and *normals*. In the following we present the Jacobian derivation of each of these cues.

**Intensity:** The mapping function does not affect the intensity, thus we have:

$$\left.\frac{\partial \operatorname{map}^{\text{intensity}}(\mathbf{X} \boxplus \mathbf{x}, \mathbf{p})}{\partial \mathbf{x}}\right|_{\mathbf{x}=0} = 0 \tag{3.27}$$

**Depth:** We have already computed the derivative of the transformation function $\mathbf{J}_{\text{tf}}$ in Equation (3.26). For RGBD sensors, the depth is computed as the $z$ coordinate of the transformed point $\mathbf{p}$. Thus, the Jacobian of the mapping function for the depth channel is the third row of $\mathbf{J}_{\text{tf}}$

$$\left.\frac{\partial \operatorname{map}^{\text{depth}}(\mathbf{X} \boxplus \mathbf{x}, \mathbf{p})}{\partial \mathbf{x}}\right|_{\mathbf{x}=0} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{J}_{\text{tf}} \tag{3.28}$$

**Range:** When using a 3D LiDAR, the range $r = \|\mathbf{p}\|$ replaces the depth. Thus, the Jacobian $\mathbf{J}_{\text{map}}^{\text{range}}$ is computed as

$$\begin{aligned}
\mathbf{J}_{\text{map}}^{\text{range}} &= \left.\frac{\partial \operatorname{map}^{\text{range}}(\mathbf{X} \boxplus \mathbf{x}, \mathbf{p})}{\partial \mathbf{x}}\right|_{\mathbf{x}=0} \\
&= \left.\frac{\partial \left\|\mathtt{v2t}(\mathbf{x})\, \mathbf{X}\mathbf{p}\right\|}{\partial \mathbf{x}}\right|_{\mathbf{x}=0} \\
&= \left.\frac{\partial \|\breve{\mathbf{p}}\|}{\partial \breve{\mathbf{p}}}\right|_{\breve{\mathbf{p}}=\mathbf{X}\mathbf{p}} \left.\frac{\partial\, \mathtt{v2t}(\mathbf{x})\, \mathbf{X}\mathbf{p}}{\partial \mathbf{x}}\right|_{\mathbf{x}=0} \\
&= \left.\frac{\partial \|\breve{\mathbf{p}}\|}{\partial \breve{\mathbf{p}}}\right|_{\breve{\mathbf{p}}=\mathbf{X}\mathbf{p}} \mathbf{J}_{\text{tf}} \\
&= \frac{1}{\|\mathbf{p}\|} \begin{bmatrix} \mathbf{p}_x & \mathbf{p}_y & \mathbf{p}_z \end{bmatrix} \cdot \mathbf{J}_{\text{tf}}
\end{aligned} \tag{3.29}$$

**Normals:** The mapping function applied to the normals cue depends on the rotational part of $\mathbf{X}$ and the Jacobian is given by

$$\mathbf{J}_{\text{map}}^{\text{normal}} = \left.\frac{\partial \operatorname{map}^{\text{normal}}(\mathbf{X} \boxplus \mathbf{x}, \mathbf{p})}{\partial \mathbf{x}}\right|_{\mathbf{x}=0} = \begin{bmatrix} \mathbf{0} & -[\mathbf{R}\, \mathbf{n}_{\mathbf{p}}]_{\times} \end{bmatrix}, \tag{3.30}$$

where $\mathbf{R}$ denotes a rotation matrix, $\mathbf{n}_{\mathbf{p}}$ the normal defined for the point $\mathbf{p}$, and $[\mathbf{R}\, \mathbf{n}_{\mathbf{p}}]_{\times}$ the skew-symmetric matrix.

**3.1.2.5.3 Image Jacobian** The Image Jacobian $\mathbf{J}_{\text{img}}^{\text{c}}$ is numerically computed for each channel c with pixel-wise derivation:

$$\begin{aligned}
\frac{\partial\, \mathfrak{I}_{u,v}^{\text{c}}}{\partial u} &= \frac{1}{2}\left(\mathfrak{I}_{u+1,v}^{\text{c}} - \mathfrak{I}_{u-1,v}^{\text{c}}\right) \\
\frac{\partial\, \mathfrak{I}_{u,v}^{\text{c}}}{\partial v} &= \frac{1}{2}\left(\mathfrak{I}_{u,v+1}^{\text{c}} - \mathfrak{I}_{u,v-1}^{\text{c}}\right)
\end{aligned} \tag{3.31}$$

**3.1.2.5.4 Jacobian of the projection function** The Jacobian of the projection depends directly on the projection function $\mathbf{J}_{\text{proj}} = \frac{\partial \text{proj}(\mathbf{p})}{\partial \mathbf{p}}$. In the following, we provide details for the two models given in Section 3.1.2.3.

**Pinhole model:** We derive the projection function Jacobian for a pinhole camera from Equation (3.22):

$$\begin{aligned} \mathbf{J}_{\text{proj}}^{\text{pinhole}} &= \frac{\partial \, \text{proj}(\mathbf{p})}{\partial \, \mathbf{p}} && (3.32) \\ &= \frac{\partial \, \pi(\mathbf{p}')}{\partial \, \mathbf{p}'}\bigg|_{\mathbf{p}'=\mathbf{K}\,\mathbf{p}} \frac{\partial \mathbf{K}\,\mathbf{p}}{\partial \mathbf{p}} && (3.33) \\ &= \frac{1}{z^2}\begin{bmatrix} z & 0 & -x \\ 0 & z & -y \end{bmatrix}\bigg|_{(x,y,z)=\mathbf{K}\,\mathbf{p}} \mathbf{K} \end{aligned}$$

**Spherical model:** The projection function for the range sensors is defined in Equation (3.23). We make use of the substitution $\mathbf{a}_2 = \sqrt{\mathbf{p}_x^2 + \mathbf{p}_y^2}$, and define the Jacobian as follows:

$$\mathbf{J}_{\text{proj}}^{\text{spherical}} = \frac{\partial \, \text{proj}(\mathbf{p})}{\partial \, \mathbf{p}} = \begin{bmatrix} \frac{1}{\mathbf{a}_2^2}\begin{bmatrix} -\mathbf{p}_y & \mathbf{p}_x & 0 \end{bmatrix} \\ \frac{1}{\mathbf{a}_2^2+\mathbf{p}_z^2}\begin{bmatrix} -\frac{\mathbf{p}_x\,\mathbf{p}_z}{\mathbf{a}_2} & -\frac{\mathbf{p}_y\,\mathbf{p}_z}{\mathbf{a}_2} & \mathbf{a}_2 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \end{bmatrix} \mathbf{K} \tag{3.34}$$

### 3.1.2.6 Hierarchical approach to photometric minimization

A central challenge for photometric minimization approaches is the choice of the resolution in order to optimize the trade-off between the size of the convergence basin and the accuracy of the solution. A high image resolution has a positive effect on the accuracy of the solution given the initial guess lies in the convergence basin. This is due to the fact that more measurements are taken into account and the precision of the typical sensor is exploited in a better way. A high resolution, however, often comes at the cost of reducing the convergence basin since the iterative minimization can get stuck more easily in a local minimum arising from a high frequency spectral component of the image. Thus, using lower resolution, the photometric approach exhibits an increased convergence basin at the cost of a lower precision.

In our implementation, we leverage on these considerations by using a pyramidal approach. The optimization is performed at different resolutions, starting from low to high resolutions. After convergence on one level, the optimization switches to the next level by increasing the resolution. The optimization on the higher resolution uses the solution from the lower level as the initial guess. Our termination criterion for the optimization on each level of this resolution pyramid analyzes the evolution of the value of the objective function in Equation (3.3), normalized by the number of inliers. We stop the iterations at one level if this

value does not decrease between two subsequent iterations or if a maximum number of iterations is reached.

### 3.1.3 Experimental evaluation

Our method is a general and efficient framework for multi-cue sensor data registration. We implemented and released a C++ implementation that closely follows the description presented in this chapter as open source software. Our general methodology works for RGBD sensors, such as the Microsoft Kinect and 3D laser scanners using the following cues: intensity, depth, range, and surface normal. Thus, the evaluation is done based on these sensors and cues. Note, that further cues or similar sensors can be added easily.

The evaluation presented here is designed to support the remaining three claims made in the introduction. To simplify comparisons, we conducted our experiments on publicly available datasets:

- TUM benchmark suite [126], acquired with RGBD sensors in office-like environments.
- S. Gennaro catacomb dataset [108], recorded with a RobotEye 3D LiDAR in a catacomb environment.
- KITTI dataset [44], where we used the Velodyne HDL-64E data recorded in urban environments in the context of autonomous driving.

Furthermore, we provide comparisons to the state-of-the art approaches DVO and NICP. The outcomes of these comparisons highlight that our method, although being general and relatively easy to implement, yields an accuracy that is comparable to those achieved by systems dedicated to specific setups.

#### 3.1.3.1 Registration performance and comparison

This section is designed to show that our method can accurately register typical sensor cues such as RGBD, as provided by Microsoft Kinect, or LiDAR data exploiting the color, depth, and normal information and that it can do so under realistic disturbances of the initial guess. The first experiment is designed to evaluate the accuracy of our approach. To do that, we rely on RGBD data and provide a comparison to dense visual odometry (DVO) [67], which is the current state-of-the-art method and the one most closely related to our work. We used the three available channels, namely the intensity, the depth and the normals.

We performed the comparison on the four desk sequences also used in [67] and report the relative pose error (RPE). For DVO, we used the author's open

| Approach / setup | fr1/desk2 | | fr1/desk | | fr2/desk | | fr2/person | |
|---|---|---|---|---|---|---|---|---|
| | [m/s] | [deg/s] | [m/s] | [deg/s] | [m/s] | [deg/s] | [m/s] | [deg/s] |
| DVO (as reported in [67]) | 0.0687 | - | 0.0491 | - | 0.0188 | - | 0.0345 | - |
| DVO (implementation) | 0.0700 | 5.14 | 0.0580 | 3.83 | 0.0318 | 1.15 | 0.0360 | 0.99 |
| Our approach | 0.0920 | 5.14 | 0.0614 | 3.32 | 0.0365 | 1.65 | 0.0481 | 1.45 |
| DVO (without intensity cue) | - | - | - | - | - | - | - | - |
| Our approach (without intensity cue) | 0.1073 | 5.20 | 0.0630 | 3.66 | 0.0329 | 1.52 | 0.0411 | 1.27 |

Table 3.1: Relative Pose Error on TUM desk sequences.

source implementation[1]. For completeness, we report in Table 3.1 both, the results presented in the paper [67] and the ones we obtained with the open source implementation. All values have been computed using the evaluation script provided with the TUM benchmark suite. Our approach provides slightly lower but overall comparable performance than DVO yielding a low relative error both, in terms of translation and rotation, respectively in the order of $10^{-2}$ m/s and 1 deg/s, see Table 3.1.

We conducted a second experiment with the TUM dataset, where we removed the intensity channel, using only the depth images and the normals derived from the depth image to perform the registration. By exploiting the normal cue, our approach provides results consistent to the ones obtained when using also the intensity, see last two rows on Table 3.1. In contrast to that, DVO was unable to perform the registration without the intensity channel. This is coherent with the operating conditions which DVO was designed for, and at the same time supports the general applicability of our method to different cues.

The third experiment aims at showing the effectiveness of our algorithm when dealing with dense 3D laser data. We used a dataset acquired in the S. Gennaro catacomb of Naples within the EU project ROVINA. The data was recorded with an Ocular RobotEye RE05 3D laser scanner using a maximum range of 30 m. Since the dataset does not provide ground truth, we performed a qualitative comparison with Normal ICP (NICP) by Serafin *et al.* [110].

The 3D point clouds have been recorded in a stop and go fashion at an average distance of about 1.7 m between two consecutive scans. The robot has tracks and this provides a rather poor odometry information. This odometry is used as the initial guess for the optimization and the same guess was also used for NICP, that

[1] `https://github.com/tum-vision/dvo_slam`

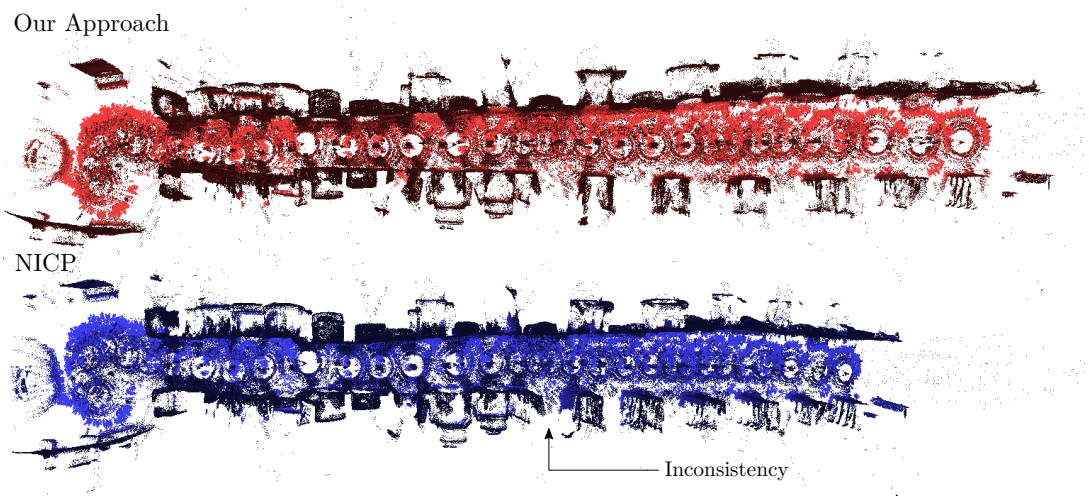


Figure 3.3: S. Gennaro catacombs dataset, recorded with a RobotEye 3D LiDAR. Direct comparison of our approach with Normal ICP (NICP). The latter shows a registration inconsistency in the middle of the trajectory, with an unexpected change of the orientation around the vertical axis.

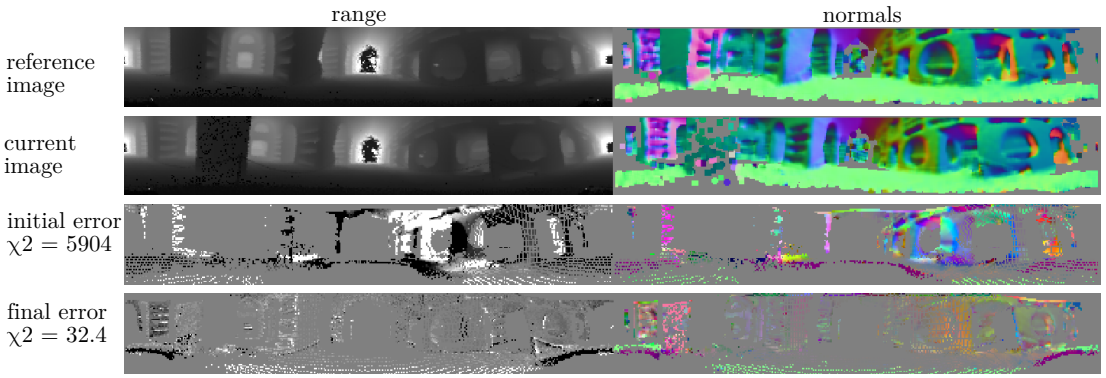


Figure 3.4: Error reduction while registering two scans from S. Gennaro dataset (best viewed on screen).

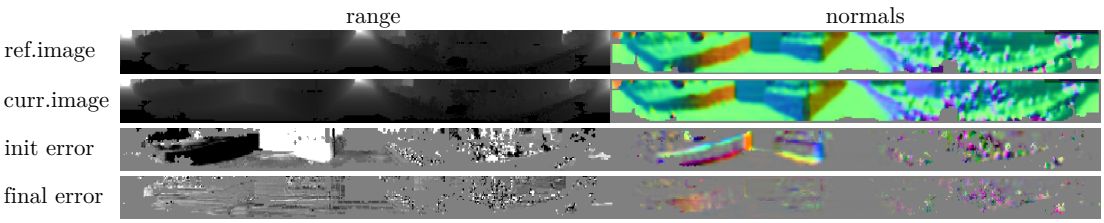


Figure 3.5: Illustration of the error before and after registering two images of the KITTI dataset. The images are obtained by projecting the Velodyne HDL-64 clouds using a spherical projector (best viewed on screen).

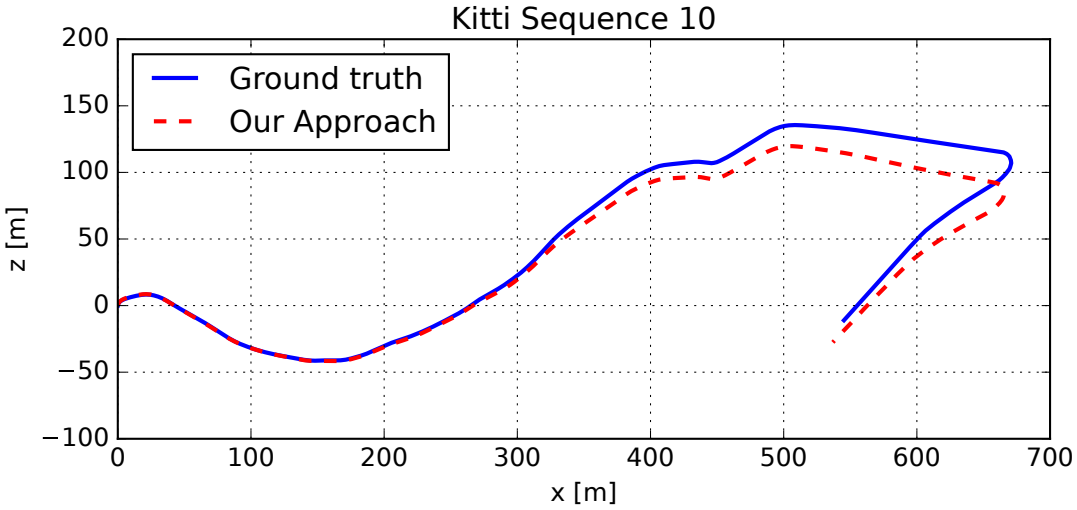


Figure 3.6: Ground truth comparison for sequence 10 of the KITTI dataset.

is used for comparisons in this experiment. To perform the registration we use both, range channel and normals channel computed on the range image. There is no intensity information available for this dataset.

Figure 3.3 illustrates the two reconstructions of one of the sequences of the S. Gennaro catacombs, while Figure 3.4 shows the error of a pairwise registration before and after the alignment. Thanks to the abstraction provided by the mapping function and the projective model (see Section 3.1.2.2 and Section 3.1.2.3), our method can deal with both RGBD data and 3D scans in a uniform manner.

To further stress the generality of our method, we conducted an additional experiment using exactly the same code and parameters on sequence 10 of the KITTI dataset [44]. We used the data from a Velodyne HDL-64E LiDAR, using range and normals cues. Figure 3.5 illustrates the error reduction.

As shown in Figure 3.6, our output reflects the ground truth with an error of the 6.1% of the trajectory length, with a rotational error of 0.023 deg/m. Albeit reasonable, this accuracy is still below the one provided by approaches dedicated to the sparse LiDAR such as LOAM [143]. We see the reason for this lower performance in the quantization effects affecting the projections when the clouds

| Sensor | **laptop computer** i7-3630MQ 2.4 GHz | **desktop computer** i7-7700K 4.2 GHz |
|---|---|---|
| Microsoft Kinect | 49 ms ± 0.8 ms ≈ 20.1 Hz | 28 ms ± 0.2 ms ≈ 35.5 Hz |
| RobotEye | 324 ms ± 1.5 ms ≈ 3.1 Hz | 189 ms ± 0.9 ms ≈ 5.2 Hz |
| HDL-64E | 67 ms ± 0.3 ms ≈ 14.8 Hz | 39 ms ± 0.1 ms ≈ 25.2 Hz |

Table 3.2: Average image processing runtime with std. deviation.

are sparse. We plan to address this aspect in future versions by using anisotropic projection functions. Moreover, approaches such as LOAM take advantage from edge and planar surface features, the use of which particularly helps in cases of structure lack.

#### 3.1.3.2 Runtime

The next set of experiments is designed to support the last claim, namely that our approach can be executed fast enough to allow for online processing of the sensor data without sensor-specific optimizations. Thus, we report in the remainder of this section the runtime statistics. We performed all the presented experiments on two different computers running Ubuntu 16.04. One is a laptop equipped with a i7-3630MQ CPU with 2.40 GHz and the second one is a desktop computer equipped with a i7-7700K CPU with 4.20 GHz. Our software runs on a single core and in a single thread.

Table 3.2 summarizes the runtime results for the different configurations presented above. We provide the average results obtained in the four TUM desk sequences for the Microsoft Kinect in RGBD and depth-only configurations, as well as for the two LiDAR setups. As listed in the table, our method can be executed fast and in an online fashion. On a mobile i7 CPU, we achieve average frame rates of 20 Hz with Microsoft Kinect RGBD sensor and 14 Hz with the Velodyne HDL-64E data, while we achieve average of 30 Hz and 23 Hz on an i7 desktop computer for the same configurations. Furthermore note, that our approach has a small memory footprint. For the whole registration procedure of the three datasets, we always required less than 200 MB (Microsoft Kinect: 178 MB; RobotEye: 140 MB; HDL-64E: 198 MB).

Note that recording a single RobotEye point clouds takes around 30 s, i.e. the robot stops, records, and then moves on. Thus, this setup may not be considered for real-time usage. In contrast, the Velodyne clouds of the KITTI dataset have been recorded at an average frame rate of 10 Hz and the data can be processed online.

In summary, our evaluation suggests that our method provides competitive

results in several different scenarios compared to approaches dedicated to a specific setup. At the same time, our method is fast enough for online processing and has small memory demands, proportional to the number of channels in use. Thus, we conclude that we supported all our four claims made in the introduction.

### 3.1.4 Related work

There exist a large number of different registration approaches. One general way for aligning 3D point clouds is the ICP algorithm, which has often been used in the context of range data. Popular approaches use ICP together with point-to-point or point-to-plane correspondences [62] and generalized variants such as GICP [107]. There exist approaches exploiting normal information such as NICP [110] as well as global approaches [141] that use branch-and-bound technique coupled with standard ICP formulation. A popular and effective approach is LOAM [142, 143] by Zhang and Singh that extracts distinct surface and corner features from the point cloud and determines plane-to-point and line-to-point distances to a voxel-grid representation.

Traditional approaches to visual odometry track sparse features in monocular images or stereo pair to estimate the relative orientation of the images [37, 89]. To deal with outliers in the data association between feature points, most approaches use RANSAC to identify inlier and outlier points, combined with tracking over multiple frames. Other approaches rely on a prior for the motion estimate, such as constant motion model. In presence of external sensors, such as an IMU, the measurements can be filtered with the achieved motion estimate [9, 144].

Another group of approaches exploits the depth data from RGBD streams to register scans and build dense models of the scene. KinectFusion by Newcombe *et al.* [86], for example, largely impacted the RGBD SLAM developments over the last 6 years. Similar to Newcombe *et al.*, the approach of Keller *et al.* [64] uses projective data association for RGBD SLAM in a dense model and relies on a surfel-based map [133] for tracking. Similar approaches exploit the RGBD streams by defining signed distance fields where a direct voxel-based difference is computed to perform the motion estimation [115], making intense use of both CPU and GPU parallelization. ICP is a frequently used approach for RGBD data and special variants for denser depth images have been proposed [109, 110]. Recently, the team around Daniel Cremers has proposed semi-dense approaches using image data [35] to solve the visual odometry and SLAM problem as well as dense approaches for featureless visual odometry for RGBD data [65, 66, 67].

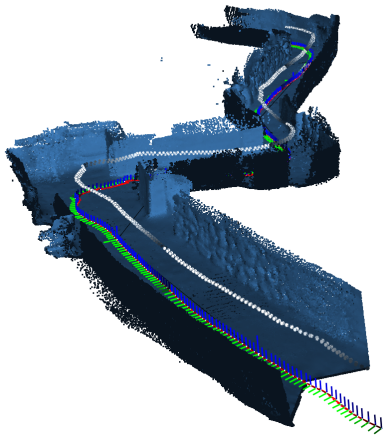

Figure 3.7: Part of the environment constructed by incremental point cloud matching. This environment represents the environment in which the robot must operate. It showcases narrow corridors, turns and uneven terrain. The poses near the ground show the poses of the robot frame while navigating. The black and write pyramids denote the positions of the cameras on the robot when taking each frame.

### 3.1.5 Conclusion

In this section, we presented a general framework for registering sensor data such as RGBD or 3D LiDAR data. Our approach extends dense visual odometry and operates on different available cues such as color, depth, and normal information. Our method avoids explicit data association and operates by direct error minimization using projections of the sensor data or model. This allows us to successfully register data effectively without tricky, sensor-specific adaptations. We implemented and evaluated our approach on different datasets and provided comparisons to other existing techniques and supported all claims made in this section. The experiments suggest that we can accurately register RGBD and 3D data under realistic configurations and that the computations can be executed at the sensor frame rate on a regular notebook computer using a single core.

## 3.2 Constructing a map of environment

By using the method outlined in the previous section, we use incremental matching to produce locally consistent parts of the environment. An example of such an incremental local map can be seen in Figure 3.7. The coordinate frame triplets show the poses of the robot where the clouds were taken, the black and white pyramids depict the positions of the cameras that acquired the data.

As briefly discussed in the introduction to this chapter, we use our multi-cue photometric point cloud registration method to generate the edges of the pose graph that we use to minimize accumulated error by solving the constraints in

this graph in the least squares sense. We use graph-based SLAM to create a full map of the environment that the robot navigates in. While SLAM system is not the main contribution of this thesis, we are using a classical graph-based SLAM system and optimize the pose graph in a least squares sense with the methods outlined in Section 2.1.

The pose graph that we use to construct our map has three types of edges: odometry edges, incremental matching ones and, in addition to these, we can use any loop closing technique available to generate the loop closing edges in order to correct any accumulated incremented matching and odometry errors. We then optimize the produced graph in the least squares sense to produce a consistent map, part of which can be seen in Figure 3.8.

We use this map throughout this thesis. In addition to a simple metric map we also add additional information such as traversability to it. We analyze the traversability of every scene observed by the robot and maintain this information as an additional 2D map constructed from the full 3D map shown in Figure 3.8. We use this 2D map representation for robot navigation. Having a consistent map with traversability information is crucial for navigation of the robot. We construct and use a 2D projection of the map in the exploration phase and actively analyze its quality in order to be sure that the path planned on such a map can be carried out. We also present the method to return the robot to the starting position should the SLAM system fail to construct a consistent map. The next chapter focuses on all these contributions extensively.

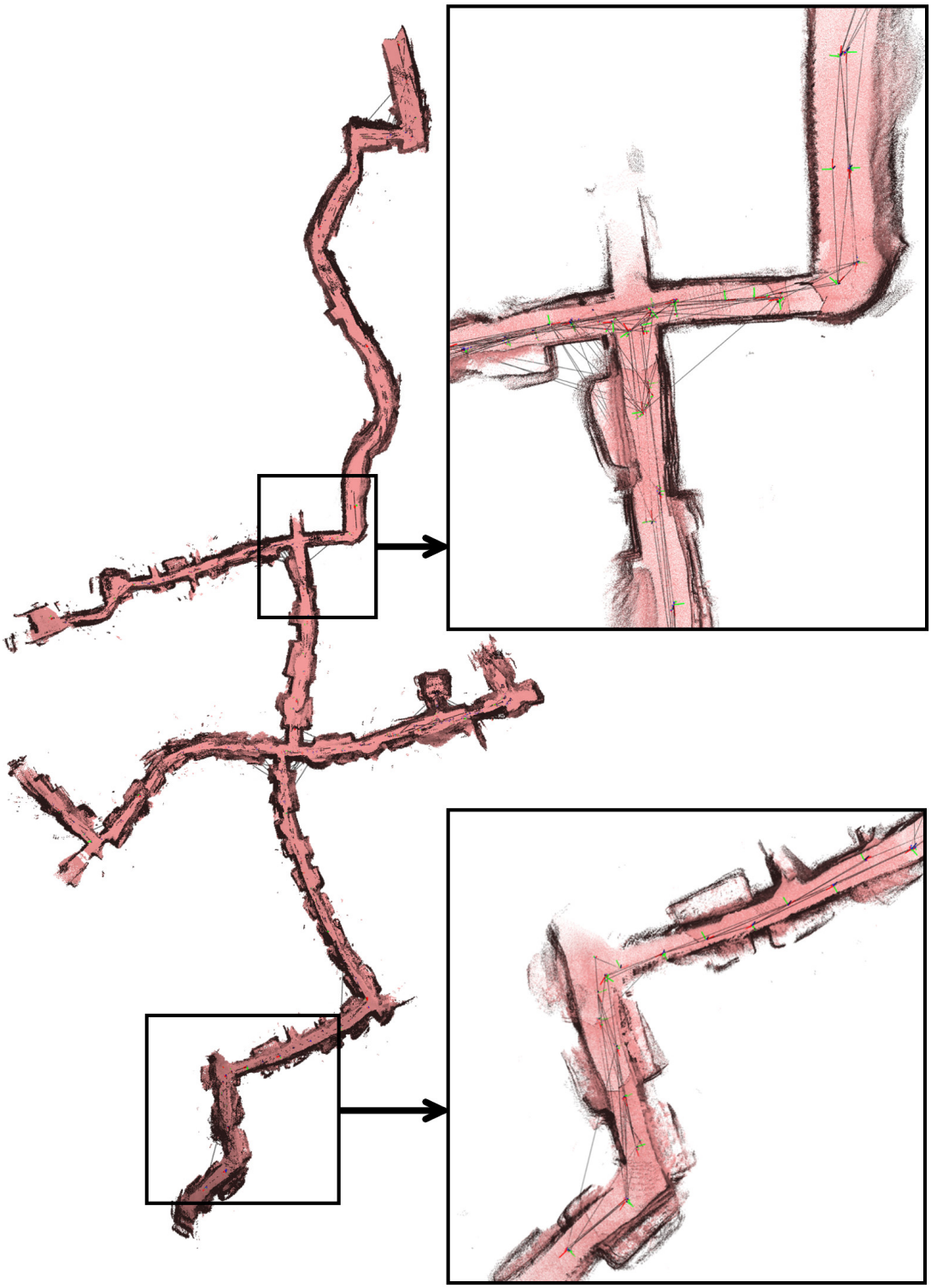

Figure 3.8: A map of Roman catacombs viewed from above reconstructed by performing incremental mapping and optimizing the full SLAM pose graph. The poses are shown with coordinate triplets and the lines connecting them represent the nodes on the graph.

# Chapter 4

# Navigation in static environments

While the robot mapping capabilities are, without doubt, a cornerstone of robot navigation in an unknown static environment, the robot still must make decisions on where it is safe to move and how to move there. The pipeline for moving in an unknown environment can be briefly summarized into two distinct tasks: first, the robot must analyze the surroundings, and, second, it must plan and execute a movement. In this section we cover the components that the robot requires to safely navigate the environment, namely (i) traversability analysis of the scene currently perceived by the robot, (ii) navigation and exploration capabilities of the robot. In addition to that, we (iii) provide a system that is able to detect when the map is not reliable anymore and return the robot to the base safely not relying on any global map.

Our first contribution in this chapter is the traversability analysis, presented in Section 4.1. The robot navigating in an unknown environment must analyze every scene that it captures. Our method relies on 3D data from the current scene and analyzes if the environment is traversable given the geometric constraints of the robot. We have tested this approach in the real Roman catacombs as part of ROVINA project using the data coming from the Asus Xtion (Figure 2.1) depth sensors. Even though this method was tested mostly with RGBD cameras it can be extended to work with any other sensor that provides 3D information and we have additionally tested it with the Ocular 3D LiDAR.

The second contribution regards the navigation and exploration stacks of the robot and is presented in detail in Section 4.2. An autonomous robot must explore the environment in order to construct its map. During the exploration stage it relies on robust navigation capabilities in order to navigate to a specific place given only a part of the map and the positions of itself and the goal in that map. The robot must plan the best path to the goal taking traversability of the environment into account, and carry it out. We implemented two distinct strategies for the

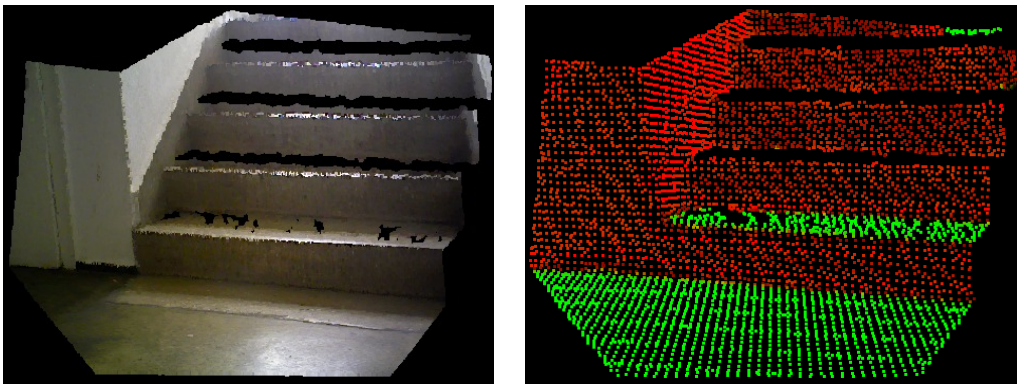

Figure 4.1: Stair case observed with a Microsoft Kinect and a corresponding labeling into traversable (green) and non-traversable (red) areas.

robot to explore the environment: a frontier exploration technique, as well as a more sophisticated approach that relies on background knowledge, and have integrated them into our system [96].

Despite using such a robust exploration approach that actively searches for loop closures and improves robots localizability there is no method that *guarantees* map consistency during robot navigation. To address this issue, we present an algorithm to continuously monitor the quality of the map and return the robot safely to the base in case the map cannot be used reliably anymore. This contribution is presented in detail in Section 4.3. In this work, we build upon the work of Mazuran *et al.* [80]. Their static map consistency analysis tool have been proven useful to analyze the maps generated by a high-precision 2D LiDAR. We extend that approach to work with our pipeline while using much cheaper 3D sensors as well as to run online continuously on the robot hardware. In addition to extending their approach for map consistency checking, we implemented an algorithm to "unwind" the trajectory taken by the robot to return the robot to the base without relying on any kind of consistent global map.

## 4.1 Traversability analysis

Autonomous outdoor navigation is an active research field in robotics. In most scenarios, the classification of terrain into traversable and non-traversable areas plays an important role. Failing to stay on roads or other traversable surfaces can introduce wheel slippage, which in turn leads to errors in the odometry. It can lead to accidents, risk of getting stuck, or even of destroying the platform. Therefore, the ability to robustly detect traversable areas is important for safe navigation, especially in fully autonomous settings.

The main contribution of the approach presented in this section is an accurate, fast to compute, and comparably easy to implement traversability analysis

approach for mobile robots. Our system operates on the depth images from a Microsoft Kinect or an Asus Xtion camera and analyzes the visible area in front of the robot at 10 fps-25 fps on a notebook computer without using the graphics processing unit (GPU). Not relying on GPUs has the advantage of requiring less energy, which is a relevant issue for small-scale autonomous robots. Our approach has been implemented and evaluated in several sites including a real catacomb. An example of a depth image labeled with the traversability information is shown in Figure 4.1.

### 4.1.1 Our approach to traversability analysis

The main objective of our work is the development of an accurate and fast-to-compute traversability analysis system for mobile robots. A central focus lies on the online capabilities of the system on a standard notebook computer without requiring a GPU, so that the traversability analysis can be computed and integrated into the model on the fly. Our approach considers only the depth image. Our motivation is that the underground environments are completely dark (if one does not carry own light source) and RGB data is often useless.

Our approach can be split up into two main steps. After a preprocessing step, our system first estimates the local traversability based on a single depth image. This step takes into account the navigation capabilities of the vehicle. Second, the integration of the single-image traversability estimates into a local traversability map.

Note that we assume the depth images to be locally registered, i.e., the traversability analysis does not account for any pose uncertainty of the vehicle. We can use our incremental ICP-based matching approach from Section 3.1 or any other that uses the point clouds and normal vectors, and can be executed on the fly. Our approach can also be used with global methods such as graph-based SLAM to obtain a globally consistent model. This is straight forward if the local traversability estimates are stored in the nodes of the SLAM graph. After the optimization, the global traversability map can be rendered from the local views in the global frame.

#### 4.1.1.1 What is traversable for a robot

Given a typical robotic platform such as a Pioneer 2AT or a Mesa Element platform, the traversability is mainly governed by three factors. First, a maximum step height limits the vehicle from climbing steps higher than 5 cm-15 cm (depending on the wheels/tracks and the exact setup). Second, the maximum slope the robot can climb or descend. The exact figures depend on the weight distribution of the platform and its sensors. In our case, the maximum slope was 15 degrees.

The third factor, that limits the traversability is the height of the platform, which may prevent the robot from driving into low niches or similar places. Additional factors may impact the traversability of a platform such as mud or water — such surfaces are, however, hard or even impossible to be identified with a depth sensor such as the Microsoft Kinect and are therefore not considered here.

Our approach only considers the environment and not explicitly the shape of the platform such as its width and height. The question if the robot physically fits into an area that is labeled as traversable has to be answered by the planner itself as the state of the robot, especially its orientation, influences that. Therefore, also currently unreachable places can be safely classified as traversable as the planner will not expand these states.

Based on these constraints, the tasks of estimating the traversability consists of analyzing the environment covered by the sensor and identifying height constraints, steps, and slopes. Before providing further details on that, we briefly introduce the sparse data structure that we use to store the 3D data. Such a structure is key for fast online processing.

#### 4.1.1.2 Sparse 3D map

We store the measured endpoints and normals, computed in the way described in Section 2.2.3 in a 3D data structure. In theory, a raw point cloud could be used but they have the disadvantage that finding neighboring points is expensive. As finding neighbors is explicitly required later on, we propose to approximate the points using a grid-like structure. As our data is sparse compared to the number of cells of a full 3D grid, we store the data in a sparse grid that is realized via a two-layered hash-table like structure.

First, a hash-table is used to index points in the x-y plane in world coordinates. The key of the hash-table is the discretized $(x, y)$ coordinate of points that can, given the range and resolution of the Microsoft Kinect sensor, be modeled by a single 32 bit integer. This hash table acts as a sparse 2D grid as only those cells are instantiated for which 3D points with a corresponding $(x, y)$ coordinate exist. For every non-empty $(x, y)$ grid cell, we initiate a red-black tree, i.e., a self-balancing binary search tree that allows for quickly accessing elements and for processing them in a sorted order. The keys of each tree are the discretized $z$ coordinates of the endpoints. From an implementation point of view, this may sound complex but note that the C++ standard template library implements a red-black tree within `std::map` and thus can directly be used without additional efforts. The same holds for the hash table via `std::unordered_map`.

The overall data structure models a sparse 3D grid that allows us to store discretized $(x, y, z)$ triplets and, thanks to the red-black tree, allows for parsing the $z$ coordinate efficiently in an ordered way. This is used to compute steps for

$(x, y)$ cells and to estimate if the height constraints of the platform are violated.

In sum, the points of the point cloud that is computed from the depth image are added to the sparse 3D grid. For each 3D cell, we compute the average 3D coordinate and normal on the fly. In our current implementation, we use a discretization of 4 cm. As a result of that, we obtain a (deterministically) sub-sampled point cloud that is stored in a sparse 3D grid. The following two operations can be therefore conducted efficiently: accessing any cell including neighboring cells and iterating over the sorted $z$-coordinates of all cells for a given $(x, y)$ coordinate.

#### 4.1.1.3 Accounting for the vehicle height

The first criterion that impacts the traversability of the terrain is the height constraint of the vehicle. For every $(x, y)$ location in our sparse 3D map $M$, we start from the lowest measured $z$ value and compute the free space in $z$ direction to the next obstacle. Any obstacle for which the difference in the $z$ coordinate to the previous obstacle is larger than the height of the platform can be discarded as it does not constrain the motion of the vehicle. All obstacles for which this distance is smaller than the height of the platform are maintained in $M$ and will, in the subsequent steps, be analyzed according to their step height and slope.

#### 4.1.1.4 Efficient step detection

Based on the sparse 3D map $M$, we can efficiently query the neighboring points for any 3D coordinate. By analyzing the height differences between points stored in neighboring cells, we can quickly check for large steps that the robot cannot traverse. The neighbors of a point $\mathbf{p}$ up to a distance of $d$ can be written as

$$\mathcal{N}(\mathbf{p}, d) \;=\; \{\mathbf{q} \mid \mathbf{q} \in M \wedge ||\mathbf{p} - \mathbf{q}|| \leq d\}. \tag{4.1}$$

For each instantiated grid cell $\mathbf{p} \in M$, we inspect the $z$ coordinates of the neighbors

$$\mathbf{q}_z \;=\; \{z \mid [x\, y\, z]^\top \in \mathcal{N}(\mathbf{p}, d)\}, \tag{4.2}$$

and test if the coordinate is larger then $z_{\max}$, which is the maximum step that the vehicle can climb or descend. The decision about traversability is then done by the following function

$$\tau_{\text{step}}^{(\mathbf{p})} \;=\; \begin{cases} 1 & \exists\, z \in \mathbf{q}_z \,:\, ||\mathbf{p}_z - z|| \leq z_{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

The expression is equal to 1 if there is any non-traversable step at $\mathbf{p}$ and to 0 otherwise. In our implementation, the neighbor distance $d$ was set to 10 cm and the maximum step height $z_{\max}$ that the robot can traverse was 10 cm as well.

#### 4.1.1.5 Robust slope detection

Besides steps, there is a second criterion that is important for deciding if terrain is traversable or not: up to which degree can the robot climb slopes?

For computing the slope of a local area, we consider local normal vectors which are efficiently computed as explained in Section 2.2.3. Based on the normal vector $\mathbf{n}$ and the gravity vector $\mathbf{n}_g$, a straight forward test allows us to estimate the traversability of a perceived surface based on the slope as

$$\mathbf{n}_g \cdot \mathbf{n} \;\leq\; \cos(\alpha_{\max}), \tag{4.4}$$

where $\cdot$ is the scalar product and $\alpha_{\max}$ the maximum slope the robot can handle. In our setup, the gravity vector $\mathbf{n}_g$ is obtained from a standard IMU. We use an XSens MTi, which provides the gravity vector up to an error of approximately 0.5 degrees.

The test in Equation (4.4) allows us to efficiently detect normal vectors that yield a steeper slope that exceeds the navigation capabilities of the robot. However, testing only for the slope is not enough. Consider a small step that the robot can traverse. The vertical surface of the step creates normal vectors that are orthogonal to the gravity vector and thus report a steep slope that cannot be traversed. Mathematically, that surface is correctly labeled as 90 degree slope, but it should not affect the traversability labeling as long as the step is small enough to be traversed by the robot. Thus, we are only interested in slopes that have a minimum extension in the x-y plane in order to be classified as slopes, which cannot be traversed.

To achieve this, we apply a rather standard erosion-dilation filter [16] with a 5-cross structuring element to our traversability map. Let $\tau^{(x,y)}_{\text{slope}}$ be the traversability label for the position $(x, y)$ where 0 refers to traversable and 1 to non-traversable. The erosion step updates the estimate

$$\tau^{(x,y)}_{\text{slope}} \;\leftarrow\; \max\left(\tau^{(x,y)}_{\text{slope}},\ \tau^{(x-d,y)}_{\text{slope}},\ \tau^{(x+d,y)}_{\text{slope}},\ \tau^{(x,y-d)}_{\text{slope}},\ \tau^{(x,y+d)}_{\text{slope}}\right), \tag{4.5}$$

and is followed by the dilation step

$$\tau^{(x,y)}_{\text{slope}} \;\leftarrow\; \min\left(\tau^{(x,y)}_{\text{slope}},\ \tau^{(x-d,y)}_{\text{slope}},\ \tau^{(x+d,y)}_{\text{slope}},\ \tau^{(x,y-d)}_{\text{slope}},\ \tau^{(x,y+d)}_{\text{slope}}\right). \tag{4.6}$$

Here, the scalar $d$ describes the distance in which the neighbor considered (as in the previous subsection on step detection). As a result of the erosion-dilation filtering, small slopes, such as steps, are filtered out while "real slopes", i.e., larger areas with a steep inclination angle, are maintained. This process is illustrated in Figure 4.2.

Finally, the traversability $\tau^{(x,y)}$ is obtained by combining the traversability extracted from slopes $\tau^{(x,y)}_{\text{slopes}}$ and steps and height constraints $\tau^{(x,y)}_{\text{steps}}$ by

$$\tau^{(x,y)} \;=\; \max\left(\tau^{(x,y)}_{\text{slope}},\ \tau^{(x,y)}_{\text{step}}\right). \tag{4.7}$$

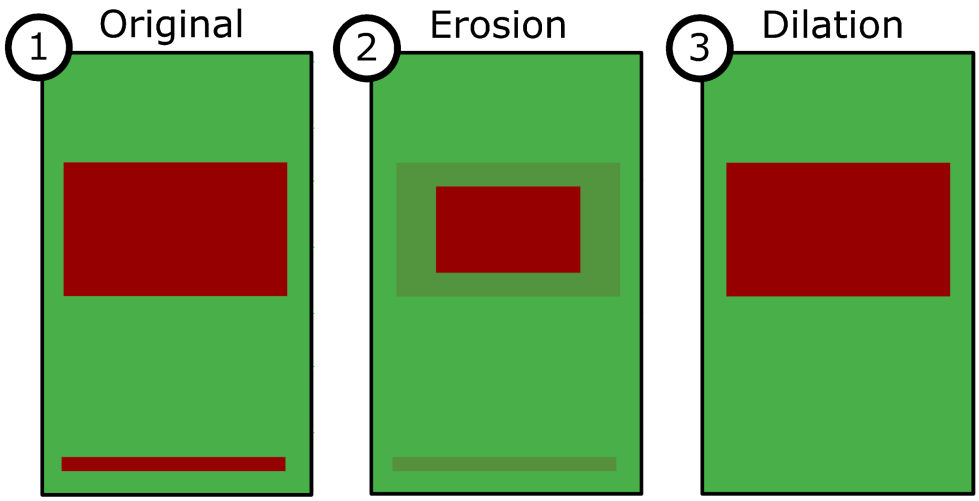


Figure 4.2: *Left:* traversability estimate with both, a non-traversable slope (top red rectangle) a traversable step (bottom red stripe) marked as non-traversable slopes. *Middle:* erosion shrinks all non-traversable areas by the neighborhood distance $d$. *Right:* dilation restores a real non-traversable slope, while leaving a step as traversable, thus, avoiding labeling a step as a small non-traversable slope.

In sum, this approach provides a traversability estimate given a single depth image.

#### 4.1.1.6 Traversability map estimation

Let $\tau_t$ be such a traversability estimate of a local area obtained from a single RGBD image taken at time $t$. As this estimate is not free of errors, we integrate multiple of such measurements into one model. We achieve that by employing a static state binary Bayes filter that integrates the information for every non-empty cell $i$ in $M$.

Following the work of Moravec [84], summarized in Section 2.3, we can compute a recursive update formula for $p(\mathfrak{T}_i \mid \tau_{1:t})$ as

$$p(\mathfrak{T}_i \mid \tau_{1:t}) = \left[1 + \frac{1 - p(\mathfrak{T}_i \mid \tau_t)}{p(\mathfrak{T}_i \mid \tau_t)} \frac{1 - p(\mathfrak{T}_i \mid \tau_{1:t-1})}{p(\mathfrak{T}_i \mid \tau_{1:t-1})} \frac{p(\mathfrak{T}_i)}{1 - p(\mathfrak{T}_i)}\right]^{-1}. \tag{4.8}$$

In order to gain efficiency, one can furthermore use the log-odds formulation, so that the operations in Equation (4.8) are realized via addition and subtractions in the log-odds space.

To apply the Bayes filter, we need to specify the inverse observation model $p(\mathfrak{T}_i \mid \tau_t)$. As the depth resolution decreases quadratically with increasing distance from the sensor, we use the inverse sensor model (assuming a prior of $p(\mathfrak{T}_i) = 0.5$)

$$p(\mathfrak{T}_i \mid \tau_t) = 0.5 + \frac{2\,\mathfrak{T}_i - 1}{2 + l^2}, \tag{4.9}$$

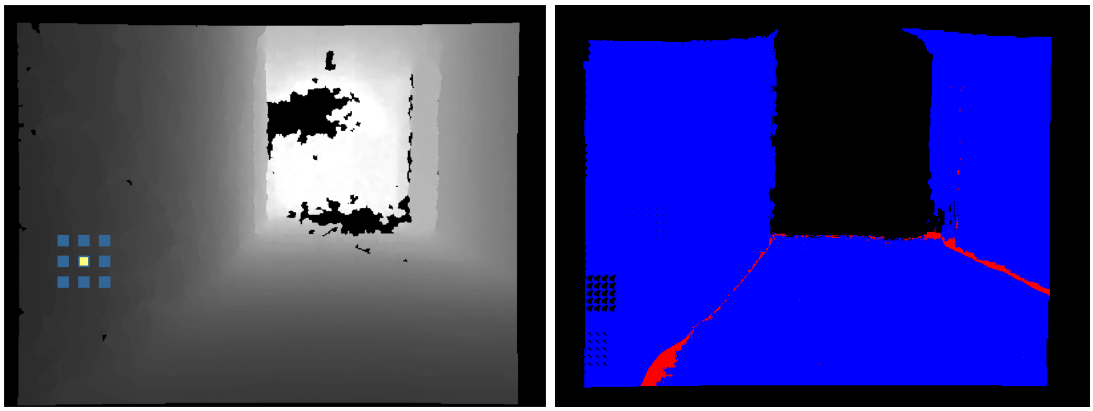

Figure 4.3: *Left:* An example of the pattern-based descriptor overlaid onto a depth image taken with an Microsoft Kinect RGBD sensor. Yellow shows the query pixel in which we compute the descriptor while the pixels outlined in blue show its neighbors. We use these descriptors to predict traversability information directly from depth as an alternative to the approach presented in previous sections. *Right:* An evaluation of traversability prediction accuracy. Blue shows values where the predicted traversability is matching the reference traversability information, while the red ones represent the pixels where the prediction is wrong. Black pixels denote parts of the image with no prediction. Note, that we cut off the depth values at 4 meters for traversability estimation here as the readings become too imprecise beyond this distance.

where $l$ is the distance between the camera and the measured cell (0.7 m-4 m) and $\mathfrak{T}_i \in \{0, 1\}$. With this filter, we integrate the individual traversability estimates that are computed per depth image into one consistent model.

### 4.1.2 Learning traversability directly

Computing the normals over images, merging them into a hash map and performing computation over the whole grid to decide if the scene before the robot is traversable, requires many operations to be performed. These operations all take CPU time and, therefore, energy resources that can be limited on a mobile robot platform. While using the robot in the real-world Roman catacombs we have acquired datasets where the range images could be mapped to the computed traversability information. We can project the traversability information from the 3D representation into the range image domain. This generates data that consists of 2D images with traversability information stored in every pixel, which can be treated as a mapping from depth data to traversability information. If we use this data as training data, we can learn these mappings, which will henceforth allow us to compute the traversability information directly from a range image. Learning this information only makes sense if the system that we use gives a performance boost over the standard pipeline presented in the previous sections. We therefore explore the use of the linear SVMs to map a simple descriptor over the range image pixels in a rigid pattern. This results in a

method that can predict the traversability of the environment directly from the depth values captured from the sensor with 98% precision.

In our implementation, we experimented with relatively simple descriptor patterns. Due to our runtime constraints, we constrained our implementation to use the descriptors consisting of a value of a center pixel as well as its $N$ neighbors spaced at a rigid pattern. We considered neighborhoods of sizes: 3×3, 5×5 and 9×9. An example 3×3 pattern, shown in blue, overlaid with a depth image from the catacombs, can be seen on the left of Figure 4.3. We experimented with all the neighborhood sized presented above, and concluded that the neighborhood of 5×5 yields the best trade-off between the runtime efficiency and accuracy. The smaller pattern generates more errors, while using the bigger pattern makes the prediction substantially slower than the reference method.

The right side of Figure 4.3 shows an example image that shows a comparison between the predicted learned traversability and the actual traversability. The blue pixels represent the values where the predicted and the actual traversability values match, while the red pixels show the prediction errors. Black pixels are ignored and have no traversability value predicted in them. We observe that mainly the prediction errors happen in the areas that are expected to be hard to predict, i.e., on the border of the walls and floor or on steps in depth.

Overall we achieved good performance, but did not achieve a substantial runtime improvement. We think that the minor gain in runtime does not justify the loss in accuracy in this case, especially considering that the reference algorithm is capable of running at frame rate of the sensor even on computation-constrained hardware. Therefore, in the remainder of this section we will focus on the experiments carried out using the standard pipeline, and not using the SVM-based approach.

### 4.1.3 Experiments

The experimental evaluation is designed to show the capabilities of our traversability estimation system in different environments. Throughout all our experiments, we used either a Microsoft Kinect (catacomb scenes) or an Asus Xtion (office and outdoor scenes) installed on a mobile robot. We only used the depth information for the traversability estimate, the RGB information is used solely for visualizations.

#### 4.1.3.1 Timing experiments

The first experiment is designed to illustrate that our method runs online on a notebook computer without GPU usage and can process the incoming depth images at high frame rate. We tested our system on two standard notebooks,

| | 640×480 pixels | | | 320×240 pixels | | |
|---|---|---|---|---|---|---|
| CPU | normals | trav. | fps | normals | trav. | fps |
| i7 | 38 ms | 55 ms | 10.7 | 18 ms | 22 ms | 25 |
| i5 | 85 ms | 70 ms | 6.6 | 40 ms | 25 ms | 15.3 |

Table 4.1: Average timings of the normal computation and traversability analysis and the overall rate for two different resolutions of the depth image.

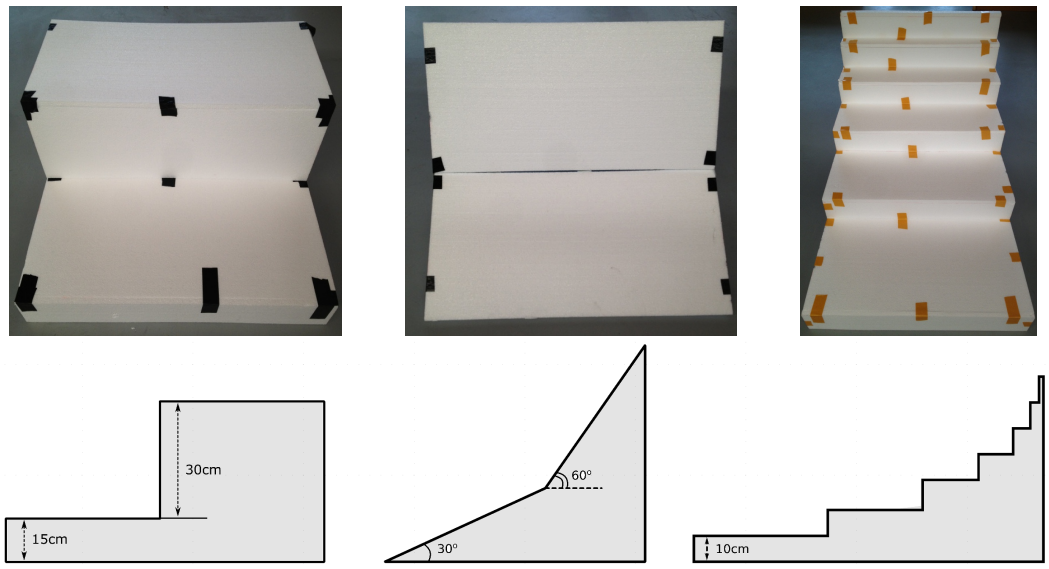


Figure 4.4: Photos and sketches of the individual test objects. *Left:* a shape that is formed by two steps of different size. The step closer to us is a small step, while the one further from us is significantly bigger. *Middle:* a shape that consists of two slopes connected to each other in the middle. The slope closer to us is shallow, while the slope further away from the camera is much steeper. *Right:* a shape that consists of multiple steps that become steeper with distance from the camera.

one equipped with a 2.3 GHz Intel i7 processor and one with an Intel i5-2410M processor. Table 4.1 illustrates the results. As can be seen, we achieve frame rates between 10 fps and 25 fps on an i7 notebook depending on the depth image resolution. Thus, the environment in front of the robot can be analyzed on the fly allowing for autonomous navigation and exploration. Even on the i5 processor, the data can be analyzed at a frame rate of 15 fps on 320×240 pixel depth images.

#### 4.1.3.2 Ground truth comparison

The next experiment is designed to analyze the error of our traversability estimate. It is non-trivial to provide a ground truth analysis outside a simulator, but we put our best efforts to achieve a near ground truth evaluation by observing custom-made structures with known 3D geometry, shown in Figure 4.4 and

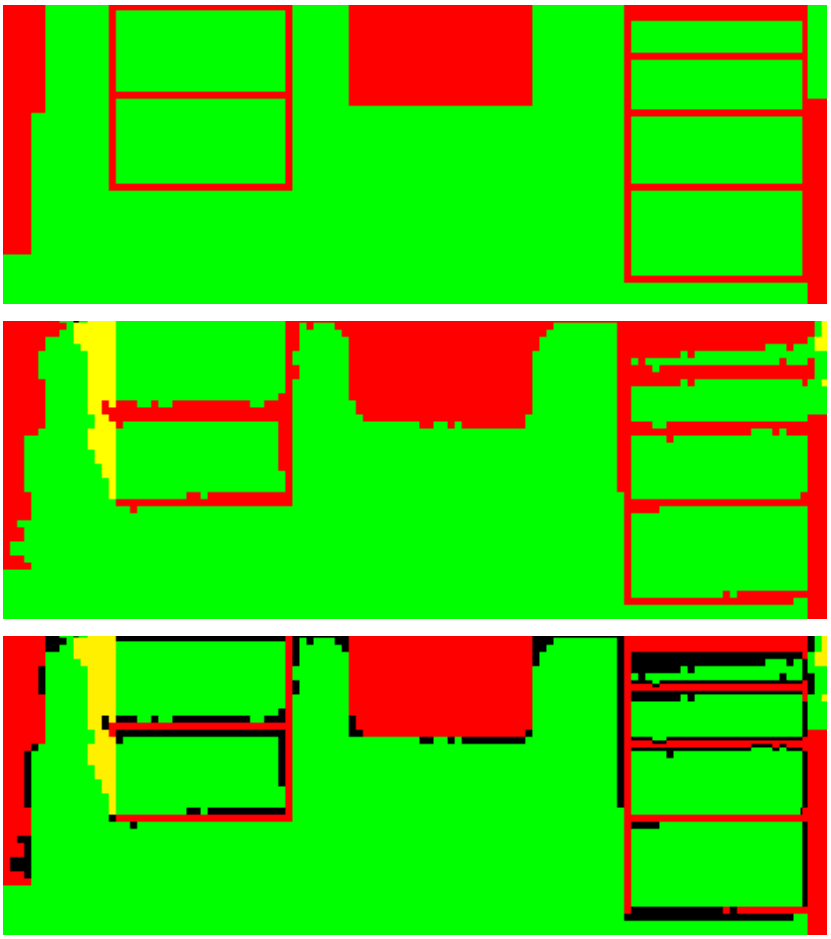

Figure 4.5: Ground truth comparison using the objects pictured in Figure 4.4. *Top:* ground truth labeling. Both steps on the left, steeper slope and final steps of the last shape are all non-traversable given the constraints of the robot here. *Middle:* our approach. *Bottom:* overlaid images (truth above estimate). Green cells are labeled as traversable, red refers to non-traversable and yellow to cells for which not enough observations have been made to allow for a confident labeling. Wrongly labeled pixels are marked black. Note that most error are due to discretization effects.

compare the traversability estimates pixel by pixel with the geometric model. Figure 4.5 illustrates the objects and the traversability estimates. The right image shows the overlay of the ground truth (left) and estimated (middle) maps. All errors based on a pixel-by-pixel comparison are highlighted in black. In this experiment, 7.2% of the cells are wrongly labeled if considered independently. However, nearly all wrongly labeled cells occur at the borders of the obstacles and are between one and two cells sizes away from the real obstacle. Most of these errors actually result from discretization errors or slight smoothing effects at steps when computing the normal. In addition to that, 1.9% of the cells were not observed sufficiently to allow for an appropriate labeling. This occurs if $p(\mathfrak{T}_i \mid \tau_{1:t})$ has a value close to the prior (here 0.5). These cells are colored yellow.

This experiment shows that our approach provides an accurate labeling when

Figure 4.6: *Left:* robot driving in the Priscilla catacombs. *Middle:* non-traversable staircase. *Right:* non-traversable heap of rubble.

ignoring the discretization errors. This can typically be done on most real navigation settings and the system classifies the input data well for our application—exploring an unknown catacomb with a mobile platform.

#### 4.1.3.3 Traversability estimates obtained in different scenes

We obtained the traversability information computed with our system in real-world scenes. First, this includes the deployment of a prototype robot based on a Pioneer2 AT system in the Catacombe di Priscilla in the underground of Rome, see Figure 4.6. The robot was steered through the environment, incrementally aligning the depth images from a Microsoft Kinect RGBD camera and building the traversability map. Figure 4.7 illustrates a fraction of the traversability map showing the traversable and non-traversable areas. This traversability

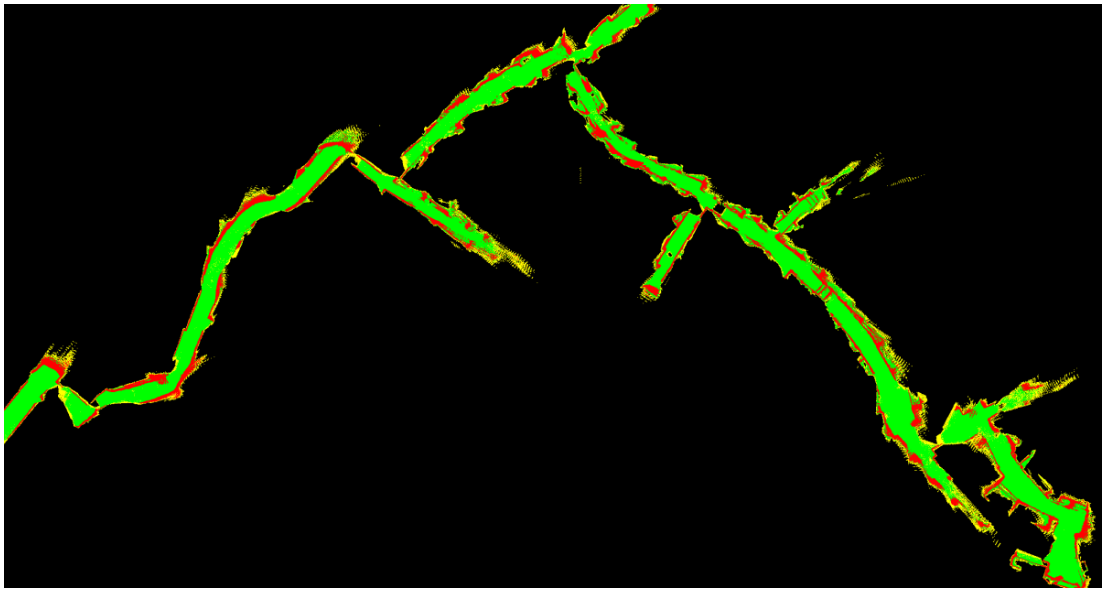

Figure 4.7: Fraction of the explored space of the Catacombe di Priscilla. This is a 2D traversability map that resembles the same environment as shown in Figure 3.8.

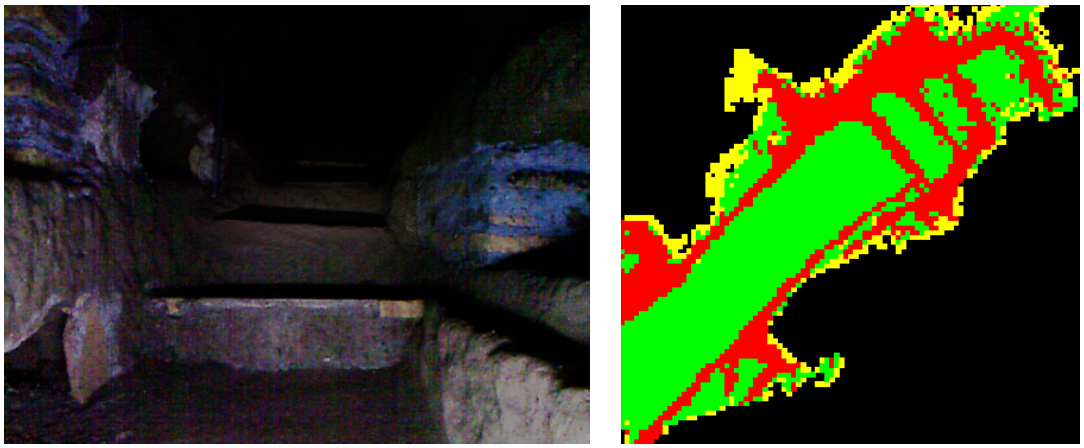

Figure 4.8: A staircase experienced during the mapping of a catacomb site that is not traversable for the Pioneer robot. *Left:* RGB image from the Microsoft Kinect. The image is dark as the on-board light was not powerful enough to appropriately illuminate the scene. *Right:* traversability estimate.

Figure 4.9: A situation in which the ground level is flat and traversable but the height of the platform prevents the robot from entering the niche. *Left:* RGB image from the Microsoft Kinect. *Right:* traversability estimate.

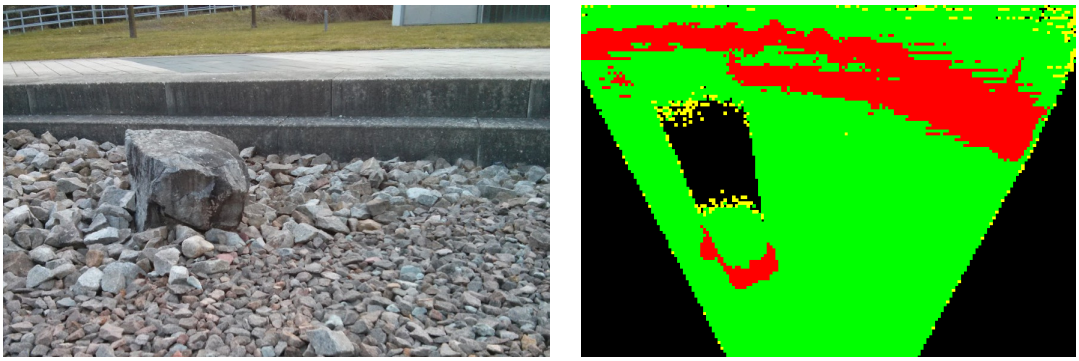

Figure 4.10: Example of an outdoor scene on campus observed by a rough terrain robot. The left image shows a photo and the right one shows the local traversability map (distorted). The black region corresponds the area behind the obstacle that was not visible.

map is generated from the same environment as shown in Figure 3.8. Figure 4.8 and 4.9 illustrate two selected places showing the RGB image from the Microsoft Kinect and the local traversability map. Second, Figure 4.10 shows the results obtained on campus outdoors on a cloudy day. As can be seen, the small rocks are traversable for the outdoor platform but not the big rock and two steps. Finally, the motivating example in Figure 4.1 illustrates a labeled depth image (without the integration into a traversability map).

#### 4.1.3.4 Limitations

We also experienced limitations of our system. There are situations in which parts of the environment are traversable only if the robot steers in a specific way, otherwise not. An example are track-like obstacles that are too tall to be traversed from a side but are at the same time low enough to fit between the wheels of the robot. Another example is a v-shaped valley in which the slopes are traversable except a corner-shaped bottom, which is not non-traversable (it is depended on the position of the wheels on the platform).

### 4.1.4 Related work

Estimating traversable areas is essential for most navigation tasks and thus has been investigated intensively in the past. For example, Rasmussen [101] proposes an approach to trail following for mobile cross country robots. The robot investigates the local variance of depth measurements, structural texture, and contrast to identify and follow a trail. The work closest to our approach is probably the work of Renner *et al.* [103] that aims at estimating environment properties such as positive obstacles, flexibility, shape, dimensions, slope, etc. using a camera and a PMD depth sensor. Besides visual information such as texture, color, and variance in contrast, they also consider surface normals and steps to identify obstacles and create a polar, robot-centric model based on which they navigate. Similar to Renner *et al.*, De Cubber *et al.* [23] address outdoor terrain traversability using a PMD and a stereo camera. They estimate a ground plane and seek for pixels that have a high probability of belonging to the ground plane. Then, they use the color information of other pixels to classify all image pixels as traversable.

Other approaches perform a semantic scene analysis to support navigation [102, 121]. Ren *et al.* [102] propose an approach for indoor scenes using a Microsoft Kinect. They compute a combination of color and depth features using kernel descriptors and achieve a high labeling performance by combining Markov random fields with segmentation trees.

Katramados *et al.* [63] present a real-time approach for traversable surface detection using a monocular camera mounted on robot. Based on the currently

assumed to be traversable location, the system searches similar areas in the image given color and texture features. The approach of Maier *et al.* [77] combines a monocular camera with sparse laser range data on a humanoid to identify obstacles on the ground plane. They infer the traversability information based on the vision data after learning a classifier from sparse laser information.

Terrain types have also been classified using vibration sensors [18, 134]. Here, the vibration measurements are usually analyzed in the Fourier domain. Brooks and Iagenemma [18] use a combination of PCA and LDA to classify terrain and Weiss *et al.* [134] use SVMs. Vibration-based approaches typically offer highly accurate classification results. The drawback of such methods, however, is that only the terrain the robot is moving on can be classified and not the terrain in front of the robot.

There exist approaches that apply self-supervised learning to classify terrain and detect obstacles. A number of methods have been developed that exploit local terrain knowledge to predict surface terrain in the far range. These near-to-far approaches use color information [49, 53], 3D geometry information [79], or texture information [2]. Self-supervised learning using laser and vision data is also used by Dahlkamp *et al.* [25] in a vision-based road detection system. Finally, traversability analysis is also the motivation for several approaches that aim at detecting vegetation such as grass. They typically use laser remission values [137], laser range data [56, 136], and also combinations with vision [17, 30].

### 4.1.5 Conclusion

Traversability information is important for autonomous mobile robots. This section presents a system for estimating the local traversability for a mobile robot based on RGBD images online. Our approach can process the depth data at 10 fps-25 fps on a standard laptop computer with an Intel i7 processor without a GPU and allows for robustly identifying the areas in front of the sensor that are safe for navigation. The component presented here is one of the building blocks of the EU project ROVINA that aims at the exploration and digital preservation of hazardous archaeological sites with mobile robots. As we showed in our experimental evaluation, our approach is able to reliably estimate the traversability in different environments, ranging from the lab to outdoor scenes as well as in a real, partially unexplored, and nearly 1700 year old Roman catacomb in the underground of Rome.

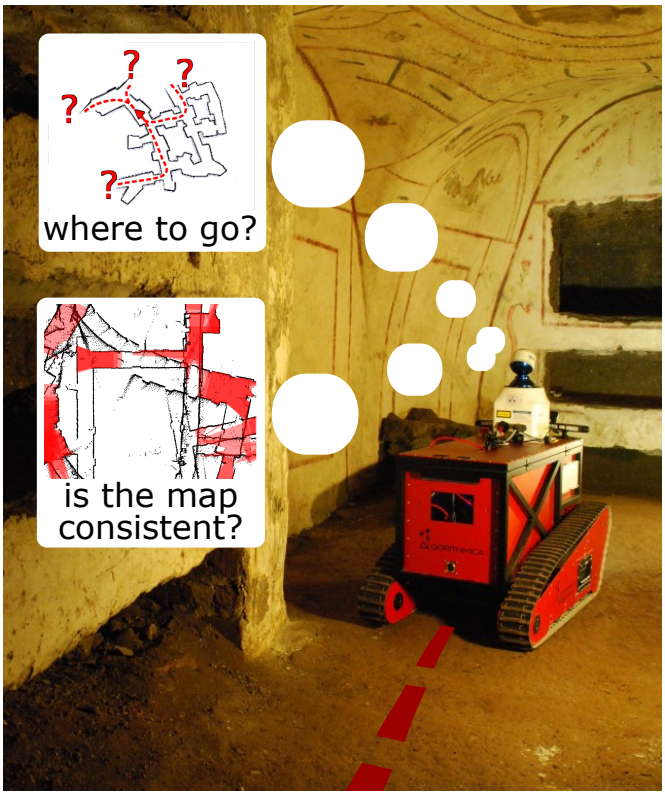


Figure 4.11: The robot must analyze the environment and pick an exploration strategy to cover it efficiently. If at any time of the exploration procedure the statistical map consistency tester provides the robot with the information that the map along the chosen path is inconsistent the robot starts rewinding the trajectory using our method.

## 4.2 Exploration

Exploration is the task of selecting view points so that a robot can cover the environment with its sensors to build a map. The ability to robustly operate without user intervention is an important capability for exploration robots, especially if there are no means for communication between the robot and an operator. Most robots exploring a new environment start assuming zero knowledge about this environment and do not exploit any background knowledge about the particular context they are in or about the typical environments in this domain. They build a map of the environment online and make all navigation decisions based on this map. As long as such a map is consistent, the robot can perform autonomous navigation by planning the shortest path, for example using A*, from its current location to its next vantage point using the map. Although recent SLAM systems are fairly robust, there is a chance that they fail, for example, due to wrong data associations generated by the front-end. Even current state-of-the-art SLAM approaches cannot *guarantee* the consistency of the resulting map. Computing a path based on an inconsistent map, however, is likely to lead to a failure, and possibly, to losing the robot if operating in a hazardous environment. Thus, ex-

ploring robots should always decide where to go next, and at the same time verify if their map is still consistent (see sketch in Figure 4.11). Considering existing approaches, however, it is fair to say that most exploration systems follow the paradigm that they (a) make their navigation and exploration decisions using the current map only and (b) assume that the map is consistent and thus can be used as the basis for path planning and navigation.

In this section we cover the typical frontier-based exploration method that we implemented to work on our robot platform in Section 4.2.1, as well as adapt and extend a method that utilizes the background knowledge about the environment in order to explore it with less uncertainty and to aid loop closing performance. This method is not the main contribution of this thesis, but it is a crucial building block for a robust homing system that we present, and tightly integrate with the exploration and navigation stacks of the robot, later in this section.

The key idea to our extension to the frontier-based exploration method, is to relax the assumption that the robot performs exploration tasks with no knowledge about the environment. Using the background knowledge about the environment allows the robot to consider the information gained from previously conducted exploration missions and use it to support the navigation system of the robot. This is motivated by the fact that selecting appropriate target locations during exploration supports the mapping process, and can increase the probability of building a consistent map. The key idea is to use previously experienced environments to reason about what to find in the unknown parts of the world. To achieve this, we equip our robot with a database to store all acquired (local) maps and exploit this knowledge when selecting target locations. Our research is motivated by an exploration project for autonomously digitizing the Roman catacombs, which are complex underground environments with repetitive structures. To predict possible geometries of the environment the robot may experience during exploration, we exploit previously visited areas and consider the similarities with the area around the currently planned next view point. This makes sense for environments with repetitive structures such as catacombs but also holds for various indoor environments. Exploiting such structures allows the robot to actively seek for loop-closures and in this way actively reduce its pose uncertainty. Our experiments indicate that this approach is beneficiary for robots when comparing it to a standard frontier-based exploration method.

### 4.2.1 Frontier-based exploration

As a baseline, we have implemented a classic frontier-based exploration technique that loosely follows the approach of Yamauchi [140] with the difference that it is based on the traversability maps shown in Figure 4.7. Yamauchi introduces the concept of frontiers, which are the cells of an occupancy grid map at the

boundary between the free and the unexplored space. In the standard setting, this approach seeks to minimize the time that is needed to cover the environment with the robot's sensors and is a popular choice in mobile robotics. We analyze the frontiers of explored map by finding all those cells of the traversability map that have both traversable and unobserved neighbors. We pick the closest of these candidates and plan a path towards it with A*. After carrying out this path we repeat the process. Even though this method follows greedy logic it still works well in practice.

A small example of a robot exploring catacomb-like environment can be seen in Figure 4.12. Here the robot starts navigating the environment in the top left image and different images in this figure show different stages of navigation. The black dot with the white line shows the current position and the orientation of the robot. Small white dots showcase the centers of the detected frontiers. We detect the frontiers by overlaying a mask over the 8-neighborhood of a pixel in the traversability map. The points in the map that have a minimum number of neighbors with no traversability information are marked as frontier points. These points are merged into unique frontiers by detecting the connected components over the frontier points on the 8-neighborhood grid defined over the traversability map. The blue circle shows the frontier that the robot has decided to explore on each step. When deciding which frontier should be explored we take into account the distance to a particular frontier and the cost of carrying out the route to it. For example, a frontier behind the robot is down-weighted with respect to a frontier in front of the robot as the robot must make fewer turns on the way to the latter. The main reason for choosing this behavior over choosing the closes frontier is that the odometry of the robot that drives on tracks is more stable and reliable when the robot traverses a straight line than in turns.

When choosing the frontier to be explored we take into considerations these factors:

1. if the frontier is reachable given the constraints of the robot;
2. how complicated it is to reach the frontier given the constraints of the robot and the currently available map;
3. how much information do we gain from a frontier, i.e. how many frontier points form a particular frontier;
4. if the particular frontier has already been visited before by the robot

All of these factors play a role in the choice of the frontier. The first criterium is very intuitive and discards all the frontiers that the robot cannot reach. We check the reachability of each frontier by planning a path on a 2D map of the environment using the A* algorithm as described in Section 2.8. This map is

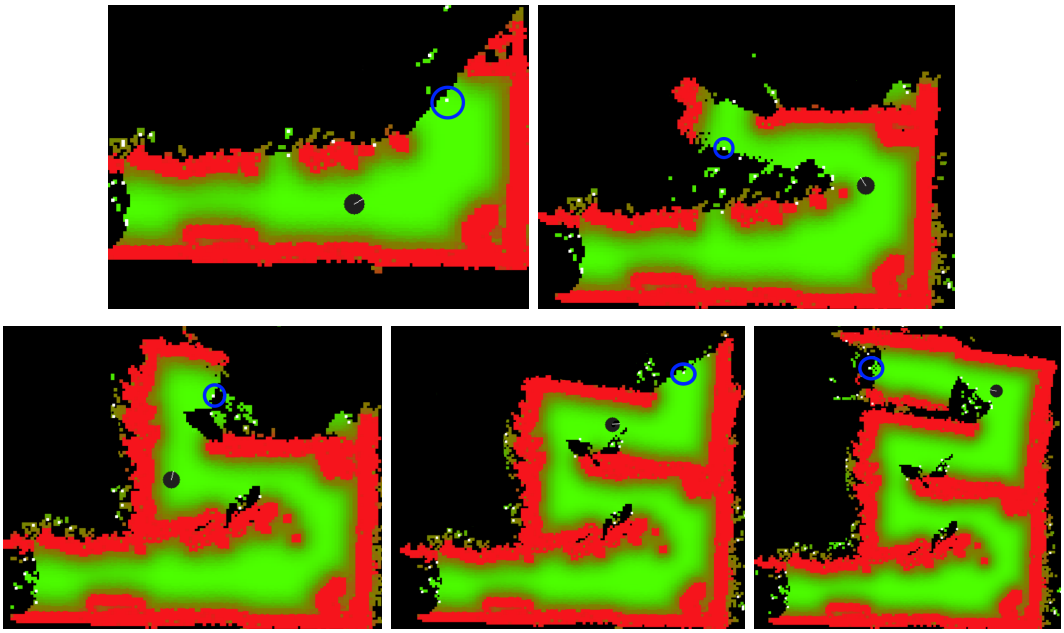

Figure 4.12: A sequence of images that shows how the robot is choosing the best frontier for exploration. The black circle depicts the robot, blue circle shows the currently picked frontier and white points depict all detected frontiers. The robot starts in the situation depicted in the top left corner of this figure. Every next image from left to right, top to bottom depicts the state of the robot when reaching the frontier chosen on the previous step.

generated from the traversability map of the environment by extending the size of all walls by the size of the robot. A valid path is the path that stays in the traversable area of the map and reaches the center of the frontier starting from the current position of the robot.

A frontier can be close to the robot, but would require a multitude of turns on the way. Every turn the robot makes reduces its certainty about its position and creates additional complications on the SLAM system that is running in the background. The odometry of the robot with tracks is more precise when moving on a straight line compared to doing turns. Therefore, when planning a route to the frontiers, we compute the complexity of this route and use this score in our selection algorithm.

The next criterium that we take into account is the potential information gain of the frontier. A frontier that consists of more points that form it, potentially will provide the robot with more information about the environment, when observed. Therefore, we prioritize the frontiers based on their volume.

Lastly, we make sure we do not take into account the frontiers already observed. Most of the times, when the robot reaches a particular frontier it would observe a new part of the environment, thus, pushing the frontier further away from itself. As an example of this we can look at the top-left and top-right images in Figure 4.12. Initially, the frontier that the robot decided to explore was on

the right of the map in the apex of the turn, however, after reaching that point, the newly observed part of the environment removed that frontier and created a new one in the start of the next turn of the environment. This behavior can be observed most of the times with most frontiers. Some points in the map, however, would always lie on the border with an unknown area. This happens if this area in impossible for the robot to observe. This situation can happen if there is a gap in the surface of the floor making the bottom of this hole unseen for the robot given the position of its sensors. Such points will always form a frontier following our definition of the frontier. To deal with this situation we assume that all the frontiers within the sensor reach of the robot are marked as observed and do not play a role in the further exploration procedure.

In the remainder of this section we present an extention to the frontier-based exploration technique that takes into account background information that the robot might have about the environment.

### 4.2.2 Robust exploration using background knowledge

Given the fact that most real robots maintain a probabilistic belief about their pose and the map of the environment, an alternative approach is to select the target location that is expected to minimize the uncertainty in the belief of the robot. In this setting, the exploration problem can be formulated as follows. At each time step $t_i$, the robot has to decide, which action $a$ to execute (where to move next). During the execution of $a$, the robot obtains a sequence of observations $z_{1:t}$ at times $1..t$ that we will denote as $\mathbf{Z}^a$ for better readability. Thus, we can define the expected information gain $I(\mathbf{X}, M; \mathbf{Z}^a)$, also called mutual information, of selecting the action $a$ as the expected change in entropy in the belief about the robot's poses $\mathbf{X}$ and the map $M$:

$$I(\mathbf{X}, M; \mathbf{Z}^a) = H(M, \mathbf{X}) - H(M, \mathbf{X} \mid \mathbf{Z}^a), \tag{4.10}$$

where $H$ stands for Shannon's entropy [112], $M$ for the currently available map and $\mathbf{X}$ for the history of all poses of the robot.

The second term in Equation (4.10) is the conditional entropy and is defined as

$$H(M, \mathbf{X} \mid \mathbf{Z}^a) = \int p(\mathbf{Z} \mid a)\, H(M, \mathbf{X} \mid \mathbf{Z}^a = \mathbf{Z})\, d\mathbf{Z}, \tag{4.11}$$

where $\mathbf{Z}^a$ stands for the observations the robot would make should it carry out action $a$ and $p(\mathbf{Z} \mid a)$ is the probability of a particular observation sequence $\mathbf{Z}$ given the action $a$ has been taken by the robot.

Unfortunately, reasoning about all potential observation sequences $\mathbf{Z}$ in this equation is intractable in nearly all real world applications since the number of

potential measurements grows exponentially with the dimension of the measurement space and with time. It is therefore crucial to approximate the integral of Equation (4.11) so that it can be computed efficiently with sufficient accuracy.

A suitable approximation, however, depends on the environment model, the sensor data, and the application so that no general one-fits-all solution is available. Following the work of Stachniss *et al.* [118], we consider these types of actions: First, *exploration actions* that guide the robot to the closest frontier and reduce the map uncertainty. As we have no further information about the unseen area, it is difficult to distinguish two frontiers with respect to the expected uncertainty reduction. Second, *loop-closing and re-localization actions*, which are key to the uncertainty reduction about the robot's pose.

In this work, we aim at combining these types of actions into a single one. We seek to predict what the so far unseen environment beyond a frontier *may* look like based on background knowledge of previously seen environments and select the frontier that potentially leads to a loop-closure. In this way, we maximize the expected uncertainty reduction in the belief of the robot about the world.

#### 4.2.2.1 Utility function for information-driven exploration

Most exploration systems define a utility function to relate the expected gain in information with the cost of obtaining the information. As long as no constraints such as available energy or similar are considered, the distance that the robot has to travel to obtain its measurements is a standard choice. This yields a utility function of the form

$$U(a) = I(M, \mathbf{Z}; \mathbf{Z}^a) - \text{cost}(a), \tag{4.12}$$

so that the task of selecting the best action can be formulated as

$$a^* = \underset{a}{\text{argmax}}(I(M, \mathbf{Z}; \mathbf{Z}^a) - \text{cost}(a)). \tag{4.13}$$

Throughout this work, we define the cost function $\text{cost}(a)$ as the path length corresponding to action $a$, i.e., the length of the trajectory from the current location of the robot to the designated target location.

As mentioned in the previous section, estimating the expected information gain is challenging and computationally demanding and thus we use the following approximation. We assume that actions can reduce the robot's uncertainty about the map by exploring unseen areas and/or can reduce its uncertainty about the trajectory by closing a loop:

$$a^* = \underset{a}{\text{argmax}}(I_{map}(a) + I_{traj}(a) - \text{cost}(a)). \tag{4.14}$$

As we do not know how large the unknown area and thus the number of unknown grid cells behind a frontier is, we may argue that all frontiers yield the


Figure 4.13: Example of the submap retrieval using FabMAP2. The left image shows the query map, the other ones the best four matches from the database.

same expected information gain with respect to the map uncertainty. Thus, we can simplify Equation (4.14) as long as we consider only exploration actions to frontiers:

$$a^* = \underset{a}{\operatorname{argmax}} \left( I_{traj}(a) - \operatorname{cost}(a) \right). \tag{4.15}$$

The expected information gain about the trajectory $I_{traj}(a)$ is mainly influenced by loop closures. The more likely a loop closure can be obtained when executing an exploration action $a$, the higher its expected gain. Thus, the remainder of this section addresses the problem of predicting possible loop closures.

### 4.2.3 Predictive exploration

The key contribution here is to model the predictive belief describing what the environment may look like in the unexplored areas. To compute this belief, the robot exploits environment structures it has seen in the past—either in the environment explored so far or even from previous missions. Our exploration system uses this predictive belief to evaluate the frontiers as possible target locations for the exploration. This allows us to select the frontiers that are likely to lead to a loop-closure and thus to an active reduction of the uncertainty in the robot's belief. As we show during the experimental evaluation, this approach outperforms the traditional frontier-based exploration system.

Here we present the predictive exploration method from the earlier publication by Perea-Ström *et al.* [96]. We believe the inclusion of this work here aids understanding of the thesis as a whole and therefore Sections 4.2.3.1-4.2.4.1 should be read as a quote.

#### 4.2.3.1 Querying for similar environment structures

The key idea of this approach is to look for similarities between the known areas around a frontier and portions of previously mapped environments. Under the assumption that environments are not random but expose certain structures and that these structures tend to appear more than once, we can use the already mapped areas in order to predict what the environment beyond the frontier *may* look like.

Figure 4.14: Illustration of the loop closures prediction. *Left:* So far explored map with the frontier under consideration (blue circle). *Middle:* One map from the predictive belief (in red) superimposed on the map explored so far. *Right:* Voronoi diagram used for the path search.

The first step is to look for portions of the already mapped environments that are similar to the area around the frontier for which the prediction should be performed. To do this, we incrementally build a database storing all local grid maps that the robot experienced. To perform a similarity query, we compare our local maps with the maps stored in the database. To avoid a large number of expensive map-to-map comparisons to search for similar submaps, we rely on a bag-of-words inspired approach, a technique that is frequently used in computer vision to search for image similarities. More concretely, we apply FabMAP2 by Cummins and Newman [24], an appearance-based approach we can use to efficiently query our database. Although FabMAP2 was originally designed to match camera images, it turns out that we can also use it to effectively search for local grid maps in a large database of maps. As FabMAP2 also provides a likelihood $l(m)$ for each match $m$, we can obtain a belief about possible environment structures. Figure 4.13 shows an illustration of this procedure. The image on the left is a query image and the other images are the top 4 matches reported by FabMAP2.

#### 4.2.3.2 Loop closures prediction

As we are mainly interested in the possible paths through the unknown environment in order to find loop closures and not necessarily the exact geometry, we reduce the maps reported by FabMAP2 to extended Voronoi graphs [5] and do all further computations on these graphs.

FabMAP2 provides us with candidates of matching maps but no geometric alignment between the query map and the reported ones. Thus, we align each map reported by FabMAP2 with our query map. This can be done in a robust manner through a RANSAC-based alignment of the Voronoi graphs using its junction points. Figure 4.14 shows an example of a Voronoi graph aligned with the map explored so far.

The next step, is to search for possible loop closures, for which we use the

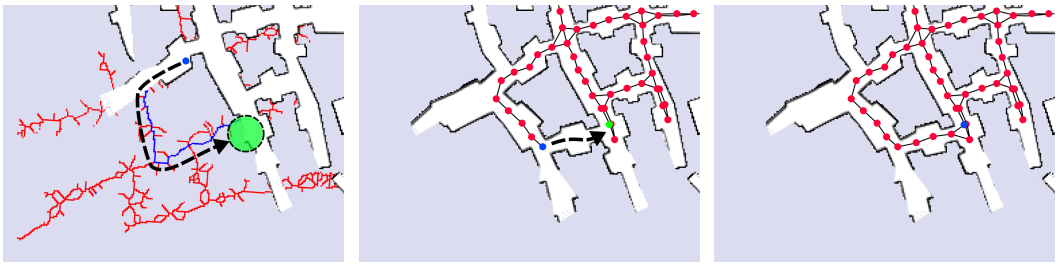

Figure 4.15: Illustration of the active loop closing. *Left:* prediction of the possible path with the loop closure shown in blue. *Middle:* the robot explores the path along the predicted loop closure and perceives the actual structure of the scene. The graph in the already explored environment shows the pose graph of the SLAM system. *Right:* successful loop closure Please note that the predicted environment is actually not identical with the real environment but reveals a similar structure. This similarity resulted in the shown loop closure.

extended Voronoi graph. Starting from the frontier point, we traverse the Voronoi graph in a breadth-first manner. During the traversal, we check if the Voronoi graph leads to a position that is close to any other frontier in the map built so far. If this is the case, we regard that as a possible loop closure. Such a situation is illustrated in the left image of Figure 4.15. This process is executed for each frontier.

#### 4.2.3.3 Estimating the probability of closing a loop

Each map reported by FabMAP2 comes with a likelihood. Thus, we can approximate the probability of closing a loop when executing an exploration action as

$$S_f = \sum_{m \in \mathfrak{M}(f)} l(m) \sum_{c \in \mathfrak{C}(f,m)} l(c \mid m) \tag{4.16}$$

Here, $\mathfrak{M}(f)$ is the set of matches returned by FabMAP2 when querying with the frontier $f$, and $l(m)$ the likelihood of a match $m$. The term $\mathfrak{C}(f, m)$ refers to the set of possible loop closures computed according to the breadth-first traversal explained above and $l(c \mid m)$ is the likelihood that the loop closures can be reached. We assume that $l(c \mid m)$ is proportional to the inverse length of the path of the predicted loop closure. This means that short loop closures are more likely than long ones.

Assuming that every executed loop closure through unknown areas of the map yields the same expected uncertainty reduction, we can approximate the expected information gain $I_{traj}$ of Equation (4.15) with the score $S_f$ according to Equation (4.16). This is clearly a strong assumption but we argue that a high score indicates a high expected gain from exploring the frontier.

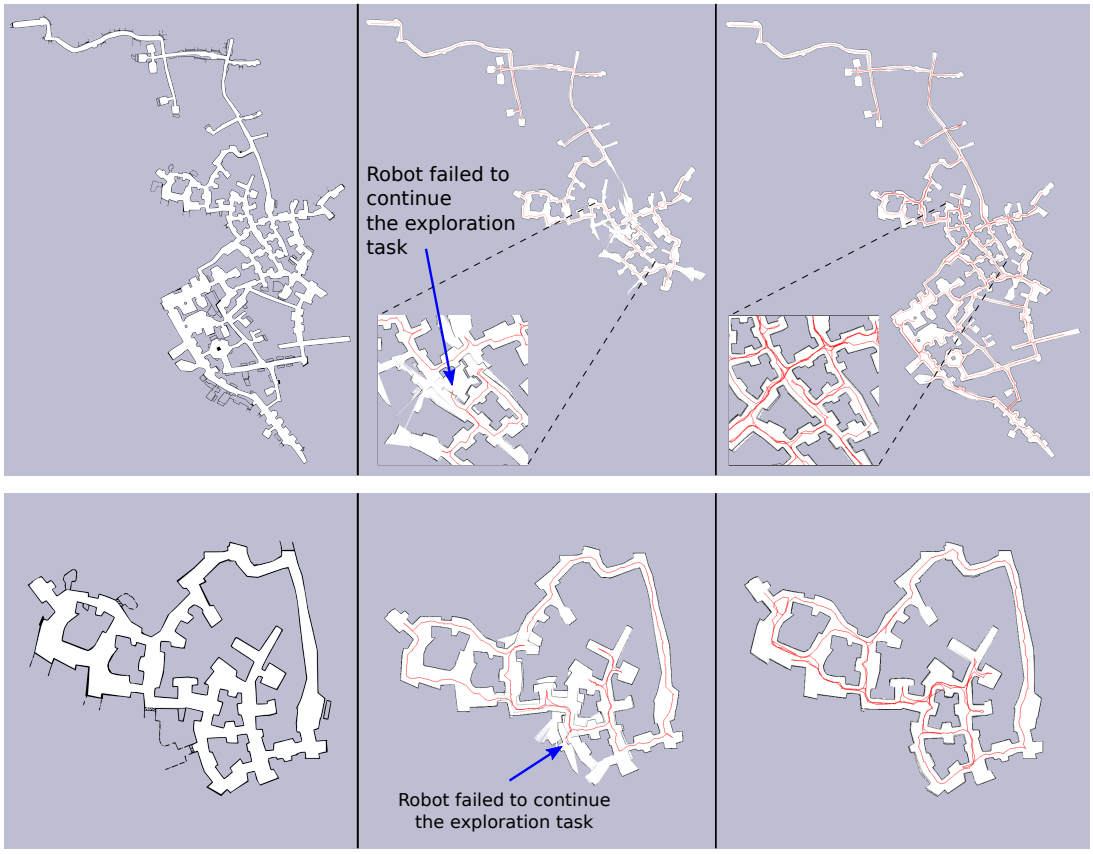


Figure 4.16: Two performance comparisons in constant odometry bias scenario. *Left:* the original map. *Middle:* the closest frontier approach. *Right:* our prediction-based approach. Note that the nearest frontier approach produces a map that is not consistent with the original one, so that the robot can not continue the exploration task. The map produced by the prediction-based approach is instead consistent with the original one.

### 4.2.4 Predictive exploration experiments

We compare our predictive exploration approach to the frontier-based exploration approach described in Section 4.2.1 as a baseline and show that our approach selects frontiers that lead to loop closures which in turn result in improved maps of the environment. The underlying mapping framework for all exploration experiments is a state-of-the-art graph-based SLAM system that uses scan matching for incremental alignments and loop closures.

#### 4.2.4.1 Map comparisons

In this section we show that our system outperforms the frontier-based one in terms of the resulting map quality. Figure 4.16 illustrates the obtained results for two environments using exactly the same mapping system and identical parameters for the comparison. The map database used for predictions consists of maps constructed from other catacomb sites representing a similar type of envi-

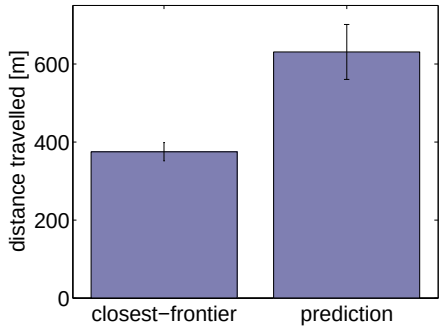


Figure 4.17: Mean and standard deviation of the distances traveled in the frontier-based approach and in the proposed approach.

ronment but not the same one. The images on the left are the "ground truth" maps obtained from manual surveys. The images in the middle correspond to the results of the frontier-based exploration, while the images on the right show our approach. Our approach yields a consistent model of the environment, while the frontier-based approach fails as can be seen from the inconsistencies shown by the blue arrows in the figure. Using the frontier-based approach, the robot was unable to continue its exploration task due to an inconsistent map that prevented the computation of further exploration actions. We performed similar experiments in different nested tunnel environments and obtained comparable results.

The exploration task strongly benefits from achieving loop closures early, avoiding a high uncertainty in the pose-graph. Our approach improves the amount of loop closures whenever the current environment resembles previously seen maps, either in previous or current explorations runs. Thus, a new environment with recurrent structures also benefits from this approach.

In case there is no similarity between the current environment and the maps stored in the database, no map should be retrieved and thus the system falls back to the frontier-based exploration.

The execution time of our approach depends on the number of unexplored frontiers, as well as on the map size and resolution. On a standard computer and a map size of 150 m by 100 m with a grid resolution of 4 cm, next view point selection time ranged from 131 ms up to 4.8 s in the most complex situation. In practice, time consumed by the robot reaching the next view point usually dominates over the time consumed by the selection task.

#### 4.2.4.2 Exploration path length

The advantages of the prediction-based approach come at a cost—the cost of traversing exploration paths that are longer than the ones generated by the frontier-based approach. This experiment is designed to evaluate the increase in path length. As we are not able to obtain consistent maps for the frontier-

based approach under a realistic noise model for the task under consideration, we executed this evaluation under zero noise assumption in the simulator. Using a zero noise odometry, also the frontier-based system is able to build consistent maps. In this setting there is no advantage in using our predictive approach as the pose uncertainty is zero and no uncertainty reduction is gained from closing loops. We compared the distances traveled for the frontier-based and our approach. The distances traveled are summarized in Figure 4.17. In the worst-case scenario, the path generated by our approach was 1.85 times longer than the one of the frontier-based approach. The minimum increase was a factor of 1.5. Generating on average a 1.7 times longer trajectory is clearly an overhead paid for actively closing loops and in this way reducing uncertainty, however, this price must be paid.

### 4.2.5 Related work

The majority of techniques for mobile robot exploration focus on generating motion commands that minimize the time needed to cover the whole terrain. Several techniques also assume that an accurate position estimate is available during exploration [70, 140]. Whaite and Ferrie [135] present an approach that uses the entropy to measure the uncertainty in the geometry of objects that are scanned with a laser range sensor. Similar techniques have been applied to mobile robots [117, 104], but such approaches still assume to know the correct pose of the vehicle. Such approaches take the map but not the pose uncertainty into account when selecting the next vantage point. There are, however, exploration approaches that have been shown to be robust against uncertainties in the pose estimates [33, 69].

Besides the idea of navigating to the next frontier [140], techniques based on stochastic differential equations for goal-directed exploration have been proposed by Shen *et al.* [113]. Similar to that, constrained partial differential equations that provide a scalar field into unknown areas have been presented by Shade *et al.* [111]. An information-theoretic formulation that seeks to minimize the uncertainty in the belief about the map and the trajectory of the robot has been proposed by Stachniss *et al.* [118]. This approach builds upon the works of Makarenko *et al.* [78] and Bourgault *et al.* [15]. Both extract landmarks out of laser range scans and use an Extended Kalman Filter to solve the underlying SLAM problem. They furthermore introduce a utility function which trades-off the cost of exploring new terrain with the potential reduction of uncertainty by measuring at selected positions. A similar technique has been presented by Sim *et al.* [114], who consider actions to guide the robot back to a known place in order to reduce the pose uncertainty of the vehicle. Such information-driven techniques have also been used for perception selection to limit the complexity of

the underlying optimization problems in SLAM [73].

In general, the computation of the expected entropy reductions is a complex problem, see Krause and Guestrin [72], and in all real world systems, approximations are needed. Suitable approximations often depend on the environment model, the sensor data, and the application. In some cases, efficient approximations can be found, for example in the context of monitoring lakes using autonomous boats [59].

Other approaches, especially in the context of autonomous micro aerial vehicles (MAVs), seek to estimate the expected feature density in the environment in order to plan a path through areas that support the helicopter localization [105]. This can be seen as related to information-theoretic approaches, although Sadat *et al.* [105] do not formulate their approach in this framework. A number of approaches related to MAV exploration seek to select new vantage points during exploration, so that the expected number of visible features and thus the information gain is maximized, see Mostegel *et al.* [85] and Palazzolo *et al.* [95].

An interesting approach by Fox *et al.* [39] aims at incorporating knowledge about *other* environments into a cooperative mapping and exploration system for multiple robots. This allows for predicting simplified laser scans of an unknown environment. This idea was an inspiration for our paper for predicting possible loop closures given the environment structure explored so far. We use this approach for exploring ancient catacombs, which are repetitive underground environments, with a mobile platform, see Figure 4.11. Chang *et al.* [21] propose an approach for predicting the environment using repetitive structures for SLAM. Other background knowledge about the environment, for example semantic information [121], can support the exploration process as shown by Wurm *et al.* [138], Stachniss *et al.* [120] as well as Holz *et al.* [60]. In addition to this, if a topo-metric graph of the environment is known a-priori methods can use it as additional information. Oßwald *et al.* [92] makes use of this information to significantly reduce the exploration time with respect to the standard frontier-based approach.

### 4.2.6 Conclusion

In this section, we presented an approach for robust and reliable autonomous exploration that uses the previously-acquired knowledge about the environment. This approach ensures that the loop closures are part of the exploration utility function, thus improving the robot localizability. The work presented here is based on a conference publication [124], which described the idea of predictive exploration. While the exploration technique is not the main contribution of this thesis, building on top of this work, we integrate it into our system and tightly couple it with the robust homing method presented in Section 4.3 and in the related journal article [96].

## 4.3 Robust homing

The ability to robustly operate without user intervention is an important capability for exploration robots, especially if there are no means for communication between the robot and an operator. In the previous section we have introduced a method for exploration that actively closes the loops, thus, making the map estimation process more robust. However, even despite our best effort to navigate the robot in a way to maintain a consistent map, state estimation processes in the robot's navigation system may still fail, and there is no guarantee that they will operate flawlessly throughout the time the robot navigates. Thus, it is of great importance to design a system that enables the robot to navigate back to its starting position without the risk of getting lost, even if a component as central as the mapping system fails. This is the challenge that we address in this section.

Our work is motivated by the autonomous exploration robot depicted in Figure 4.11. It is used to explore difficult-to-access underground environments such as ancient Roman catacombs. As no communication between the robot and an operator is possible most of the time, the platform has to be truly autonomous. During the course of an exploration task, as described in Section 4.2, the robot constructs a map of the environment via a graph-based SLAM system. As long as this map is consistent, the robot can perform autonomous navigation by planning the shortest path—for example using A*—from its current location to its starting point using the map. Although modern SLAM systems are fairly robust, they may still fail, for example, due to wrong data associations generated by the SLAM front-end. Even current state-of-the-art SLAM system cannot *guarantee* the consistency of the resulting map, nor do they provide robust means to decide whether the constructed map is consistent or not. Computing a path based on an inconsistent map, however, is likely to lead to a failure and a possibility of losing the robot if operating in a hazardous environment.

To avoid that the robot gets lost, we propose a novel robust homing system consisting of two distinct parts. The first part performs a statistical analysis of the map and thus provides information about its consistency. We build upon the work of Mazuran *et al.* [80] for performing a cascade of pair-wise consistency checks using observations perceived from the same areas with a high-precision 2D LiDAR. We adapt this method to RGBD sensors and extend it to avoid performing such checks on the overall map by reducing the area that must be analyzed. We plan the shortest homing route for the robot assuming a consistent map and analyze the map consistency only along that path. This allows us to estimate online if the map around this path is consistent with a given confidence level. The second part of our approach is responsible for driving the robot back to its starting location by rewinding the trajectory that the robot took to reach

its current pose. This operation is performed without any map knowledge (as the map is inconsistent). If the motions of the robot were perfect, i.e., if they would lead *exactly* to the desired robot pose in the world frame, we would be able to simply invert the motion commands performed by the robot and could safely reach the starting location. Both, motion execution and odometry, however, are often noisy and can generate erroneous estimates. As a result, simply following inverse motion commands will not bring the robot to the starting location in the real world. Therefore, we take into account the sensor information to guide the robot back by matching the observations with the ones taken in the past.

### 4.3.1 Robust homing using map consistency checks

Under the assumption of map consistency, homing can be solved with existing tools. This can be done by computing a collision-free path from the current to the starting location and executing this path with a standard navigation pipeline. Such a navigation system would, for instance, localize the robot in the map built so far and plan the shortest path towards home using A* or a similar approach. If the map, however, is not consistent due to a failure in the underlying SLAM system, this approach is likely to lead to a deadlock situation from which the robot cannot escape easily.

We address the problem of robust homing in a two-stage approach. First, we plan a path from the current location under the assumption of a consistent map. Then, we apply a map consistency estimation to evaluate if the map along the planned path is consistent with a given confidence. In the second stage, we navigate the robot back home. If the planned path is consistent, we simply execute this plan. Otherwise, we aim at reversing the trajectory that the robot has executed so far by aligning the current observation with the observations obtained on the way from the starting location to the current one to return the robot safely to its starting position. As we show in this section, this yields a robust strategy to bring a robot back to its starting location.

Here, we first introduce the consistency check taken from an earlier work by Mazuran *et al.* [80] in Sections 4.3.1.1-4.3.1.3. These sections should be therefore read as a quote and we include them here to aid understanding of the big picture depicted in this thesis. This approach has been originally proposed for 2D LiDARs and in Section 4.3.2 we focus on adaptations made to this approach to be used in our setup with a 3D depth sensor.

#### 4.3.1.1 Pairwise inconsistency measure

Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two laser scans, taken from two different poses expressed in global coordinates. The key part of this method is to compute a polygon spanning

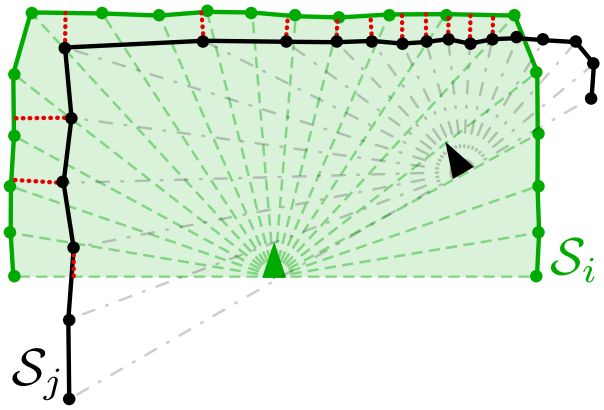


Figure 4.18: Example showing sample occlusions by $\mathcal{S}_i$ by $\mathcal{S}_j$ using the method of Mazuran *et al.* [80]. The set of green and black line segments compose the polylines of $\mathcal{S}_i$ and $\mathcal{S}_j$, while the shaded green area represents the visibility polygon of $\mathcal{S}_i$. The lengths of the dotted red lines represent the sample occlusions of $\mathcal{S}_i$ by $\mathcal{S}_j$, for all obstacle vertices of $\mathcal{S}_j$ falling into the visibility polygon of $\mathcal{S}_i$. Image courtesy of Mazuran *et al.* [80].

over its end points and the position of the robot. Each polygon describes the free space that is covered by the scan. The intuition is that $\mathcal{S}_1$ and $\mathcal{S}_2$ are considered to be consistent with each other if none of the end points of $\mathcal{S}_1$ lies inside the polygon of $\mathcal{S}_2$ and vice versa. We call the points of $\mathcal{S}_1$ lying inside $\mathcal{S}_2$ the *inconsistent points* of $\mathcal{S}_1$ w.r.t. $\mathcal{S}_2$.

The measure we derive from this intuition computes the sum of the distances of each inconsistent point of the first scan to the boundary of the polygon surrounding the second scan and vice versa. Let $\boldsymbol{\mathcal{V}}_i$ be the set of points defining the visibility polygon of $\mathcal{S}_i$ and $\mathbf{p}_i^k$ its $k$-th end point. We define the *inconsistency distance* $d_i(\mathbf{p})$ for a point $\mathbf{p}$ as:

$$d_i(\mathbf{p}) = \begin{cases} \operatorname{dist}(\mathbf{p}, \boldsymbol{\mathcal{V}}_i) & \text{if } \mathbf{p} \text{ inside } \boldsymbol{\mathcal{V}}_i \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

where $\operatorname{dist}(\mathbf{p}, \boldsymbol{\mathcal{V}})$ is the Euclidean distance of a point $\mathbf{p}$ to the closest point on the polygon boundary $\boldsymbol{\mathcal{V}}$. By summing over all the end points of the scans, we obtain the *pairwise inconsistency measure*

$$\mathfrak{M}_{ij} = \sum_k d_i(\mathbf{p}_j^k) + \sum_k d_j(\mathbf{p}_i^k). \tag{4.18}$$

See Fig. 4.18 for an example. In the remainder of this section, we refer to the total number of inconsistent points in $\mathcal{S}_i$ w.r.t. $\mathcal{S}_j$ and in $\mathcal{S}_j$ w.r.t. $\mathcal{S}_i$ as $n_{ij}$. A naïve implementation computes $\mathfrak{M}_{ij}$ in $\mathcal{O}(K^2)$, where $K$ is the number of end points.

#### 4.3.1.2 Hypothesis test for pairs of scans

Under the assumption of a correct alignment of two scans $\mathcal{S}_i$ and $\mathcal{S}_j$, we expect that on average 50% of the end points in $\mathcal{S}_i$ are inconsistent points of $\mathcal{S}_j$ and vice versa. This is due to the inherent sensor noise in the laser measurements.

If $\mathcal{S}_i$ and $\mathcal{S}_j$ are obtained from the exact same pose and the laser range values are normally distributed, the inconsistency distances are half-normally distributed, with the scale parameter $s^2$ given by twice the range variance of the sensor. The half-normal distribution is the distribution of $Y = |X|$, with $X \sim \mathcal{N}(0, s^2)$.

We further assume that the inconsistencies are i.i.d. random variables with finite mean $\mu$ and variance $\sigma^2$. In the case of the half-normal distribution, we have

$$\mu = s\sqrt{\frac{2}{\pi}} \qquad\qquad \sigma^2 = s^2\left(1 - \frac{2}{\pi}\right). \tag{4.19}$$

According to the central limit theorem, $\mathfrak{M}_{ij}$ follows

$$\frac{\mathfrak{M}_{ij} - \mu\, n_{ij}}{\sigma\sqrt{n_{ij}}} \underset{n_{ij}\to\infty}{\sim} \mathcal{N}(0,1). \tag{4.20}$$

The former can be generalized to situations in which the random variable follows different distributions, as long as they are independent and they satisfy Lindeberg's condition [3]. In this case, the limit of the mean is still normally distributed.

As a result of that, we can conduct a hypothesis test if $n_{ij}$ is sufficiently large and $\mu$ and $\sigma^2$ are known. The hypothesis test allows us to determine whether $\mathfrak{M}_{ij} \sim \mathcal{N}(\mu\, n_{ij},\, \sigma^2 n_{ij})$. It requires a one-sided test, as only large $\mathfrak{M}_{ij}$ values are relevant; a low $\mathfrak{M}_{ij}$ value only implies that larger deviations from the perfect map assumption are accepted.

Based on $N$ laser scans, we can build a $N \times N$ symmetric matrix $\mathbf{\Psi}$ containing standardized $\mathfrak{M}_{ij}$ values:

$$\mathbf{\Psi} = [\Psi_{ij}] = \left[\frac{\mathfrak{M}_{ij} - \mu\, n_{ij}}{\sigma\sqrt{n_{ij}}}\right]_{\substack{1\le i\le N \\ 1\le j\le N}}. \tag{4.21}$$

Here, the sparsity of this matix depends on the amount of overlap between laser scans.

Given $\mathbf{\Psi}$, we can infer whether two scans $\mathcal{S}_i$ and $\mathcal{S}_j$ are consistent with confidence $1 - \alpha$ by verifying the inequality:

$$\Psi_{ij} \le F^{-1}(1 - \alpha), \tag{4.22}$$

where $F^{-1}(\mathbf{p})$ is the inverse CDF of the normal distribution.

By adopting a bounding box test and assuming that the pairwise measure of compatibility is determined in $\mathcal{O}(K^2)$ complexity, $\mathbf{\Psi}$ can be computed in $\mathcal{O}(N^2 + NRK^2)$, where $R$ denotes the average number of laser scans a scan overlaps with respect to the bounding box and $K$ is the number of laser beams per scan.

#### 4.3.1.3 One-vs-all consistency test for scans

We conduct the above-described test for all pairs of scans and consider a map to be consistent if all tests are successful. The problem, however, is that a single statistical test will produce the wrong result with probability $\alpha$. Thus, testing a single scan that overlaps with $r$ other scans yields a type I error probability of $1 - (1 - \alpha)^r$. This renders the direct application of the pairwise approach unsuitable.

To overcome this issue, we represent the outcome of a pairwise hypothesis test as a Bernoulli-distributed random variable with parameter $\alpha$. Thus, the number of failed tests follows a binomial distribution with parameters $\alpha$ and $r$. Given that, we can compute the maximum number $\hat{\xi}$ of tests that can fail for a confidence level $1 - \alpha'$ as:

$$\hat{\xi} = \min_{0 \leq \xi \leq r} \left\{ \xi \left| \sum_{i=\xi+1}^{r} \binom{r}{i} \alpha^i (1-\alpha)^{r-i} \leq \alpha' \right. \right\}. \tag{4.23}$$

This allows for computing a cascaded hypothesis test for the consistency of a scan with respect to all scans it overlaps with. We first perform all pairwise hypothesis tests. Then, if the number of failed tests is smaller than $\hat{\xi}$, the overall consistency test is positive.

### 4.3.2 Adapting consistency check to RGBD data

The approach of Mazuran *et al.* [80] has been initially formulated for high-precision 2D laser scans. There are two different ways of how this method can be applied in our setting. It can either be extended towards 3D data by substituting the polygons with triangulations of the full Kinect-generated 3D scan or, alternatively, the 3D scans can be reduced to a 2D scan and analyzed in a similar way to the original approach. We argue that there is no need for the more complex 3D-based approach[1] as the robot is a track-based platform that is—at least locally—operating roughly on a plane. We found that the 2D solution is well suited for the task at hand, at least for our type of environments.

On our robot, each 3D point cloud is created by combining the point clouds generated by two Asus Xtion cameras, see Figure 4.19. For every local 3D point

[1]We refer only to the consistency check and not to the SLAM system, which takes into account all six degrees of freedom.

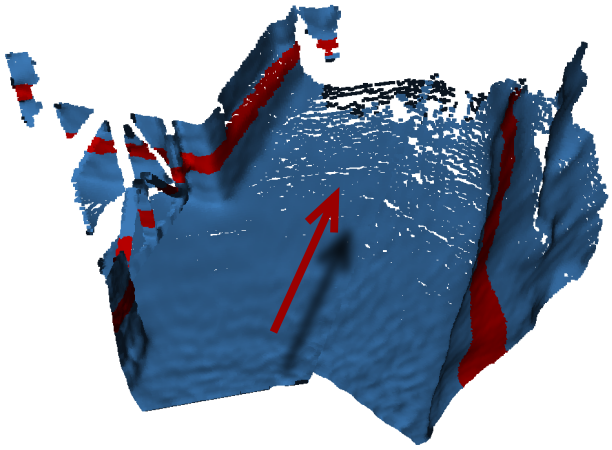

Figure 4.19: An example of the point cloud from the double Asus Xtion system we use to log data in the Priscilla catacomb. The red line shows the "scan" line to generate the simulated 2D scan from 3D point cloud. We discretize all the points within the red line into bins according to their horizontal dispersion from the camera viewing direction shown here as a floating red arrow.

$[x\ y\ z]^\top$ that is within a small margin from the height of the scan line illustrated by the red stripe in the image, we compute its bearing from the center of the robot (red arrow in the figure). The relative bearing can directly be computed through $\alpha = \text{atan2}(y, x)$ and the virtual range reading by the Euclidean distance from the robot's center to $(x, y)$. For our setup, this results in an approximate field of view of $[-\pi/4, \pi/4]$. Such virtual range scans are used for the statistical consistency check described above. A map combined from the scans generated in this manner can be seen in the Figure 4.20. In order to account for much lower precision of the RGBD cameras with respect to 2D LiDARs we adapt the scale parameter $s^2$ that depends on the range variance of the sensor. All the decisions about map consistency are carried out on a 2D map constructed from these virtual scans.

### 4.3.3 Map consistency estimate for the way to home

Given the consistency test presented above, we can perform a mathematically sound statistical test to evaluate if a map is consistent or not. However, what the robot really needs to know is not the consistency of the full map. Instead, it is sufficient to know if it can safely move along a specific path through the environment to the start location. Thus, we plan a path with A* *assuming* that the current map is consistent and extend the statistical consistency check to consider only the scans along that path. To achieve this, we select all recorded locations that were closer than twice the maximum sensor range away from the trajectory

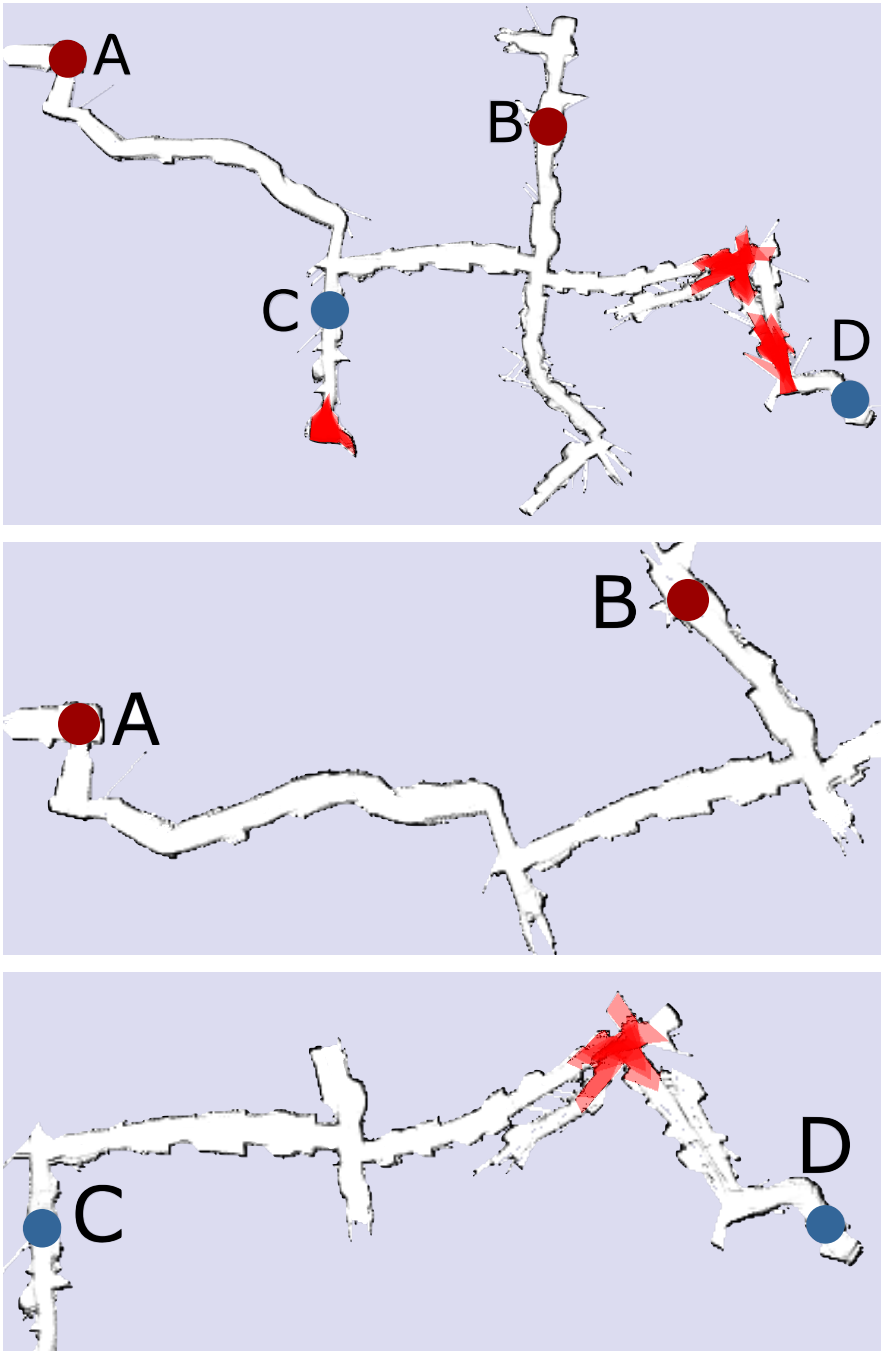


Figure 4.20: *Top:* a map built so far with the detected inconsistencies (inconsistent scans are shown in red). *Middle:* image shows a submap that is built using only the scans recorded around the A* path from A to B computed in the full map. In this example, no inconsistencies are present and none are detected. *Bottom:* image is constructed in the same way as the middle one, but the A* path is computed from C to D and here, the map inconsistencies are correctly detected.

planned with A*. Examples of such partial maps can be found in Figure 4.20. The top image shows an inconsistent 2D map of the Priscilla catacomb. Directly applying the approach explained in Section 4.3.1 would label the whole map as inconsistent. In contrast to that, if the robot only takes into account the shortest route from A to B, it can still safely perform the navigation task, as shown in the middle image of the same figure. This is not the case if the robot wants to go from C to D as it will encounter an inconsistent part of the map on its way.

### 4.3.4 Homing by rewinding the trajectory

Once the consistency check has identified that the submap including the current path is inconsistent, we need to perform the trajectory rewinding to bring the robot home safely. We can view the robot's forward trajectory as a series of poses of the robot $\boldsymbol{\mathcal{P}} = \{\mathbf{p}_0, \ldots, \mathbf{p}_n\}$, where $\mathbf{p}_i = [x_i \; y_i \; \theta_i]^\top$. The trajectory is expressed as a sequence of 2D positions $(x_i, y_i)$ and orientations $\theta_i$ as we can only steer the robot on a ground plane. The task of rewinding the trajectory is to drive the robot from $\mathbf{p}_n$ to $\mathbf{p}_0$ backwards. This, however, requires us to correct the error in odometry. We do this correction by aligning the sequence of point clouds that the robot perceives with the ones taken before (corresponding to $\mathbf{p}_n, \ldots, \mathbf{p}_0$). We subsample the trajectory in such a way that each pose in $\boldsymbol{\mathcal{P}}$ is either 1 m away from the previous one or that there is a rotation of at least 10° between two poses.

Without loss of generality, we consider that the robot has to carry out an action to move from $\mathbf{p}_i$ to $\mathbf{p}_j$ and at the same time to compensate for the error in odometry. To do so, at each time step $t$, the robot exploits the point cloud $\mathcal{C}_t$ obtained after executing the movement from $\mathbf{p}_i$ to $\mathbf{p}_j$. In an ideal scenario, the command should have brought the robot to the pose $\mathbf{p}_j$. In reality, there is an error caused by slippage, uneven ground, etc. Thus, we align current cloud $\mathcal{C}_t$ with the expected one, i.e. the previously stored during the forward pass cloud $\mathcal{C}_j$. To achieve that, we can use any ICP algorithm, for example the approach presented in Section 3.1. However, this method has not been developed at the time of implementation of this part of our work and NICP [109] has been used instead to find the discrepancy between the point cloud that the robot *expects* to perceive and the one that it *actually* perceives. The NICP method extends the point-to-plane error metric proposed in Generalized ICP [107] by accounting not only for the metric distance between the points but also the curvature of the underlying surface. The transformation between the point clouds provided by the matching algorithm is an SE(3) transformation between the poses at which the two point clouds $\mathcal{C}_t$ and $\mathcal{C}_j$ were taken. As our robot moves on the ground, we project the poses onto the ground plane in order to obtain an SE(2) transformation. This transformation is parameterized by $\mathbf{p} = [\Delta x \; \Delta y \; \Delta\theta]^\top$. The coordinates $\Delta x$ and $\Delta y$ are the coordinates of the point cloud $\mathcal{C}_t$ expressed in the local coordinate

frame defined by the position where the robot expects to arrive. Knowing the pose $\mathbf{p}_j$ and the local position of $\mathcal{C}_t$ enables us to compute the current position of the robot in the global odometry frame.

In addition to a 2D pose, we find the orientation of the robot $\theta_t$ as a sum of $\theta_j$ and $\Delta\theta$ normalized to $(-\pi, \pi]$. We use this new pose to generate a motion command to reach the next pose from the recorded trajectory. We continue this process until the robot is within $d_{\max} = 1\,\mathrm{m}$ from the starting pose $\mathbf{p}_0$.

### 4.3.5 Scalability

The map consistency test requires each scan to be tested against all other scans in its vicinity, therefore the number of checks grows with the length of the trajectory under test. However, we only need to perform the consistency check when the robot plans a new path, i.e., before the actual movement. Therefore it is safe to spend even longer time for the consistency check should it be needed, although our experiments in real-world catacombs show that doing a consistency check along a planned path usually takes substantially less than a second.

The path rewinding homing procedure computations do not depend on the length of the trajectory. At each moment only two point clouds need to reside in memory and ICP is performed pairwise. The rest of the point clouds are stored on a hard disk in a strict order and are loaded in a sequential order into memory on demand.

### 4.3.6 Experiments

The evaluation is designed to illustrate the capabilities of our approach and to support the claims made in this section. Our two main claims are that (i) the map consistency check works on the virtual 2D scans and is able to evaluate map consistency in an online fashion and (ii) we can rewind trajectories in case of failures of the mapping system and yields a homing behavior not reliant on a map. All experiments have been conducted on real world data and on a real robot. All components are ROS nodes and operate on the notebook computer installed on the robot, available as open-source software as part of the ROVINA software suite.

The robot is controlled through an own navigation system that uses the ROS communication infrastructure. Its SLAM system is a pose graph-based system [47] that aligns the depth images from the cameras. The optimization is done with g$^2$o [48] and loop closure hypotheses are generated based on the similarity of local maps stored in the nodes of the graph. The local maps are generated by incrementally aligning the input data until a certain distance of orientation change is accumulated. In this part any ICP algorithm can be used for incre-

mental point cloud alignment for example the approach presented in Section 3.1. However, this method has not been developed at the time of implementation of this part of work and NICP [109] has been used instead.

In terms of the persistent data structure that is used to store all the information, we use a generalization of a pose graph. Each node in the graph corresponds to a pose of the robot at time $t$. In addition to that, each node stores the original odometry pose $\mathbf{p}_t$ and the corresponding 3D point cloud $\mathcal{C}_t$ from the RGBD cameras. To efficiently handle this data structure even for large environments, the pose graph with the nodes $\mathbf{p}_t$ itself is kept in memory but the corresponding point clouds $\mathcal{C}_t$ are stored on disk and are loaded on demand.

The data is gathered using a setup with two Asus Xtion RGBD cameras. Both cameras point forward, one slightly rotated to the left and the other one to the right with a minimal overlap in the middle. The robot computes the point clouds and generates 2D scans from the point clouds for the consistency check as described above.

First, Figure 4.20 illustrates an example of the statistical map consistency check performed on the real data (virtual 2D scans from Kinect-generated point clouds) from the Priscilla catacomb in Rome. The partial maps computed around the shortest path are usually substantially smaller than the map of the whole environment, especially if the environment has multiple alternative branches and forms a complicated network of corridors or rooms. This is often the case in catacombs or underground mines. Testing smaller maps results in a significant speed-up of the statistical consistency evaluation procedure. The timings for the maps presented in Figure 4.20 are as follows, top map: 2.93 s; middle map: 0.14 s; bottom map: 0.17 s. The maps in the figure contain, respectively, 1101, 137 and 164 different scans. The computational time depends on the number of scans to analyze and the gain in speed grows with the reduction in size of the tested map, and with the reduction of overlap between the scans. We performed the map consistency test on 5 different maps recorded in real catacombs and the consistency check always generated correct results. In sum, we prove that it is feasible to test a map in less than 200 ms, and therefore also to execute the algorithm online. Furthermore, if needed, most of the computations can be cached during navigation. This is the case when dealing with huge maps. In such instances, the test would require a recomputation only if the SLAM back-end changes the configuration of the pose graph substantially.

As a second step, we need a robust method to return the robot to the starting location if the statistical consistency check claims the map to be inconsistent. Thus, the next set of experiments is designed to backup our second claim, i.e., that our approach can robustly rewind trajectories yielding a homing behavior that does not rely on a map. We evaluate the ability of our approach to rewind

the trajectory by carrying out 20 experiments in our lab environment as well as 10 experiments in a catacomb-like, man-made cave in Niederzissen near Bonn. The latter consists of long tunnels and several small room-like structures.

In Figures 4.21-4.24 we depict different tracks from our homing procedure. The original odometry measurements from the forward path are drawn in black (hardly visible as the red trajectory overlays it). The red line illustrates the subsampled trajectory that the robot has selected as its sequence $\boldsymbol{\mathcal{P}}$ for rewinding the trajectory. Both trajectories overlay because the robot does not use any map (it would be inconsistent) and relies solely on the observations and poses it recorded on the forward path to navigate back. The blue line shows the poses from $\boldsymbol{\mathcal{P}}$ (poses on the red line) after the alignment by NICP, thus yielding an estimate of the robot's real position in the odometry frame.

One of the lab experiments is illustrated in Figure 4.21. Here, we steered the robot on a trajectory through an obstacle parcour containing narrow passages as well as areas with numerous flat walls. The robot activated the "trajectory rewind" behavior after we manually introduced a fault in the SLAM system (by adding incorrect constraints). Consequently, the robot followed the way in reverse order using pair-wise point-cloud alignment with the NICP-based pose correction. Three examples are illustrated in Figure 4.22.

We also executed the same system in the cave of Niederzissen, see Figure 4.23 and Figure 4.24. Here, the floor is covered with dust and dirt and it is quite uneven, which causes substantial track slippage and a comparatively poor odometry. Even under such conditions, the robot is able to rewind the trajectory as illustrated in Figure 4.23. Again, we varied the trajectory 10 times, a second longer experiment is shown in Figure 4.24 and we were always able to robustly drive the robot back to the start location.

As can be seen from the close similarity between the rewinding and corrected trajectories in Figures 4.21-4.24, our approach ensures that the robot is able to accurately rewind its trajectory. The deviation of the rewinding trajectory (blue line) from the original trajectory (black line) is approximately 5 cm to 7 cm in the shown datasets and is largely caused by the imprecise motion command execution and strong track slippage. The gray line depicts the pure odometry measurements recorded while rewinding the trajectory. From the gray trajectory, we can see that the matching of observations must be taken into account—otherwise, the robot would deviate substantially from the reference path (and would collide with walls and obstacles).

Overall, our evaluation illustrates that our robot is able to rewind its trajectories through different environments. While rewinding the trajectory, the robot moved backwards most of the time and thus it cannot observe obstacles on the path before it fully passed them. Only by following the reference trajectory

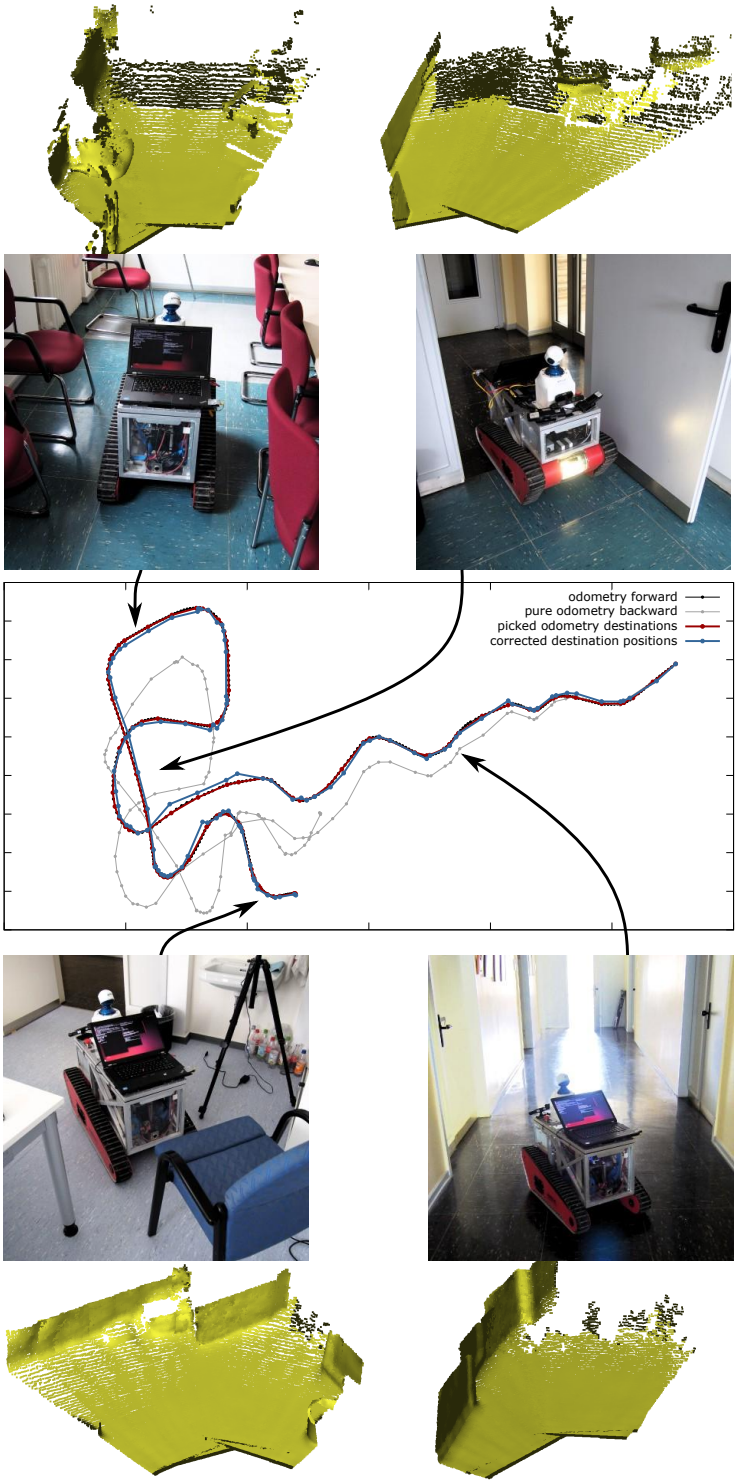


Figure 4.21: Illustration of rewinding the trajectory through the office environment. The robot is steered from the bottom "tail" of the depicted trajectory to the upper-right one. The black line denotes the odometry poses saved while the robot is steered, gray denotes the odometry on the way back, red shows the temporary destination poses picked from the odometry and blue shows the same poses after the ICP correction. The pictures show several example locations visited by the robot. These feature tight doors to rooms as well as feature-scarce corridors.

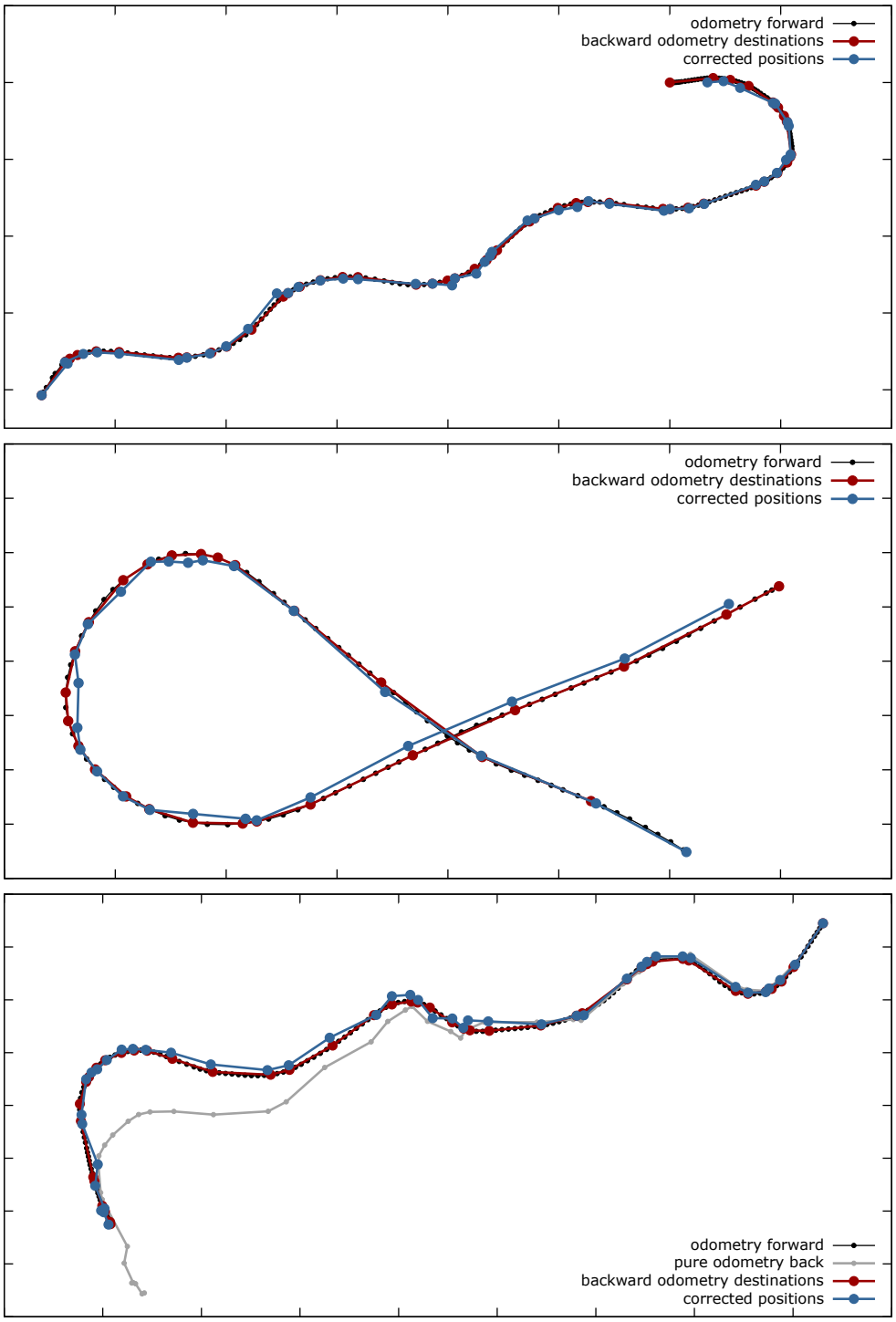


Figure 4.22: Three experiments performed in different settings. The meaning of the lines is the same as in Figure 4.21. The average deviation from the original trajectory is between $4\,\mathrm{cm} (top dataset) and 6\,cm$ (middle dataset).

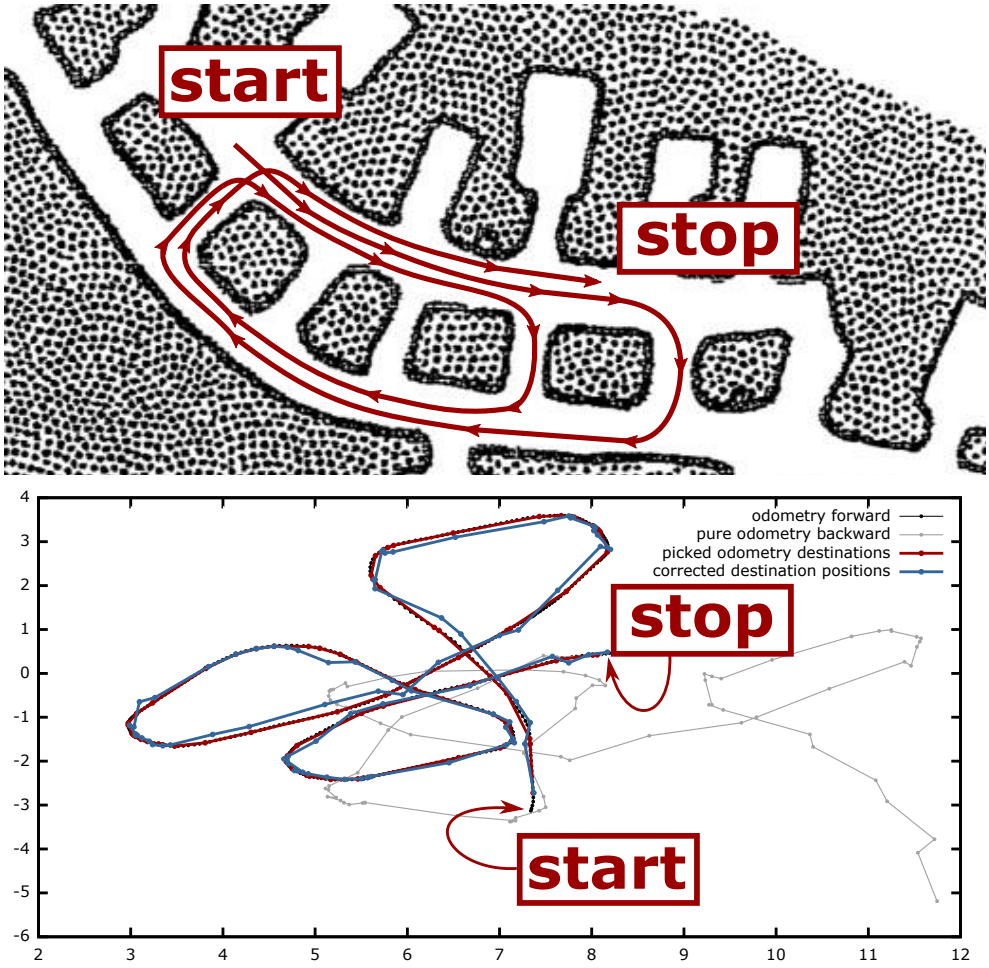


Figure 4.23: An experiment in a man-made cave in Niederzissen near Bonn that consists of long tunnels and several small room-like structures. The narrow passages allow for approx. 10 cm margin for fitting a robot. The schematic drawing (top image) shows the approximate robot path drawn on top of a map. The bottom image shows the actual movement of the robot in the odometry frame while being steered from "start" to "stop" (black lines) as well as the waypoints the robot has chosen returning from "stop" to "start". The blue lines show the positions of the robot reported after ICP registration. The gray lines show pure odometry while performing homing. As can be seen by the distance between the odometry readings while exploring and while performing homing our method is essential for safe trajectory rewinding.

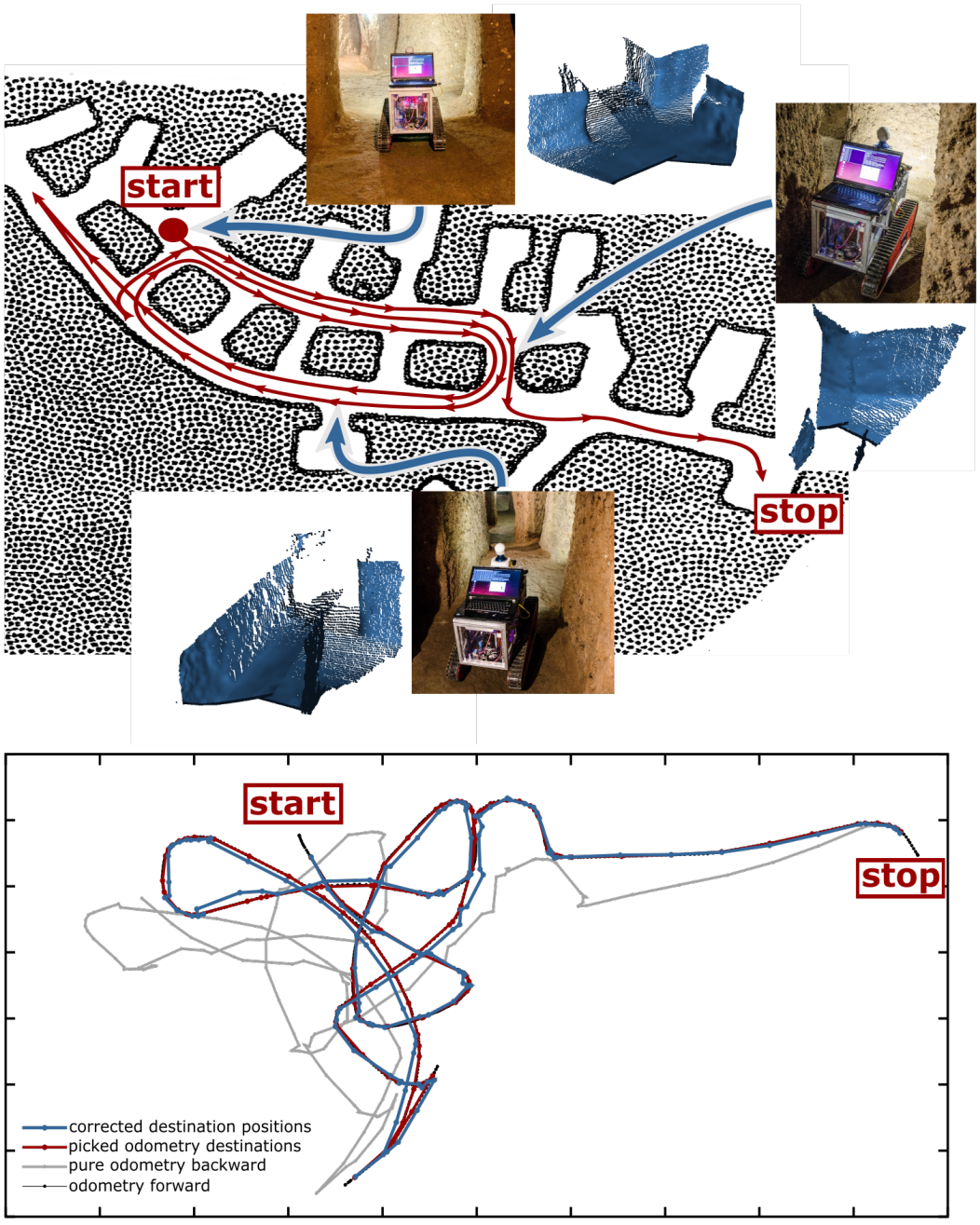


Figure 4.24: Second experiment the cave as in Figure 4.23. The robot was steered from "start" to "stop" and has performed homing autonomously from "stop" to "start". In the left corner of the top picture, the robot was steered forward and then backwards. It has repeated the same path during the homing route as can be seen in the bottom of the second image. The length of the trajectory in this experiment is approximately 70 meters.

precisely, the robot can return. We illustrated here that even when there is a substantial amount of track slippage on the ground, our method is still capable of robustly handling corridor-like environments with multiple narrow passages such as the doorways or narrow winding corridors.

In addition to the presented qualitative and quantitative evaluations, we have tested and shown the performance of the robust homing system presented in this section at multiple review meetings during the ROVINA project in the real Roman catacombs, and have proven it capable of safely returning the robot back to the starting position without relying on a consistent map.

### 4.3.7 Related work

A key problem in autonomous exploration, is that in case of a SLAM failure, the map becomes inconsistent. This can prevent the robot from continuing its exploration mission and—even worse—from being able to navigate back. It is therefore important to be able to perform reliable navigation without relying on a map. We are, however, not aware of any robotic navigation system that monitors the map to detect a SLAM failures to trigger a homing action if the map becomes inconsistent.

Rewinding a trajectory is related to teach-and-repeat navigation systems. One of important outposts of teach-and-repeat is visual teach-and-repeat. Works of Furgale *et al.* [40, 41], Nitsche *et al.* [90] and Ostafew *et al.* [93] show that this approach is indeed suitable to steer a robot in complicated environments based solely on visual sensors without relying on a consistent map. These visual methods, however, need substantial adaptation in order to be used in a setup similar to ours: using monocular cameras to localize through feature detection relies on having enough visual information, which ancient catacombs typically do not possess.

Teach-and-repeat approaches have also been used in LiDAR-based settings. Sprunk *et al.* [116] present an approach that relies on precise localization of the robot based on LiDAR measurements recorded during a trajectory demonstration. Similarly, Furgale *et al.* [42] perform an ICP-based teach-and-repeat approach on an autonomous robot equipped with a high precision 3D spinning LiDAR. They extend the standard teach-and-repeat approach by adding a local motion planner to account for dynamic changes in the environment. Our method to rewind the trajectory is similar to the teach-and-repeat setup in this formulation. However, in contrast to the mentioned methods, we use a substantially less accurate robot and thus have to cope with somewhat larger deviation from the reference trajectory. Further, we are rewinding a trajectory, not repeating it, and the sensor suite also differs significantly: Sprunk *et al.* use a high-precision 2D SICK laser range finder, which allows for accurate position correction. Furgale *et al.* also rely

on a precise LiDAR sensor which is able to provide them with more information than our RGBD camera setup, thus simplifying scene matching.

### 4.3.8 Conclusion

The ability to robustly operate without user intervention is an important capability for exploration robots and safely returning after the mission is a key requirement in real-world settings. In this section, we presented a complete homing system that addresses the problem of returning a robot operating in an unknown environment to its starting position even if the underlying SLAM system fails.

Despite our best efforts in steering the robot with active loop closures in mind, as shown in Section 4.2, there is still no system that can *guarantee* a consistent map during the whole navigation procedure. Therefore, we have combined a statistical map consistency test with a scan alignment approach to rewind a previously taken trajectory in case the map constructed by the robot is broken. The approach can be executed online on a notebook computer with two Asus Xtion RGBD cameras recording 3D point cloud information. We implemented our approach in ROS and executed it on a real autonomous robot designed to explore and map hard-to-access underground environments. We evaluated the consistency check in our lab environment as well as on real-world data acquired in the Priscilla catacomb in Rome and in a cave in Nierderzissen near Bonn. The evaluation suggests that our approach is well suited for homing in situations in which the mapping system failed. We illustrated that our proposed method can accurately rewind trajectories guiding the robot back to its starting location. It does so in challenging real-world settings in which the robot has to navigate through narrow passages and over uneven ground. In conjunction with the methods described in Section 4.2 our robust homing system is a valuable tool to allow for safe autonomous exploration using a mobile robot.

# Part II

# Towards dynamic environments

# Chapter 5

# Detecting motion

Until this point in this thesis, we have focused on navigation and perception in a static world, i.e., we assumed that no changes take place in the environment surrounding the robot. This allows us to simplify many parts of the navigation pipeline. For example, once we have analyzed the current robot surroundings for traversability using the approach presented in Section 4.1, the robot is safe to navigate in such an environment. If we drop the assumption that the world is static this is not true anymore. What is perceived as traversable in one moment can be occupied by a moving object in the next one. This makes reasoning about the environment as well as selecting actions for the robot in this environment more difficult.

The same holds for map building. If the world is static, the point clouds taken from the same position should always match perfectly, up to the precision of the used sensor. Likewise, if the robot is moving through such an environment and is performing incremental matching to estimate its own pose, the discrepancy between the consequent point clouds is caused exclusively by the ego-motion of the robot. In the dynamic world, this discrepancy can be caused by the changing environment, which might cause localization and mapping errors if the change is substantial.

In reality, the environments often are not static. Therefore, there is a need for methods that deal with the complexity that arises from the potential dynamic objects populating the scene. There are different approaches to deal with that. We believe that the first step towards safe navigation is to cluster the world into meaningful objects. Although, the robot has no knowledge if the objects can move or are static, the main purpose is to simplify the reasoning about the surroundings in terms of separate objects. We present our novel approach for efficient and robust clustering of such data in Section 5.1.

Once the objects are clustered, we can track them to determine if they are part of the static environment or are moving objects. A common approach is

to put bounding boxes around these objects and use a tracking approach such as an extended Kalman filter to track their movement. In conjunction with the ego-motion estimated through incremental scan matching, the objects can be categorized into moving and static ones. With this information at hand, we can ignore the moving objects to improve the scan matching procedure as well as estimating the movement of the dynamic objects to aid the planning step.

Tracking objects is a well studied problem and not the focus of this thesis. However, matching segmented objects based solely on the bounding boxes is error-prone due to overlaps and occlusions, and we present an approach that helps the robot to better match clustered objects. We propose a novel method for matching point clouds that correspond to different objects in order to determine if any particular object has to be tracked or is occluded in the next frame. The core idea behind this method is to represent objects as their depth map. This is possible because every object is part of a single depth image. The depth map representation allows us to use additional information about every object, such as the free space surrounding them, to improve matching. This free space information can be exploited in the matching step to produce a single probabilistic measure of how well the individual objects match. We present this approach in detail in Section 5.2. Our measure is fast to compute and makes it easier to determine if any two point clouds representing objects should be thought of as a single object or not.

In the remainder of this chapter we present our method for robust ground detection from LiDAR scans, along with our fast clustering method to separate a scene into meaningful objects in Section 5.1, and a robust measure that can be used as a validation step on top of any ICP algorithm, while tracking objects in the vicinity of the robot in Section 5.2.

## 5.1 Ground estimation and scene clustering

Image segmentation in RGB and multi-spectral data is a common problem in photogrammetric image analysis, computer vision, and remote sensing. Separating individual objects in 3D laser range data is also an important task for autonomous navigation of mobile robots or instrumented cars. An autonomous vehicle that is navigating in an unknown environment faces the complicated task of reasoning about its surroundings, see [45, 55, 58, 74, 122, 127, 130, 138]. There might be objects that constrain the possible actions of the robot or that may interfere with the robot's own plans. Thus, the interpretation of the robot's surroundings is key for robust operation and is probably one of the most complex problems in building autonomous vehicles today. While some approaches focus on finding specific objects in a dynamic scene [52, 75, 81], most perception pipelines per-

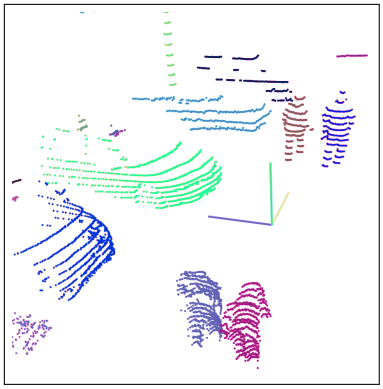

Figure 5.1: *Left:* Segmentation of objects such as people, cars, and trees generated from sparse 3D range data recorded with Velodyne VLP-16 scanner. Colors correspond to different segments. *Right:* Clearpath Husky robot used for the experiments driving at the University of Bonn campus.

form a segmentation of the environment into individual objects before a further interpretation is performed. Therefore, there is the need for an efficient online segmentation approach for 3D range data as this allows the robot to directly react to individual objects in its surroundings. This segmentation must be available in real time, as the system needs to reason about what it sees at the moment when the data become available in order to react appropriately.

Object segmentation from raw sensor data is especially relevant when operating in dynamic environments. In busy streets with cars and pedestrians, for example, the maps can be influenced by wrong data associations caused by the dynamic nature of the environment. A key step to enable a better reasoning about such objects and to potentially neglect dynamic objects during scan registration and mapping is the segmentation of the 3D range data into different objects so that they can be tracked separately [27].

Besides rather expensive terrestrial laser scanners, there are also less accurate and cheaper scanners targeted at mobile robotics applications. One example is the 16-beam LiDAR by Velodyne, which is becoming increasingly popular and can be installed on relatively low-cost platforms. If we compare the data provided by the 16-beam LiDAR with those provided by the 64-beam variant or even terrestrial scanners, we observe a substantial drop in the vertical angular resolution, i.e., the 16-beam LiDAR has a vertical angular resolution of 2 degrees. This poses several challenges to a segmentation algorithm operating on such 3D data. Sparser point clouds lead to an increase in Euclidean distance between neighboring 3D points, even if they stem from the same object. Thus, such sparse 3D points render it more difficult to reason about segments. The situation becomes even more complex with the increase in distance between the object and the sensor.

The contribution outlined in this section is a fast and effective method for separating ground from the rest of the scene and a fast and effective segmentation approach for 3D range data obtained from modern laser range finders such as Velodyne scanners. To achieve the final segmentation, we first perform robust ground separation, which can detect the surface the robot is operating on in a fast and reliable manner. In contrast to several other approaches, the ground can have slight curvature and does not necessarily have to be entirely flat. We also do not use any kind of sub-sampling and decide for each pixel of the range image whether it belongs to ground or not. An example of our segmentation with ground removed is depicted in Figure 5.1, where people and cars are correctly segmented using data from a Velodyne VLP-16 scanner.

Our segmentation method provides meaningful segmentations and runs multiple times faster than the acquisition of the scan. Even on a mobile CPU, we can process the scans of a Velodyne with over 70 Hz (64 beam scanner) or 250 Hz (16 beam scanner) and thus faster than the scans are acquired by the sensor, which acquired data at 10 Hz. We achieve this by performing all computations on a cylindrical range image. This method is advantageous, as the range image is often small, dense, and maintains the neighborhood information implicitly. Moreover, our approach is suited for scanners that provide comparably sparse point clouds as these clouds can still be represented as a dense range image. In addition to that, we made our approach available as open source software at:

`https://github.com/PRBonn/depth_clustering`

### 5.1.1 Range image based ground removal

Before performing object segmentation, we remove the ground from the scan. A standard approach to ground removal simply discards all 3D points that are lower than the vehicle (assuming we know where the sensor has been mounted on the mobile base/robot). While this approach may work in simple scenes, it fails if the vehicle's pitch or roll angle is unequal to zero or if the ground is not a perfect plane. Using RANSAC-based plane fitting improves the situation but even using this strategy, non-zero curvatures remain a challenge, and the operation can be time consuming. Thus, we take a different approach.

Most laser range scanners provide raw data in the form of individual range readings per laser beam with a time stamp and an orientation of the beam. This allows us to directly convert the data into a range image. The number of rows in the image is defined by the number of beams in the vertical direction, i.e., 16, 32 or 64 for the Velodyne scanners. The number of columns is given by the number of range readings per 360° revolution of the laser. Each pixel of such a virtual image stores the measured distance from the sensor to the object. To

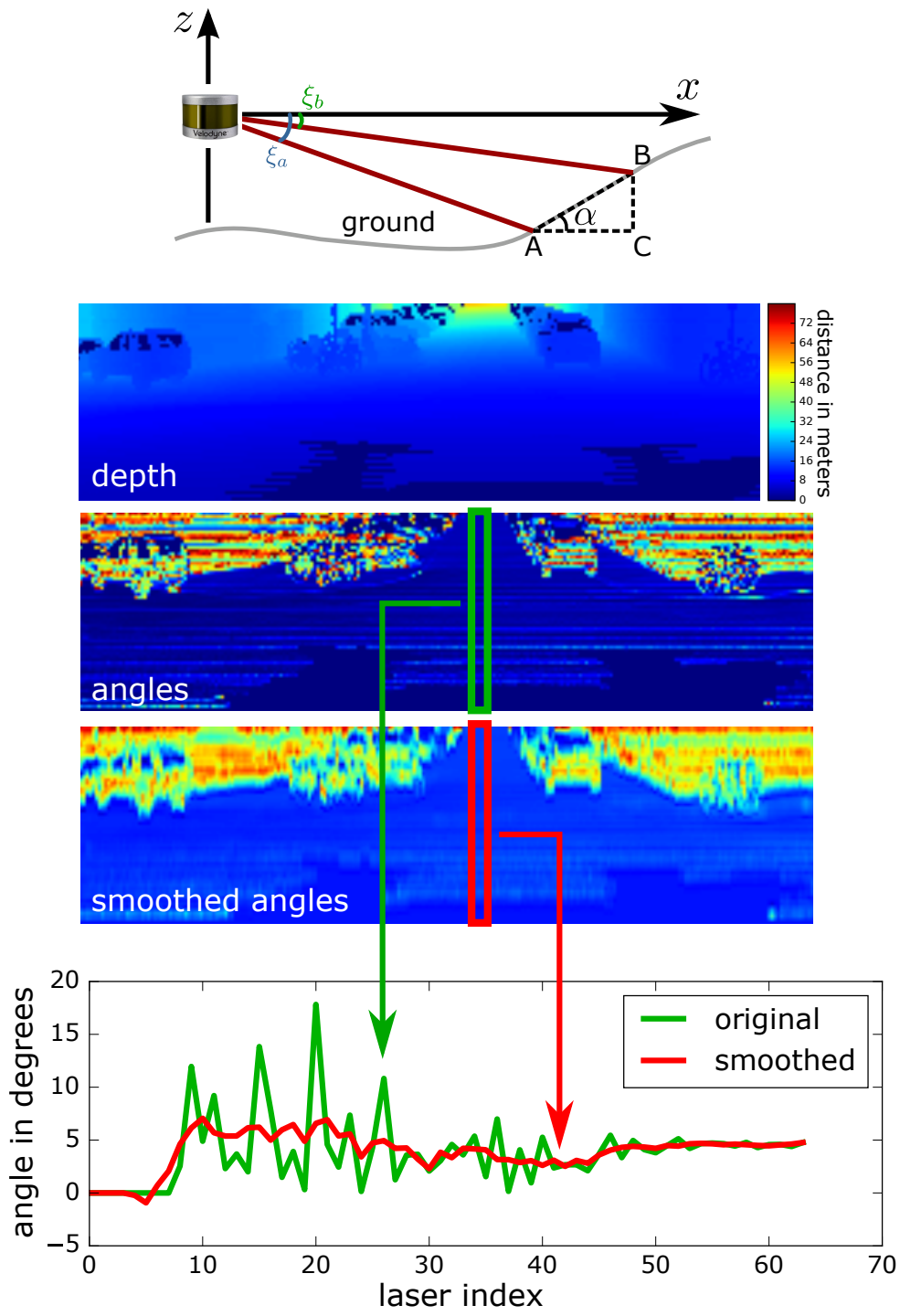


Figure 5.2: *Top:* An illustration of $\alpha$ angle. A Velodyne LiDAR on the left of this picture shoots two beams that form a triangle with the angle that we are interested in. *Middle:* Three images representing the depth, i.e. part of the original range image generated from data acquired from a Velodyne, computed $\alpha$ angles and the same angles after applying Savitsky-Golay smoothing to them. *Bottom:* illustration of the smoothing for a column of $\alpha$ angles as marked in the left image.

speed up computations, one may even consider to combine multiple readings in the horizontal direction into one pixel if needed.

In our implementation, we use range images and construct them directly from the raw measurements of the laser scanner, not computing the 3D point cloud. In case, however, a different laser scanner or a different device driver is used that only provides a 3D point cloud per revolution and not the individual range measurements, one can project the 3D point cloud onto a cylindrical image, compute the Euclidean distance per pixel, and proceed with our approach. This will increase the computational demands by approximately a factor of 2 for the whole approach, but still allows for a comparably fast segmentation.

For identifying the ground plane, we make three assumptions. First, we assume that the sensor is mounted roughly horizontally on the mobile base/robot. Second, we assume that the curvature of the ground is low. Third, we assume that the robot observes the ground plane at least in some pixels of the lowest row of the range image (corresponding to the laser beam scans hitting the ground close to the robot).

With these assumptions in place we start by turning each column $c$ of the range image $\mathfrak{I}$ into a stack of angles $\alpha^r_{r-1,c}$, where each of these angles represents the angle of inclination of a line connecting two points A and B derived from two range readings $\mathfrak{I}_{r-1,c}$ and $\mathfrak{I}_{r,c}$ in neighboring rows $r-1$, $r$ of the range image, respectively, as depicted in the top right part of Figure 5.2. Knowing two range readings of vertically consecutive individual laser beams, we can compute the angle $\alpha$ using trigonometric rules as follows:

$$\begin{aligned} \alpha &= \text{atan2}(\|BC\|, \|AC\|) \\ &= \text{atan2}(\Delta z, \Delta x), \\ \Delta z &= |\mathfrak{I}_{r-1,c} \sin \xi_a - \mathfrak{I}_{r,c} \sin \xi_b|, \\ \Delta x &= |\mathfrak{I}_{r-1,c} \cos \xi_a - \mathfrak{I}_{r,c} \cos \xi_b|, \end{aligned} \tag{5.1}$$

where $\xi_a$ and $\xi_b$ are vertical angles of the laser beams corresponding to rows $r-1$ and $r$.

Note that we need two range readings for each $\alpha$ computation and so the size of the stack of $\alpha$ angles has size one less than the number of rows in the range image. We treat all stacks of these angles as a matrix $\mathbf{M}_\alpha = [\alpha^r_{r-1,c}]$, where $r$ and $c$ are row and column coordinates of the corresponding range readings from the range image.

Unfortunately, LiDAR sensors such as the Velodyne HDL-64 produce a substantial amount of outliers in the range measurements, which is discussed in more details in the work of Leonard *et al.* [76], and this impacts the computation of the angle $\alpha$ in Figure 5.2. We therefore need a way to eliminate such outliers. Weinmann and Jutzi [131] address this problem by computing features over a

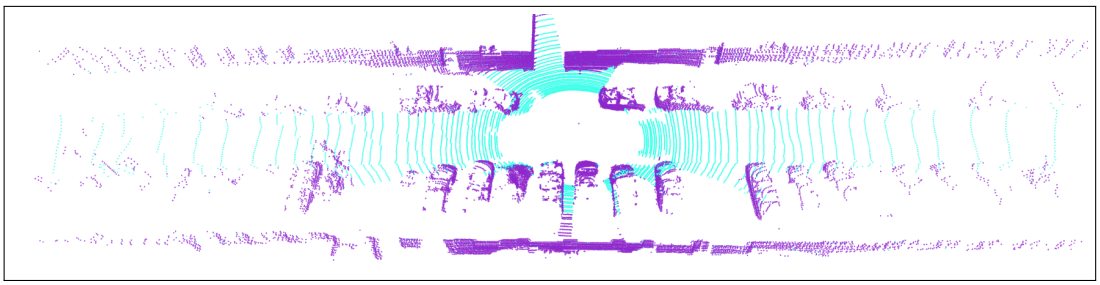

Figure 5.3: An example scene taken with a 64-beam Velodyne LiDAR seen from above with ground marked light blue.

small local neighborhood of every pixel of a range image to detect if a reading can be treated as reliable or not. This approach filters out unreliable readings but also the points on the borders of the objects. As these points are important for performing segmentation, we instead compute the corresponding angles from all available data points and smooth the computed angles afterwards. To achieve such smoothing, we apply the Savitsky-Golay filter to every column of $\mathbf{M}_\alpha$. This filter performs least-squares optimization to fit a local polynomial for a given window size to the data. In their work, Savitzky and Golay [106] show that one can avoid the explicit least squared fitting of the polynomials and compute an effective approximation relying on precomputed coefficients, which allows for greater computational efficiency.

We carry out the ground labeling on the matrix $\mathbf{M}_\alpha$ after applying the Savitsky-Golay filter to its columns starting with the entries that we expect to belong to the ground and labeling similar components together using breadth-first search. Breadth-first search (BFS) is a popular graph search or traversal algorithm. It starts at a given node of the graph and explores the directly neighboring nodes first, before moving to the next level of neighbors. In our approach, we consider the difference in the calculated angles $\alpha$ over an N4 neighborhood on a grid to decide if two neighboring elements of the matrix $\mathbf{M}_\alpha$ should be labeled together by the breadth-first search. For that purpose we select a threshold $\Delta\alpha$, set to 5 degrees in our experiments.

We start by labeling each element of the lowest row as ground if the corresponding $\alpha^1_{0,c}$ is smaller than a pre-defined angle (45 degrees in our current implementation), i.e., we do not label any almost vertical objects such as walls. Let the set $\mathbf{G}$ be a set of all column indices in the first row that we have labeled as ground.

For every $c \in \mathbf{G}$ we label the connected component using BFS starting from $\alpha^1_{0,c}$ as ground as depicted in procedure `LabelGround` in Algorithm 3. By the time we have processed all $c \in \mathbf{G}$, all the ground pixels in the image have been labeled as such. Figure 5.3 shows an example point cloud with the ground detected by our algorithm marked in light blue.

**Algorithm 3** Ground labeling

```
1: procedure LabelGround(R)
2:     M ← [α^r_{r−1,c}], matrix of angles α computed with Equation (5.1).
3:     for c = 1 ... 𝔍_cols do
4:         if M(0, c) not labeled then
5:             LabelGroundBFS(0, c);
6: procedure LabelGroundBFS(r, c)
7:     queue.push({r, c})
8:     while queue is not empty do
9:         {r, c} ← queue.top()
10:        {r, c} ←labeled as ground
11:        for {r_n, c_n} ∈ neighborhood{r, c} do
12:            if |M(r, c) − M(r_n, c_n)| < 5° then
13:                queue.push({r_n, c_n})
14:        queue.pop()
```

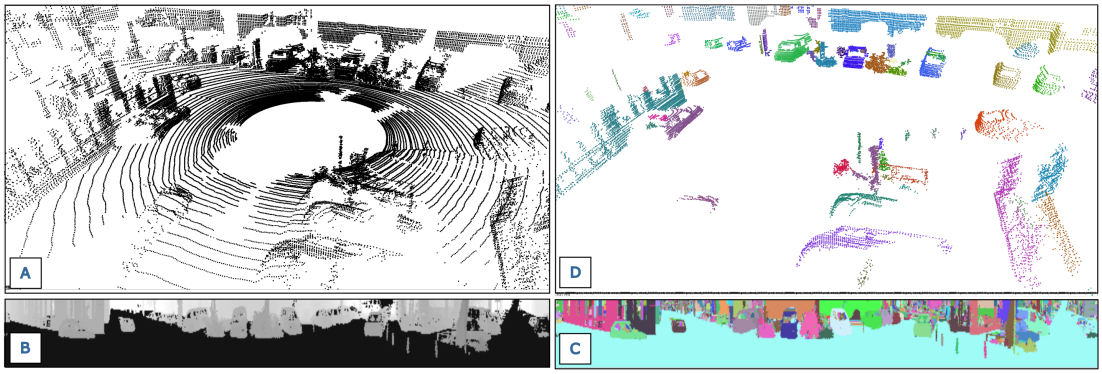


Figure 5.4: Illustration of our method. *A:* Point cloud from Velodyne, which is shown for illustration reasons only. *B:* We build up a range image not considering points lying on the ground plane and *C:* perform the segmentation in the range image directly. *D:* This allows us to provide individual small point clouds for the different segments. The different objects are shown with random colors. Range and label images are scaled for better visibility.

### 5.1.2 Fast and effective segmentation on laser range data

The vertical resolution of the sensors which differs between 16, 32, or 64 beams variants has an impact on the difficulty of the segmentation problem. For every pair of neighboring points, one basically has to decide if the laser beams have been reflected by the same object or not. When reasoning in 3D, this decision turns more complex with the growth of the Euclidean distance between the points.

In our approach, which is outlined in Figure 5.4, we avoid the explicit creation of the 3D point cloud and perform our computations using a laser range image, in our case a cylindrical one for the Velodyne scanners. This has two advantages.

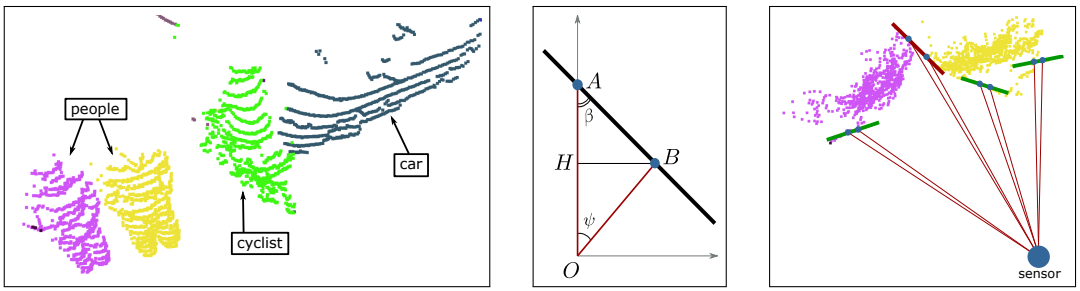


Figure 5.5: *Left:* example scene with two pedestrians, a cyclist and a car. *Middle:* Given that the sensor is in $O$ and the lines $OA$ and $OB$ represent two laser beams, the points $A$ and $B$ spawn a line that estimates the surface of an object should they both belong to the same object. We make the decision about this fact based on the angle $\beta$. If $\beta > \theta$, where $\theta$ is a predefined threshold, we consider the points to represent one object. *Right:* a top view on the pedestrians from the example scene. The green lines represent points with $\beta > \theta$ while the red one shows an angle that falls under the threshold and thus labels objects as different.

First, we can exploit the clearly defined neighborhood relations directly in the range image, which makes the segmentation problem easier. Second, we avoid the generation of the 3D point cloud, which makes the overall approach faster to compute.

We assume that the vehicle moves on the ground (see Figure 5.1 for our setup) and we expect the sensor to be oriented roughly horizontally with respect to the wheels. We obtain an estimate of the ground plane by analyzing the columns of such a range image, as described in Section 5.1.1. This allows us to remove the ground from the range image, see Figure 5.4 (B).

The key component of our approach is the ability to estimate which measured points originate from the same object for any two laser beams. We explicitly avoid feature computation and work with raw sensor data, taking a decision for each point of the 3D range data.

We present an easy-to-implement and fast-to-compute, but yet, effective approach to find the components that belong to one object. To answer the question if two laser measurements belong to the same object, we use an angle-based measure, which is illustrated in Figure 5.5 and is described in the following paragraphs.

The left image of Figure 5.5 shows an example scene with two people walking close to each other in front of a cyclist, who passes between them and a parked car. This scene has been recorded using our Velodyne VLP-16 scanner. The middle image shows an illustration of two arbitrary points $A$ and $B$ measured from the scanner located at $O$ with the illustrated laser beams $OA$ and $OB$. Without loss of generality, we assume the coordinates of $A$ and $B$ to be in a coordinate system which is centered in $O$ and the $y$-axis to be oriented along the longer of two laser beams. We define the angle $\beta$ as the angle between the laser beam and the line

connecting $A$ and $B$ in the point that is further away from the scanner (in our example that is $A$). In practice, the angle $\beta$ turns out to be a valuable piece of information to determine if the points $A$ and $B$ lie on the same object or not.

Given the nature of the laser range measurements, we know the distance $\|OA\|$ as it corresponds to the first laser measurement as well as $\|OB\|$ (second laser measurement). We will call these range measurements $d_1$ and $d_2$ respectively. One can use this information to calculate $\beta$ by applying trigonometric equations

$$\beta = \text{atan2}(\|BH\|, \|HA\|) = \text{atan2}(d_2 \sin\psi,\ d_1 - d_2 \cos\psi), \tag{5.2}$$

where $\psi$ is the known angle between the beams and is usually provided in the documentation of the scanner. The middle image in Figure 5.5 illustrates the computation in the $xy$-plane from a top-down view of the scene. Note that we can compute the angle $\beta$ for pairs of points $A$ and $B$ that are neighbors either in row or in column direction in the range image. In the first case, the angle $\psi$ corresponds to the angular increment in row direction, in the other case to the increment in column direction.

The intuition behind the angle $\beta$ is that it stays relatively large for most objects and only takes small values if the depth difference between neighboring points given the range image is substantially larger than their displacement in the image plane, that is defined through the angular resolution of the scanner. This insight allows us to define a parameter $\theta$ that acts as a threshold on the angle $\beta$. This threshold enables us to take a decision about whether to separate any two points in the range image into separate clusters or merge them into one. If $\beta$ is smaller than the user-defined value $\theta$, we argue that the change in depth is too large and take the decision to separate the points into different segments. Otherwise, the points are considered as lying on the same object.

A threshold-based criterion on $\beta$ is clearly a heuristic but works well in practice as we illustrate in the experimental evaluation. A failure case can be a situation in which the scanned object is planar, such as a wall, and oriented nearly parallel to the laser beams. In this case the angle $\beta$ will be small and it is therefore likely for the object to be split up in multiple segments. This essentially means that if $\beta$ is smaller than $\theta$, it is difficult to find out if two points originate on two different objects or just lie on a planar object nearly parallel to the beam direction. However, despite this shortcoming, our experiments suggest that the method is still useful in practice. The aforementioned behavior occurs rarely and if so, it usually results only in an over-segmentation of particularly inclined planar objects.

With the separating threshold in mind, we approach the segmentation directly in the range image. We regard two endpoints as being neighbors stemming from the same object if they are neighbors in a the range image (we use an N4

**Algorithm 4** Range image labeling

```
1:  procedure LabelRangeImage(ℑ)
2:      Label ← 1, L ← zeros(ℑ_rows × ℑ_cols)
3:      for r = 1 ... ℑ_rows do
4:          for c = 1 ... ℑ_cols do
5:              if L(r, c) = 0 then
6:                  LabelComponentBFS(r, c, Label);
7:                  Label ← Label + 1;
8:  procedure LabelComponentBFS(r, c, Label)
9:      queue.push({r, c})
10:     while queue is not empty do
11:         {r, c} ← queue.top()
12:         L(r, c) ← Label
13:         for {r_n, c_n} ∈ neighborhood{r, c} do
14:             d_1 ← max(ℑ(r, c), ℑ(r_n, c_n))
15:             d_2 ← min(ℑ(r, c), ℑ(r_n, c_n))
16:             if atan2 (d_2 sin ψ)/(d_1 − d_2 cos ψ) > θ then
17:                 queue.push({r_n, c_n})
18:         queue.pop()
```

neighborhood on the grid) and the angle $\beta$ between them is larger than $\theta$. Given this definition of a neighborhood, we can view the segmentation problem as the problem of finding the connected 2D components exploiting the structure of the range image and the constraint on $\beta$.

Algorithm 4 depicts the algorithm that we use to find the connected components that define the segments. We use a variant of a pass-through filter with complexity $\mathcal{O}(N)$, where $N$ is the number of pixels, i.e., the number of range readings per scan. The algorithm guarantees visiting each point in the range image at maximum twice. Please note that at this point in time, all pixels of the range image that were labeled as ground (see Section 5.1.1) are set to zero and do not take part in the following procedure.

We start in the top left corner of the range image and pass through every pixel from top to bottom, left to right (line 3–4). Whenever we encounter an unlabeled pixel (line 5), we start a breadth-first search from this pixel (line 6). The goal of this search is to label every pixel of this component. For this purpose, the BFS uses a queue and an N4 neighborhood consisting of the left, right, lower and top pixels (line 13). The decision if a point in the N4 neighborhood should be added to the queue of the BFS is taken based on the angle $\beta$ generated by the neighbor and the current point (line 14–17). This procedure guarantees that the

whole connected component will receive the same label. Once the queue of BFS is empty, we continue to traverse the range image sequentially until we reach a new unlabeled point.

It has to be noted that the connected components algorithm is not the main contribution of this work but its effective application to the segmentation of range images considering the value of $\beta$ for two neighboring measurements. For more information on the comparison between different implementations of connected components algorithms, we refer the reader to Cabaret *et al.* [19]. Overall, our approach yields an easy-to-implement and fast method that does not require a lot of parameters tuning to achieve good segmentation performance.

### 5.1.3 Experimental evaluation

Our experiments are designed to show the capabilities of our method and to support our key claims, which are: (i) all computation can be executed fast, even on a single core of a mobile CPU with around 70 Hz, (ii) we can segment typical 3D range data obtained by mobile robots into meaningful segments, and (iii) the approach performs well on sparse data such as those obtained from a 16-beam Velodyne Puck scanner. In our evaluation, we also provide comparisons to the grid-based segmentation method proposed by Teichman and Thrun [127] as used by Behley *et al.* [6] as well as to Euclidean clustering implemented in the point cloud library PCL and to range-based segmentation with a simple range threshold. Throughout all experiments, we used our default parameter $\theta = 10°$. The choice of this default parameter value is described later in this chapter in Figure 5.8.

#### 5.1.3.1 Runtime

The first experiment is designed to support the claim that our approach can be executed fast, supporting online processing in real time. We therefore tested our approach on point clouds computed with different Velodyne laser scanners and processed the data on different computers. On the robot, we use an Acer notebook with an Intel i5 5200U 2.2 GHz CPU, and we also process the data on a desktop computer with an Intel i7 4770K 3.5 GHz CPU, in both cases using only one core of the CPU.

Table 5.1 summarizes the runtime results for nearly 2,500 point clouds (each generated by a single revolution of the scanner) recorded in urban outdoor environments. The numbers support our first claim, namely that the computations can be executed fast and in an online fashion. The frame rate of our segmentation pipeline including ground removal is multiple times faster than the frame rate of the sensor. On a mobile i5 CPU, we achieve average frame rates of 74 Hz —

| scanner | segmentation only | | ground removal + segmentation | |
|---|---|---|---|---|
| | mobile<br>i5 U5200 2.2 GHz | desktop<br>i7 4770K, 3.5 GHz | mobile<br>i5 U5200 2.2 GHz | desktop<br>i7 4770K 3.5 GHz |
| 64-beam | 8.6 ms ± 2.6 ms<br>116 Hz | 4.7 ms ± 1.2 ms<br>212 Hz | 13.3 ms ± 1.0 ms<br>74 Hz | 8.6 ms ± 0.6 ms<br>116 Hz |
| 32-beam | 4.4 ms ± 1.2 ms<br>227 Hz | 2.6 ms ± 0.5 ms<br>385 Hz | 8.3 ms ± 0.7 ms<br>120 Hz | 4.5 ms ± 0.7 ms<br>222 Hz |
| 16-beam | 2.4 ms ± 0.5 ms<br>416 Hz | 1.5 ms ± 0.2 ms<br>667 Hz | 4.0 ms ± 0.8 ms<br>250 Hz | 2.8 ms ± 1.0 ms<br>354 Hz |

Table 5.1: Average runtime and standard deviation per 360° laser scan.

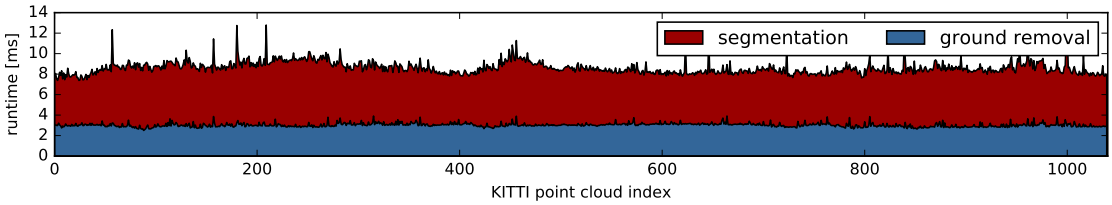


Figure 5.6: Timings for ground removal and clustering obtained on the KITTI dataset. The x-axis depicts the index of individual point clouds while the y-axis shows the processing time in ms.

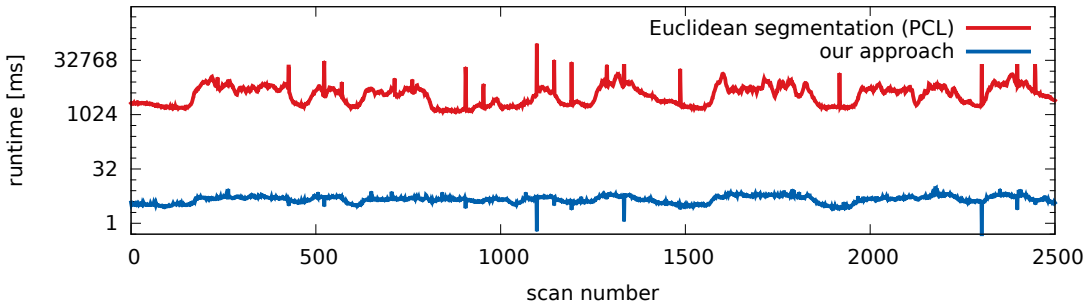


Figure 5.7: Timings for segmenting approximately 2,500 scans from a 64-beam Velodyne dataset with our approach and Euclidean segmentation from PCL (ground removal time not included here).

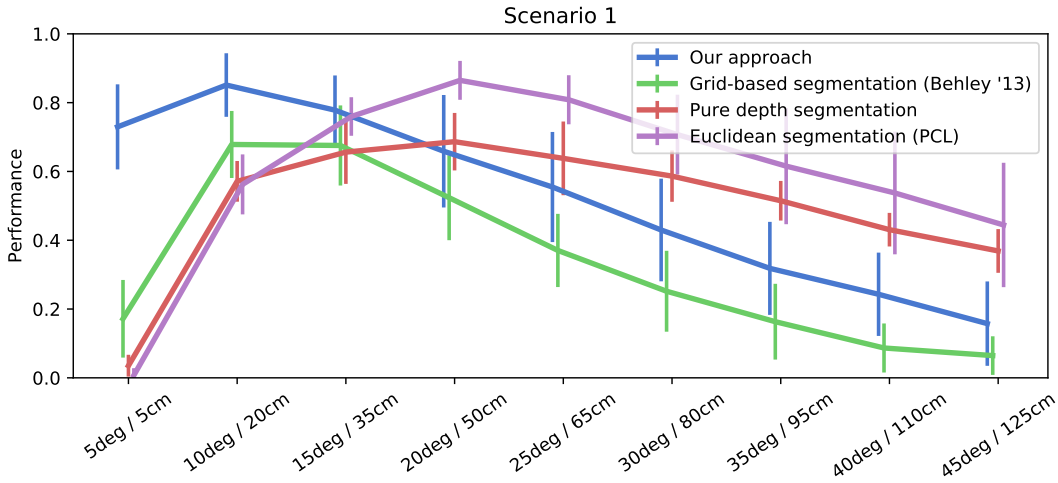


Figure 5.8: Performance of our algorithm computed as a fraction of the number of found objects over the number of all manually labeled objects in the scene compared to the grid-based segmentation by Behley *et al.* [6], segmentation through Euclidean clustering as provided by PCL and to segmentation on range images using depth difference threshold for varying parameters on 30 different, manually labeled point clouds from the datasets available from Frank Moosmann's website. On the x-axis, the first value is the parameter $\theta$ for our method and the second one serves as both the cell size for the grid-based approach and as the distance threshold for the Euclidean clustering and depth thresholding approaches.

250 Hz for the whole approach and 116 Hz — 354 Hz on an i7 computer. The pure segmentation without ground removal can run with up to 667 Hz. We obtained similar timings on the publicly available KITTI datasets [44], see Figure 5.6.

We also compared the speed of our segmentation pipeline to Euclidean clustering for segmentation as provided by the point cloud library PCL. For a fair comparison, we used the same ground removal for both approaches and thus the reported timing refers to the segmentation only. As can be seen from Figure 5.7, our approach is on average around 1,000 times faster than Euclidean clustering in the 3D space, here using 64-beam Velodyne data.

#### 5.1.3.2 Segmentation results

The next set of experiments is designed to illustrate the obtained segmentation results. We consider the results on 16-beams and 64-beams laser range data. For the 64-beam evaluation, we rely on the publicly available street scenes dataset by Moosman *et al.* [82] and the KITTI dataset [44], while we recorded the 16-beam datasets using our robot in Bonn, Germany, see also Figure 5.1.

We evaluate the performance of our method and compare it to a popular grid-based approach by Behley *et al.* [6], to a range image clustering method that uses pure depth as the cluster separation criterion, and to segmentation through Euclidean clustering as provided by PCL. For that purpose, we manu-

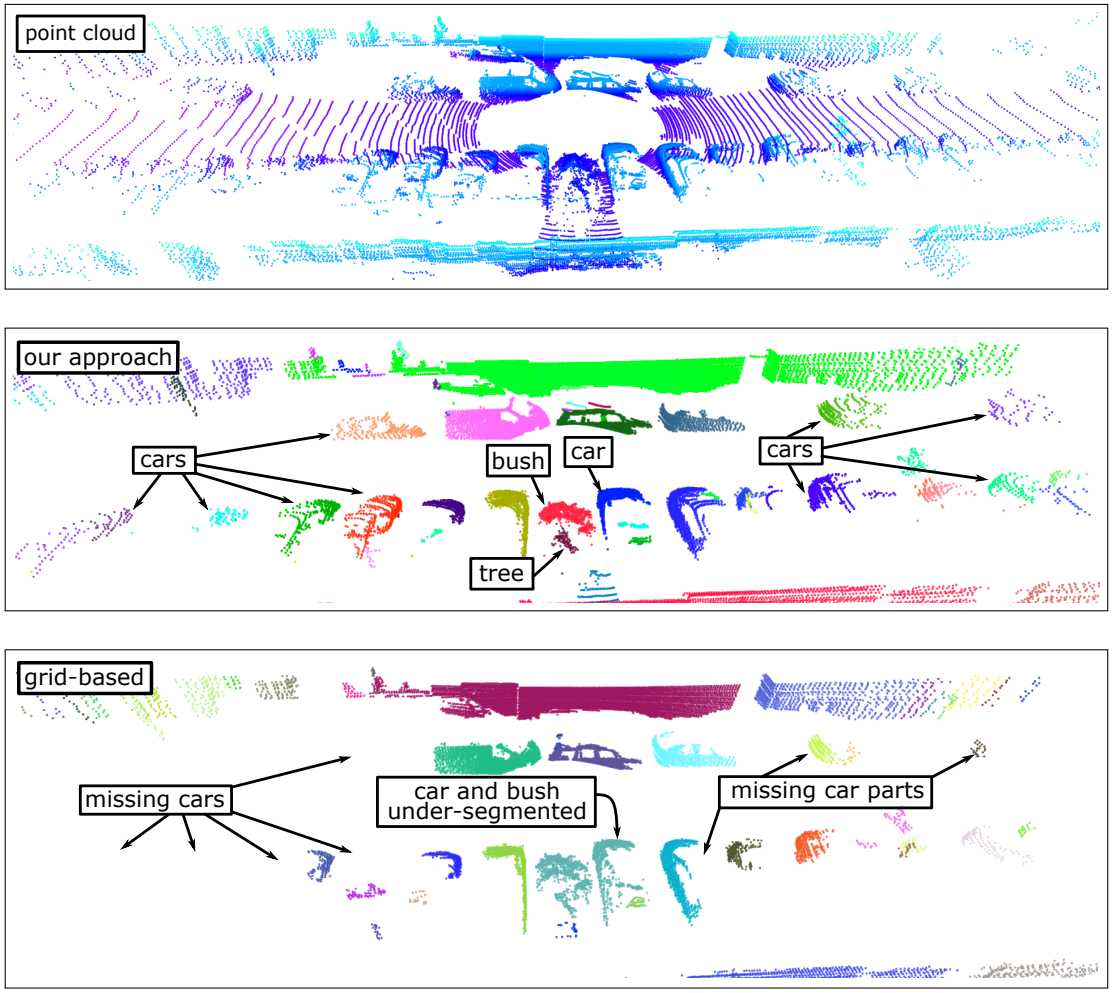


Figure 5.9: *Top:* Point cloud of an outdoor scene taken with a 64-beam Velodyne (shown for illustration only). *Middle:* Our segmentation that provides correct segmentation even for distant objects while not under-segmenting the close ones. *Bottom:* Segmentation provided by a grid-based approach with cell size set to 0.2 m. There is a number of cars that are situated further from the sensor missing and one car is merged with a bush.

ally segmented 30 point clouds from different scenes from two datasets provided by Moosman *et al.* [82] and ran all three methods on these data varying their parameters. For our method, we have chosen different values for $\theta$, while for the grid-based approach we have varied the size of the grid cells. For the pure depth thresholding method we varied the range threshold and for Euclidean clustering we have varied the distance between clusters that should be separated. We have chosen values for $\theta$ from 5° to 45° and for the grid cell resolution (grid-based) and the distance threshold (Euclidean and pure depth thresholding method) values between 0.05 m to 1.25 m. We have evaluated the performance of the algorithms by counting how many of the manually labeled objects have been found by the algorithms. For every ground truth cluster, we search for a found segment with the biggest overlap. We consider the cluster as correctly found if the point-wise


Figure 5.10: An example segmentation of a group of people from KITTI dataset. *Top:* an RGB image that shows the scene, shown for visualization purpose only. *Bottom:* the scene segmented with our algorithm. Different clusters are assigned random colors.

overlap is substantial (80% in our implementation). We then count the number of successful matches and divide them by the number of expected ground truth clusters. We compute the performance measure for every scan and present the mean and standard deviation of these values with relation to the chosen parameter in Figure 5.8. As can be seen with $\theta = 10°$, our method outperforms both the grid-based approach and the pure depth thresholding approach in terms of segmentation quality in all parameters settings. In comparison to Euclidean clustering, our approach shows a comparable performance on the 64-beam datasets, while being around three orders of magnitudes faster (4 ms vs. 4 s per scan). This nicely illustrates the benefits of our method for online processing. Typical examples of a resulting segmentation are shown in Figure 5.9 and Figure 5.10, both using a 64-beam Velodyne scanner.

Finally, we aim at supporting our claim that our segmentation pipeline can handle sparse data coming from a scanner with 16 beams in the vertical direction (Velodyne VLP-16) well. For this purpose, we analyzed the results using data recorded from our scanner and compared them to manually labeled ground truth clouds. Examples are depicted in Figure 5.11. Although this is only a qualitative evaluation, we can clearly see that our approach handles the sparse range data

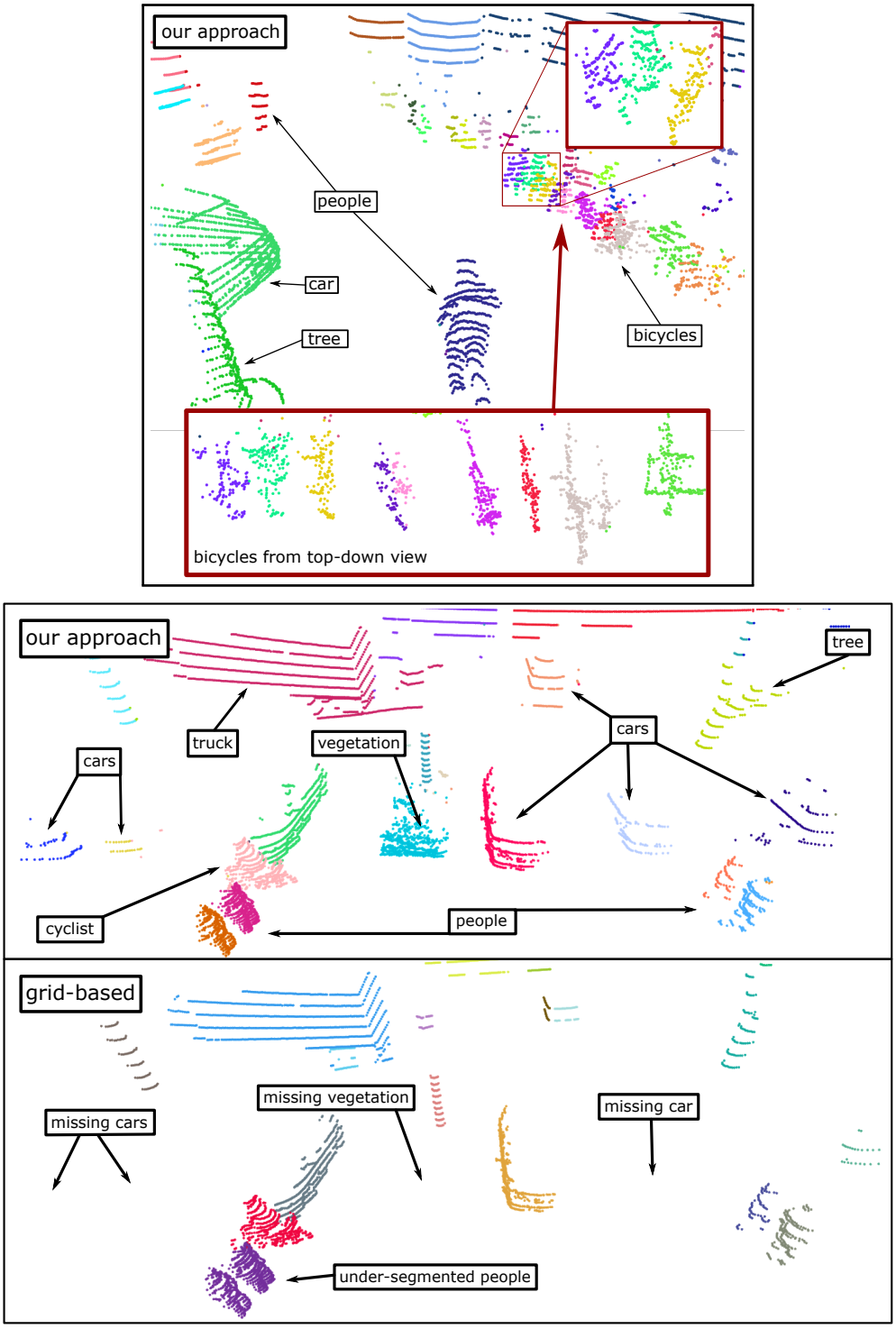


Figure 5.11: *Top:* An outdoor scene recorded with a 16 beam Velodyne that shows that our approach is able to segment even complicated scenes with multiple small objects like bicycles placed very close to each other. The grid-based approach in this scene merged all the bicycles into two big clusters. *Middle:* Our segmentation of an example outdoor scene taken with a 16-beam Velodyne. Our approach was able to find objects omitted by the grid-based method while correctly segmenting people that stand close to each other. *Bottom:* Grid-based segmentation result. Some objects are missing and people on the bottom left are under-segmented.

better than the approaches that work in the space of 3D points.

In summary, our evaluation suggests that our method provides competitive segmentation results compared to existing methods on dense range images and outperforms them on sparse scans. At the same time, our method is fast enough for online processing and has small computational demands. Thus, we supported all our claims with this experimental evaluation.

### 5.1.4 Related work

Segmenting objects from 3D point clouds is a relatively well-researched topic. There is substantial amount of work that targets acquiring a global point cloud and segmenting it off-line [1, 34, 45, 55, 130]. These segmentation methods have been used on a variety of different data produced by 3D range sensors or 2D lasers in push-broom mode. The photogrammetric society has also been active in the field of segmenting large point clouds into different objects. Velizhev *et al.* [129] focus on learning the classes of the objects and detecting them in huge point clouds via a voting-based method. These point clouds can be large, and the work by Hackel *et al.* [50] targets the runtime along with the quality of classification. In contrast with these works, we focus on the segmentation of range data that comes from a 3D laser scanner such as a Velodyne that provides a 360 degree field of view in a single scan and is used for online operation on a mobile robot. Additionally, we target segmentation of a scene without the knowledge about the objects in it, any prior learning, and not using complex features. For a comprehensive analysis of methods that perform supervised scene segmentation, we refer the reader to the work of Weinmann *et al.* [132].

Ground removal is an often used pre-processing step and is therefore well-discussed in the literature. There is a number of papers that use RANSAC for fitting a plane to the ground and removing points that are near this plane such as the work by Ošep *et al.* [94]. Another prominent method of ground detection is a side-product of full semantic segmentation of the scene, where all parts of the scene get a semantic label. The ground is then segmented as one class; for more details we refer the reader to the papers by Hermans *et al.* [57] and Bansal *et al.* [4]. A couple of approaches use a 2D-grid and analyze the heights of the points that fall into its bins, taking decisions about points being parts of the ground based on this information. The decisions can be taken based on the inclination of lines between consecutive cells [76, 97] or by analyzing the height above the lowest local point [6, 46].

Segmentation techniques for single scans without requiring additional information can be divided into three groups. The first group performs the segmentation in the 3D domain by defining sophisticated features that explain the data in 3D [31, 32] or by removing the ground plane and segmenting the clouds with a

variant of a nearest neighbor approach [22, 68]. Feature-based approaches, while allowing for accurate segmentation, are often comparably time-consuming and may limit the application for online applications to a robot with substantial computational resources. Our goal was to avoid computing complicated geometric features allowing for very fast execution times.

The second group focuses on projecting 3D points onto a 2D grid positioned on the ground plane. The segmentation is then carried out on occupied grid cells [6, 58, 71, 122]. These algorithms are fast and suitable to run online. Quite often, however, they have a slight tendency to under-segment the point clouds, i.e. multiple objects may be grouped as being one object if they are close to each other. This effect often depends on the choice of the grid discretization, so that the grid width may need to be tuned for individual environments. Additionally, some of these approaches can suffer from under-segmenting objects in the vertical direction.

The third group of approaches performs the segmentation on a range image and our approach belongs to this group of techniques. For example, Moosmann *et al.* [82, 83] use a range image to compute local convexities of the points in the point cloud. In contrast to that, our approach avoids computing complex features and, thus, is easier to implement, runs very fast and produces comparable results. We therefore believe that our approach is a valuable contribution to a vast and vibrant field of 3D point cloud segmentation, and consequently we contribute our approach to the open source ROS community by providing the source code for our implementation.

There are also several works that perform segmentation on RGBD data acquired from a LiDAR registered with a camera [100, 125]. Registering one or multiple cameras with the laser scanner requires a more sophisticated setup and the segmentation becomes more demanding. Using both cues may improve the results but it is seldom possible at the frame rate of the sensor. Therefore, we focus on segmenting unknown objects from pure 3D range data not requiring any additional visual or intensity information.

Visual information is not the only information that aids segmentation. Temporal information and tracking are also shown to be useful to enhance the segmentation performance [36, 127]. While the benefit of using the information about the moving objects is clear, we show that it is possible to perform a fast and meaningful segmentation on single scans even without relying on temporal integration.

### 5.1.5 Conclusion

This section presents a fast and easy to implement method for 3D laser range data segmentation including fast ground removal. Instead of operating in the 3D space, our approach performs all computations directly on the range images. This speeds up the segmentation of the individual range images and allows us to directly exploit neighborhood relations. It enables us to successfully segment even sparse laser scans like those recorded with a 16-beam Velodyne scanner. We implemented and evaluated our approach on different publicly available and self-recorded datasets and provide comparisons to other existing techniques. On a single core of a mobile i5 CPU, we obtain segmentation results at average frame rates between 74 Hz and 250 Hz and can run up to 667 Hz on an i7 CPU. We have released our code that can either be used standalone with C++ or as a ROS module.

## 5.2 Temporal object matching

Nowadays many applications, such as autonomous driving, rely on 3D data and this data is crucial for autonomous systems to understand the world and especially the moving objects around them. Many methods have been developed for interpreting such type of data. Such an interpretation, for example to estimate the speed of a moving object, requires the scans of the individual objects to be correctly registered with respect to each other. The information about the correctness and quality of the alignment is crucial when dealing with the real world. Not knowing the quality of the match can lead to serious damage of the robot or the environment, especially if the robot relies on scan matching while navigating in dynamic or hazardous environments [10]. Thus, this aspect is of great importance for automated driving as an autonomous car needs to be able to track obstacles on the road, e.g., by matching features defined on the 3D data [28]. Performing a wrong match can potentially lead to estimating speed of other cars wrongly or result in failing to recognize a pedestrian or losing track of an object.

In this section, we investigate the problem of evaluating the quality of 3D point clouds of objects in potentially dynamic scenes. We assume that the scan has already been segmented into objects, for example using our segmentation appoach presented in Section 5.1 and that the segmented objects have been individually aligned with an arbitrary registration method such as ICP, for example one presented in Section 3.1. The objective of this section is to provide an approach that evaluates such matches. It should provide a high score in situations in which the same objects are correctly aligned and low scores if both objects are not aligned well or if two objects from different classes (such as car and pedestrian) are aligned. The approach proposed here aims at achieving this, it is of probabilistic in nature, and takes into account occupancy and free-space information, which is typically ignored when matching point clouds using ICP.

The main contribution of this work is a novel approach that can generate a probabilistically motivated quality measure for the alignment of two point clouds corresponding to individual objects. Our approach is based on the observations of two point clouds and focuses on the projections of these point clouds to virtual image planes with expected free space information around objects. We explicitly consider potentially moving objects segmented from a range scan such as people or cars, see Figure 5.12 for an example. To illustrate the usefulness of the proposed evaluation, we present several experiments analyzing aligned 3D scans of objects and also show that our measure can be computed in the order of 1 ms/object. We furthermore illustrate that the analysis of an alignment can be used to support an object-tracker estimating the trajectories of dynamic objects or to help clustering real world objects and works better than considering a point-to-point similarity score.

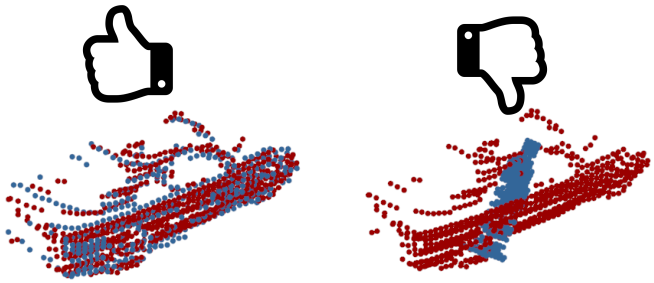

Figure 5.12: It is a non-trivial task to determine the alignment quality of matched point clouds of scanned objects as can be seen from the following example. *Left:* Two well-aligned point clouds of a car (red and blue). *Right:* Two different objects (car shown in red and person shown in blue) have been aligned with each other. Although the residuals of point-to-point differences may not be too large, the objects should not be matched. In this section we make an attempt to disambiguate such situations. Images best viewed in color.

### 5.2.1 Evaluating the alignment quality

Our approach is supposed to evaluate the alignment of two already aligned, 3D range scans of (partially) scanned objects such as those depicted in Figure 5.12. It is completely agnostic to the used alignment method as long as it computes the 3D transformation between the sensor poses at scanning time.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ refer to two 3D point clouds that have been registered in a common but arbitrary reference frame. Let $O_1$ be the origin of cloud $\mathcal{C}_1$, i.e., the pose of the sensor with respect to $\mathcal{C}_1$ when recording the cloud. The projection of cloud $\mathcal{C}_1$ onto the image plane positioned in $O_1$ results in a *real* depth image $\mathfrak{I}_{11}$ as shown in Figure 5.13. The same holds for $O_2$ with respect to $\mathcal{C}_2$. We can furthermore define two virtual image planes, one close to $O_1$ and one close to $O_2$ pointing towards the object, see Figure 5.14 for an illustration. The comparison of the two clouds is performed based only on the projections of the point clouds as depth images on the *virtual* image planes. This leads to four projected depth images $\mathfrak{I}_{11}$, $\mathfrak{I}_{12}$, $\mathfrak{I}_{21}$, and $\mathfrak{I}_{22}$, where $\mathfrak{I}_{ci}$ refers to the projection of the cloud $\mathcal{C}_c$ into the first or second image plane (index $i$). Each pixel in $\mathfrak{I}_{ci}$ stores the distance between the 3D point in $\mathcal{C}_c$ and the origin $O_i$. It is important to note that for the projections $\mathfrak{I}_{11}$ and $\mathfrak{I}_{22}$, we can also exploit negative information, i.e., knowledge about the free space behind the objects. We label the pixels surrounding the objects, which are generated from a scan segmentation approach (here using the approach presented in Section 5.1), as free space if their depth readings are larger than the distance to the object itself. This free space information is available only for the depth images $\mathfrak{I}_{11}$ and $\mathfrak{I}_{22}$, as we know that there is free space around the object as seen from the origin. This information may not be available for $\mathfrak{I}_{12}$ and $\mathfrak{I}_{21}$ as here the projected image does not necessarily lie between the point cloud and the physical scanner location during data acquisition.

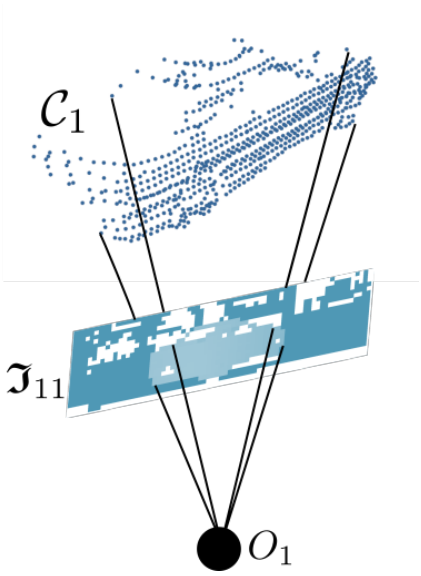


Figure 5.13: A sketch of the projection with darker shades on the projection plane depicting free space information.

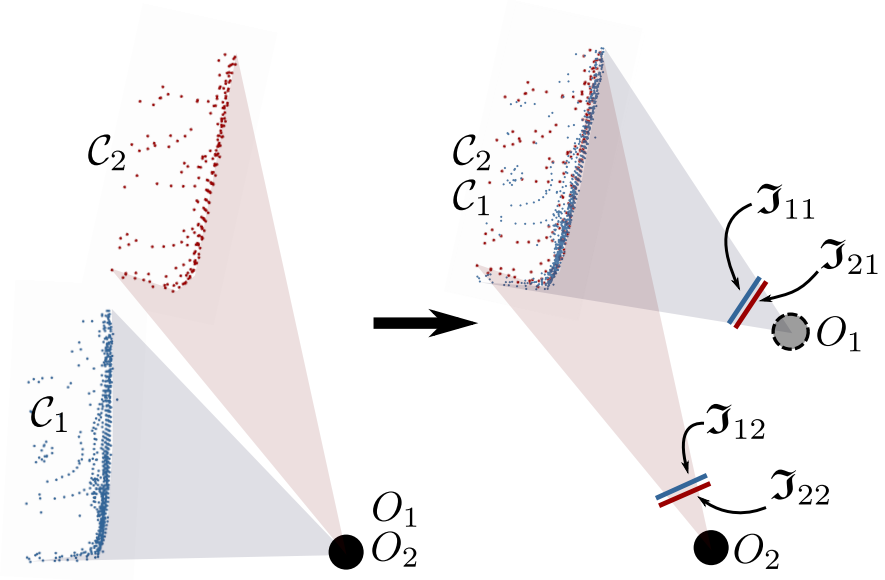


Figure 5.14: This image depicts two clouds viewed from above before and after registration with ICP. Note that both clouds are projected to all four depth images: $\mathfrak{I}_{11}$, $\mathfrak{I}_{21}$, $\mathfrak{I}_{12}$ and $\mathfrak{I}_{22}$. An example projection $\mathfrak{I}_{11}$ is shown in Figure 5.13. We compute a similarity measure for each pixel of corresponding projections following Equation (5.3). Knowing individual probabilities from pixels we continue to combine them with Equation (5.4).

Our analysis relies on comparing the depth images $\mathfrak{I}_{11}$ to $\mathfrak{I}_{21}$ and $\mathfrak{I}_{12}$ to $\mathfrak{I}_{22}$. Thus, we are comparing the projections of the two clouds in the same (potentially virtual) image plane. As only $\mathfrak{I}_{11}$ and $\mathfrak{I}_{22}$ encode free space information, there are the following possibilities when comparing the range images pixel by pixel. Let $d_j^{ci}$ be the depth information of pixel $j$ in $\mathfrak{I}_{ci}$, then we have three possible cases:

- Both images $\mathfrak{I}_{cc}$ and $\mathfrak{I}_{ci}$ with $c \in \{1, 2\}, i \neq c$ have a depth value stored in pixel $j$. In this case, we compute the probability of the two depth values to be generated by the same object by

$$\begin{aligned} p_j &= 1 - \frac{1}{\sqrt{2\pi}\sigma} \int_{-\Delta_j}^{\Delta_j} e^{\frac{-t^2}{2\sigma}} dt \\ &= 1 - \Big(\Phi(\frac{\Delta_j}{\sigma}) - \Phi(\frac{-\Delta_j}{\sigma})\Big), \end{aligned} \tag{5.3}$$

  where $\Delta_j$ is the distance between the depth readings in the virtual image plane at pixel $j$. Equation (5.3) considers Gaussian measurement noise with standard deviation $\sigma$ and thus the probability $p_j$ is the area under the tails of the normal distribution. This area can be computed via a difference of the following cumulative distribution functions (CDFs) of the given normal distribution.

- Pixel $j$ of image $\mathfrak{I}_{ci}$ has a depth reading while a pixel with the same coordinates in image $\mathfrak{I}_{cc}$ is marked as free space. In this case, we set the probability generated by these pixels to a low value corresponding to a value of $2\sigma$ as $\Delta_j$ in Equation (5.3).

- Pixel $j$ in either of the images $\mathfrak{I}_{cc}$ or $\mathfrak{I}_{ci}$ contains no value at all. In this case, we do not have enough information to make any decision and ignore the pixel.

We perform this computation for all the pixels $j = 1, \ldots, K$ in the projected depth image that have values in cloud $\mathcal{C}_1$ and $\mathcal{C}_2$. At this point, we have a probability $p_j$ for each pixel $j \in \{1, K\}$, where $K$ is the total number of pixels that have a non-zero value for both projections.

We consider the evaluation of each pixel as an expert that tells us the probability that the scans match. We can now apply a method to combine multiple expert opinions into one probability so that this probability defines degree of similarity between clouds $\mathcal{C}_1$ and $\mathcal{C}_2$. This problem is a well-known problem called opinion pooling. If we have no further information, we use a so-called democratic opinion pool [123], i.e., the similarity between two point clouds $\mathcal{C}_1$ and $\mathcal{C}_2$ is determined by a linear opinion pool:

$$p(\mathcal{C}_1, \mathcal{C}_2) = \sum_{j=1}^{K} \lambda_j p_j \tag{5.4}$$

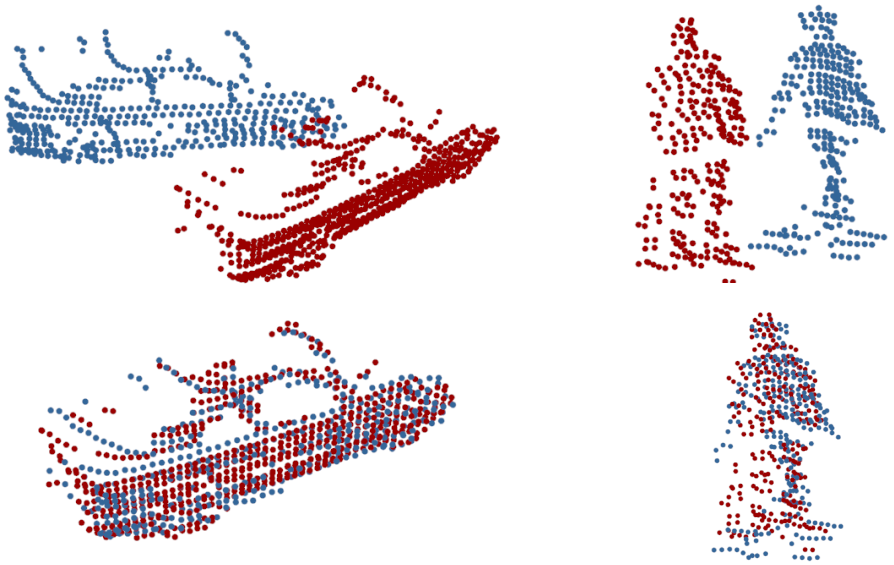


Figure 5.15: Example clouds of two cars and two pedestrians aligned using ICP algorithm along with the value of the similarity measure reported by our algorithm.

where $\lambda_j = \frac{1}{K}, \forall j \in \{1, K\}$, and $p_j$ are opinions reported by a corresponding expert, i.e. probabilities computed using Equation (5.3). Given this approach, we finally obtain with $p(\mathcal{C}_1, \mathcal{C}_2)$ a similarity measure between two 3D point clouds exploiting projections and free space information.

### 5.2.2 Experimental evaluation

We propose a measure to evaluate how well two 3D point clouds of objects are aligned. Thus, this evaluation is designed to show that (i) this measure is a useful tool for quantifying the alignment quality of 3D range data of objects, (ii) our approach can be executed fast, typically below 1 ms. Furthermore, we illustrate that (iii) it can support tracking and (iv) we can even use it to cluster different objects perceived in 3D scans to obtain semantically meaningful objects, and perform better than an ICP-like point-to-point measure, here using the implementation of PCL.

For our evaluation, we used several scans from the KITTI dataset [44] that have been recorded with a 64-beam Velodyne scanner. In this part of the KITTI dataset, typical objects are cars, people, vans, etc. We furthermore use sparser 3D data from a 16-beam Velodyne VLP-16 scanner recorded with a mobile robot on our campus in Bonn.

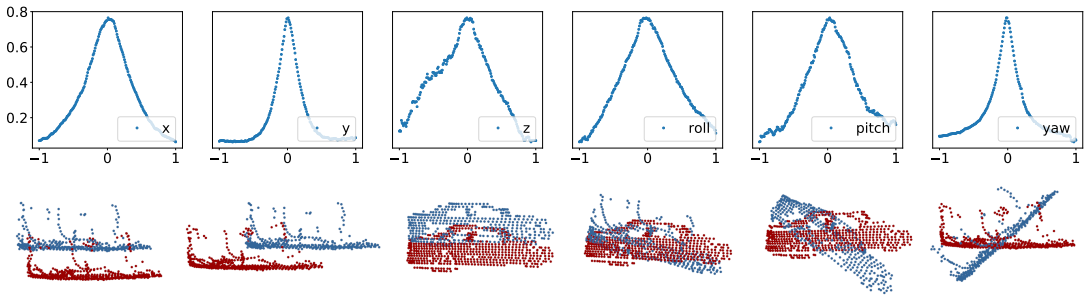


Figure 5.16: This image shows the changes in proposed similarity measure with a change in the relative position of two matched clouds. We have evaluated the changes in $x$, $y$, $z$ directions by displacing point clouds by up to 1 m as well as changing the roll, pitch and yaw of one of the clouds by an angle of up to 1 rad.

#### 5.2.2.1 Alignment quality

The first set of experiments is designed to illustrate that our approach for analyzing the alignment of scanned objects is a useful tool and provides meaningful scores with respect to the alignment quality. We analyze two different types of experiments here. First, the registration of the same physical object observed from different locations based on typical street scenes. Second, we evaluate how well different objects of the same class, e.g., two pedestrians or two cars can be aligned.

Figure 5.15 depicts two range scans and similarity scores of an object before and after registration. Figure 5.16 shows how the disturbance of the alignment changes the similarity score of our approach. As can be seen, the function peaks at the correct alignment. The plots illustrate how deviation from the true alignment changes the score. The larger the deviation the smaller the similarity score.

#### 5.2.2.2 Runtime

The next experiment is designed to show that the alignment score for two 3D point clouds recorded with a regular laser range scanner can be computed in an efficient manner so that it can be used for online operations easily. To quantify the runtime requirements, we executed the evaluation of different objects of different size and measured the runtime on a regular desktop computer with an Intel i7 CPU. The timings are summarized in Table 5.2. As can be seen from the table, the average computation time for typical objects such as cars or pedestrians can be executed in below 0.5 ms. Thus, our approach is suitable for an online assessment of the alignment quality for up to 100+ individual objects in the scene considering a frame rate of 10 Hz of the Velodyne laser scanner.

| dataset | objects | average runtime per object |
|---|---|---|
| KITTI (64-beams) | cars | approx. 0.45 ms |
| KITTI (64-beams) | pedestrians | approx. 0.38 ms |

Table 5.2: Average runtime for evaluating pairs of 3D scans of different objects on an Intel i7 CPU

#### 5.2.2.3 Support for tracking dynamic objects

The next experiment is designed to illustrate that the quality analysis of point clouds can support trackers that seek to estimate the trajectories of dynamic objects in the environment. To do this, we compute the quality measure for a made data association and subsequent point cloud alignment and reject associations that receive a low score. This approach rejects matchings in which people are fused with nearby walls or other flat objects. An example of such a situation is depicted in Figure 5.17. The sequence of images from 1 to 4 shows the result of an EKF tracker aided by our similarity measure rejecting data associations that receive a low score using our evaluation. As can be seen, here the tracks are not fused and the objects get tracked correctly.

#### 5.2.2.4 Support for clustering objects

Finally, we want to illustrate that our score is not only suited for evaluation of the alignment of scans taken from the same objects but could also be used for clustering different types of scanned objects in an unsupervised way and works better than the score that standard ICP provides. To illustrate that, we extracted the scanned cars, vans, and pedestrians from the KITTI dataset and computed a pair-wise ICP alignment after shifting all clouds so that the barycenter of each of them was in the origin. After the ICP alignment, we compute the similarity value using our approach between all pairs and store the values in a similarity matrix:

$$\mathbf{P}_{N,N} = \begin{pmatrix} p(\mathcal{C}_1, \mathcal{C}_1) & p(\mathcal{C}_1, \mathcal{C}_2) & \cdots & p(\mathcal{C}_1, \mathcal{C}_N) \\ p(\mathcal{C}_2, \mathcal{C}_1) & p(\mathcal{C}_2, \mathcal{C}_2) & \cdots & p(\mathcal{C}_2, \mathcal{C}_N) \\ \vdots & \vdots & \ddots & \vdots \\ p(\mathcal{C}_N, \mathcal{C}_1) & p(\mathcal{C}_N, \mathcal{C}_2) & \cdots & p(\mathcal{C}_N, \mathcal{C}_N) \end{pmatrix}, \tag{5.5}$$

where $p(\mathcal{C}_a, \mathcal{C}_b)$ refers to the similarity score between point clouds $\mathcal{C}_a$ and $\mathcal{C}_b$, while $N$ being the number of objects. For a comparison, we perform the clustering also based on the RMSE resulting from the point-to-point metric. Figure 5.18 illustrates two visual representations of such a matrix with two and three different types of objects. The left image always corresponds to the similarity matrix of our

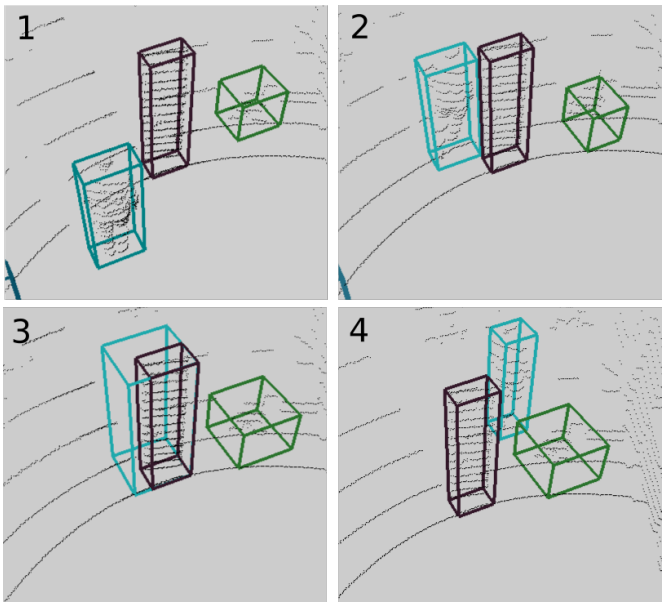


Figure 5.17: Our measure supports a dynamic object tracker and can help to reject data associations generated by matching nearby point clouds. The sequence of images shows the result of an EKF-based tracker performing a cloud validation step using our measure. We only update an object track if the similarity measure between a new object and a tracked one is high. As can be seen in the sequence, the person (cyan) is not fused with the flat vertical object (brown) and so this illustrates that our approach helps to disambiguate the objects so that separate tracks can be maintained. In the Images 1 and 2, the person approaches an object. In Image 3 the person is occluded by the object and the bounding box shows an EKF prediction. Note that the data association is done correctly. Image 4 depicts that EKF is able to resume tracking of the person once the person is seen again maintaining the original track id (given by the color).

approach while the one on the right corresponds to the RMSE-based matrix. In the images, dark blue corresponds to $p(\mathcal{C}_i, \mathcal{C}_j) = 0$, while light green to $p(\mathcal{C}_i, \mathcal{C}_j) = 1$. The color of the bars on the top represent the class of the objects below it after performing the spectral clustering. Note, that the text labels are for clarity only, and in reality are not providede by the algorithm.

We sorted the point clouds according to the class of scanned object and thus distinct squares in the matching matrix indicate that the measure can be used for clustering objects. To verify this, we perform spectral clustering [87] with both scores and test if the different classes of objects are correctly found given manually labeled ground truth data.

We use out-of-the-box spectral clustering as a standard approach to group objects. There may be more sophisticated ways for unsupervised clustering of 3D objects but this experiment suggests that our score serves as a better indicator than point-to-point RSME that the two point clouds match well. Figure 5.18 shows that the classes are separated in a meaningful way when using our method. A couple of failure cases for our (and the point-to-point) approach are depicted

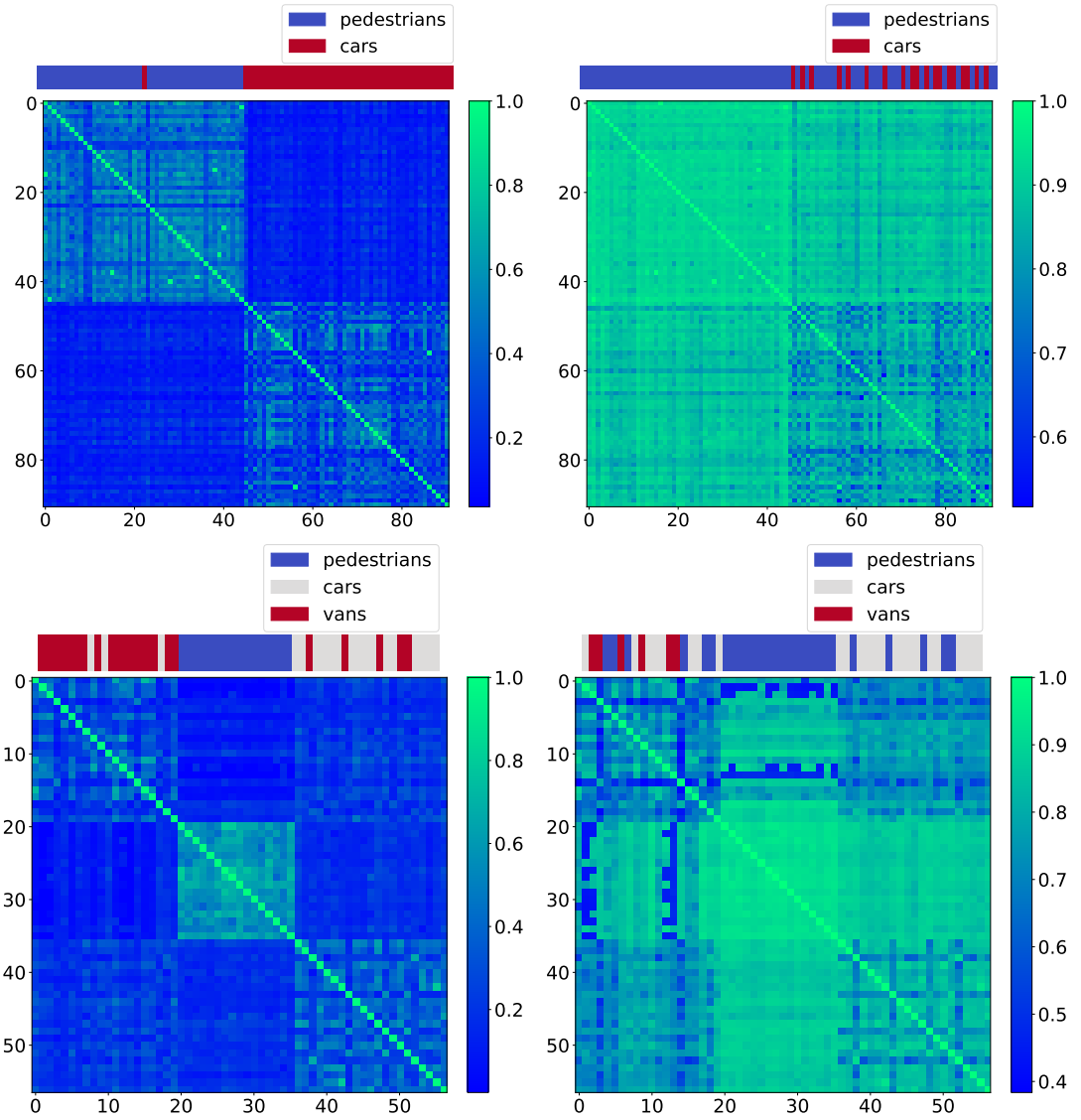


Figure 5.18: *Top row:* Left and right top images show similarity matrices of pairwise compared between 45 people scans against 54 cars. Left image is estimated using our method, while the right image using RMSE provided by PCL (with RBF kernel on top of it to generate similarity measures). The bars on top of the matrices (blue and red refer to scans of different classes of objects, here cars and people) show the results of unsupervised spectral clustering run on the corresponding matrices. *Bottom row:* The images show our similarity measure on the left and RMSE put through a robust kernel on the right. The data is sampled randomly from the annotations from driving sequences 91 and 51 from the KITTI dataset. In this example, we used 20 vans, 15 pedestrians and 25 cars. They are depicted in that order on the axis of each matrix as well as along the x-axis of the top bar. As can be seen from these bars on top of the matrices, the clustering performs better using our measure than for the RMSE. Both algorithms segment people into a single class correctly, however, our measure provides better separation between cars and vans. Note that the labels in the legend are provided for convenience purposes only as spectral clustering separates the data into clusters based on similarity with no knowledge of the semantic class of objects in each cluster.

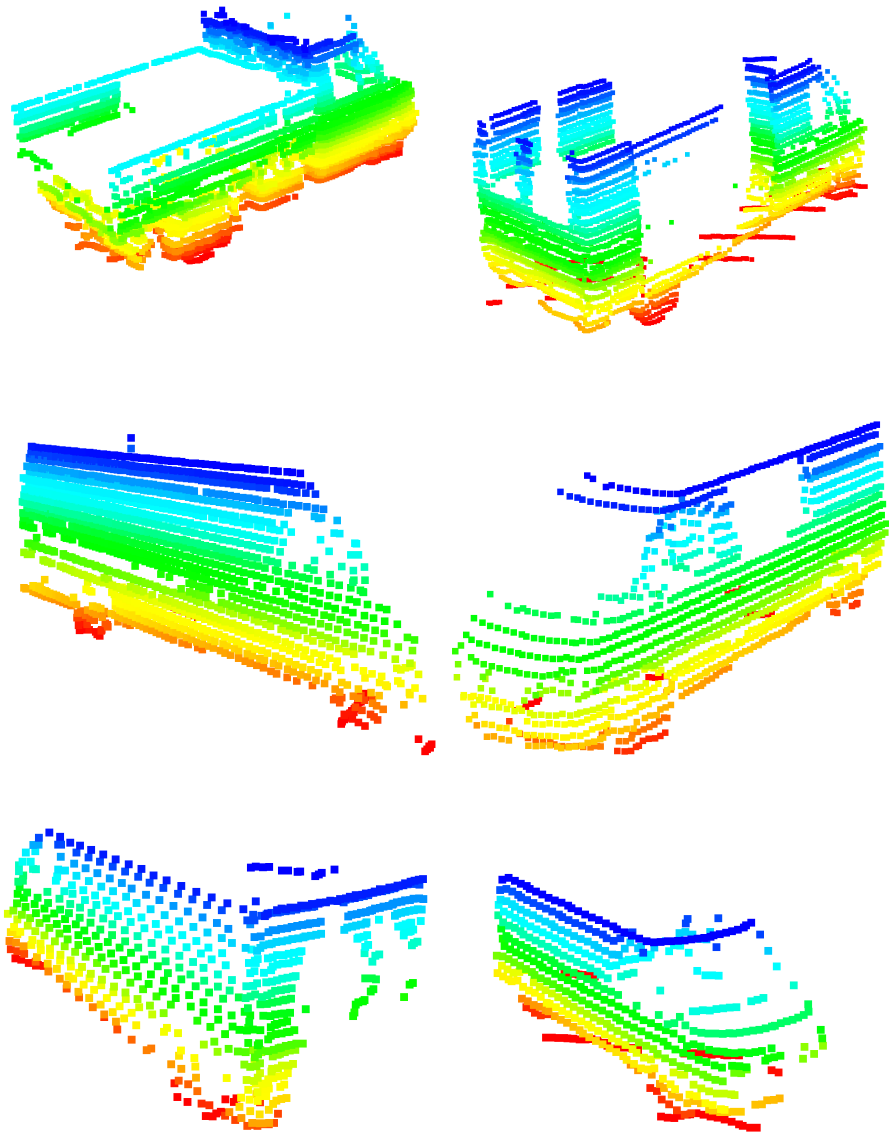

Figure 5.19: This image shows examples of vans that are hard to match. The colors represent the z coordinate and range from red to blue. The vans are very different from different sides and therefore sometimes match poorly. Also, the top-left van is hard to match against any other van in the dataset as it has a shape that differs substantially from the others. This explains bad matching scores in the top-left corner of Figure 5.18.

in Figure 5.19. The examples of vans driving in opposite directions and substantial shape differences make these point clouds hard to match. While the point clouds shown in Figure 5.19 have good density of points for visualization purposes, in reality it is often the case that the clouds are rather sparse, which further complicates matching. When performing spectral clustering based on the RMSE similarity matrix, the performance drops clearly. This suggests that our method supports such similarity-based clustering of objects better than the RMSE metric and provides enough information for an unsupervised clustering algorithm to find classes from unlabeled data.

### 5.2.3 Related work

Finding a good scan alignment is a well studied problem and several approaches have been proposed in the past. The most popular solution is probably the ICP algorithm [7] and many variants of the original algorithm exist. Most ICP-based approaches minimize the point-to-point distances between potentially corresponding points but other variants such as point-to-plane of generalized ICP [107] exist. We recommend the paper by Pomerleau *et al.* [98] for an overview.

To the best of our knowledge, mostly the probabilistic evaluation of the quality of the point cloud matching has been done in correlation with the actual matching process. For example, Olson [91] proposes a system that performs correlative scan matching, maximizing the likelihood of the match between full scans. This approach is suited for 2D data and would probably be computationally expensive if extended to 3D. To the best of our knowledge, there is no efficient 3D variant that searches in the 6D transformation space. In 2D, this method provides covariance estimates, which is valuable in determining the quality of the scan alignment. We think of this covariance matrix as an orthogonal information to our approach.

Another notable example of a method that provides a covariance matrix is a method by Censi [20]. He treats scan matching as a probability distribution approximation problem and provides an estimate of the matching uncertainty. His method seems to be reliable under severe sensor occlusions and it handles gracefully in constrained situations. However, this methods has been proposed for 2D matching and, to the best of our knowledge, is mainly used as a 2D method.

A 3D variant of covariance matrix estimation can be found in modern techniques for performing ICP. A recent example of such a method is a work by Prakhya *et al.* [99], which is an extension of the work by Censi [20] to account for 3D data. They also show that the covariance estimate of their method is lower in the global minimum comparing to local ones. However, it is still unclear which values of the covariance matrix should be considered good and which are not. Our method, while not forbidding to use the covariance matrix for match

likelihood estimation provides additional probabilistic measure, that is a single number and therefore is relatively easy to interpret.

Recently, Yang *et al.* [141] proposed a variant of ICP that searches for the globally optimal alignment of scans using a branch and bound approach. This is an interesting techniques, with guarantees on the performance. Even if a global approach is used, however, our metric will still be beneficial, as it could help to detect a individual, wrong alignment of a moving object to a different object.

Another notable method for registering dense point clouds is NICP by Serafin and Grisetti [109] that is designed for aligning full point clouds from Kinect-style sensors. This approach considers normal information for the matching and its open source implementation makes use of projections for matching. A related form of projection is also used in our work. A further work that has a similar motivation but is targeted to 2D SLAM is the work by Mazuran *et al.* [80]. They analyze the map consistency by performing cascaded statistical tests on pairs of 2D scans and overlapping 2D polygons and use this information to determine parameters in the used SLAM backend. A further approach that bears similarities to the ideas presented here is the work by Hähnel *et al.* [51]. As part of their approach, they analyze the log-likelihood of data associations in SLAM, when searching in the space of data associations. The log-likelihood of each measurement is obtained by superimposing a scan onto a local 2D occupancy grid map built by another scan.

One more interesting method by Endres *et al.* [34] models object classes as distributions over features and use Latent Dirichlet Allocation to learn clusters of 3D objects according to similarity in shape. However, it is interesting for us to avoid using features to describe the scene and to work on pure 3D data.

To the best of our knowledge, there are not many approaches to estimating the scan alignment quality that go beyond sums of squared distances between points or planes or estimated covariance matrices in the space of transformations. Most approaches furthermore use scans as a whole and not partial scans of objects. A notable exception is the work by Blais and Levine [8] where the range images of partial scans of the objects are registered minimizing the projective distance between the points sampled in the range image. Our work is as well as their work relies on projective distance evaluation, however, we avoid sampling the points, make use of the projections of the clouds on multiple planes, introduce the notion of free space and focus on making a decision about the quality of the match in a probabilistic sense.

### 5.2.4 Conclusion

We propose a novel way for analyzing the alignment quality of registered point clouds of individual objects. We provide a fast to compute, probabilistic similarity measure for any pair of registered point clouds of objects and do not rely on any specific registration or segmentation procedure. Our approach uses projections of the point clouds alongside with information about the free space that surrounds the scanned objects to evaluate a match. As our experiments suggest, the probabilistic measure is well-suited to analyze matches, supports tracking dynamic objects, and allows us to cluster point clouds of different types of objects in an unsupervised way better than a point-to-point metric would do.

# Chapter 6

# Conclusion

Robots take on increasingly more complex roles in our daily life. They can deliver goods, drive people around, help in search-and-rescue scenarios or map historical sites. These tasks are far from simple. To be successful, a robot must gain knowledge about the environment, build its map, and localize itself in it. When such an environment is dynamic, the robot must also reason about the objects that surround it: what will move and what will remain static. Because of this variety of challenges that the robot must address, usually, the software that steers the robot is designed with a specific sensor configuration in mind, and is not trivial to adapt to other robots, scenarios, or sensor configuration. This thesis challenges this concept by presenting a number of approaches that target a typical robot navigation pipeline while using algorithms that work with general range sensors such as RGBD cameras and LiDARs without adaptations to individual sensor models, and aims to provide methods for navigation and scene perception in both static and dynamic environments. For a static world, we provide contributions in mapping by presenting our general multi-cue photometric point cloud registration method, analyze the traversability of the surroundings of the robot to allow for safe navigation in potentially hazardous environments such as mines, catacombs or caves, and provide solutions for efficient and safe exploration and homing. In the dynamic setting, we aim to identify distinct objects to aid navigation and scene understanding. These objects can be tracked with any available tracking algorithm, and we provide a robust measure that provides the information if the two objects match each other. In the remainder of this section, we will briefly summarize the contributions of each of the parts of this thesis on the per-component basis.

## 6.1 Short summary of key contributions

To aid robot mapping, we have developed a novel flexible and robust framework for point cloud registration that can work with both, RGBD and LiDAR sensors and runs at frame rate of these sensors. In contrast to the classical ICP implementations, our approach does not need the computationally expensive data association step and in contrast to the dense visual odometry method [67], it can be easily extended to new sensors or data modalities. Our method allows for matching the point clouds with no explicit data associations by using range image representation of the data. We have implemented and thoroughly tested our approach and have released our implementation as open source C++ code. The experiments show that our approach allows for an accurate registration of the sensor data without requiring an explicit data association or model-specific adaptations to datasets or sensors. Our approach exploits the different cues in a natural and consistent way and the registration can be done at the frame rate of a typical range or imaging sensor. The main contribution of this method is the ability to extend the matching procedure by any modality while still performing at the frame rate of the sensor.

On the side of the scene analysis, we have developed an effective method for classifying the robots surrounding into traversable and non-traversable areas. This capability is crucial for safe navigation in an unknown environment and to safely return to the base. Our approach processes the depth data at 10 fps-25 fps on a standard notebook computer without using the GPU and it allows for robustly identifying the areas in front of the sensor that are safe for navigation. The component presented here is one of the building blocks of the EU project ROVINA that aims at the exploration and digital preservation of hazardous archaeological sites with mobile robots. Real world evaluations have been conducted in controlled lab environments, in an outdoor scene, as well as in a real, partially unexplored, and roughly 1700 year old Roman catacomb. The navigation and mapping systems have been released as open source software as part of ROVINA software suite and is now used by the United Nations International Council on Monuments and Sites.

For the navigation of the robot, we built systems to include previous knowledge to detect the best way to explore the environment. In autonomous exploration tasks, robots usually rely on a SLAM system to build a map of the environment online and then use it for navigation purposes. These maps are not always reliable and a number of wrong associations can break a map making the navigation of the robot much more complex and potentially even dangerous. To target this challenge, we developed a system that is able to detect that the map is in an inconsistent state, and can safely return the robot to the base. We implemented the proposed system in C++, integrated it with the robot operating

system (ROS) and showcase its effectiveness on an autonomous exploration robot in an underground cave in Niederzissen, in the Priscilla catacomb in Rome and in office environments.

In addition to the contributions to the operation of robots in static scenes we have also developed a number of methods that aid scene understanding when dynamic objects are present in the scene. The ability to extract individual objects in the scene is key for a large number of autonomous navigation systems such as mobile robots or autonomous cars. Many systems that detect and track the dynamic objects rely on a pre-segmentation of the scene into objects which are consequently analyzed in order to determine if they belong to the static or the dynamic part of the scene. In this part of our work, we have developed an effective method that removes the ground from the scan and then segments the 3D data into different objects using a range image representation. A key focus of our work is a fast execution with several hundred hertz. Our implementation has small computational demands so that it can run online on most mobile systems. We explicitly avoid the computation of the 3D point cloud and operate directly on a 2.5D range image, which enables fast segmentation for each 3D scan. This approach can furthermore handle sparse 3D data well, which is important for scanners such as the new Velodyne VLP-16 scanner. We implemented our approach in C++ and integrated it with ROS, thoroughly tested it using different 3D scanners, and have released the source code of our implementation on GitHub. Since the time we have uploaded the code to GitHub, it has gained popularity: 135 stars and 87 forks to date. Our method can operate at frame rates that are substantially higher than those at which the sensors provide data, while using only a single core of a mobile CPU and producing high quality segmentation results. This efficiency, bundled with a single bounded parameter that guides the clustering, is the main contribution of this method.

Simply detecting the surrounding objects is not enough for safe navigation, thus, we have implemented an extension to our object cloud matching procedure in order to make sure that the matched clouds of the objects actually represent the objects of the same class. Most matching methods such as numerous flavors of ICP provide little information about the quality of the match, i.e., how well the matched objects correspond to each other, which goes beyond point-to-point or point-to-plane distances. We propose a projective method that yields a probabilistic measure for the quality of matched scans. It not only considers the differences in the point locations but also takes free-space information into account. Our approach provides a probabilistic measure that is meaningful enough to cluster real-world data such as scans taken with Velodyne scanner in urban scenes in an unsupervised manner. The main contribution of this work is the idea of using the free space around the object seen from the position of the sensor as

well as the method that is able to quickly and efficiently provide a single bounded measure for likeliness between the point clouds of the objects.

Overall, we have developed and implemented a number of components crucial for robot navigation in static and dynamic environments. While all of these novel components made scientific contributions over the state of the art, and have been tested on real robots and real-world datasets, there is still a substantial amount of engineering effort required to build a complete system that would be able to run on various mobile robots. Designing and implementing a completely generic system that is able to map the environment, localize the robot in such a map, perform exploration and navigation tasks while making sure that the robot and the objects surrounding it are safe is an ambitious task and is, probably, outside of the scope of a single Ph.D. thesis, as more software engineering effort is required. However, we believe that the components implemented in this thesis will be helpful in making such a system a reality. To support this statement, we released most of our software as open source, and several of our software releases have become quite popular among other researchers.

An interesting direction for the future work would be to integrate the object detection and tracking into a fully robust system for working with dynamic objects and learn their traits to be able to integrate these objects with the mapping system. For example, if learnt, the objects that have a dynamic class but happened to be static during data acquisition can be found in the map and removed from it, potentially aiding the matching algorithms. However, these tasks require some form of unsupervised or semi-supervised learning, which is a research direction orthogonal to the one of this thesis.

## 6.2 Contributions to the ROVINA project

We developed the traversability analysis, navigation and robust homing systems, presented in this thesis, as part of a successful EC-funded project ROVINA, supported under FP7-600890-ROVINA. This is a project for autonomous exploration and digital preservation of hard-to-access archaeological sites such as catacombs. The financial support of the EC through the ROVINA project is gratefully acknowledged.

On September 26, 2016, the final review of the 42-month project was successful and ROVINA has been evaluated excellently in all four review meetings. The systems presented in this thesis were put under test in real Roman catacombs during multiple review meetings and the achieved performance has played an important role in the excellent final evaluation of the project. Figure 6.1 shows photos from various integration and project review meetings that showcase our team working on the robot and Albert Gauthier, officer of the European Com-

Figure 6.1: Photos from various integration and review meetings of the ROVINA project. *Top left:* a test run with the robot in a man-made cave in Niederzissen. *Top middle:* an integration meeting taking place in Rome. *Top right:* Albert Gauthier, officer of the European Commission for cultural heritage, looking at the 3D map of the Priscilla catacombs built by the robot, using an Oculur Rift headset.

mission for cultural heritage, looking at the 3D map of the Priscilla catacomb through an Oculus Rift headset.

## 6.3 Open source contributions

Several methods presented in this thesis have been published as open source software. Here we provide a list of links where each of the components is published:

- Multi-cue photometric point cloud matching, presented in Section 3.1: `https://gitlab.com/srrg-software/srrg_mpr`
- Traversability analysis, presented in Section 4.1, parts of the navigation and exploration pipelines presented in Section 4.2, and robust homing, presented in Section 4.3, are parts of the ROVINA project and their code is distributed from the ROVINA website as part of ROVINA code base: `http://www.rovina-project.eu/research/software_releases`
- Range image-based clustering, presented in Section 5.1: `https://github.com/PRBonn/depth_clustering`

# Bibliography


[1] S.M. Abdullah, M. Awrangjeb, and G. Lu. LiDAR segmentation using suitable seed points for 3d building extraction. *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(3):1–8, 2014.

[2] A. Angelova, L. Matthies, D. Helmick, and P. Perona. Dimensionality reduction using automatic supervision for vision-based terrain learning. In *Proc. of Robotics: Science and Systems (RSS)*, 2007.

[3] R.B. Ash and C. Doléans-Dade. *Probability and Measure Theory*. Harcourt/Academic Press, 2000.

[4] M. Bansal, B. Matei, H. Sawhney, S. H. Jung, and J. Eledath. Pedestrian detection with depth-guided structure labeling. In *Proc. of the Int. Conf. on Computer Vision (ICCV) Workshops*, pages 31–38, 2009.

[5] P. Beeson, N.K. Jong, and B. Kuipers. Towards autonomous topological place detection using the extended voronoi graph. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2005.

[6] J. Behley, V. Steinhage, and A. B. Cremers. Laser-based segment classification using a mixture of bag-of-words. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4195–4200, 2013.

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

[8] G. Blais and M. D. Levine. Registering multiview range data to create 3d computer objects. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 17(8):820–824, 1995.

[9] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 298–304, 2015.

[10] I. Bogoslavskyi, M. Mazuran, and C. Stachniss. Robust Homing for Autonomous Robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2016.

[11] I. Bogoslavskyi and C. Stachniss. Fast range image-based segmentation of sparse 3d laser scans for online operation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[12] I. Bogoslavskyi and C. Stachniss. Analyzing the quality of matched 3d point clouds of objects. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[13] I. Bogoslavskyi and C. Stachniss. Efficient online segmentation for sparse 3d laser scans. In *Photogrammetrie – Fernerkundung – Geoinformation (PFG)*, volume 85, pages 41–52, 2017.

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

[15] F. Bourgoult, A.A. Makarenko, S.B. Williams, B. Grocholsky, and F. Durrant-Whyte. Information based adaptive robotic exploration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2002.

[16] A. Bovik. *Handbook of Image and Video Processing*, chapter 2. Elsevier, 2005.

[17] D. Bradley, R. Unnikrishnan, and J. Bagnell. Vegetation detection for driving in complex environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2007.

[18] C.A. Brooks, K. Iagnemma, and S. Dubowsky. Vibration-based terrain analysis for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3415–3420, 2005.

[19] L. Cabaret, L. Lacassagne, and L. Oudni. A review of world's fastest connected component labeling algorithms: Speed and energy estimation. In *International Conference on Design and Architectures for Signal and Image Processing*, pages 1–6, 2014.

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

[21] H.J. Chang, C.S.G. Lee, Y. Lu, and Y.C. Hu. P-slam: Simultaneous localization and mapping with environmental-structure prediction. *IEEE Trans. on Robotics (TRO)*, 23(2):281–293, 2007.

[22] Y. Choe, S. Ahn, and M. J. Chung. Fast point cloud segmentation for an intelligent vehicle using sweeping 2d laser scanners. In *International Conference on Ubiquitous Robots and Ambient Intelligence*, pages 38–43, 2012.

[23] G. De Cubber, D. Doroftei, H. Sahli, and Y. Baudoin. Outdoor terrain traversability analysis for robot navigation using a time-of-flight camera. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2011.

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

[25] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski. Self-supervised monocular road detection in desert terrain. In *Proc. of Robotics: Science and Systems (RSS)*, Philadelphia, USA, 2006.

[26] B. Della Corte*, I. Bogoslavskyi*, C. Stachniss, and G. Grisetti. A general framework for flexible multi-cue photometric point cloud registration. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2018. * authors have contributed equally.

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

[28] A. Dewan, T. Caselitz, G.D. Tipaldi, and W. Burgard. Rigid scene flow for 3d lidar scans. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[29] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[30] B. Douillard, D. Fox, and F. Ramos. Laser and vision based outdoor object mapping. In *Proc. of Robotics: Science and Systems*, 2008.

[31] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel. On the segmentation of 3d lidar point clouds. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2798–2805, 2011.

[32] B. Douillard, J. Underwood, V. Vlaskine, A. Quadros, and S. Singh. A pipeline for the segmentation and classification of 3d point clouds. In *International Symposium on Experimental Robotics*, pages 585–600, 2014.

[33] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287 – 300, 2002.

[34] F. Endres, C. Plagemann, C. Stachniss, and W. Burgard. Unsupervised discovery of object classes from range data using latent dirichlet allocation. *Proc. of Robotics: Science and Systems (RSS)*, 2:113–120, 2009.

[35] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 834–849, 2014.

[36] G. Floros and B. Leibe. Joint 2d-3d temporally consistent semantic segmentation of street scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2823–2830, 2012.

[37] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. SVO: semidirect visual odometry for monocular and multicamera systems. *IEEE Trans. on Robotics (TRO)*, 33(2):249–265, 2017.

[38] W. Förstner. Graphical models in geodesy and photogrammetry-graphische modelle in geodäsie und photogrammetrie. *Photogrammetrie-Fernerkundung-Geoinformation*, 2013(4):255–267, 2013.

[39] D. Fox, J. Ko, K. Konolige, and B. Stewart. A hierarchical bayesian approach to the revisiting problem in mobile robot map building. In *Proc. of the Int. Symposium on Robotic Research (ISRR)*, 2003.

[40] P. Furgale and T. Barfoot. Stereo mapping and localization for long-range path following on rough terrain. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4410–4416, 2010.

[41] P. Furgale and T. Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics (JFR)*, 27:534–560, 2010.

[42] P. Furgale, P. Krüsi, F. Pomerleau, U. Schwesinger, F. Colas, and R. Siegwart. There and back again — dealing with highly-dynamic scenes and long-term change during topological/metric route following. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2014.

[43] A. Geiger, M. Lauer, F. Moosmann, B. Ranft, H. Rapp, C. Stiller, and J. Ziegler. Team annieway's entry to the 2011 grand cooperative driving challenge. *IEEE Trans. on Intelligent Transportation Systems (ITS)*, 13(3):1008–1017, 2012.

[44] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[45] A. Golovinskiy and T. Funkhouser. Min-cut based segmentation of point clouds. In *International Conference on Computer Vision Workshops*, pages 39–46, 2009.

[46] B. Gorte, S Oude Elberink, B. Sirmacek, and J. Wang. Tree separation and classification in mobile mapping lidar data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(3/W3):607–612, 2015.

[47] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based SLAM. *IEEE Trans. on Intelligent Transportation Systems Magazine*, 2:31–43, 2010.

[48] G. Grisetti, D. Lordi Rizzini, C. Stachniss, E. Olson, and W. Burgard. Online constraint network optimization for efficient maximum likelihood map learning. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Pasadena, CA, USA, 2008.

[49] G. Grudic, J. Mulligan, M. Otte, and A. Bates. Online learning of multiple perceptual models for navigation in unknown terrain. In *Field and Service Robotics*, pages 411–420. Springer, 2008.

[50] T. Hackel, J.D. Wegner, and K. Schindler. Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 3:177–184, 2016.

[51] D. Hähnel, W. Burgard, B. Wegbreit, and S. Thrun. Towards lazy data association in SLAM. In *Proc. of the Int. Symposium on Robotic Research (ISRR)*, 2003.

[52] A. Hanel, H. Klöden, L. Hoegner, and U. Stilla. Image based recognition of dynamic traffic situations by evaluating the exterior surrounding and interior space of vehicles. *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(3):161–168, 2015.

[53] M. Happold, M. Ollis, and N. Johnson. Enhancing supervised terrain classification with predictive unsupervised learning. In *Proceedings of Robotics: Science and Systems*, 2006.

[54] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[55] M. Hebel and U. Stilla. Pre-classification of points and segmentation of urban objects by scan line analysis of airborne lidar data. *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37(B3a):105–110, 2008.

[56] M. Hebert and N. Vandapel. Terrain classification techniques from ladar data for autonomous navigation. In *Proc. of the Collaborative Technology Alliances conference*, College Park, MD., 2003.

[57] A. Hermans, G. Floros, and B. Leibe. Dense 3d semantic mapping of indoor scenes from rgb-d images. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2631–2638, 2014.

[58] M. Himmelsbach, F. v Hundelshausen, and H. Wuensche. Fast segmentation of 3d point clouds for ground vehicles. In *IEEE Intelligent Vehicles Symposium*, pages 560–565, 2010.

[59] G. Hitz, A. Gotovos, F. Pomerleau, M.-E. Garneau, C. Pradalier, A. Krause, and R.Y. Siegwart. Fully autonomous focused exploration for robotic environmental monitoring. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2014.

[60] D. Holz, N. Basilico, F. Amigoni, and S. Behnke. A comparative evaluation of exploration strategies and heuristics to improve them. In *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, pages 25–30, 2011.

[61] S. Holzer, R.B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[62] M. Jaimez and J. Gonzalez-Jimenez. Fast visual odometry for 3-d range sensors. *IEEE Trans. on Robotics (TRO)*, 31(4):809–822, 2015.

[63] I. Katramados, S. Crumpler, and T.P. Breckon. Real-time traversable surface detection by colour space fusion and temporal analysis. In *Proc. International Conference on Computer Vision Systems*, 2009.

[64] M. Keller, D. Lefloch, M. Lambers, and S. Izadi. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of the Int. Conf. on 3D Vision (3DV)*, pages 1–8, 2013.

[65] C. Kerl, J. Stuckler, and D. Cremers. Dense continuous-time tracking and mapping with rolling shutter rgb-d cameras. In *Proc. of the IEEE Int. Conf. on Computer Vision (ICCV)*, pages 2264–2272, 2015.

[66] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for RGB-D cameras. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2100–2106, 2013.

[67] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for RGB-D cameras. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3748–3754, 2013.

[68] K. Klasing, D. Wollherr, and M. Buss. A clustering method for efficient segmentation of 3d laser data. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4043–4048, 2008.

[69] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai. A practical, decision-theoretic approach to multi-robot mapping and exploration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3232–3238, Las Vegas, NV, USA, 2003.

[70] S. Koenig and C. Tovey. Improved analysis of greedy mapping. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2003.

[71] D. Korchev, S. Cheng, Y. Owechko, and K. Kim. On real-time lidar data segmentation and classification. *International Conference on Image Processing, Computer Vision, and Pattern Recognition*, 1:42–49, 2013.

[72] A. Krause and C. Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proc. of Uncertainty in Artificial Intelligence (UAI)*, 2005.

[73] H. Kretzschmar and C. Stachniss. Information-theoretic pose graph compression for laser-based SLAM. *Int. Journal of Robotics Research (IJRR)*, 31:1219–1230, 2012.

[74] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. A navigation system for robots operating in crowded urban environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3225–3232, 2013.

[75] B. Leibe, K. Schindler, N. Cornelis, and L. Van Gool. Coupled object detection and tracking from static cameras and moving vehicles. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 30(10):1683–1698, 2008.

[76] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

[77] D. Maier, C. Stachniss, and M. Bennewitz. Vision-based humanoid navigation using self-supervised obstacle detection. *The Int. Journal of Humanoid Robotics (IJHR)*, 10, 2013.

[78] A.A. Makarenko, S.B. Williams, F. Bourgoult, and F. Durrant-Whyte. An experiment in integrated exploration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2002.

[79] L. Matthies, M. Turmon, A. Howard, A. Angelova, B. Tang, and E. Mjolsness. Learning for autonomous navigation: Extrapolating from underfoot to the far field. *Journal of Machine Learning Research*, 1, 2005.

[80] M. Mazuran, G.D. Tipaldi, L. Spinello, W. Burgard, and C. Stachniss. A Statistical Measure for Map Consistency in SLAM. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Hong Kong, China, 2014.

[81] M. Menze, C. Heipke, and A. Geiger. Joint 3d estimation of vehicles and scene flow. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(3/W5):427–434, 2015.

[82] F. Moosmann. *Interlacing self-localization, moving object tracking and mapping for 3d range sensors*. PhD thesis, KIT, 2013.

[83] F. Moosmann, O. Pink, and C. Stiller. Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. In *Intelligent Vehicles Symposium*, pages 215–220, 2009.

[84] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, 1988.

[85] C. Mostegel, A. Wendel, and H. Bischof. Active monocular localization: Towards autonomous monocular exploration for multirotor mavs. In *Proc. of*

*the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Hong Kong, China, 2014.

[86] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proc. of the Int. Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[87] A.Y. Ng, M.I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[88] W. Niemeier. *Ausgleichungsrechnung: Statistische Auswertemethoden*. Walter de Gruyter, 2008.

[89] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–652–I–659 Vol.1, 2004.

[90] M. Nitsche, T. Pire, T. Krajnik, M. Kulich, and M. Mejail. Monte carlo localization for teach-and-repeat feature-based navigation. In M. Mistry, A. Leonardis, M. Witkowski, and C. Melhuish, editors, *Advances in Autonomous Robotics Systems*, volume 8717 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2014.

[91] E.B. Olson. Real-Time Correlative Scan Matching. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4387–4393, 2009.

[92] S. Osswald, M. Bennewitz, W. Burgard, and C. Stachniss. Speeding-Up Robot Exploration by Exploiting Background Information. *IEEE Robotics and Automation Letters (RA-L)*, 2016.

[93] C.J. Ostafew, A.P. Schoellig, and T.D. Barfoot. Visual teach and repeat, repeat, repeat: Iterative learning control to improve mobile robot path tracking in challenging outdoor environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 176–181, 2013.

[94] A. Ošep, A. Hermans, F. Engelmann, D. Klostermann, M. Mathias, and B. Leibe. Multi-scale object candidates for generic object tracking in street scenes. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3180–3187, 2016.

[95] E. Palazzolo and C. Stachniss. Information-driven autonomous exploration for a vision-based mav. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017.

[96] D. Perea-Ström, I. Bogoslavskyi, and C. Stachniss. Robust exploration and homing for autonomous robots. In *Journal on Robotics and Autonomous Systems (RAS)*, volume 90, pages 125–135, 2017.

[97] A. Petrovskaya and S. Thrun. Model based vehicle tracking for autonomous driving in urban environments. *Proc. of Robotics: Science and Systems (RSS)*, 34, 2008.

[98] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013.

[99] S.M. Prakhya, L. Bingbing, Y. Rui, and W. Lin. A closed-form estimate of 3d icp covariance. In *Machine Vision Applications (MVA), 2015 14th IAPR International Conference on*, 2015.

[100] T. Pylvanainen, K. Roimela, R. Vedantham, J. Itaranta, and R. Grzeszczuk. Automatic alignment and multi-view segmentation of street view data using 3d shape priors. *Symposium on 3D Data Processing, Visualization and Transmission*, 737:738–739, 2010.

[101] C. Rasmussen. Kinects for low- and no-sunlight outdoor trail-following. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[102] X. Ren, L. Bo, and D. Fox. Indoor scene labeling using rgb-d data. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[103] A. Renner, T. Foehst, and K. Berns. Perception of environment properties relevant for off-road navigation. In *Proc. of Autonome Mobile Systeme*, 2009.

[104] R. Rocha, J. Dias, and A. Carvalho. Exploring information theory for vision-based volumetric mapping. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2409–2414, 2005.

[105] S.A. Sadat, K. Chutskoff, D. Jungic, J. Wawerla, and R. Vaughan. Feature-rich path planning for robust navigation of mavs with mono-slam. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2014.

[106] A. Savitzky and M.J.E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

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

[108] J. Serafin, M. Di Cicco, T. M. Bonanni, G. Grisetti, L. Iocchi, D. Nardi, C. Stachniss, and V. A. Ziparo. Robots for exploration, digital preservation and visualization of archeological sites. In *Artificial Intelligence for Cultural Heritage*, pages 121–140, 2016.

[109] J. Serafin and G. Grisetti. NICP: dense normal based point cloud registration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 742–749, 2015.

[110] J. Serafin and G. Grisetti. Using extended measurements and scene merging for efficient and robust point cloud registration. *Journal on Robotics and Autonomous Systems (RAS)*, 92(C):91–106, 2017.

[111] R. Shade and P. Newman. Choosing where to go: Complete 3d exploration with stereo. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[112] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[113] S. Shen, N. Michael, and V. Kumar. 3d indoor exploration with a computationally constrained mav. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012.

[114] R. Sim, G. Dudek, and N. Roy. Online control policy optimization for minimizing map uncertainty during exploration. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2004.

[115] M. Slavcheva, W. Kehl, N. Navab, and S. Ilic. Sdf-2-sdf: Highly accurate 3d object reconstruction. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 680–696. Springer International Publishing, 2016.

[116] C. Sprunk, G.D. Tipaldi, A. Cherubini, and W. Burgard. Lidar-based teach-and-repeat of mobile robot trajectories. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013.

[117] C. Stachniss and W. Burgard. Mapping and exploration with mobile robots using coverage maps. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 476–481, 2003.

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

[119] C. Stachniss, J. Leonard, and S. Thrun. *Springer Handbook of Robotics, 2nd edition*, chapter Chapt. 46: Simultaneous Localization and Mapping. Springer Verlag, 2016.

[120] C. Stachniss, O. Martinez Mozos, and W. Burgard. Efficient exploration of unknown indoor environments using a team of mobile robots. *Annals of Mathematics and Artificial Intelligence*, 52:205ff, 2009.

[121] C. Stachniss, O. Martínez-Mozos, A. Rottmann, and W. Burgard. Semantic labeling of places. In *Proc. of the Int. Symposium on Robotic Research (ISRR)*, San Francisco, CA, USA, 2005.

[122] D. Steinhauser, O. Ruepp, and D. Burschka. Motion segmentation and scene classification from 3d lidar data. In *Intelligent Vehicles Symposium*, pages 398–403, 2008.

[123] M. Stone. The opinion pool. *Ann. Math. Statist.*, 32(4):1339–1342, 1961.

[124] D. Perea Ström, F. Nenci, and C. Stachniss. Predictive exploration considering previously mapped environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015.

[125] J. Strom, A. Richardson, and E. Olson. Graph-based segmentation for colored 3d laser point clouds. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2131–2136, 2010.

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

[127] A. Teichman and S. Thrun. Tracking-based semi-supervised learning. *Int. Journal of Robotics Research (IJRR)*, 31(7):804–818, 2012.

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

[129] A. Velizhev, R. Shapovalov, and K. Schindler. Implicit shape models for object detection in 3d point clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(3):179–184, 2012.

[130] J. Wang and J. Shan. Segmentation of lidar point clouds for building extraction. In *Anual Conference of the American Society for Photogrammetry and Remote Sensing*, pages 9–13, 2009.

[131] M. Weinmann and B. Jutzi. Geometric point quality assessment for the automated, markerless and robust registration of unordered tls point clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(3/W5):89–96, 2015.

[132] M. Weinmann, B. Jutzi, S. Hinz, and C. Mallet. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 105:286–304, 2015.

[133] T. Weise, T. Wismer, B. Leibe, and L. Van Gool. Online loop closure for real-time interactive 3D scanning. *Computer Vision and Image Understanding*, 115:635–648, 2011.

[134] C. Weiss, H. Frohlich, and A. Zell. Vibration-based terrain classification using support vector machines. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[135] P. Whaite and F. P. Ferrie. Autonomous exploration: Driven by uncertainty. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):193–205, 1997.

[136] D.F. Wolf, G. Sukhatme, D. Fox, and W. Burgard. Autonomous terrain mapping and classification using hidden markov models. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2005.

[137] K.M. Wurm, H. Kretzschmar, R. Kümmerle, C. Stachniss, and W. Burgard. Identifying vegetation from laser data in structured outdoor environments. *Journal on Robotics and Autonomous Systems (RAS)*, 2012.

[138] K.M. Wurm, C. Stachniss, and W. Burgard. Coordinated Multi-Robot Exploration using a Segmentation of the Environment. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

[139] X. Xu. PathFinding.js. `http://qiao.github.io/PathFinding.js/visual/`. Accessed: 2018-06-01.

[140] B. Yamauchi. Frontier-based exploration using multiple robots. In *International Conference on Autonomous Agents*, pages 47–53, 1998.

[141] J. Yang, H. Li, D. Campbell, and Y. Jia. Go-icp: A globally optimal solution to 3d icp point-set registration. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 38(11):2241–2254, 2016.

[142] J. Zhang and S. Singh. LOAM: Lidar Odometry and Mapping in Real-time. In *Proc. of Robotics: Science and Systems (RSS)*, 2013.

[143] J. Zhang and S. Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, pages 1–16, 2016.

[144] X. Zheng, Z. Moratto, M. Li, and A. Mourikis. Photometric Patch-Based Visual-Inertial Odometry. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.