UNIVERSITÄT BONN

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

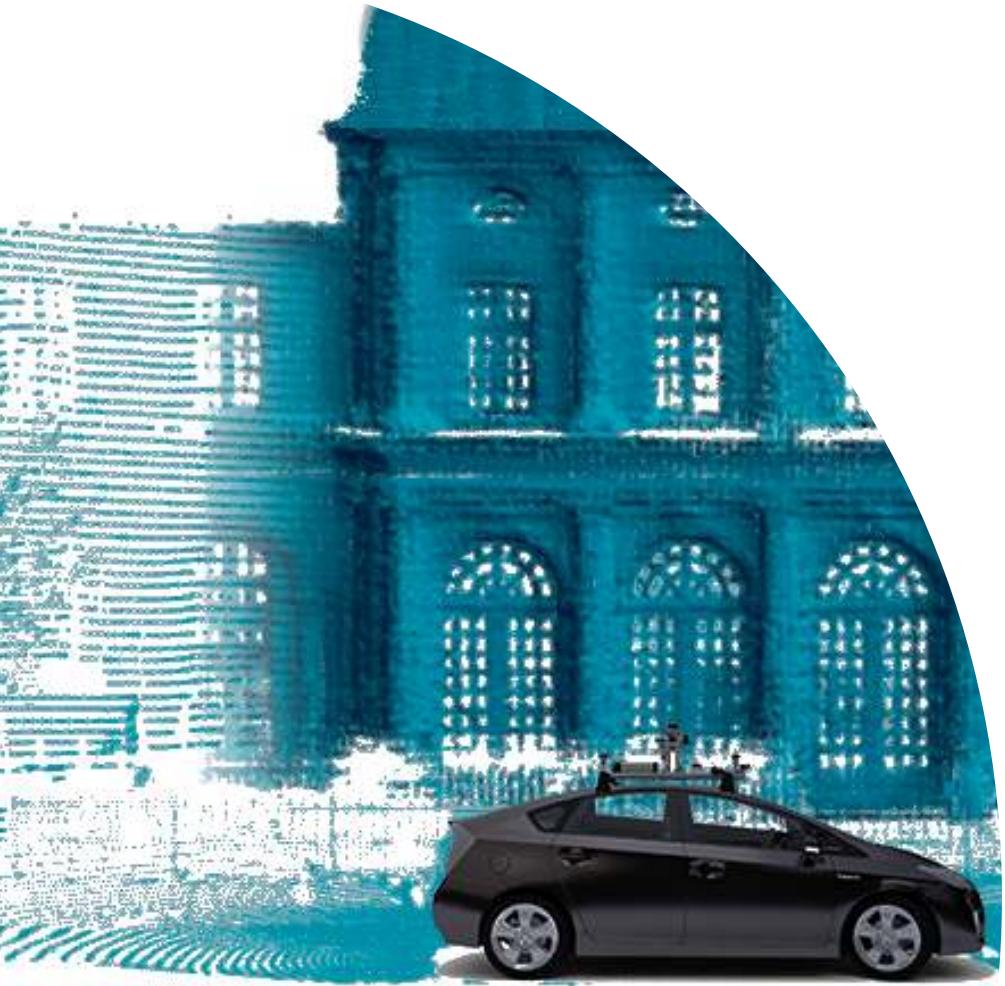
Robot Mapping with 3D LiDARs

von

Ignacio Martin Vizzo

aus

Ciudad Autónoma de Buenos Aires, Argentina



Bonn 2024

Referent:
Prof. Dr. Cyrill Stachniss, University of Bonn, Germany

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

Tag der mündlichen Prüfung: 01. December 2023

Angefertigt mit Genehmigung der Landwirtschaftlichen Fakultät der Universität Bonn

Zusammenfassung

Roboter sind in der Lage, Menschen auf unterschiedliche Arten zu unterstützen. Zum Beispiel können sie für uns Menschen unliebsame Aufgaben übernehmen, wie das tägliche Staubsaugen im eigenen Haushalt. Sie führen Kraftfahrzeuge, bei denen menschliche Fehler fatale Auswirkungen haben. Roboter ermöglichen uns, viele unserer heutigen Aufgaben effizienter und akkurater zu lösen. Ein Roboter kann zum Beispiel ununterbrochen große Lagerhäuser scannen und Informationen für eine optimierte nachfolgende Logistik liefern. Wir können Roboter auch auf anderen Planeten einsetzen, beispielsweise auf dem Mars, wo Bodenfahrzeuge das Gelände durchkämmen, Daten sammeln und zurück zur Erde schicken und uns somit Informationen über potenzielles Leben zugänglich machen.

Durch die hohe Komplexität der einzelnen Komponenten eines Robotersystems stellen uns diese Aufgaben vor große Herausforderungen. Ein Roboter ohne jegliches Vorwissen über seine Umgebung muss gleichzeitig ein Modell der Umgebung erstellen, seine Position innerhalb dieser bestimmen, sein Umfeld analysieren und eine effiziente Route zur Erkundung der teils noch unbekannten Umgebung entwickeln. Oft dient eine Karte als Repräsentation der Umgebung und liefert eine räumliche Darstellung des Gebiets, in der Hindernisse, Wege und andere wichtige Merkmale verzeichnet sind. Mit diesem Wissen kann ein Roboter effektiv navigieren, Kollisionen vermeiden und die oben genannten Aufgaben sicher ausführen.

Eine wesentliche Herausforderung stellt die dreidimensionale Welt dar, in der ein Roboter sich bewegt. Der Einsatz moderner Sensoren, wie zum Beispiel 3D-LiDARs, ist daher für reale Anwendungen in der Robotik unerlässlich. Durch die Nutzung von dreidimensionalen Daten können wir die Fähigkeiten und Anwendungsmöglichkeiten von mobilen Robotern erweitern und über die Grenzen des Möglichen hinaus verschieben.

Die zentrale Frage dieser Arbeit lautet: “Können wir anhand der Daten von 3D-LiDAR Sensoren abschätzen, wie unsere Umgebung aussieht?” Um diese Frage zu beantworten, entwickeln wir eine umfassende Pipeline für die 3D Kartierung. Wir präsentieren zunächst eine zuverlässige Strategie zur Erfassung

von Sensordaten aus der Umgebung. Dann stellen wir eine Methode vor, um die räumliche Bewegung der Sensoren in der Umgebung zu bestimmen. Schließlich untersuchen wir unterschiedliche Repräsentationen der Welt, die für verschiedene nachgelagerte Aufgaben wie z.B. Navigation, Lokalisierung oder Interpretation der Umgebung verwendet werden können. Mit den in dieser Arbeit vorgestellten Ideen können mobile Roboter selbständig 3D Karten erstellen, mit denen sie die Welt besser verstehen und in ihr navigieren können.

Die in dieser Arbeit beschriebenen Methoden leisten mehrere wichtige Beiträge zur Kartierung mit 3D-LiDAR Sensoren und erweitern den aktuellen Stand der Technik in Bezug auf Robustheit und Effizienz. Alle Beiträge wurden mit realen Datensätzen validiert und in Konferenzbeiträgen und Zeitschriftenartikeln nach dem Peer Review Verfahren veröffentlicht. Darüber hinaus wurden die Arbeiten als freie Software öffentlich zugänglich gemacht, um Transparenz zu gewährleisten und zukünftige Forschung zu erleichtern.

Abstract

Robots can assist humans in a multitude of ways. For example, robots can handle tedious tasks that humans prefer not to do, such as vacuum cleaning daily to keep a house clean. They can tackle challenging problems that, when attempted by humans, might result in fatal errors, such as driving a car. Furthermore, robots can perform tasks we already do but with greater efficiency and accuracy. An example of this could be a robot that constantly scans large warehouses, providing insights on optimizing logistics worldwide. Additionally, robots can be deployed to foreign planets like Mars, where rovers can traverse the terrain, collect data, and send it back to Earth, giving us insights into the potential viability of human habitation there.

Addressing these tasks effectively is a significant challenge due to the complex nature of each component that constitutes a robotics system. A robot without prior knowledge about its environment must simultaneously create a map, determine its location within that map, analyze its surroundings, and devise an efficient route to explore an unfamiliar environment. Often, a map serves as the robot's foundational understanding of its surroundings and provides a spatial representation of the area, identifying obstacles, paths, and other significant features. This knowledge is essential for the robot to effectively navigate, avoid collisions, and perform the aforementioned tasks strategically and safely.

In addition to these challenges, robots exist and navigate within a three-dimensional world. Consequently, using and exploiting modern sensors, such as 3D LiDARs, become essential in tackling real-world robot applications. By relying on 3D data, we can expand mobile robots capabilities and potential applications, pushing the boundaries of what they can accomplish.

The central question of this thesis is: "Can we estimate what the world looks like based on sensor data from 3D LiDARs?" To answer this question, we develop a comprehensive 3D mapping pipeline. We first propose a reliable mechanism to collect data from the real world. Second, we introduce a method to understand the spatial movement of the sensors within the world. Finally, we investigate diverse world representations for different downstream robotic tasks, such as navigation, localization, scene understanding, and others. The ideas presented in this

thesis empower mobile robots to create 3D maps on their own, allowing them to understand and navigate the world more effectively.

The work described in this thesis makes several significant contributions to robot mapping using 3D LiDARs. As a result, this work advances the current state of the art regarding robustness and efficiency in robot mapping with 3D LiDARs. All contributions have been validated with tests on real-world datasets, undergone rigorous scientific review, and published in conference papers, workshop papers, and journal articles, all subject to peer review. Furthermore, these contributions have been made publicly available as open-source software to promote transparency and facilitate further research.

Acknowledgements

Embarking on the path toward a Ph.D. is unquestionably a challenging endeavor. Only those unfamiliar with its intricacies might think such a monumental task could be accomplished alone. In this chapter, I would like to thank everyone who has accompanied me during the last five years, making this thesis possible.

First and foremost, I would like to express my gratitude to my advisor, Prof. Dr. Cyrill Stachniss, for his unconditional support, insightful guidance, and constant encouragement throughout my journey. His deep understanding of the subject matter, together with his meticulousness, helped shape my academic perspective. He not only offered me academic wisdom but also taught me to persist and never give up in the face of challenges. Without any doubt, he is the academic advisor I wish everyone had.

I also want to thank those who paved the way for me to embark on my Ph.D. journey. I thank Mario Munich for giving me my first opportunity in robotics and Vittorio Ziparo for being a great leader and friend during this time. I would also like to thank Andres Milioto for helping me in all possible ways at the beginning of my Ph.D. career, making his home mine, and for countless mate [129] sessions, where every possible topic in life was for debate.

Sharing an office with someone like myself poses an equally challenging experience as pursuing a Ph.D. For this, I would like to thank all those who have tolerated my screams and curses in four languages during the last few years. I thank my friend Lorenzo Nardi for being supportive during the beginning of my Ph.D. I thank Irvin Aloise for teaching me many ways to express my deepest thoughts in Italian during his short stay in Bonn. And lastly, Benedikt Mersch, for his exceptional patience, countless academic and non-academic discussions we shared in the office, for blossoming into a true friend, and for being my go-to person for anything related to German matters.

I also want to thank all my colleagues and lab mates. I am incredibly fortunate to be surrounded by such remarkable individuals, each unique, who provide unconditional support and motivation in my professional and personal life. Although the list of names should be longer, I would like to mainly express my grat-

itude for the moments I have shared with Daniel Casado, Elias Marks, Federico Magistri, Hyungtae Lim, Louis Wiesmann, Luca Lobefaro, Lucas Nunes, Meher Malladi, Nived Chebrolu, Olga Vysotska, Rodrigo Marcuzzi, Xieyuanli (Rhiney) Chen, and Yue (Linn) Chong. I would also like to thank Igor Bogoslavskyi for being a remarkable Ph.D-uncle and always being supportive and ready for help, especially during the last time of my Ph.D. In addition to all my colleagues, I want to thank Jens Behley for the uncountable late nights of writing scientific papers together, for his invaluable help in thoroughly reviewing and improving the manuscripts before submission, and for consistently serving as reviewer two. I also express my sincere appreciation to Birgit Klein, whose invaluable assistance with administrative matters has spared me from the potential difficulties and frustrations that often accompany such tasks.

Over the past five years, I have had the opportunity and the challenge of teaching many students. What I did not anticipate was the immense amount of knowledge and wisdom I would acquire from my students. I thank in particular the now friends Alessandro Riccardi, Andrzej Reinke, Dhagash Desai, Saurabh Gupta, Sumanth Nagulavancha, Vardeep Singh Sandhu, Lukas Arzoumanidis, Preetham Guruprasad Yashoda, and Yash Goel for always being proactive and pushing the barriers of what I can teach further. Thank you for being my fuel to keep running and keeping me motivated on my academic path when everything seemed to fall apart.

Living in Germany for an extended period has blessed me with numerous new friendships and a sense of belonging, similar to having an extended family. In addition to the people mentioned above, I would like to thank all my German friends, Aurélie Lehmann and Fabian Waser, Angelika Schmitz, Alice Matheis and Christian Kunze, Leonardo and Sol Everling, Stefan Nelles and Joana Dionisio, and Xavier and Mariele Donoso Olave. Additionally, I would like specially express my gratitude to Violette Schmidt for making me and my wife part of her family. I also would like to thank all the friends of the large Spanish-speaking community of Bonn. In particular, I would like to thank Nàdia Villacampa and Martin Cándido Tejeda, Maria Vicente and Rodrigo Ariza, Clara Lucía Urbina, and Alvaro Peralta. Without this extended family in Germany, life outside of work would have been unfortunate. Their friendship made my journey even more vibrant and meaningful.

I also want to thank my family for their support and guidance. Thanks to their constant encouragement and advice, I have been able to seize the opportunities that have led me to where I am today. Words cannot adequately transmit the depth of my gratitude for the countless possibilities they enabled me, allowing me to embrace and enjoy my Ph.D. journey. I also want to extend this acknowledgment to all my friends from Argentina, who constantly supported me despite

the enormous distance and never let me down.

Despite not being a conventional acknowledgment, I must express my gratitude to my therapist, Mariano Scarone. His guidance and support have been instrumental in helping me navigate the challenges of my Ph.D. journey and preventing burnout. I am immensely grateful for his spiritual guidance and its profound impact on my well-being and overall success.

This section would not be complete without mentioning "il mio fratello", Tiziano Guadagnino, who became my close friend, mentor, and guide over the last few years. Thank you for teaching me to trust myself again once everyone else convinced me of the opposite. Without his constant drive for scientific excellence and his relentless push to challenge my limits, a significant portion of the work I have accomplished for this thesis would not have been possible.

Lastly, I want to express my deepest gratitude to my beloved wife Mailén Raposo, who has been by my side throughout this long and challenging journey we call life. Words cannot adequately capture my immense appreciation and love for her support, understanding, and patience. If I were to dedicate one page for each page written in this thesis, it still wouldn't be enough to convey the depth of my gratitude. She often says she could write a book on "How to Accompany a Ph.D. Student Without Losing Your Mind", given her incredible support and understanding throughout this journey. Additionally, I would like to thank her for generously designing the city-like CAD model used to generate the MaiCity dataset. This dataset has played an essential role in numerous recent research endeavors, particularly in evaluating robot mapping techniques. Still, many researchers in the field have yet to discover that the name of the dataset is inspired by her first name, precisely for this reason.

The work presented in this thesis is partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy, EXC-2070 – 390732324 – PhenoRob, by the European Union's HORIZON research and innovation programme under grant agreement No 101070405 (Digiforest) and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017008 (Harmony). The financial support is gratefully acknowledged.

For my father, may he rest in peace.

Contents

Acronyms

CAD computer-aided design

CI continuous integration

CI/CD continuous integration/continuous deployment

CNN convolutional neural network

DDA differential digital analyzer

fps frames per second

GbE Gigabit Ethernet

GNSS global navigation satellite system

GPIO general-purpose input/output

GPS global positioning system

GPU graphics processing unit

HD high-definition

ICP iterative closest point

IMU inertial measurement unit

LiDAR laser imaging, detection, and ranging

LIO LiDAR-Inertial odometry

LOAM LiDAR odometry and mapping

MCL Monte Carlo localization

OS operating system

PSR Poisson surface reconstruction

PTP precise time protocol

RADAR radio detection and ranging

RMSE root mean square error

ROS robot operating system

RTK real-time kinematic

SDF signed distance field

SDK software development kit

SLAM simultaneous localization and mapping

TSDF truncated signed distance field

UAV unmanned aerial vehicle

URDF unified robotics description format

Chapter 1

Introduction

Many tasks humans perform regularly are mundane, repetitive, or dangerous. Typical examples of such tasks range from driving a car to keeping a house clean daily. Mobile robots have the potential to assist or even substitute humans in such tasks. For example, they can be deployed on distant planets such as Mars, where rovers can traverse the terrain, gather data, and transmit the gathered information back to Earth. This type of robot could provide valuable insights into the feasibility of human settlement outside our planet. Moreover, robots can perform mundane tasks that humans generally find unappealing, such as cleaning the house daily. They can also handle complex challenges that, if attempted by humans, may result in fatal errors, such as driving a vehicle. Furthermore, robots can perform tasks we already know how to do, but they can excel in efficiency, precision, and the ability to carry out these tasks continuously without any breaks. For example, consider a robot that constantly scans large warehouses without interruption. This type of robot can provide valuable information to optimize space utilization and increase warehouse logistics efficiency on a global scale. An illustration of how real-world robots look today is shown in Fig. 1.1.

A crucial requirement for accomplishing all these tasks is having a map of the environment that allows the system to determine its location accurately and facilitates other robotic tasks such as navigation, exploration, and others. These downstream tasks build upon/use the map learned by the robot. Therefore, mapping is one of the canonical problems that a mobile robot must solve to navigate the surrounding environment safely. Take, for example, the autonomous car shown in Fig. 1.1 (c) moving from one location to another. An accurate map of the environment allows the vehicle to understand the spatial layout, identify possible paths, and calculate the optimal route to a destination. This is particularly crucial in real-world applications such as autonomous driving, where vehicles must safely navigate through complex environments. Therefore, robots

(a) Mars Rover Explorer[1]

(b) Vaccum Cleaner[2]

(c) Autonomous Car

(d) Warehouse Robot[3]

Figure 1.1: Examples of modern mobile robots in action. In (a), we can see a Mars rover designed to explore foreign planets, (b) shows the trajectory traversed by an autonomous vacuum cleaner within a house after a completed cleaning mission, (c) depicts the sensory platform used by a modern autonomous car, and (d) shows an autonomous warehouse robot that scans a large warehouse non-stop.

that can create accurate maps of their surroundings are crucial to achieve genuine autonomy, thus making robot mapping a relevant and essential field of study.

Robots are typically equipped with different exteroceptive sensors, such as cameras, RADARs, and LiDARs. These sensors are used by robots to learn how the world looks. In this thesis, our focus is solely on 3D laser imaging, detection, and ranging (LiDAR) sensors, as such sensors exhibit many advantages for mapping. For instance, these sensors provide precise long-distance range measurements of the surrounding environment, surpassing other ranging technologies such as radio detection and ranging (RADAR) sensors or RGB-D cameras. In addition, modern 3D LiDARs offer a 360-degree view of the environment at high frequency. Furthermore, the sensor output directly represents the three-dimensional world, eliminating the need for additional steps required with cameras. Moreover, LiDAR sensors are illumination invariant.

Building robot maps using sensor data is a complex task that involves sev-

[1]Image courtesy of NASA – `https://photojournal.jpl.nasa.gov/jpegMod/PIA04413_modest.jpg`.
[2]Image courtesy of Chris Bartle – `https://www.flickr.com/photos/13963375@N00/3533146556`.
[3]Image courtesy of Dexory – `https://www.dexory.com`.

eral challenges. Some of these challenges include accurately capturing data and figuring out how to combine it into a unified representation of the surrounding environment. Moreover, it requires significant computational resources, but robots often have limited processing power, memory, and energy constraints. Therefore, efficient algorithms and techniques need to be developed to optimize available resources and operate minimizing delays. In this thesis, we introduce several techniques to overcome the challenges that arise when a robot maps the world with LiDAR sensors. We introduce a new methodology to build reliable perception platforms to capture sensor data and develop techniques for generating various map representations.

In modern robotics systems, maps are constructed either offline or online. *Offline* mapping systems learn the map *after* a data collection phase and can afford higher computation times. These maps typically have a higher level of detail and are used for other robotic tasks, such as place recognition and localization. *Online* mapping techniques require the creation of a map *while* the robot is performing a mission. Therefore, the map must be made available in (near) real-time to support other downstream tasks, such as path planning and exploration. As a consequence, the choice of the mapping approach depends on the specific requirements of the task at hand.

A common thread connecting the methods explored in our work is the need for information about their environment in the form of a map. However, no single map representation satisfies all the tasks' requirements. For example, in the context of pose estimation, a point cloud map representation allows to accurately estimate the position and orientation of the sensor within its environment. A mesh representation allows for complex surface representations, which is beneficial when dealing with localization tasks. Furthermore, a voxel-based map is helpful for autonomous navigation, as it allows to represent distances from obstacles.

This thesis presents different techniques that we developed for mobile robots to autonomously produce accurate models of the environment from sensor data in both online and offline scenarios. Our approach to solving these problems is unique in that, unlike the standard practice in current research, we focus on systems capable of operating in previously unseen situations. Take, for example, the Mars rover shown in Fig. 1.1 (a). Most of the environment it needs to map remains unexplored by humans, leaving little information regarding its specific characteristics and appearance. Consequently, this thesis emphasizes the generalization of the techniques presented. A fundamental approach to accomplishing this generalization capability is to create functional robot software that extends beyond the scope of scientific papers. Thus, this thesis incorporates meticulously designed open-source and freely available software alongside this manuscript.

In sum, this work presents a methodology for autonomously creating 3D mod-

els of the environment using robots. Our approach enables us to create efficient 3D maps for various robotic tasks. In this context, we showcase advancements in robot mapping with 3D LiDAR sensors.

1.1 Map Representations in Robotics

Our work explores different map representations, including point clouds, triangle meshes, and voxel grids. Each map representation poses distinct advantages and disadvantages that we exploit accordingly to solve various robotic tasks. These map representations are visually illustrated in Fig. 1.2.

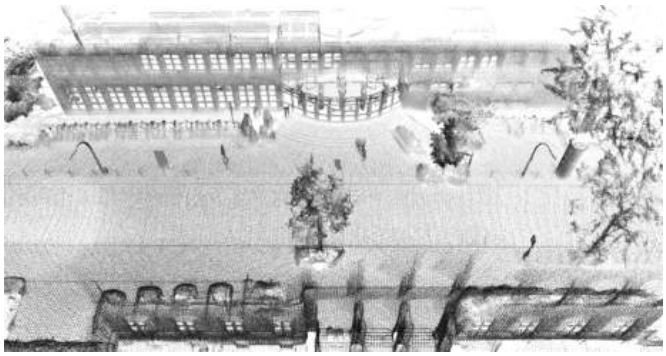
Point clouds are typically constructed by merging sensor data into a common reference frame. An illustration of such a point cloud map is shown in the uppermost image of Fig. 1.2. This map representation is straightforward to implement, as it directly aggregates input scans. Modern 3D LiDAR sensors produce high-resolution point clouds with reduced noise compared to RGB-D sensors, and as a result, these maps capture fine-grained geometric details of the surrounding environment. Additionally, the model resolution is only limited by the floating-point precision of the computing platform. However, these maps require significant memory resources and cannot encode occupancy information.

Triangle meshes, as depicted in the middle section of Fig. 1.2, are well suited to represent complex surfaces and can improve memory efficiency by eliminating redundant measurements found in point clouds. Furthermore, they provide a smooth, potentially watertight model of the environment. However, their computation often requires significant computational resources. Additionally, updating the current model with recent scans is not straightforward: Recomputing the entire mesh is typically required for updating the map. This operation is computationally expensive and cannot be used for real-time applications.

Voxel grids, shown at the bottom of Fig. 1.2, provide a 3D grid-based model that facilitates spatial reasoning and enables efficient access, making this representation attractive for various robotic tasks. This map representation enables more varied usages, as it can store occupancy information, signed distance to the closest surface, semantic information, or other feature types. However, it is essential to note that voxel grids can consume more memory than point cloud maps and triangle meshes if not implemented properly, and they can also introduce discretization errors in the representation.

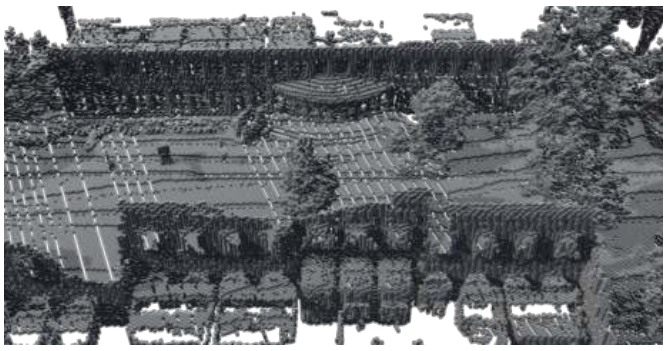
In essence, each representation has strengths and weaknesses regarding memory usage, resolution, and computational complexity. These factors should be taken into account according to the specific task requirements. This thesis works with different map representations for various robotic tasks. In Chapter 4, we employ point cloud maps to address the registration problem. In Chapter 5, we

leverage triangle meshes to generate high-resolution 3D offline maps, which can be used to tackle robot localization, as we demonstrate in Chapter 6. Finally, in Chapter 7, we utilize voxel grids to generate 3D online maps, a representation that we further extend for operation with low-resolution sensors in Chapter 8.



(a) Point Cloud Map

(b) Triangle Mesh Map



(c) Voxel Grid Map

Figure 1.2: Different map representations of the facade of the "Anatomisches Institut der Universität Bonn" in Germany. (a) The topmost illustration employs a point cloud map as the primary map representation for the robot. It exhibits a high level of geometric detail. (b) In the middle, we showcase the triangle mesh representation of the same building. This model shows smoother contours and eliminates redundant measurements, improving memory efficiency. Finally, in (c) at the bottom, a voxel grid serves as the map representation. Each voxel stores the signed distance to the closest surface on the map, allowing for clear differentiation between explored and unexplored space.

1.2 Main Contributions

This thesis presents novel solutions to robot mapping using 3D LiDARs. It contributes to multiple aspects of 3D mapping tasks, data collection, pose estimation, data representation, and surface reconstruction. This section summarizes the thesis' significant achievements and practices contributing to the state of the art in robotics.

The first contribution of this thesis is a systematic and practical approach that improves existing perception platforms used for data collection [192]. In essence, the creation of reliable 3D maps for robotics heavily depends on the reliability of the perception platform in use. Any errors or inconsistencies in the data-gathering process can lead to inaccuracies in the representation of the 3D space. We present our method to achieve reliable data recordings in Chapter 3. Our system can record data while logging the exact state of the sensor platform, used software, and parameter settings simultaneously, making the setup and the data generated reliable and reproducible.

The second contribution of this thesis is in the context of pose estimation. Our method [191] advances state of the art by systematically analyzing the required components and focusing on reducing them to their bare minimum essentials. In contrast to recent developments in the field, we build on top of the first publication in the point cloud registration domain: the classical iterative closest point (ICP) algorithm, introduced more than 30 years ago by Besl and McKay [11]. Our system performs extraordinarily well in various real-world scenarios without requiring us to make assumptions about the sensor in use or the environment. The proposed registration method is presented in-depth in Chapter 4.

The third contribution of this thesis is an offline 3D mapping method that produces high-resolution triangle meshes [188]. Building on the foundations of another computer graphics technique, we extend the Poisson surface reconstruction (PSR) approach to the robotics domain. Our system produces a high level of detail in 3D maps. Furthermore, we present a novel registration approach to register point clouds to triangle meshes to refine the pose estimates. Our approach to offline 3D mapping is described in detail in Chapter 5. We also show how to exploit the generated maps to build a global localization pipeline based on triangle meshes in Chapter 6.

The fourth contribution of this thesis targets 3D online mapping. Our method [190] extends traditional volumetric fusion pipelines [44] to large outdoor environments. To achieve this, we exploit an existing technique used in the filmmaking industry that is yet to be fully discovered by the robotics community, the VDB data structure [125]. Using VDBs, we build an efficient 3D online mapping system for mobile robots. Our approach stands out from the

state of the art as it eliminates the need to impose spatial restrictions due to memory limitations. Additionally, we do not require prior knowledge of the volume to be mapped, a common limitation in previous works [61, 187]. Our method is strictly geometric and operates at the frame rate of the sensor, enabling deployment in real-world applications. Details on exploiting this data structure to build 3D online mapping systems are described in Chapter 7.

Our fifth contribution is based on the system introduced in Chapter 7 and expands our contributions to 3D mapping when using low-resolution sensors. By exploiting a 3D convolutional neural network (CNN) to perform geometric completion, we obtain dense 3D maps even when employing such sensors [189]. This novel work enables the capabilities of 3D mapping systems for robots that use low-cost sensors. The method is described in-depth in Chapter 8.

Our last contribution is a set of open-source software tools that represent the practical manifestation of the research findings. A clear message resonates throughout this thesis: robots are driven by software, not merely equations. Our focus shifts from purely theoretical formulations to the practical implementation and execution of algorithms. Recognizing that robots are complex systems operating in real-world environments, our work emphasizes developing robust, efficient, and adaptable software frameworks to enable their functionality. This shift in perspective acknowledges the importance of software as the backbone of robotic systems. By embracing this understanding, the thesis highlights the importance of software development in unlocking the true potential of robots. Moreover, considerable effort has been invested in ensuring that the open-source contributions are user-friendly, as accessibility and ease of use are paramount for pushing the boundaries of research and fostering wider adoption and innovation in the field.

In sum, this thesis presents five contributions, each of which targets an essential part of the robot mapping pipeline. All the methods presented in this thesis have been published in peer-reviewed international conferences, workshops, and journals, all of them are open-source software, and all the methods presented here are tested on real-world data. In addition, the methods introduced are also capable of processing data from modern 3D LiDARs at the sensor frame rate.

1.3 Thesis Organization

This thesis is organized as follows. In Chapter 2, we present an overview of the current state of the art in 3D robot mapping. In Chapter 3, we introduce a modern infrastructure for reliable data collection, a prerequisite for any mobile robot that aims to gather sensor data. In Chapter 4, we focus on efficiently solving the point cloud registration problem and demonstrate that our solution can provide

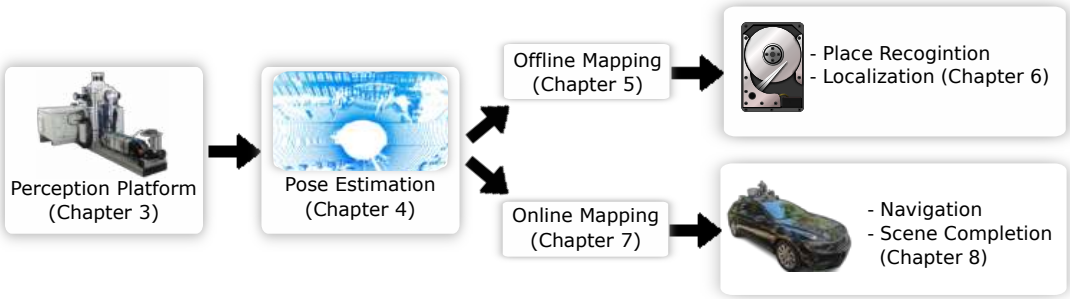


Figure 1.3: Overview of different techniques explored in this thesis. We start providing a methodology for reliable data collection in Chapter 3, following that, in Chapter 4, we introduce an efficient and generic pose estimation approach. By relying on this method, we explore different mapping approaches. In Chapter 5, we investigate how to create offline maps that can be consumed by tasks such as place recognition or localization (Chapter 6). Furthermore, in Chapter 7, we introduce an online mapping system for a mobile robot while navigating the environment. We later extend this system with low-resolution sensors in Chapter 8.

accurate pose estimates for different robots and scenarios. By providing a general solution to the pose estimation problem, we can rely on this component to independently create 3D maps of the environment by integrating sensor data into different map representations. Therefore, we demonstrate how to take advantage of these poses to build 3D maps in both offline (Chapter 5) and online (Chapter 7) settings. Furthermore, we show in Chapter 6 an application on how to exploit offline maps to solve the global localization problem. Furthermore, in Chapter 8, we show how to extend online mapping techniques to operate with low-resolution LiDAR sensors.

1.4 Publications

Parts of this thesis have been published in peer-reviewed journals, conferences, and workshop papers.

- I. Vizzo, X. Chen, N. Chebrolu, J. Behley, and C. Stachniss. Poisson Surface Reconstruction for LiDAR Odometry and Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021

- X. Chen, I. Vizzo, T. Läbe, J. Behley, and C. Stachniss. Range Image-based LiDAR Localization for Autonomous Vehicles. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021

- I. Vizzo, B. Mersch, R. Marcuzzi, L. Wiesmann, , J. Behley, and C. Stachniss. Make It Dense: Self-Supervised Geometric Scan Completion of Sparse 3D Lidar Scans in Large Outdoor Environments. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):8534–8541, 2022

- I. Vizzo, T. Guadagnino, J. Behley, and C. Stachniss. VDBFusion: Flexible and Efficient TSDF Integration of Range Sensor Data. *Sensors*, 22(3):1296, 2022

- I. Vizzo, B. Mersch, L. Nunes, L. Wiesmann, T. Guadagnino, and C. Stachniss. Toward Reproducible Version-Controlled Perception Platforms: Embracing Simplicity in Autonomous Vehicle Dataset Acquisition. In *Worshop on Building Reliable Ratasets for Autonomous Vehicles, IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*, 2023

- I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss. KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1029–1036, 2023

1.5 Further Scientific Contributions

The following peer-reviewed journal and conference publications in which I was involved during my doctorate as a collaborator are not part of this thesis.

- A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019

- X. Chen, B. Mersch, L. Nunes, R. Marcuzzi, I. Vizzo, J. Behley, and C. Stachniss. Automatic Labeling to Generate Training Data for Online LiDAR-Based Moving Object Segmentation. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):6107–6114, 2022

- F. Magistri, E. Marks, S. Nagulavancha, I. Vizzo, T. Läbe, J. Behley, M. Halstead, C. McCool, and C. Stachniss. Contrastive 3D Shape Completion and Reconstruction for Agricultural Robots using RGB-D Frames. *IEEE Robotics and Automation Letters (RA-L)*, 7(4):10120–10127, 2022

- R. Marcuzzi, L. Nunes, L. Wiesmann, I. Vizzo, J. Behley, and C. Stachniss. Contrastive Instance Association for 4D Panoptic Segmentation for Sequences of 3D LiDAR Scans. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022

- B. Mersch, X. Chen, I. Vizzo, L. Nunes, J. Behley, and C. Stachniss. Receding Moving Object Segmentation in 3D LiDAR Data Using Sparse 4D Convolutions. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):7503–7510, 2022

- B. Mersch, T. Guadagnino, X. Chen, Tiziano, I. Vizzo, J. Behley, and C. Stachniss. Building Volumetric Beliefs for Dynamic Environments Exploiting Map-Based Moving Object Segmentation. *IEEE Robotics and Automation Letters (RA-L)*, 8(8):5180–5187, 2023

- L. Wiesmann, T. Guadagnino, I. Vizzo, G. Grisetti, J. Behley, and C. Stachniss. DCPCR: Deep Compressed Point Cloud Registration in Large-Scale Outdoor Environments. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):6327–6334, 2022

- L. Wiesmann, T. Guadagnino, I. Vizzo, N. Zimmerman, Y. Pan, H. Kuang, J. Behley, and C. Stachniss. LocNDF: Neural Distance Field Mapping for Robot Localization. *IEEE Robotics and Automation Letters (RA-L)*, 8(8):4999–5006, 2023

1.6 Open Source Contributions

This thesis resulted in the release of several open-source packages and datasets. Additionally, parts of the modules developed for the methods and the experiments were made open source not as stand-alone packages but as contributions to existing open-source libraries, such as CARLA, NVdiffrast, ONXX-tensorrt, Open3D, Open3D-ML, OpenVDB, PyMeshFix, Sophus, and others. Some highlighted contributions are:

- **KISS-ICP** C++, Python, and ROS 1/2 package, presented in Chapter 4: `https://github.com/PRBonn/kiss-icp`

- **VDBFusion** C++ and Python library, presented in Chapter 7: `https://github.com/PRBonn/vdbfusion`

- **VDBFusion** ROS 1 package, presented in Chapter 7: `https://github.com/PRBonn/vdbfusion_ros`

- **make-it-dense** scene completion network, presented in Chapter 8: `https://github.com/PRBonn/make_it_dense`

- **PUMA** 3D offline mapping pipeline, presented in Chapter 5: `https://github.com/PRBonn/puma`

- **range-mcl** Range image localization code, presented in Chapter 6 `https://github.com/PRBonn/range-mcl`

- **Mai City** Synthetic Dataset, presented in Chapter 5: `https://www.ipb.uni-bonn.de/data/mai-city-dataset/`

- **lidar_visualizer**, Visualization tool for LiDAR data: `https://github.com/PRBonn/lidar-visualizer`
- **vdb_to_numpy**, Python bindings to work with VDBs: `https://github.com/PRBonn/vdb_to_numpy`
- **voxblox_pybind**, Python bindings for the Voxblox library `https://github.com/PRBonn/voxblox_pybind`
- **voxblox_pybind**, Python bindings for the Voxblox library `https://github.com/PRBonn/voxblox_pybind`
- **manifold_python**, Python bindings for the Manifold library `https://github.com/PRBonn/manifold_python`
- **ros_in_docker**, Container library to use with ROS `https://github.com/nachovizzo/ros_in_docker`
- **Open3D** C++, and Python Robust Kernel Library: `http://www.open3d.org/docs/release/tutorial/pipelines/robust_kernels.html`
- **Open3D** C++, and Python Generalized ICP: `https://github.com/isl-org/Open3D/pull/3181`

Chapter 2

Related Work

Robot mapping typically comprises two main components: pose estimation and, map building and updating. Pose estimation determines the robot's position and orientation within an environment. Map building and updating involves modifying the current environment representation using newly acquired sensor data, assuming the current pose is already known. This thesis mainly focuses on the map update part using 3D LiDAR sensors. However, we also provide a widely applicable pose estimation method to relax the requirement of knowing the robot poses a priori.

The techniques investigated in this thesis build upon established methods widely recognized in robotics; therefore, we do not delve into the basics extensively. Instead, we rely on the following literature sources to satisfy the requirements for understanding the text: (i) For simultaneous localization and mapping (SLAM), readers can refer to an introduction to robotic exploration and mapping provided by Stachniss [68, 168], (ii) Pomerlau et al. [141] reviewed point cloud registration algorithms in the context of pose estimation, and (iii) Berger et al. [10] surveyed 3D surface reconstruction, presenting a thorough task description. While not strictly necessary, readers can also refer to a brief introduction to convolutional neural networks by O'Shea et al. [131] to better understand the concepts and techniques discussed in Chapter 8 of this thesis. By consulting these key sources, the reader can grasp the methods presented without needing explicit coverage of the basic techniques.

In this chapter, we provide an overview of the current advancements in 3D mapping within the field of robotics. Specifically, in Sec. 2.1, the latest developments in pose estimation through point cloud registration techniques are presented. Following that, in Sec. 2.2, we briefly explore various data representations and their use in mapping systems and analyze recent advances in 3D surface reconstruction. Lastly, Sec. 2.2.1, explores related approaches that exploit robot maps to solve the localization task.

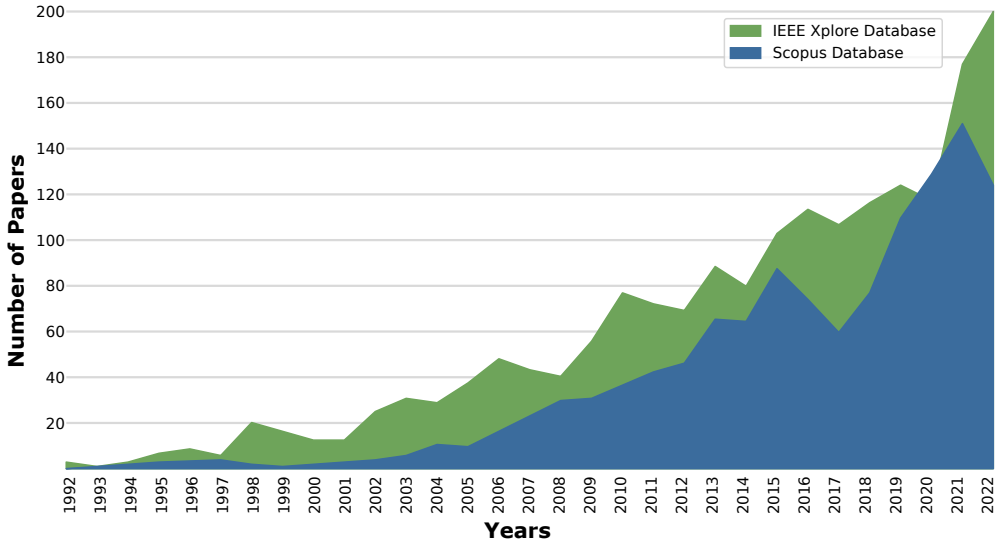


Figure 2.1: The progression of publication numbers over the years was examined, explicitly focusing on publications containing the keywords "iterative closest point", "ICP", or "point cloud registration" in either the title, abstract, or keyword. The dark blue area represents data gathered from the IEEE Xplore database, while the light green area represents data from the Scopus database. This figure can be considered an extension, ten years later, of an analysis presented by Pomerlau et al. [142].

2.1 Point Cloud Registration in Robotics

Point cloud registration has been an active area of research for the last decades [11, 41, 76, 141] and is still relevant today [49, 52, 53, 102, 133, 134, 145, 158, 163, 205, 219]. In Fig. 2.1 we examined the progression of the number of publications over the years, specifically those that include words such as "iterative closest point", "ICP", or "Point Cloud Registration" in their titles, abstracts or keywords. Fig. 2.1 extends a previous analysis conducted by Pomerlau et al. [141] a decade ago. The figure illustrates the increasing interest of the research community in point cloud registration, thereby highlighting that even three decades after its initial publication [11], point cloud registration remains a vibrant field of study.

The ICP algorithm [11] can solve the problem of finding a transformation that brings two different point clouds into a common reference frame, and it is a particular case of the absolute orientation problem in photogrammetry [59]. ICP typically consists of two parts. The first one is to find correspondences between the point clouds. The second one computes the transformation that minimizes an objective function defined on the correspondences from the first step. This process is repeated until a convergence criterion is met. Most ICP variants [6, 52, 53, 133, 163, 219] utilize a maximum distance threshold in the data association module in addition to a robust kernel [2, 29] and a maximum number of iterations.

In contrast, in this thesis, we propose a threshold estimation method that adapts to changing scenarios by reasoning about the system kinematics and the nature of the data in combination with a robust kernel. Additionally, we avoid controlling the number of iterations of the ICP to achieve better generalization.

Computing the relative transformation between two successive LiDAR scans through ICP allows us to employ this technique as an odometry source. Therefore, ICP and LiDAR odometry are used somewhat interchangeably in this thesis. In the LiDAR sensing domain, current odometry pipelines typically use some form of ICP to estimate poses incrementally [52, 122, 163, 188, 219]. Although LiDAR odometry has been an active area of research for the last three decades [72, 103, 104, 105, 181], the design of current systems is generally coupled with assumptions about robot motion [52] and the structure of the environment [165] to achieve accurate and robust alignment results. To the best of our knowledge, no existing 3D LiDAR odometry approach is free of parameter tuning and works out of the box in different scenarios, using arbitrary LiDAR sensors, supporting different motion profiles, and consequently, types of robots, such as ground and aerial robots.

The majority of modern SLAM systems build on top of odometry algorithms. Zhang et al. [219] proposed LiDAR odometry and mapping (LOAM) that computes the robot's odometry by registering the planar and edge features to a sparse feature map. LOAM inspired numerous other works [164, 194], such as Lego-LOAM [165], which adds ground constraints to improve accuracy, and recently F-LOAM [194], which revised the original method with a more efficient optimization technique enabling faster operation. However, these methods rely on hand-tuned feature extraction, which typically requires tedious parameter tuning that depends on sensor resolution, environment structure, etc. In contrast, our method relies only on point coordinates, which removes this data-dependent parameter adaptation.

Behley and Stachniss [6] propose the surfel-based method SuMa to achieve LiDAR odometry estimation and mapping. It has also been extended to consider semantics [38] and explicitly handle dynamic objects [36]. In contrast to surfel-based mapping, Deschaud [53] introduced IMLS-SLAM [53] selecting an implicit moving least squares surface [92] as the representation of the map. Along a similar research line, we present an approach to register point clouds to 3D maps as triangle meshes in Sec. 5.2.4. Compared to our main contribution to LiDAR odometry (Sec. 4.1), this method does not run in real-time. These approaches are based on a point-to-plane [154] metric to register consecutive scans. This requires normal estimation, which introduces additional data-dependent parameters. Furthermore, noisy 3D information can negatively impact normal computation and registration. By minimizing a more straightforward point-to-point metric, we will

show that our method obtains on-par or better odometry performance. Moreover, this design choice enables us to represent the internal map as a voxelized, downsampled point cloud, simplifying the implementation.

Several new approaches [52, 133, 164] have been proposed to solve the odometry estimation problem. Most of these works focus on the runtime operation of the system and its accuracy. Pan et al. [133] propose a multi-metric approach (MULLS) that obtains good results in many challenging scenarios at the cost of tuning many parameters for each run. Dellenbach et al. [52] introduced CT-ICP, incorporating motion un-distortion into the registration, showing excellent results but adding more complexity. Additionally, the robots' motion profile must be known a priori. We challenge the need for sophisticated optimization techniques to cope with motion distortion that requires only the constant velocity model [182]. Furthermore, our system is based only on a few parameters, and we do not need to know the motion profile in advance.

A current trend in LiDAR odometry is to incorporate information from inertial measurement units (IMUs) [3, 32, 33, 99, 164, 198, 210, 212, 214, 220], a technique commonly referred to as LiDAR-Inertial odometry (LIO). An early contribution to this field is the work of Ye et al. [214], who introduced a tightly coupled LiDAR-IMU fusion method to minimize the cost derived from LiDAR and IMU measurements jointly. A more recent advancement in the field is the development of LIO-SAM [164], which employs a tightly-coupled, factor-graph optimized LIO framework to deliver superior odometry accuracy, particularly in challenging dynamic environments. In contrast to these trends, this thesis refrains explicitly from incorporating data from IMUs, as it focuses solely on the principles of LiDAR-based odometry, rendering the use of IMUs outside the scope and intent of the present investigation.

Many state-of-the-art systems [6, 52, 133, 164] also rely on pose graph optimization [51, 68, 83, 95] to achieve better alignment. In contrast, we do not exploit such techniques and state that pose graph optimization is orthogonal to the concepts presented in this thesis. If desired, the registration algorithm introduced in this thesis can be seamlessly integrated into a pose graph framework.

In sum, we step back from the common mainstream work on LiDAR odometry and propose a system that does not employ pose graph optimization or rely on any other external source such as IMUs. Our approach in Chapter 4 relies on point-to-point ICP combined with adaptive thresholding for correspondence matching, a robust kernel, a simple but widely applicable motion compensation approach, and a point cloud subsampling strategy. This yields a system with only a few parameters that, in most cases, do not have to be tuned to a specific LiDAR sensor. The proposed method runs online on mobile robots, handheld devices, and other platforms without fine-tuning the system for a particular application.

2.2 Map Building in Robotics

3D surface reconstruction is a field in photogrammetry, computer vision, and robotics and has been an active area of research for the last three decades [10]. 3D scene reconstruction involves techniques for interpreting different types of data, including 2D images, depth images, or point clouds gathered from multiple viewpoints or sensors to generate a comprehensive and geometrically accurate three-dimensional model of the environment.

The classical volumetric integration method introduced by Curles and Levoy [44] made truncated signed distance functions popular in computer graphics, computer vision, and robotics applications. More than a decade later, Newcombe massively popularized 3D surface reconstruction as a mapping technique [126] and established a new standard for 3D robotics mapping, mainly for indoor scenes and employing RGB-D sensors [17, 21, 22, 57, 87, 94, 113, 121, 130, 132, 134, 146, 157, 161, 177, 179, 187, 203, 204]. Although numerous systems have been proposed to extend KinectFusion, few address the problem of mapping environments larger than an office-sized room. Notable exceptions are the approach introduced by Whelan et al. [202, 203] and recent work exploiting octrees [61, 187].

LiDARs are more precise and less noisy than RGB-D sensors. Nevertheless, two challenges remain open to building efficient 3D maps with such sensors. First, the existing 3D mapping systems are typically hand-crafted for a particular sensor, such as Microsoft's Kinect, often rendering the implementation unusable for other kinds of sensor modalities. Second, extending RGB-D mapping systems to the LiDAR domain might seem straightforward, however, in practice, a significant refactoring process is required to adapt the system to this type of sensor. For that, consider that it is not trivial to project a 360° LiDAR scan using a pinhole camera model. In addition, despite the publicly available mainstream open-source libraries for computer vision and 3D data processing, such as OpenCV [15] or Open3D [221], the provided implementations for volumetric integration do not work out of the box with LiDAR data. At the same time, a simple and naïve implementation using a dense voxel grid as for RGB-D [126] is simply not possible due to memory constraints. Second, most of these techniques rely on GPUs, which improves the effectiveness of the mapping method but also leads to increased computational and energy demands. Although this is generally not an issue for desktop computers, it poses a limitation for power-constrained mobile robots. These robots typically need to run the mapping system simultaneously alongside tasks such as localization, obstacle detection, and path planning. Consequently, even if the mapping system can utilize the available computing resources on a mobile platform, it remains essential to address the requirements

of other tasks. Thus, the need for desktop-style GPUs still represents a strong constraint in robotics today.

To expand KinectFustion capabilities to larger scenes, Whelan et al. [202, 203] introduced one of the first methods that dealt with larger environments by employing a rolling grid that streams out a triangle mesh when exiting the volume being mapped. In this line, sparse data structures have also been explored to build such systems, for example, octrees [61, 174, 187, 197] or voxel hashing approaches [121, 128, 130, 132]. More recently, due to developments in LiDAR technology, numerous other mapping systems have been proposed in the SLAM community [6, 43, 70, 81, 160, 219]. To cope with the lack of GPUs on mobile platforms, OpenChisel [90] provides an effective solution to the problem of volumetric reconstruction on the CPU, but is limited to depth sensors. OpenChisel was later extended by Oleynikova et al. [130] to Voxblox. To the best of our knowledge, Voxblox was considered the state-of-the-art open-source system for building volumetric maps on a CPU and is used effectively in numerous methods based on it [67, 121, 146]. Our VDBFusion method [190] is challenging this status of Voxblox. More recently, Wang et al. [197] proposed a system that extends SuperEight [61, 187] using LiDARs that exploit the octree data structure instead of relying on voxel grids, but to the best of our knowledge, the implementation is not publicly available.

Despite achieving promising results in 3D robot mapping, unresolved challenges still need to be addressed. The extension of SuperEight [61] makes many assumptions, such as knowing the map size in advance and truncating the LiDAR measurements' range to a specific range (60 m) to cope with memory and runtime requirements. In addition to these assumptions, the system runs at 3 frames per second (fps) and is, therefore, slower than the LiDAR sensor frame rate. Furthermore, Voxblox [130] is the approach most similar to the online mapping system presented in this thesis (Sec. 7.3.7) since it is the first of its kind to process point clouds instead of raw depth measurements. However, Voxblox can be difficult to use and virtually impossible to employ outside the ROS 1 ecosystem. This thesis looks at generic mapping systems that can be deployed in different robotics architectures without necessarily relying on a particular framework.

In contrast to 3D surface reconstruction, there are mapping systems built on top of other techniques [182], such as Octomap [31, 75, 211]. Octomap significantly improves memory consumption compared to TSDF-based mapping approaches. However, the runtime of its mapping pipeline prohibits the deployment of this method for 3D LiDARs in the real world, as it is only capable of integrating such data at 1 fps.

In summary, building a 3D mapping system that is memory efficient and fast at the same time remains a challenge. In this thesis, we aim to fill this gap with the

aid of the VDB data structure [125], designed to model and render photorealistic scenes in movie animation. We develop a robust open-source 3D volumetric integration library that can work with any 3D sensor modality. The complete details of our online mapping approach are discussed in detail in Chapter 7. Our system is easy to use and extensible for various applications. We built our online mapping approach using the VDB data structure [125].

Related to this thesis, Macenski et al. [108] developed a spatio-temporal voxel system for 3D mapping using VDBs [125]. This system encodes sensor observations into an occupancy grid map that implements voxel decay and decay acceleration [108]. More recently, Besselmann et al. [12] have also shown promising results using the VDB data structure to implement an octree-inspired representation similar to Octomap [75]. Although both works are closely related to our system, a key difference is that we represent the environment as a smooth truncated signed distance field (TSDF) surface, and the other works represent it as an occupancy grid.

The system presented in Sec. 7.3.7 can fuse data from LiDARs, RGB-D cameras, or any other 3D sensor that produces point clouds. Experiments show that our system runs at 20 fps on average for a 64-beam LiDAR and at 10 fps for RGB-D sensors without using GPUs. Our system runs entirely on a single core of a CPU, making our system applicable to being deployed in mobile robots, where power consumption and CPU resources are limited. At the same time, it is highly memory efficient.

Despite the advances this thesis brings in accelerating the construction of TSDF-based maps for robotics applications, there may be underperformance. Specifically, in scenarios where time is not a critical factor, the effectiveness of these techniques might be reduced. It is important to note that these volumetric maps are usually utilized for downstream tasks like path planning, obstacle avoidance, and overall robot exploration and navigation. Therefore, an accurate description of the surface or visual quality of the map may not always be a prerequisite to effectively accomplishing such tasks.

Broadening the scope of this thesis's contributions, we also introduce in Chapter 5 a technique for 3D mapping that capitalizes on the ability to operate at speeds slower than the sensor frame rate. For generating high-fidelity maps, we employ Poisson surface reconstruction [85] in Sec. 5.1. In this line, traditional approaches determine an implicit function modeling the underlying surface, for example, using tangent planes [74], radial basis functions [24], TSDF [44], or polynomial representations [92]. Poisson surface reconstruction [84, 85, 86] provides a geometrically accurate reconstruction based on this principle. The method we propose in Sec. 5.1 of this thesis produces 3D maps in the form of triangle meshes.

The use of 3D triangle meshes in robotics has gained significant attention

due to its potential to provide detailed and accurate environmental representations. Several previous works have explored this representation of the map. For example, Marton used triangle meshes for fast surface reconstruction methods for noisy and large point clouds et al. [112]. In recent years, such representations have also been explored for visual-inertial systems [152], LiDAR-based approaches [39, 153], and purely vision-based systems [139]. More recently, the work by Niedźwiedzki et al. [127] presents an incremental algorithm to generate triangle meshes from LiDAR data that produce a triangle mesh directly from the LiDAR scans without storing a dense point cloud to create a high-quality triangle mesh. In contrast to our work, these approaches typically compute a sparse reconstruction of the traversed environment, while we aim to reconstruct a continuous triangle surface capturing geometric details.

By relying on higher computational times, it is possible to use triangle meshes as the primary representation and achieve a more detailed description of the environment. While such precision may appear excessive for tasks such as obstacle avoidance, where robots plan to circumvent obstacles with a substantial safety margin, other tasks can significantly benefit from meticulous surface reconstruction of the environment. For example, localization tasks, as we will show in Chapter 6, can substantially benefit from highly detailed environmental reconstructions.

Recent work has shown the importance of obtaining a dense observation of the environment without accumulating multiple frames [195]. The ability to predict how the environment looks beyond current observations can be exploited for robotic tasks such as navigation [195]. Thus, being able to complete the geometry of single observations is also a relevant (and interesting) research question.

With the recent developments in deep learning [65], many approaches have been developed to solve 3D reconstruction [118, 135, 137, 147] and scene completion [45, 47]. However, most approaches only consider relatively small objects (from ShapeNet [27]) and do not apply to our target setting, namely large outdoor scenes.

More recently, learning-based approaches for RBG-D sensors have been proposed that improve volumetric aggregation in a TSDF [199] or learn to complete or improve the appearance of the generated reconstruction [47, 119, 123]. The work we present in Chapter 8 is related to the work of Dai et al. [47] and Atlas [123]. In contrast to these methods, which work on aggregated volumes, we target the single scan setting to avoid the time-consuming buffering of scans.

With the availability of large annotated LiDAR datasets [5], new approaches have been proposed for *semantic* scene completion. Typically, these approaches require a relatively large amount of training data to produce good results [148, 150]. Additionally, adapting such systems to target geometry-only predictions

is not straightforward due to the complexity of the network architectures. Instead, in our work, our objective is to produce a smooth surface representation of the map. Furthermore, our geometric scan completion does not require labels (Sec. 8.2.5), allowing our system to be trained from pure real-world data.

Furthermore, semantic scene completion systems typically require that the input data be fit to a fixed volume to cope with memory limitations, typically 256×256×32 voxels [148, 150]. One common drawback of this design choice is that all information on the negative x axis is discarded, which is essential for mapping applications but of little relevance for semantic scene completion. Additionally, such methods use only a voxel size of 0.2 m, while we can produce a higher resolution model by picking 0.1 m, instead. In contrast to those works, we do not assume the size or volume to be processed, allowing us to complete a full outdoor-like TSDF entirely volume. Seeing our approach as a subset of the semantic scene completion task might be tempting at first sight. However, this is not the case, as we explain in Sec. 8.2. Our contribution is to produce denser, semantic-free geometric models using low-resolution scanners without making any assumption on the input data size.

2.2.1 Exploiting Maps for Robot Localization

For localization, given a map, one often distinguishes between pose tracking and global localization. In pose tracking, the vehicle starts from a known pose, and the pose is updated over time. In global localization, no pose prior is available. In this thesis, we address global localization using 3D LiDAR data without assuming any pose prior from global navigation satellite system (GNSS), global positioning system (GPS), or other sensors. Therefore, we concentrate here mainly on LiDAR-based approaches.

In the context of autonomous cars, many approaches were proposed for accurate pose tracking using multiple sensor modalities and high-definition (HD) maps. Levinson et al. [97] utilize GPS, IMU, and LiDAR scans to build HD maps for localization. They generate a 2D surface image of ground reflectivity in the infrared spectrum and define an observation model that uses these intensities. The uncertainty in intensity values has been handled by building a prior map [209]. Barsan et al. [79] use a fully CNN together with HD maps to perform online-to-map matching to improve the robustness of dynamic objects and eliminate the need for LiDAR intensity calibration. Merfels and Stachniss [115] present an efficient chain-like pose graph for vehicle localization that takes advantage of graph optimization techniques and different sensing modalities. Based on this work, Wilbers et al. [208] propose a LiDAR-based localization system performing a combination of local data association between laser scans and HD map features, temporal data association smoothing, and a map matching approach for

robustification. The approaches above show good performance for tracking vehicles' poses but require GNSS information to operate on HD maps. In contrast, our approach addresses global localization using only 3D LiDAR data without assuming any pose prior.

To achieve global localization, traditional approaches rely on probabilistic state estimation techniques [182]. A popular framework is Monte Carlo localization [50, 60, 183], which uses a particle filter to estimate the robot's pose and is widely used in robot localization systems [8, 35, 96, 178, 213].

Several approaches have been proposed exploiting deep neural networks and semantic information for 3D LiDAR localization. For example, Ma et al. [107] combine semantic information such as lanes and traffic signs in a Bayesian filtering framework to achieve accurate and robust localization within sparse HD maps. Yan et al. [213] takes advantage of information about buildings and intersections from a LiDAR-based semantic segmentation system [120] to localize in OpenStreetMap data. Schaefer et al. [159] detects and extracts pole landmarks from 3D LiDAR scans for long-term urban vehicle localization, whereas Tinchev et al. [184] propose a learning-based method to match tree segments and localize in both urban and natural environments. Sun et al. [178] use a deep-probabilistic model to accelerate the initialization of the Monte Carlo localization and achieve a fast localization in outdoor environments. In our previous work [34, 35], we also use semantic CNNs to predict the overlap between LiDAR scans and their yaw angle offset and use this information to build a learning-based observation model for Monte Carlo localization. The learning-based methods perform well in trained environments, while they usually cannot generalize well in different environments or LiDAR sensors.

In sum, our method only uses LiDAR data to achieve global localization outdoors without using any GNSS. Furthermore, our approach uses range information directly without exploiting neural networks, semantics, or extracting landmarks. Therefore, it generalizes well to different environments and LiDAR sensors and does not require new training data when moving to different environments.

Chapter 3

A Modern Infrastructure for Reliable Data Collection

Building reliable perception platforms for autonomous vehicles have become an essential element of robotics research. Perception platforms serve as the robot's sensory system, collecting information about its environment, much like how humans use their senses to understand the world around them. Working with and thus recording real-world sensor data is vital for developing robust estimation algorithms. Naturally, reliable perception platforms are a prerequisite for creating 3D robot maps. Failures in the sensory system, such as missing data in LiDAR frames or synchronization issues, could introduce significant and often irreversible errors in the final map representation. Attempting to construct a map without a fully operational and reliable perception platform can compromise the map's usability and reliability.

This chapter investigates building reliable perception platforms that capture sensor data, ensuring the integrity and quality of the datasets generated across multiple runs. We propose using version control systems to enhance dataset acquisition's efficiency, reliability, and scalability. We present a method that can launch the system and record data while logging the exact state of the system, making the setup and the data generated with it reliable and reproducible. Our method is based solely on standard tools, independent of the chosen sensor suite or host system, and therefore applies to existing perception platforms. We also provide a step-by-step tutorial for existing systems to replicate the ideas introduced in this chapter. Additionally, we demonstrate its practical implications through various use cases, showcasing the value of incorporating version control systems into perception platform operation. Furthermore, we illustrate how the concepts introduced here are applied to a real-world scenario, the IPB-Car perception platform.

3.1 Reproducible Version-Controlled Perception Platforms

Version control tools such as git [100] or SVN [140] track file changes over time, allowing multiple developers to collaborate efficiently and manage code revisions. Git has revolutionized software development by simplifying collaboration among team members, enabling easy rollbacks, and facilitating the management of complex projects. To uniquely identify a specific commit (a snapshot of the codebase), git uses a commit hash. The commit hash is a unique alphanumeric identifier that version control systems generate to track changes and manage the repository's history. Although version control has drastically changed the way software is developed in many areas of computer science, little attention has been paid to applying these concepts to perception systems, where other devices, such as sensors, are also part of the software stack. In this chapter, we investigate how to transfer version control practices that are standard in software engineering to use when developing robotics perception platforms.

Validation and testing algorithms performance is crucial in robotics research and ensures safety when deploying robotic systems in the real world. This requires the availability of large, diverse, and reliable datasets. Building datasets in robotics [5, 16, 19, 28, 63, 144, 196, 215] has taken research forward, allowing easy comparison between approaches for different robotic tasks such as odometry estimation, object detection, semantic segmentation, localization, etc. Nevertheless, properly recording data, mainly if multiple sensors are used, is a significant effort in robotics. Although various datasets have been developed, there is relatively limited literature on *how* to build a good perception platform.

The main contribution of this section is a systematic and practical method that will improve existing perception platforms used for data collection in most research labs. Our method implements version control across the entire system and uses docker containers. By referencing the commit hash in the logs, the configurations, software, and recording sessions of the datasets can be retrieved precisely. Our method not only simplifies the setup process but also increases the accessibility of the platform for users who may not have specialized equipment or knowledge. This facilitates collaboration among researchers and ensures the use of up-to-date software for the perception platform.

One of the first data collection platforms for autonomous vehicles [63, 64] comprised two color and grayscale cameras, a Velodyne LiDAR, and a GNSS with IMU and real-time kinematic (RTK) correction. This platform was used to record the popular KITTI dataset. To synchronize the sensors, the LiDAR is used as a reference to trigger the cameras after each rotation. For each LiDAR-triggered time, the closest GNSS/IMU measurement is taken since the localization system

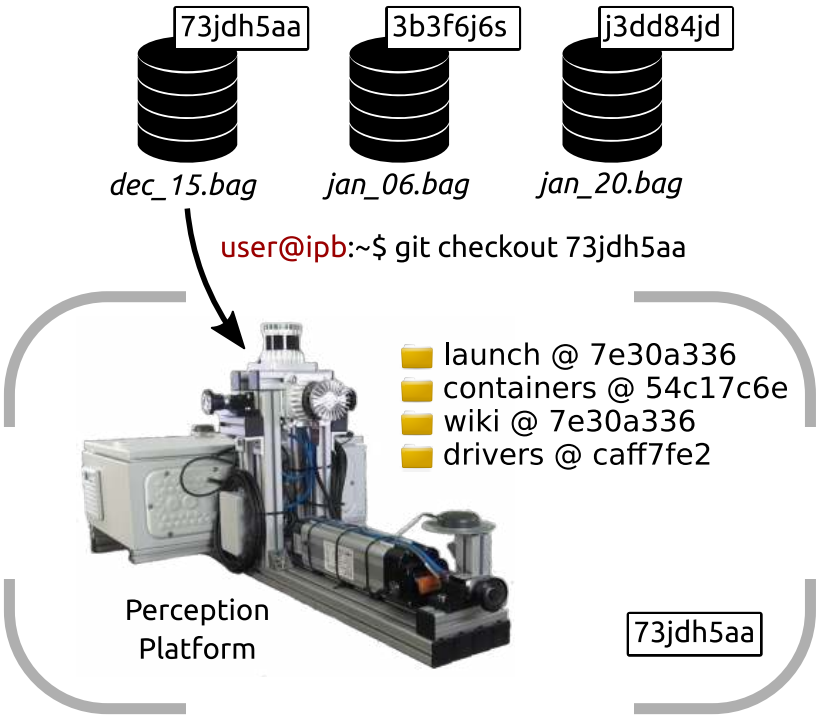


Figure 3.1: After recording data at different points in time using different driver versions or configurations, our proposed version-controlled perception platform allows to checkout at a recording's unique hash describing the state of the full system at that moment.

runs at a higher frequency than the other sensors.

Following KITTI, other datasets were proposed [16, 88, 109, 144, 196, 215]. However, since those works focused on collecting datasets, their data collection platform was often not discussed in detail, which kept the challenge of developing a system able to collect synchronized data continuously. To the best of our knowledge, only the ApolloScape dataset [77] released a software architecture for data collection. However, unlike our work, many specific hardware requirements must be met.

Moreover, in robotics research, an equally challenging aspect is the ability to reproduce software code, which is crucial for validating research findings and promoting collaboration within the community [25, 26, 58, 93, 185, 201]. Software in research labs is rarely in the final state, and researchers continuously test, use, discard, and deploy new developments, configurations, and setups. Additionally, the tools used to record datasets often lack reproducibility due to scattered software, hardware descriptions, and documentation. This makes it difficult to determine the system's state for a specific recording. In addition, system documentation is often maintained separately and not as part of the main software, leading to potential errors in the recording phase due to outdated instructions.

One solution to these challenges is what we call *version-controlled perception platforms*. These platforms allow the creation of datasets by exploiting

version control beyond the software, enabling tracking changes in the configuration, setup, sensor drivers, documentation, hardware description, and operating system configuration. Our method simplifies the setup process and enhances the accessibility of the such platforms. As illustrated in Fig. 3.1, we integrate the entire description of the system into a version-controlled environment, in which each recording generated with the platform is associated with a commit hash that captures the complete system's state at the time of recording. In addition, we provide a straightforward networking approach by specifying the network setup in a configuration file. This eliminates the need to manually configure complex network settings, making the deployment and communication of the perception platform more efficient and user-friendly and transforming the platform into a *plug-and-play* solution, as no manual intervention on the host machine is required.

To implement our version-controlled perception platform, we utilize standard tools available in most modern operating systems, namely git [100] and docker [14]. We propose a method to easily install and launch the system on a new machine, reliably record data, and simultaneously log the exact state of the system. Furthermore, we introduce a sensor synchronization setup that minimizes hardware intervention using a single network cable, such as Ethernet or Thunderbolt. This unique network interface is sufficient to capture the complete sensor stream. This approach facilitates the creation of experimental branches for development and testing. Furthermore, our methodology uses a time-synchronization scheme that enables the synchronization of different sensors solely through network cables, eliminating the need for additional hardware triggering systems, enhancing the system's simplicity, and avoiding the complexities and costs associated with external triggering mechanisms.

3.2 The Meta-Workspace Concept

An essential component of our system is what we call the meta-workspace. It is a git repository containing only two files sufficient to describe, build, and run the entire system: *.gitmodules* and *launch.yaml*. This implies that cloning it into an empty directory on the host system is sufficient to download all components, initiate the build process, launch the system, and record data.

The *launch.yaml* is a docker-compose description file, which specifies the network configuration of the host machine, the docker services to run, the IP addresses of all sensors, and the network interface used to stream data. The *.gitmodules* file lists all the components our system needs. This requires utilizing git-submodules and docker, nowadays standard tools included in every GNU/Linux distribution. Encoding the network configuration in the launch.yaml file and the container specifications are crucial components that enable development on a host

machine completely decoupled from the operating system used to run the sensor suite. Furthermore, by listing all the required components in the .gitmodules, we can have different configurations for continuous integration (CI), issue trackers, and potentially separate git servers for each module.

Using a monorepo[1] to store all the software components is also common. However, it does have a significant drawback: the loss of individual version control for modules. This makes updating specific modules with newly released code from sensor vendors difficult. Additionally, a monorepo environment lacks the transparency to change individual module configurations. These limitations can hinder the flexibility and efficiency of managing and updating modules within the perception platform, especially when larger systems are built and modern sensors are used. Therefore, we opt against a monorepo for our setup.

A significant benefit of our system is that, unlike other robotics platforms with more specific requirements, our platform only needs a computer running any GNU/Linux distribution and a network interface to connect to the setup. The meta-workspace handles all other dependencies and configurations. This approach not only simplifies the setup process but also increases the accessibility of the platform for users who may not have specialized equipment or knowledge.

Employing git for all components in our perception platform brings the inherent benefits of version control, such as checking out, tracing back changes (blaming), branching, and others. This ensures the reliability of the datasets created with the system since the perception platform state at the time of dataset recording is preserved through the associated git hash. As a result, users can have confidence in the datasets, knowing that all components' states can be accurately recovered. Furthermore, reproducing the platform state is simple by checking the hash associated with a specific recording, as illustrated in Fig. 3.1.

3.3 Building Blocks of a Version-Controlled Perception Platform

In the following sections, we outline the essential components required to realize a version-controlled perception platform, regardless of the chosen sensor suite or the host machine. Our approach is not tied to a specific platform, but we will give a practical example of how to implement the concepts discussed here in a real-world case study. Unless explicitly stated otherwise, each subsection refers to one or more git repositories that are part of our meta-workspace. The meta-workspace is illustrated in Fig. 3.2.

[1] In version-control systems, a monorepo is a software-development strategy in which the code for several projects is stored in the same repository.

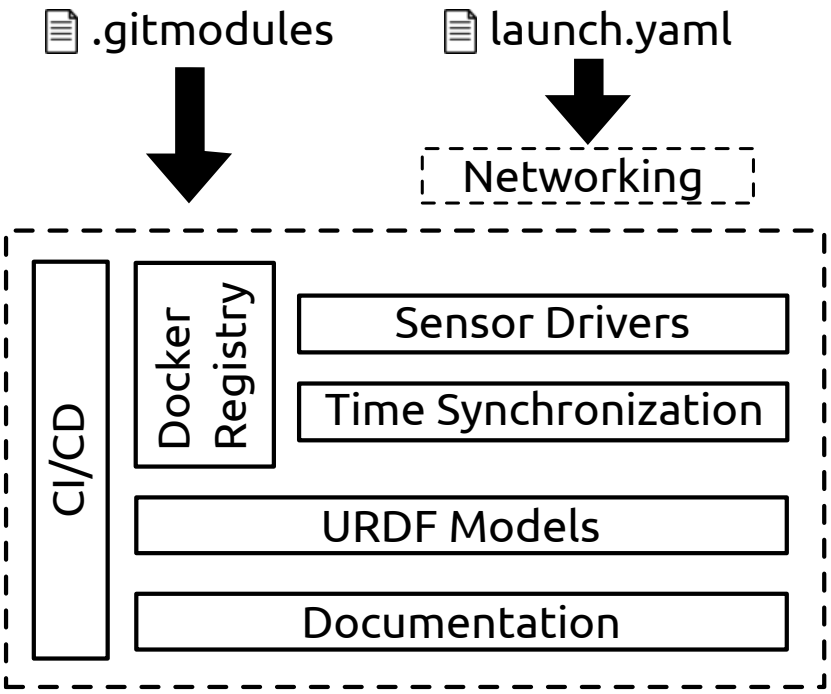


Figure 3.2: Main components of a version-controlled perception platform. Our meta-workspace is defined by two files, .gitmodules, and launch.yaml. As shown in this illustration, the launch.yaml file specifies all the necessary network configurations to operate the system. The .gitmodules file, instead, specifies all the necessary components to build and run the system.

3.3.1 Docker Registry

We containerize all software components required to operate the sensor suite to decouple the host system from the perception platform. Furthermore, we maintain a private docker registry on our git server that provides pre-built docker images for all the containers required to run our system. The docker files used to build these containers are also included in our meta-workspace, allowing us to have a local copy of the files used to generate these containers in our registry. Additionally, this approach enables us to modify the containers locally, as necessary, for example, in the early stages of development or for debugging purposes.

3.3.2 Sensor Drivers

Our perception platform uses the robot operating system (ROS) to launch the sensors and log data. However, we do not enforce a specific ROS version, and other robotic frameworks can also be used. We containerize all required sensor drivers and custom ROS nodes. Adding a new sensor is as simple as adding the sensor-specific driver to our git submodules list. If a specific software development kit (SDK) or system package is required to operate the sensor, then the Dockerfile can be updated without affecting the development on the host machine.

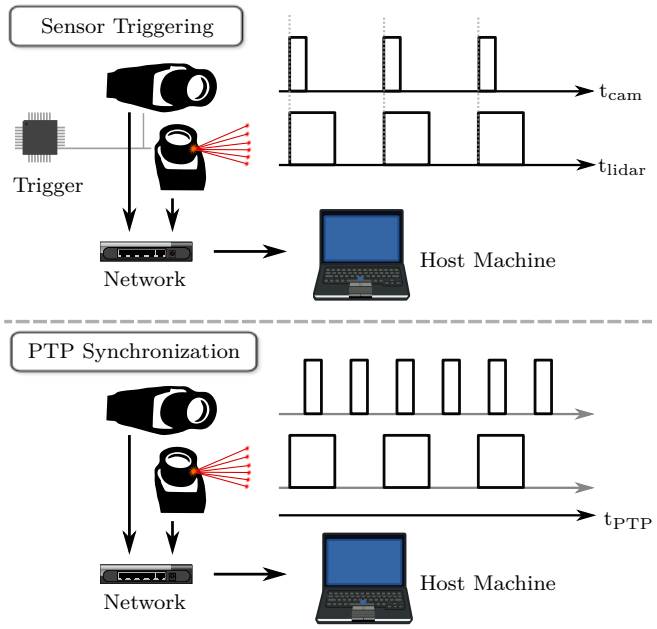


Figure 3.3: Difference between synchronized sensor triggering *(top)* and time synchronization *(bottom)*. Traditional sensor hardware triggering relies on external hardware sources to capture data from sensors within the network. Our approach to time synchronization using PTP eliminates the need for additional hardware components. Instead, it synchronizes the internal clocks of the sensors themselves, ensuring precise and coordinated data acquisition without reliance on external triggers.

An important benefit of this design choice is that the sensor suite can be operated independently with different ROS versions, such as ROS 1 or ROS 2, simply by changing the meta-workspace definition. A possible realization could be maintaining separate git branches for each ROS version, with modified git submodules and Dockerfiles. This enables switching between ROS versions without changing the host machine or its operating system, hardware, or sensor configuration.

3.3.3 Time Synchronization

Ensuring time-synchronized sensor data is a crucial requirement for modern perception platforms. It is important to distinguish between two concepts: *sensor time synchronization* and *synchronized sensor triggering*, which are often misunderstood and (wrongly) used interchangeably. We discuss both setups in Fig. 3.3.

Sensor time synchronization refers to the use of a shared master clock by all sensors in a running system that allows them to report timestamps relative to the same clock, and often, it is possible to also trigger sensors via software, for example, to take images at a specific point in time. However, it does *not* necessarily mean that the sensors will capture data frames in perfect sync as a hardware trigger would do.

On the other hand, synchronized sensor triggering refers to the use of a particular signal, often generated by a hardware trigger or another sensor general-purpose input/output (GPIO), to capture the sensor data frames in a coordinated fashion. This approach is often used for stereo cameras or overlapping multi-camera systems. For example, in the KITTI odometry dataset [63], the Velodyne LiDAR triggers the cameras at the end of each laser sweep. Although this approach can ensure that data frames from different sensors can be associated, it has some limitations. For example, it enforces additional hardware modifications, such as wiring the GPIO pins from one sensor to another, and some sensors are hard to trigger, such as rotating 3D LiDARs. This makes the platform setup more challenging to reproduce in scenarios where hardware engineers are unavailable. It also limits the flexibility of triggering by the LIDAR hardware. Furthermore, some sensors, such as a 3D LiDAR, cannot be actively triggered. They can only generate a trigger output in specific configurations. Additionally, this approach does not scale well as the number of sensors in the suite grows.

In contrast, we advocate building a sensor suite entirely configurable by software and not requiring additional hardware intervention. To achieve this, we use the IEEE 1588 precise time protocol (PTP) [55], a widely adopted standard for clock synchronization in networked measurement and control systems today. PTP provides accurate clock synchronization across multiple sensors and other devices, which is critical to obtaining reliable measurements. Note that the PTP protocol can be established through standard plug-and-play network connectors, but it enforces the requirement that the sensor must come with PTP support from the vendor. Although our time synchronization setup drastically simplifies the hardware connections, stereo cameras or multi-sensor arrays with overlapping fields of views used for point triangulations may still require a hardware trigger if exactly synchronized exposure is essential.

PTP Grandmaster Clock. Our approach to network-synchronized sensors requires a precise Grandmaster node in the PTP network [55]. In some scenarios, any sensor or device on the network can act as this master clock. Nevertheless, we design and control the Grandmaster clock as part of the sensor suite to achieve reliability and absolute timestamps for each sensor. We propose using an onboard computing system like a Raspberry Pi or Intel NUC with additional GNSS hardware to synchronize time via the GNSS satellites.

Although not strictly required for realizing a version-controlled system, we utilize a GPS-disciplined host to develop our synchronization setup. The GPS satellite time initially adjusts the Grandmaster's internal clock and is subsequently synchronized with a pulse-per-second (PPS) signal from the GPS sensor once per second. Once the Grandmaster clock is synchronized with the GPS satellites, all PTP-enabled sensors in the network will synchronize to this Grandmaster clock,

eliminating the need for user intervention or hardware configuration. Moreover, each sensor's timestamps will contain absolute timestamp information, as the PTP Grandmaster clock receives its time from the GPS satellites. Therefore, all datasets recorded with this synchronization setup will store the exact time a particular data frame was recorded.

This choice might introduce an additional vulnerability that could disrupt the functioning of the perception platform. Once the Grandmaster clock machine is set, it becomes crucial to ensure that any interventions or modifications to the system do not compromise its internal configuration. To address this weak point, we have developed a custom operating system (OS) distribution for the Grandmaster clock computer, incorporating an additional monitoring layer. Before launching our system, we conduct a sanity check to verify that the configuration of the Grandmaster clock computer aligns precisely with the specifications outlined in the current working environment. This enables us to identify any potential mismatches or discrepancies that may arise promptly.

3.3.4 Networking

Our methodology introduces a minimal requirement for the sensors, specifically an Ethernet connection with support for PTP, which is now commonly available. In contrast to the traditional approach in the robotics community, our system simplifies network configurations by specifying all essential details, such as IP addresses, within the launch.yaml file. Consequently, no modifications to the host machine are needed, resulting in a streamlined and portable setup. Another advantage is the ease of transferring and running our perception platform on another host machine without any changes or adjustments. Importantly, our setup does not override any local configuration on the host machine, making it a non-invasive approach.

Depending on the sensor configuration, it might also be necessary to ensure the network is 10 Gigabit Ethernet (GbE) instead of more traditional 1 GbE setups for fast and collision-free routing, an option we chose to handle multiple cameras and 3D LiDARs in our setup.

3.3.5 Documentation

Traditionally, the documentation for perception platforms is stored in a separate repository or documentation system, independent of the system's main codebase. While this separation may have benefits, we have experienced that including the documentation repository in the main codebase of our system has several advantages. Firstly, it ensures that the documentation is always tightly coupled with the codebase. Second, it allows developers to search for source code and

documentation consistently. Third, it allows one to check the documentation even when no internet connection is available. Lastly, it provides a comprehensive system overview by bringing together all relevant information in one place. We strongly recommend this approach, especially for university research labs with changing personnel.

3.3.6 Unified Robotics Description Format (URDF) Models

In our meta-workspace, we also incorporate the calibration of the sensors utilized in our system. Our system's URDF models are included in our meta-workspace, which completes the description of the current working setup. By storing the calibration in the URDF git submodule, any changes to the calibration parameters can be tracked and visually inspected when launching the system.

3.3.7 Continuous Integration/Continuous Deployment CI/CD

CI/CD is a software development practice that relies on a build server or similar infrastructure to automatically integrate code changes, build software artifacts, run tests, and deploy applications in a streamlined and continuous manner. The stability of our proposed system is highly dependent on our CI/CD infrastructure. We use our CI/CD to pull new changes from the stable branches into the meta-workspace definition to keep it up to date. In addition, the CI/CD is in charge of building all the ROS nodes, docs, URDF models, and more on each commit of each git submodule. We use the same containers to run the perception platform and the CI/CD infrastructure to ensure we can build the components locally. Furthermore, the CI/CD performs nightly builds and deploys the docker images, ensuring the docker containers are up-to-date. Lastly, our CI/CD infrastructure builds our custom operating system for the Grandmaster clock computer, meaning that at any point in time, we can use the build artifacts and start from a new and known state.

3.4 Building a Version-Controlled Platform – A Concrete Use Case Example

This section details a three-step process for implementing the principles outlined in this chapter to an existing perception platform. Although the concepts described in our work apply to virtually any hardware configuration, we illustrate

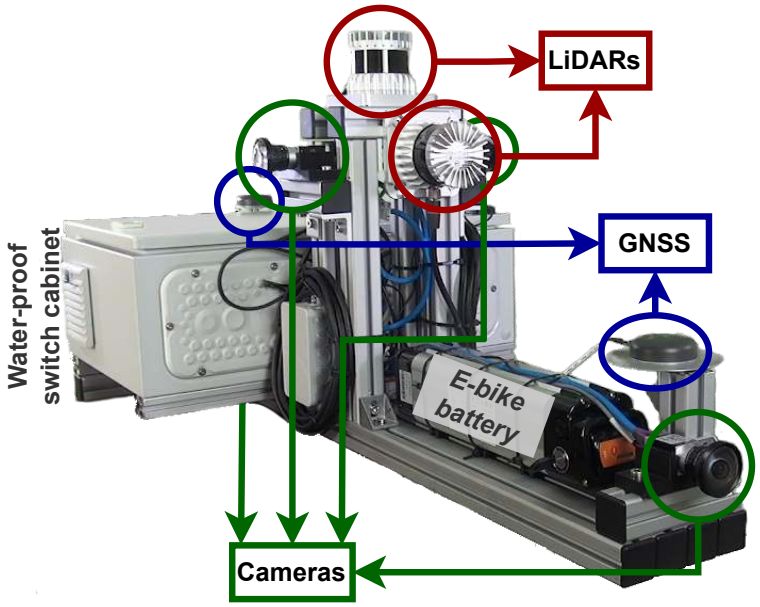


Figure 3.4: The IPB-Car platform: a practical example of a version-controlled approach.

how to follow these steps using our platform called IPB-Car, the perception platform we use to record datasets for research on autonomous driving.

3.4.1 The IPB-Car Platform

The sensor setup of the IPB-Car platform is shown in Fig. 3.4. It consists of the following sensors:

- 1 x 3D LiDAR Ouster OS1-128
- 1 x 3D LiDAR Ouster OS1-32
- 4 x industrial-grade cameras, Basler ACA2040-35GC
- 1 x navigation system (GNSS/IMU), SBG Ellipse-D
- 1 x PTP Grandmaster computer, based on a Raspberry Pi 4 with an additional GPS receiver
- 1 x QNAP WSW-M2108-2C 10 GbE network switch
- 1 x QNAP QNA-T310G1T 10 GbE, thunderbolt converter

Our platform is designed with a single network cable to allow the seamless flow of the entire sensor stream from the platform to the host machine. For this, we use a 10-Gigabit Ethernet setup. This requires that all sensors have available Ethernet ports, but not necessarily 10 GbE capable ones. In our setup,

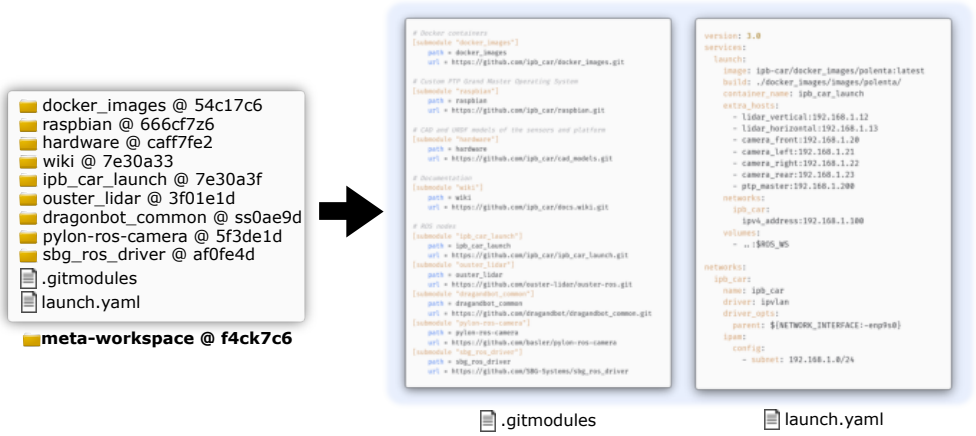


Figure 3.5: An example of our IPB-Car meta-workspace. All the necessary components are described within the .gitmodules file. Additionally, sensor IP addresses and additional network configurations are defined in the launch.yaml file on the right of the picture.

for example, all cameras are 1 GbE cameras connected to the network switch, and 1 GbE streams are automatically integrated into the 10 GbE network. We connect all Ethernet sensors to a network switch, which is then connected to an Ethernet-to-thunderbolt converter. As a result, a single thunderbolt cable is the only cable required to connect to the host machine to record data from the platform.

In what follows, we describe the steps to implement the method described in Sec. 3.3.

3.4.2 Step 1: Identify System git Components

The most crucial aspect of version control of a new platform is creating git repositories for each system component. This is a common practice in robotics nowadays. However, we push this technique further by creating repositories for non-software components, such as the URDF models describing the system, the documentation, etc. The reasoning behind this design choice is to enable us to take a snapshot of the entire system with a simple git hash. Because of this design choice, the first step is identifying all the components required to run the perception platform and ensuring the selected components are accessible through the git repositories.

3.4.3 Step 2: Containerize All Components

Once all components have been identified, a container must be provided for each component. There are two approaches to solving this step. The first is to create one container for each service, for example, one for the cameras, one for the LiDARs, and so on. The second approach is to create one unique container

for all the services in the system. We chose the second option since we found that, in practice, it is easier to maintain and adapt to different types of sensor configurations.

3.4.4 Step 3: Create a Meta-Workspace Repository

Assuming all system components can be used in docker containers and the information where each git repository is already gathered, the next step is to create the meta-workspace repository, introduced in Sec. 3.2. This is a regular git repository that contains the two files sufficient to describe, build, and run the perception system: the files .gitmodules and the launch.yaml. The file .gitmodules contains all the necessary repositories that describe the perception platform. The containers identified in the previous stage should be added to the file launch.yaml. Additionally, all network configurations, such as the host machine's subnet, mask, and IP address, should be included in the launch.yaml. An example is shown in Fig. 3.5.

3.5 Practical Applications of Our Method

This section exemplifies how using our method simplifies everyday tasks encountered in the data collection process in robotic research labs.

3.5.1 How to Reliably Record Data

We can ensure consistency across multiple datasets by generating a git tag for the meta-workspace when the system is known to be in a functional state. This tag represents the exact setup, configuration, and code for recording the datasets. Consequently, it is easy to reproduce the same conditions and maintain a reliable and consistent environment for dataset generation, even over long recording periods.

In practice, even small changes in the codebase can result in non-working system configurations. With our framework, as long as we can identify a functioning commit within the git infrastructure, we can utilize standard tools like git-bisect to reproduce the problem, pinpoint the introduction of the bug, and finally eliminate it. It enables us to restore the last known working system state and effectively debug the issue. Furthermore, by using development branches, other team members can continue their work without disrupting the main recording setup. This allows for parallel development efforts while ensuring the stability and integrity of the system.

3.5.2 How to Retrieve System State From Data Recordings

Backtracing the state of the perception platform for a previously recorded dataset is usually challenging. Our approach allows us to checkout the entire system's state with the git hash associated to the recording. This enables us to know precisely how data was recorded and gives valuable insights, for example, if any pre-processing or filtering was applied or what a specific sensor configuration was. This increases trust in the collected data and allows fair comparisons between different recordings.

3.5.3 How to Work on Different Hardware Configurations

When deployed in practice, our methodology facilitates using the perception platform in various types of robots, considering their specific constraints. For example, customization of the meta-workspace becomes necessary for a wheeled mobile robot that cannot accommodate a laptop for data recording. Consequently, the sensor drivers and the corresponding SDK associated with the cameras could be removed. In that case, the build time and system footprint can be significantly reduced, allowing for the use of the system on a more resource-constrained platform. This adaptability allows the perception platform to be tailored to different robot configurations and requirements, ensuring its applicability across a wide range of robotic applications. It is also important to note that calibration procedures may need to be adjusted accordingly, which are also tracked when using our methodology.

3.5.4 How to Run the System on Different Machines

In practice, it is highly advantageous to have the ability to operate the system using different computers within a team. Sharing a single laptop can become a bottleneck, affecting productivity and flexibility. Our methodology enables developers to have individual copies of the meta-workspace, allowing them to build, launch, and test the system independently without relying on a shared machine or coordination. Furthermore, the only requirement for the host operating system is to run any GNU/Linux distribution. This allows for the operation of the perception platform using an OS that may not support parts of the stack, such as ROS 1. As a result, team members can work independently on different requirements while maintaining a consistent and efficient development environment. This eliminates the challenges of sharing a single machine and allows for

seamless collaboration and progress on different requirements within the perception platform.

3.5.5 How to Migrate between ROS 1 and ROS 2

Migrating to ROS 2 can be challenging, as existing systems may not work seamlessly, and installing ROS 1 and ROS 2 on the same operating system can be problematic. One possible solution is to dedicate a separate computer solely for ROS 2 migration. Our framework offers a more elegant approach. By creating a new branch of the meta-workspace, developers can change the base image of the Docker containers and work towards achieving a functional ROS 2 setup. Once the migration goal is achieved, the team can decide whether to merge the ROS 2 branch into the main development branch or keep it separate. Importantly, even if the ROS 2 branch is merged, the older ROS 1-compatible branch remains accessible at any time through git tags or releases, so previously used recording setups can be reproduced on demand easily. This allows developers to switch between ROS versions without changing the host operating system, computer, or other configurations. Our framework simplifies the migration process, reduces associated challenges, and allows us to quickly revert to previous working configurations.

3.6 Conclusion

This chapter presented a novel methodology to develop version-controlled perception platforms. Our approach to building a sensor suite enhances the reproducibility of the system's state and is easily adapted to any existing perception platform. Our methodology relies only on standard tools in any modern operating system, namely git and docker. It is, from our point of view, key when building a complex robot or sensing infrastructure. This allows us to successfully build a containerized software stack that can run on multiple host machines without setting the host to a particular state. All the software and tools used to develop our perception platform are open-source for the benefit of the community. We envision this work as a step toward building more reliable perception platforms. Relying on such platforms can accelerate the research of new algorithms for robots.

Chapter 4

LiDAR-Based Pose Estimation

Robust and accurate pose estimation of a robotic platform from sensor data, the so-called sensor-based odometry, is essential for any robotic mapping application. Knowing the robot's pose allows us to describe its position and orientation in the environment accurately. A small error in the pose estimates can lead to significant inaccuracies in the resulting map, affecting the overall quality and usability for navigation, localization, and other tasks. Consequently, computing the robot's pose before integrating new measurements into the map is often a prerequisite. This chapter presents an effective solution to the pose estimation problem from sensor data that different mapping applications can use. The method presented in this chapter is transversal to the rest of the concepts presented in this thesis.

Our approach relies on point-to-point ICP combined with adaptive thresholding for correspondence matching, a robust kernel, a simple but widely applicable motion compensation approach, and a point cloud subsampling strategy. This yields a system with only a few parameters that, in most cases, do not even have to be tuned to a specific LiDAR sensor. Our system performs on par with state-of-the-art methods under various operating conditions using different platforms and the same parameters: automotive platforms, unmanned aerial vehicles (UAVs), self-stabilizing vehicles such as segways, or handheld LiDARs. We do not require integrating IMU data and solely rely on 3D point clouds obtained from a wide range of 3D LiDAR sensors, thus enabling a broad spectrum of different applications and operating conditions.

This chapter builds on the assumption that the robot can autonomously collect sensor data reliably, as explained in Chapter 3. However, the data obtained with the perception platform introduced in Sec. 3.4.1 is currently not accessible to the public. Therefore, to demonstrate the effectiveness of our method and make comparisons with existing approaches, we conduct experiments using publicly available real-world datasets.

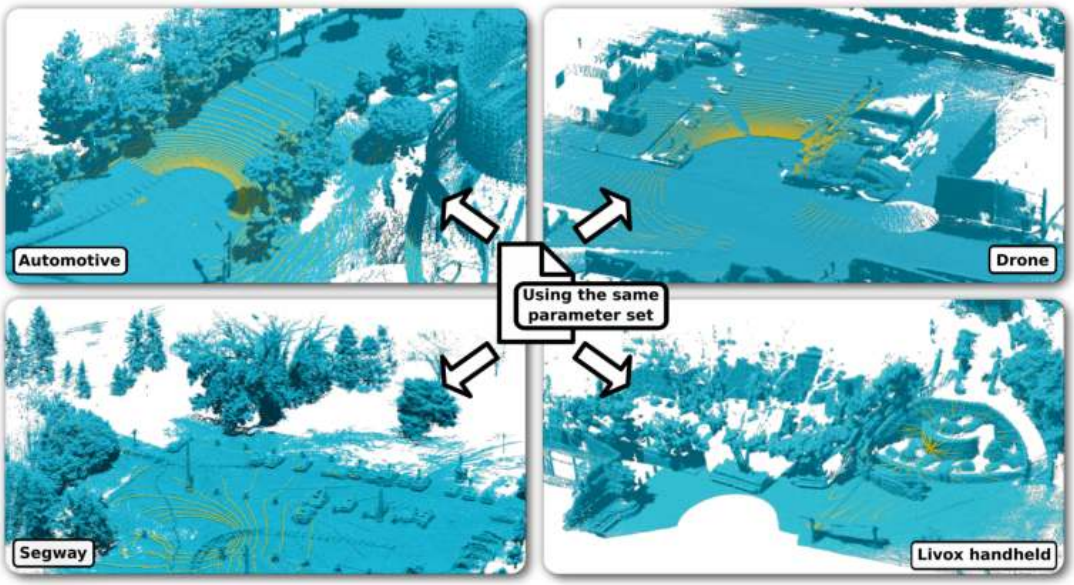


Figure 4.1: Point cloud maps (blue) generated by our proposed odometry pipeline on different datasets with the same set of parameters. We depict the latest scan in yellow. The scans are recorded using different sensors with different point densities, orientations, and shooting patterns. The automotive example stems from the MulRan dataset [80]. The drone of the Voxgraph dataset [146] and the segway robot used in the NCLT dataset [23] show a high acceleration motion profile. The handheld Livox LiDAR [98] has a completely different shooting pattern than the commonly used rotating mechanical LiDAR.

4.1 KISS-ICP – Keep It Small and Simple

The main contribution of this chapter is a simple yet highly effective approach for building LiDAR odometry systems that can accurately compute a robot's pose online while navigating through an environment. It does so in a setup as general as possible and with a small set of parameters.

Although many sensor odometry systems made progress by adding more complexity to the ego-motion estimation process, we move in the opposite direction. We identify the core components and adequately evaluate the impact of different modules on such systems. By removing most parts and focusing on the core elements, we obtain a surprisingly effective system that is simple to realize and can operate under various environmental conditions using different LiDAR sensors.

This chapter returns to the roots: classical point-to-point ICP, introduced 30 years ago by Besl and McKay [11]. We aim to tackle the inherent problems of sequentially operating LiDAR odometry systems that prohibit current approaches from generalizing to different environments, sensor resolutions, and motion profiles using a single configuration. We present simple yet effective reasoning about the robot kinematics exploiting and an effective downsampled point cloud representation that allows us to minimize the need for parameter tuning.

Our system challenges even extensively hand-tuned existing SLAM systems. Our design uses neither sophisticated feature extraction techniques nor learning methods. The same parameter set works in various challenging scenarios, such as highway drives of robot cars with many dynamic objects, drone flights, handheld devices, segways, etc. Thus, we take a step back from mainstream research in LiDAR odometry estimation and focus on reducing the components to their essentials. This makes our system perform extraordinarily well in various real-world scenarios, see Fig. 4.1 for a qualitative evaluation of our approach.

We show that we obtain competitive odometry results with the proper use of ICP that builds on basic reasoning about the system's physics and the nature of the sensor data. In addition to motion prediction, spatial scan downsampling, and a robust kernel, we introduce an adaptive threshold approach for ICP in the context of robot motion estimation that makes our approach effective and, at the same time, generalizes easily.

We make three key claims: Our "keep it small and simple" approach exploiting point-to-point ICP is (i) on par with state-of-the-art odometry systems, (ii) can accurately compute the robot's odometry in a large variety of environments and motion profiles with the same system configuration, and (iii) provides an effective solution to motion distortion without relying on IMUs or wheel odometers. In sum, "good old point-to-point ICP" is a surprisingly powerful tool, and there is little need to move to more sophisticated approaches if the essential components are done well. We provide an open-source implementation that follows precisely the description of this chapter[1] .

4.1.1 3D Point Cloud Registration for Pose Estimation

Our approach aims to compute the trajectory of a moving LiDAR sensor incrementally. This is achieved by sequentially aligning the captured point clouds from the scanner and determining the sensor's position and orientation relative to a global coordinate system, which defines the robot's six degrees of freedom.

For each 3D scan in the form of a local, egocentric point cloud $\mathcal{P} = \{\boldsymbol{p}_i \,|\, \boldsymbol{p}_i \in \mathbb{R}^3\}$, we perform the following four steps to obtain a global pose estimate $\mathsf{T}_t \in SE(3)$ at time t. First, we apply sensor motion prediction and motion compensation, often called deskewing, to undo the distortions of the 3D data caused by the sensor's motion during scanning. Second, to accelerate the computation, we subsample the current scan. Third, we estimate the correspondences between the input point cloud and a reference point cloud, which we call the local map. We use an adaptive thresholding scheme for correspondence estimation, restricting possible data associations, and filtering

[1] `https://github.com/PRBonn/kiss-icp`

out potential outliers. Fourth, we register the input point cloud to the local map using a robust point-to-point ICP algorithm. Finally, we update the local map with a downsampled version of the registered scan. Below, we describe these components in detail.

4.1.2 Motion Prediction and Scan Deskewing

We advocate for rethinking point cloud registration in the context of mobile robots, which continuously stream data. One should think of it as something other than registering arbitrary pairs of 3D point clouds. Instead, one should phrase it as *estimating how much the robot's actual motion deviates from its expected motion by registering consecutive scans.*

Different approaches can be used to compute the robot's expected motion before considering the LiDAR data. The three most popular choices are the constant velocity model, wheel odometry obtained through encoders, and IMU-based motion estimation. The constant velocity [182] model assumes that a robot moves with the same translational and rotational velocity as in the previous time step. It requires no additional sensors, wheel encoder, or IMU and, thus, is the most widely applicable option.

Our approach uses the constant velocity model for two reasons: first, it is generally applicable, requires no additional sensors, and avoids the need for time synchronization between sensors. Second, as we will show in our experimental evaluation, it works well enough to provide a solid initial guess when searching for data associations and deskewing 3D scans. This follows from the fact that robotic LiDAR sensors commonly record and stream point clouds at 10 Hz to 20 Hz, i.e., every 0.05 s to 0.1 s. In most cases, the acceleration or deceleration, i.e., the deviations from the constant velocity model that occurs within such short time intervals, are relatively small. If the robot accelerates or decelerates, the constant velocity estimation of the robot's pose will be slightly off, and therefore, we need to correct this estimate through registration. These accelerations determine the possible displacements of the (static) 3D points.

The constant velocity model approximates the translational and angular velocities, denoted as $\boldsymbol{v}_t \in \mathbb{R}^3$ and $\boldsymbol{\omega}_t \in \mathbb{R}^3$ at time t respectively, by using the previous pose estimates $\mathsf{T}_{t-1} = (\mathsf{R}_{t-1}, \boldsymbol{t}_{t-1})$ and $\mathsf{T}_{t-2} = (\mathsf{R}_{t-2}, \boldsymbol{t}_{t-2})$, represented by a rotation matrix $\mathsf{R}_t \in SO(3)$ and a translation vector $\boldsymbol{t}_t \in \mathbb{R}^3$ for the time step t. We first compute the relative pose $\mathsf{T}_{\text{pred},t}$ that we will use as motion prediction as:

$$\mathsf{T}_{\text{pred},t} = \begin{bmatrix} \mathsf{R}_{t-2}^{\top}\mathsf{R}_{t-1} & \mathsf{R}_{t-2}^{\top}(\boldsymbol{t}_{t-1} - \boldsymbol{t}_{t-2}) \\ \boldsymbol{0} & 1 \end{bmatrix}, \tag{4.1}$$

then derive the corresponding velocities as:

$$\boldsymbol{v}_t = \frac{\mathsf{R}_{t-2}^{\top}\left(\boldsymbol{t}_{t-1} - \boldsymbol{t}_{t-2}\right)}{\Delta t}, \tag{4.2}$$

$$\boldsymbol{\omega}_t = \frac{\mathrm{Log}(\mathsf{R}_{t-2}^{\top}\,\mathsf{R}_{t-1})}{\Delta t}, \tag{4.3}$$

where Δt is the acquisition time of one LiDAR sweep, typically 0.05 s or 0.1 s, and Log: $SO(3) \to \mathbb{R}^3$ extracts the axis-angle representation.

Note that wheel odometry or an IMU-based motion prediction approach can be used instead to compute $\boldsymbol{v}_t$ and $\boldsymbol{\omega}_t$ for each time step. However, we use constant velocity as a generally applicable approach.

Within the acquisition time Δt of one LiDAR sweep, multiple 3D points are measured by the scanner. The relative timestamp $\iota_i \in [0, \Delta t]$ for each point $\boldsymbol{p}_i \in \mathcal{P}$ describes the recording time relative to the scan's first measurement. This relative timestamp allows us to compute the motion compensation resulting in a deskewed point $\boldsymbol{p}_i^* \in \mathcal{P}^*$ of the corrected scan $\mathcal{P}^*$ reading by

$$\boldsymbol{p}_i^* = \mathrm{Exp}(\iota_i \boldsymbol{\omega}_t)\boldsymbol{p}_i + \iota_i \boldsymbol{v}_t, \tag{4.4}$$

where Exp: $\mathbb{R}^3 \to SO(3)$ computes a rotation matrix from an axis-angle representation. Note that $\mathrm{Exp}(\iota_i \boldsymbol{\omega}_t)$ is equivalent to performing SLERP in the axis-angle domain.

This form of scan deskewing, especially with the constant velocity model, is easy to implement, generally applicable, and does not require additional sensors, high-precision time synchronization between sensors, or IMU biases to be estimated. As we show in Sec. 4.2, this approach often performs even better than more complex compensation systems [52], at least as long as the motion between the start and end of the sweep is small, as it is for most robotics applications.

4.1.3 Point Cloud Subsampling

Identifying a set of keypoints in the point cloud is a common approach for scan registration [71, 154, 219]. It is typically done to achieve faster convergence and higher robustness in the data association. However, complex filtering of the point cloud usually comes with an extra layer of complexity and parameters that often need to be tuned.

Rather than extracting 3D keypoints, which often require environment-dependent parameter tuning, we propose to compute only a spatially downsampled version $\hat{\mathcal{P}}^*$ of the deskewed scan $\mathcal{P}^*$. Downsampling is done using a voxel grid. As we explain in Sec. 4.1.4 below in more detail, we use a voxel grid as our local map, where each voxel cell has a size of $v_{\text{size}} \times v_{\text{size}} \times v_{\text{size}}$ and each cell only stores a certain number of points. Each

time we process an incoming scan, we first downsample the point cloud of the scan to an intermediate point cloud $\mathcal{P}^*_{\text{merge}}$, which is later used to update the map when the relative motion of the robot has been determined with ICP. To obtain the points in $\mathcal{P}^*_{\text{merge}}$, we use the voxel size $\alpha\, v_{\text{size}}$ with $\alpha \in (0.0, 1.0]$ and keep only one point per voxel.

For ICP registration, an even lower resolution scan is beneficial. Therefore, we compute a further reduced point cloud $\hat{\mathcal{P}}^*$ by downsampling $\mathcal{P}^*_{\text{merge}}$ again using a voxel size of $\beta\, v_{\text{size}}$ with $\beta \in [1.0, 2.0]$ keeping only a single point per voxel. This further reduces the number of points processed during registration and allows for a fast and highly effective alignment. The idea of this "double downsampling" stems from CT-ICP [52], a top-performing open-source LiDAR odometry system on KITTI.

However, most voxelization approaches select the center of each occupied voxel to downsample the point cloud [155, 221]. Instead, we found it advantageous to maintain the original point coordinates, select only one point per voxel for a single scan, and keep its coordinates to avoid discretization errors. This means the reduced cloud is a subset of the deskewed one, i.e., $\hat{\mathcal{P}}^* \subseteq \mathcal{P}^*$. In our implementation, we keep only the first point inserted into the voxel.

4.1.4 Local Map and Correspondence Estimation

In line with prior work [6, 52, 126, 219], we register the deskewed and subsampled scan $\hat{\mathcal{P}}^*$ to the point cloud built so far, i.e., a local map, to compute an incremental pose estimate $\Delta\mathsf{T}_{\text{icp},t}$ at timestamp t. We use frame-to-map registration as it proves more reliable and robust than the frame-to-frame alignment [6, 126]. To do that effectively, we must define a data structure representing the previously registered scans.

Modern approaches have used very different types of representations for this local map. Popular approaches are voxel grids [219], triangle meshes [188], surfel representations [6], or implicit representations [53]. As mentioned in Sec. 4.1.3, we utilize a voxel grid to store a subset of 3D points. We use a grid with a voxel size of $v_{\text{size}} \times v_{\text{size}} \times v_{\text{size}}$ and store up to $N_{\max}$ points per voxel. After registration, we update the voxel grid by adding the points $\{\mathsf{T}_t\, \boldsymbol{p}_i \mid \boldsymbol{p}_i \in \mathcal{P}^*_{\text{merge}}\}$ from the new scan using the global pose estimate T_t. Voxels containing $N_{\max}$ points are not updated anymore. Given the current pose estimate, we remove voxels outside the maximum range $r_{\max}$. Thus, the size of the map will stay bounded.

Instead of a 3D array, we use a hash table to store the voxels, allowing a memory-efficient representation and a fast nearest neighbor search [52, 128]. However, the data structure used can be easily replaced with VDBs [125, 190], Octrees [187, 216], or KD-Trees [9].

4.1.5 Adaptive Threshold for Data Association

ICP typically performs a nearest neighbor data association to find corresponding points between two point clouds [11]. When searching for associations, it is common to impose a maximum distance between the corresponding points, often using a value of 1 m or 2 m [6, 188, 219]. This maximum distance threshold can be seen as an outlier rejection scheme, as all correspondences with a distance larger than this threshold are considered outliers and are ignored.

The required value for this threshold τ depends on the expected initial pose error, the number, the type of dynamic objects in the scene, and, to some degree, the sensor noise. It is typically selected heuristically. Based on the considerations about the constant velocity motion prediction in Sec. 4.1.2, we can, however, estimate a likely limit from the data by analyzing how much the odometry may deviate from the motion prediction over time. This deviation $\Delta\mathsf{T}$ in the pose corresponds precisely to the local ICP correction to be applied to the predicted pose (but it is unknown beforehand). Intuitively, we can observe the robot's acceleration in the $\Delta\mathsf{T}$ magnitude. If the robot does not accelerate, then $\Delta\mathsf{T}$ will have a small magnitude, often around zero, which means that the constant velocity assumption holds and no correction has to be done by ICP.

We integrate this information into our data association search by exploiting the so-far successful ICP executions. We can estimate the possible point displacement between corresponding points in successive scans in the presence of a potential acceleration expressed through $\Delta\mathsf{T}$ as:

$$\delta(\Delta\mathsf{T}) = \delta_{\text{rot}}(\Delta\mathsf{R}) + \delta_{\text{trans}}(\Delta\boldsymbol{t}), \tag{4.5}$$

where $\Delta\mathsf{R} \in SO(3)$ and $\Delta\boldsymbol{t} \in \mathbb{R}^3$ refer to the rotational and translational component of the deviation, given by

$$\delta_{\text{rot}}(\Delta\mathsf{R}) = 2\ r_{\max} \sin\left(\frac{1}{2}\underbrace{\arccos\left(\frac{\text{tr}(\Delta\mathsf{R})-1}{2}\right)}_{\theta}\right), \tag{4.6}$$

$$\delta_{\text{trans}}(\Delta\boldsymbol{t}) = \|\Delta\boldsymbol{t}\|_2, \tag{4.7}$$

where the trace operator tr is defined as the sum of elements on the main diagonal of the square matrix $\Delta\mathsf{R}$.

The term $\delta_{\text{rot}}(\Delta\mathsf{R})$ represents the displacement that occurs for a range reading with maximum range $r_{\max}$ subject to the rotation $\Delta\mathsf{R}$, see also Fig. 4.2. Note that Eq. (4.5) constitutes an upper bound for the point displacement as

$$\|\Delta\mathsf{R}\,\boldsymbol{p} + \Delta\boldsymbol{t} - \boldsymbol{p}\|_2 \leq \delta_{\text{rot}}(\Delta\mathsf{R}) + \delta_{\text{trans}}(\Delta\boldsymbol{t}), \tag{4.8}$$

which follows from the triangle inequality.

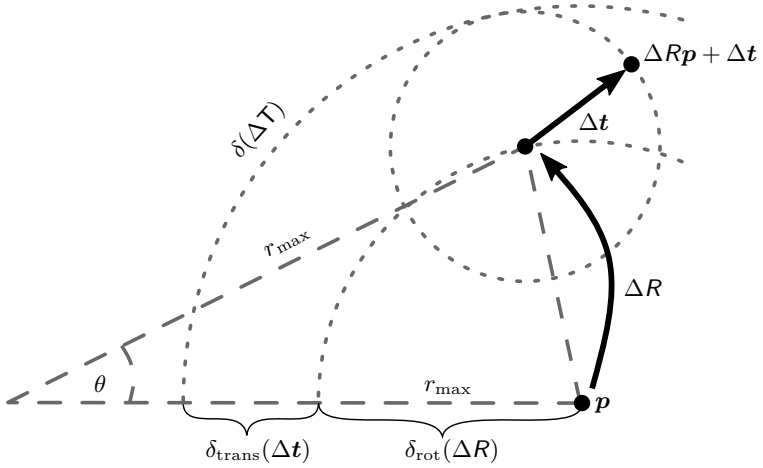


Figure 4.2: Exemplary computation of the maximum point displacement $\delta(\Delta \mathsf{T})$ caused by a rotational and translational deviation $(\Delta \mathsf{R}, \Delta \boldsymbol{t})$ from the predicted motion.

For obtaining δ_{rot}, other approaches could be considered, such as considering individual ranges for the adaptive threshold computation [13]. In our tests, we did not see any difference in the results but a 3-fold increase of the overall runtime; thus, we use $r_{\max}$ instead of the individual range of each point to calculate δ_{rot}.

To compute the threshold τ_t at time t, we consider a Gaussian distribution over δ using the values of Eq. (4.5) over the trajectory computed so far whenever the deviation was larger than a minimum distance $\delta_{\min}$, i.e., situations where the robot motion was deviating from the constant velocity model. Its standard deviation is

$$\sigma_t = \sqrt{\frac{1}{|\mathcal{J}_t|} \sum_{i \in \mathcal{J}_t} \delta(\Delta \mathsf{T}_i)^2}, \tag{4.9}$$

where the index set $\mathcal{J}_t$ of deviations up to t is given by

$$\mathcal{J}_t = \{i \mid i < t \wedge \delta(\Delta \mathsf{T}_i) > \delta_{\min}\}. \tag{4.10}$$

This avoids reducing the value of σ_t too much when the robot is not moving or moving at a constant velocity for a long time. In our experiments, we set this threshold $\delta_{\min}$ to 0.1 m. We then compute the threshold τ_t as the three-sigma bound $\tau_t = 3\,\sigma_t$, which we use in the next section for data association.

4.1.6 Alignment Through Robust Optimization

We base our registration on point-to-point ICP [11]. The advantage of this choice is that we do not need to compute data-dependent features such as normals, curvature, or other descriptors, which may depend on the scanner or the environment. Furthermore, with noisy or sparse LiDAR scanners, features such as normals are

often unreliable. Thus, neglecting quantities such as normals in the alignment process is an explicit design decision that allows our system to generalize well to different sensor resolutions.

To obtain the global estimation of the pose T_t of the robot, we start by applying our prediction model $\mathsf{T}_{\text{pred},t}$ to the scan $\hat{\mathcal{P}}^*$ in the local frame. Successively, we transform it into the global coordinate frame using the previous pose estimate T_{t-1}, resulting in the source points

$$\mathcal{S} = \left\{ \boldsymbol{s}_i = \mathsf{T}_{t-1}\mathsf{T}_{\text{pred},t}\,\boldsymbol{p}_i \;\middle|\; \boldsymbol{p}_i \in \hat{\mathcal{P}}^* \right\}. \tag{4.11}$$

For each iteration j of ICP, we obtain a set of correspondences between the point cloud $\mathcal{S}$ and the local map $\mathcal{Q} = \{\boldsymbol{q}_i \mid \boldsymbol{q}_i \in \mathbb{R}^3\}$ through nearest neighbor search over the voxel grid (Sec. 4.1.4) considering only correspondences with a point-to-point distance below τ_t. To compute the current pose correction $\Delta\mathsf{T}_{\text{est},j}$, we perform a robust optimization minimizing the sum of point-to-point residuals

$$\Delta\mathsf{T}_{\text{est},j} = \underset{\mathsf{T}}{\operatorname{argmin}} \sum_{(\boldsymbol{s},\boldsymbol{q})\in\mathcal{C}(\tau_t)} \rho\Big(\big\|\mathsf{T}\boldsymbol{s} - \boldsymbol{q}\big\|_2\Big), \tag{4.12}$$

where $\mathcal{C}(\tau_t)$ is the set of nearest neighbor correspondences with a distance smaller than τ_t and ρ is the Geman-McClure robust kernel [2, 29], i.e., an M-estimator with a strong outlier rejection property, given by

$$\rho(e) = \frac{e^2/2}{\underbrace{\sigma_t/3}_{\kappa_t} + e^2}, \tag{4.13}$$

where the scale parameter κ_t of the kernel is adapted online using σ_t. Lastly, we update the points $\boldsymbol{s}_i$, i.e.,

$$\left\{ \boldsymbol{s}_i \leftarrow \Delta\mathsf{T}_{\text{est},j}\,\boldsymbol{s}_i \mid \boldsymbol{s}_i \in \mathcal{S} \right\}, \tag{4.14}$$

and repeat the process until the convergence criterion is met.

As a result of this process, we obtain the transformation $\mathsf{T}_t = \Delta\mathsf{T}_{\text{icp},t}\mathsf{T}_{t-1}\mathsf{T}_{\text{pred},t}$, where $\Delta\mathsf{T}_{\text{icp},t} = \prod_j \Delta\mathsf{T}_{\text{est},j}$. While we apply the prediction model $\mathsf{T}_{\text{pred},t}$ (i.e., the constant velocity prediction) to the local coordinate frame of the scan, we perform the ICP correction $\Delta\mathsf{T}_{\text{icp},t}$ in the global reference frame of the robot. This is done for efficiency reasons as it allows us to transform the source points $\mathcal{S}$ only once per ICP iteration. With this, the local pose deviation $\Delta\mathsf{T}_t$ at time t used in the Eq. (4.5) can be expressed as

$$\Delta\mathsf{T}_t = \left(\mathsf{T}_{t-1}\mathsf{T}_{\text{pred},t}\right)^{-1} \Delta\mathsf{T}_{\text{icp},t}\mathsf{T}_{t-1}\mathsf{T}_{\text{pred},t}. \tag{4.15}$$

A standard termination criterion for ICP is to control the number of iterations. Most approaches also have a further criterion based on the minimum change in

Parameter	Value
Initial threshold τ_0	$2\,\mathrm{m}$
Min. deviation threshold $\delta_{\min}$	$0.1\,\mathrm{m}$
Max. points per voxel $N_{\max}$	20
Voxel size map v_{size}	$0.01\,r_{\max}$
Factor voxel size map merge α	0.5
Factor voxel size registration β	1.5
ICP convergence criterion γ	10^{-4}

Table 4.1: All seven parameters of our approach. $r_{\max}$ is the maximum range of the sensor.

the solution. On the contrary, we found that controlling the number of iterations does not allow the algorithm to always find a solution. Therefore, we only employ the termination criterion based on the applied correction being smaller than γ, without imposing a maximum number of iterations.

Finally, the ICP correction is applied to the point cloud $\mathcal{P}^*_{\text{merge}}$, and the points are integrated into the local map.

4.1.7 Parameters

Our implementation depends on a small set of seven parameters. All are shown in Tab. 4.1. We used the same parameters for all experiments. Most other approaches use a substantially larger set of parameters: MULLS [133] has 107 parameters, SuMa [6] has 49 parameters, and CT-ICP [52] has 30 parameters in their respective configuration files. In contrast, our approach only has two parameters for the correspondence search (τ_0 and $\delta_{\min}$), four for the map representation ($N_{\max}$, v_{size}, α, and β) and scan subsampling, and one for the ICP termination (γ). Note that the maximum range of a scanner is a value that depends on the specific sensor in use. As such, we do not consider it a system parameter. However, for some scenarios, the value of $r_{\max}$ might also be adapted to the specific environment in which the system operates, e.g., not considering far away measurements that are usually less accurate.

4.2 Experimental Evaluation

Our approach provides a simple yet effective LiDAR odometry pipeline with a small set of parameters. We present our experiments to show the capabilities of our method. The results of our experiments support our key claims, namely that our approach (i) is on par with more complex state-of-the-art odometry systems, (ii) can accurately compute the robot's odometry in a large variety

of environments and motion profiles with the same system configuration, and (iii) provides an effective solution to motion distortion without relying on IMUs or wheel odometers.

4.2.1 Experimental Setup

We use numerous datasets and common evaluation methods. We start with the KITTI odometry dataset [63] to evaluate our system against state-of-the-art approaches to LiDAR odometry. To investigate how we perform in other datasets of autonomous driving that employ a different sensor, we evaluated our approach in the MulRan dataset [80]. Furthermore, we show that our approach can be used in different scenarios, such as the one present in the NCLT dataset [23], a segway dataset, and the Newer College dataset [144] recorded using a handheld device. We also analyze our method's different components, such as the motion-compensation scheme and the adaptive threshold.

4.2.2 Performance on the KITTI-Odometry Benchmark

This experiment evaluates the performance of different odometry pipelines on the popular KITTI benchmark dataset. Since most systems do not do motion compensation, we use the already compensated KITTI scans for a fair comparison and turn off motion compensation for our approach and for CT-ICP [52], the performance of the motion compensation module will be studied later in Sec. 4.2.4.1. Tab. 4.2 exhibits how our system challenges most state-of-the-art systems, which are typically more sophisticated than our point-to-point ICP. Based on the official KITTI benchmark, at the time of submission of KISS-ICP [191] on January 2023, we rank second among the open-source approaches (behind CT-ICP) and ninth among all submissions. This indicates that our comparably simple system performs better than all publicly available systems, except CT-ICP. Note that CT-ICP is a complete SLAM system, and it uses loop closures to correct for the accumulated drift of the odometry estimation. We, in contrast, obtain our results using only open-loop registration without any loop closing.

4.2.3 Comparison to State-of-the-Art Systems

We proceed to analyze the performance of our system on different datasets, scenarios, and types of robots. For that, we use the MulRan dataset [80], a handheld device [144], and a segway dataset [23]. Odometry pipelines typically deal with those challenging scenarios but employ IMUs [164] or a different system configuration [52]. Our system performs on par with state-of-the-art systems using the same parameter values for all experiments and datasets. For this experiment,

	Method	Seq. 00-10	Seq. 11-21
SLAM	SuMa++ [6]	0.70	1.06
	MULLS [133]	0.52	-
	CT-ICP [52]	0.53	**0.59**
Odometry	IMLS-SLAM [53]	0.55	0.69
	MULLS [133]	0.55	0.65
	F-LOAM [194]	0.84	1.87
	SuMa [6]	0.80	1.39
	KISS-ICP [191]	**0.50**	0.61

Table 4.2: KITTI Benchmark results with motion compensated data. We report the average relative translational error in % [64]. These results are then averaged across all sequences being studied. We compare across SLAM methods employing pose-graph optimization for improved results *(top)* and odometry methods *(bottom)*. We omit the relative rotational error, but these results are available at `https://www.cvlibs.net/datasets/kitti/eval_odometry.php`.

we compare with the state-of-the-art odometry systems, namely MULLS [133], SuMa [6], F-LOAM [194], and CT-ICP [52]. Note that we do not provide an evaluation of CT-ICP for the MulRan dataset since CT-ICP does not provide support for this dataset.

For the MulRan dataset [80], we tested the systems under evaluation on all available public sequences. Since the dataset provides three similar runs for each sequence, we report the average number of each sequence in Tab. 4.3. Our method outperforms all state-of-the-art approaches by a large margin in relative and absolute error.

We used both available sequences to evaluate the Newer College dataset and achieved similar results in the short experiment compared to CT-ICP. For the long experiment, the performance gap can be explained by the additional loop closing module of CT-ICP, a complete SLAM system. For the NCLT dataset experiment, we used the sequence evaluated in the original work of CT-ICP. We could not reproduce the results reported in CT-ICP and, therefore, report the results in the original manuscript [52] in Tab. 4.4. We achieve similar results than CT-ICP. However, we observed errors in the GPS ground-truth poses and missing frames. Therefore, the numbers reported for the NCLT dataset should be taken with a grain of salt and only serve as an estimate of how the systems perform. We discourage using NCLT to evaluate odometry systems: misalignments in the ground-truth poses, missing frames, and inconsistencies in the data turn the evaluation of odometry systems on such a dataset not a good evaluation tool from our perspective. However, we provide the results for completeness.

We show qualitative results in Fig. 4.1 generated using our KISS-ICP poses.

Sequence	Method	Avg. tra.	Avg. rot.	ATE tra.	ATE rot.
KAIST	MULLS [133]	2.94	0.86	37.24	0.11
	SuMa [6]	5.59	1.73	43.61	0.14
	F-LOAM [194]	3.43	0.99	46.17	0.15
	KISS-ICP [191]	**2.28**	**0.68**	**17.40**	**0.06**
DCC	MULLS [133]	2.96	0.98	38.35	0.12
	SuMa [6]	5.20	1.71	36.22	0.11
	F-LOAM [194]	3.83	1.14	42.70	0.13
	KISS-ICP [191]	**2.34**	**0.64**	**15.16**	**0.05**
Riverside	MULLS [133]	5.42	2.21	91.16	0.16
	SuMa [6]	13.86	2.13	227.24	0.38
	F-LOAM [194]	5.47	1.18	138.09	0.22
	KISS-ICP [191]	**2.89**	**0.64**	**49.02**	**0.08**
Sejong*	MULLS [133]	5.93	0.84	2151.00	0.49
	F-LOAM [194]	7.87	1.20	3448.97	0.82
	KISS-ICP [191]	**4.69**	**0.70**	**1369.54**	**0.33**

Table 4.3: Quantitative results on the MulRan dataset. We report the relative translational error and the relative rotational error using the KITTI [64] metrics. Additionally, we show the absolute trajectory error for translation in m and for rotation in rad. As the dataset provides three similar runs for each sequence, we report the average number of each sequence.

Method	NCD 01-short	NCD 02-long	NCLT 2012-01-8
MULLS [133]	0.82	1.23	-
F-LOAM [194]	2.02	fails	-
CT-ICP [52]	**0.48**	**0.58**	**1.17**
KISS-ICP [191]	0.51	0.96	1.27

Table 4.4: Quantitative results for Newer College and NCLT. We report the relative translational error in % [64].

Using a single system configuration, we can produce consistent maps on different sensor setups (Velodyne/Ouster vs. Livox) and different motion profiles (car, drone, segway, handheld) with the same parameters.

Method	Avg. tra	Avg. rot	Avg. freq.
MULLS [133]	1.41	-	12 Hz
IMLS-SLAM [53]	0.71	-	1 Hz
CT-ICP [52]	0.55	-	15 Hz
KISS-ICP [191] without deskewing	0.91	0.27	51 Hz
Ours + Deskewing (IMU)	0.51	0.19	**38 Hz**
Ours + Deskewing (CV)	**0.49**	**0.16**	**38 Hz**

Table 4.5: Results of evaluating different state-of-the-art systems on KITTI-raw dataset (without motion compensation). We report the relative translational error and the relative rotational error using the KITTI [64] metrics. Additionally, we report the runtime operation of the systems being in consideration for this experiment.

4.2.4 Ablation Studies

To understand how each component of our system impacts the odometry performance, we perform ablation studies on the different components of our approach, namely, the motion compensation scheme, the adaptive threshold, and the robust kernel. To carry out these studies, we use KITTI odometry dataset [63] as it is probably the best-known one.

4.2.4.1 Motion Compensation

To assess the impact of our motion compensation scheme, we utilize the raw LiDAR point clouds without any compensation applied. Note that the KITTI odometry benchmark point cloud data [63] is already compensated for and, therefore, cannot be used for this study. Thus, we use the KITTI raw dataset [64]. We present the results in a familiar fashion, selecting only the sequences corresponding to those in the motion-compensated dataset [63]. As we can see in Tab. 4.5, our motion compensation scheme can produce state-of-the-art results and is on par with substantially more sophisticated and thus complex compensation techniques such as the one introduced by CT-ICP [52]. Additionally, we study how our system performs without applying motion compensation, as shown in Tab. 4.5. We also evaluate the performance of our constant velocity model for motion compensation. To assess this, we compare the same compensation strategy but replace the velocity estimation with sensor data taken by the IMU. The results show that our velocity estimation is on par or even slightly better with the IMU.

Besides the fact that CT-ICP's elastic formulation yields good results, our simpler approach produces even better results. This result shows that the constant velocity model employed in our approach for compensating motion distortion is

Dataset	Data-Association Threshold τ				
	0.3 m	0.5 m	1.0 m	2.0 m	Ours
KITTI Seq. 00	0.54	**0.51**	0.53	0.55	**0.51**
KITTI Seq. 04	0.39	0.41	0.37	0.39	**0.36**
KITTI Avg. Seq. 00-10	0.53	0.51	0.51	0.53	**0.50**

Table 4.6: Comparison of different fixed thresholds vs. our proposed adaptive threshold on the KITTI dataset. We report the relative translational error in % [64].

sufficient to cope with the slight reduction in performance when no compensation is applied. Consequently, we believe that more sophisticated techniques are unnecessary for odometry estimation for *most* robotic vehicles. In contrast, devices operating under extreme accelerations could benefit from an IMU here.

4.2.4.2 Adaptive Data-Association Threshold

We finally assess how the adaptive threshold τ_t impacts the performance of our system by comparing it to a different set of fixed thresholds commonly used in open-source systems. To conduct this experiment, we identify the two KITTI sequences with the largest (00) and the smallest (04) average acceleration, indicating different motion profiles as we can see in Tab. 4.6, the best-fixed threshold for sequence 00 is 0.5 m and 1.0 m for sequence 04. This means that a fixed threshold has to be tuned depending on the motion profile and, thus, to the dataset to achieve top performance. In contrast, our adaptive algorithm exploits the motion profile to estimate the threshold online, resulting in on-par or better performance without finding a new fixed threshold for each sequence. Finally, our proposed adaptive threshold strategy achieves the best average result in the KITTI training sequences. Please note that all the experiments from this ablation study use the robust kernel.

4.2.4.3 Impact of Using a Robust Kernel

In this subsection, we assess the impact of the selected outlier rejection technique in the registration pipeline. To carry out this ablation study, we disable the robust kernel entirely from the optimization loop and run the system on all the train sequences of the KITTI dataset. The results are shown in Tab. 4.7. We show the metrics for each dataset sequence to allow for a more fine-grain level of detail. As observed, using a robust kernel in the optimization significantly improves the odometry performance. We can see in the results averaged over the sequences that not using the kernel produces 0.67% for the translational error and 0.25% for the rotational error [64] compared to 0.50% and 0.15% respectively.

Figure 4.3: Left, registration results using a Livox LiDAR [98] with the robust kernel, and right, results on the same datasets without the robust kernel.

Translational error results using KITTI Metrics [64]

Method	avg.	00	01	02	03	04	05	06	07	08	09	10
w kernel	**0.50**	**0.51**	**0.63**	**0.51**	0.66	**0.35**	**0.31**	**0.26**	**0.33**	**0.82**	**0.50**	**0.56**
w/o kernel	0.67	0.74	0.66	0.75	**0.63**	0.41	0.42	0.28	0.50	1.06	0.89	1.02

Rotational error results using KITTI Metrics [64]

Method	avg.	00	01	02	03	04	05	06	07	08	09	10
w kernel	**0.15**	**0.18**	0.15	**0.15**	**0.16**	0.15	**0.14**	**0.08**	**0.17**	**0.18**	**0.13**	**0.19**
w/o kernel	0.25	0.30	**0.13**	0.28	0.22	**0.13**	0.28	0.10	0.37	0.29	0.30	0.45

Table 4.7: Ablation study on the use of a robust kernel for the optimization of the training sequence of the KITTI Odometry dataset. We use the adaptive threshold we propose in our approach for both runs. We report the relative translational and rotational errors using the KITTI [64] metrics.

Upon examining urban sequences like 00 or 07, we observe a significant degradation in pose estimates when robust kernels are not utilized. Contrary to popular belief [36, 38, 40], applying robust kernels doesn't drastically alter the outcomes on highway sequences with dynamic objects, such as 01 or 04.

Additionally, we show a qualitative result in Fig. 4.3 of one of the sequences of the handheld Livox LiDAR [98] dataset, where KISS-ICP completely fails when we turn off the robust kernel.

4.3 Conclusion

This chapter presents a simple yet highly effective approach to LiDAR odometry and shows that point-to-point ICP works well – when used properly. Our approach operates solely on point clouds and does not require an IMU, even when dealing with high-frequency driving profiles. Our approach exploits the classical point-to-point ICP to build a generic odometry system that can be used in different challenging environments, such as highway runs, and for different types

of robotics systems, such as handheld devices, segways, and drones. Moreover, our system operates efficiently with reliance on just a few parameters and can use different range-sensing technologies and scanning patterns. We only assume that point clouds are generated sequentially as the robot moves through the environment. We implemented and evaluated our approach on different datasets, provided comparisons with other existing techniques, supported all claims made in this chapter, and released our code. The experiments suggest that our approach is on par with substantially more sophisticated state-of-the-art LiDAR odometry systems and performs well on various datasets under different conditions with the same parameter set. Finally, our system operates faster than the sensor frame rate in all presented datasets. We believe this work will be a new baseline for future sensor odometry systems and a solid, high-performance starting point for future approaches. Our open-source code is robust, simple, easy to extend, and performs well, pushing the state-of-the-art LiDAR odometry to its limits and challenging the most sophisticated systems.

Chapter 5

Offline Mapping Using Poisson Surface Reconstruction

Offline mapping systems can enhance robotics tasks such as localization or place recognition. Consider a localization scenario as an example. The preliminary step to facilitate a robot to localize within a map is to build that map. This map-building step typically happens after a data collection phase using, for example, a perception platform such as the IPB-Car, described in Chapter 3. Intuitively, the more accurate these maps are, the more precise the localization will be.

This chapter presents a novel approach to offline 3D LiDAR mapping, focusing on improving mapping quality. Note that since the map's construction occurs offline, there are no hard runtime constraints, which we exploit to enhance the level of detail in the produced maps. We represent the map as a triangle mesh computed via Poisson surface reconstruction. This enables us to build a 3D map showing more geometric details than standard mapping approaches that rely on a truncated signed distance function or surfels. Our experimental evaluation indicates quantitatively and qualitatively that our maps offer higher geometric accuracies than other map representations.

In addition, achieving precise scan registration may be necessary to enhance the quality of the model. For example, when external sources of odometry are used, such as KISS-ICP, discussed in Chapter 4, the scans are not guaranteed to align accurately with the generated mesh map. This is because KISS-ICP utilizes a point cloud map representation and operates independently of the mapping algorithm proposed in this chapter. To address this issue, this chapter presents a novel frame-to-mesh ICP technique for registering new scans in the mesh. This approach utilizes a ray-casting-based data association, which facilitates the alignment of scans to the mesh. Additionally, we show how this method can be used for LiDAR-based odometry estimations even without any other odometry source.

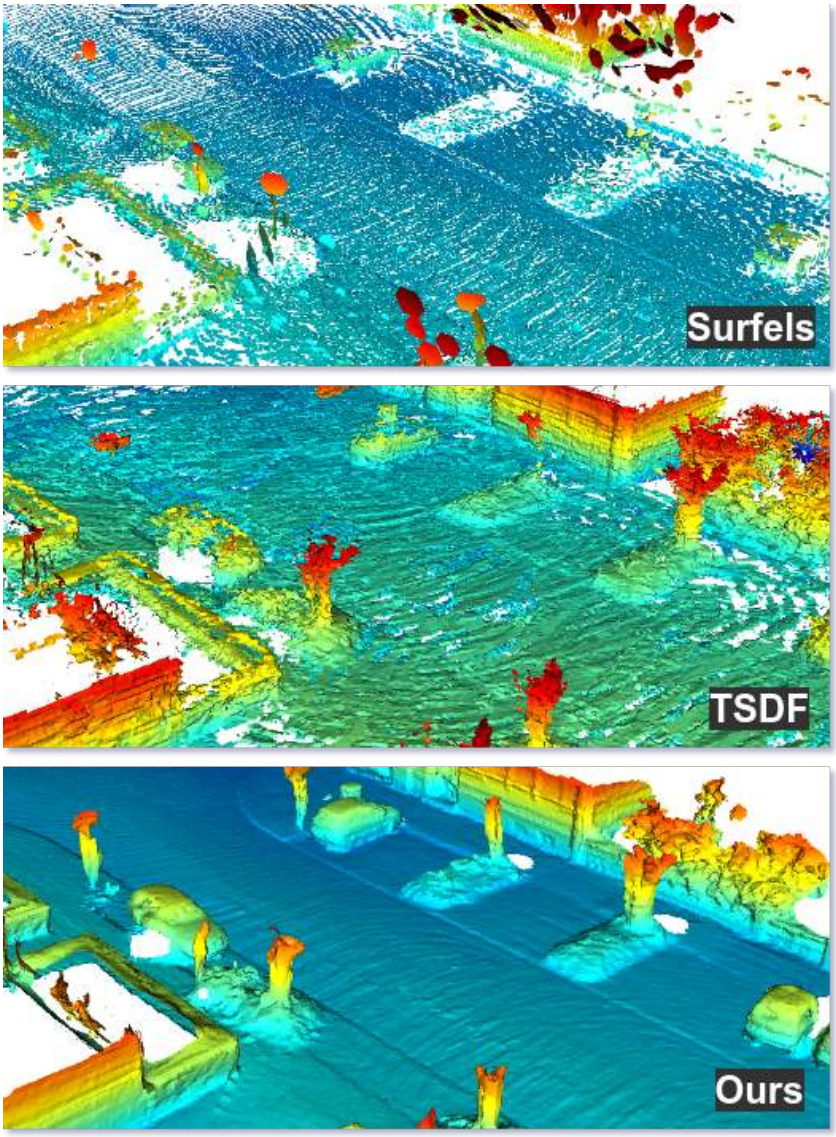


Figure 5.1: Qualitative comparison between the different mapping techniques for sequence 00 of the KITTI odometry benchmark [63].

5.1 Poisson Surface Reconstruction for 3D Mapping

The main contribution of this chapter is a novel LiDAR odometry and mapping system that builds upon a surface reconstruction method providing accurate geometrical maps. Most autonomous systems cannot navigate effectively without a map of the environment and knowledge of their pose. Thus, localization, mapping, and SLAM [18, 171] are essential building blocks of autonomous systems.

Here, we investigate the use of an alternative scene representation for mapping and registration. Scene representation is essential since it is used to register

incoming scans. To obtain accurate relative pose estimates and compelling mapping results, a scene representation must capture and represent the environment with a high level of detail.

This chapter aims to improve the geometrical accuracy of LiDAR-based mapping while simultaneously estimating the vehicle's pose with low drift over time. We achieve this using a triangle mesh representation computed using the Poisson surface reconstruction technique [85]. This is in contrast to other state-of-the-art approaches, which often use either surfels [6] or a truncated signed distance function [44, 126] as a representation, that usually provides a relatively low reconstruction quality, at least for large outdoor scenes. With our approach, we reconstruct meshes from 3D LIDAR data for outdoor environments with a quality previously common only at the object level, for indoor settings, by using terrestrial scanners, or by aggregating multiple passes over the same scene.

We aggregate individual scans into a local point cloud and use these to reconstruct a triangular scene mesh. Our experimental evaluation shows that such a triangle mesh is well-suited for registering 3D LiDAR scans. It is comparably compact, preserves rather detailed structures, and allows for accurate frame-to-mesh registration. This yields a new 3D LiDAR-based mapping approach that provides geometrically accurate maps and can be used for pose estimation, as shown in Fig. 5.1. We show that the proposed map representation (i) is a geometrically accurate representation of the environment, (ii) has better memory efficiency compared to other map representations, and (iii) allows for accurate registration of incoming scans with the model using a novel frame-to-mesh registration algorithm. We support these claims through experimental evaluation on synthetic and real-world data. The work described in this chapter has been published in a peer-review conference [188], and the source code of our approach is open source as a software package called PUMA[1].

5.1.1 Approach Overview

We perform the following three steps for each scan: First, we compute per-point normals, second, we register the scan to the local map, and third, we fuse the registered scan into a global map. We furthermore propose a novel frame-to-mesh registration strategy that exploits the fact that the map is a mesh.

Our approach distinguishes between a local map and a global map. The local map is used for the odometry estimation and is built from the last N localized scans. The global map is the aggregated mesh of the entire environment.

[1] `https://github.com/PRBonn/puma`

5.1.2 Normal Computation

The Poisson surface reconstruction algorithm requires an oriented point cloud, thus requiring normals for all the points in the input cloud. For computing this, we project the point cloud into the vertex map $\mathcal{V}_D : \mathbb{R}^2 \to \mathbb{R}^3$ mapping a 2D coordinate $(u, v) \in \mathbb{R}^2$ to a point $(x, y, z) \in \mathbb{R}^3$. We estimate the normal vectors using the cross product of neighboring pixels [6]. Although this is sometimes less accurate than estimating normals via principal component analysis on the covariance of a point neighborhood, it is far more efficient as it does not need to determine a point neighborhood. The normal map $\mathcal{N}_D$ is calculated for each coordinate (u, v) using cross products over forward differences of the corresponding vertex map pixel, i.e.,

$$\begin{aligned}\mathcal{N}_D((u,v)) = (\mathcal{V}_D((u+1,v)) - \mathcal{V}_D((u,v))) \\ \times (\mathcal{V}_D((u,v+1)) - \mathcal{V}_D((u,v)))\,.\end{aligned} \tag{5.1}$$

5.1.3 Point Cloud Registration Between Scans and Triangle Mesh

For point cloud registration, we perform data association of the point cloud with the triangle mesh and determine pose increments to minimize an error metric.

For the data association between point clouds, the closest point association found via neighbor search is a common choice [154]. However, as we show in the experimental evaluation, this is suboptimal. We propose to use ray-casting to determine ray-triangle intersections. For each intersection, we extract the point and associated normal of the intersected triangle. To this end, we first apply the last estimated pose at time $t-1$, i.e., $\mathsf{T}_{t-1} \in SE(3)$, to the current scan as an initial alignment. We then create a set of rays $\mathcal{R} = \{\boldsymbol{r}_i \,|\, \boldsymbol{r}_i \in \mathbb{R}^3\}$. Each ray $\boldsymbol{r}_i$ is defined by:

$$\boldsymbol{r}_i(\gamma) = \boldsymbol{o}_i + \gamma \frac{\boldsymbol{d}_i}{\|\boldsymbol{d}_i\|_2}, \tag{5.2}$$

with the origin $\boldsymbol{o}_i = \boldsymbol{t}_t$ at the currently estimated sensor position and directions $\boldsymbol{d}_i = \mathsf{T}_t \boldsymbol{p}_i$, passing through all points $\boldsymbol{p}_i$ of the current scan. Here, T_t is the estimated pose, and $\boldsymbol{t}_t \in \mathbb{R}^3$ is the translational part of T_t.

The intersection of each ray $\boldsymbol{r}_i \in \mathcal{R}$ with the mesh results in the correspondence for point $\boldsymbol{p}_i$, denoted as $\boldsymbol{q}_i$, and the normal of the intersected triangle is the corresponding normal $\boldsymbol{n}_i$. To compute the relative transformation between the scan and the mesh, we can now use different error metrics, such as the point-to-point metric investigated for KISS-ICP in Chapter 4 [191], point-to-plane [154], or plane-to-plane error [162].

The data association step can also result in wrong correspondences, where a given point from a surface is associated with an intersected point in the mesh

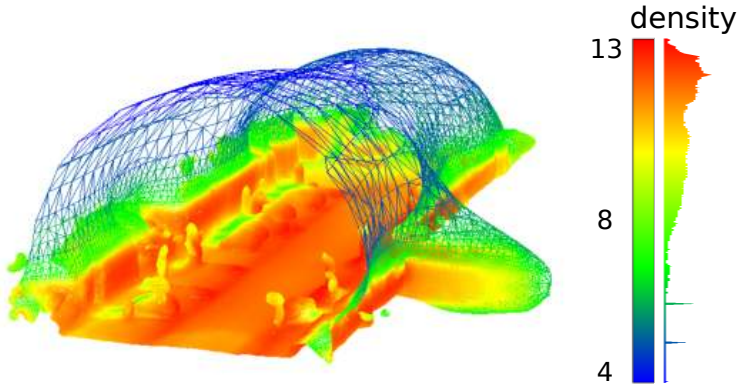


Figure 5.2: Cross-section view of the vertex densities for an urban scene reconstruction, where the colors represent the density of the vertices, as shown in the color bar. We trim away the lower 10% of the vertices based on the density threshold $\zeta(\boldsymbol{\nu})$.

from another surface. This typically happens when the ray does not hit any nearby surface and hits a faraway triangle. Therefore as outlier rejection, we remove correspondences $(\boldsymbol{p}_i, \boldsymbol{q}_i) \in \mathcal{C}$ from the set of correspondences $\mathcal{C}$ that satisfy $\|\boldsymbol{p}_i - \boldsymbol{q}_i\|_2 > \tau$. In our implementation, we use $\tau = 1\,\text{m}$. The reason for using such a fixed threshold is that this work was done before KISS-ICP [191]. In a new implementation, one would probably investigate an adaptive threshold.

Similar to Eq. (4.12), we obtain the incremental pose estimation by iteratively optimizing the following objective:

$$\Delta \mathsf{T}_{\text{est},j} = \underset{\mathsf{T}}{\operatorname{argmin}} \sum_{(\boldsymbol{s},\boldsymbol{q})\in\mathcal{C}} \rho\Big(\big\|(\mathsf{T}\boldsymbol{s} - \boldsymbol{q}).\boldsymbol{n})\big\|_2\Big), \tag{5.3}$$

but considering now a point-to-plane metric [154] between correspondent points. The choice of point-to-plane metric over point-to-point was determined empirically due to its best performance for this particular registration problem. To reduce the influence of outliers that were not filtered, we use a Huber weighting kernel $\rho(e)$ [78] instead of the previously used German-McClure in Sec. 4.1.6. As a result of this process, we obtain the pose of the vehicle $\mathsf{T}_t = \Delta\mathsf{T}_{\text{icp},t}\mathsf{T}_{t-1}$, where $\Delta\mathsf{T}_{\text{icp},t} = \prod_j \Delta\mathsf{T}_{\text{est},j}$, and j is the current iteration.

The main advantage of the proposed data association is that it does not need to compute nearest neighbors. Instead, the association step exploits the map representation, which, as we show later in the experiments, turns out to be faster, especially when dealing with high-resolution meshes. Nevertheless, this approach must handle large rotational motions properly and requires a reasonable initial estimate to converge, even though this also holds for nearest-neighbor methods.

5.1.4 Meshing Algorithm

A common technique for performing 3D surface reconstruction is to build an implicit function to recover the underlying surface of the input data [74]. Such implicit function f is usually defined as a scalar field in $\mathbb{R}^3$, i.e., $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, where typically the zero-level set of f represents the surface we aim at modeling. A popular technique in robotics and SLAM is to approximate f by the signed distance function [44], i.e., the projected distance from the sensor to the surface.

Our work, in contrast, explores the use of PSR [84, 85] to build high-quality, smooth, consistent maps for mobile robots, particularly autonomous vehicles. We refer to the original publications [84, 85] for details of the reconstruction algorithm. Our goal is to investigate the use of triangle meshes in SLAM in addition to the particular choice of the algorithm used for reconstruction[2].

Next, we explain mesh post-processing. The aforementioned Poisson reconstruction is designed to recover the closed surfaces of a single object in 3D, as illustrated in Fig. 5.2. Our 3D world, especially outside environments, is not composed of closed surfaces. Therefore, we need to refine the reconstructed surface and perform a post-processing step, which involves removing low-density vertices. The density $\zeta(\boldsymbol{\nu})$ of a vertex $\boldsymbol{\nu}$ on the mesh measures how many points from the input point cloud support the vertex $\boldsymbol{\nu}$. Intuitively, a low value means that the vertex is only supported by a low number of points and, therefore, is not densely measured in the original LiDAR scan or not measured at all (since the Poisson surface reconstruction algorithm will also extrapolate points where there is no data). After reconstructing the mesh, we compute the distribution of the per-vertex densities, as illustrated by the histogram in Fig. 5.2, right to the legend. The vertices of interest have a high density, i.e. those closer in space to point cloud data, and are colored yellow to red in the figure. We trim away the low-density vertices independently of the size of the mesh triangles. We make this decision only based on a vertex's density $\zeta(\boldsymbol{\nu})$. We consider the cumulative histogram of densities starting with the highest density values and trim those vertices that belong to the last 10%. This means that we remove 10% of the vertices with the smallest density values.

This post-processing produces a tighter reconstruction of the input data, showing little artifacts, allowing us to incrementally build the global mesh as described in Sec. 5.1.5. Without this step, creating a global map of the environment is impossible due to the artifacts shown in blue in Fig. 5.2.

Note that, as an interesting side effect, this density-based filtering also tends to eliminate most of the moving objects in the scene since 3D points on the

[2]We only consider horizontally placed LiDAR sensors. Therefore, we do not examine LiDARs shooting towards the sky, like profile scanners.

surface of moving objects often only support a small number of triangles as the position of the surface changes in every scan. This leads to low-density triangles on moving objects, and thus, no surface will be reconstructed at these locations.

5.1.5 Local and Global Map

In our approach, we distinguish between a local and a global map. The local map is built from the last N aggregated scans. The global map is only used for visualization and reporting the final output but is not used inside our approach, which will change when adding loop closure. A new mesh is reconstructed from the local map each time a new LiDAR frame has been registered to the local map. This produces a rolling grid-like mesh that moves with the estimated pose of the vehicle and stores enough information for the registration of new incoming scans. During the initial N scans, we turn off the mesh reconstruction module and rely on the standard point-to-plane ICP for estimating the vehicle's pose. After M scans have been registered, the last generated local mesh is integrated into the global mesh map. This means the global mesh will be updated only after M scans have arrived and registered (in contrast to the local mesh updated every time a new scan comes). To do so, we add all the triangles in the local mesh to the global one and then remove the duplicated triangles that can occur due to overlaps in the local map region. In our implementation, we use $N = M = 30$.

5.2 Experimental Evaluation

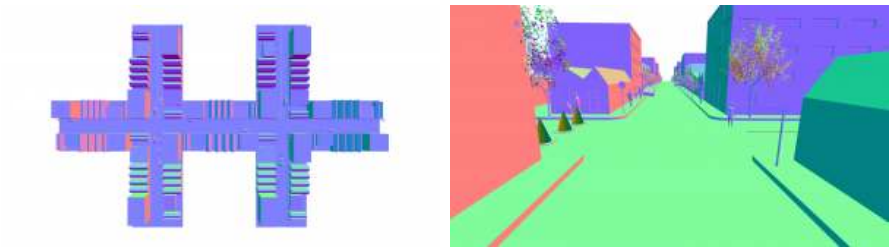
We implemented our approach on top of the Open3D library [221] and use the Embree library [193] for efficient ray-to-triangle intersection queries. Our algorithm was tested on an Intel Xeon W-2145 CPU with 8 cores and 32 GB RAM.

For all of our experiments, we used the default SuMa settings and a voxel size of 0.10 m for TSDF. For our approach, we set the depth of the octree used in the Poisson surface reconstruction [84] to $\Delta_{\text{tree}} = 10$.

5.2.1 Datasets

We use simulated and real-world data for our evaluation. To assess the accuracy of our map, we need ground truth data, and thus, we generate synthetic sensor data for the virtual urban environment shown in Fig. 5.3[3]. To obtain ground truth data, we sample points from the computer-aided design (CAD) model using a virtual LiDAR sensor model with the same field of view and properties as the one used to obtain the scans but using 320 beams instead of 64. This way, we

[3]The data is available at `http://www.ipb.uni-bonn.de/data/mai-city-dataset`.



(a) Birds eye view (b) First person view

Figure 5.3: The Mai City dataset CAD model utilized for evaluating the mapping results, where virtual sensors were placed on the model to obtain LiDAR scans through ray-casting techniques. This dataset was used to evaluate the mapping accuracy of different approaches by providing realistic ground truth LiDAR data for comparison.

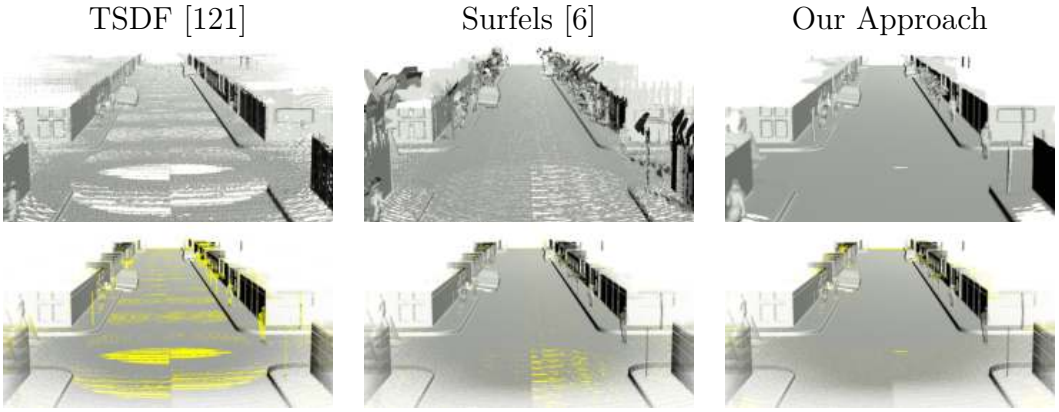


Figure 5.4: Qualitative examples showing the map accuracy. The first row exhibits the three map representations, TSDF, surfels, and our approach, respectively. The second row depicts the dense ground truth point clouds used to compute the metrics in Tab. 5.1. Along with the ground truth clouds, we highlight in yellow the points in the ground truth model whose distances to the closest point in the built models shown in the first row are greater than $\epsilon_d = 3\,\text{cm}$. Intuitively, the greater the number of yellow points, the more mistakes or gaps a model contains.

obtain a dense ground-truth point cloud but only from the parts of the scene that can be observed with the sensor, which is crucial to a fair comparison of the built 3D model. The final ground truth point cloud contains 62.5 million points and is shown in the second row of Fig. 5.4. Additionally, we use the odometry benchmark from the KITTI dataset [63] for real-world experiments.

5.2.2 Mapping Accuracy

The first set of experiments analyses the geometric accuracy of our map representation. We compare our mesh generation pipeline with two commonly used map representations: surfels [6] and TSDF [121]. To decouple errors in the resulting map caused by the representation itself and the pose estimation accuracy, we use

Method	Chamfer Distance $d_{\mathcal{CD}}(\mathcal{F},\mathcal{G})$	Precision $P(\epsilon_d)$	Recall $R(\epsilon_d)$	F-score $F(\epsilon_d)$
TSDF [121]	0.66	79.96	85.42	82.6
Surfels [6]	0.11	75.54	98.43	85.48
Ours	**0.05**	**93.28**	**98.69**	**95.91**

Table 5.1: Distance evaluation metrics for mapping accuracy. In all experiments $\epsilon_d = 0.03\,\text{m}$.

the same ground truth poses for all representations.

For a quantitative evaluation of the accuracy of the mapping systems, we densely sample the map representations with a density of 1,000 points per cm^2. Sampling the evaluated map representation allows a fair comparison between each approach and the ground truth model using standard point-cloud metrics [56, 91].

For our evaluation, we used the following metrics. Let $\mathcal{F}$ be the point cloud sampled from the map and let $\mathcal{G}$ be the ground truth point cloud. For a point $\boldsymbol{l} \in \mathcal{F}$, we define the distance to the ground truth model as:

$$d(\boldsymbol{l}, \mathcal{G}) = \min_{\boldsymbol{g} \in \mathcal{G}} \|\boldsymbol{l} - \boldsymbol{g}\|_2, \tag{5.4}$$

where $\|\cdot\|_2$ is the L2 norm. Analogously, we define the distance for a point $\boldsymbol{g} \in \mathcal{G}$ to the reconstructed map as $d(\boldsymbol{g}, \mathcal{F}) = \min_{\boldsymbol{l} \in \mathcal{F}} \|\boldsymbol{g} - \boldsymbol{l}\|_2$. For computing precision, recall, and f-score metrics, we follow exactly the work of Knapitsch et al. [91].

We also employ the Chamfer distance [56]:

$$d_{\mathcal{CD}}(\mathcal{F}, \mathcal{G}) = \frac{1}{2|\mathcal{F}|} \sum_{\boldsymbol{l} \in \mathcal{F}} d(\boldsymbol{l}, \mathcal{G})^2 + \frac{1}{2|\mathcal{G}|} \sum_{\boldsymbol{g} \in \mathcal{G}} d(\boldsymbol{g}, \mathcal{F})^2. \tag{5.5}$$

The results of the mapping accuracy are shown in Tab. 5.1. We are especially interested in the areas where the 3D models deviate from the ground truth. Therefore, we highlight all points that have an error of $\epsilon_d \geq 3\,\text{cm}$ in *yellow* (second row of Fig. 5.4).

We can see that the TSDF-based approach fails to reconstruct important distinctive objects of the scene, such as trees, pedestrians, or poles. It is important to note that this is a limitation of the method, as the framework used [121] does not drop any input scans. The surfel-based approach often provides more accurate reconstructions. However, it also adds spurious artifacts to the reconstructed scene. Our method outperforms both in terms of accuracy and correctness of the model since it does not miss important features and does not add artifacts at the same time.

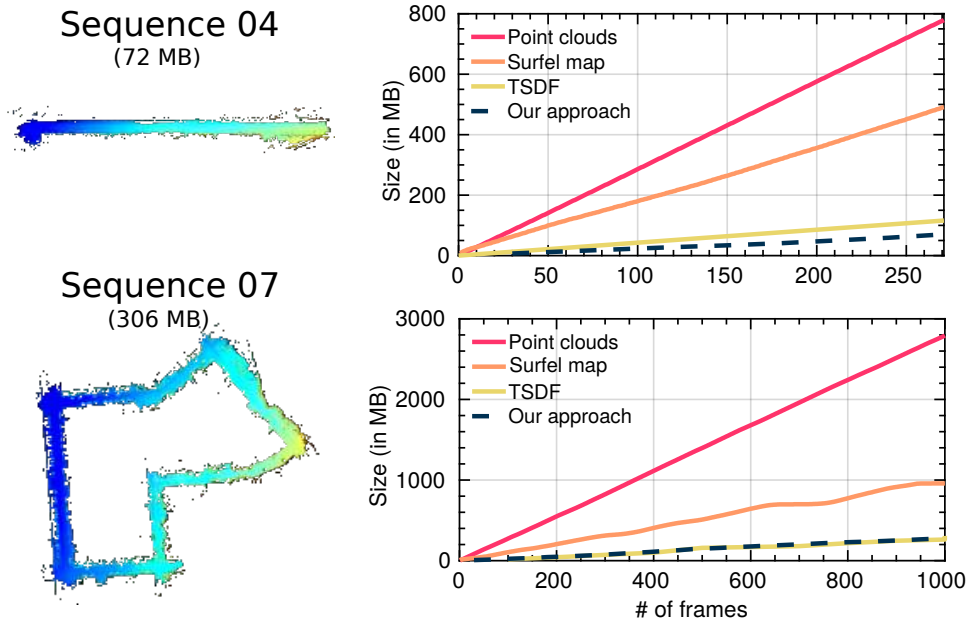


Figure 5.5: Memory consumption for different map representations for two KITTI sequences. On the left, we show the triangle mesh maps built by our approach and their final size. Sequence 04 is a country environment, and sequence 07 was recorded in an urban scenario. We see that point clouds and surfels result in higher memory usage. TSDF performs similarly to our approach regarding memory usage, but as seen in Fig. 5.4, our approach outperforms TSDFs in terms of geometrical accuracy.

5.2.3 Memory Efficiency

To use these high-definition maps in practice, it is necessary to be able to represent this map efficiently. The second evaluation investigates the memory requirements of different map representations. We show that our map representation, based on triangle meshes, is a good choice regarding a trade-off between geometrical accuracy and the corresponding memory footprint. We evaluated the amount of memory needed to represent a given map using our approach, TSDFs [121], surfels [6], and the LiDAR point clouds, only considering the raw geometric model, not any other additional information (such as normals, color information, etc.) that could be stored. The voxel size of $v_{\text{size}} = 0.1\,\text{m}$ for the TSDF-based [121] approach is chosen such that the resultant spatial resolution of the extracted mesh results is similar to the one built with our approach. SuMa [6] does not provide a mechanism to control its map size.

For a mesh-based map representation, the minimum amount of memory needed to represent a triangle mesh is given by $N_v\,\nu_{\text{size}} + N_f\,f_{\text{size}}$, where N_v and N_f are the numbers of vertices and faces in the model respectively and ν_{size} and f_{size} are the size of a vertex and a triangular face respectively.

In the case of a TSDF-based map representation, Whelan et al. [202] show that storing the TSDF volume as a map representation for large outdoor environments

is not directly feasible. In practice, we must extract the triangle mesh from the TSDF volume to store these maps. As a result, while comparing the memory footprint of the TSDF approach, we use the size of the extracted mesh instead of the TSDF volume. For a surfel, we need to represent the center position of the surfel, the normal, and the radius. Therefore, the size of a surfel map is given by $N_S\,(c_{\text{size}} + n_{\text{size}} + r_{\text{size}})$, where N_S is the number of surfels, c_{size} is the size of the center position, n_{size} is the size of the normal vector, and r_{size} is the radius size. For the case of a point cloud map, the size of the map is given by $N_P\,p_{\text{size}}$, where N_P is the number of points and p_{size} is the size of a point.

Fig. 5.5 shows the memory consumption of the different approaches over time for two different sequences of the KITTI odometry benchmark. Our approach scales well with the number of input scans. On the other hand, surfel-based maps or point cloud maps require much more memory, making running on mobile platforms challenging. However, TSDF-based maps exhibit similar memory consumption with a reduced mapping accuracy, as shown in Sec. 5.2.2.

5.2.4 Odometry and Localization Accuracy

In this experiment, we illustrate that our approach is well-suited for estimating the vehicle's pose by registering the LiDAR scans using a triangle mesh as a map. The primary objective of this experiment is to investigate how the suggested map representation can be utilized to determine the robot's pose without any other odometry source. However, KISS-ICP (Chapter 4) significantly surpasses the scan-to-mesh registration performance. Despite this, we retain the experiment for the sake of completeness.

We evaluate our performance on the KITTI dataset [63]. We compare our approach to commonly used registration algorithms based on ICP, point-to-plane ICP [154], and generalized-ICP [162]. For these approaches, we use the same optimization framework described in Sec. 5.1.3, including a Huber loss [78] function to reject outliers. As we do with our approach, we also initialize the ICP with the last increment estimate obtained at time $t-1$. Additionally, we consider a different scenario for these methods. Instead of frame-to-frame ICP, we perform frame-to-model ICP, where the *model* is the last N aggregated scans. This is represented as a *point cloud map* in Tab. 5.2. In this way, we can evaluate the benefits of running the reconstruction algorithm on this local map. We employ the nearest-neighbor (NN) search to perform data associations on all the variants. We also show the results from KISS-ICP as a reference. This method exploits a point cloud map with the past scans (Sec. 4.1.4). It is important to notice that KISS-ICP [191] was developed later than the work introduced in this chapter [188].

Lastly, we also consider a different data-association scenario, the projective

Map	Method	DA	avg.	00	01	02	03	04	05	06	07	08	09	10
None	point-to-plane ICP [154]	NN	7.60	9.12	10.00	6.19	6.04	4.51	7.69	8.24	5.53	8.70	9.37	8.18
			3.49	3.89	1.53	2.42	3.05	2.65	3.79	4.10	4.17	3.97	3.50	5.29
	GICP [162]	NN	14.35	7.35	73.10	14.40	9.37	13.60	5.23	2.23	6.40	7.27	10.90	8.00
			4.78	3.00	24.10	4.04	2.85	2.15	2.14	1.06	2.87	3.17	3.78	3.38
	SuMa [6]	Proj.	2.93	2.09	4.05	2.30	1.43	11.90	1.46	0.95	1.75	2.53	1.92	1.81
			0.92	0.93	1.22	0.79	0.75	1.06	0.79	0.64	1.17	0.96	0.78	0.97
Point cloud	point-to-plane ICP [154]	NN	18.92	9.99	77.10	11.70	2.31	70.00	2.62	1.84	1.79	3.67	17.40	9.70
			4.01	4.29	17.60	2.70	1.01	5.21	1.17	0.84	1.16	1.47	5.47	3.15
	GICP [162]	NN	20.43	4.34	93.10	10.70	2.21	83.70	1.56	1.42	1.19	2.33	21.80	2.37
			2.76	0.94	17.20	2.25	1.19	1.81	0.73	0.68	0.78	0.95	2.91	0.90
Mesh	Ours ($\Delta_{\text{tree}} = 10$)	NN	2.15	3.14	4.32	1.91	1.34	2.09	1.56	1.41	1.88	1.97	1.80	2.21
			1.14	1.49	1.04	0.73	1.07	1.46	1.07	0.72	1.36	1.10	0.82	1.67
	Ours ($\Delta_{\text{tree}} = 10$) (2021)	RC	**1.55**	**1.46**	**3.38**	**1.86**	**1.60**	**1.63**	**1.20**	**0.88**	**0.72**	**1.44**	**1.51**	**1.38**
			0.74	**0.68**	**1.00**	**0.72**	**1.10**	**0.92**	**0.61**	**0.42**	**0.55**	**0.61**	**0.66**	**0.84**
	KISS-ICP (Chapter 4) (2022/2023)	NN	**0.50**	**0.51**	**0.63**	**0.51**	**0.66**	**0.35**	**0.31**	**0.26**	**0.33**	**0.82**	**0.50**	**0.5**
			0.15	**0.18**	**0.15**	**0.15**	**0.16**	**0.15**	**0.14**	**0.08**	**0.17**	**0.18**	**0.13**	**0.19**

Table 5.2: Odometry estimation results for the KITTI Odometry benchmark [62]. The rows highlighted in gray correspond to the translational error, and the row below to the rotational error. All errors are averaged over trajectories of 100 to 800 m length. The translational error is in %, and the relative rotational error is in degrees per 100 m. DA for data association, RC for ray casting, and NN for nearest neighbor. The numbers in **bold** indicate the best approach for the given sequence, and the numbers in **blue** indicate the second best performing approach.

		Mesh vertex-sampling			Mesh ray-casting		
$\Delta_{\textbf{tree}}$	**Vertices**	t_{err}	r_{err}	runtime	t_{err}	r_{err}	runtime
8	30K	2.82	1.23	682	**1.73**	**0.90**	**395**
9	100K	2.05	1.08	645	**1.53**	**0.74**	**418**
10	300K	2.14	1.17	789	**1.56**	**0.74**	**535**

Table 5.3: Ray-casting vs mesh-sampling registration evaluation in the full KITTI [63] training sequences. Relative errors are averaged over trajectories of 100 to 800 m length. Relative translational error (t_{err}) is in % and relative rotational error (r_{err}) is in degrees per 100 m. The runtime values are expressed in *milliseconds*.

data association. For this, we compare our approach to *frame-to-frame* SuMA [6]. The TSDF [121] framework does not provide pose estimates and, therefore, is omitted in this experiment. It is important to note that to compare all methods, we use the exact normal computation for all the approaches described in Sec. 5.1.2.

In terms of estimation performance, we can see in Tab. 5.2 that our approach provides solid pose estimate performance when compared to the GNSS/IMU poses provided by KITTI. However, it is noteworthy that KISS-ICP performs better.

5.2.5 Registration

We briefly show that the novel registration scheme depicted in Sec. 5.1.3 is superior in terms of accuracy and speed. We compare our registration pipeline with a standard point-to-plane ICP [154], where the source in both cases is the incoming scan from the LiDAR sensor, and the target is set to be the triangle mesh built for our approach. We compare two different registration algorithms, one that samples all vertices and normals of the mesh to obtain a point cloud and the second that uses our novel data association algorithm based on ray-casting (RC) as explained in Sec. 5.1.3. While traditional ICP uses nearest-neighbor searches, we perform ray-casting through the mesh to obtain the target correspondences and the normal information. The registration is evaluated by running our registration algorithm in the full KITTI training sequences, registering all LiDAR scans to the local mesh-map (Sec. 5.1.5), using different map resolutions, i.e., by setting different resolutions of the Δ_{tree} on the Poisson surface reconstruction [85]. To investigate the accuracy of the registration method, we compute the average translational and rotational errors in the overall training sequences of the dataset. We see that the proposed registration algorithm scales better when the size of the input mesh increases. The results of this experiment are shown in Tab. 5.3.

5.2.6 Runtime

The pre-processing and the normal estimation take 45 ms on average per scan, and the scan-matching algorithm takes another extra 500 ms. However, the bottleneck is the meshing algorithm, which takes on average 5 s when executed on a CPU. This makes our approach infeasible for online operation on autonomous vehicles.

5.3 Conclusion

In this chapter, we presented a novel approach for creating 3D maps from LiDAR data offline. We represent the map as a triangle mesh estimated using Poisson surface reconstruction in a sliding window over past scans. The main contribution of this chapter is a novel LiDAR odometry and mapping system that builds surface reconstruction method accurate geometrical maps. Additionally, we release the source code of our approach as an open-source package, PUMA. Our approach can operate independently of other sources of odometry by performing a novel frame-to-mesh registration method. We obtain high-quality local meshes that show more details than standard alternative methods, such as state-of-the-art TSDF or surfel representations. We also show that our map representation is well-suited for incremental scan registration for pose estimation. Although the registration results of our approach are promising, KISS-ICP remains a better option for robot pose estimation. Our mapping system cannot operate online but can serve other robotics tasks, such as localization or place recognition. Furthermore, in the following chapter, we show how the high-detail level can help improve localization accuracy when using such triangle mesh maps generated by our approach. In this chapter, we have focused primarily on using triangle meshes in developing mapping pipelines and show that the reconstruction quality and the pose estimation accuracy are promising.

Chapter 6

Localization Using Mesh Maps

Robust and accurate map-based localization is crucial for autonomous mobile systems as it allows to localize the robot in previously built maps. In this chapter, we build on top of PUMA, the method previously introduced in Chapter 5, for generating a mesh map and tackling a localization problem. By exploiting such map representation, we can determine the precise location and orientation of autonomous vehicles within large-scale outdoor environments. Using range images generated from the current LiDAR scan and synthetic rendered views from the mesh, we propose a new observation model to be integrated into a Monte Carlo localization framework, achieving better localization performance and generalization to different environments. We test the proposed localization approach on multiple publicly available datasets collected in different environments with different LiDAR scanners. Additionally, we employ the IPB-Car perception platform described in Sec. 3.4.1 to record a new localization dataset. The dataset contains LiDAR sensor data in different seasons with multiple sequences, repeatedly navigating through the same crowded urban area in Bonn, Germany. The experimental results show that our method can reliably and accurately localize a mobile system in different environments and operate online at the LiDAR sensor frame rate to track the vehicle pose.

The content of this chapter is part of a joint work with Dr. Xieyuanli Chen, who was the lead researcher on this work. My contribution in this context regards the meshing algorithm to create the map presented in Sec. 6.1.2 and part of the experimental evaluation. The remaining parts belong to Dr. Xieyuanli Chen. For completeness, we present the overall contribution in the following sections.

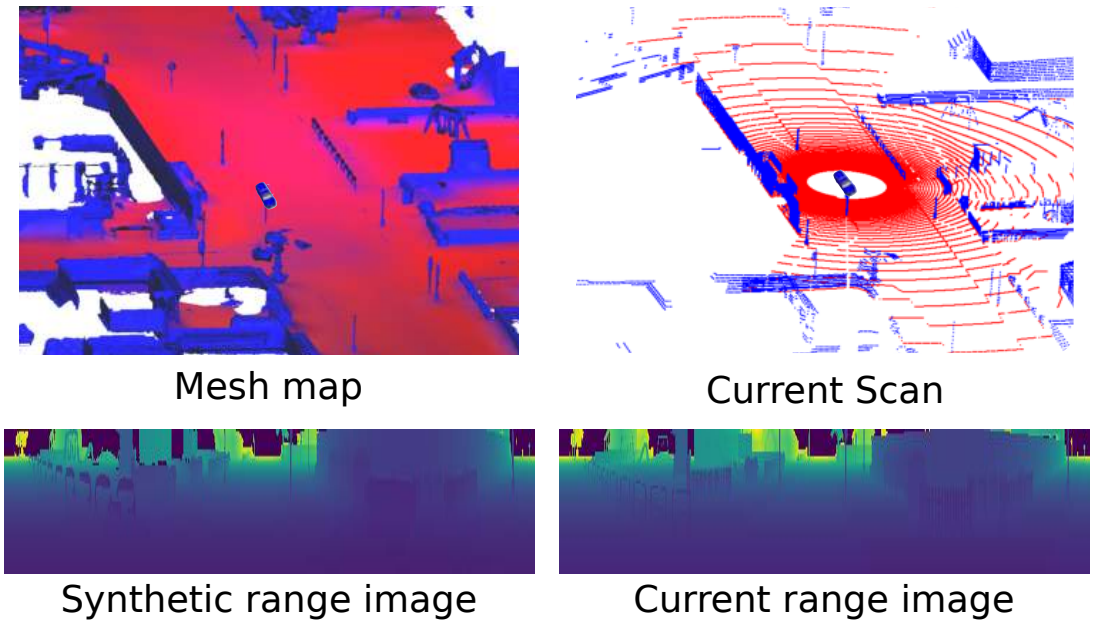


Figure 6.1: Visualization of range images, a triangle mesh map, and a single LiDAR scan. On the left, we show the mesh used as the map and the rendered synthetic range image from the mesh. In the mesh map, red parts correspond to ground planes, and blue parts represent non-ground structures. On the right, we show the LiDAR point cloud at the same location and the corresponding range image generated from the LiDAR scan.

6.1 Range Image-based LiDAR Localization

In this chapter, we tackle the problem of vehicle localization based on 3D LiDAR sensors. The main contribution of this chapter is a novel observation model for 3D LiDAR-based localization. Precise localization is a fundamental capability required by most autonomous mobile systems. With a localization system, a mobile robot or an autonomous car can estimate its pose on a map based on observations obtained with onboard sensors. Precise and reliable LiDAR-based global localization is needed for autonomous driving, especially in GPS-denied environments or situations where GPS cannot provide accurate localization results.

Our approach is based on range images generated from both the real LiDAR scans and synthetic renderings of the mesh map. We use the difference between them to formulate the observation model for a Monte Carlo localization (MCL) for updating the importance weights of the particles. Based on our novel observation model, our approach provides (x, y, θ)-pose estimates for the vehicle and achieves global localization using 3D LiDAR scans. Furthermore, our approach generalizes well to different environments collected with different LiDAR scanners.

Probabilistic state estimation techniques are used in most localization systems today. In particular, particle filters are versatile, as they do not need to restrict the motion or observation model to follow a specific distribution, such as a Gaus-

sian. When using particle filters, we need to design an appropriate observation model in line with the map representation. Frequently used observation models for LiDARs are the beam-end point model, also called the likelihood field [182], the ray-casting model [50], or models based on hand-crafted features [172, 218]. These methods either only work efficiently with 2D LiDAR scanners or require carefully designed features to work correctly. Recently, researchers have also focused on data-driven learning of such observation models [35, 79, 200], which provide accurate and reliable results as long as the environment is close to the environment used to learn the model.

Instead of using raw point clouds obtained from a 3D LiDAR sensor or features generated or learned from the point clouds, we investigate range images for 3D LiDAR-based localization for autonomous vehicles. As shown in Fig. 6.1, we project the point clouds into range images and localize the robot with rendered views from a map represented as a triangle mesh. There are several reasons for using these representations. The cylindrical range image is a natural and lightweight scan representation from a rotating 3D LiDAR. Additionally, as shown Sec. 5.2.3, a mesh map is a more compact representation than a large point cloud. Those properties enable our approach to achieve global localization in large-scale environments. Furthermore, rendering range images from a mesh map can be performed efficiently using computer graphics techniques. Therefore, range images and mesh maps are a perfect match for achieving LiDAR-based global localization.

In sum, we make three key claims: Our approach is able to (i) achieve global localization accurately and reliably using 3D LiDAR data, (ii) be used for different types of LiDAR sensors, and, (iii) generalize well over different environments. These claims are supported in our experimental evaluation. The implementation of our approach is open source[1].

In our work, we propose a probabilistic global localization system for autonomous vehicles using a 3D LiDAR sensor; see Fig. 6.2 for an illustration. To this end, we project the LiDAR point cloud into a range image (see Sec. 6.1.1) and compare it to synthetic range images rendered at each particle location from a map represented by a triangle mesh (see Sec. 6.1.2 and Sec. 6.1.3). Based on the range images, we propose a new observation model for LiDAR-based localization (see Sec. 6.1.5) and integrate it into a Monte Carlo localization system (see Sec. 6.1.4). Furthermore, we employ a tile map to accelerate the rendering and decide when the system converges (see Sec. 6.1.6). Using an OpenGL-based rendering pipeline, the proposed system operates online at the frame rate of the LiDAR sensor after convergence.

[1] `https://github.com/PRBonn/range-mcl`

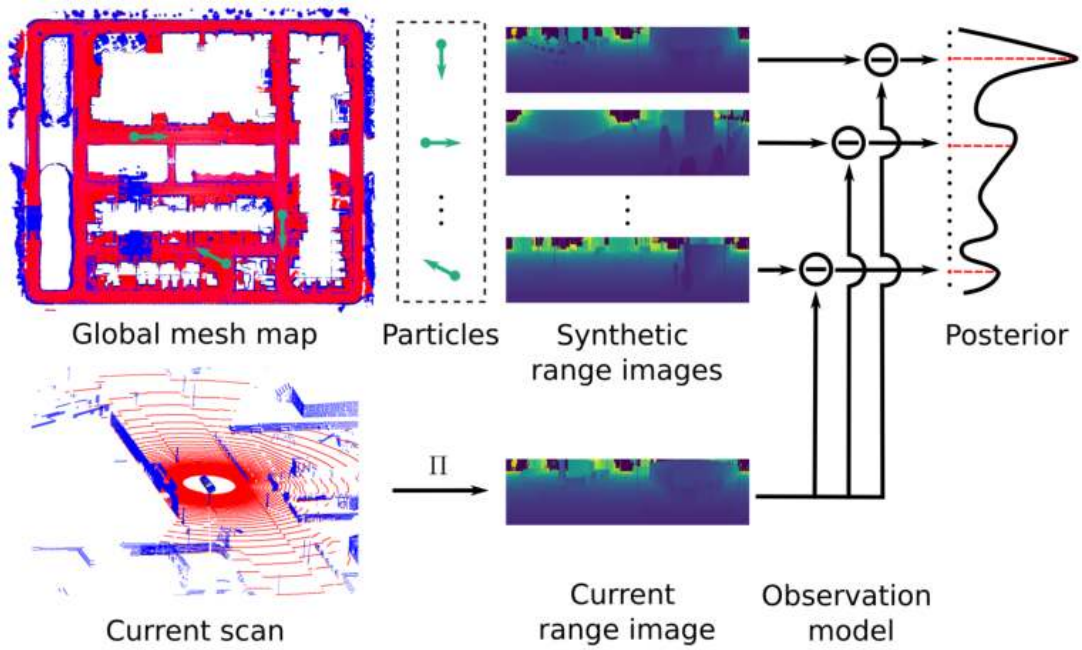


Figure 6.2: Overview of our approach. We project the LiDAR point cloud into a range image and compare it to synthetic range images rendered at each particle location from a mesh map. Based on the range images, we propose a new observation model for localization and integrate it into a Monte Carlo localization system to estimate the pose posterior of the vehicle.

6.1.1 Range Image Generation

The key idea of the proposed method is to use range images generated from LiDAR scans and rendered from the triangle mesh map for robot localization. To generate range images, we use a spherical projection [6, 34, 38, 120]. We project the point cloud $\mathcal{P}$ onto the so-called vertex map $\mathcal{V}_D : \mathbb{R}^2 \to \mathbb{R}^3$, where each pixel contains the nearest 3D point. Each point $\boldsymbol{p}_i = (x, y, z) \in \mathcal{P}$ is converted via the function $\Pi : \mathbb{R}^3 \to \mathbb{R}^2$ to spherical coordinates and finally to image coordinates $(u, v)^\top$, i.e.,

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\left[1 - \arctan(y, x)\pi^{-1}\right] w \\ \left[1 - \left(\arcsin(zr^{-1}) + \mathrm{f}_{\mathrm{up}}\right) \mathrm{f}^{-1}\right] h \end{pmatrix}, \tag{6.1}$$

where $r = \|\boldsymbol{p}_i\|_2$ is the range, $\mathrm{f} = \mathrm{f}_{\mathrm{up}} + \mathrm{f}_{\mathrm{down}}$ is the vertical field-of-view of the sensor, and w, h are the width and height of the resulting vertex map $\mathcal{V}_D$. Given the vertex map $\mathcal{V}_D$ and the range r of points at each coordinate (u, v), we generate the corresponding range image $\mathcal{R}_D$, on which the subsequent computations are based upon.

6.1.2 Mesh Map Representation

We represent the model of the environment as a triangle mesh $\mathcal{M}$. As shown in Sec. 5.2.3, a triangle mesh provides a memory-efficient representation that enables us to render the aforementioned range images at the frame rate of the LiDAR sensor.

To generate the map, we build on top of PUMA (Chapter 5) to obtain the map representation as a triangle mesh from the LiDAR scans. As the PSR algorithm used in PUMA requires an oriented point cloud, we use the method described in Sec. 5.1.2 to estimate the surface normals. Note that our approach does not need to use the same LiDAR sensor for map generation and localization.

The PSR algorithm provides a global solution that considers all input data at once without resorting to heuristic partitioning [24] or blending [1, 166]. Therefore, instead of building the map incrementally as done previously in Sec. 5.1.5, we aggregate all the oriented point clouds into a global reference frame using a system such as KISS-ICP (Chapter 4), or GNSS/IMU poses. This cloud map is directly fed to the PSR algorithm, yielding a globally consistent triangle mesh of the environment.

We use a ground segmentation algorithm to decrease the storage size of the map further. First, we compute the empirical covariance matrix Σ of all points in the cloud. We then compute the normalized Eigenvectors of $\Sigma : \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ with corresponding Eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$. We use a simple yet effective approach to label a point $\boldsymbol{p}_i = (x, y, z)$ and its corresponding normal $\boldsymbol{n}_i$ as ground point. A point $\boldsymbol{p}_i$ is considered to belong to the ground surface if it satisfies the following criteria: $\boldsymbol{n}_i \cdot \boldsymbol{e}_3 > \cos(\alpha_{\text{thres}})$ and $z < z_{\text{thres}}$. Note that this imposes the limitation of assuming a flat map.

After all ground points have been labeled, we run the meshing algorithm (Sec. 5.1.4) and retain the labels encoded as two different colors of the vertices in the map. We split the full mesh into the non-ground and ground mesh to simplify the final model. For the ground mesh, we contract all vertices to a single vertex inside a voxel of a given size v_{size}.

Then, we filter the ground vertices using an average filter by replacing each vertex $\boldsymbol{\nu}_i$ with $\boldsymbol{\nu}_i^*$ averaging all adjacent vertices $\hat{\boldsymbol{\nu}}_n \in \mathcal{N}$:

$$\boldsymbol{\nu}_i^* = \frac{\boldsymbol{\nu}_i + \sum_{n \in \mathcal{N}} \hat{\boldsymbol{\nu}}_n}{|\mathcal{N}| + 1} \,. \tag{6.2}$$

After averaging, invalid edges are removed. Once the ground surface has been simplified, it is combined with the rest of the mesh without further processing. This simplification allows us to decrease the size of the mesh model to about 50% of its original size.

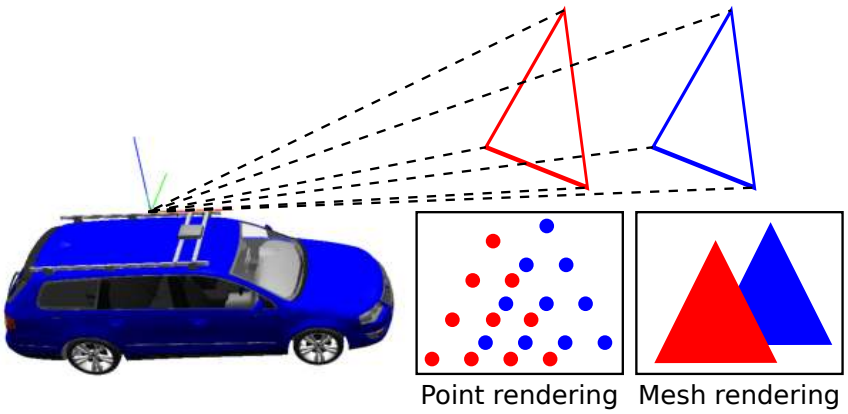


Figure 6.3: rendering example. In contrast to a map represented by a point cloud, triangle meshes are smoother and more compact. For each triangle, only three vertices need to be projected. Moreover, triangles can better represent the occlusion relationship between different objects.

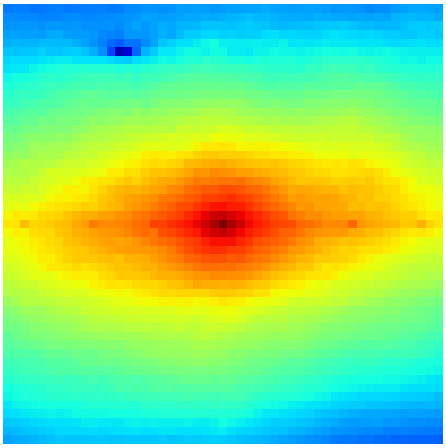
6.1.3 Rendering Synthetic Range Images

Given a particle j with its state vector $(x, y, \theta)^j$ and the triangle mesh map $\mathcal{M}$, we use OpenGL to render a synthetic range image for that particle. Using the spherical projection cf. Eq. (6.1), we project vertices of the triangles from the given particle pose and let OpenGL shade the triangle surface considering the occlusion, as shown in Fig. 6.3. To further accelerate rendering, we render batches of range images for multiple particles using instancing of the map. This prevents us from reading the vertex positions multiple times and minimizes the number of draw calls.

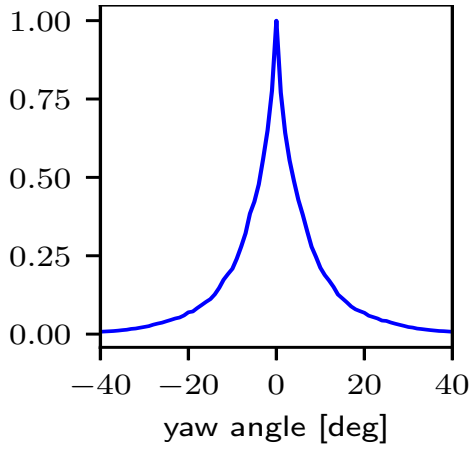
6.1.4 Monte Carlo Localization

MCL is a popular, particle filte-=based localization approach [50]. In our case, each particle represents a hypothesis for the autonomous vehicle's 2D pose $\boldsymbol{x}_t = (x, y, \theta)_t$ at time t. When the robot moves, the pose of each particle is updated based on a motion model with the control input $\boldsymbol{u}_t$. The expected observation from the predicted pose of each particle in the map $\mathcal{M}$ is then compared to the actual observation $\boldsymbol{z}_t$ acquired by the robot to update the particle's weight based on an observation model. The particles are sampled according to their weight distribution, and resampling is triggered whenever the effective number of particles drops below a specific threshold [69]. After several iterations of this procedure, the particles eventually converge around the true pose.

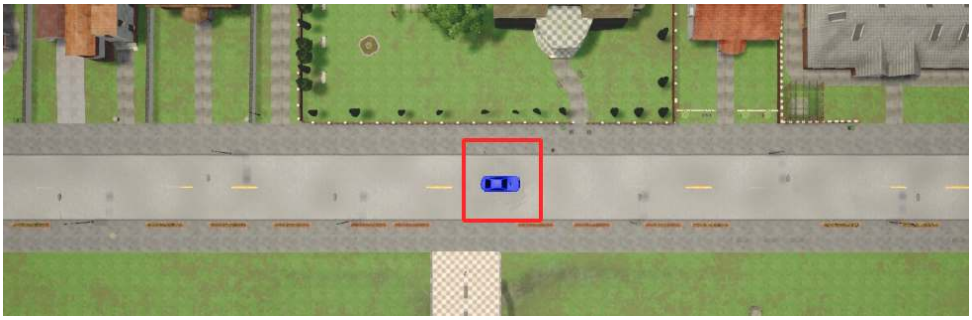
MCL realizes a recursive Bayesian filter estimating a probability density $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t})$ over the pose $\boldsymbol{x}_t$ given all observations $\boldsymbol{z}_{1:t}$ up to time t



(a) Location heatmap



(b) Heading likelihood



(c) Scene used to generate the heatmap

Figure 6.4: Range image-based observation model. (a) A local heatmap shows the location likelihood of the scan at the car's position with respect to the map with the same heading. Red shades correspond to higher weights. (b) Heading likelihood of the observation model when changing the yaw angle with the same location. (c) A top-down view of the Carla scene used in this example.

and motion controls $\boldsymbol{u}_{1:t}$ up to time t. This posterior is updated as follows:

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) = \eta \; p(\boldsymbol{z}_t \mid \boldsymbol{x}_t, \mathcal{M}) \cdot \int p(\boldsymbol{x}_t \mid \boldsymbol{u}_t, \boldsymbol{x}_{t-1}) \; p(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t-1}) \; d\boldsymbol{x}_{t-1}, \tag{6.3}$$

where η is a normalization constant, $p(\boldsymbol{x}_t \mid \boldsymbol{u}_t, \boldsymbol{x}_{t-1})$ is the motion model, $p(\boldsymbol{z}_t \mid \boldsymbol{x}_t, \mathcal{M})$ is the observation model, and $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t-1})$ is the probability distribution for the prior state $\boldsymbol{x}_{t-1}$.

Our work focuses on the observation model and employs a standard motion model for vehicles [182].

6.1.5 Range Image-based Observation Model

We design an observation model based on the generated range image from the current LiDAR scan and the rendered synthetic range images for all particles.

Each particle j represents a pose hypothesis $\boldsymbol{x}_t^j = (x, y, \theta)_t^j$ at time t. Given the corresponding synthetic rendered range image $\boldsymbol{z}^j$ for the j-th particle rendered at the particle's pose hypothesis, we compare it to the current range image $\boldsymbol{z}_t$ generated from the LiDAR point cloud. The likelihood $p\left(\boldsymbol{z}_t \mid \boldsymbol{x}_t, \mathcal{M}\right)$ of the j-th particle is then approximated using a Gaussian distribution:

$$p\left(\boldsymbol{z}_t \mid \boldsymbol{x}_t, \mathcal{M}\right) \propto \exp\left(-\frac{1}{2}\frac{d\left(\boldsymbol{z}_t, \boldsymbol{z}^j\right)^2}{\sigma_d^2}\right), \tag{6.4}$$

where d corresponds to the difference between or similarity of the range images $\boldsymbol{z}_t$ and $\boldsymbol{z}^j$ and σ_d is the standard deviation of the distribution.

There are several ways to calculate this similarity. For example, we could directly compare two range images at pixel level with absolute differences or use a cross-correlation. One could also generate features and compare two images in the feature space. Recently, many deep learning-based algorithms have also been proposed [35, 79, 178, 200].

Our goal is to investigate the use of range images generated from LiDAR scans and triangle meshes for a Monte Carlo localization system, and the particular choice of similarity computation is application-dependent. To keep the whole system fast and easy to use, we opted for a fast and effective method and used $d = N^{-1} \sum |\boldsymbol{z}_t - \boldsymbol{z}^j|$, i.e., the mean of the absolute pixel-wise differences, where N is the number of valid pixels in the current range image. Our results show that this choice is effective and generalizes well to datasets collected in different environments with different types of LiDAR sensors (see Sec. 6.2.5).

Fig. 6.4 shows the probabilities in a local area calculated by our proposed observation model and shows that it encodes the pose hypotheses very well. In contrast to previous work [34], which decouples the observation model into two parts, location likelihood and heading likelihood, the proposed observation model can estimate the likelihood for the whole state space $\boldsymbol{x}_t = (x, y, \theta)_t$ at once using one model, which is elegant and fast.

6.1.6 Tiled Map Representation

We divide the global mesh into smaller, regular-shaped tiles [205, 206, 207], to accelerate Monte Carlo localization by more efficient rendering. We only use parts of the mesh associated with a tile close to the particle position. In addition to more efficient rendering, we also use tiles to determine when the localization has converged. If all particles are localized in, at most, N_{conv} tiles, we assume that the localization has converged and reduced the number of particles to track the pose. Tiles also enable the runtime of our method to be independent of the size of the whole environment after converging.

6.2 Experimental Evaluation

We present our experiments to show the capabilities of our method and to support our claims that our approach is able to: (i) achieve global localization accurately and reliably using 3D LiDAR data, (ii) be used for different types of LiDAR scanners, and, (iii) generalize well to different environments without changing parameters.

6.2.1 Implementation Details

We implement our code based on Python and OpenGL. We use a modified version of PUMA, introduced in Chapter 5, to generate a triangle mesh map. For ground point extraction, we use $\alpha_{\text{thres}} = 30^\circ$ and z_{thres} as the sensor mounted height and employ a voxel grid with voxel size $v_{\text{size}} = 1.0\,m$. We use tiles of size $100 \times 100\,\text{m}^2$, and we reduce the number of particles from initially $10,000$ to 100 particles after convergence, i.e., only $N_{\text{conv}} = 1$ tile is covered by particles. The size of the tile map and the reduced number of particles are trade-offs between runtime and accuracy. We set $\sigma_d = 5$ in Eq. (6.4), and only update the weights of the particles when the car moves. All parameters are tuned with one dataset (IPB-Car) and kept the same for all other experiments with different datasets and sensors.

6.2.2 Datasets

We evaluate the generalization ability of our method using multiple datasets, including Carla [54], IPB-Car [35], MulRan (KAIST) [88] and Apollo (Columbia-Park) [222]. These datasets are collected in different environments with different types of LiDAR scanners at different times, see Tab. 6.1 for more details. For the Carla simulator, we added objects for the test sequences, which are not present in the map, to simulate a changing environment like in real datasets. For all experiments on different datasets, we only changed the intrinsic and extrinsic calibration parameters of the LiDAR sensors to generate the range images. We kept all other parameters, especially those of the MCL, the same.

6.2.3 Baselines

We use the same MCL framework in the following experiments and only change the observation models. We compare our method with three baseline observation models: the typical beam-end model [182], a histogram-based model derived from the work of Röhling et al. [149], and a deep learning-based model [35].

The beam-end observation model is often used for 2D LiDAR data. For 3D LiDAR scans, many more particles must be added to ensure that they converge

Dataset	Sensor	Sequence	Acquisition	Length
Carla (Simulator)	8 - 128 LiDARs	map (0.2 Gb)	n/a	3.5 km
		00	n/a	0.7 km
IPB-Car (Germany)	Ouster 64	map (0.8 Gb)	02/2020	6.2 km
		00	09/2019	1.7 km
		01	11/2019	1.9 km
MulRan (Korea)	Ouster 64	map (0.5 Gb)	08/2019	6.0 km
		00	06/2019	6.1 km
Apollo (U.S.)	Velodyne 64	map (5.4 Gb)	09/2018	44.8 km
		00	10/2018	8.8 km

Table 6.1: Dataset overview.

to the correct pose, which causes the computation time to increase substantially. We implement the beam-end model with a downsampled point cloud map using voxelization with a resolution of 10 cm.

Our second baseline for comparison is inspired by Röhling et al. [149], which exploits the use of similarity measures in histograms extracted from 3D LiDARs.

The third baseline is the overlap-based localization [35]. It uses a deep neural network to estimate the overlap and yaw angle offset between a query scan and map data and, on top of this, builds an observation model for MCL. The overlap-based method uses a grid map and stores virtual frames for each grid cell. We refer to the corresponding manuscript [35] for more details.

6.2.4 Localization Performance

The experiment presented in this section investigates the localization performance of our approach. It supports our claim that we achieve global localization accurately and reliably using 3D LiDAR data. For qualitative results, we show the trajectories of the localization results tested on the IPB-Car dataset in Fig. 6.5. The results illustrate that the proposed method localizes well in the map using only LiDAR data collected in dynamic environments at different times. The proposed method tracks the pose with more accuracy than the baseline methods.

For quantitative results, we first calculate the success rate for different methods with different numbers of particles, comparing our approach to the methods above, see Fig. 6.6. We tested the methods using five different numbers of particles $N = \{1000, 5000, 10000, 50000, 100000\}$. For each setup, we sample 10 trajectories and perform global localization. The x-axis represents the number of particles, while the y-axis is the success rate of different setups. The success rate

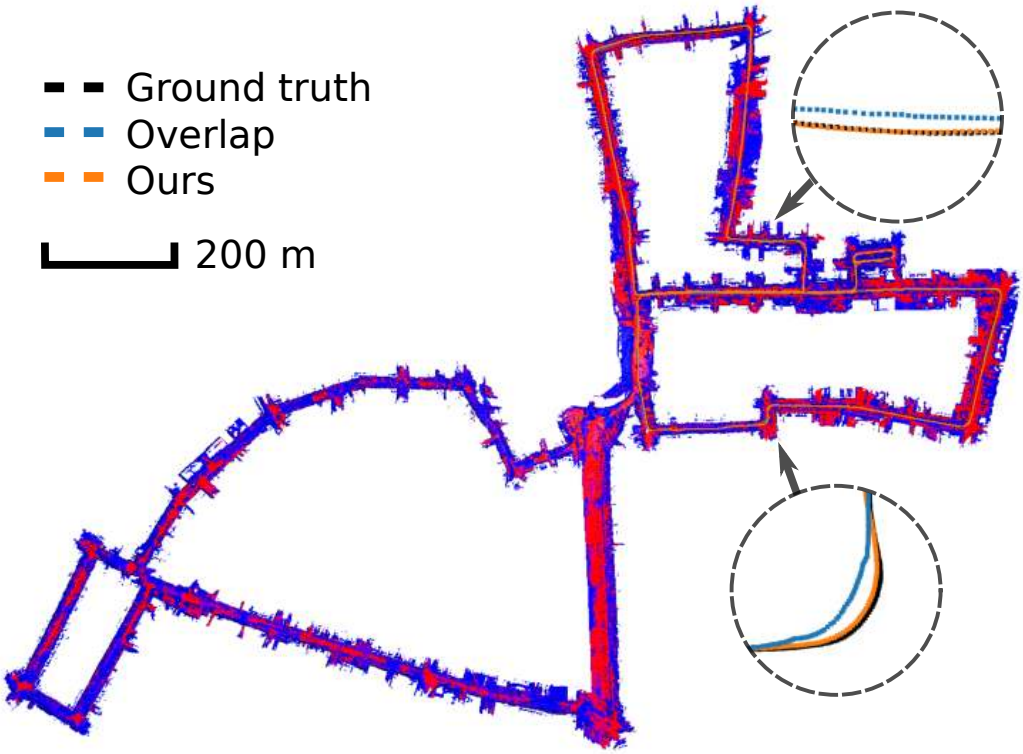


Figure 6.5: Localization results using 10,000 particles on the IPB-Car dataset. Shown are the mesh map, the ground truth trajectory (black), the overlap-based result (blue), and the result of our proposed method (orange).

for a specific setup of one method is calculated using the number of success cases divided by the total number of tests. To decide whether one test is successful, we check the location error by every 100 frames after convergence. If the location error is smaller than 5 m, we count this run as a success. The success rate of our approach is slightly lower than the Overlap-based method, but we require fewer particles for convergence. Compared to the histogram or beam-end models, our method outperforms them independently of the number of particles.

Quantitative results of localization accuracy are shown in Tab. 6.2. The upper part shows the location error of all methods tested with both sequences. The location error is defined as the root mean square error (RMSE) of each test in terms of (x, y) Euclidean error with respect to the ground truth poses. It shows the mean and standard deviation of the error for each observation model. Note that the location error is only calculated for success cases with 10,000 particles. The lower part shows the yaw angle error with respect to the ground truth poses. As before, the yaw angle error is only calculated for cases where the global localization converges with 10,000 particles.

The quantitative results show that our method outperforms all baseline methods in localization accuracy while achieving a similar heading accuracy. The reason is that our method uses online rendered range images and does not rely on

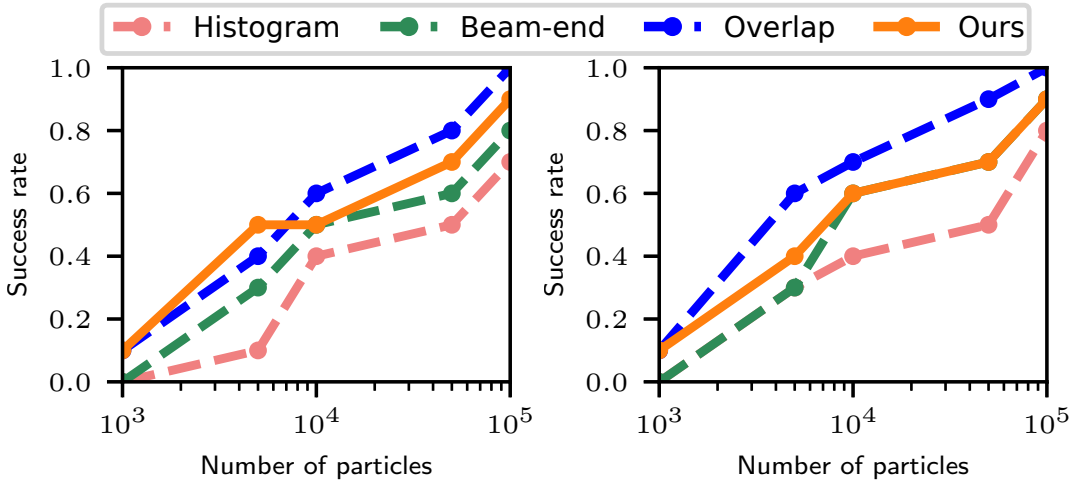


Figure 6.6: Success rate of the different observation models for 10 globalization runs. Here, we use sequence 00 (left) and sequence 01 (right) to localize in the map of the IPB-Car dataset.

	Location error [m]			
Sequence	**Beam-end**	**Histogram-based**	**Overlap-based**	**Ours**
00	0.92 ± 0.27	1.85 ± 0.34	0.81 ± 0.13	**0.66 ± 0.12**
01	0.67 ± 0.11	1.86 ± 0.34	0.88 ± 0.07	**0.44 ± 0.03**
	Yaw angle error [deg]			
Sequence	**Beam-end**	**Histogram-based**	**Overlap-based**	**Ours**
00	1.87 ± 0.47	3.10 ± 3.07	1.74 ± 0.11	**1.69 ± 0.11**
01	2.10 ± 0.59	3.11 ± 3.08	**1.88 ± 0.09**	2.53 ± 0.79

Table 6.2: Localization results on the IPB-Car dataset.

discrete grids. Therefore, our method will not be affected by the resolution of the grid. However, our method requires more particles to achieve the same success rate. Thus, we use a large number of particles for initialization. It will achieve a high success rate without influencing the runtime after convergence due to using tiles, see Sec. 6.2.6.

6.2.5 Generalization

This experiment supports the claim that our method can use different types of LiDAR to localize on the same mesh map. We tested 5 different sensors in the Carla simulator, including Quanergy MQ-8, Velodyne Puck, Velodyne HDL-32E, Velodyne HDL-64E, and Ouster OS1-128. We used the parameters of the LiDAR mentioned above, including the number of beams and the field of view. As shown in Tab. 6.3, our method works well with all types of sensors. It achieves good

Dataset	Scanner	Location RMSE [m]	Yaw angle RMSE [deg]
Carla	8-beams	0.48	3.87
	16-beams	0.43	3.87
	32-beams	0.42	3.40
	64-beams	0.36	3.46
	128-beams	0.33	3.31
MulRan	Velodyne 64	0.83	3.14
Apollo	Ouster 64	0.57	3.40

Table 6.3: Localization results on datasets using different sensors.

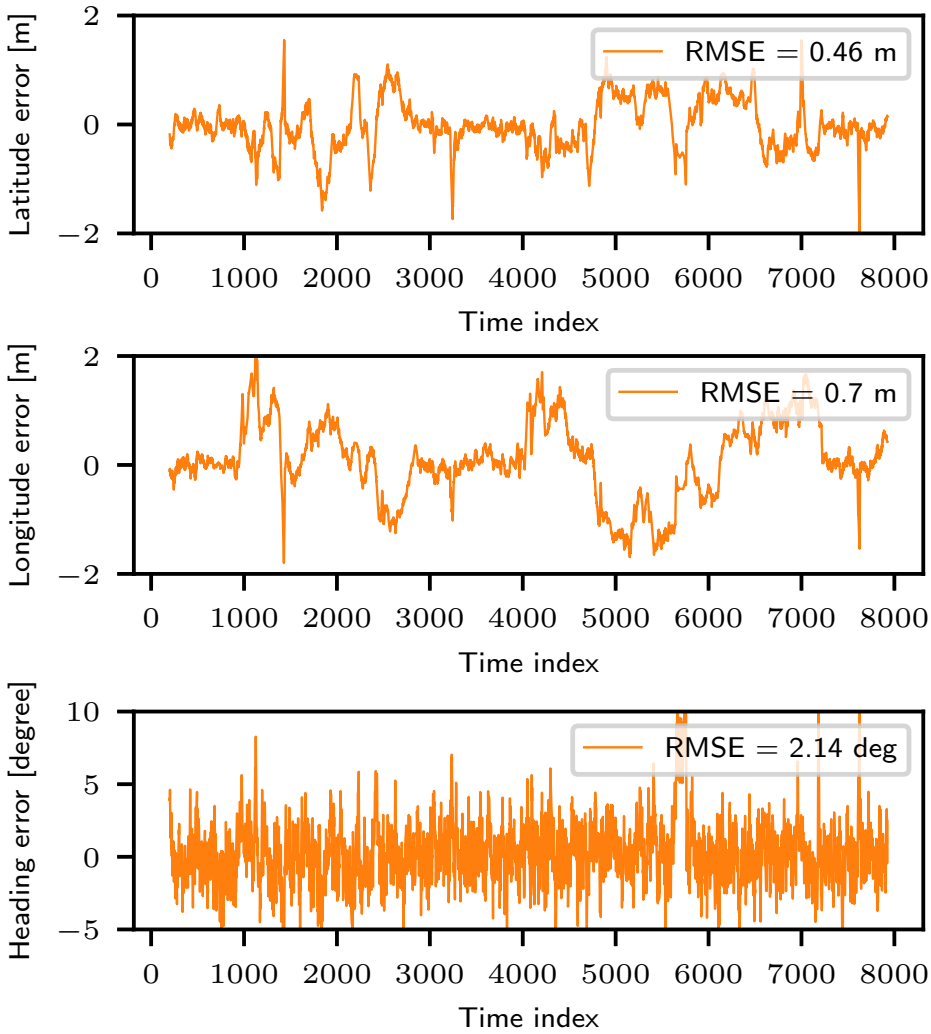


Figure 6.7: Localization results on the MulRan dataset w.r.t. provided GPS locations. The top figure shows the latitude error, the middle figure shows the longitude error, and the bottom figure shows the heading error.

localization results even with relatively sparse scans (location RMSE of 0.48 m with the 8-beam LiDAR).

Tab. 6.3 and Fig. 6.7 also verify the claim that our method generalizes well over different environments. We tested our method on the MulRan and Apollo datasets with the same parameters used in the Carla and IPB-Car datasets. Our method also works well in Korean and U.S. urban environments.

6.2.6 Runtime

Here, we show that our approach runs fast enough to support online processing on the robot at the sensor frame rate. We tested our method on a regular computer with an Intel i7-8700 with 3.2 GHz and an Nvidia GeForce GTX 1080 Ti with 11 GB of memory. After convergence, the average frame rate of our method is 21.8 Hz with 100 particles and the tile maps size of $100 \times 100\,\mathrm{m}^2$.

6.3 Conclusion

This chapter presented a novel range image-based online LiDAR localization approach. The main contribution of this chapter is a novel observation model for 3D LiDAR-based localization. Our method exploits range images generated from LiDAR scans and the offline mapping approach presented in the previous chapter. We show that exploiting a triangle mesh as the map representation allows us to localize autonomous systems in the given map successfully. We implemented and evaluated our approach on different datasets and provided comparisons with other existing techniques. Furthermore, we evaluate our localization approach on a self-collected dataset with the IPB-Car perception platform introduced in Chapter 3. The experiments suggest that our method can achieve global localization reliably and accurately. Moreover, our method generalizes well to different environments and can be used with different LiDAR sensors. After the convergence of the localization, our method runs online at the sensor frame rate. Additionally, we show that the offline mapping method presented is well-suited for solving the localization task and that the lack of real-time operation of the mapping system does not impact the localization performance.

Chapter 7

Online Mapping Using VDBs

Online mapping is a crucial task for robots deployed in the real world without prior knowledge about the environment. The requirements for online mapping differ from those for offline mapping, primarily because it must operate at a minimum of the sensor frame rate and must be able to provide the map built so far at any point in time. In this context, PUMA, introduced in Chapter 5 is unsuitable because it does not run at the sensor frame rate. This chapter showcases a practical approach to 3D online mapping based on TSDF. We revisit the basics of this mapping technique and offer a method for building an effective and efficient real-world mapping system. In contrast to most state-of-the-art mapping approaches, we make no assumptions about the size of the environment or the employed range sensor.

To build indoor 3D maps, dense data structures, like a regular voxel grid, are typically employed due to their easy implementation and use. However, it becomes highly impractical when dealing with large-scale outdoor environments. The reason is that the amount of data collected by modern sensors, like a 64-beam LiDAR, is extraordinarily voluminous. To illustrate, generating the TSDF representation of a single scan from such a sensor, with a voxel resolution of 10 cm, consumes 1 GB of memory. Thus, building 3D mapping systems on top of such data structures is impractical in real-world applications.

By employing sparse data structures, we can selectively store and process only those voxels that contain meaningful information. However, implementing such data structures is a challenging task. Our approach avoids reinventing the wheel by exploiting the OpenVDB library[1]. We adopt this library used mainly in the film-making industry and employ it for a robot mapping system: VDBFusion. Using a single-core CPU, our method can effectively manage large-scale outdoor environments and integrate point clouds from 3D LiDARs at 20 fps.

[1]VDB is just the name of the data structure and has no particular meaning: `https://www.openvdb.org/documentation/doxygen/faq.html#sMeaningOfVDB`

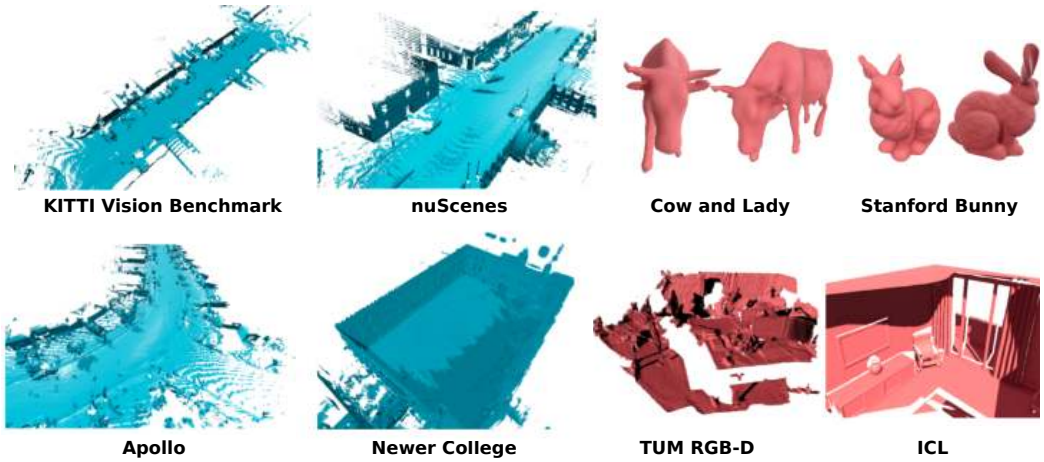


Figure 7.1: Results of our mapping approach on publicly available datasets showing the versatility of our proposed fusion pipeline. The models colored in light blue correspond to 3D LiDAR datasets, while the red ones correspond to RGB-D datasets.

7.1 3D Online Volumetric Mapping Using OpenVDB

Robots that are expected to navigate autonomously through real-world environments need maps to orient themselves and to plan trajectories [61, 66, 138, 143, 170]. These maps are typically built from sensor data, and thus, robotic systems typically rely on some form of mapping system. Nowadays, robots are equipped with various sensors, depending on the size of the environment, application, payload constraints, and budget available. Typically, 3D sensors are a part of such a sensor suite; popular examples are RGB-D cameras or LiDARs. Creating detailed 3D maps from such data sources can be challenging due to the size of a detailed world representation, especially when building high-resolution maps of large areas.

The main contribution of this chapter is an effective mapping system [190] that also comes as an open-source TSDF library[2] and does not require making assumptions about the size of the environment to be mapped. Our system has been tested on range sensor datasets using different 3D LiDARs and RGB-D cameras but can easily be adapted to other range-sensing modalities. The core algorithm of our system is based on the seminal work by Curless and Levoy [44], and our system is accessible to others through our library using a few lines of C++ code. Our implementation provides excellent results when employed in 3D mapping applications (see Fig. 7.1) while running two to three times faster than state-of-the-art implementations using only a CPU, consuming less memory, and

[2] `https://github.com/PRBonn/vdbfusion`

producing compressed map files. Moreover, our system is easy to use, which we showcase by conducting a user study.

Numerous data structures to realize effective map representations have been proposed [48, 66, 75, 126, 136, 156, 169, 173, 175, 186]. Most of these systems make assumptions about the specific sensor setup and do not provide systems that tackle the problem from a generic point of view. For example, commonly made assumptions about the sensor modality often render these systems unsuitable for other sensors. Furthermore, several TSDF fusion systems rely on hardware accelerators, such as graphics processing unit (GPU), that may not be available on mobile robots. In this chapter, we revisit the problem of creating a 3D map of the environment, trying to minimize the implicit and explicit assumptions about the sensing modality or the spatial bounds of the environment. We argue that working directly with point clouds, instead of raw sensor data such as RGB-D images or LiDAR range images, makes it possible to realize a mapping system that can handle different range sensors.

To this end, we base our system on top of the OpenVDB library [125]. OpenVDB is an open-source C++ library implementing a hierarchical data structure paired with a rich set of tools for the efficient storage and manipulation of sparse volumetric data. The library was originally developed by Museth and colleagues at DreamWorks Animation for rendering films. It offers unbounded volumetric space access, compact storage, and fast I/O operations. Building a robotic mapping system on top of OpenVDB enables us to provide a simple but effective and fast 3D volumetric fusion pipeline without reinventing the wheel.

We release a well-designed and carefully crafted C++ implementation with a rich and powerful Python API for rapid prototyping of mapping pipelines. The terminal command `pip install vdbfusion` is the only command needed to get started. We designed the Python API of VDBFusion to take numpy arrays as input and produce numpy arrays as output, making the library easy to plug into any existing robotics system without dealing with custom data structures. It also supports user-defined data loaders to parse already existing datasets as well as potential future data streams. We also provide a variety of usage examples in both programming languages, C++ and Python.

7.2 The VDB Data Structure

When dealing with 3D data such as point clouds in robotics, it is common to employ tree structures, such as octrees [7, 30, 61, 143, 187, 197, 216]. One of the key reasons behind using such structures is to have a virtually unbounded sparse representation of the scene that can be efficiently deployed on robotics systems where memory and CPU resources are constrained. Such data structures do not

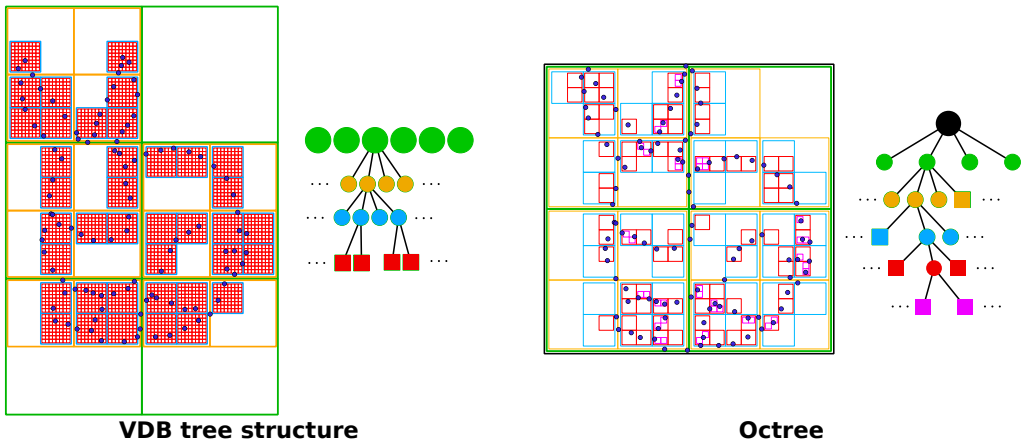


Figure 7.2: The VDB data structure [125] compared to octrees [114]. A conventional octree subdivides the space increasingly by a factor of 2 on each spatial dimension, starting at a single root node until it reaches leaf nodes (shown as squares) that contain a predefined number of points or size of the octant. In contrast, VDB has multiple root nodes (shown as green circles) and a fixed depth with leaf nodes comprising $8 \times 8 \times 8$ voxels. Due to the fixed depth, access in a VDB data structure is highly efficient compared to traversal in an octree.

require knowing in advance the size of the environment to be mapped.

In line with this, other domains have similar needs. For example, when computing fluid simulations, the volume of the simulation space is not known a priori, as the fluids can virtually expand infinitely. To provide an efficient solution for such applications, the VDB data structure was proposed in the computer graphics community targeting unbounded volumetric data manipulation in the context of creating animated movies.

The VDB representation is a sparse collection of blocks of voxels (typically $8 \times 8 \times 8 = 512$ voxels) that can be accessed through a hierarchical tree structure with two internal levels, i.e., a fixed height of the involved trees. This height-balanced construction of the VDB results in a shallow and wide representation compared to the deep octrees that only have a small branching factor of two on each spatial dimension, see Fig. 7.2 for an illustration. The fixed height of the involved trees allows the use of random access algorithms that operate in constant time. Additionally, the representation mimics modern CPU memory architectures with a fixed number of cache levels (L1, L2, etc.) of decreasing size and increasing random-access performance. We invite the curious reader to read more about VDBs and the difference to octrees in the original publication by Museth et al. [125].

The OpenVDB library [124, 125] is an open-source implementation of VDB and has been used in numerous movie production applications over the last decade. OpenVDB is supported by the Academic Software Foundation, which

OpenVDB	**Robotics**
Level Set	Signed Distance Field: The zero-level crossing represents the iso-surface.
Narrow-band	Truncation region close to the surface.
Half-width	Half width of the narrow-band or truncation distance (typically 3 voxels).
Background-value	Implicit value associated with empty voxels (typically the truncation distance).
Transformation	Representing the voxel size, or resolution, but extended to Affine transformation in general.
Differential Digital Analyzer	Similar to Bresenham's line algorithm applied to a 3D ray intersecting voxels.

Table 7.1: Terminology used in OpenVDB [125] and corresponding terms commonly used in robotics.

ensures the project's longevity. Moreover, the fact that the library is well maintained and accepts community contributions allowed us to make modifications to OpenVDB to make this work possible[3][4][5]. Another advantage of using OpenVDB is the large set of tools developed for the library that can be used out of the box. Examples are the OpenVDB visualizer employed to generate the grid plots shown in this chapter, out-of-core storage of grid values, or compression of grids.

To the best of our knowledge, OpenVDB is the only well-supported, open-source library dedicated to volumetric data applications. Although OpenVDB [125] has been open source for almost a decade, the robotics community paid little attention to it [12, 108] in relation to the benefits that the library provides. Potentially, this is rooted in the lack of common vocabulary between communities. For example, the term TSDF does not appear in the original publication [125] nor the word "truncated". To cope with the difference in terminology, we introduce a table of translations, see Tab. 7.1, between the common keywords found in VDB applications and robotics. We recommend the reader to inspect the original publication [125] with the table in hand or when inspecting our implementation. In sum, we invite the robotics community to take advantage of the suite of tools associated with VDB that highly match the needs of 3D robotic applications.

To build mapping applications using volumetric data structures, one could potentially adapt existing voxel-hashing systems [61, 128, 187]. Nevertheless,

[3] `https://github.com/AcademySoftwareFoundation/openvdb/pull/1048`
[4] `https://github.com/AcademySoftwareFoundation/openvdb/pull/1055`
[5] `https://github.com/AcademySoftwareFoundation/openvdb/pull/1105`

most existing methods focus on the implementation details of the voxel-hashing approach and require substantial time to develop a custom application. Additionally, the publications associated with such systems often focus on how the data structures have been implemented, not how to use them. When using our library that builds on top of OpenVDB, the application developer can safely ignore the underlying data structure implementation and use the structure as if it would be a dense voxel grid. The details are handled transparently under the hood.

7.3 The VDBFusion Library

In this section, we describe our system for volumetric mapping using truncated signed distance functions [44, 126] that exploit VDB via OpenVDB [125]. We describe our design decisions and provide details on the implementation and usage. We kept our system as a mapping library and decoupled it from a SLAM system to be flexible and easy to use in various applications.

7.3.1 System Overview

Our fusion library aims to realize TSDF-based mapping [44], allowing us to incrementally fuse incoming point clouds from an arbitrary 3D sensor into a map representation. We should not be required to know the size of the map a priori, and the representation should be memory-efficient, fast to access, and easy to use.

We designed our system to process 3D point clouds to achieve this goal. Instead of working with the raw sensor output or a specific sensor model, such as the pinhole model for depth cameras, we base our system on 3D point clouds. For every point, we use the location of that point, either in the global or a local coordinate frame, plus the pose of the sensor (position and rotation) when taking the measurement. This pose is available for every 3D point (or point clouds in case no motion distortion occurs). This approach allows us to build a sensor-agnostic mapping system.

The input to our system is a set of N points $\mathcal{P} = \{\boldsymbol{p}_1, \dots, \boldsymbol{p}_N\}$, where $\boldsymbol{p}_i \in \mathbb{R}^3$. We assume to have an estimate of the current pose of the sensor $\mathsf{T}_t \in SE(3)$ available from KISS-ICP (Chapter 4) or from a GPS/IMU combination. To simplify the description of this section, we assume here that all points $\boldsymbol{p}_i \in \mathcal{P}$ are expressed in the global coordinate frame. In our library, however, the user can provide the points in global coordinates or individual sensor viewpoints together with 3D points in the local sensor frame.

An overview of the high-level design of our system is shown in Fig. 7.3. The data loader module carries out any projection, pre-processing, or noise filtering

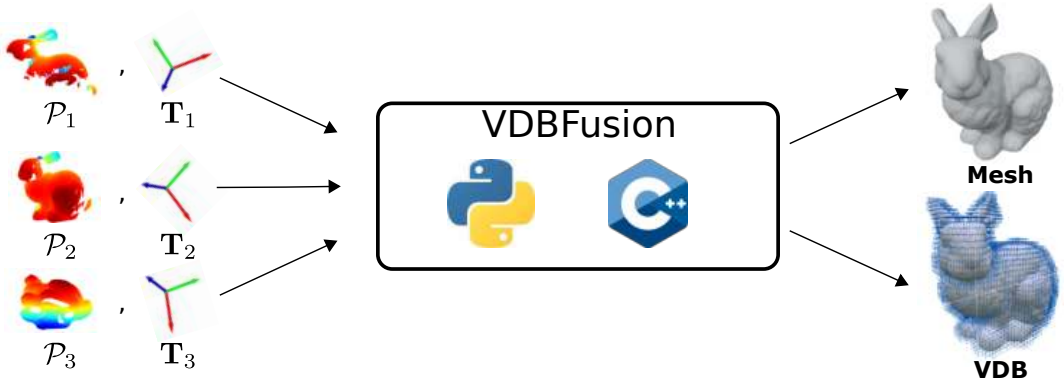


Figure 7.3: High-level overview of VDBFusion. Our system only takes as input point clouds $\mathcal{P}_i$ with their corresponding poses $\mathbf{T}_i$. Based on this information, VDBFusion integrates the sensor data into a sparse TSDF representation, which can be used to compute a triangle mesh representation and to access the underlying VDB representation.

as desired. Its implementation is not of relevance to our system. We require the data loader module to output the point clouds in the form of `numpy.ndarray` or `std::vector<Eigen::Vector3d>` for the Python and C++ APIs, respectively; analogously, the sensor origin $\boldsymbol{t}_i \in \mathbb{R}^3$ must also be a `numpy.ndarray` or a `Eigen::Vector3d`. This assumption is the key ingredient towards a more generic system. Given such point clouds and sensor locations, we can now integrate the data into the TSDF, exploiting the VDB data structure via the VDBFusion approach. After integrating the scans, we can extract a triangle mesh, individual TSDF values, or the underlying VDB data structure.

To the best of our knowledge, only Voxblox [130] offers a similar fusion pipeline. In contrast to Voxblox, we move all pre-processing to the sensor/dataset-dependent data loader, which performs all pre-processing steps such as minimum range filtering, maximum range filtering, motion undistortion, bilateral filtering, etc. We keep the sensor-data-specific operations in the data loader and outside VDBFusion. This allows us to realize a more elegant fusion pipeline and minimize the data-dependent parameters that depend on the employed sensor. As a comparison, our system has only three parameters, while Voxblox requires setting 14 parameters for the mapping algorithm.

7.3.2 Integration Pipeline Implementation

We follow the approach of Curless and Levoy [44] to integrate point clouds with known sensor location into the current internal map representation represented by a VDB volume.

To integrate a new measurement, $\mathcal{P}$, we must first compute the voxels to be updated in the global grid. To compute the voxel locations $\boldsymbol{x} \in \mathbb{Z}^3$ in the VDB

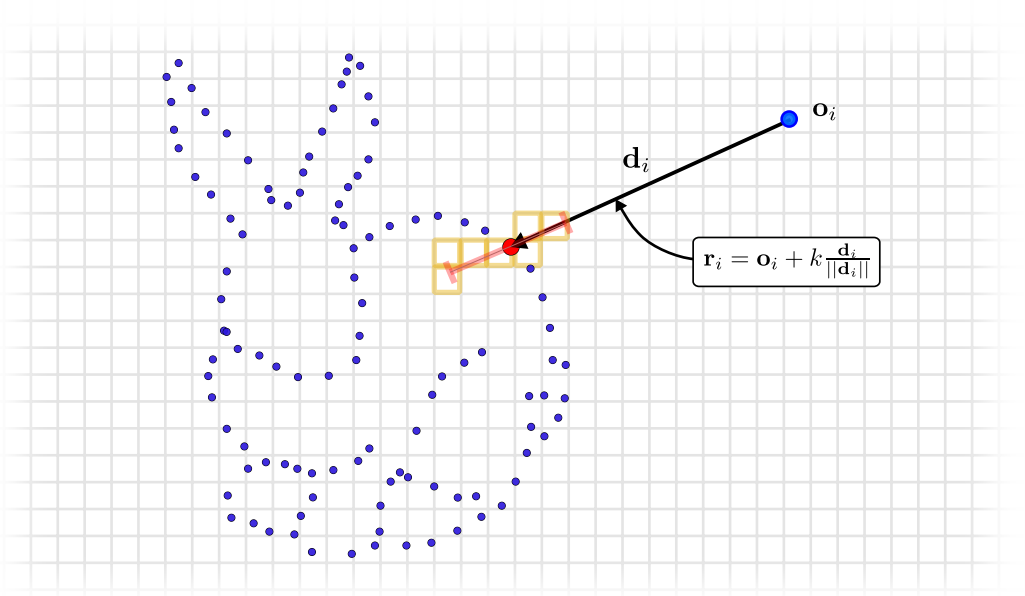


Figure 7.4: Integration of a measurement (red point) into the global grid representation by ray casting to determine which voxels are passed by the given ray. Only voxels (highlighted in orange) are updated inside the truncation distance (shown in red).

grid, we raycast a set of rays $\mathcal{R} = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N\}$. Each ray $\boldsymbol{r}_i$ is defined by:

$$\boldsymbol{r}_i(\gamma) = \boldsymbol{o}_i + \gamma \frac{\boldsymbol{d}_i}{\|\boldsymbol{d}_i\|_2}, \tag{7.1}$$

with the origins $\boldsymbol{o}_i$ and directions $\boldsymbol{d}_i = \boldsymbol{p}_i - \boldsymbol{o}_i$. All the rays are at the sensor origin in the global coordinate frame, i.e., $\boldsymbol{o}_i = \boldsymbol{t}_i$. In line with traditional methods [44, 126], we truncate the rays by considering only $\gamma \in \left[\|\boldsymbol{d}_i\|_2 - \psi_{\text{TD}}, \|\boldsymbol{d}_i\|_2 + \psi_{\text{TD}}\right]$ for fast integration, where ψ_{TD} is the truncation distance. This process is depicted in Fig. 7.4 and expressed in lines 19-21 of the fusion algorithm shown in Fig. 7.5.

We determine the voxel locations $\boldsymbol{x}$ by the ray–voxel intersections determined via differential digital analyzer (DDA) available in the OpenVDB library. Using the DDA allows us to significantly shrink the length of our integration code and keep it clean using a less error-prone implementation than our own handcrafted voxel traversal. The use of the DDA is shown in lines 24, 26, and 39 of the code snippet in Fig. 7.5. In Fig. 7.4, the voxel locations $\boldsymbol{x}$ that must be updated are highlighted in orange.

Once the voxels to be updated have been determined, we compute the projective signed distance, $d_{t-1}(\boldsymbol{x})$ (line 28 in Fig. 7.5), from the point to the center of each voxel. These signed distance values are then weighted with a weighting function $w_t(\boldsymbol{x})$, we implement this function in the form of a lambda expression passed at runtime (line 31 in Fig. 7.5). The weighted measurements are then integrated into two distinct VDB grids, $\boldsymbol{D}_t(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}$, which is a sparse volumetric scalar field representing the signed distances values for each voxel location

```cpp
void VDBVolume::Integrate(const std::vector<Eigen::Vector3d>& points,
                          const Eigen::Vector3d& origin,
                          const std::function<float(float)>& weighting_function) {
  // Get the D(x) and W(x) grid accessors(read and write)
  openvdb::FloatTree::Accessor tsdf_acc = tsdf_->getAccessor();
  openvdb::FloatTree::Accessor weights_acc = weights_->getAccessor();

  // Main loop, all the computation happens here
  std::for_each(points.cbegin(), points.cend(), [&](const auto& point) {
    // Get the range value measured by the sensor for the given point.
    const Eigen::Vector3d measurement = point - origin;
    const float range = static_cast<float>(dir.norm());

    // Create the origin of the ray and the normalized direction vector in openvdb types.
    const openvdb::Vec3R eye(origin.x(), origin.y(), origin.z());
    const openvdb::Vec3R dir(measurement.x(), measurement.y(), measurement.z()).normalize();

    // Truncate the Ray before and after the source, unless space_carving_ is set
    const float kmin = space_carving_ ? 0.0f : range - sdf_trunc_;
    const float kmax = range + sdf_trunc_;
    const auto ray = openvdb::math::Ray<float>(eye, dir, kmin, kmax).worldToIndex(*tsdf_);

    // Ray cast the ray through the sparse grid
    openvdb::math::DDA dda(ray);
    do {
      const openvdb::math::Coord voxel = dda.voxel();
      const Eigen::Vector3d voxel_center = GetVoxelCenter(voxel, voxel_size_);
      const float sdf = ComputeSDF(origin, point, voxel_center);
      if (sdf > -sdf_trunc_) {
        const float tsdf = std::min(sdf_trunc_, sdf);
        const float weight = weighting_function(sdf);
        const float last_weight = weights_acc.getValue(voxel);
        const float last_tsdf = tsdf_acc.getValue(voxel);
        const float new_weight = weight + last_weight;
        const float new_tsdf = (last_tsdf * last_weight + tsdf * weight) / (new_weight);
        tsdf_acc.setValue(voxel, new_tsdf);
        weights_acc.setValue(voxel, new_weight);
      }
    } while (dda.step());
  });
}
```

Figure 7.5: A complete fusion pipeline coded in C++ with a few lines of code.

$\boldsymbol{x}$, and the weight values $\boldsymbol{W}_t(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}^+$, also in the form of a sparse volumetric scalar field. The integration of these measurements is performed by following the equations introduced by Curless and Levoy [44] representing the TSDF (lines 25–39 in Fig. 7.5):

$$\boldsymbol{D}_t(\boldsymbol{x}) = \frac{\boldsymbol{W}_{t-1}(\boldsymbol{x}) \cdot \boldsymbol{D}_{t-1}(\boldsymbol{x}) + w_t(\boldsymbol{x}) \cdot d_t(\boldsymbol{x})}{\boldsymbol{W}_{t-1}(\boldsymbol{x}) + w_t(\boldsymbol{x})} \tag{7.2}$$

$$\boldsymbol{W}_t(\boldsymbol{x}) = \boldsymbol{W}_{t-1}(\boldsymbol{x}) + w_t(\boldsymbol{x}) \tag{7.3}$$

The zero set of the scalar field $\boldsymbol{D}_t^{-1}(\boldsymbol{0})$ represents the reconstructed surface. It can be computed from the TSDF representation $\boldsymbol{D}_t(\boldsymbol{x})$ employing techniques based on the popular marching cubes algorithm [101] or by raycasting the TSDF representation $\boldsymbol{D}_t(\boldsymbol{x})$ [126].

Reading and writing values in sparse data structures are usually the most expensive and hard-to-implement steps for these mapping pipelines. Levering the VDB data structure, we efficiently carry out read/write operations in our global map grid.

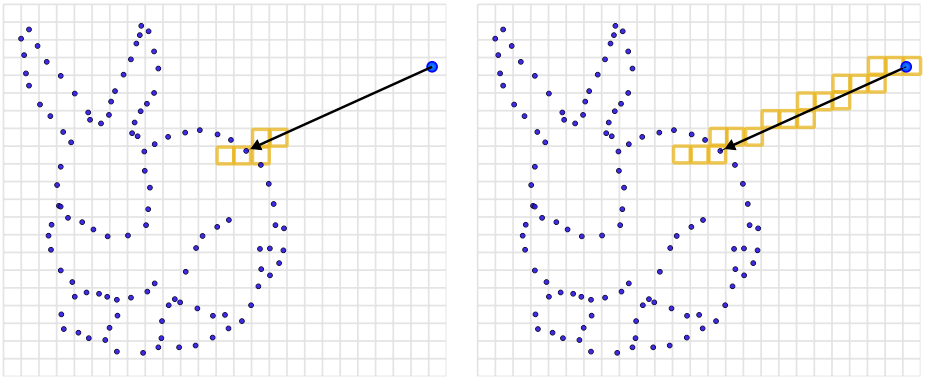


Figure 7.6: Difference between integration without space carving (left) and with space carving (right). Without space carving, only voxels inside the truncation region are updated. Space carving updates all voxels along the ray (colored in orange).

We highlight that even when possible, we decided to avoid low-level optimizations of the implementation to make our code more readable for the community. Nevertheless, our not necessarily optimized but clean implementation is on par or even faster and more memory efficient than state-of-the-art.

7.3.3 Space Carving

Some mapping applications need to be aware of the free space in the vicinity of the sensor or must distinguish free from unobserved regions. To enable such applications, we also provide space carving on demand. Our approach to space carving modifies the voxels along the ray $\boldsymbol{r}_i$ until the measured distance plus the truncation distance is reached, as shown in Fig. 7.6. We truncate it only after the surface by considering voxels in the truncation region, i.e., $\gamma \in [0, \|\boldsymbol{d}_i\|_2 + \psi_{\text{TD}}]$. This will mark all visited voxels as active in the VDB grid with a value that is the same as the truncated value (background value).

Although this might be relevant for mapping applications, it will highly impact the system's runtime performance. Some valuable conclusions and implementation details about the use of space carving and dynamic object removal are also explored in the experiments.

Note that we do not employ any probabilistic framework, such as occupancy probabilities used in Octomap [75]. We made this design choice with simplicity in mind, envisioning a clean implementation that allows other application developers to extend our system easily.

7.3.4 Weighting

The choice of the weighting functions $w_t(\boldsymbol{x})$ is not trivial and is extrinsically dependent on the sensor noise and the range of the measurement. Different

weighting functions have been extensively studied in the work of Bylow et al. [17]. Existing implementations have tried to cope with this by providing abstract functions modeling a fixed family of weighting functions [90] or by picking among a variety of hand-picked functions with configuration flags [130]. Virtual functions typically impact the application's runtime, and the weighting functions provided at compilation time [130] might not be required for a given application.

We revise this design choice and employ functional programming, letting the library user pick any arbitrary weighting function at runtime without recompiling the whole library. The user only needs to provide a lambda expression indicating how to compute the weighting function $w_t(\boldsymbol{x})$. This enables the faster operation of the integration method and more flexibility when designing a new mapping pipeline for a given sensor modality. For more details on how to use this functionality of our library, see Sec. 7.3.7.

7.3.5 Mapping Parameters

As motivated in Sec. 7.3.1, we only need a small set of three parameters for mapping since all the sensor-specific pre-processing is performed in the data loader. Therefore, we only have the following three parameters to parameterize the fusion pipeline:

- **Voxel size** (v_{size}) as a floating-point number: The voxels' side length determines the map's resolution. The bigger the voxel size, the smaller the map, which comes at the cost of losing high-level details. Analogously, the smaller the voxel size, the higher the level of detail that will be obtained for the final map at the cost of more memory usage and slower runtime operation.

- **Truncation distance** (ψ_{TD}) as a floating-point number represents the narrow band close to the surface we aim at modeling with the TSDF, i.e., the number of voxels to update in close to the surface. Large truncation distances allow for better smoothing of noise effects of the sensor but make the reconstruction of thin surfaces challenging due to thickening artifacts and lead to slower runtime operation because it requires more voxel visits. On the other hand, small truncation distances will lead to a faster runtime operation but are heavily impacted by the sensor's noise.

- **Space carving** as a boolean value indicates if space carving should be performed or not. Space carving can effectively remove dynamic objects from the map but comes at the cost of high computational time. In contrast, not performing space carving will lead to higher runtime speeds to the cost of having some dynamic objects artifacts on the map.

7.3.6 Meshing

To extract a triangle mesh from the grid map, we adapted the marching cubes [101] implementation from the Open3D [221] library to work with the VDB data structure. We also extended the implementation to allow filling holes, using the hole-filling algorithm described in the work by Curless et al. [44]. Moreover, we introduce an optional *min_weight* threshold to extract triangles from the grid only if this matches a given density. For the standard meshing algorithm typically employed with TSDFs, the user can set this threshold to 0. We empirically discovered that similar to what we have explored in the meshing algorithm introduced in Sec. 5.1.4, this simple modification enables an out-of-the-box dynamic object removal from the map. The library user can also select the value of the *min_weight* at runtime.

7.3.7 The Online Mapping Library

We implement our system entirely in C++ and provide a robust yet transparent set of Python bindings for efficient and easy usage. In the remainder of this section, we introduce code snippets that serve as starting points for writing a new mapping pipeline using our library. For a complete implementation, we invite the reader to check our MIT-licensed open-source implementation on GitHub[6].

Most existing approaches require a significant effort to get the mapping system up and running. In contrast, we aim to provide an easy-to-use interface. Moreover, we also facilitate an easy library installation using a simple `pip install vdbfusion` terminal command. The only Python dependency for the installation is numpy, making our Python package widely portable to different systems.

Below is a draft snippet on using our system for both C++ and Python API. We also provide a rich set of examples together with the source code of our library.

7.3.7.1 The C++ API

Our C++ API only consists of roughly 200 lines of code that enable a powerful yet efficient 3D mapping system. To get started with the library, the user only needs a dataset containing some form of 3D sensor or point cloud data. Although not required, we recommend creating a data loader module like the one shown in Fig. 7.7. Once the data are ready to be used for our system, a simple C++ or Python application can be written following the structure depicted in Fig. 7.8.

As described in Sec. 7.3.4, if the weighting strategy needs to be changed, one must pass a function to the `integrate` method, specifying how to compute the operation. As an example of how to achieve this, we demonstrate how to

[6] `https://github.com/PRBonn/vdbfusion`

realize the exponential weighting function introduced by the work of Bylow [17] in Fig. 7.9. We realize the code in the form of a lambda expression.

7.3.7.2 The Python API

Python has become the most popular programming language for prototyping nowadays. To leverage this popularity to expose our framework to a larger community, we provide a transparent and easy-to-use Python API, which reflects the same functionality as the C++ API. Our experiments show that choosing Python instead of C++ does not impact the performance. Thus, we are making the selection of Python or C++ a matter of preference without any drawbacks concerning the functionality or runtime performance of the mapping pipeline. As shown in previous snippets, both APIs are highly similar.

C++

```cpp
class Dataset {
  // Initialize your dataset here ..
  Dataset(...);

  /// Return length of the dataset
  std::size_t size() const { return n_scans_; }

  /// Returns a PointCloud(std::vector<Eigen::Vector3d>)
  /// and the sensor origin(Eigen::Vector3d) in the global
  /// coordinate frame.
  std::tuple<PointCloud, Point> operator[](int idx) const;
};
```

Python

```python
class Dataset:
    def __init__(self, *args, **kwargs):
        # Initialize your dataset here ..

    def __len__(self) -> int:
        return len(self.n_scans)

    def __getitem__(self, idx: int):
        # Returns a PointCloud(np.array(N, 3)) and sensor
        # origin(Eigen::Vector3d) in the global coordinate
        # frame.
        return points, origin
```

Figure 7.7: C++ and Python code snippets that implements the suggested 3D data loader code.

C++

```cpp
#include "vdbfusion/VDBVolume.h"

vdb_fusion::VDBVolume vdb_volume(voxel_size,
                                 sdf_trunc,
                                 space_carving);
const auto dataset = Dataset(...);

for (const auto& [scan, origin] : iterable(dataset)) {
  vdb_volume.Integrate(scan, origin);
}

// Extract triangle mesh (Eigen) from internal map.
auto [vert, triangles] = vdb_volume.ExtractTriangleMesh();
```

Python

```python
import vdbfusion

vdb_volume = vdbfusion.VDBVolume(voxel_size,
                                 sdf_trunc,
                                 space_carving
dataset = Dataset(...)

for scan, origin in dataset:
    vdb_volume.integrate(scan, origin)


# Extract triangle mesh (numpy arrays) from internal map.
vertices, triangles = vdb_volume.extract_triangle_mesh()
```

Figure 7.8: C++ and Python code snippets that implement a fusion pipeline including meshing.

C++

```cpp
const float sigma = 0.01;
const float eps = voxel_size;
vdb_volume.Integrate(
    scan, origin, [=](float sdf) -> float {
      if (sdf < eps) {
        return 1.0f;
      }
      return std::exp(-sigma * std::pow(sdf - eps, 2.0));
});
```

Python

```python
sigma = 0.01
eps = voxel_size
vdb_volume.integrate(
    scan,
    origin,
    lambda sdf: 1.0
    if sdf < eps else np.exp(-sigma * (sdf - eps) ** 2.0),
)

```

Figure 7.9: C++ and Python code snippets showing how to change the weighting function $w_t(x)$ for TSDF integration. In the example, we implement the exponential weighting function proposed by Bylow et al. [17].

Like the C++ case, the user only needs a 3D dataset in the form of `numpy` arrays when using Python. We also recommend (but do not require) defining a data loader similar to the one shown in Fig. 7.7.

7.4 Experiments

The experiments are designed to illustrate the abilities and flexibility of our approach. They showcase that our mapping pipeline is easy to use, flexible, fast, and memory- as well as disk-efficient. We further provide comparisons to existing open-source mapping systems. We only compare our system against existing methods that can process LiDAR and RGB-D data.

We run all the experiments on a CPU without multithreading. The motivation behind this choice is to analyze the system's performance besides the threading model implemented. We tested all methods on a GNU/Linux 64 bits system with GCC 9.3.0, the processor was an Intel Xeon W-2145 with 8 cores @3.70 GHz, and the system had 32 GB RAM.

To evaluate our system, we pick two datasets, one containing LiDAR data and one with RGB-D data. For the LiDAR dataset, we use the KITTI Odometry benchmark [63], and for the RGB-D, we choose the Cow and Lady dataset [130]. From the KITTI benchmark, we sample sequence 07, a small urban-like sequence with few dynamic objects. The reason behind this choice is that some baselines cannot map bigger sequences efficiently and thus make some experiments not executable.

For KITTI, we use a voxel size of $v_{\text{size}} = 10\,\text{cm}$ for all integration methods. The maximum usable LiDAR range is 70 m, and the minimum range is 2 m. For the experiments using RGB-D data, we process all points within the range of 0.1 m and 5 m and use a voxel size of $v_{\text{size}} = 2\,\text{mm}$. The truncation distance is three times the voxel size in all cases.

As baselines, we use the popular Voxblox [130] and Octomap [75] approaches. To the best of our knowledge, these are the only two systems with an implementation available that can effectively map both LiDAR and RGB-D data without modifying the implementation. Unless explicitly stated, all methods are evaluated using their C++ implementations. We compile both baseline methods from the source with all optimizations enabled. We do not make use of potentially provided ROS wrappers.

The main idea of this section is to put all three systems under test and benchmark runtime performance, memory usage, disk usage, and mapping accuracy. We run the experiments independently of the sensor modality of the dataset. Additionally, we conducted an experiment to show the ease of use of our system.

Dataset	w/o Space Carving		w/ Space Carving		
	Voxblox	**Ours**	**Octomap**	**Voxblox**	**Ours**
KITTI [63]	10.11 fps	**19.57 fps**	0.42 fps	0.60 fps	**1.37 fps**
Cow [130]	4.76 fps	**14.14 fps**	**1.05 fps**	0.42 fps	0.84 fps

Table 7.2: Runtime results for the Kitti Odometry dataset Sequence 07 [63] and the Cow and Lady Dataset [130]. All values are expressed in frames per second (higher is better). The best numbers are highlighted in bold font.

Implementation	KITTI 07	Cow and Lady
VDBFusion Python	18.93 fps	13.28 fps
VDBFusion C++	**19.57 fps**	**14.14 fps**

Table 7.3: Python vs. C++ runtime results for VDBFusion, expressed in average frames per second. The best numbers are highlighted in bold font.

7.4.1 Runtime

The first experiment is designed to evaluate the runtime performance. It is a common practice to evaluate this experiment iteratively, integrating synthetic scans and analyzing the statistics of the results. For this evaluation, we use a sequence from the KITTI Odometry Benchmark dataset. Sequence 07 is an urban driving sequence. We integrate all the scans on all the internal map representations and then average the runtime of the whole sequence to estimate the results.

We use the Google benchmark suite to compute the results. We explicitly stop the timers when data are being loaded or some data conversion is being performed. This provides a fair comparison and guarantees that all results reflect how long the systems take to integrate new scans into the internal map representation. We also distinguish between the mapping approaches, running with and without space carving.

The baseline methods evaluated in this experiment are Voxblox and Octomap. Since Octomap does not support an integration method without performing space carving, we skip this baseline when considering no space carving.

Depending on the mapping application, space carving may be required. As we show in this experiment, including space carving will heavily impact the runtime of the mapping system. A simple integration method with no space carving could also yield good results with the additional benefit of having a faster runtime, although not without losing the empty vs. unseen space information in the map.

As seen in Tab. 7.2, our system can integrate LiDAR data at roughly two times the standard sensor frame rate, making it a ready-to-use system in real applications. Additionally, we are 2–3 times faster than the baselines. While

Dataset	PCD	DVG	Octomap	Voxblox	Ours
KITTI [63]	2.95 GB	30.6 GB	1.12 GB	n/a	**847.0 MB**
Cow [130]	8.57 GB	363.5 MB	124.5 MB	n/a	**122.9 MB**

Table 7.4: Memory consumption of different investigated variants (lower is better). The best numbers are highlighted in bold font. PCD stands for point cloud maps, and DVG for Dense Voxel Grid.

Octomap can effectively cope with dynamic objects in the scene, its runtime makes it challenging to deploy on real-world applications requiring high sensor frame rate integration.

While Python is commonly thought of as a slow language, for our implementation, we spent extra effort to make the interoperability between C++ and Python as transparent and efficient as possible. The motivation behind this choice is that nowadays, Python is a common choice for prototyping systems and gives a fast entry point to the library. To assess the runtime performance of our Python API, we use the same datasets as before. As seen in Tab. 7.3, our Python API is on par with the C++ implementation regarding integration speed. The other baselines do not provide official Python implementations, so we skipped Octomap and Voxblox for this experiment.

7.4.2 Memory Efficiency

The second experiment analyzes memory usage during mapping. It illustrates that our system does not consume excessive memory, even for mapping large scenes. Note that the VDB data structure also provides other out-of-the-box possibilities to cope with even larger environments than the ones we study in this chapter, including out-of-core value storage, where the grid's topology is stored in RAM. However, the values can be offloaded to a hard drive. We do not consider such memory optimizations in this experiment since none of the baselines have similar capabilities. Voxblox does not provide any means to compute the memory usage of the internal map representation, and therefore, we skip this baseline for evaluation. To carry on this experiment, we proceed as in Sec. 7.4.1 and process the entire sequence 07 of the KITTI Dataset and the Cow and Lady dataset. We also provide the in-RAM consumption of more naïve mapping approaches, namely, point clouds and dense voxel grids. We do not conduct any particular experiment to obtain these values but compute the memory usage since it is deterministic. For point clouds, each point consumes three times the size of a floating-point value, and for dense voxel grids, we compute the bounding box of the resulting map and then fit a regular dense voxel grid.

As shown in the Tab. 7.4, for the case of the LiDAR sensor modality, the usage of point clouds or dense voxel grids as the map representation is virtually impossible. In contrast, Octomap requires less memory to represent the map, but VDBFusion shows an overall smaller memory footprint. When using RGB-D data, the difference between the memory footprint of the dense voxel grids, Octomap, and VDBFusion is less pronounced.

7.4.3 Disk Usage

One aspect of a mapping pipeline is its capability to store the map efficiently, and in this experiment, we evaluate the disk footprint of VDBFusion compared to other options. We compare the size of a point cloud map, which is still a popular choice despite its file size requirements. For this, we aggregate all the point clouds into a global coordinate frame and export the result to a binary file format using the Open3D library. Storing the raw point clouds is the upper bound in disk space consumption.

The VDB data structure also supports lossless compression out-of-the-box. This allows for an efficient reduction in the size on the disk, which is especially attractive for very large scenes. Likewise, Octomap also supports an optimized serialization protocol, which we use here for comparison. Contrarily, Voxblox does not provide any serialization mechanism; therefore, we use the triangle mesh provided by Voxblox as the map representation on disk. We also report the mesh size that can be extracted from VDBFusion to better compare with Voxblox.

Tab. 7.5 shows the size on the disk of the different options above when the resulting maps are serialized and stored. Here, storing the raw point clouds is not a viable option. To store its representation, Octomap discards the per-node probabilities and keeps only the maximum likelihood estimate of the map. In the resulting file, each node occupies only 16 bits of memory. Although this is a highly compressed map, the reconstruction details, as shown in Fig. 7.10, are not on par with our results. Furthermore, Octomap cannot integrate new measurements into an existing map once this has been serialized to the disk. The VDB file size is less efficient to store compared to Octomap because it requires a floating-point value for each voxel in $\boldsymbol{D}(\boldsymbol{x})$ and one floating-point value for each voxel in the weight grid $\boldsymbol{W}(\boldsymbol{x})$. However, the VDB representation on disk enables us to update the map even after storing it on the disk.

Compared to Voxblox, the mesh representation of VDBFusion is much smaller as it contains fewer artifacts, as shown in Fig. 7.11. Due to the additional artifacts produced by Voxblox, the serialized mesh size tends to be higher than the triangle mesh extracted from VDBFusion. Analogously to the Octomap case, the mesh representation can not be updated after the map has been stored on disk.

7.4.4 Mapping Accuracy

In this experiment, we evaluate the mapping accuracy of Voxblox, Octomap, and our mapping pipeline. To this end, we densely sample the maps generated by Voxblox and VDBFusion into a point cloud. Octomap provides an out-of-the-box method for converting the map representation to point clouds, so we use this

Dataset	PCD	Octomap 0/1 Output	Voxblox Mesh Export	Ours Mesh Export	Ours TSDF Volume
KITTI 07	3.0 GB	17.0 MB	672.0 MB	254.0 MB	170.0 MB
Cow	8.6 GB	1.6 MB	208.0 MB	15.0 MB	47.0 MB

Table 7.5: Disk usage of the serialized representation of different approaches (lower is better).

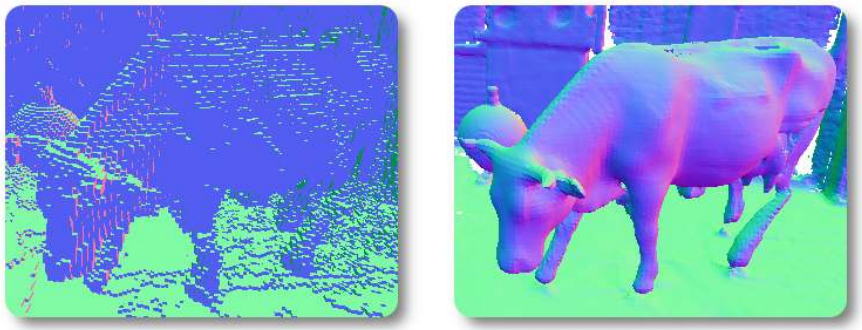


Figure 7.10: Qualitative comparison between Octomap and VDBFusion. While Octomap clearly models the scene, it does not achieve a high level of detail. Our fusion pipeline shows a higher level of detail on the surface when compared with Octomap.

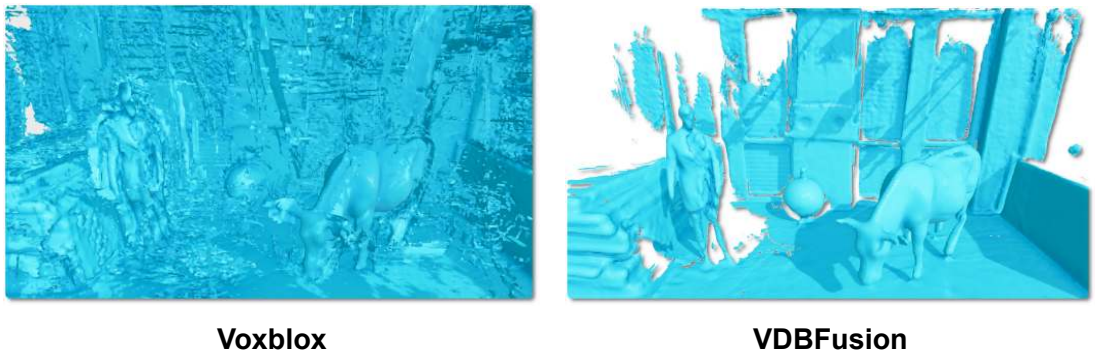


Figure 7.11: Qualitative comparison between Voxblox and VDBFusion, where we compute a triangle mesh from the TSDF volumes [101]. While Voxblox shows many artifacts in the background, our fusion pipeline shows a much cleaner surface reconstruction, recovering fine details of the scanned scene.

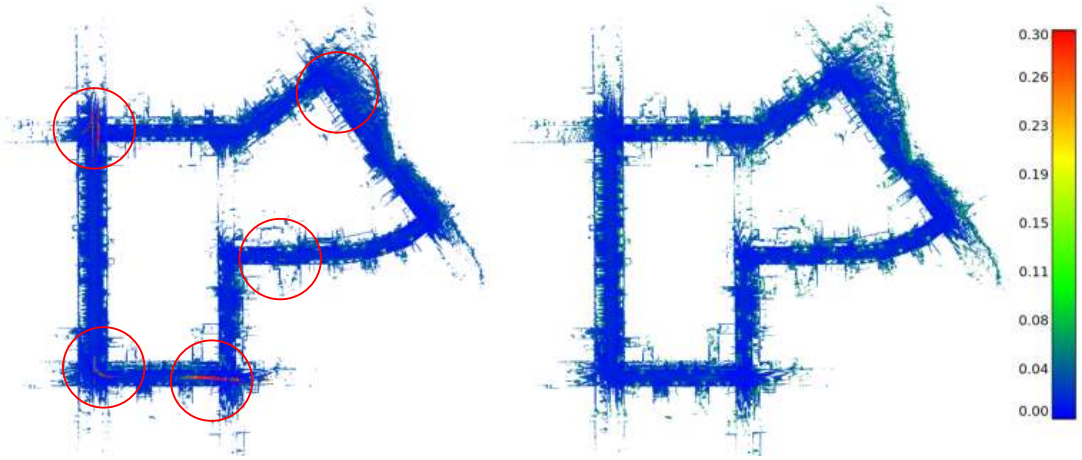
	Without Space Carving		
	Voxblox		**Ours**
KITTI 07	failed		**0.031 ± 0.102 m**
Cow	0.236 ± 0.298 m		**0.049 ± 0.065 m**
	With Space Carving		
	Octomap	**Voxblox**	**Ours**
KITTI 07	0.033 ± 0.035 m	0.497 ± 1.991 m	**0.023 ± 0.022 m**
Cow	0.195 ± 0.262 m	0.319 ± 0.398 m	**0.045 ± 0.062 m**

Table 7.6: Mapping accuracy comparisons. We report the mean and standard deviation of the point-to-point distance (in meters) between the estimated and ground truth map (lower is better). The best numbers are highlighted in bold font.

one instead for the evaluation. This sampled point cloud is then compared to the reference point cloud. For the case of the KITTI dataset [63], we aggregate all the points in the sequence without downsampling, and we further remove all dynamic objects by using manual annotations from the SemanticKITTI dataset [5]. The Cow and Lady reference point cloud was obtained with a high-resolution scanner and is provided along with the original dataset [130]. We use a uniform sampling strategy to sample the point clouds from the mapping baselines. In the case of the KITTI, we sample 100.000.000 points, and we sample 1.000.000 points for the Cow and Lady dataset. The motivation behind the choice of the number of points to be sampled is to match the density of the reference point clouds used for the evaluation. To assess the performance of the obtained map, we compute the point-to-point distance in meters between the reference point cloud and the sampled models under evaluation. We report the mean average distance between reference-model clouds and standard deviation. A lower metric corresponds to a model that closely models the reference point cloud, while a large point–point distance indicates that the map deviates from the reference point cloud.

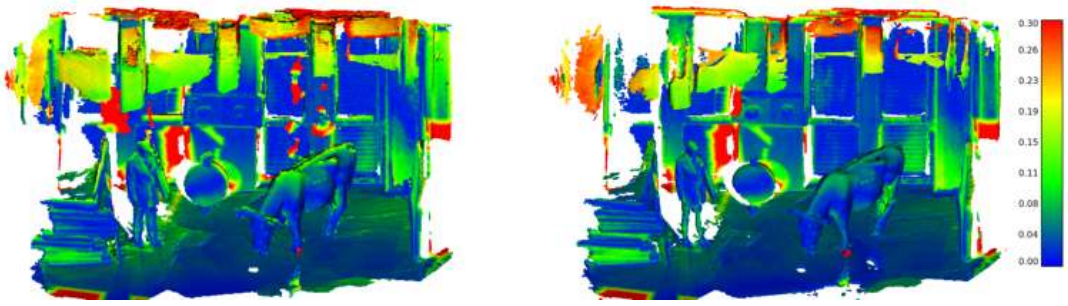
Tab. 7.6 shows the mapping accuracy results with and without space carving for the different datasets. Generally, the results with space carving are more accurate as it remove dynamics and can further clean up the free space of erroneous measurements or noise.

On the KITTI dataset, we see that our mapping pipeline can produce more accurate maps than Octomap and Voxblox, while the quantitative difference between Voxblox and our pipeline is striking. Fig. 7.12 qualitatively shows the point-wise difference between the ground truth map and the mapping result of our pipeline. Note that mapping without space carving includes dynamics caused by cars and pedestrians (shown by the green/red traces). In contrast, the map with



(a) Without space carving (b) With space carving

Figure 7.12: Mapping accuracy results for the KITTI Dataset. In (a), we show the results without space carving. The reconstructed dynamic objects can be seen by the red color corresponding to a large error (red circles). (b), shows the results with space carving, removing the dynamics and parts of the static scene, as visible at the boundaries of the cars.



(a) Without space carving (b) With space carving

Figure 7.13: Mapping accuracy results for the Cow and Lady dataset. In (a), we show the results without space carving with some visible errors on the wall in the back. (b) shows the results without space carving, these artifacts are effectively removed, and the resulting map is more accurate.

space carving effectively removes the parts corresponding to dynamics. Overall, the point-wise error is very low, i.e., the reconstructed map is very accurate, as indicated by the blue color of the distances.

On the Cow and Lady dataset, we can observe that the mapping results of our pipeline are an order of magnitude more accurate than the results produced by Octomap and Voxblox. Fig. 7.11 shows the extracted meshes from Voxblox and our pipeline. We can observe much more artifacts in the extracted mesh of Voxblox. These artifacts explain why the Voxblox map less accurately represents the environment compared to our reconstructed surface. Fig. 7.13 shows qualitatively the attained point-wise reconstruction accuracy, where we again note

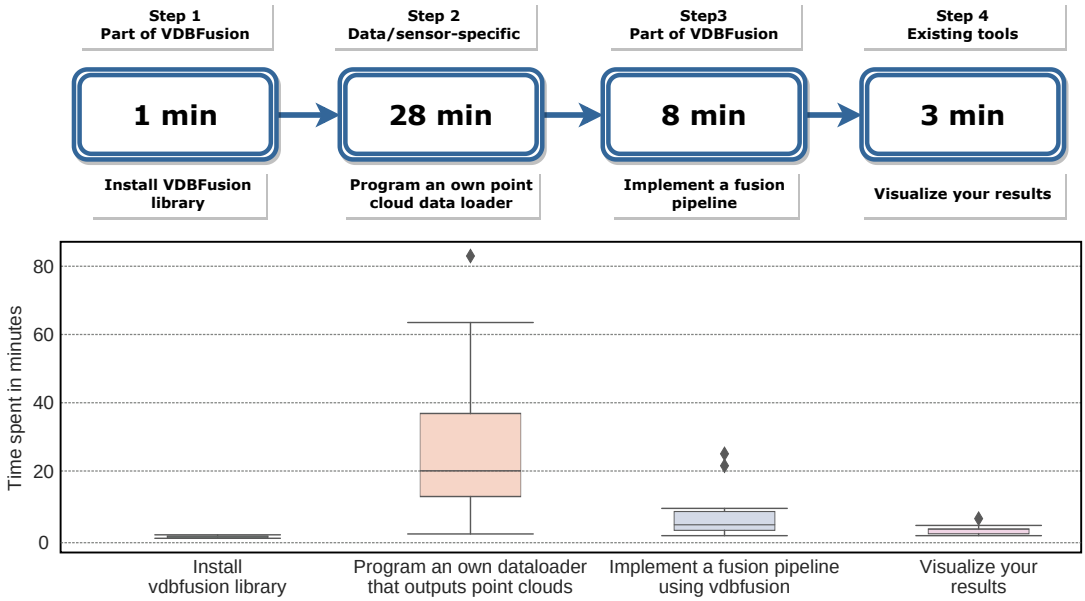


Figure 7.14: Results of our user study. Here, we show the essential steps and the time needed to perform them as timed by the participants. As can be seen, the most time-consuming part is writing a data loader. Top: average time spent per task; lower: plot distribution of collected results that highlights the data distribution and the outliers. The diamonds indicate the outliers of the distribution.

that the overall blue color shows the highly accurate reconstruction results of our pipeline.

Overall, the provided quantitative evaluation of the mapping accuracy on indoor and outdoor data shows that our pipeline can reconstruct the different datasets accurately.

Please note that, for the experiments, we used the C++ Voxblox library, although the results do not correlate with the ones shown in the original publication [130]. We suspect that is due to a numeric error in the internal non-standard transformation library. We contacted the authors to investigate a solution but could not arrive at one. We also remind the reader that although it would be possible to use the ROS interface, we aim at investigating framework-independent systems; in addition, it requires converting publicly available datasets into ROS[7].

7.4.5 User Study on the Ease-of-Use

While other approaches claim that they are easy to use or provide a generic and extensive library, we investigate this property by conducting a user case study on using our VDBFusion library to provide a quantitative evaluation.

For this, we sampled people from a group of Master's and Ph.D. students. They all took a robotics lecture at university in the past, but most participants

[7] `https://github.com/ethz-asl/voxblox/issues/373`

had no prior experience writing volumetric integration pipelines.

We only provided the participants with the pip package (Python API) and a small set of instructions for using it. We did not instruct them on which dataset to use nor which sensor modality, and we also did not enforce any third-party library since our library only requires numpy to work. We asked the participants to record the times for different steps in generating a map model from the data of their choice. More specifically, we identified the following essential steps needed to get our pipeline running: (1) installing the library, (2) coding the data loader to read the data from disk and provide point clouds as numpy arrays, (3) setting up the fusion pipeline to fuse the data, and, finally, (4) visualizing the generated maps using the provided tools. The instructions provided to the participants while conducting this study are publicly available[8].

Fig. 7.14 shows the times reported by the participants with the mean and standard deviation. We notice in this experiment that the most time-consuming task is to write the data loader for the pipeline, i.e., turning the data that the participants have on disk into a point cloud as a numpy array as required by our pipeline. We also highlight that this step is independent of our system; we also note that this step is nowadays a common task for any computer vision and mobile robotics application, meaning that existing data loaders should be available or can be adapted to our system easily.

As our library is distributed via `pip`, the installation of the library is only limited by the quality of the internet connection. The installation is achieved via running a single terminal command. Setting up the pipeline is a simple for loop that obtains data from the data loader and calls the integration method. Finally, the visualization of the results is also achieved with a couple of lines of code.

As indicated by the user study, we can attest that the claim of ease-of-use of the provided fusion pipeline is well supported. All participants were able to write a complete mapping pipeline, including their data loaders, in less than 1 h, on average 40 min, without any external assistance. We could speed up the process by providing data loaders for the most commonly used mapping datasets.

7.4.6 Qualitative Results

In this section, we aim to showcase multiple usages of our system (see Figures 7.15–7.20 on pages 109–111). We do not include extensive explanations for the experiments for brevity. We only mention the parameters used for the results and some qualitative numbers such as memory consumption, dense grid equivalent, the size on disk, etc. In this section, we do not aim to compare

[8] `https://www.ipb.uni-bonn.de/html/software/vdbfusion/vdbfusion_user_case_study.pdf`

against any other baseline but rather show that our system can be applied to multiple domains and sensors. All necessary codes for these experiments are part of the open-source release. Most examples require around 30 lines of Python code, similar to the snippets in Sec. 7.3.7.

7.4.6.1 KITTI Odometry Dataset

In Fig. 7.15, we present qualitative results on sequence 00 of the KITTI Odometry Dataset [63]. The dataset uses a 64-beam rotating Velodyne LiDAR sensor mounted on the roof of a car. The dataset provides outdoor scenes of urban, country, and highway environments in Germany. The loop-closed poses used to build the map shown in Fig. 7.15 are the output of the 3D-SLAM system SuMa [6].

7.4.6.2 Newer College Dataset

The newer college dataset was obtained using a hand-held device through New College, Oxford. For our experiments, we only use the LiDAR data from the 64-beam Ouster sensor used in the system. We use the poses provided as a reference with the dataset, which are not globally optimized. In Fig. 7.16, we present the results of our system using the short experiment sequence.

7.4.6.3 nuScenes Dataset

The nuScenes dataset is a large-scale public dataset that includes LiDAR data from a 32-beam Velodyne LiDAR scanner mounted on the roof of a car. The dataset was recorded in urban environments in Boston and Singapore. We exhibit the results of our fusion pipeline on sequence scene-0061 in Fig. 7.17. We use the ground-truth poses provided by the dataset.

7.4.6.4 Apollo Dataset

The Apollo-SouthBay Dataset [106] is a dataset that was collected by driving through different areas in the southern San Francisco Bay Area. The point clouds are obtained with a Velodyne HDL-64E LiDAR mounted on the roof of a car, and the ground-truth poses used for the experiment are obtained with the integrated navigation system for data collection. We show the results of our system on the Columbia Park sequence of the dataset in Fig. 7.18.

7.4.6.5 ICL-NUIM Dataset

The ICL-NUIM dataset is a synthetic RGB-D dataset. It consists of two different scenes with ground truths. We use the Living room scene without the simulated

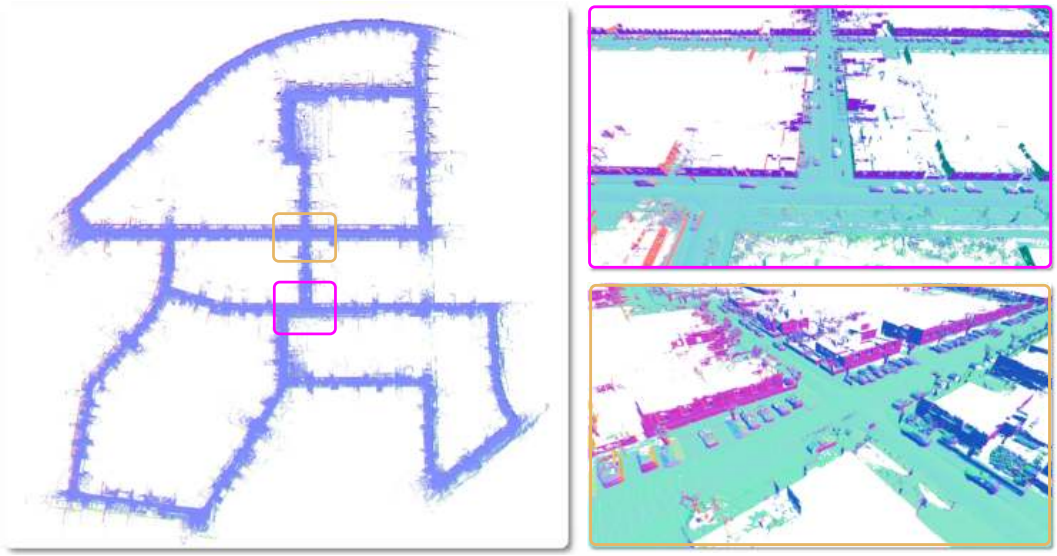
noise to obtain the result shown in Fig. 7.19. This scene has the depth maps used in our system after converting it to point clouds, together with ground truth camera poses.

7.4.6.6 TUM Dataset

The TUM Dataset [176] is a large dataset containing RGB-D and ground-truth data. The dataset contains the color and depth images of a Microsoft Kinect sensor along the ground-truth trajectory of the sensor. The data were recorded at a full frame rate of 30 Hz and a sensor resolution of 640x480. We test our system on the TUM dataset and display the qualitative results of the freiburg1_xyz sequence in Fig. 7.20.

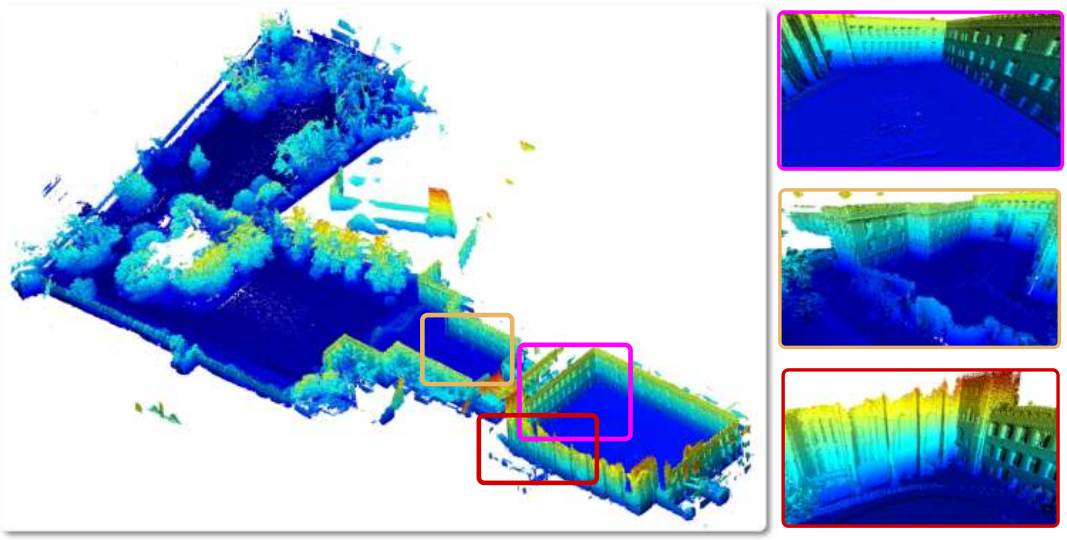
7.5 Conclusions

In this chapter, we proposed an approach for volumetric mapping based on TSDF representation that exploits a readily available efficient and sparse data structure called VDB. The main contribution of this chapter is an effective mapping system: VDBFusion, which also comes as an open-source library and does not require making assumptions about the size of the environment to be mapped. We described our practical implementation that provides an easy-to-use mapping framework that can be used for various sensors providing 3D measurements, such as RGB-D or LiDAR sensors. To handle different kinds of data, we argue that working directly with point clouds makes it possible to use a standard mapping pipeline. To this end, we use sensor/dataset-specific data loaders that prepare the data so that our mapping pipeline can consume it. Experiments on runtime and memory efficiency show that our implementation is more efficient than other open-source mapping frameworks supporting LiDAR and RGB-D data and providing high-quality maps. The evaluation of the mapping accuracy reveals that our approach is more accurate than another TSDF-based pipeline. Lastly, we conducted a user study to verify our claim of easy usage. Our library is provided as open source under the MIT license and available in C++ and Python. We hope our open-source library opens the door for further research by providing a sane starting point for TSDF-based mapping. Our system can be used in practice to build high-resolution mapping systems for mobile robots. The deployment of such systems in the real world is only possible due to the high speed of execution of the integration pipeline implemented in combination with the low memory requirements.



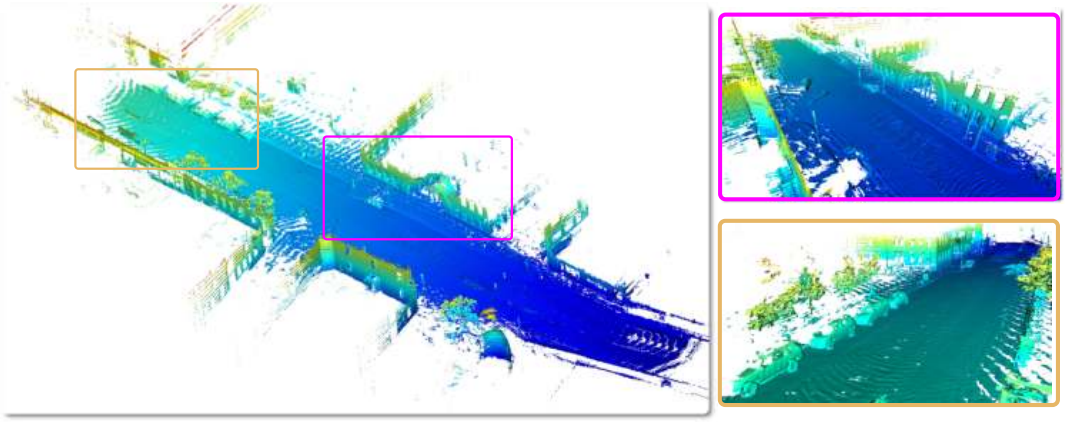
Mapping parameter		Dataset statistics		Dataset size	
voxel size	0.1 m	#scans	4541	extent [m]	627 × 661 × 40
truncation	3 voxels	points/scan	124,323	memory size	3,302.96 MB
space carving	False	min. range	2.0 m	dense size	122.01 GB
frame rate	18.51	max. range	70.0 m	disk size	760 MB

Figure 7.15: Qualitative results on the KITTI Vision Benchmark [63] with given parameters, dataset statistics, and size of the dataset in respect to the spatial extent and memory.



Mapping parameter		Dataset statistics		Dataset size	
voxel size	0.1 m	#scans	15,301	extent [m]	387 × 412 × 90
truncation	3 voxels	points/scan	25,377	memory size	1470.9 MB
space carving	False	min. range	1.0 m	dense size	106.29 GB
frame rate	32.18	max. range	200.0 m	disk size	591.83 MB

Figure 7.16: Qualitative results on the Newer College Dataset [144] with given parameters, dataset statistics, and size of the dataset in respect to the spatial extent and memory.

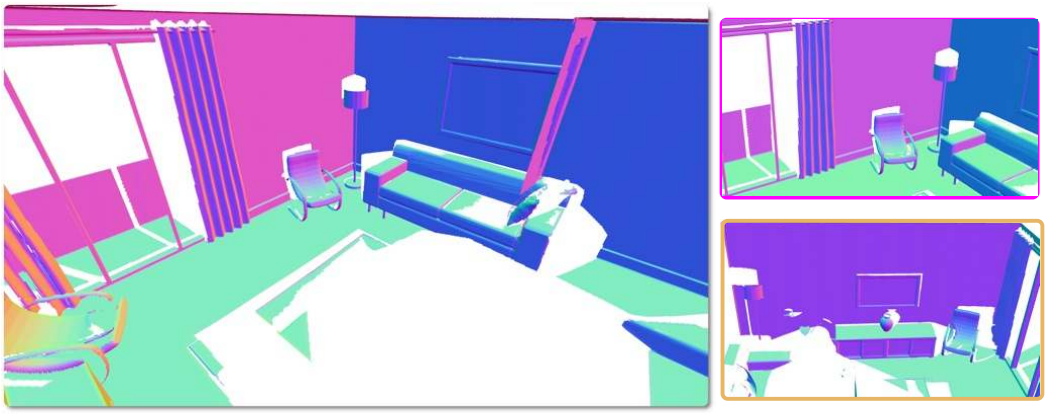


Mapping parameter		Dataset statistics		Dataset size	
voxel size	0.1 m	#scans	389	extent [m]	198 × 311 × 41
truncation	3 voxels	points/scan	20,311	memory size	277.54 MB
space carving	False	min. range	2.0 m	dense size	18.63 GB
frame rate	64.06	max. range	100.0 m	disk size	51.43 MB

Figure 7.17: Qualitative results on the nuScenes dataset [20] with given parameters, dataset statistics, and size of the dataset in respect to the spatial extent and memory.

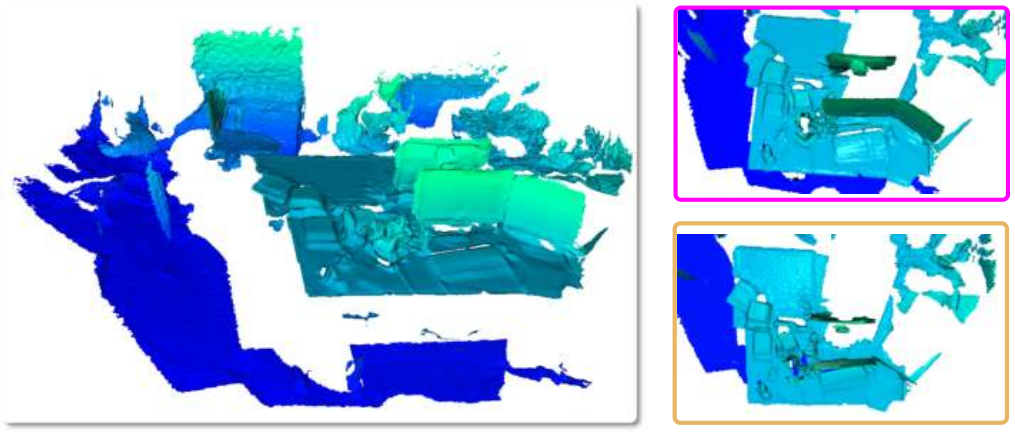
Mapping parameter		Dataset statistics		Dataset size	
voxel size	0.01 m	#scans	20,801	extent [m]	945 × 567 × 58
truncation	3 voxels	points/scan	105,287	memory size	2886.92 MB
space carving	False	min. range	1 m	dense size	228.50 GB
frame rate	18.33	max. range	100 m	disk size	574.54 MB

Figure 7.18: Qualitative results on the Apollo dataset [106] with given parameters, dataset statistics, and size of the dataset with respect to the spatial extent and memory.



Mapping parameter		Dataset statistics		Dataset size	
voxel size	0.01 m	#scans	1,240	extent [m]	$7 \times 7 \times 10$
truncation	3 voxels	points/scan	307,200	memory size	115.33 MB
space carving	False	min. range	unspecified	dense size	1.14 GB
frame rate	7.83	max. range	unspecified	disk size	33.63 MB

Figure 7.19: Qualitative results on the ICL-NUIM dataset [73] with given parameters, dataset statistics, and size of the dataset in respect to the spatial extent and memory.



Mapping parameter		Dataset statistics		Dataset size	
voxel size	0.01 m	#scans	798	extent [m]	$7 \times 6 \times 5$
truncation	8 voxels	points/scan	229,875	memory size	57.94 MB
space carving	False	min. range	unspecified	dense size	1.14 GB
frame rate	4.33	max. range	unspecified	disk size	20.55 MB

Figure 7.20: Qualitative results on the TUM RGB-D dataset [176] with given parameters, dataset statistics, and size of the dataset in respect to the spatial extent and memory.

Chapter 8

Dense Mapping Using Low-Resolution Sensors

The price for a robotic LiDAR sensor scales roughly linearly with the number of beams and thus the vertical resolution of the scanner. In general, the cheaper the sensors, the sparser the point cloud. This chapter addresses the problem of building dense models using sparse range data. Instead of requiring the vehicle to move slowly through the environment or traverse the scene multiple times to cover the space densely, we investigate geometric scan completion through a learning-based approach. We revisit VDBFusion, introduced in Chapter 7, and propose a neural network to aid 3D reconstruction frame-by-frame by completing each scan towards a dense TSDF volume. We propose a geometric scan completion network trained self-supervised without manually annotated datasets. Our experiments illustrate that such frame-wise completion leads to maps that are on par or better than maps generated using a higher-resolution LiDAR sensor. We also show that our system improves the performance of existing SLAM systems when low-resolution LiDARs are used.

This chapter presents our approach to solving the problem of completing sparse 3D LiDAR scans in a frame-to-frame fashion for TSDF mapping using deep learning. Fig. 8.1 illustrates our central question: "Based on data obtained from a single scan of a 16-beam LiDAR, can we estimate and hallucinate how the scene looks like?" To investigate this, we revisit the proposed volumetric mapping system introduced in Chapter 7 and combine it with a learning approach. We aim to exploit the best of geometric and deep learning worlds. In contrast to recent works on scan completion [47, 123, 195], we target single geometric scan completion in large outdoor environments where existing completion approaches fail to operate because of the memory limitations of commonly used GPUs.

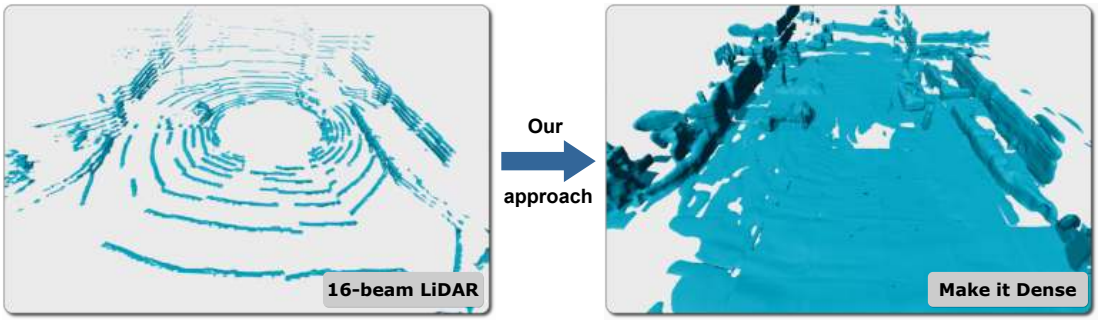


Figure 8.1: A TSDF-based surface model from a single 16-beam LiDAR scan (left) turned into a denser, completed TSDF-based surface model (right) by the learning-based approach proposed in this chapter.

8.1 Geometric Scan Completion

Most autonomous vehicles rely on some form of mapping. Modern outdoor robots and self-driving cars are equipped with 3D sensors such as RGB-D cameras or LiDARs to perceive their surroundings. Volumetric mapping has shown to be an effective approach to creating models of the surrounding environment [44, 126, 190]. Most of the existing volumetric mapping works rely on the fact that they integrate dense 3D data at a relatively high frame rate into a model, often at 10 fps to 30 fps.

Outdoor robots often use 3D LiDARs such as Velodyne or Ouster scanners, which have between 16 and 128 beams. Their price scales roughly linearly with their number of beams and, thus, the vertical resolution of the scans. The denser a single scan, the more expensive the scanner. Thus, it is a relevant research question whether we can build a mapping or SLAM system that generates dense models but only requires sensors with a low vertical (or horizontal) resolution.

The main contribution of this chapter is a self-supervised approach for turning a sparse 3D LiDAR scan into a comparably dense TSDF representation. We propose a 3D CNN trained in a self-supervised manner that completes the reconstructed scene on a frame-to-frame basis. We aim at completing *single scans* instead of completing a scene created from aggregated scans offline. In our approach, we process the 3D LiDAR data and pass it to our CNN. The network output is a TSDF representation that encodes the most recent observation plus synthetically completed data fused into a global map.

Our experiments show that our approach can complete sparse LiDAR scans and improve mapping results, as well as the performance of existing SLAM systems [6]. We see this as a step towards getting better representations with cheaper scanners, thus reducing hardware costs through more intelligent software. We also

publish our code with the pre-trained models[1].

In sum, we propose a geometry-driven system that integrates sparse LiDAR scans into volumetric maps by completing each scan using a 3D CNN. Our approach aims at taking the best of both worlds: classical mapping and deep learning.

8.2 Make it Dense

Our approach creates a dense reconstruction of single LiDAR scans recorded with a low vertical resolution, meaning few (e.g., 16) beams, to obtain results similar to those recorded with a larger number of beams (e.g., 64). We target processing every single scan as it gets recorded and we do not target a post-processing solution as Dai et al. [47] for RGB-D mapping or PUMA, the offline mapping system presented in Chapter 5.

Fig. 8.2 shows an overview of our processing pipeline that combines classical mapping with a learning-based refinement to generate high-fidelity representations from sparse LiDAR measurements. We first generate a TSDF-based volumetric representation of a single scan, which we denote as $\boldsymbol{D}_t(\boldsymbol{x})$, and (optionally) corresponding weights, denoted as $\boldsymbol{W}_t(\boldsymbol{x})$. Here, $\boldsymbol{x}$ refers to the voxel location in the grids. Then, we use $\boldsymbol{D}_t(\boldsymbol{x})$ to predict a new volume of TSDF values, denoted as $\boldsymbol{D}_t^P(\boldsymbol{x})$ using a fully convolutional network. The network is trained to fill in values in a self-supervised way as if the data would have been recorded with a high-resolution LiDAR.

As the grids are globally aligned, we integrate the scan-based predicted grid, $\boldsymbol{D}_t^P(\boldsymbol{x})$, into a global grid $\boldsymbol{D}_t^G(\boldsymbol{x})$. From the global signed distance fields, we can determine a surface representation using marching cubes [101]. In the following sections, we describe the individual processing steps in more detail.

8.2.1 Scan Integration Using TSDF

The LiDAR generates a point cloud $\mathcal{P}_t = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\}$ of N points $\boldsymbol{p}_i \in \mathbb{R}^3$ at time t. We assume to have an estimate of the current pose of the sensor $\mathsf{T}_t \in SE(3)$ available from KISS-ICP (Chapter 4) or from a GPS/IMU combination.

To obtain the TSDF representation of the scan, we employ VDBFusion with voxel size of v_{size}, truncation distance of ψ_{TD} and no space carving enabled, see Sec. 7.3.2 and Sec. 7.3.3 for further details. We then extract the truncated signed distance field $\boldsymbol{D}_t(\boldsymbol{x})$ and the weight grid $\boldsymbol{W}_t(\boldsymbol{x})$ from the mapping system. As a result of this step, we end up with a volumetric scalar field $\boldsymbol{D}_t(\boldsymbol{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ representing the TSDF.

[1] `https://github.com/PRBonn/make_it_dense`

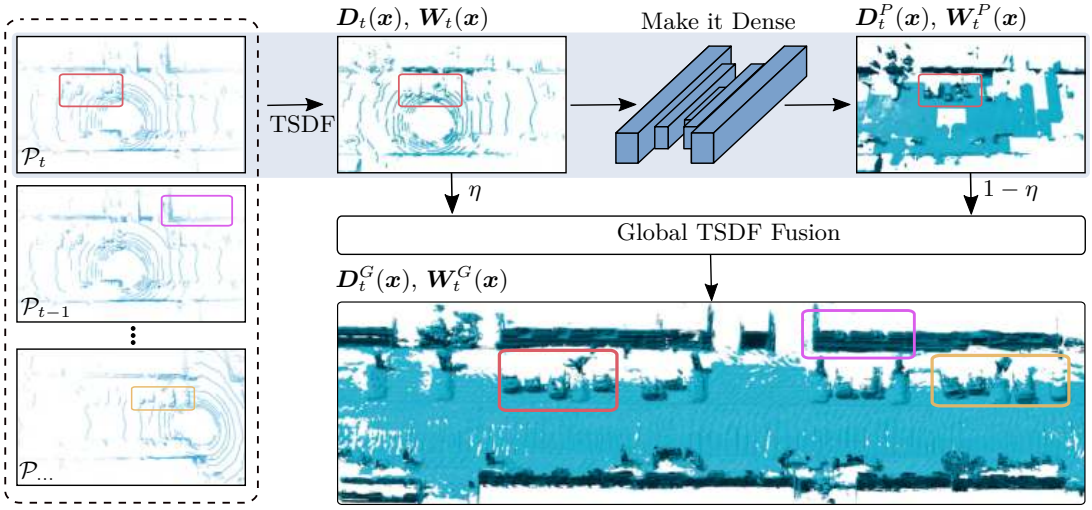


Figure 8.2: Overview of our approach. We first generate a TSDF volume $\boldsymbol{D}_t(\boldsymbol{x})$ of a single scan $\mathcal{P}_t$ at time t. We then apply our geometric scan completion network in a strided fashion, such that missing values are added to the single-scan TSDF, giving a more complete TSDF volume $\boldsymbol{D}_t^P(\boldsymbol{x})$. The predicted TSDF values are then used to update the global TSDF representation $\boldsymbol{D}_t^G(\boldsymbol{x})$ using a weighting term η to avoid integrating the same observation twice into the map. The colored boxes highlight different areas of the input scans and their corresponding reconstruction in the final fusion results.

8.2.2 Geometric Scan Completion

The TSDF grid $\boldsymbol{D}_t(\boldsymbol{x})$ encodes the surface representation of the single scan and is then passed through our CNN architecture that predicts a new TSDF grid $\boldsymbol{D}_t^P(\boldsymbol{x})$ with the exact dimensions as $\boldsymbol{D}_t(\boldsymbol{x})$ to create the dense information. This is done for each incoming scan individually.

Network input. Instead of learning to regress a TSDF value directly from raw sensor data, we input a batch of TSDF volumes to the network and train it to improve its quality towards a more expensive sensor with more LiDAR beams. To input these volumes to our network, the TSDF representation $\boldsymbol{D}_t(\boldsymbol{x})$, is split into non-overlapping regions. These volumes are batched into a dense multidimensional tensor and then fed to our network. By storing the coordinates of the origin of each of these volumes, we can later reconstruct the original scene once the network has completed the TSDF values. Instead of using point-wise operations [150, 180], we can directly use 3D convolutions on these volumes. Since we operate on smaller sub-volumes from a single scan instead of a whole scene, our network architecture is small and fast to train, making it relatively compact and deployable on mobile robots. The fully convolutional design and the small number of parameters also make it hard for the network to overfit to a particular dataset [131, 217].

8.2.3 Architecture

Our architecture is based on a 3D convolutional encoder-decoder architecture with skip connections between each encoder stage's output and the decoder stage's input with the exact feature map dimensions. Our model is very similar to the architecture proposed by Dai et al. [47], with the difference that our architecture can process directly dense volumes. Therefore, there is no need to convert back and forth between dense and sparse tensors. Analogously, our architecture is also similar to the Atlas [123] architecture, but in contrast, we do not fix the volume size but allow the scene to be arbitrarily large. In essence, our model reassembles a 3D-UNet [42, 151] architecture. More specifically, we first compute volumetric features using standard, dense 3D convolutions to increase the number of channels. For each subsequent encoder step, we increase the number of output channels by a factor of 2 using standard 3D convolutions and subsequently reduce the volumetric resolution with strided 3D convolutions. For upsampling, we use transposed convolutions to regain resolution in the output. Each convolution is followed by batch normalization and a ReLU activation.

Overall, we have a symmetrical encoder-decoder structure with $S = 3$ stages to achieve the exact spatial resolution in the output as with the input voxel grid. Let $\mathcal{D}_i$ be the i^{th} decoder stage of a transposed convolution followed by a convolutional layer with an output dimension $M \times M \times M$ of the corresponding scalar field. Consequently, the output of the $\mathcal{D}_{i-1}$ stage is $M/2 \times M/2 \times M/2$ and therefore $\mathcal{D}_1$ denotes the first decoder stage after the last encoder stage, which corresponds to a strided convolution followed by a convolutional layer.

We not only predict a scalar field after the final decoder stage, i.e., $\mathcal{D}_S$, but also generate intermediate outputs of the intermediate decoder stages $\mathcal{D}_1, \ldots, \mathcal{D}_{S-1}$. To transform the output of each decoder stage at every resolution level, we use convolutions with kernel size 1 to reduce the number of channels to 1 (the scalar field) and use tanh as an activation function to predict both positive and negative values.

8.2.4 Multi-Resolution Loss

We optimize our network end-to-end with a masked multi-resolution ℓ_1 loss between the predicted TSDF values (at all decoder stages, i.e., $\mathcal{D}_1, \ldots, \mathcal{D}_S$) and the target TSDF values. In line with prior work [46, 47, 123, 177], we log-transform the predicted and target values before applying the ℓ_1 loss. This enables us to obtain better predictions near the surface [46, 47]. In contrast to recent works in scene completion [47, 195], we do not add any classification layer to predict invalid or occlusions.

At each decoder stage $\mathcal{D}_i$, we mask out regions where the predicted TSDF val-

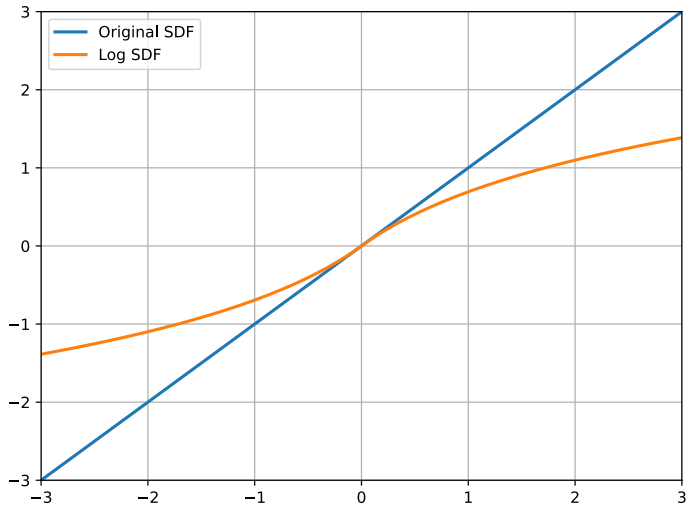


Figure 8.3: Illustration of how the log transformation function changes the originally signed distance values, close to the origin (and thus, close to the surface) the values are not much affected, but when the signed distance value increase, the log transformation will decrease the value, meaning that it will have less influence when computing the ℓ_1 loss.

ues are equal to the background value ψ_{TD}. The following decoder stages will skip the loss computation on those voxels and focus more on the fine-grained TSDF predictions at surface boundaries. Furthermore, we mask all the planes in the target volumes equal to the background value [123], which avoids artifacts on the predicted scalar field.

We denote by $\mathcal{X}_i$ all locations relevant for computation of the loss at decoder stage $\mathcal{D}_i$, since they are not already predicted by stage $i-1$ or masked out. Therefore, we use the following loss for the i-th resolution:

$$\mathcal{L}(\hat{D}_i, D_i) = \sum_{\boldsymbol{x} \in \mathcal{X}_i} \|\phi(\hat{D}_i(\boldsymbol{x})) - \phi(D_i(\boldsymbol{x}))\|_1, \tag{8.1}$$

where $\hat{D}_i \in \mathbb{R}^{M \times M \times M}$ and $D_i \in \mathbb{R}^{M \times M \times M}$ correspond to the target TSDF volume and the predicted TSDF volume at the decoder stage i, respectively, and $\phi(x) : \mathbb{R} \to \mathbb{R}$ is the log transform, defined as

$$\phi(x) = \text{sgn}(x)\ \log(|x| + 1) \tag{8.2}$$

$$\text{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{8.3}$$

An illustration of how the log-transform modifies the signed distance values is illustrated in Fig. 8.3.

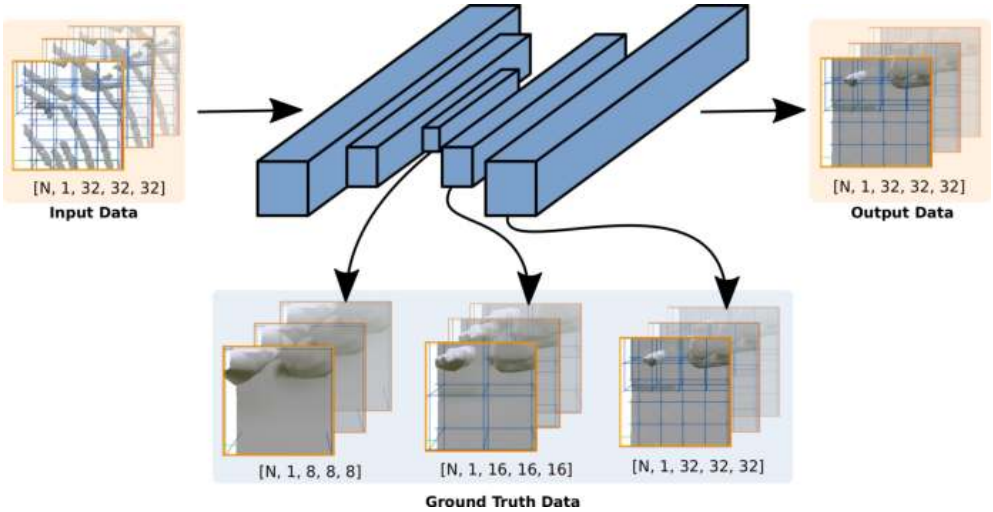


Figure 8.4: Data pipeline flow of our approach. We extract the training samples and batch them together, forming a tensor of dimension $[N, 32, 32, 32]$. In this figure, we show how the volumes pass through our network and which are the sizes of the intermediate representation of the environment obtained by the network. At the end of the network, we have a tensor with the same dimensions as the input but with completed TSDF values.

In sum, to obtain better predictions close to the surface, we log-transform the TSDF values of the network output and the supervision data before applying the ℓ_1 loss. Overall, we, therefore, minimize the following loss:

$$\mathcal{L} = \sum_{i \in \{1,\dots,S\}} \mathcal{L}(\hat{D}_i, D_i). \tag{8.4}$$

8.2.5 Self-Supervised Training

We train our geometric scan completion network self-supervised by running VDB-Fusion presented in Chapter 7 on the scans of the KITTI odometry dataset [63]. An overview of our training pipeline is depicted in Fig. 8.4. To achieve self-supervision, we take the sequential scans with known poses and build the following TSDF representations. First, we generate the desired input for our network using a subset of beams of the Velodyne HDL-64E sensor. To this end, we keep every 4^{th} row from a range-image representation of the original scan. While this input scan does not represent the exact sensor characteristics of a low-cost LiDAR sensor, e.g., a Velodyne VLP-16, it is close to it in terms of density.

Next, we generate the multi-resolution targets, i.e., $\hat{D}_1, \dots, \hat{D}_S$, by applying the TSDF pipeline on the original scan without dropping beams. To increase the density of the targets, we aggregate n_{past}=25 scans in the past and n_{future}=25 scans in the future for the current frame. Furthermore, to avoid dynamic objects corrupting the supervision data, we use SemanticKITTI labels [5] and remove

dynamic objects from aggregated scans.

To train the network efficiently, we split the generated target TSDF representations into non-overlapping volumes of 32×32×32, 16×16×16, and 8×8×8 voxels, respectively. Due to the different voxel sizes for the three representations, each extracted volume spans a total of $3.2\,\mathrm{m}^3$ in the volumetric space. At test time, we use non-overlapping contiguous volumes of 32×32×32 voxels since this empirically provided the best results.

8.2.6 Global Map Update

Once we have the two observations, one based on sensor data and the other based on the geometric scan completion network, we integrate this information into our global volumetric grid $\boldsymbol{D}_t^G(\boldsymbol{x})$.

It is important to note that we need to provide a way to combine the two data sources without integrating the same observation twice. The real TSDF measurements are sparse and incomplete but do not contain reconstruction artifacts. In contrast, the predicted TSDF volumes are denser and more complete, but they hold reconstruction artifacts due to prediction errors in the network. To fuse these two sources, we introduce a weighting term $\eta \in [0, 1]$. The term η specifies how much more we want to trust an actual measurement over a predicted value.

In TSDF mapping using RGB-D cameras, one may use the weight matrix $\boldsymbol{W}_t(\boldsymbol{x})$ to control the update of the TSDF grid [17, 126, 130] and, for example, down-weight certain measurements within a single image. This, however, is much less relevant for LiDAR scans where measurements have similar accuracy. Thus, we generally set $\boldsymbol{W}_t(\boldsymbol{x}) = 1$ for all cells next to the truncation distance ψ_{TD} and 0 otherwise. We do the same for the weight for the predicted scan $\boldsymbol{W}_t^P(\boldsymbol{x})$. One can introduce different weights here if more knowledge about uncertainties is available.

We use the factor η to weigh the actual observations and the predicted ones consistently, making sure information is not incorporated multiple times. We can effectively fuse both data sources without duplicating the observation at every timestamp by:

$$\Delta\mathbf{D}(\boldsymbol{x}) = \eta \boldsymbol{W}_t(\boldsymbol{x}) \boldsymbol{D}_t(\boldsymbol{x}) + (1-\eta) \boldsymbol{W}_t^P(\boldsymbol{x}) \boldsymbol{D}_t^P(\boldsymbol{x}) \tag{8.5}$$

$$\Delta\mathbf{W}(\boldsymbol{x}) = \eta \boldsymbol{W}_t(\boldsymbol{x}) + (1-\eta) \boldsymbol{W}_t^P(\boldsymbol{x}). \tag{8.6}$$

Intuitively choosing a η factor close to 1 will completely discard the prediction of the network while keeping the real TSDF values, producing fewer artifacts on the output but having a sparse and incomplete model of the scene. In contrast, picking η close to 0 will entirely reject the real TSDF measurements and keep only the prediction of the network, producing a dense map representation but with

more artifacts on the reconstruction. In our work, we find empirically that a good compromise is to set η=0.7. Once we have the aggregated signed distance field (SDF) values $\Delta\mathbf{D}(\boldsymbol{x})$ for the two sources and their respective weights $\Delta\mathbf{W}(\boldsymbol{x})$, we fuse both analogously to Curless and Levoy [44] for all voxels at location $\boldsymbol{x}$ as follows:

$$\boldsymbol{D}_t^G(\boldsymbol{x}) = \frac{\boldsymbol{W}_{t-1}^G(\boldsymbol{x}) \cdot \boldsymbol{D}_{t-1}^G(\boldsymbol{x}) + \Delta\mathbf{D}(\boldsymbol{x})}{\boldsymbol{W}_{t-1}^G(\boldsymbol{x}) + \Delta\mathbf{W}(\boldsymbol{x})} \tag{8.7}$$

$$\boldsymbol{W}_t^G(\boldsymbol{x}) = \boldsymbol{W}_{t-1}^G(\boldsymbol{x}) + \Delta\mathbf{W}(\boldsymbol{x}). \tag{8.8}$$

8.3 Experimental Evaluation

The main focus of our work is to complete sparse LiDAR scans to aid in mapping the environment while navigating through it. The experiments are designed to showcase the capabilities of our geometric scan completion approach. We will see that we can estimate comparably dense TSDF representations from a single 16-beam LiDAR scan using purely self-supervised training for our network. We obtain improved maps fusing real observations and observations produced by a CNN. Furthermore, we also show that our approach improves the accuracy of existing SLAM systems [6] when low-resolution scanners are employed.

8.3.1 Experimental Setup

We tested our method on a regular computer with an Intel i7-8700 with 3.2 GHz and an Nvidia GeForce ground truthX 1080 Ti with 11 GB of memory. We optimized our network with the Adam optimizer [89] and with an adaptive learning rate of 0.001. Our network has about 1.4 M parameters and already shows promising results after 1 hour of training. Nevertheless, we keep training for as long as 8 hours for fine-tuning some details. We use the KITTI odometry dataset [63] for evaluation and use labels provided by the SemanticKITTI dataset [4] to filter dynamic objects for the evaluation of the mapping results. We train our network on sequence 07 and report quantitative results on sequences 00 to 06 and 08 to 10. We use the GPS/IMU poses provided by the KITTI dataset. For all our experiments, we used a voxel size of v_{size}=0.10 m at the finest resolution. The truncated distance, also known as the background value, is set to ψ_{TD}=3 voxels.

For quantitative evaluation, we use the standard scene completion metric proposed by Song et al. [167], which measures the intersection-over-union (IoU) between a ground truth voxel grid of occupied voxels and the predicted occupancies. Note that, as explained in Sec. 8.3.2, we account for areas that are never observed and occluded and ignore these voxels.

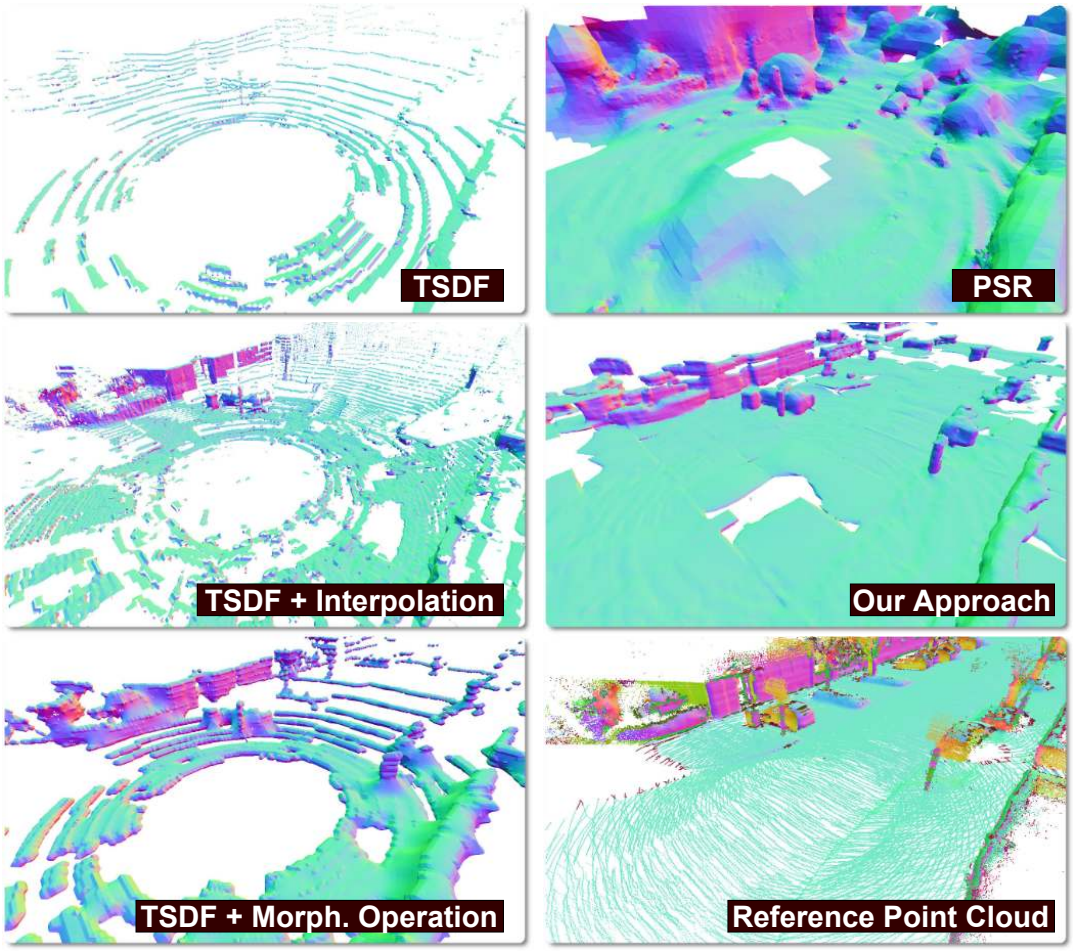


Figure 8.5: Illustration of the results obtained through different geometric scan completion approaches. All representations were built with just one 16-beam LiDAR point cloud except for the ground truth point cloud used for evaluation.

8.3.2 Geometric Scan Completion Completeness

The first experiment evaluates the performance of our approach for geometric scan completion. It shows we can predict a local scene in an urban environment using a single 16-beam LiDAR scan. As baselines for comparison, we use three other geometric approaches. First, a combination of morphological operations [82] such as opening and closing plus smoothing. Second, we use the meshing algorithm from PUMA, introduced in Chapter 5, based on PSR. For further details on the modified PSR [84] algorithm, see Sec. 5.1.4. Third, we apply tri-linear interpolation on the range-image representation of the 16-beam LiDAR scanner to up-sample the point cloud to a 64-beam scan. This experiment is vital to evaluate the performance of our approach as the neural network processes each scan and is immediately incorporated into the mapping pipeline.

To use scene completion metrics [167] we need to obtain ground truth voxel grids that contain only occupancy information. To this end, we follow the ap-

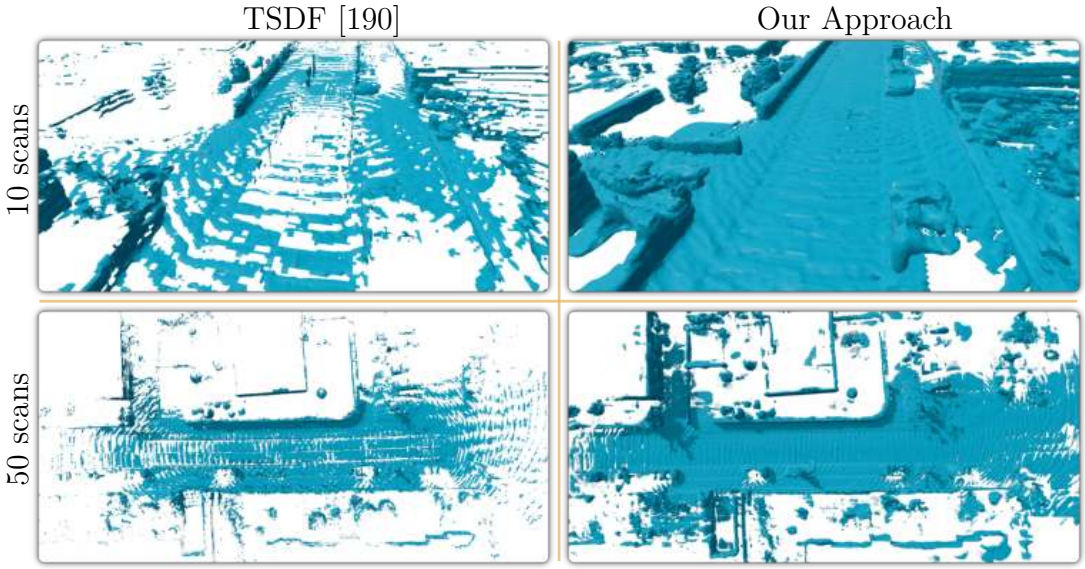


Figure 8.6: Results of TSDF-based mapping (left) and our approach (right) after aggregating and subsequently fusing 10 (top row) and 50 (bottom row) scans from a 16-beam LiDAR. As can be seen, our approach yields a more complete model than standard TSDF mapping. The LiDAR frames are taken from sequence 03 of the KITTI odometry benchmark [63].

proach of Behley et al. [5], and for each input scan at timestamp t, we aggregate $2L + 1$ full-resolution scans (L before, L after, and the current scan) into a reference point cloud. We use the labels from SemanticKITTI [5] and filter out dynamic objects in this reference point cloud. Lastly, we crop the reference point cloud with the bounding box of the input scan under evaluation. We then determine the occluded voxels by raycasting the point cloud from the known poses, such that never observed voxels, i.e., voxels that the raycasting cannot reach, are marked as occluded. We then proceed to obtain occupancy voxel grids for the methods under consideration. To ensure that the IoU is not affected by the density of the point sampling [206], we first obtain a triangle mesh representation for each method. We then sample a dense point cloud on these meshes and use it to obtain an occupancy voxel grid as described above. A qualitative illustration of this process is depicted in Fig. 8.5, and all methods visible in this figure will be compared to the reference point cloud. For the evaluation, we use every 500^{th} scans from each test sequence and average errors across each sequence. The reasoning behind this decision is based on our experimental observations. We found that utilizing more scans for evaluation did not significantly affect the metrics. Consequently, this approach enables us to speed up the system evaluation.

The results of this experiment are shown in Tab. 8.1. The first row of Tab. 8.1 provides the quantitative results for all methods under consideration when using a single 16-beam LiDAR scan. This is the most relevant experiment for our work, as it shows how well we can complete a single 16-beam LiDAR observation. Nevertheless, we also run all the methods using a single 64-beam scan. Our

approach outperforms all baselines in all the testing sequences for both sensor resolutions. It shows that we can complete sparse as well as dense scans.

To illustrate what such numbers mean, Fig. 8.5 shows a single scan and the completion results. The traditional TSDF-based method VDBFusion, presented in Chapter 7, can reconstruct the surfaces only where observations are available. Even when trying to augment the results of a traditional TSDF pipeline with morphological operations and upsampling, the results are not on par with our approach. PSR [85] showed overall the best performance among the baselines, but as seen on Fig. 8.5, the method tends to over-smooth the reconstruction results. Our approach outperforms all baselines and provides accurate reconstructions close to the observations, plus smooth and complete results for areas without measured points.

8.3.3 Improving Existing SLAM Systems

This experiment showcases how our approach improves existing 3D LiDAR SLAM pipelines, e.g. SuMa [6]. To conduct this experiment, we compare the pose estimates from the SLAM system using the standard KITTI metrics [63] with and without our network. As a baseline, we run SuMa with the 16-beam LiDAR using the default parameters. For our approach, we first complete each scan individually and then project the completed scan into the vertex and normals maps needed for SuMa [6]. Note that given the high resolution of the completed scan, we can choose any vertical resolution for the projection. For our experiments, we choose to use a vertical resolution of 64.

As shown in Tab. 8.2 the original SLAM system performs relatively poorly when employing low-resolution LiDAR scanners. We overcome the sparseness of the data by employing our geometric scan completion network, and, as a result, we can improve the pose estimation on all evaluated sequences, and our results show to be almost three times better than the baseline[2].

8.3.4 Qualitative Evaluation on Scan Sequences

16 beams. The next experiment illustrates how the presented self-supervised approach can aid a traditional volumetric reconstruction pipeline when considering whole *scan sequences* and not only individual scans. To do so, we illustrate the resulting local maps obtained when combining 10 and 50 consecutive 16-beam scans. We compare our results against VDBFusion (Chapter 7). As can be seen in Fig. 8.6, the traditional TSDF pipeline creates consistent maps, but the results are by far less complete when compared to our results. For example, the scene's street, sidewalks, cars, and other major shapes are properly reconstructed.

[2]Sequences 01 and 02 are excluded from the evaluation since both methods fail

	Approach / Sequence	00	01	02	03	04	05	06	08	09	10	Avg.
16-beams	Voxblox [130]	0.74	0.75	0.81	0.76	0.74	0.71	0.74	0.74	0.68	0.71	0.75
	TSDF [190]	7.01	7.20	7.83	6.93	7.03	7.02	7.25	6.84	6.75	7.05	7.20
	TSDF [190] + Upsample	14.08	14.10	15.83	13.97	14.12	14.21	14.65	13.66	13.76	14.55	14.10
	TSDF [190] + Morphological	13.39	13.68	14.32	13.11	13.33	13.76	13.81	13.14	13.72	13.67	13.68
	PSR [85]	15.38	15.19	16.09	15.20	15.27	15.37	15.60	14.98	14.94	15.67	15.19
	Our approach	**24.57**	**24.95**	**25.71**	**24.11**	**24.57**	**24.52**	**25.36**	**24.01**	**24.33**	**24.73**	**24.95**
64-beams	Voxblox [130]	13.35	13.88	14.66	13.30	13.52	13.54	14.07	12.84	12.89	13.72	13.88
	TSDF [190]	23.27	24.17	25.34	23.07	23.37	23.69	24.30	22.54	23.06	23.93	24.17
	TSDF [190] + Morphological	14.16	14.46	14.76	14.18	13.97	14.44	14.72	13.99	15.25	14.67	14.46
	PSR [85]	25.22	25.57	26.12	24.87	25.19	25.24	26.06	24.48	25.01	25.59	25.57
	Our approach	**33.44**	**33.70**	**34.31**	**33.09**	**33.29**	**33.53**	**34.64**	**32.79**	**33.78**	**33.58**	**33.70**

Table 8.1: Intersection-over-union results of the single geometric scan completion experiment. The first row exhibits the IoU for a 16-beam sensor, while the second row shows the results for a 64-beam one. Sequences are taken for the KITTI odometry benchmark [63].

Method	Error	00	03	04	05	06	07	08	09	10	Avg.
SuMa	trans.	3.39	8.83	8.25	3.27	4.37	2.03	4.60	6.18	6.60	5.28
(16 beams)	rot.	1.62	4.53	1.32	1.53	2.10	1.66	2.10	2.52	2.32	2.19
Ours	trans.	**1.45**	**4.99**	**1.05**	**0.66**	**0.87**	**1.06**	**1.52**	**2.52**	**2.87**	**1.89**
(16 beams)	rot.	**0.63**	**1.60**	**0.85**	**0.37**	**0.44**	**0.83**	**0.73**	**0.73**	**1.08**	**0.81**

Table 8.2: Pose estimates results on the KITTI dataset [63]. The relative translational error (trans.) is in % and the relative rotational error (rot.) in degrees per 100 m. All errors are averaged over trajectories of 100 to 800 m length. Bold numbers indicate the best approach for the given sequence. Sequences 01 and 02 are not reported due to failure in both methods.

64 beams. The following experiment evaluates qualitatively how our system can improve the mapping results in cases where the frame rate of the scanner is relatively low or when driving at high speeds. We generate synthetic data from an urban sequence by dropping frames but keeping the full resolution of the scanner. We choose sequence 00 from the KITTI dataset [63] and run VDBFusion presented in Chapter 7 as baseline. As shown in Fig. 8.7, traditional integration methods fail to reconstruct a dense map of the environment when the number of frames is relatively low. Our approach to geometric scan completion can help to cope with this type of situation, improving the results of the mapping pipeline as well as its completeness.

8.4 Conclusion

In this chapter, we proposed a novel mapping approach that builds dense 3D models from sparse LiDAR data. The main contribution of this chapter is a self-supervised approach for turning a sparse 3D LiDAR scan into a comparably dense TSDF representation. We propose a 3D CNN trained in a self-supervised manner that completes the reconstructed scene on a frame-to-frame basis. We investigate sparse 3D geometric scan completion through a learning-based approach. We combine traditional TSDF-based volumetric mapping with 3D convolutional neural networks to aid reconstruction on a frame-to-frame basis. From a single sparse scan, we can generate comparably dense TSDF surface models. Our completion network is trained in a fully self-supervised fashion. Our experiments suggest that maps built with 16 LiDAR beams are on par or even better than traditionally built TSDF maps generated using 64-LiDAR beams. Additionally, our results suggest that in modern robotics applications, it is beneficial to mix methods from two different worlds: deep-learning techniques, such as deep neural networks, combined with traditional geometric techniques, such as TSDF.

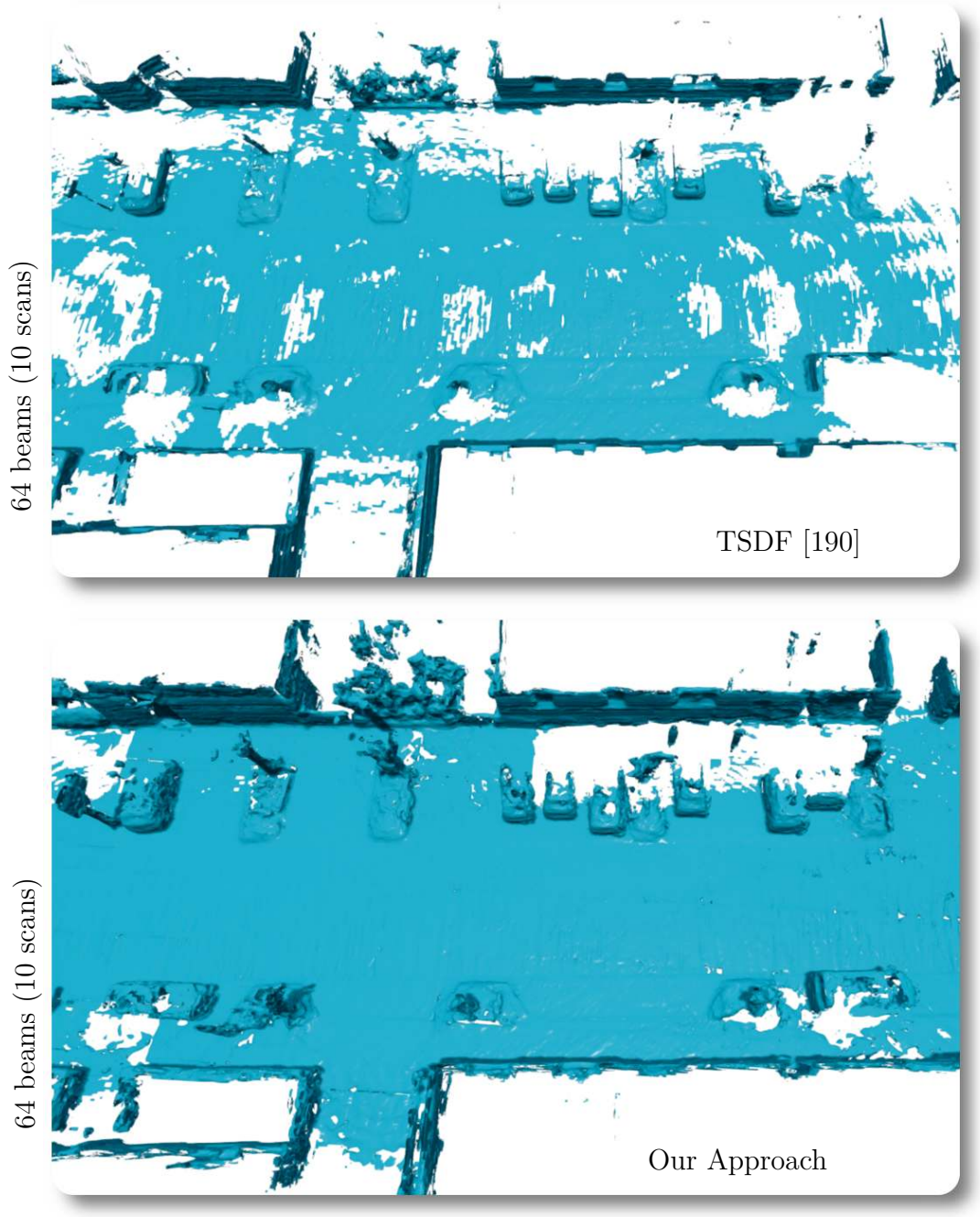


Figure 8.7: Qualitative results of a high-speed/low frame rate LiDAR sequence. The LiDAR used for this experiment uses 64 beams. Nevertheless, classical integration methods [190] (top row) cannot properly map the environment. Our geometric scan completion approach (bottom row) can aid the reconstruction system, enabling a more complete scene representation.

Chapter 9

Conclusion

Robots are valuable tools that can significantly assist humans in various fields. They can explore and gather data on remote planets like Mars or handle mundane or tedious tasks, such as daily housecleaning, which humans often find uninteresting. They also excel in managing complex situations where human errors could lead to severe consequences, such as driving vehicles. Additionally, robots enhance efficiency, precision, and continuity in tasks that we already do, like scanning large logistics warehouses. Their uninterrupted operation offers valuable insights for improving global logistical operations. However, facilitating these tasks comes with significant challenges. For example, a robot must collect data about its surroundings, interpret the sensor readings, create a map, and accurately determine its position and orientation in that environment. This entire process must happen in real time, reflecting the current conditions of the surroundings.

In this thesis, we addressed various challenges related to robot mapping using 3D LiDARs. Specifically, we introduced an approach to reliably and consistently record sensor data, which is crucial for robots navigating the real world. Additionally, we presented a technique for estimating the robot's current position and orientation in real-time as it moves through its surroundings. Furthermore, we tackled two different mapping scenarios and their practical applications. Firstly, we proposed an algorithm for creating 3D maps represented as triangle meshes. This representation was later utilized to solve a localization task. Second, we developed a technique for generating online 3D models of the environment as the robot processes new data. We leveraged data structures commonly used in the film industry for this purpose. In addition, this map representation was further utilized to extend the results to work effectively with low-resolution sensors. We conducted experiments using both real-world and synthetic data to validate our approaches. Additionally, we designed our algorithms to operate online on a robotic platform equipped with readily available sensors for consumer use.

9.1 Summary of the Key Contributions

We initially focused on developing a perception platform incorporating version control to enhance the reproducibility and reliability of the data recordings. To achieve this, we rely on widely used tools in modern operating systems, specifically git and docker. These tools are crucial for constructing complex robots or sensing platforms. By employing them, we have successfully created a containerized software stack that can operate on various host machines without requiring the host to be in a specific state. We view this work as a significant advancement in developing more reliable perception platforms for mobile robots, which, in turn, can accelerate research on new algorithms for autonomous vehicles.

The second problem we addressed is pose estimation using sensor data. We developed a simple yet highly effective approach to LiDAR odometry. Our approach operates exclusively on point clouds, eliminating the need for an IMU or other external odometry sources. By leveraging the classical point-to-point ICP, we created a versatile odometry system that can be used in various challenging environments, including highway runs, cars, handheld devices, segways, and drones. Furthermore, our system is compatible with different range-sensing technologies and scanning patterns. We implemented and evaluated our approach on various datasets, providing comparisons with other existing techniques. The experiments demonstrate that our system is on par with substantially more complex state-of-the-art LiDAR odometry systems. It achieves this by relying on only a few parameters and performing well on different datasets under various conditions using the same parameter set. Our system operates faster than the sensor frame rate in all the tested datasets. This work establishes a new baseline for future sensor odometry systems, offering a robust and high-performance starting point for future approaches and pushing the boundaries of state-of-the-art by challenging even the most sophisticated systems available.

The third problem we tackled involved the construction of high-resolution models of the environment. We introduced a novel approach for offline creation of 3D maps using LiDAR data. Our method represents the map as a triangle mesh, estimated using Poisson surface reconstruction within a sliding window that considers past scans. Our approach operates independently from other odometry sources while incorporating a novel frame-to-mesh registration method. The resulting local meshes exhibit higher detail levels than conventional methods like TSDF or surfel representations. Additionally, we demonstrate that our map representation is well suited for incremental scan registration, facilitating accurate pose estimation. Although our mapping system does not operate in real-time, it can be employed for other robotics tasks such as localization or place recognition. Furthermore, we explored how the high level of detail in our triangle mesh maps

can contribute to improved localization accuracy.

The fourth problem we addressed was building online 3D maps for robot operation. Our proposed approach utilizes a TSDF representation and leverages a sparse and efficient data structure from the filmmaking industry called VDB. We outlined a practical implementation that offers a user-friendly mapping framework suitable for various 3D sensing modalities, including RGB-D and LiDAR. Our experiments assessing runtime and memory efficiency demonstrate that our implementation surpasses other open-source mapping frameworks in terms of efficiency while still delivering high-quality maps. Through mapping accuracy evaluations, we established that our approach outperforms other TSDF-based pipelines. Additionally, we conducted a user study to validate our claim of ease of use. Our open-source library is a solid starting point for further research in TSDF-based mapping, opening doors for future advancements. Furthermore, our system can be practically deployed to construct high-resolution mapping systems for mobile robots. The high execution speed of the integration pipeline, combined with low memory requirements, enables the deployment of such systems in real-world scenarios.

Finally, we targeted the challenge of extending our online mapping pipeline to operate effectively with low-resolution LiDAR sensors. To tackle this, we proposed a novel mapping approach focusing on building dense 3D models from sparse LiDAR data. We explored the concept of sparse 3D geometric scan completion using a learning-based approach. By combining traditional TSDF-based volumetric mapping with 3D convolutional neural networks, we facilitate reconstruction on a frame-to-frame basis. This allows us to generate denser TSDF surface models from a single sparse scan. The completion network is trained in a fully self-supervised manner. Our experiments showed that the maps generated using 16 LiDAR beams using our approach are comparable to or even better than traditionally built TSDF maps generated using 64 LiDAR beams.

In summary, we have presented solutions to several significant problems within the field of robot mapping, from how to capture data from a robotics platform and compute the precise location and orientation of the robot within a given environment to produce different types of map representation that can be used for various robotics tasks. This thesis advances state-of-the-art robot mapping significantly, enabling a mapping pipeline with 3D LiDARs that spawn from data collection and pose estimation to map building. Building on top of our contributions, one can build a perception platform equipped with 3D LiDARs and operate this platform effectively. At the same time, simultaneously estimate the pose of the sensors and create a map of the world as the robot navigates through the environment. Furthermore, thanks to the easy-to-use open-source software we release as part of this thesis, even non-roboticists can record data and

build 3D maps in simple steps. Through experiments on real-world and synthetic datasets, we have shown the effectiveness of our methods, often exceeding the current state of the art. While it is essential to acknowledge that the challenges we addressed are not exhaustive in the context of this thesis topic, the algorithms we have proposed are fundamental components of a comprehensive autonomous robot. These algorithms can be seamlessly integrated into the complete system pipeline, contributing to developing a fully functional and autonomous system. All the software and tools we utilized in developing our methods are open-source.

9.2 Open Source Contributions

We strongly believe that besides sharing scientific findings in papers, sharing code is vital to advance the state of the art effectively, thus, this thesis resulted in the release of several open-source packages and datasets. Additionally, parts of the modules developed for the methods and the experiments were made open source not as stand-alone packages but as contributions to existing open-source libraries, such as CARLA, NVdiffrast, ONXX-tensorrt, Open3D, Open3D-ML, OpenVDB, PyMeshFix, Sophus, and others. Some highlighted contributions are:

- **KISS-ICP** C++, Python, and ROS 1/2 package, presented in Chapter 4: `https://github.com/PRBonn/kiss-icp`
- **VDBFusion** C++ and Python library, presented in Chapter 7: `https://github.com/PRBonn/vdbfusion`
- **VDBFusion** ROS 1 package, presented in Chapter 7: `https://github.com/PRBonn/vdbfusion_ros`
- **make-it-dense** scene completion network, presented in Chapter 8: `https://github.com/PRBonn/make_it_dense`
- **PUMA** 3D offline mapping pipeline, presented in Chapter 5: `https://github.com/PRBonn/puma`
- **range-mcl** Range image localization code, presented in Chapter 6 `https://github.com/PRBonn/range-mcl`
- **Mai City** Synthetic Dataset, presented in Chapter 5: `https://www.ipb.uni-bonn.de/data/mai-city-dataset/`
- **lidar_visualizer**, Visualization tool for LiDAR data: `https://github.com/PRBonn/lidar-visualizer`

- **vdb_to_numpy**, Python bindings to work with VDBs: `https://github.com/PRBonn/vdb_to_numpy`
- **voxblox_pybind**, Python bindings for the Voxblox library `https://github.com/PRBonn/voxblox_pybind`
- **voxblox_pybind**, Python bindings for the Voxblox library `https://github.com/PRBonn/voxblox_pybind`
- **manifold_python**, Python bindings for the Manifold library `https://github.com/PRBonn/manifold_python`
- **ros_in_docker**, Container library to use with ROS `https://github.com/nachovizzo/ros_in_docker`
- **Open3D** C++, and Python Robust Kernel Library: `http://www.open3d.org/docs/release/tutorial/pipelines/robust_kernels.html`
- **Open3D** C++, and Python Generalized ICP: `https://github.com/isl-org/Open3D/pull/3181`

Bibliography

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva. Point Set Surfaces. In *Proc. of the IEEE Visualization (VIS)*, 2001.

[2] P. Babin, P. Giguere, and F. Pomerleau. Analysis of Robust Functions for Registration Algorithms. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[3] C. Bai, T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao. Faster-LIO: Lightweight Tightly Coupled Lidar-Inertial Odometry Using Parallel Sparse Incremental Voxels. *IEEE Robotics and Automation Letters (RA-L)*, 7(2):4861–4868, 2022.

[4] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, J. Gall, and C. Stachniss. Towards 3D LiDAR-based Semantic Scene Understanding of 3D Point Cloud Sequences: The SemanticKITTI Dataset. *Intl. Journal of Robotics Research (IJRR)*, 40(8–9):959–967, 2021.

[5] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[6] J. Behley and C. Stachniss. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2018.

[7] J. Behley, V. Steinhage, and A.B. Cremers. Efficient Radius Neighbor Search in Three-dimensional Point Clouds. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2015.

[8] M. Bennewitz, C. Stachniss, W. Burgard, and S. Behnke. Metric Localization with Scale-Invariant Visual Features using a Single Perspective Camera. In *European Robotics Symposium 2006*, 2006.

[9] J. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

[10] M. Berger, A.Tagliasacchi, L. Seversky, P. Alliez, G. Guennebaud, J. Levine, A. Sharf, and C. Silva. A Survey of Surface Reconstruction from Point Clouds. *Computer Graphics Forum*, 36(1):301–329, 2017.

[11] P. Besl and N. McKay. A Method for Registration of 3D Shapes. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 14(2):239–256, 1992.

[12] M.G. Besselmann, L. Puck, L. Steffen, A. Roennau, and R. Dillmann. VDB-Mapping: A High Resolution and Real-Time Capable 3D Mapping Framework for Versatile Mobile Robots. In *Proc. of the International Conf. on Automation Science and Engineering (CASE)*, 2021.

[13] J.L. Blanco-Claraco. Mobile Robot Programming Toolkit (MRPT). URL: http://www. mrpt. org/, 2014.

[14] C. Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[15] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 120:122–125, 2000.

[16] G. Brostow, J. Fauqueur, and R. Cipolla. Semantic Object Classes in Video: A High-Definition Ground Truth Database. *Pattern Recognition Letters*, 30(2):88–97, 2008.

[17] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions. In *Proc. of Robotics: Science and Systems (RSS)*, 2013.

[18] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. *IEEE Trans. on Robotics (TRO)*, 32(6):1309–1332, 2016.

[19] H. Caesar, V. Bankiti, A. Lang, S. Vora, V. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A multimodal dataset for autonomous driving. *arXiv preprint*, arXiv:1903.11027, 2019.

[20] H. Caesar, V. Bankiti, A. Lang, S. Vora, V. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[21] D. Canelhas, T. Stoyanov, and A. Lilienthal. SDF Tracker: A Parallel Algorithm for On-Line Pose Estimation and Scene Reconstruction from Depth Images. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[22] D. Canelhas, T. Stoyanov, and A. Lilienthal. From Feature Detection in Truncated Signed Distance Fields to Sparse Stable Scene Graphs. *IEEE Robotics and Automation Letters (RA-L)*, 1(2):1148–1155, 2016.

[23] N. Carlevaris-Bianco, A. Ushani, and R. Eustice. University of Michigan North Campus long-term vision and lidar dataset. *Intl. Journal of Robotics Research (IJRR)*, 35(9):1023–1035, 2016.

[24] J. Carr, R. Beatson, J. Cherrie, T. Mitchell, W. Fright, B. McCallum, and T. Evans. Reconstruction and Representation of 3D Objects With Radial Basis Functions. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2001.

[25] E. Cervera. Try to Start It! The Challenge of Reusing Code in Robotics Research. *IEEE Robotics and Automation Letters (RA-L)*, 4(1):49–56, 2019.

[26] E. Cervera and A.P. Del Pobil. Roslab: Sharing Ros Code Interactively With Docker and Jupyterlab. *IEEE Robotics and Automation Magazine (RAM)*, 26(3):64–69, 2019.

[27] A. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[28] M. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays. Argoverse: 3D Tracking and Forecasting with Rich Maps. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[29] N. Chebrolu, T. Läbe, O. Vysotska, J. Behley, and C. Stachniss. Adaptive Robust Kernels for Non-Linear Least Squares Problems. *IEEE Robotics and Automation Letters (RA-L)*, 6(2):2240–2247, 2021.

[30] J. Chen, D. Bautembach, and S. Izadi. Scalable Real-Time Volumetric Surface Reconstruction. *ACM Trans. on Graphics*, 32(4):113, 2013.

[31] J. Chen and S. Shen. Improving Octree-Based Occupancy Maps Using Environment Sparsity with Application to Aerial Robot Navigation. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[32] K. Chen, R. Nemiroff, and B.T. Lopez. Direct LiDAR-Inertial Odometry and Mapping: Perceptive and Connective SLAM. *arXiv preprint*, arXiv:2305.01843, 2023.

[33] K. Chen, R. Nemiroff, and B.T. Lopez. Direct lidar-inertial odometry: Lightweight lio with continuous-time motion correction. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2023.

[34] X. Chen, T. Läbe, A. Milioto, T. Röhling, O. Vysotska, A. Haag, J. Behley, and C. Stachniss. OverlapNet: Loop Closing for LiDAR-based SLAM. In *Proc. of Robotics: Science and Systems (RSS)*, 2020.

[35] X. Chen, T. Läbe, L. Nardi, J. Behley, and C. Stachniss. Learning an Overlap-based Observation Model for 3D LiDAR Localization. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[36] X. Chen, S. Li, B. Mersch, L. Wiesmann, J. Gall, J. Behley, and C. Stachniss. Moving Object Segmentation in 3D LiDAR Data: A Learning-based Approach Exploiting Sequential Data. *IEEE Robotics and Automation Letters (RA-L)*, 6(4):6529–6536, 2021.

[37] X. Chen, B. Mersch, L. Nunes, R. Marcuzzi, I. Vizzo, J. Behley, and C. Stachniss. Automatic Labeling to Generate Training Data for Online LiDAR-Based Moving Object Segmentation. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):6107–6114, 2022.

[38] X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley, and C. Stachniss. SuMa++: Efficient LiDAR-based Semantic SLAM. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[39] X. Chen, I. Vizzo, T. Läbe, J. Behley, and C. Stachniss. Range Image-based LiDAR Localization for Autonomous Vehicles. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[40] X. Chen. *LiDAR-Based Semantic Perception for Autonomous Vehicles*. PhD thesis, University of Bonn, 2022.

[41] Y. Chen and G. Medioni. Object Modelling by Registration of Multiple Range Images. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 1991.

[42] Ö. Çiçek, A. Abdulkadir, S.S. Lienkamp, T. Brox, and O. Ronneberger. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. In *Medical Image Computing and Computer-Assisted Intervention*, 2016.

[43] M. Colosi, I. Aloise, T. Guadagnino, D. Schlegel, B. Corte, K. Arras, and G. Grisetti. Plug-And-Play SLAM A Unified SLAM Architecture for Modularity and Ease of Use. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[44] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1996.

[45] A. Dai, A. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[46] A. Dai, C.R. Qi, and M. Niessner. Shape Completion Using 3D-Encoder-Predictor CNNs and Shape Synthesis. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[47] A. Dai, C. Diller, and M. Nießner. SG-NN: Sparse Generative Neural Networks for Self-Supervised Scene Completion of RGB-D Scans. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[48] N. Dalmedico, M.A. Simões Teixeira, H. Barbosa Santos, R.d.C.M. Nogueira, L.V. Ramos de Arruda, F. Neves, D. Rodrigues Pipa, J. Endress Ramos, and A. Schneider de Oliveira. Sliding Window Mapping for Omnidirectional RGB-D Sensors. *Sensors*, 19(23), 2019.

[49] 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 Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[50] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization for Mobile Robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 1999.

[51] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous Localization and Mapping Via Square Root Information Smoothing. *Intl. Journal of Robotics Research (IJRR)*, 25(12):1181–1203, 2006.

[52] P. Dellenbach, J. Deschaud, B. Jacquet, and F. Goulette. CT-ICP Real-Time Elastic LiDAR Odometry with Loop Closure. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[53] J. Deschaud. IMLS-SLAM: scan-to-model matching based on 3D data. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[54] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An Open Urban Driving Simulator. In *Proc. of the Conf. on Robot Learning (CoRL)*, 2017.

[55] J.C. Eidson, M. Fischer, and J. White. Ieee-1588™ standard for a precision clock synchronization protocol for networked measurement and control systems. In *Proc. of the Annual Precise Time and Time Interval Systems and Applications Meeting*, 2002.

[56] H. Fan, H. Su, and L. Guibas. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[57] M. Fehr, F. Furrer, I. Dryanovski, J. Sturm, I. Gilitschenski, R. Siegwart, and C. Lerma. TSDF-Based Change Detection for Consistent Long-Term Dense Reconstruction and Dynamic Object Discovery. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[58] T. Fischer, W. Vollprecht, S. Traversaro, S. Yen, C. Herrero, and M. Milford. A RoboStack Tutorial: Using the Robot Operating System Alongside the Conda and Jupyter Data Science Ecosystems. *IEEE Robotics and Automation Magazine (RAM)*, 29(2):65–74, 2021.

[59] W. Förstner and B. Wrobel. *Photogrammetric Computer Vision – Statistics, Geometry, Orientation and Reconstruction.* Springer Verlag, 2016.

[60] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 1999.

[61] N. Funk, J. Tarrio, S. Papatheodorou, M. Popović, P.F. Alcantarilla, and S. Leutenegger. Multi-Resolution 3D Mapping With Explicit Free Space Representation for Fast and Accurate Mobile Robot Motion Planning. *IEEE Robotics and Automation Letters (RA-L)*, 6(2):3553–3560, 2021.

[62] M. Gehrig, E. Stumm, T. Hinzmann, and R. Siegwart. Visual Place Recognition with Probabilistic Voting. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[63] 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)*, 2012.

[64] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets Robotics: The KITTI Dataset. *Intl. Journal of Robotics Research (IJRR)*, 32(11):1231–1237, 2013.

[65] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[66] D. Gregorio and L. Stefano. SkiMap: An Efficient Mapping Framework for Robot Navigation. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[67] M. Grinvald, F. Furrer, T. Novkovic, J.J. Chung, C. Cadena, R. Siegwart, and J. Nieto. Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery. *IEEE Robotics and Automation Letters (RA-L)*, 4(3):3037–3044, 2019.

[68] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based SLAM. *IEEE Trans. on Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[69] G. Grisetti, C. Stachniss, and W. Burgard. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Trans. on Robotics (TRO)*, 23(1):34–46, 2007.

[70] G. Grisetti, T. Guadagnino, I. Aloise, M. Colosi, B. Della Corte, and D. Schlegel. Least Squares Optimization: From Theory to Practice. *Robotics*, 9(3), 2020.

[71] T. Guadagnino, X. Chen, M. Sodano, J. Behley, G. Grisetti, and C. Stachniss. Fast Sparse LiDAR Odometry Using Self-Supervised Feature Selection on Intensity Images. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):7597–7604, 2022.

[72] J.S. Gutmann and C. Schlegel. AMOS: comparison of scan matching approaches for self-localization in indoor environments. In *Proc. of the Euromicro Workshop on Advanced Mobile Robots (EUROBOT)*, 1996.

[73] A. Handa, T. Whelan, J. McDonald, and A. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[74] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface Reconstruction from Unorganized Points. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1992.

[75] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[76] X. Huang, G. Mei, J. Zhang, and R. Abbas. A Comprehensive Survey on Point Cloud Registration. *arXiv preprint*, arXiv:2103.02690, 2021.

[77] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang. The ApolloScape Dataset for Autonomous Driving. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops*, 2018.

[78] P.J. Huber. *Robust Statistics*. Wiley, 1981.

[79] I.A. I.A. Barsan, S. Wang, A. Pokrovsky, and R. Urtasun. Learning to Localize Using a LiDAR Intensity Map. In *Proc. of the Conf. on Robot Learning (CoRL)*, 2018.

[80] J. Jeong, Y. Cho, Y. Shin, H. Roh, and A. Kim. Complex Urban LiDAR Data Set. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[81] J. Jeong, T. Yoon, and J. Park. Towards a Meaningful 3D Map Using a 3D Lidar and a Camera. *Sensors*, 18(8):2571, 2018.

[82] M.W. Jones, J.A. Baerentzen, and M. Sramek. 3D distance fields: A survey of techniques and applications. *IEEE Trans. on Visualization and Computer Graphics*, 12(4):581–599, 2006.

[83] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics (TRO)*, 24(6):1365–1378, 2008.

[84] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson Surface Reconstruction. In *Proc. of the Eurographics Symposium on Geometry Processing*, 2006.

[85] M. Kazhdan and H. Hoppe. Screened Poisson Surface Reconstruction. *ACM Trans. on Graphics*, 32(3):1–13, 2013.

[86] M. Kazhdan, M. Chuang, S. Rusinkiewicz, and H. Hoppe. Poisson surface reconstruction with envelope constraints. In *Computer Graphics Forum*, 2020.

[87] M. Keller, D. Lefloch, M. Lambers, and S. Izadi. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2013.

[88] G. Kim, Y. Park, Y. Cho, J. Jeong, and A. Kim. Mulran: Multimodal Range Dataset for Urban Place Recognition. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[89] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2015.

[90] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao. Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device Using Spatially Hashed Signed Distance Fields. In *Proc. of Robotics: Science and Systems (RSS)*, 2015.

[91] A. Knapitsch, J. Park, Q. Zhou, and V. Koltun. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *ACM Trans. on Graphics*, 36(4):1–13, 2017.

[92] R. Kolluri. Provably Good Moving Least Squares. *ACM Transactions on Algorithms (TALG)*, 4(2):18, 2008.

[93] A. Koubâa et al. *Robot Operating System (ROS)*, volume 1. Springer, 2017.

[94] T. Kühner and J. Kümmerle. Large-Scale Volumetric Scene Reconstruction using LiDAR. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

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

[96] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. Autonomous Robot Navigation in Highly Populated Pedestrian Zones. *Journal of Field Robotics (JFR)*, 34(4):565–589, 2014.

[97] J. Levinson, M. Montemerlo, and S. Thrun. Map-Based Precision Vehicle Localization in Urban Environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2007.

[98] J. Lin and F. Zhang. Loam_livox A Robust LiDAR Odemetry and Mapping LOAM Package for Livox LiDAR. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[99] J. Lin and F. Zhang. R3LIVE: A Robust, Real-time, RGB-colored, LiDAR-Inertial-Visual tightly-coupled state Estimation and mapping package. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[100] J. Loeliger and M. McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.", 2012.

[101] W. Lorensen and H. Cline. Marching Cubes: a High Resolution 3D Surface Construction Algorithm. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1987.

[102] F. Lu, G. Chen, Y. Liu, L. Zhang, S. Qu, S. Liu, and R. Gu. Hregnet: A Hierarchical Network for Large-Scale Outdoor Lidar Point Cloud Registration. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[103] F. Lu and Milios. Robot pose estimation in unknown environments by matching 2D range scans. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1994.

[104] F. Lu and E. Milios. Optimal global pose estimation for consistent sensor data registration. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 1995.

[105] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems (JIRS)*, 18:249–275, 1997.

[106] W. Lu, Y. Zhou, G. Wan, S. Hou, and S. Song. L3-Net: Towards Learning Based Lidar Localization for Autonomous Driving. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[107] W. Ma, I. Tartavull, I.A. Bârsan, S. Wang, M. Bai, G. Mattyus, N. Homayounfar, S.K. Lakshmikanth, A. Pokrovsky, and R. Urtasun. Exploiting Sparse Semantic HD Maps for Self-Driving Vehicle Localization. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[108] S. Macenski, D. Tsai, and M. Feinberg. Spatio-Temporal Voxel Layer: A View on Robot Perception for the Dynamic World. *Intl. Journal of Advanced Robotic Systems*, 17(2), 2020.

[109] W. Maddern, G. Pascoe, C. Linegar, and P. Newman. 1 Year, 1000 Km: The Oxford Robotcar Dataset. *Intl. Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017.

[110] F. Magistri, E. Marks, S. Nagulavancha, I. Vizzo, T. Läbe, J. Behley, M. Halstead, C. McCool, and C. Stachniss. Contrastive 3D Shape Completion and Reconstruction for Agricultural Robots using RGB-D Frames. *IEEE Robotics and Automation Letters (RA-L)*, 7(4):10120–10127, 2022.

[111] R. Marcuzzi, L. Nunes, L. Wiesmann, I. Vizzo, J. Behley, and C. Stachniss. Contrastive Instance Association for 4D Panoptic Segmentation for Sequences of 3D LiDAR Scans. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[112] Z. Marton, R. Rusu, and M. Beetz. On Fast Surface Reconstruction Methods for Large and Noisy Point Clouds. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2009.

[113] J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger. Fusion++: Volumetric Object-Level SLAM. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2018.

[114] D. Meagher. Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer. *Technical Report*, Image Processing Laboratory, Rensselaer Polytechnic Institute (IPL-TR-80-111), 1980.

[115] C. Merfels and C. Stachniss. Pose Fusion With Chain Pose Graphs for Automated Driving. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[116] B. Mersch, X. Chen, I. Vizzo, L. Nunes, J. Behley, and C. Stachniss. Receding Moving Object Segmentation in 3D LiDAR Data Using Sparse 4D Convolutions. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):7503–7510, 2022.

[117] B. Mersch, T. Guadagnino, X. Chen, Tiziano, I. Vizzo, J. Behley, and C. Stachniss. Building Volumetric Beliefs for Dynamic Environments Exploiting Map-Based Moving Object Segmentation. *IEEE Robotics and Automation Letters (RA-L)*, 8(8):5180–5187, 2023.

[118] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[119] M. Mihajlovic, S. Weder, M. Pollefeys, and M.R. Oswald. Deepsurfels: Learning Online Appearance Fusion. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[120] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[121] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena. C-Blox: A Scalable and Consistent TSDF-based Dense Mapping Approach. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[122] F. Moosmann and C. Stiller. Velodyne SLAM. In *Proc. of the IEEE Vehicles Symposium (IV)*, 2011.

[123] Z. Murez, T. van As, J. Bartolozzi, A. Sinha, V. Badrinarayanan, and A. Rabinovich. Atlas: End-to-End 3D Scene Reconstruction from Posed Images. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[124] K. Museth. Nanovdb: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation. In *ACM SIGGRAPH 2021 Talks*, 2021.

[125] K. Museth, J. Lait, J. Johanson, J. Budsberg, R. Henderson, M. Alden, P. Cucka, D. Hill, and A. Pearce. OpenVDB: An Open-source Data Structure and Toolkit for High-resolution Volumes. In *ACM SIGGRAPH 2013 courses*, 2013.

[126] 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 Intl. Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[127] J. Niedźwiedzki, P. Lipinski, and L. Podsedkowski. IDTMM: Incremental Direct Triangle Mesh Mapping. *IEEE Robotics and Automation Letters (RA-L)*, 8(9):5416–5423, 2023.

[128] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale using Voxel Hashing. In *Proc. of the SIGGRAPH Asia*, 2013.

[129] N.C. of Argentina. Argentinean Law 26.871. Declaration of Mate as National Infusion, 2013.

[130] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwar, and J. Nieto. Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-Board Mav Planning. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[131] K. O'Shea and R. Nash. An Introduction to Convolutional Neural Networks. *arXiv preprint*, arXiv:1511.08458, 2015.

[132] E. Palazzolo, J. Behley, P. Lottes, P. Giguere, and C. Stachniss. ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[133] Y. Pan, P. Xiao, Y. He, Z. Shao, and Z. Li. MULLS: Versatile LiDAR SLAM Via Multi-Metric Linear Least Square. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[134] J. Park, Q. Zhou, and V. Koltun. Colored Point Cloud Registration Revisited. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2017.

[135] J.J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[136] C.W. Peng, C.C. Hsu, and W.Y. Wang. Cost Effective Mobile Mapping System for Color Point Cloud Reconstruction. *Sensors*, 20(22), 2020.

[137] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger. Convolutional Occupancy Networks. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[138] P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, W. Burgard, and R. Siegwart. Towards Mapping of Cities. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2007.

[139] E. Piazza, A. Romanoni, and M. Matteucci. Real-Time CPU-Based Large-Scale 3D Mesh Reconstruction. *arXiv preprint*, arXiv:1801.05230, 2018.

[140] C.M. Pilato, B. Collins-Sussman, and B.W. Fitzpatrick. *Version control with subversion: next generation open source version control.* " O'Reilly Media, Inc.", 2008.

[141] F. Pomerleau, F. Colas, and R. Siegwart. A Review of Point Cloud Registration Algorithms for Mobile Robotics. *Foundations and Trends in Robotics*, 4:1–104, 2015.

[142] F. Pomerleau, F. Colas, F. Ferland, and F. Michaud. Relative Motion Threshold for Rejection in ICP Registration. In *Field and Service Robotics*, 2010.

[143] M. Popović, F. Thomas, S. Papatheodorou, N. Funk, T. Vidal-Calleja, and S. Leutenegger. Volumetric Occupancy Mapping With Probabilistic Depth Completion for Robotic Navigation. *IEEE Robotics and Automation Letters (RA-L)*, 6(3):5072–5079, 2021.

[144] M. Ramezani, Y. Wang, M. Camurri, D. Wisth, M. Mattamala, and M. Fallon. The Newer College Dataset: Handheld Lidar, Inertial and Vision With Ground Truth. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[145] C. Raposo and J. Barreto. Using 2 Point+Normal Sets for Fast Registration of Point Clouds With Small Overlap. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[146] V. Reijgwart, A. Millane, H. Oleynikova, R. Siegwart, C. Cadena, and J. Nieto. Voxgraph: Globally Consistent, Volumetric Mapping Using Signed Distance Function Submaps. *IEEE Robotics and Automation Letters (RA-L)*, 5(1):227–234, 2019.

[147] G. Riegler, A.O. Ulusoy, H. Bischof, and A. Geiger. OctNetFusion: Learning Depth Fusion from Data. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[148] C. Rist, D. Emmerichs, M. Enzweiler, and D. Gavrila. Semantic Scene Completion Using Local Deep Implicit Functions on Lidar Data. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(10):7205–7218, 2021.

[149] T. Röhling, J. Mack, and D. Schulz. A Fast Histogram-Based Similarity Measure for Detecting Loop Closures in 3-D LIDAR Data. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[150] L. Roldão, R. de Charette, and A. Verroust-Blondet. LMSCNet: Lightweight Multiscale 3D Semantic Completion. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2020.

[151] O. Ronneberger, P.Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Proc. of the Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[152] A. Rosinol, T. Sattler, M. Pollefeys, and L. Carlone. Incremental Visual-Inertial 3D Mesh Generation With Structural Regularities. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[153] F. Ruetz, E. Hernández, M. Pfeiffer, H. Oleynikova, M. Cox, T. Lowe, and P. Borges. OVPC Mesh: 3D Free-Space Representation for Local Ground Vehicle Navigation. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[154] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. of Int. Conf. on 3-D Digital Imaging and Modeling*, 2001.

[155] R.B. Rusu and S. Cousins. 3D Is Here: Point Cloud Library (Pcl). In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2011.

[156] J. Saarinen, T. Stoyanov, H. Andreasson, and A. Lilienthal. Fast 3D Mapping in Highly Dynamic Environments Using Normal Distributions Transform Occupancy Maps. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[157] R.F. Salas-Moreno, B. Glocker, P.H.J. Kelly, and A.J. Davison. Dense Planar SLAM. In *Proc. of the Intl. Symposium on Mixed and Augmented Reality (ISMAR)*, 2014.

[158] V. Sarode, X. Li, H. Goforth, Y. Aoki, R.A. Srivatsan, S. Lucey, and H. Choset. PCRNet: Point Cloud Registration Network using PointNet Encoding. *arXiv preprint*, arXiv:1908.07906, 2019.

[159] A. Schaefer, D. Büscher, J. Vertens, L. Luft, and W. Burgard. Long-term urban vehicle localization using pole landmarks extracted from 3-D lidar scans. In *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2019.

[160] D. Schlegel, M. Colosi, and G. Grisetti. ProSLAM: Graph SLAM from a Programmer's Perspective. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[161] L. Schmid, J. Delmerico, J. Schönberger, J. Nieto, M. Pollefeys, R. Siegwart, and C. Cadena. Panoptic Multi-Tsdfs: a Flexible Representation for Online Multi-Resolution Volumetric Mapping and Long-Term Dynamic Scene Consistency. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

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

[163] J. Serafin and G. Grisetti. NICP: Dense Normal Based Point Cloud Registration. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[164] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus. LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[165] T. Shan and B. Englot. LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[166] C. Shen, J. O'Brien, and J. Shewchuk. Interpolating and Approximating Implicit Surfaces from Polygon Soup. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2004.

[167] S. Song, F. Yu, A. Zeng, A. Chang, M. Savva, and T. Funkhouser. Semantic Scene Completion from a Single Depth Image. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[168] C. Stachniss. *Robotic Mapping and Exploration*. Springer Verlag, 2009.

[169] C. Stachniss and W. Burgard. Mapping and Exploration with Mobile Robots using Coverage Maps. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2003.

[170] C. Stachniss, G. Grisetti, and W. Burgard. Information Gain-based Exploration Using Rao-Blackwellized Particle Filters. In *Proc. of Robotics: Science and Systems (RSS)*, 2005.

[171] C. Stachniss, J. Leonard, and S. Thrun. *Springer Handbook of Robotics, 2nd edition*, chapter Chapt. 46: Simultaneous Localization and Mapping. Springer Verlag, 2016.

[172] B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard. Place Recognition in 3D Scans Using a Combination of Bag of Words and Point Feature Based Relative Pose Estimation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[173] F. Steinbrucker, C. Kerl, and D. Cremers. Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2013.

[174] F. Steinbrücker, J. Sturm, and D. Cremers. Volumetric 3D Mapping in Real-Time on a CPU. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[175] J. Stückler and S. Behnke. Multi-Resolution Surfel Maps for Efficient Dense 3D Modeling and Tracking. *Journal of Visual Communication and Image Representation (JVCIR)*, 25(1):137–147, 2014.

[176] 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 Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[177] J. Sun, Y. Xie, L. Chen, X. Zhou, and H. Bao. NeuralRecon: Real-Time Coherent 3D Reconstruction From Monocular Video. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[178] L. Sun, D. Adolfsson, M. Magnusson, H. Andreasson, I. Posner, and T. Duckett. Localising Faster: Efficient and precise lidar-based robot localisation in large-scale environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[179] M. Tanner, P. Pinies, L. Paz, and P. Newman. What Lies Behind: Recovering Hidden Shape in Dense Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[180] H. Thomas, C. Qi, J. Deschaud, B. Marcotegui, F. Goulette, and L. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[181] S. Thrun, W. Burgard, and D. Fox. A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2000.

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

[183] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, 128(1-2), 2001.

[184] G. Tinchev, A. Penate-Sanchez, and M. Fallon. Learning to see the wood for the trees: Deep laser localization in urban and natural environments on a CPU. *IEEE Robotics and Automation Letters (RA-L)*, 4(2):1327–1334, 2019.

[185] P. Triantafyllou, R. Afonso Rodrigues, S. Chaikunsaeng, D. Almeida, G. Deacon, J. Konstantinova, and G. Cotugno. A Methodology for Approaching the Integration of Complex Robotics Systems: Illustration Through a Bimanual Manipulation Case Study. *IEEE Robotics and Automation Magazine (RAM)*, 28(2):88–100, 2021.

[186] R. Triebel, P. Pfaff, and W. Burgard. Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[187] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. Kelly, and S. Leutenegger. Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping. *IEEE Robotics and Automation Letters (RA-L)*, 3(2):1144–1151, 2018.

[188] I. Vizzo, X. Chen, N. Chebrolu, J. Behley, and C. Stachniss. Poisson Surface Reconstruction for LiDAR Odometry and Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[189] I. Vizzo, B. Mersch, R. Marcuzzi, L. Wiesmann, , J. Behley, and C. Stachniss. Make It Dense: Self-Supervised Geometric Scan Completion of Sparse 3D Lidar Scans in Large Outdoor Environments. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):8534–8541, 2022.

[190] I. Vizzo, T. Guadagnino, J. Behley, and C. Stachniss. VDBFusion: Flexible and Efficient TSDF Integration of Range Sensor Data. *Sensors*, 22(3):1296, 2022.

[191] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss. KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1029–1036, 2023.

[192] I. Vizzo, B. Mersch, L. Nunes, L. Wiesmann, T. Guadagnino, and C. Stachniss. Toward Reproducible Version-Controlled Perception Platforms: Embracing Simplicity in Autonomous Vehicle Dataset Acquisition. In *Worshop on Building Reliable Ratasets for Autonomous Vehicles, IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*, 2023.

[193] I. Wald, S. Woop, C. Benthin, G. Johnson, and M. Ernst. Embree: a Kernel Framework for Efficient CPU Ray Tracing. *ACM Trans. on Graphics*, 33(4):1–8, 2014.

[194] H. Wang, C. Wang, C. Chen, and L. Xie. F-LOAM: Fast LiDAR Odometry and Mapping. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2021.

[195] L. Wang, H. Ye, Q. Wang, Y. Gao, C. Xu, and F. Gao. Learning-Based 3D Occupancy Prediction for Autonomous Navigation in Occluded Environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2021.

[196] R. Wang, M. Schwörer, and D. Cremers. Stereo DSO: Large-Scale Direct Sparse Visual Odometry with Stereo Cameras. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2017.

[197] Y. Wang, N. Funk, M. Ramezani, S. Papatheodorou, M. Popovic, M. Camurri, S. Leutenegger, and M. Fallon. Elastic and Efficient LiDAR Reconstruction for Large-Scale Exploration Tasks. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[198] Z. Wang, L. Zhang, Y. Shen, and Y. Zhou. D-LIOM: Tightly-coupled Direct LiDAR-Inertial Odometry and Mapping. *IEEE Trans. on Muultimedia*, pages 1–1, 2022.

[199] S. Weder, J.L. Schonberger, M. Pollefeys, and M.R. Oswald. Neuralfusion: Online Depth Fusion in Latent Space. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[200] X. Wei, I.A. Bârsan, S. Wang, J. Martinez, and R. Urtasun. Learning to Localize Through Compressed Binary Maps. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[201] A. Wendt and T. Schüppstuhl. Proxying ROS communications–enabling containerized ROS deployments in distributed multi-host environments. In *Proc. of the Intl. Symp. on System Integration (SII)*, 2022.

[202] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially Extended KinectFusion. In *Proc. RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[203] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J.J. Leonard, and J. McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. Journal of Robotics Research (IJRR)*, 34(4-5):598–626, 2014.

[204] T. Whelan, S. Leutenegger, R.S. Moreno, B. Glocker, and A. Davison. ElasticFusion: Dense SLAM Without A Pose Graph. In *Proc. of Robotics: Science and Systems (RSS)*, 2015.

[205] L. Wiesmann, T. Guadagnino, I. Vizzo, G. Grisetti, J. Behley, and C. Stachniss. DCPCR: Deep Compressed Point Cloud Registration in Large-Scale Outdoor Environments. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):6327–6334, 2022.

[206] L. Wiesmann, A. Milioto, X. Chen, C. Stachniss, and J. Behley. Deep Compression for Dense Point Cloud Maps. *IEEE Robotics and Automation Letters (RA-L)*, 6(2):2060–2067, 2021.

[207] L. Wiesmann, T. Guadagnino, I. Vizzo, N. Zimmerman, Y. Pan, H. Kuang, J. Behley, and C. Stachniss. LocNDF: Neural Distance Field Mapping for Robot Localization. *IEEE Robotics and Automation Letters (RA-L)*, 8(8):4999–5006, 2023.

[208] D. Wilbers, C. Merfels, and C. Stachniss. Localization with Sliding Window Factor Graphs on Third-Party Maps for Automated Driving. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[209] R. Wolcott and R. Eustice. Fast lidar localization using multiresolution gaussian mixture maps. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2015.

[210] W. Wu, X. Zhong, D. Wu, B. Chen, X. Zhong, and Q. Liu. LIO-Fusion: Reinforced LiDAR Inertial Odometry by Effective Fusion With GNSS/Relocalization and Wheel Odometry. *IEEE Robotics and Automation Letters (RA-L)*, 8(3):1571–1578, 2023.

[211] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems. In *Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation, IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[212] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang. FAST-LIO2: Fast Direct LiDAR-Inertial Odometry. *IEEE Trans. on Robotics (TRO)*, 38(4):2053–2073, 2022.

[213] F. Yan, O. Vysotska, and C. Stachniss. Global Localization on OpenStreetMap Using 4-bit Semantic Descriptors. In *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2019.

[214] H. Ye, Y. Chen, and M. Liu. Tightly Coupled 3D Lidar Inertial Odometry and Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[215] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[216] M. Zeng, F. Zhao, J. Zheng, and X. Liu. Octree-Based Fusion for Realtime 3D Reconstruction. *Graphical Models*, 75(3):126–136, 2013.

[217] A. Zhang, Z.C. Lipton, M. Li, and A.J. Smola. Dive into Deep Learning. *arXiv preprint*, arXiv:2106.11342, 2021.

[218] C. Zhang, M.H. Ang, and D. Rus. Robust lidar localization for autonomous driving in rain. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[219] J. Zhang and S. Singh. LOAM: Lidar Odometry and Mapping in Real-time. In *Proc. of Robotics: Science and Systems (RSS)*, 2014.

[220] C. Zheng, Q. Zhu, W. Xu, X. Liu, Q. Guo, and F. Zhang. FAST-LIVO: Fast and Tightly-coupled Sparse-Direct LiDAR-Inertial-Visual Odometry. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2022.

[221] Q. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv preprint*, arXiv:1801.09847, 2018.

[222] Y. Zhou, G. Wan, S. Hou, L. Yu, G. Wang, X. Rui, and S. Song. DA4AD: End-to-End Deep Attention-Based Visual Localization for Autonomous Driving. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

List of Figures

List of Tables