UNIVERSITÄT BONN

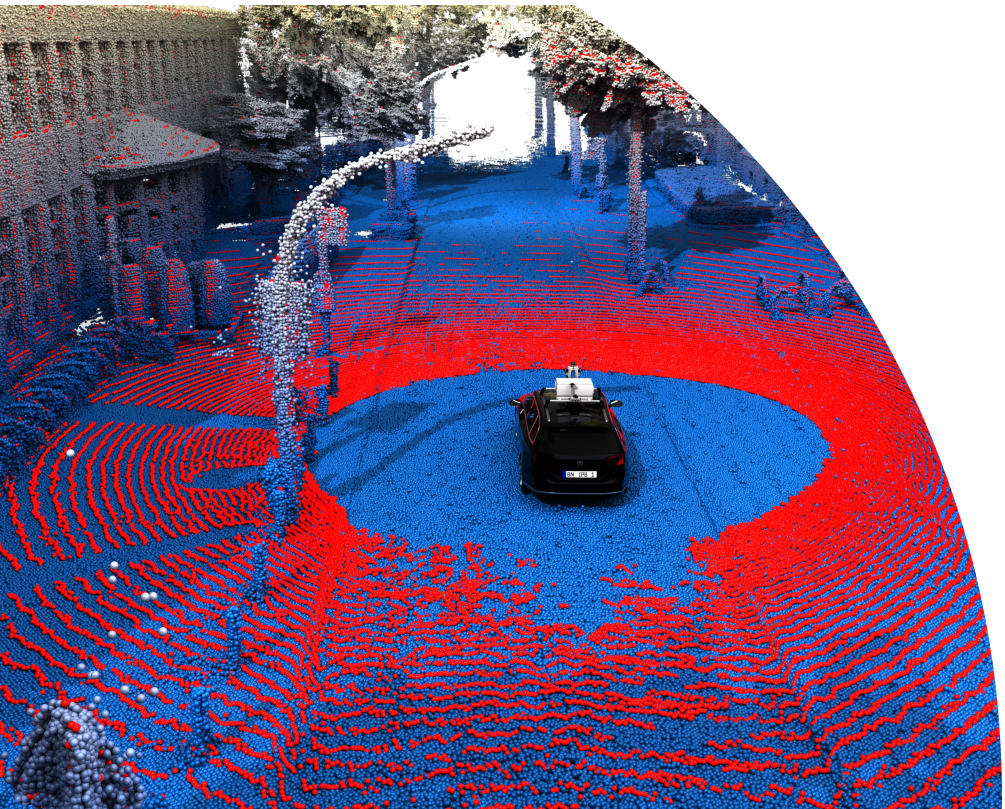
Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
Agrar-, Ernährungs- und Ingenieurwissenschaftliche Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn
Institut für Geodäsie und Geoinformation

Efficient LiDAR-Based Mapping and Localization in Outdoor Environments

von

Louis Wiesmann

aus
Münster, Deutschland



Bonn 2025

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

1. Korreferent:
PD Dr. rer. nat. Lasse Klingbeil, University of Bonn, Germany

2. Korreferent:
Prof. Dr. Abhinav Valada, University of Freiburg, Germany

Tag der mündlichen Prüfung: 02.07.2025

Erscheinungsjahr: 2025

Angefertigt mit Genehmigung der Agrar-, Ernährungs- und Ingenieurwissenschaftlichen Fakultät der Universität Bonn

Zusammenfassung

AUTONOM operierende Roboter bieten das Potenzial Menschen in ihrem täglichen Leben zu unterstützen. Roboter können Aufgaben erledigen, die man selbst nicht lösen kann, oder einfach nicht lösen möchte. Serviceroboter, selbstfahrende Taxis, Staubsaugerroboter oder autonome Rasenmäher werden wahrscheinlich für viele immer alltäglicher. Gleichzeitig, schätzt die Industrie die Präzision, Effizienz und Effektivität von Robotern in ihren Produktionslinien. Damit mobile Roboter sicher und zuverlässig operieren können, sind sie oft auf eine Karte der Einsatzumgebung angewiesen. Der Roboter muss seine eigene Position in der Karte kennen, um die darin befindlichen Informationen voll auszuschöpfen. Viele Roboter sind mit Sensoren wie Laserscannern ausgestattet, welche die Positionsbestimmung durch Beobachtung der Umgebung ermöglichen. Die Aufgabe, die eigene Position in einer Karte zu finden wird Lokalisierung genannt, und ist ein geläufiges Problem in der Robotik. Etablierte Lokalisierungsmethoden liefern zuverlässige Ergebnisse vor allem in kleineren, räumlich stark begrenzten Gebieten. Mit der Entwicklung von komplexeren Systemen, wie zum Beispiel autonom fahrenden Autos, bleibt die Frage offen, ob diese Techniken skalierbar sind und dass bei fortwährend hoher Genauigkeit.

In dieser Dissertation fokussieren wir uns auf Karten mit Größen, wie sie in der automobilen Welt anzutreffen sind. Durch die schiere Größe von dreidimensionalen Karten benötigen wir Methoden zur effizienten Repräsentation um diese praktisch nutzen zu können. Solche 3D-Karten müssen effizient speicherbar sein, aber sie müssen auch eine Repräsentation haben, in der sich Roboter lokalisieren können. Obwohl sich viele vorherige Arbeiten entweder mit dem Forschungsfeld Datenkomprimierung oder mit dem Forschungsfeld der Lokaliserung beschäftigt haben, ist die Schnittstelle weitestgehend unerforscht. Im Speziellen liegt eine Forschungslücke im Bereich der speicher-effizienten Karten mit dem Ziel der Lokalisierung vor.

Wir präsentieren mehrere Algorithmen zur Kartierung und Lokalisierung in größeren Außengebieten. Unsere erste Methode stellt sich der fundamentalen Aufgabe der Sensorkalibrierung von robotischen Systemen, welche notwendig ist, um

konsistente und unverzerrte Beobachtungen der Umgebung aufzunehmen. Dies ist notwendig damit die folgenden Methoden zuverlässige Ergebnisse liefern. Anschließend beschreiben wir unsere Methode, um globale und konsistente dreidimensionale Punktwolken aus Laserscannerdaten zu generieren. Unsere nachfolgende Methode beschäftigt sich mit der Komprimierung solcher Daten mithilfe von maschinellem Lernen. Die restliche Dissertation beschäftigt sich mit der Lokalisierung in solchen Karten. Wann immer man später durch die kartierte Umgebung fährt, wollen wir unsere Position in der komprimierten Karte finden, ohne auf GNSS angewiesen zu sein. Dafür haben wir eine Methode entwickelt, die erst grob abschätzt, in welchem Bereich man sich in der Karte befindet, um dann in einem zweiten Schritt die Position Zentimeter genau zu bestimmen. Nachdem wir unsere Position gefunden haben, können wir unsere Methode zur inkrementellen Schätzung der Bewegung relativ zur Karte verwenden. Zu wissen, wo sich der Roboter zu jedem Zeitpunkt befindet, ermöglicht, die beobachteten Daten mit den Daten in der Karte abzugleichen oder auch zu kombinieren. Außerdem können unsere vorgestellten Methoden darauf aufbauende Aufgaben ermöglichen, wie zum Beispiel Routenplanung oder Navigation.

In dieser Dissertation stellen wir neue wissenschaftliche Methoden zur dreidimensionalen Kartierung und Lokalisierung vor. Wir haben unsere Methoden auf öffentlichen Datensätzen evaluiert und in begutachteten Fachzeitschriften und Konferenzen publiziert. Die Implementierungen und Software der hier vorgestellten Methoden sind öffentlich zugänglich gemacht worden, um als Basis für zukünftige Wissenschaft zur Verfügung zu stehen.

Abstract

AUTONOMOUS robotic systems can help people in their daily lives. Robots can do tasks people cannot or simply do not want to do. Service robots like autonomous vacuum cleaner, lawn mower or even autonomous driving taxis are likely becoming an integral part of our lives. Meanwhile, the industry values the precision, efficiency, and capacity of robots to support their production lines. For reliable and safe operation, mobile robots usually rely on some sort of map of the target environment. For that, a robot needs to know its location within the map to utilize the information that is stored in there to the full extent. Many robots are equipped with sensors, like laser scanners, which can be used to figure out their location within the map, based on the current sensor observations. This task of finding the own position within a given map is usually called localization, and is a well studied problem in robotics. Established methods have shown great success, especially in smaller, mostly indoor environments. However, with the rise of advanced systems, like autonomous driving cars, it remains an open question if those techniques are scalable to such an extent at high precision.

In this thesis, we focus on larger-scale maps in outdoor environments such as those encountered in the automotive domain. Due to the sheer size of 3D maps, we have to develop techniques for efficiently representing the data. The resulting map needs to be compact in memory, but also usable for the localization of the robots. Many previous works focus on either compressing the data, or the localization algorithm. However, there is little research tackling both at the same time: trying to build memory efficient maps which are well suited for localization.

To tackle these problems, we propose several algorithms towards city-scale mapping and localization. We start with the fundamental task of calibrating the sensors of robotic platforms to obtain consistent and undistorted data that is needed for all the subsequent tasks. We then describe a method for building consistent point cloud maps using the raw recorded sensor observations. Afterward, we investigate how to utilize machine learning to compress our point cloud maps to be more memory efficient. The remaining thesis focuses on localizing robots in such maps. When navigating at a later point in time through the environment,

we want to find out our position with respect to the compressed map without any further cues such as GNSS. For this, we have developed an algorithm that first coarsely figures out in which part of the map the robot is located. Afterward, we look into a fine registration where we aim at centimeter-accurate localization. Note that both, the coarse localization and the fine registration, operate directly on the compressed representation to enable localization in a compressed map. Once we have found our initial position, we can track the robots' movements within the map using our developed pose-tracking method. By this, we can estimate for any point in time the position of the robot in the map. Knowing the robot's position allows relating and fusing the measurements from the robot with the available information that is stored in the map. Additionally, it enables subsequent tasks, like path planning, which require the robot's position.

In this thesis, we advanced towards mapping large-scale environments and localizing in those resulting maps. The methods proposed here have been evaluated on publicly available datasets and are published in peer-reviewed journals and conferences. The software and implementations of our methods are open-source to enable new research to be built upon our works.

Acknowledgements

Retrospectively, time during my PhD passed incredibly fast. Five and a half years of learning, teaching, questioning, and exploring have changed my life. I am grateful for all the wonderful experiences I have had. While a crucial part of the PhD is driven by self-motivation, acquiring knowledge, and pushing the boundaries of science forward, all this is not possible without the help of others. Here, I want to thank the people who supported me on the path that led to this thesis.

First, and foremost, I want to thank my supervisor Cyrill Stachniss for enabling me this incredible journey. I want to thank him for giving me the opportunity to explore the topics I was interested in, in a way I thought it would be the best. Allowing me to make mistakes, but guiding me whenever I needed help. Even in his most stressful periods, when he did not really have any time, one could always approach him and ask for advice. I also want to thank him for creating this motivating, and inspirational work environment I truly loved working in.

Next, I want to thank Jens Behley, the post-doc who guided me through my PhD. We spent countless hours discussing how we could finally make my methods work. Pointing me always to relevant work, and motivating me to explore the newest techniques and ideas. He might have told me several times: *"In the end, you anyway do whatever you want!"*, but I always valued the comments and advice he gave me.

Alongside Cyrill and Jens, the colleagues I learned the most from are Ignacio Vizzo and Andres Milioto. Andres supervised me back during my masters with the thesis, as well as the master project. In the beginning of my PhD, he cleared out so many misconceptions I had and supported me to find my way into machine learning. Ignacio Vizzo has shown me so many things to program more like a software engineer, rather than the geodetic engineer I am. We went together through good and bad times, and I hope I have supported him just a bit as much as he helped me.

A PhD is a pretty long and intense time. A time I could not have managed without the emotional support and ease that my colleagues gave me. Especially, I want to thank Federico, Rodrigo, Elias, Lucas, Gianmarco, and Matteo, which

are not only colleagues, but foremost my friends. I want to thank them for all the great times we spent together in the lab, the conference trips they made so unforgettable, and the time outside work, where we just relaxed, played games and typically ate a bunch of food.

Besides them, I thank all my other colleagues I had the opportunity to work with and learn from: Igor Bogoslavskyi, Daniel Casado, Nived Chebrolu, Xieyuanli (Rhiney) Chen, Yue (Linn) Chong, Tiziano Guadagnino, Saurabh Gupta, Liren Jin, Haofei Kuang, Luca Lobefaro, Phillip Lottes, Meher Malladi, Benedikt Mersch, Lorenzo Nardi, Emanuele Palazzolo, Yue Pan, Lasse Peters, Timo Röhling, Julius Rückin, Niklas Trekel, Olga Vysotska, Jan Weyler, Matthias Zeller, Xingguang (Starry) Zhong, and Nicky Zimmerman. I am so grateful for the cooperation and the friendly relationships. I want to thank Thomas Läbe, Perrine Aguiar, Birgit Klein and Kirsten Sadler for technical support, knowledge and the fights with bureaucracy.

I want to thank my family for fully supporting all my life decisions. Especially, my parents for their relentless effort to make a good man out of me, as well as supporting early on my academic career without putting any pressure on me. I had a wonderful childhood with their endless love and efforts without this would not have been possible. I thank my siblings for taking over so many family responsibilities.

During my PhD I found my current, and hopefully everlasting love Annika. Finally, I have to thank her for all the support and energy she gave me. For the countless times I came home, my head completely exhausted from work, and still being pleased to see me. As well as for tolerating the stressful times before conferences and paper deadlines. In the intense period of the PhD, which dictates in that time most of your life, the small breaks and timeouts are sometimes the things which truly make you push through and keep going. For this, I have to thank her more than anyone else.

Contents

Acronyms

DoF degree of freedom

FoV field of view

GNSS global navigation satellite system

ICP iterative closest point

IMU inertial measurement unit

LiDAR light detection and ranging

MCL Monte Carlo localization

MLP multi layer perceptron

NDF neural distance field

NeRF neural radiance field

ReLU rectified linear unit

SLAM simultaneous localization and mapping

TLS terrestrial laser scanner

Chapter 1

Introduction

Autonomously operating robots and vehicles are able to support humans in their daily lives. In many industrial areas, robots have become indispensable: manufactures in the automotive domain value the speed and precision robotic systems can offer, warehouses use mobile robots to transport large amounts of goods autonomously, while chemical factories assign robots to tasks that would be dangerous for human operators. Utilization of robotic systems can increase safety, productivity, efficiency, and precision, making it a key factor in modern production lines. However, robots are not only used in industry but also play an increasing role in our personal day-to-day life. Cleaning and lawnmower robots have become common household helpers available at low costs. Letting robots do the work that we do not want to do is appealing to a broad audience. The dream for many car owners that their car can drive autonomously might not be in a too far future. Even though predicting when we are ready for fully autonomous driving is difficult, at least first pilot projects are already running, and partially letting the car drive autonomously is already a reality. Reading a book, watching a show, working on a notebook, or playing with the kids are just a few options people would rather do than the tiring focus on the traffic while driving the car. It is not only a matter of convenience, but also of safety. Driving for a longer time can cause fatigue. Most car accidents are caused by driving errors. Robots do not get tired nor lose focus. Electrical signals propagate information millions of times faster than neurons, potentially allowing robots to react faster than humanly possible.

To achieve autonomy robots need to know what the environment looks like in order to operate safely and reliably. For example, an autonomous car needs to detect other traffic participants and must be aware of its structural surroundings. For this, typically, sensors like cameras, or light detection and ranging (LiDAR) sensors are used. Cameras have the advantage of providing dense photometrical information, but lack range information and thus only obtain 2D information.

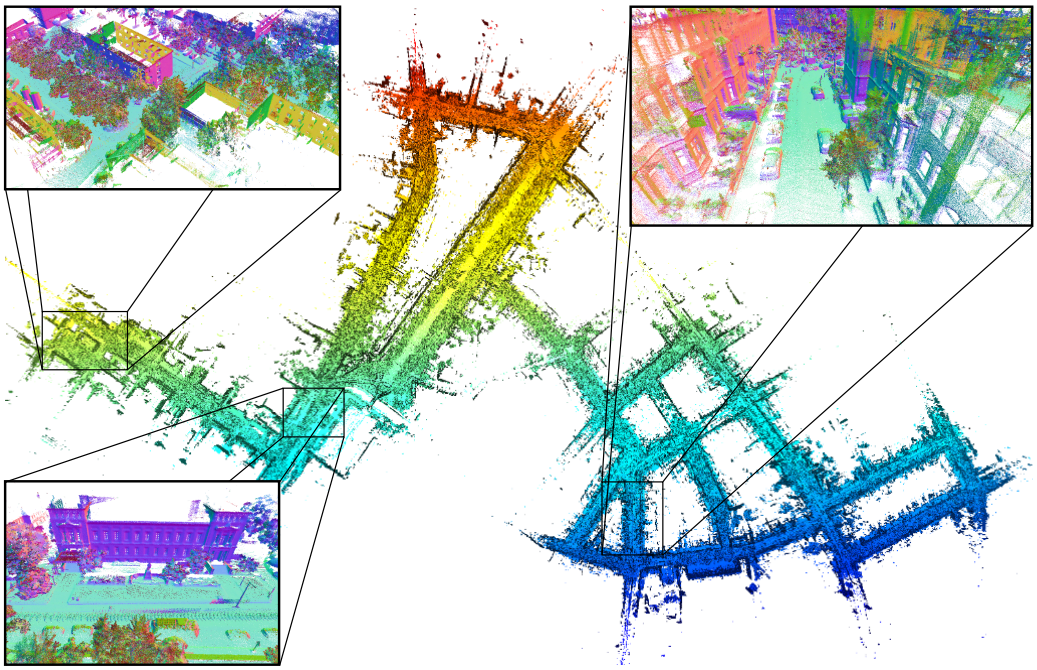


Figure 1.1: A point cloud of the Südstadt quarter in Bonn. In this thesis, we want to investigate how to map outdoor environments like these. Additionally, we want to be able to localize our robots within those maps when driving another time through the same environment. The color of the full point cloud is based on the latitude, while the points of the close-ups are colorized based on normals.

In contrast, LiDAR sensors are able to measure via time-of-flight the distance to the surrounding objects, allowing them to generate a 3D point cloud of the environment. In this work, we utilize mainly LiDAR point clouds as a direct observation of the geometrical structures.

Although knowing the environment around the robot is crucial, for many tasks the robot requires also broader knowledge about the world it should operate. This information or context can be provided by a map. For an autonomous car, a map is required to navigate from one place to another. A lawnmower robot needs to know exactly in which area it has to trim and which parts to avoid. For the robot to utilize the map, it first needs to know its location within this map. One way to localize within a map is to use the sensor observations and compare them with the given map. A main part of this thesis is about how to localize with our observed LiDAR point cloud data in a previously constructed map.

Representing the environment can be done in various ways. Road maps, topographic maps, topological maps, or landmark-based maps, are just a few map representations used by humans, but also robots. In this thesis, we focus on point cloud maps, where the 3D surface of the environment is represented by a set of points with 3D coordinates. An illustration of those point clouds is shown in Figure 1.1. This representation can provide rich geometrical structure and can

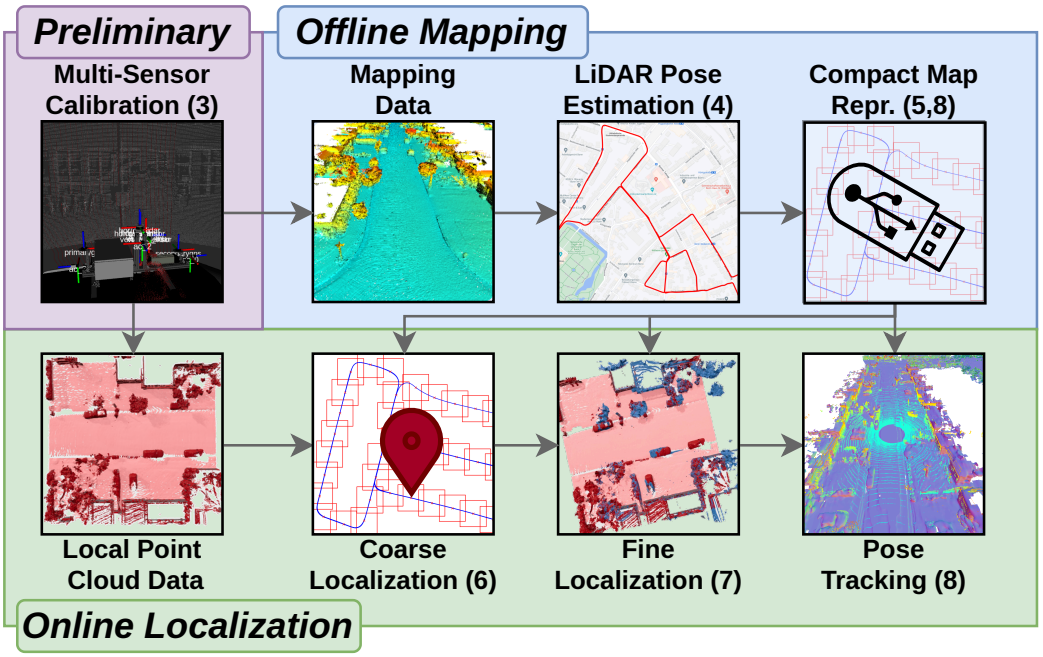


Figure 1.2: Overview of the thesis. The numbers within the brackets denote the chapters in which those topics are discussed. First, we investigate how to calibrate multi-sensor systems. Those can afterward be used to record data, e.g., for mapping, and later for localization. The mapping data needs to be registered to obtain a global map, for this, we propose a method that estimates the poses of a LiDAR to estimate its trajectory. Due to the large memory consumption of those maps, we investigated building compact map representations. Once we have built our map, we can use the local point cloud data for online localization. The first step is to do a coarse localization where we want to roughly know where the robot is located on the map. The following fine localization estimates the precise position and orientation with respect to the map. Once the position is found, one can utilize pose tracking to estimate the movement within the map. The arrows denote from which module information is passed to another method.

be obtained by processing the LiDAR data. Having a point cloud of city-scale sized environments, e.g., for autonomous driving, requires a substantial amount of memory storage and compute power to handle the vast amount of data. Point clouds of that size can often contain billions to trillions of points, making efficient storage techniques and scalable algorithms essential.

In this thesis, we investigate the problem of constructing point cloud maps of outdoor environments and explore methodologies to localize within. A possible workflow could look as follows: (1) Construct a map of the target environment in which robots shall later localize. This requires a mobile mapping system or robot, that can move around and obtain measurements of the scene. After collecting the measurements, a map needs to be constructed from the data the robot recorded. (2) Whenever a robot drives through the mapped scene, estimate the robot's

position with respect to the previously built map, and by this localize the robot. Figuring out where the robot is located can be done by comparing the robot's current sensor data to the constructed map.

In this thesis, we tackle multiple steps along the previously mentioned workflow to enable LiDAR-based localization and mapping in outdoor environments. An overview is given in Figure 1.2. First of all, we need to have an operating mobile mapping system that needs to be properly calibrated to obtain consistent and undistorted data about the environment. For this, we propose a method for calibrating perception sensors in Chapter 3. For the map construction, the recorded data needs to be geo-referenced, which we address in Chapter 4. Due to the sheer size of the resulting 3D point cloud maps, we investigate in Chapter 5 a compressed map representation for efficient storage. Afterward, we propose certain methods for localizing a robot in such compressed point cloud maps in a hierarchical manner. In Chapter 6, we first try to coarsely localize our robot in the compressed map using place recognition. The second fine localization step estimates finally the position and orientation of the robot within the compressed map as described in Chapter 7. Both localization methods directly operate on the compressed point cloud representation and do *not* need to decompress the map which is essential for large-scale operation. For tracking the movement of the robots in a given map, we investigated the usage of neural implicit fields as an alternative memory-efficient map representation in Chapter 8 and how they can be constructed. Chapter 9 finally summarizes our achievements and discusses future research directions that can build upon our proposed work.

1.1 Main Contributions

This thesis investigates the problem of localization and mapping of outdoor environments. Starting from calibrating a multi-sensor setup, over constructing a memory-efficient map to finding the robot's location at a different point in time within the map. This section summarizes the main contributions of the individual parts of this thesis.

The first contribution targets the calibration of our multi-sensor system in Chapter 3. Having multiple perception sensors like cameras and LiDAR sensors requires extrinsic and intrinsic calibration to jointly use the data. For this, we developed a calibration method that exploits the precise point clouds of a terrestrial laser scanner (TLS) in our own developed calibration environment. By using a fully designed calibration environment as a target for calibrating the sensors, we can show that this is more precise than the most commonly used checkerboard target. We show, that our system can be used to calibrate different multi-sensor

systems with different sensor types and configurations, even when the sensors do have non-overlapping field of view (FoV).

Second, we contribute to LiDAR mapping and pose estimation with our proposed LiDAR bundle adjustment method as described in Chapter 4. Obtaining globally and locally precise aligned point clouds is still a challenge in robotics, but is a crucial task to generate precise maps. Our approach refines the vehicle's trajectory estimated, e.g., by LiDAR odometry, SLAM, or even combined with global navigation satellite system (GNSS) measurements, such that the registered point cloud map is globally well aligned. For this, we jointly register all the point clouds to each other in a global LiDAR bundle adjustment approach. One major focus of the work is to be able to handle the vast amount of LiDAR scans as commonly encountered in the automotive domain. Our main contribution is a system that estimates continuous trajectories for multiple sequences, by jointly aligning all the available LiDAR data.

Our third contribution to the domain of point cloud compression is presented in Chapter 5. Compressing the large point cloud maps has the potential to reduce the memory footprint, which is especially interesting for dense large-scale environments. We propose a fully convolutional autoencoder neural network that produces an intermediate feature-based representation from which the input point cloud can be reconstructed. To recover a dense point cloud from the feature-based representation, we had to develop an upsampling block for the network architecture. Our approach does *not* target lossless compression and belongs to the class of lossy compression methods, therefore allowing for higher compression rates. To the best of our knowledge, this method is the first deep learning-based compression method for dense point clouds, that works for real-world point cloud maps.

In the fourth contribution, we try to localize our vehicle within a preconstructed compressed map using current LiDAR observation as described in Chapter 6. The compressed map within we want to localize is the compact feature-based representation that we previously discussed. For this, we directly operate on the compact features from the compression network to estimate point cloud descriptors which encode for each point cloud the structural information into a single feature vector. Computing the descriptor at multiple locations serves as keys for the database. Comparing the descriptor generated by the current LiDAR data to the keys of the database allows for finding the corresponding area in the map. We show that our approach, which works solely on a compressed representation, can provide competitive results with the current state-of-the-art in place recognition. We contributed to this field by proposing novel neural network architectures, faster training strategies, and investigating the usage of compressed data.

Fifth, we contribute to global point cloud registration in Chapter 7, which allows us to estimate the transformation between two point clouds, regardless of the quality of the initial guess – a key difference to standard methods such as ICP. Our proposed method reinterprets the attention mechanism of transformer networks to allow for fully differentiable point cloud registration. This enables us to learn features that are well-suited for feature-based matching in the registration. Since we build upon our compressed features, we can even register directly the compact representation of two compressed point clouds without the need for decompression. Our main contributions lie in the investigation of registering compressed point clouds, novel neural network architectures, and correspondence weighting schemes, as well as formulating point cloud registration as the transformer-based attention mechanism.

The sixth and last contribution tackles pose-tracking and is described in detail in Chapter 8. Here, we assume to know the initial pose of our vehicle, e.g., estimated with the previous two approaches, and from there on want to track the movement of the robot within the map. To achieve this, we investigated the usage of neural distance fields for the task of point cloud alignment and localization. We focus on generating a memory-efficient representation directly from sensor data and how we can localize a LiDAR in such maps. Our contribution is regarding the construction of those neural distance fields directly from sensor data, as well as showing how established localization methods can utilize this representation.

By the end, we have contributed to many areas for mapping and localization in outdoor environments. Beginning with the sensor-system calibration, over building precise and memory-efficient maps. We have developed multiple methods to localize directly in our proposed compact map representations. First, estimating a coarse location using place recognition, and second estimating the 6 DoF transformation allows for finding the robot's position in the compressed map. Once we have localized our system, we can track our position using our neural implicit representation. We will use the already shown roadmap in Figure 1.2 as guidance through this thesis, which we will pick up later in the chapters to provide a small overview of where we are in the grand scheme of this work. This thesis contributes to robotics, machine learning approaches, and photogrammetry. We exploit techniques from all fields including modern machine learning approaches, as well as traditional photogrammetric methods such as bundle adjustment to achieve top-level performance for mobile robotics.

1.2 Publications

Most parts of this thesis have been published in peer-reviewed conferences and journals:

- 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. DOI: 10.1109/LRA.2021.3059633

- L. Wiesmann, R. Marcuzzi, C. Stachniss, and J. Behley. Retriever: Point Cloud Retrieval in Compressed 3D Maps. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022. DOI: 10.1109/ICRA46639.2022.9811785

- 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. DOI: 10.1109/LRA.2022.3171068

- 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. DOI: 10.1109/LRA.2023.3291274

- L. Wiesmann, L. Nunes, J. Behley, and C. Stachniss. KPPR: Exploiting Momentum Contrast for Point Cloud-Based Place Recognition. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):592–599, 2023. DOI: 10.1109/LRA.2022.3228174

- L. Wiesmann, T. Läbe, L. Nunes, J. Behley, and C. Stachniss. Joint Intrinsic and Extrinsic Calibration of Perception Systems Utilizing a Calibration Environment. *IEEE Robotics and Automation Letters (RA-L)*, 9(10):9103–9110, 2024. DOI: 10.1109/LRA.2024.3457385

- L. Wiesmann, E. Marks, S. Gupta, T. Guadagnino, J. Behley, and C. Stachniss. Efficient LiDAR Bundle Adjustment for Multi-Scan Alignment Utilizing Continuous-Time Trajectories. *arXiv preprint*, arXiv:2412.11760, 2024. DOI: 10.48550/arXiv.2412.11760

1.3 Further Scientific Contributions

In addition to the aforementioned articles, which are covered in this thesis, I furthermore had the opportunity to contribute to a number of research endeavors leading to peer-reviewed conferences and journal articles of my colleagues:

- 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. DOI: 10.1109/LRA.2021.3093567

- 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. DOI: 10.1109/LRA.2022.3140439

- 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. DOI: 10.1109/LRA.2022.3187255

- N. Zimmerman, L. Wiesmann, T. Guadagnino, T. Läbe, J. Behley, and C. Stachniss. Robust Onboard Localization in Changing Environments Exploiting Text Spotting. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2022. DOI: 10.48550/arXiv.2203.12647

- M. Arora, L. Wiesmann, X. Chen, and C. Stachniss. Static Map Generation from 3D LiDAR Point Clouds Exploiting Ground Segmentation. *Journal on Robotics and Autonomous Systems (RAS)*, 159:104287, 2023. DOI: 10.1016/j.robot.2022.104287

- R. Marcuzzi, L. Nunes, L. Wiesmann, J. Behley, and C. Stachniss. Mask-Based Panoptic LiDAR Segmentation for Autonomous Driving. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1141–1148, 2023. DOI: 10.1109/LRA.2023.3236568

- 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. DOI: 10.1109/LRA.2023.3236571

- L. Nunes, L. Wiesmann, R. Marcuzzi, X. Chen, J. Behley, and C. Stachniss. Temporal Consistent 3D LiDAR Representation Learning for Semantic Perception in Autonomous Driving. In *Proc. of the IEEE/CVF*

Conf. on Computer Vision and Pattern Recognition (CVPR), 2023. DOI: 10.1109/CVPR52729.2023.00505

- E. Marks, M. Sodano, F. Magistri, L. Wiesmann, D. Desai, R. Marcuzzi, J. Behley, and C. Stachniss. High Precision Leaf Instance Segmentation in Point Clouds Obtained Under Real Field Conditions. *IEEE Robotics and Automation Letters (RA-L)*, 8(8):4791–4798, 2023. DOI: 10.1109/LRA.2023.3288383

- 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. DOI: 10.1109/ITSC57777.2023.10421988

- R. Marcuzzi, L. Nunes, L. Wiesmann, E. Marks, J. Behley, and C. Stachniss. Mask4D: End-to-End Mask-Based 4D Panoptic Segmentation for LiDAR Sequences. *IEEE Robotics and Automation Letters (RA-L)*, 8(11):7487–7494, 2023. DOI: 10.1109/LRA.2023.3320020

- Y. Wu, T. Guadagnino, L. Wiesmann, L. Klingbeil, C. Stachniss, and H. Kuhlmann. LIO-EKF: High Frequency LiDAR-Inertial Odometry using Extended Kalman Filters. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2024. DOI: 10.1109/ICRA57147.2024.10610667

- D. Casado Herraez, L. Chang, M. Zeller, L. Wiesmann, J. Behley, M. Heidingsfeld, and C. Stachniss. SPR: Single-Scan Radar Place Recognition. *IEEE Robotics and Automation Letters (RA-L)*, 9(10):9079–9086, 2024. DOI: 10.1109/LRA.2024.3426369

- Y. Pan, X. Zhong, L. Wiesmann, T. Posewsky, J. Behley, and C. Stachniss. PIN-SLAM: LiDAR SLAM Using a Point-Based Implicit Neural Representation for Achieving Global Map Consistency. *IEEE Trans. on Robotics (TRO)*, 40:4045–4064, 2024. DOI: 10.1109/TRO.2024.3422055

Chapter 2

Basic Point Cloud Processing Techniques

The focus of this work is to obtain globally aligned point cloud data from LiDAR scans, representing this data efficiently, and localizing within the resulting maps. All of our proposed methods utilize or process point cloud data in some form. In this chapter, we explain some basic techniques that we used in this thesis to process point clouds.

Formally, we define a point cloud $\mathsf{P} \in \mathbb{R}^{N \times 3}$ as a matrix of N point coordinates $\boldsymbol{p} \in \mathbb{R}^3$ sampled from the surface of the environment. Often we not only have point coordinates P but also features $\mathsf{F} \in \mathbb{R}^{N \times D}$ corresponding to those points, so that each point consists of its coordinate $\boldsymbol{p} \in \mathbb{R}^3$ and its point features $\boldsymbol{f} \in \mathbb{R}^D$.

Some features, like the timestamp of a measurement or the intensity, are usually directly provided by the LiDAR sensor. However, features can also be computed algorithmically, for example from the local neighborhood of a 3D point. Many of the methods developed in this thesis compute and utilize point features as an intermediate result for solving the respective task. One geometric point feature that we often use is the normal of a point, and we discuss its computation in Section 2.1. Furthermore, various deep learning-based approaches and networks exist that estimate point-wise features. In the following, we also look into the main network architectures for 3D deep learning, and discuss shortly their advantages and disadvantages.

2.1 Normal Computation

Point clouds are sampled coordinates from the surface of the environment. However, the individual point coordinates do not have any notion about how the world around them looks like. One geometric feature to describe the local structure around the point is the normal $\boldsymbol{n} \in \mathbb{R}^3$ that depicts the direction perpendicular

to the point's surface. Knowing for each point the direction of the underlying surface is key for many algorithms. The true point normal is often not available, therefore is computed heuristically using the points' local neighborhood $\mathcal{N}$. One way to compute the local neighborhood is based on the scatter matrix of the neighborhood

$$M = \sum_{\boldsymbol{p} \in \mathcal{N}} (\boldsymbol{p} - \bar{\boldsymbol{p}})(\boldsymbol{p} - \bar{\boldsymbol{p}})^\top \tag{2.1}$$

with the geometric mean

$$\bar{\boldsymbol{p}} = \frac{1}{|\mathcal{N}|} \sum_{\boldsymbol{p} \in \mathcal{N}} \boldsymbol{p}. \tag{2.2}$$

The scatter matrix M describes how the points are distributed in space. The normal $\boldsymbol{n}$ can then be computed by a singular value decomposition (SVD)

$$U, D, V = \mathtt{SVD}(M), \tag{2.3}$$

$$\boldsymbol{n} = V[:, 3], \tag{2.4}$$

where $V[:, 3]$ is the Eigenvector that corresponds to the smallest Eigenvalue. The neighborhood $\mathcal{N}$ is often defined either by its k-nearest neighbors or by all the points within a certain proximity r

$$\mathcal{N}(\boldsymbol{p}) = \{(\boldsymbol{q}, \boldsymbol{f}) \in \{P, F\}) \mid \|\boldsymbol{q} - \boldsymbol{p}\| < r\}. \tag{2.5}$$

The normal can play an important role, for example, when requiring point correspondences between multiple point clouds. Assuming local planarity can relax point-to-point correspondences to only require point-to-plane correspondences. This means that instead of requiring the correct corresponding point, we only require that the point lies on the same corresponding plane. Due to the typically large number of planar structures in man-made environments, e.g., walls or streets, this assumption is often easier to fulfill.

2.2 Neural Network Architectures for Point Cloud Processing

Nowadays, point features are often computed using neural networks. In this part, we want to focus on the most common neural network architectures to compute features. Remark, that those networks can be used to compute features suitable for different kinds of tasks. For which task a feature is well suited mostly depends on the loss function used to train the network architecture and not on the type of the used architecture. Here, we will solely discuss the different types of network architectures while the individual training processes and used loss functions will be covered in the individual approach sections.

2.2.1 PointNet

One of the earliest, but still relevant architectures is PointNet [180], which transforms each point into a high-dimensional nonlinear space. There are many different versions of PointNet, in the following we will describe the version used in this thesis. The input to PointNet are the point coordinates P and point features F concatenated to a single matrix $\mathsf{F}_i = [\mathsf{P} \mid \mathsf{F}] \in \mathbb{R}^{N \times D_i}$. Optionally, PointNet first estimates a transformation $\mathsf{T} \in \mathbb{R}^{D_i \times D_i}$ of the N input points F_i using a so called T-Net : $\mathbb{R}^{N \times D_i} \mapsto \mathbb{R}^{D_i^2}$. The T-Net consists of a multi-layer perceptron[1] MLP$(64, 128, 1024)$, which is followed by global max pooling and a second MLP$(512, 256, D_i^2)$ that transforms the extracted descriptor to the dimensionality of the desired transformation. The transformation T is then applied to the input points

$$\mathsf{F}_T = \mathsf{F}_i(\mathbb{1}_d + \mathsf{T}), \tag{2.6}$$

where $\mathbb{1}_d$ corresponds to the identity matrix and therefore T is just a residual added to the identity that facilitates learning. This provides the network with the possibility to extract for each point cloud a specific transformation to transform it into a common frame to achieve transformation invariance.

A following MLP$(64, 128, D_o)$ projects the transformed features $\mathsf{F}_T \in \mathbb{R}^{N \times D_i}$ of the T-Net into a higher-dimensional feature space $\mathsf{F}_o \in \mathbb{R}^{N \times D_o}$. In the following sections, we will denote this architecture as

$$\text{PointNet} : \mathbb{R}^{N \times D_i} \mapsto \mathbb{R}^{N \times D_o}, \tag{2.7}$$

that follows the previous explained steps to transform $\boldsymbol{F}_i$ into the output features F_o.

[1] Here, we use the convention that the argument of the MLP corresponds to the number of channels of the output, e.g., MLP$(4, 16)$ takes a not further specified input and produces 4 and then 16 output channels in the intermediate and final feature map.

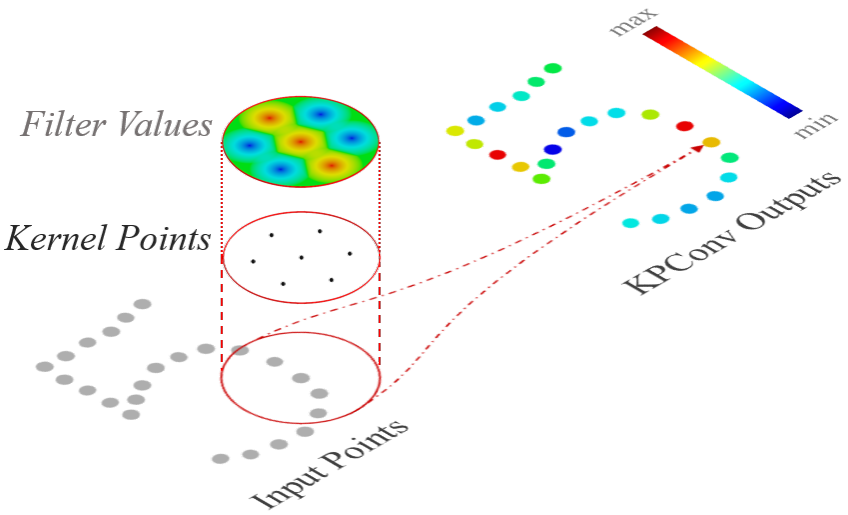


Figure 2.1: Illustration of kernel point convolutions in 2D. The output features are computed based on a weighted sum of the neighboring features. The filter weights are defined on some kernel points, such that the weights in between are interpolated. In this case, the filter values in the middle are high while the left and right values are low. The resulting features will have a high response for vertical structures and a low response for horizontal structures. Image courtesy by Thomas et al. [226].

Different variants of PointNet architectures utilize multiple T-Net's, or utilize global pooling on the output features to obtain only a single descriptor vector per point cloud. Note, that the network only operates on the individual point coordinates and does not propagate information from one point to another. The goal of this small network is to transform the input points into a feature space that is better suited for the final task.

2.2.2 KPConv

The previous PointNet architecture operates on each input point independently of the others, solely based on the individual coordinates and features. However, the way how the points are distributed in space, and how they are related with respect to each other usually contains relevant information.

A common way to aggregate information based on spatial distribution is the usage of convolutions. One way to define the convolutions directly on the point coordinates is shown by Thomas et al. [226] which results in the so-called kernel point convolution (KPConv). The convolution of the features $\boldsymbol{F} \in \mathbb{R}^{N \times D_i}$ at a point $\boldsymbol{p} \in \mathbb{R}^3$ with a convolutional kernel g is defined as a weighted sum of its neighboring features [226] as

$$(\boldsymbol{F} * g)(\boldsymbol{p}) = \sum_{(\boldsymbol{p}_i, \boldsymbol{f}_i) \in \mathcal{N}(\boldsymbol{p})} g(\boldsymbol{p}_i - \boldsymbol{p}) \boldsymbol{f}_i, \tag{2.8}$$

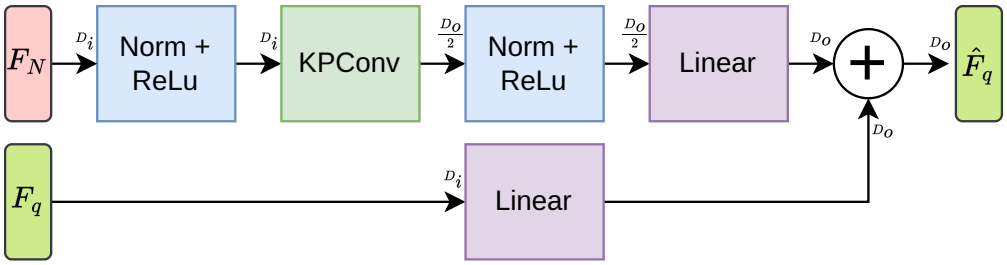


Figure 2.2: Schematic Overview of a ResNet-style KPConv block RKP. The features F_n within the neighborhood of the query q are first normalized and pass through an activation before they go into the convolution. The output of the convolution is again normalized and passed through a ReLu. A linear layer projects the feature into the desired dimension. Those features are added on a linear projection of the query features such that only a residual feature has to be learned. The dimensions of the features are denoted as D_i for the input and D_0 as the desired output dimension.

where $\mathcal{N}(\boldsymbol{p}) = \{(\boldsymbol{p}_i, \boldsymbol{f}_i) \in (P, F) \mid \|\boldsymbol{p}_i - \boldsymbol{p}\| < r\}$ are the neighbors of $\boldsymbol{p}$ within radius r. The convolutional weights are defined on M kernel points from which the weight of the neighbors $\mathcal{N}(\boldsymbol{p})$ are interpolated using first-order splines of size σ

$$g(\boldsymbol{y}_i) = \sum_{m<M} \max\left(0, 1 - \frac{\|\boldsymbol{y}_i - \hat{\boldsymbol{p}}_m\|}{\sigma}\right) \boldsymbol{W}_m, \tag{2.9}$$

with the relative coordinate $\boldsymbol{y}_i = \boldsymbol{p}_i - \boldsymbol{p}$ and the weight $\boldsymbol{W}_m \in \mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}$ of the m-th kernel point $\hat{\boldsymbol{p}}_m$. An illustration of the method can be seen in Figure 2.1.

The resulting kernel point convolution aggregates information from the neighborhood based on their relative position. We use the convolutions in ResNet-style [86] blocks RKP, as proposed by the Thomas et al. [226]. An illustration of RKP is shown in Figure 2.2. Note, that we will use LayerNorm [6] instead of batch normalization since we work with varying point cloud sizes which is not compatible with batch normalization since it assumes fixed input sizes. Multiple of those blocks are typically used in a network architecture to learn the desired features.

Convolutional neural networks aggregate in each block information from the neighboring points. However, incorporating information from faraway points into the features is hard for such network architectures. It requires either a very large neighborhood for the convolution or a very deep architecture. The problem with very large neighborhoods is that the network will be less sensitive to high frequency information, which usually degrades the performance. Since with each layer in the network the receptive field increases, one just has to make the network deep enough to have long range dependencies. However, very deep networks are considered harder to train, need more computational power, and are prone to

vanishing gradients. A network architecture which can better handle long-range dependencies is the Transformer architecture that we will discuss in the following section.

2.2.3 Transformer Architecture

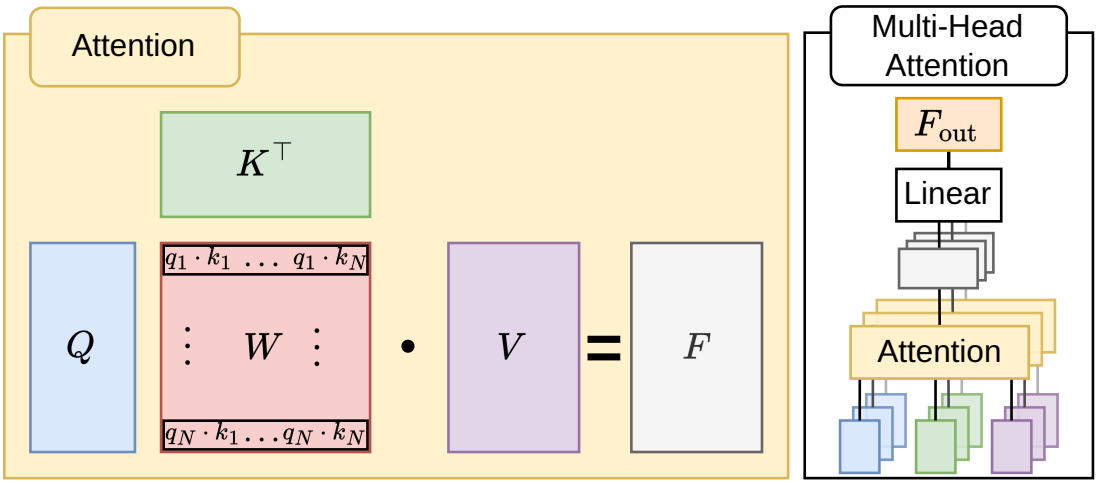
Another architecture we will look at is the Transformer [238]. Here, we discuss the Transformer in the context of point cloud processing instead of the original natural language processing domain. While convolutions aggregate information based on the spatial distribution, Transformers aggregate information based on feature relation. Therefore, focusing more on semantic context, rather than only spatial distribution. The fundamental method behind the Transformer architecture is the attention mechanism, where the goal is to compute for some target features $F_t \in \mathbb{R}^{N_t \times D}$ new features $F_o \in \mathbb{R}^{N_t \times D}$ based on the source features $F_s \in \mathbb{R}^{N_s \times D}$. Those features are in our case point features from a point cloud and might stem from a convolutional backbone. From the input features, we first compute three matrices: the keys $K = W_k F_s$ and values $V = W_v F_s$ which are linear projections of the source features F_s, and the queries $Q = W_v F_t$ that are projections from the target features F_t. The naming of the matrices is an analogy from database queries, which gives hints about their role in the method. We now want to query our key and value pairs such that we get an output feature based on how similar our query is to the keys.

For this, the attention mechanism in the Transformers computes new features $F_o \in \mathbb{R}^{N_t \times D}$ by a linear combination of value vectors $V \in \mathbb{R}^{N_s \times D}$. The weighting $W \in \mathbb{R}^{N_t \times N_s}$ of the features depend on the outer product of the queries $Q \in \mathbb{R}^{N_t \times D}$ and the keys $K \in \mathbb{R}^{N_s \times D}$

$$F_t = WV = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right) V, \tag{2.10}$$

computing the outer product for the weights means computing the dot product for each query to each key (see Figure 2.3a for illustration). Since the dot product is high when vectors are similar, zero when orthogonal, and negative when facing in the opposite direction, it results in a weighting based on feature similarity. The more similar a key to the query, the higher the corresponding weight.

Each row of W is first scaled by $1/\sqrt{D}$, followed by a softmax to ensure that the weights sum up to one and are always positive for each query. The attention mechanism is often not directly done on the whole features, but the features are split based on their dimension, such that the attention mechanism is applied for each part individually. This is the so-called multi-head attention which is depicted in Figure 2.3b. This could look like the following: you have a feature dimension of 256, split it into 8 times 32-dimensional parts, and apply



(a) Attention Mechanism (b) Multi-Head Attention

Figure 2.3: The attention mechanism computes a weight matrix W based on the feature similarity of the features Q and K. The resulting weights are used to compute a linear combination of the features V to compute the output F. The multi-head attention splits the input features into different parts and computes the attention for each part individually. The outputs are stacked together and fed into a linear layer to yield the output.

the attention mechanism to each of the 8 groups. This means that the weighting does not only depend on the overall feature similarities, but on the similarities of each group. The 8 resulting features are then concatenated and forwarded to a final linear layer to form the output feature.

We have seen that the basic attention mechanism aggregates features solely based on feature similarity. However, not considering spatial information at all might not be optimal, since spatial context and relation can contain valuable information. A positional encoder is often used to enable spatial awareness. The positional encoder enhances the features by adding position information. In the simplest case, one could just take the raw point coordinates and append them to the features. However, passing the position through different trigonometric functions has proven advantageous. An example of such a positional encoding is depicted in Figure 2.4. After evaluating the trigonometric functions the resulting positional encoding is added to the original feature.

If the source is equal to the target $F_s = F_t$ it is called self-attention. Note that for this case also the sequence lengths are equal $N_s = N_t$, meaning the weight matrix W grows quadratically with respect to the sequence length. If the source is different from the target it is referred to as cross-attention.

A transformer architecture can have several multi-head attention blocks chained together. The original architecture did not contain any downsampling, therefore for each input feature there will also be a corresponding output feature

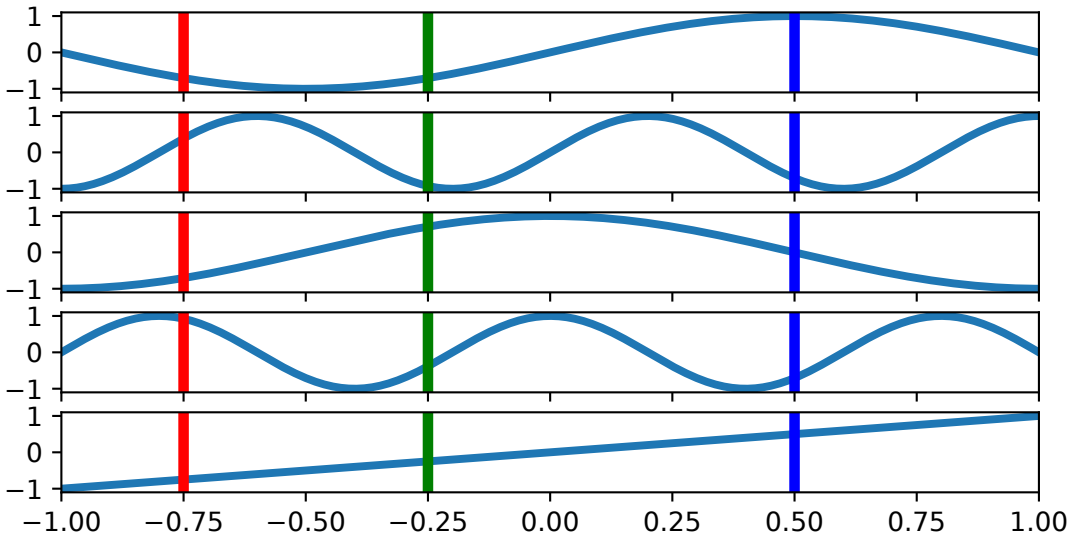


Figure 2.4: The positional encoding transforms the coordinates by applying trigonometric functions (sine and cosine) with different frequencies. Having a point $\boldsymbol{p} = [-0.75, -0.25, 0.5]$ would generate for this example a positional encoding for the x-coordinate of $\boldsymbol{f}_x = [-0.07, -0.38, -0.07, -0.92, -0.75]$. The final positional encoding would be the concatenation of all dimension-wise features $f_{\text{pos}} = [\boldsymbol{f}_x, \boldsymbol{f}_y, \boldsymbol{f}_z]$. Note that the magnitude and number of frequencies are hyperparameters and can be chosen freely.

generated. For point cloud processing, local attention [227, 296, 173] is often used, where instead of computing the attention to all points, only the points in a small neighborhood are considered.

2.2.4 Discussion

In this section, we briefly discuss the advantages and disadvantages of the methods. The PointNet architecture can not really aggregate information from other points, but transforms the points from one feature space into another. We use it in this thesis mostly if we have some features, which are already in a feature space, but want to transform them into another that is better suited for a specific task. Especially, when we take the output of a pretrained network for some task, we can use the PointNet to refine the features for another task without changing the weights of the pretrained network, which therefore can still be used for original task.

For aggregating information, we will mainly use KPConv. Convolutional neural networks have proven very descriptive, especially for generating local features. Due to the bigger compute demand for Transformer architectures, we will use them only rarely when we really want to have long range dependencies.

Note that there also exist different kinds of 3D convolutional neural networks. The probably most common 3D representation for learning is the usage of sparse voxel grids [38] with sparse convolutions. This extends the idea of images to 3D but exploits that many areas in the area are empty. The sparse grid-based convolution is mathematically quite similar to KPConv, but instead of defining it on the points, it can be defined directly on a grid. Sparse voxel grid-based methods are usually a bit faster but incorporate discretization which can lead to errors. We chose KPConv over sparse voxel grid-based convolutions to bypass discretization.

Chapter 3

Multi-Sensor System Calibration

Calibrating the sensors of any robotic system is a crucial prerequisite for mapping, localization, SLAM, and other state estimation tasks. Knowledge about the properties of the sensors is required to obtain consistent and undistorted measurements. In this thesis, we aim to build maps of the environment in which we can later localize. Before we can even go out with our multi-sensor system to map the environment, we first need to make sure we know our sensors' properties and how they are located on the robot. Each sensor measures in its own coordinate system, therefore the relative transformations (extrinsics) between the sensors must be known to operate in a common reference frame. For example, the point clouds from two LiDAR sensors that are mounted on a robot can be fused to one consistent map of the vehicle's surroundings by knowing the sensors' relative transformation to each other. Intrinsic calibration, on the other hand, aims to obtain a correct model between the measurements of a sensor and the corresponding object properties in the physical world. An RGB camera, for example, provides images that consist of pixels with color information. To infer information from an image to an object (or vice versa), we need to precisely know, for each pixel location in the image to which direction in the 3D world it corresponds. Once we have our sensor system calibrated, we can move around, record data, and process the data to obtain a map within we can localize, as depicted in Figure 3.1. In this chapter, we investigate the problem of estimating the extrinsic and intrinsic parameters of perception sensors.

The sensor system we mainly use in this thesis is shown in Figure 3.2 on the top left, which is a car mount equipped with four wide-angle cameras, two multibeam LiDAR sensors as well as navigation sensors, namely GNSS receiver and inertial measurement unit (IMU). This setup is also later used to obtain the measurements for the mapping in Chapter 4. Other systems we calibrate have, e.g., four RGB-D cameras, an RGB camera, and two 2D profile LiDAR sensors. The main challenges of these systems are no or limited overlap between

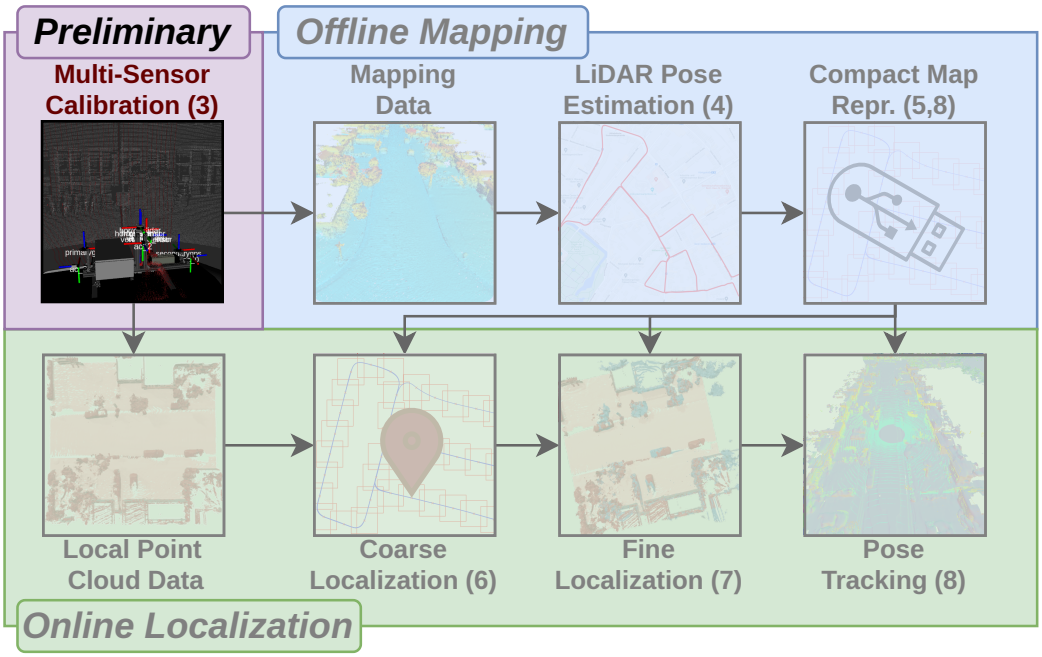


Figure 3.1: Schematic overview of the thesis. In this chapter, we want to calibrate the perception sensors of our multi-sensor systems to enable localization and mapping.

cameras and limited overlap between the LiDAR sensors. Furthermore, some of the cameras have a fisheye lens, i.e., that has a projection that cannot be described well using the pinhole model. Therefore, we need the calibration able to work for different projection models (projection on a plane or a sphere) as well. Additionally, we are interested in a calibration procedure that works with multiple sensor setups, is easy to extend, and requires minimal user input. We aim at millimeter accuracy and thus to be more precise than the sensors' noise.

The main contribution of this chapter is a flexible calibration procedure that allows to estimate the intrinsics and extrinsics of a combination of different perception sensors. Instead of directly estimating the relative transformation between the sensors, we use an external high-accuracy sensor, a terrestrial laser scanner (TLS), to obtain a reference target. This allows us to reliably and efficiently estimate the poses of each sensor. By doing so, we can exploit the strengths of each sensor individually, making it suitable across varying sensors with different configurations without the need for overlap between the sensors' field of views. We propose to utilize a calibration environment instead of using object-based targets such as checkerboards. Our calibration environment is a room equipped with AprilTag [225] targets, i.e., printed QR-like codes for calibrating the cameras. For the LiDAR, we installed structural elements in the calibration room which allows for reliable matching.

One exemplary use case would be, where we want to estimate the depth for the pixels in an image using LiDAR data, as shown in Figure 3.2. For this, we

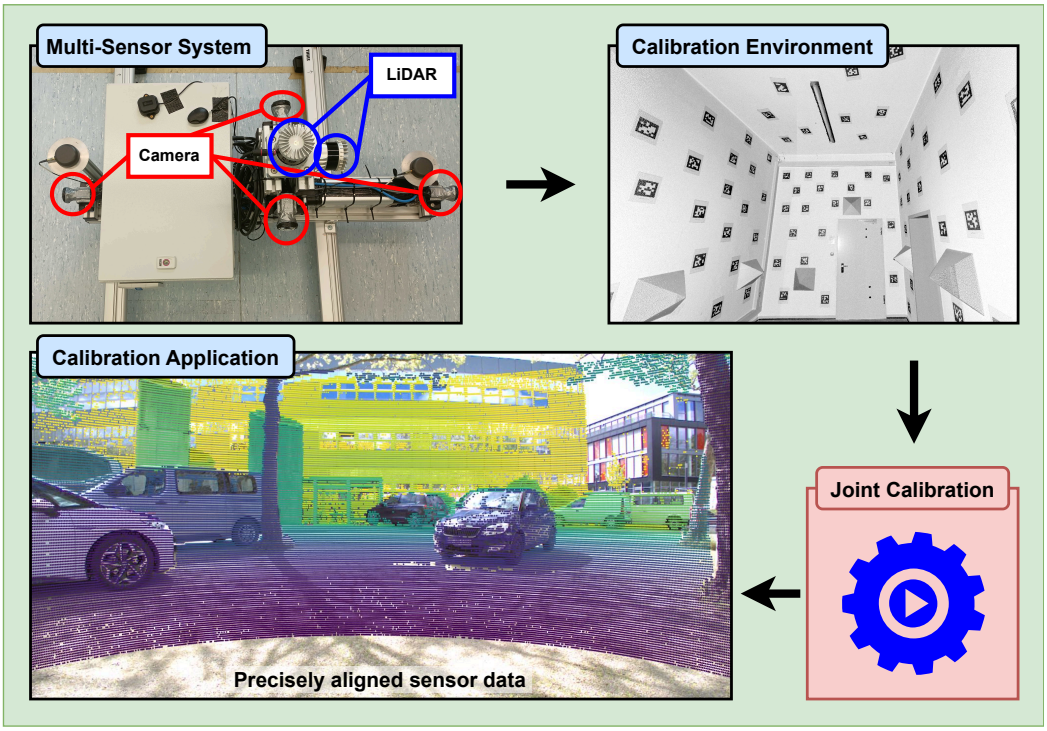


Figure 3.2: Having a system that uses multiple sensors requires calibration. Our approach calibrates different perception sensors, like LiDAR sensors or cameras, by utilizing a calibration environment. Jointly optimizing the intrinsics and extrinsics of the sensors allows for different applications, such as sensor fusion. In this application, the scan of the horizontal LiDAR is projected into the left camera, where the colors of the points denote the distance to the sensor.

need to transform the observed LiDAR point cloud into the coordinate system of the image. First, we need to acquire data with the multi-sensor system in the calibration environment and measure with each sensor the environment multiple times. The recorded data is jointly calibrated utilizing the point cloud from a high-precision TLS, yielding the extrinsics and intrinsics of the sensors. Afterward, the data of the camera and LiDAR can be fused. The point cloud of the LiDAR must be transformed into the camera system using the extrinsics of the sensors. Second, the intrinsics of the camera are used to project the point clouds into the image, such that we can assign the range values of the LiDAR measurements (in the camera frame) to the corresponding pixel. If the calibration was successful the LiDAR range overlaid on the image should be well aligned, as in Figure 3.2 on the bottom left.

In sum, our approach is able to reliably estimate the intrinsics and extrinsics of perception systems; operate with different sensor types (e.g., LiDAR, and camera) and modalities (e.g., profile scanner vs. multibeam LiDAR; wide-angle vs. fisheye cameras); and calibrate different perception systems with varying sensor configurations.

3.1 Related Work

Approaches for calibration can be divided into two categories: Methods using natural scenes, mostly outdoor, and methods using scenes with specific targets.

Early target-less approaches measured the correspondence between LiDAR and camera image points manually [197]. Recently, LiDAR intensity was used and compared with the camera image intensity for calibration [100, 115, 170]. We aim for an automated approach without manual measurements and do not want to rely on the quality of the LiDAR intensity. Geometrical approaches [93, 131] for natural scene-based calibration utilize the onboard sensors to first estimate the vehicle's trajectory or build a map, which can then be taken for localizing the other sensors. We believe that the calibration using natural scenes require a lot of user input and experience to make it work for different sensors and perception platforms.

Target-based approaches can be characterized by their targets: Planes with black rings [186], trihedrons [69], two planar triangles [51], and spheres [119] are among the less often-used objects. Some works [16, 274] define special objects that have different properties to support the different sensor characteristics. One of the most commonly used calibration targets is the checkerboard. While for the detection in the image standard tools exist [25], for the detection in the LiDAR, the methods usually vary. Some approaches [92, 251, 298] try to estimate the full geometry of the board while others [169, 233, 239, 293] only use the plane normal and the center point to estimate the transformation since the edges themselves are too inaccurate. Verma et al. [239] use a genetic algorithm for the estimation, while Tsai et al. [233] introduce a quality measure to select a subset of frames used in the estimation procedure. All these approaches have the restriction that the board must be seen completely in both, the LiDAR and the camera, which reduces the possible positions and angles of the board and limits the calibration to systems that have overlap between those sensors. Only a small fraction of the LiDAR point cloud, usually significantly below 25%, can be used to estimate the transformation between the LiDAR and the target. As our target is a 3D terrestrial laser scan of a whole room (see Figure 3.2), we are able to use nearly *all* points (except for some very small amount of outliers) of a LiDAR scan to estimate the transformation between the LiDAR sensor and the target.

Some methods [58, 234], to which we would also associate our approach, do not only rely on object-sized targets that need to be moved but rather use an infrastructure-based calibration where the whole environment is the target. The most similar approach compared to ours is the work by Xie et al. [234]. Like us, they use a room with AprilTags, on the walls and perform and estimate the extrinsics between all cameras and LiDAR sensors. The proposed method

divides the estimation of the extrinsics from the cameras and LiDAR into two parts. First, they estimate the extrinsics of the camera and the poses of the sensor system using the image data of the cameras. They try to minimize the reprojection error of AprilTag coordinates in the image, which is also the error function we try to optimize for the cameras. In a second, independent step, they optimize the extrinsics of the LiDAR using an ICP-like error function. One key difference to our approach is that they assume that the intrinsics of the cameras are already known. Thus, requires a calibration of the intrinsics beforehand. In contrast, our approach estimates the intrinsics jointly with the other parameters; we do not assume them to be given. Independent optimization of the extrinsics and intrinsics is statistically suboptimal since for the extrinsics estimation assumes the intrinsics to be correct without incorporating their uncertainty. Additionally, they split up the estimation of the LiDAR parameters from the estimation of the poses and camera parameters. Consequentially, they assume for this step the poses of the vehicle to be given and independent of the LiDAR observations. Here as well, a joint optimization is statistically better since all the observations from all the sensors are taken into account for the parameter estimation. Consequently, we optimize the parameters of the poses, cameras, and LiDAR sensors in a joint adjustment. A difference of the environments used is that we equipped our calibration room additionally with structural elements to have additional planes at different angles (c.f., Figure 3.2). This provides a more stable solution especially when handling data from the 2D profile LiDAR scanners. Fang et al. [58] also proposes an infrastructure-based calibration method. The objective functions they minimize are quite similar to our and the previously discussed approach [234]: reprojection error of markers for the camera, and ICP for the LiDAR sensors. Instead of utilizing the TLS for generating the reference point cloud, they use a stereo camera and bundle adjustment to reconstruct the calibration environment. They separate the estimation of the LiDAR sensor parameters and camera parameters and do not optimize for the intrinsics, but in contrast to Xie et al. [234] they do not fix the poses of the vehicle for the LiDAR. Still, this does not provide a statistically optimal solution. Furthermore, they do not use uniquely identifiable markers, which requires them to find the correct correspondences first.

3.2 Multi-Sensor Calibration Using a Precise Calibration Environment

Our main idea for the calibration is to relate the measurements of each sensor to a once created, precise reference map of the calibration environment. The advantage of this is that we can exploit the strengths of each sensor, do not rely on high FoV overlap between the sensors, and it is applicable to different sensor configurations and types. Our method substantially simplifies the process of obtaining a high-quality calibration, especially when using multiple different robots or perception platforms. In this chapter, we look at robots and sensor systems consisting of a combination of different cameras and/or LiDAR sensors. Our calibration procedure can be summarized in five steps:

1. Generating a reference map of the calibration environment.
2. Defining for each sensor an error function between the observations and the reference map based on the intrinsics and extrinsics of the respective sensor.
3. Collecting measurements from the sensors in the calibration environment.
4. Estimating initial values for the parameters as needed.
5. Performing a joint optimization to obtain the extrinsics and intrinsics of the whole sensor system.

Note that step 1 needs to be done only once for every calibration environment, and step 2 once for each sensor. In the following, we describe the steps in more detail.

3.2.1 Generating a Reference Map

Calibrating sensors with respect to a reference leads to some requirements on the sensors and the target. In our case, the reference is a 3D point cloud map of the calibration environment. The reference map should cover most of the scene that will be seen by each individual sensor in the calibration process. We do not rely on overlaps in the FoV between the sensors of the multi-sensor system, but between the sensors and the reference map (a calibration room in our case) instead. Additionally, the reference map should be as accurate as possible, since errors in the map could propagate into the parameters of the calibration.

We rely on transformation estimation between the sensors and the reference map. For the LiDAR, we utilize ICP to the reference map, and for the camera images rely on automatically extracted coded targets with given 3D coordinates.

We rely on the popular AprilTags [166]. A dense 3D map with the possibility to extract the AprilTag positions is required. Consequently, we propose to use a TLS as the sensor to obtain the reference point cloud map. A TLS produces point clouds with millimeter accuracy, 360° FoV, and with a high density. The 3D coordinates of the coded targets must be in the same reference frame. Thus, we extract them from the reference point cloud map.

For calibrating the LiDAR sensors, the target point cloud needs to have enough geometric structure to reliably fix the 6 degree of freedom (DoF) of the pose. Since we used an empty room for the calibration, we added some structural elements in the form of pyramids to the walls. By doing so, we can ensure that we have enough information to fix the degrees of freedom along the wall surfaces. Without having enough structure, the point clouds can arbitrarily move along the wall, which is especially a problem with data from 2D LiDAR sensors.

Given the 3D coordinates of AprilTags and their corresponding image coordinates, we can directly use them in a bundle adjustment to obtain accurate poses and intrinsics of the cameras. We directly use the positions of the AprilTags that can be extracted from the TLS point cloud. Because this point cloud is highly dense, the code of the AprilTags is clearly visible in the intensity channel of the scan, as seen in Figure 3.4b. Thus, we create an image with orthographic projection, so-called orthophoto, of each wall in the room using the intensity channel of the points. Then, we use the standard AprilTag library [225] to extract the 2D subpixel-accurate image coordinates of the AprilTag corners in the orthophotos. As every pixel in the orthophoto has its corresponding 3D coordinate in the TLS point cloud, we can easily extract the 3D coordinates of the AprilTag corners by bilinear interpolation with high precision.

This way, our reference map consists of a point cloud $\mathcal{M} = \{({}^m\boldsymbol{p}_i, {}^m\boldsymbol{c}_i)\}$ with $i = 0, \ldots, I$ points $\boldsymbol{p}_i \in \mathbb{R}^3$ in Euclidean coordinates with their associated intensity $\boldsymbol{c}_i \in \mathbb{R}$ and a set of J AprilTag corner coordinates, both located in the coordinate frame of the reference map m. The coordinate frame of m can be chosen freely and might be the origin defined by the terrestrial laser scanner's internal frame.

3.2.2 Define Error Functions

To calibrate the sensor setup, we need to define the error function that we want to optimize. Generally speaking, we want to minimize for all sensors, at each timestamp, the errors between their observations and the reference map. In this chapter, we focus on LiDAR and camera sensors, but the procedure can be used for different sensors as long as it is possible to relate the sensor measurements to the reference map. The relation is usually simply transforming sensor obser-

vations from the reference map in an unified frame and computing the deviation between those.

In the following, we denote the transformation of the frames by the right subscript and left superscript as commonly used in physics, e.g., the transformation of a point from the frame i to frame j would be ${}^{j}\boldsymbol{p} = {}^{j}\boldsymbol{R}_i \, {}^{i}\boldsymbol{p} + {}^{j}\boldsymbol{t}_i$, where ${}^{j}\boldsymbol{R}_i \in \mathbb{R}^{3\times3}$ is the rotation matrix and ${}^{j}\boldsymbol{t}_i \in \mathbb{R}^3$ the translation vector. 3D point coordinates are denoted by $\boldsymbol{p} \in \mathbb{R}^3$, while image coordinates have the variable $\boldsymbol{x} \in \mathbb{R}^2$.

3.2.2.1 Camera Error Function

For calibrating the cameras, we relate the sensor observations to the reference map by using AprilTags. For the camera c, we extract for a specific timestamp t the corners $\{{}^{i}_{t}\boldsymbol{x}_j\}$ of all AprilTags that are visible in the current image i. As an error metric, we use the reprojection error $e_{\text{camera}}(i,t,j)$ of the AprilTag coordinates, such that for the j^{th} observation, we yield:

$$e_{\text{camera}}(i,t,j) = {}^{i}_{t}\hat{\boldsymbol{x}}_j - {}^{i}_{t}\boldsymbol{x}_j \tag{3.1}$$

$${}^{i}_{t}\hat{\boldsymbol{x}}_j = \boldsymbol{F}_c \, {}^{d}_{t}\hat{\boldsymbol{x}}_j + {}^{i}\boldsymbol{t}_d \tag{3.2}$$

$${}^{d}_{t}\hat{\boldsymbol{x}}_j = \text{distort}({}^{u}_{t}\hat{\boldsymbol{x}}_j) \tag{3.3}$$

$${}^{u}_{t}\hat{\boldsymbol{x}}_j = \text{project}({}^{c}_{t}\hat{\boldsymbol{p}}_j) \tag{3.4}$$

$${}^{c}_{t}\hat{\boldsymbol{p}}_j = {}^{b}\boldsymbol{R}_c^{\top} \, {}^{b}_{t}\hat{\boldsymbol{p}}_j - {}^{b}\boldsymbol{R}_c^{\top} \, {}^{b}\boldsymbol{t}_c \tag{3.5}$$

$${}^{b}_{t}\hat{\boldsymbol{p}}_j = {}^{m}_{t}\boldsymbol{R}_b^{\top} \, {}^{m}\hat{\boldsymbol{p}}_j - {}^{m}_{t}\boldsymbol{R}_b^{\top} \, {}^{m}_{t}\boldsymbol{t}_b. \tag{3.6}$$

Namely, we transform the AprilTag coordinate ${}^{m}\hat{\boldsymbol{p}}_j \in \mathbb{R}^3$ in Equation (3.6) first from the frame of the reference map m over the base-link b (a local coordinate system on the robot) into its camera frame c using Equation (3.5). From there, we project the point in Equation (3.4) depending on the type of camera into a unified intrinsic free camera frame u in which we apply the non-linear camera distortions, i.e., as seen in Equation (3.3). After the distortion, we obtain in Equation (3.2) the AprilTag coordinate ${}^{i}_{t}\hat{\boldsymbol{x}}_j \in \mathbb{R}^2$ in the image frame i by applying the focal length $\boldsymbol{F}_c \in \mathbb{R}^{2\times2}$ and the principal point ${}^{i}\boldsymbol{t}_d \in \mathbb{R}^2$. The focal length matrix $\boldsymbol{F}_c$ is a diagonal matrix with $[f_x, f_y]$ on the main diagonal. For the distortion, we use tangential and radial distortions similar to OpenCV [25]:

$${}^{d}_{t}\hat{\boldsymbol{x}}_j = \text{distort}({}^{u}_{t}\hat{\boldsymbol{x}}_j) \tag{3.7}$$

$${}^{d}_{t}\hat{\boldsymbol{x}}_j = {}^{u}_{t}\hat{\boldsymbol{x}}_j \Big(1 + \sum_{n=1}^{N} k_{n,c} r^{2n}\Big)^{\tau} + 2 \, {}^{u}_{t}\hat{\boldsymbol{x}}_j \, {}^{u}_{t}\hat{\boldsymbol{x}}_j^{\top} \boldsymbol{p}_c + r^2 \boldsymbol{p}_c, \tag{3.8}$$

with the radial coefficients $\{k_{n,c}\}$, and the tangential coefficients $\boldsymbol{p}_c = [p_2, p_1]^{\top}$. The parameter τ can be changed for different radial distortion modeling, i.e., the classical Brown's distortion model has $\tau = 1$, while the division model has

$\tau = -1$. For the projection, we either use the classical pinhole or equidistant model [271] as follows:

$$\text{project}\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{cases} \begin{bmatrix} \frac{x}{z} & \frac{y}{z} \end{bmatrix}^\top & \text{if } c \text{ is pinhole} \\ \begin{bmatrix} \frac{x}{r_{xy}} \text{atan2}\left(r_{xy}, z\right) \\ \frac{y}{r_{xy}} \text{atan2}\left(r_{xy}, z\right) \end{bmatrix} & \text{if } c \text{ is fisheye,} \end{cases} \tag{3.9}$$

with the Euclidean distance in the image $r_{xy} = \sqrt{x^2 + y^2}$.

Additionally, we add a prior on the AprilTag coordinates $\hat{\boldsymbol{p}}_j$, defined by:

$$e_{\text{prior}}(j) = \hat{\boldsymbol{p}}_j - \boldsymbol{p}_j^{(0)}, \tag{3.10}$$

where $\boldsymbol{p}_j^{(0)}$ denotes the initially extracted AprilTag coordinates from the TLS point cloud map. By this, we can incorporate the uncertainty in the AprilTag extraction without giving too much freedom for pushing errors from the camera model into the AprilTag coordinates.

3.2.2.2 LiDAR Error Function

For estimating the extrinsics and intrinsics of the LiDAR sensors, we align the point clouds as well as possible with the reference map. Therefore, this part is similar to classical point cloud registration methods [21]. We use the classical point-to-plane error function, as often used in ICP [36]. For this, we compute, for the dense and accurate reference map, normals for each point based on their local neighborhoods. More details on normal computation are provided in Section 2.1. The key difference from most ICP-based methods is, that we do not try to independently align each point cloud with the reference, but jointly optimize all sensors and scans together. Thus, we optimize not only one pose per scan but the whole kinematic chain. This results in

$$e_{\text{LiDAR}}(l, t, j) = {}^m\boldsymbol{n}_k^\top ({}^m_t\mathsf{R}_b \, {}^b\hat{\boldsymbol{p}}_j + {}^m_t\boldsymbol{t}_b - {}^m\boldsymbol{p}_k) \tag{3.11}$$

$${}^b\hat{\boldsymbol{p}}_j = {}^b\mathsf{R}_l \, {}^l\boldsymbol{p}_j \left(s_l + \frac{o_l}{\|{}^l\boldsymbol{p}_j\|}\right) + {}^b\boldsymbol{t}_l, \tag{3.12}$$

where ${}^m\boldsymbol{p}_k$ and ${}^m\boldsymbol{n}_k$ are the corresponding map points and normals of the j$^{\text{th}}$ LiDAR point ${}^l\boldsymbol{p}_j$.

We estimate as intrinsics a scale factor s_l and offset o_l for each LiDAR to address systematic errors in the range measurements. The correspondences are obtained by searching for each LiDAR point ${}^l\boldsymbol{p}_j$ the closest point in the reference map $\mathcal{M}$. Due to structural elements, like the pyramids in our reference, we are able to use this procedure not only for 3D multibeam LiDAR sensors but also for the commonly used 2D profile LiDAR.

3.2.3 Collecting Measurements

The measuring process for our calibration setup is rather straightforward: we only assume that the measurements from each sensor are obtained at discrete points in time. By this, we can optimize the pose of the sensor system r_t based on all the observations taken at the same timestamp t from all sensors. This does not necessarily mean that all the sensors need to be hardware-triggered at the exact same time and with the same frame rate (although a good time synchronization is in general recommended). We only need to keep the scene and sensors static while taking the measurements which is easy to realize in a dedicated calibration room.

In general, we strongly suggest recording in a stop-and-go manner, i.e., (1) move the sensor system, (2) measuring with each sensor while standing still, and (3) repeat steps 1 and 2 as much as needed. Thereby, we also avoid the motion distortion in the measurements, e.g., motion blur in the cameras, rolling shutter effects, or motion distortion in the LiDAR scans.

For an optimal calibration result, the observations should cover the full field of view of the sensor. For example, for a camera, we suggest to not only have observations in the center but rather distributed over the whole image. In our room, this recommendation is more or less fulfilled automatically, because all walls, including the ceiling have a sufficient coverage of AprilTags.

Since we do not rely on any human interaction like moving checkerboards, but only on a static environment (like a separate calibration room), this method is well suited for full automation, e.g., in an industrial production line or for repeated calibrations.

3.2.4 Initial Parameter Estimates

Using the Gauss-Newton model to solve the non-linear optimization problem requires initial values for the parameters. As parameters, we have the 6 DoF pose parameters $\{({}^{m}_{t}\boldsymbol{R}_b, {}^{m}_{t}\boldsymbol{t}_b)\,\forall t\}$, i.e., the transformation parameters from the frame of the base-link b to the frame of the reference map m, as well as the extrinsics, i.e., the transformations $\{({}^{b}\boldsymbol{R}_s, {}^{b}\boldsymbol{t}_s)\,\forall s\}$ from the sensor frame s into the base-link frame b. We chose the first camera as base-link, but this choice is arbitrary. Additionally, we need for each sensor the intrinsics, e.g., focal length, principal point, and distortion coefficients for each camera, as well as scale and offset for each LiDAR. The offset can model a bias in the range measurements, while the scale can compensate when the LiDAR sensor systematically over or underestimates the ranges proportional to the distance.

As an initial guess for the intrinsics of the LiDAR sensors, we assume $s_l \approx 1$ and offset $o_l \approx 0$. The intrinsics of the cameras are estimated by the well-

established method by Zhang [295]. Since this requires all points to lie on a plane, we only use the AprilTags from the wall, which has the most visible tags. We use multiple frames with at least 3 visible tags to ensure a reliable estimation. The initial extrinsics $\{({}^{b}\boldsymbol{R}_{s}, {}^{b}\boldsymbol{t}_{s})\,\forall s\}$ can be taken using construction plans of the multi-sensor system, manual measuring, or computing the relative transformation between the sensors and the base-link from a direct solution. In our experiments, it was sufficient to provide the extrinsics with a couple of centimeters and degrees accuracy, i.e., a simple ruler is sufficient.

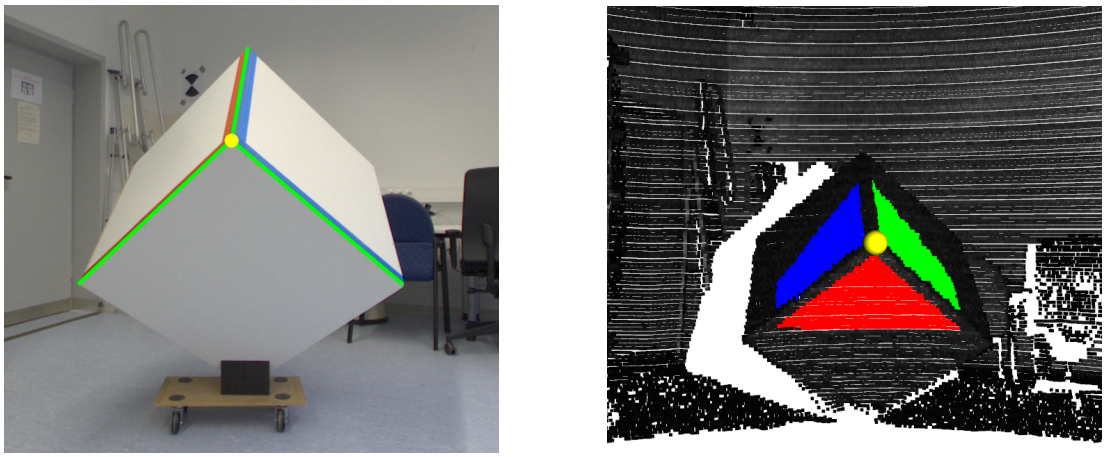
We obtain the poses $\{({}^{m}_{t}\boldsymbol{R}_{b}, {}^{m}_{t}\boldsymbol{t}_{b})\,\forall t\}$ by estimating independently for each timestamp the pose of one of the sensors in the reference map. We use the perspective-n-point [125] algorithm when taking one/ multiple cameras to estimate the poses of the base-link sensor in the reference map. In the upcoming experiments described in Section 3.4, we take for each timestamp the camera with the most visible AprilTags to estimate the pose. In the case of calibrating only multiple LiDAR sensors, we can also use global registration techniques. We used, for example, the approach by Rusu et al. [194] that uses feature-based correspondences with FPFH features, and searches for the best fit using RANSAC, which provided a sufficient estimation for an initial guess.

3.2.5 Joint Optimization

To obtain the statistically optimal solution for the calibration parameters, we optimize all the sensors in a joint least squares adjustment. Each sensor is rigidly connected to the platform and thus correlated to the other sensors. We use the Gauss-Newton model for optimization. We obtain an estimate of the accuracy of the parameters using the inverse of the normal equation system. The covariances of the observations should be chosen such that the standardized residuals are approximately standard-normal distributed.

In each iteration of the Gauss-Newton algorithm, we update the correspondences for the point clouds to ensure always having the closest points, as also done in ICP. Errors in the initial parameters can lead to wrong associations. Therefore, we use the Geman-McClure robust kernel to reduce the impact of those points.

Once the Gauss-Newton method is converged, we disable the robust kernel and optimize a second time without the robust kernel, removing outliers that are further away than three times the specified sensors standard deviation (3σ bound).



(a) Camera image

(b) LiDAR point cloud

Figure 3.3: Measurement of cube corners for evaluation. In the camera image (a) the corner point (yellow) is the intersection of the image edges (green). In the point cloud (b), the corner (yellow) is the intersection of 3 planes estimated using RANSAC based on the red, green, and blue points.

3.3 LiDAR-to-Camera Evaluation Method

Estimating all parameters in a joint least squares adjustment allows for obtaining the analytical covariances, especially when choosing realistic covariances in the optimization. But due to imperfect assumptions about the model, covariances, correlations, and linearization, the estimated covariances might be too optimistic. Thus, we evaluate the system separately. Additionally, it allows us to perform an independent comparison to other calibration methods.

Since we are interested in using the camera data in combination with the LiDAR sensors, for example, to project 3D LiDAR points into the images as shown in Figure 3.2, we focus on the analysis of the calibration between these sensors. We propose an evaluation method for the independent assessment of the accuracy between LiDAR sensors and cameras. Finding reliable corresponding points in both sensors allows us to compute the reprojection error.

Throughout our experiments, we saw that picking distinct points in the point cloud or range image of the LiDAR sensors was not precise enough, due to the limited resolution of the LiDAR. For an accurate evaluation, the resolution of the evaluation method should be higher than the accuracy of the calibration, otherwise, aliasing effects can occur. Therefore, we propose to use a physical cube to find distinct corners in the image and in the point cloud. By extracting three visible planes of the cube from the point cloud, we can precisely compute the point of intersection. We guide this process by manually selecting a point near the cube corner. The corresponding point in the image can be extracted by

(a) Calibration room (b) Reference point cloud

Figure 3.4: The calibration environment is equipped with AprilTags for the camera calibration and structural elements for the LiDAR. (a) shows a picture of the room, and (b) shows the corresponding point cloud that is used as a reference target. The point cloud is obtained using a Faro Focus3D-X130 terrestrial laser scanner.

finding the three edges between the three planes. The accuracy of the image can be computed by projecting the detected corner from the LiDAR into the image and calculating the residual at the intersection point of the three image edges. Additionally, we use the distance between the estimated point in the LiDAR and the viewing ray to the corner in the image. The first gives a residual in pixels, while the latter provides a metric error in 3D space. A visualization of the cube measurements is depicted in Figure 3.3. Note that the cube is not needed for calibration, only to evaluate the calibration results independently.

3.4 Experimental Evaluation

The main focus of this chapter is a calibration procedure that reliably works for different sensor setups. In the following, we will first look at the main sensor setup and the used calibration environment. Afterward, we evaluate our proposed method and compare it to other approaches. In the end, we will look at the calibration results of different sensor setups to show that our method is of general use.

3.4.1 Experimental Setup

In this chapter, we aim at calibrating multi-sensor perception systems with the help of a specifically designed calibration environment. The calibration environment is depicted in Figure 3.4 and the main sensor system utilized can be seen in Figure 3.6 (a). It is equipped with four Basler Ace cameras facing the front, left, right, and to the rear of the vehicle. Additionally, the system has an Ouster

OS1-128 LiDAR scanner with 128 beams and a 45° vertical field of view that is mounted horizontally. A second Ouster OS1-32 is mounted vertically and has 32 vertical beams. All sensors are PTP time-synchronized.

As a calibration environment, we place 119 AprilTags at the four walls and the ceiling in an otherwise empty room. The 3D coordinates of the tags are extracted as described in Section 3.2.1. We mounted structural elements in the shape of pyramids to the walls to fix all DoF of the pose for the LiDAR scan.

3.4.2 Calibration Evaluation

The main goal of this chapter is to provide a reliable calibration method for multi-sensor perception systems. Therefore, we look in this experiment into the accuracy of our method and compare it to other approaches. We compare our calibration environment-based approach against Ca^{2}Lib [66], a LiDAR-to-camera calibration approach using a chessboard for calibration, and a natural scene-based approach [115]. Since the approaches calibrate one laser with one camera, we perform this four times to obtain the calibration between all the sensors. Additionally, the approach assumes the cameras to be intrinsically calibrated; therefore, we provide the necessary intrinsics. Multi-sensor calibrations in a specially designed calibration environment are very rare; therefore, we implemented a baseline using standard tools provided by OpenCV [25] and Open3D [299], which we denote as CV2O3D. For this, we register each sensor to the reference map. The extrinsics between the sensors can be obtained by computing the relative pose between the sensors. We can do this for each timestamp independently, and after removing outliers, estimate the mean transformation. For the camera, we first estimate the intrinsics using Zhang's method [295], followed by estimating the pose in the map using the classic PnP algorithm with the AprilTag coordinates. The poses of the LiDAR in the map can be estimated using a RANSAC-based global registration, followed by a point-to-plane ICP for fine registration between the scans and the reference point cloud map. The difference between our approach is that the standard tools only allow for independent estimation of the sensor states, while our approach optimizes all the poses, extrinsics, and intrinsics jointly.

We evaluated all approaches using the same cube dataset (see Section 3.3) with 34 measured cube corners using the horizontal OS1-128 LiDAR and all cameras. The position and distance to the cube vary such that we have a high FoV coverage. Table 3.1 shows the RMSE errors. Our approach is outperforming the scene-based calibration [115], checkerboard baseline Ca^{2}Lib [66] as well as CV2O3D, which uses exactly the same data as our approach. We believe that our configuration in the calibration environment is more stable since we take in each timestamp the observations of all sensors into account and thus obtain better results. Additionally, with the checkerboard, only the few observations that lie

Table 3.1: Calibration evaluation

Model	RMSE [pix]	RMSE [m]
Scene-based [115]	30.71	0.064
Ca^{2}Lib [66]	11.13	0.024
CV2O3D	4.17	0.010
Ours	**2.51**	**0.007**

Table 3.2: Ablation of different models

Camera			LiDAR		Metric	
	Model	Degree	Bias	Scale	RMSE [pix]	RMSE [m]
[A]	D	3	✗	✗	3.91	0.008
[B]	D	3	✗	✓	3.22	0.009
[C]	D	3	✓	✗	2.62	0.008
[D]	B	2	✓	✓	2.72	0.008
[E]	B	3	✓	✓	2.54	0.008
[F]	D	1	✓	✓	4.89	0.013
[G]	D	2	✓	✓	2.75	0.008
[H]	D	3	✓	✓	**2.51**	**0.007**

on the board are considered. The scene-based approach has a lot of potential correspondences, but finding the correct ones, especially based on the not so reliable intensity, might be hard without incorporating at least some outliers. In the calibration environment, on the other hand, we can use all the points for the ICP, and due to the AprilTags have fixed correspondences for the cameras.

3.4.3 Model Analysis

In this experiment, we want to show the impact of different parameterizations of the sensor models to validate our design choices. Note, that this should be done for each sensor system to choose the right model for each sensor. The results are depicted in Table 3.2. When looking at the results of configurations [A]-[C], and [H], we see the impact of estimating intrinsics of the LiDAR. The results without estimating a scale and offset parameter [A] are notably worse than estimating either of those ([B] or [C]), while the best is achieved when estimating both as can be seen in [H]. When further analyzing the residuals between the LiDAR points and the reference map after optimization, as depicted in Figure 3.5, we can observe systematic errors when optimizing without the intrinsics (blue). The residuals should be normal-distributed around zero, but

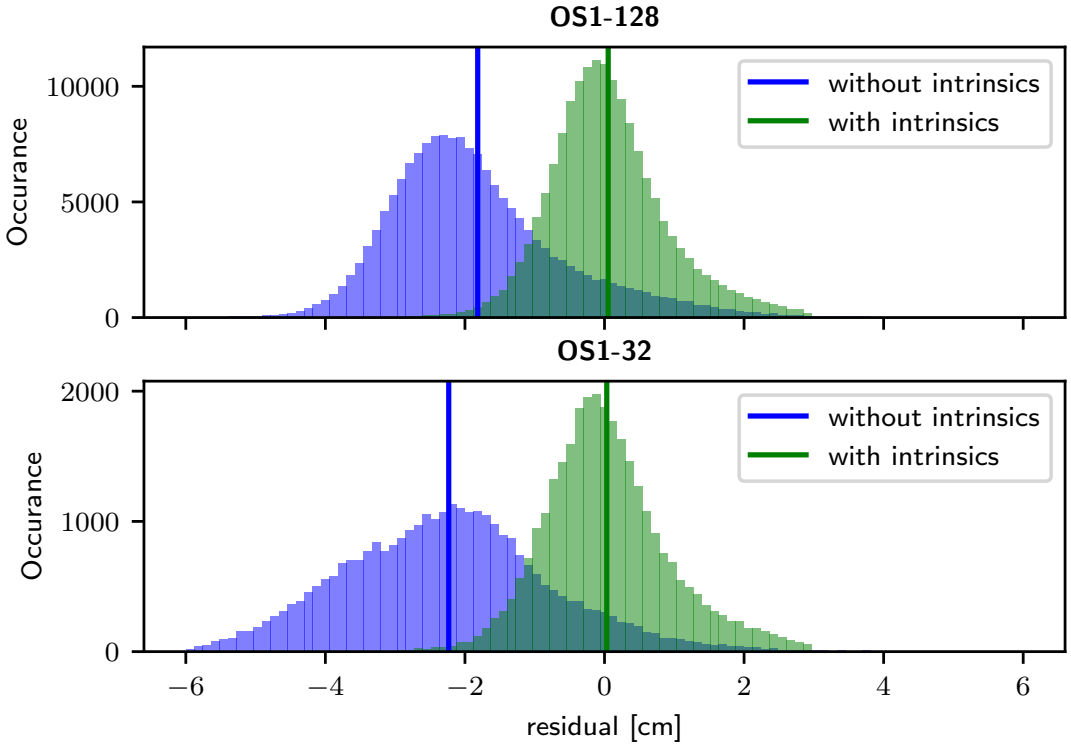


Figure 3.5: Histograms of point-to-plane residuals between the LiDAR points and the reference map after adjustment with and without estimating the LiDAR intrinsics (range scale and offset). The vertical lines denote the mean. Without calibrating the range measurements, we can see a constant offset of around 2 cm; both LiDAR sensors underestimate the range.

both seem to systematically measure around 2 cm too short. Additionally, the distributions of the residuals are not completely symmetric. When optimizing with the intrinsics (green), the residuals look normal-distributed around zero, therefore, indicating that no further systematics (like beam-wise intrinsics) are needed.

Different image distortion parameterizations are depicted in [D] - [H], where model B depicts Brown's distortion model and D the division model as discussed in Section 3.2.2.1. The degree column denotes the degree of the polynom used to model the radial distortion. In summary, the first-degree polynomial is substantially worse than the second or third polynomial. Brown's and the division model evaluate quite similarly for the same degrees.

3.4.4 Evaluation on Synthetic Data

To validate our methodology, we evaluate our approach on a synthetic dataset. This enables ground-truth reference parameters to which we can compare. For generating realistic synthetic data, we utilize the terrestrial laser scan by render-

Table 3.3: Synthetic dataset: RMSE of the parameters

Sensor	Parameter	CV2O3D	Ours
Cameras	Translation [mm]	4.11	0.79
	Rotation [°]	0.255	0.010
	Principal Point [pix]	3.74	0.60
	Focal Length [pix]	1.32	0.31
	Distortion [pix]	0.94	0.11
LiDAR	Translation [mm]	8.32	0.85
	Rotation [°]	0.524	0.010
	Bias [mm]	N/A	2.82
	Scale	N/A	0.0011

ing images and LiDAR scans from the dense point cloud (see Figure 3.4b) given a predefined set of poses, extrinsics, and intrinsics of all the sensors. Those values were chosen to be like [H] from Table 3.2 to have a realistic parameter set and trajectory. We add 2 cm of isotropic Gaussian noise to the points of the LiDAR scan. In Table 3.3 the RMSE's of the individual parameters with respect to the ground-truth parameters over all cameras and LiDAR sensors are displayed. Our approach is able to outperform CV2O3D, the best-performing baseline in the previous experiments. This shows the advantage of a combined adjustment over an individual calibration.

3.4.5 Calibration of Different Perception Systems

To show the versatility of our system, we show the calibration results for different perception systems with different sensors and configurations. For this, we provide quantitative results in the form of the analytical covariances for the relative and absolute poses, as well as qualitative results to provide a more intuitive way to see how well the sensors are calibrated and the observations are aligned to the map. The analytical standard deviations of the relative poses between the sensors, i.e., their extrinsic, as well as the standard deviation of the absolute poses are shown in Table 3.4. Both show that the translation can be estimated with below millimeter accuracy, while the rotation angles have standard deviations of around 0.06°. Propagating these errors into the image leads to errors with around 2.6 pix standard deviation, which is in line with the measured 2.51 pix RMSE from Section 3.4.2; indicating that our system can obtain realistic covariances. Qualitative results can be found in Figure 3.6. For each sensor, the observations from one timestamp can be seen in the calibration environment. The point clouds from the LiDAR are well aligned with the walls. The camera observations of the

Table 3.4: Standard deviation of the relative and absolute poses

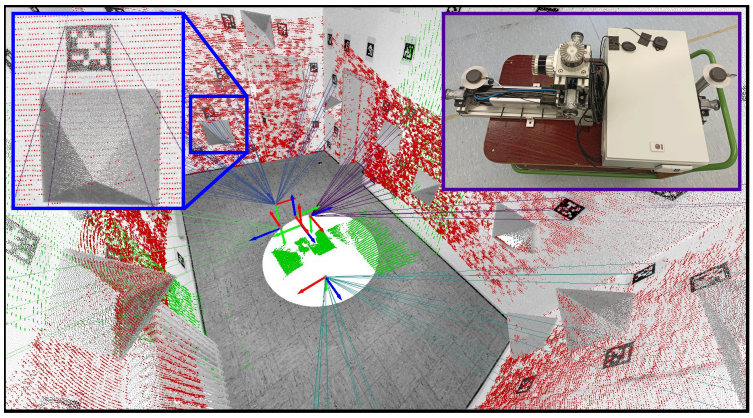
Pose	Platform	x [mm]	y [mm]	z [mm]	rx [°]	ry [°]	rz [°]
relative	IPB Car	1.26	1.24	1.04	0.0575	0.0578	0.0574
	Youbot	1.06	1.06	1.06	0.0583	0.0594	0.0585
	Dingo	1.07	1.06	1.08	0.0592	0.0634	0.0601
absolute	IPB Car	1.01	1.0	1.02	0.0575	0.0578	0.0574
	Youbot	1.03	1.03	1.06	0.0585	0.0592	0.0574
	Dingo	1.04	1.03	1.03	0.0597	0.0621	0.0574

AprilTag coordinates are visualized by the corresponding ray in the reference map frame. Furthermore, the camera rays intersect the AprilTag coordinates of the reference point cloud map M. Note that for the estimation, not only the observations from one timestamp but from around 50 timestamps at different positions are used for a reliable estimation.

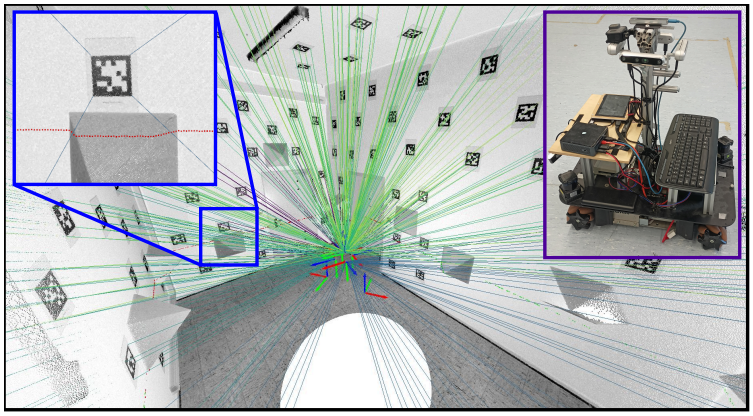
3.5 Discussion

In this section, we briefly want to discuss the advantages and disadvantages of our proposed method, as well as possible future research directions that can emerge from here. The main disadvantage we can see is that the setup can be costly; we rely on a precise high-resolution point cloud obtained by a terrestrial laser scanner (but needed only once) and have, in the best case, a dedicated room that we can modify to be a good calibration environment. A great advantage however is that once the calibration environment is prepared, the calibration does not require any special knowledge, and the whole process can be completely automated. One interesting future direction is to investigate how to reduce the costly hardware without compromising on the calibration quality.

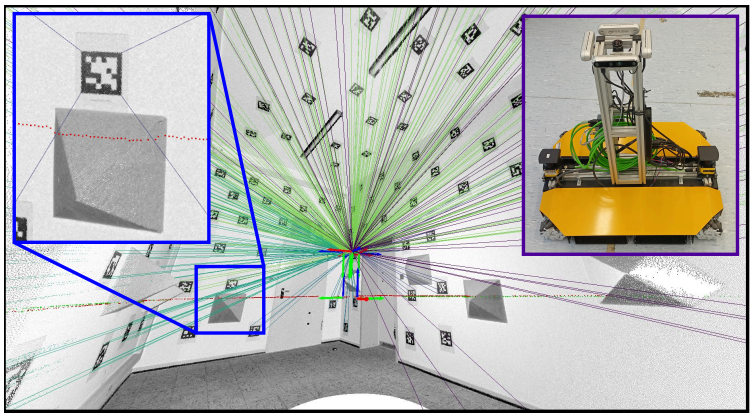
In our presented work, we focus on calibrating perception sensors. Odometry sensors, IMU, or GNSS are harder to incorporate into the pipeline since our approach relates the measurements to the reference map and not between the poses of the system at different timestamps. Incorporating an odometry sensor, or IMU, probably requires the integration of the measurements between two timestamps and relating the measured movement to the poses of the system, while when incorporating GNSS into the optimization, a globally referenced outdoor environment is required. An additional, problem with GNSS is the accuracy of the sensor observations and the correlations between the measurements due to atmospheric influence and multi-path. Modeling the errors and correlations suf-



(a) IPB-car



(b) Youbot



(c) Dingo

Figure 3.6: Visualization of the LiDAR scans and image rays to the reference map for different sensor setups. (a) IPB Car is a roof-top mount equipped with 4 Basler Ace wide-angle cameras and 2 Ouster OS1 multibeam LiDAR. (b) The Youbot is a ground vehicle equipped with 2 Hokuyo UTM-30LX profile scanners and one Realsense T265 that has 2 fisheye lenses. (c) Our robot "Dingo" is equipped with 2 SICK TIM781S profile scanners and one FLIR Blackfly S fisheye camera. Both, the Dingo and the Youbot have 4 Realsense D435 facing front, left, right, and rear. The point clouds are well aligned with the reference scan. The camera rays corresponding to the detected pixels as corners of the AprilTag intersect the corners of the reference point cloud.

ficiently to obtain millimeter-accurate extrinsics is very challenging. Errors due to wrong assumptions or modeling can propagate into the estimation of the other parameters, potentially degrading the calibration quality of all sensors. Therefore, we decided to not include the GNSS calibration in the optimization and rather rely on independent manual measurements.

3.6 Conclusion

In this chapter, we presented an approach for calibrating the intrinsics and extrinsics of perception sensors for robotic systems. The main idea is to exploit a precise reference map from a calibration room as a common target for all the sensors. This enables calibration of perception sensors that have limited or even no overlapping FoV. We equipped the environment with structural elements and uniquely identifiable targets to ensure correct correspondences and resolve ambiguities for the calibration. A joint least-squares adjustment of all the sensor observations is used to estimate the statistically optimal solution. This allows us to successfully calibrate different multi-sensor systems. We calibrated different modalities of multibeam LiDAR, profile scanners, wide-angle cameras, as well as fisheye cameras. For evaluating camera-to-LiDAR calibration, we propose an independent method to compare different calibration approaches. Our experiments show that our proposed approach provides accurate extrinsic and intrinsic calibration. We estimated the parameters with millimeter and sub-degree accuracy.

In the grand scheme of this work, the ability to obtain accurate calibration parameters is a necessary prerequisite for all the upcoming tasks, since we always work with sensor data. This makes the calibration of the sensors indispensable. Once we have our system calibrated, we can tackle the upcoming challenges and take our robot out and map the environment. Our final goal is to localize in the preconstructed map using the live LiDAR data. Potentially, multiple robots, with different sensors and different tasks have to be localized in the map. Having a calibration system that can handle different kinds of sensor systems and configurations allows for a unified calibration procedure without relying on different calibration methods for each individual setup.

In the next chapter, we will discuss our method for constructing a globally aligned point cloud map. We will utilize a similar joint optimization of the LiDAR data for the point cloud alignment as the one used in this chapter for the calibration. However, due to the absence of a highly accurate target map (here we used the TLS for this), we have to match the LiDAR point clouds with each other instead. Additionally, in the following chapter, we have to deal with the ego-motion of our vehicle while scanning, and the substantially larger amount of data.

Chapter 4

LiDAR Bundle Adjustment for High Accuracy Simultaneous Localization and Mapping

Constructing precise and consistent 3D maps is crucial for localization, surveying, monitoring, or the generation of digital twins. Mobile mapping systems are often used for constructing models in urban environments. Those systems are usually equipped with 3D LiDAR sensors that scan the vehicle's or robot's surroundings, obtaining each second millions of points sampled from the surfaces. Moving with a LiDAR sensor through a scene allows for capturing large areas in a relatively short time. The sensors however provide point cloud data in their own local coordinate frames, such that the pure agglomeration of the raw point measurements does not lead to a globally aligned map. To obtain such a globally consistent point cloud map, one needs to register and align all LiDAR scans, and thus transform each point into a unified coordinate frame. In other words, we need to know the time-aligned trajectory of our LiDAR sensor to account for the ego-motion when constructing a global point cloud.

In the previous chapter we have calibrated the sensors of our mobile mapping system, allowing us to drive through the environment and obtain consistent and undistorted measurements. This chapter investigates the problem of registering large sets of LiDAR scans into a globally consistent point cloud map. This is the second step towards localizing in a recorded map as depicted in Figure 4.1. In the later chapters, we will first utilize such global point cloud maps to compute a more memory efficient map representation, which we can afterward use to localize our robots in.

The alignment of two point clouds with each other is a well-studied problem in literature, where the most prominent method is the iterative closest point (ICP) algorithm [21] and its variants [36, 193, 201]. For building large scale maps,

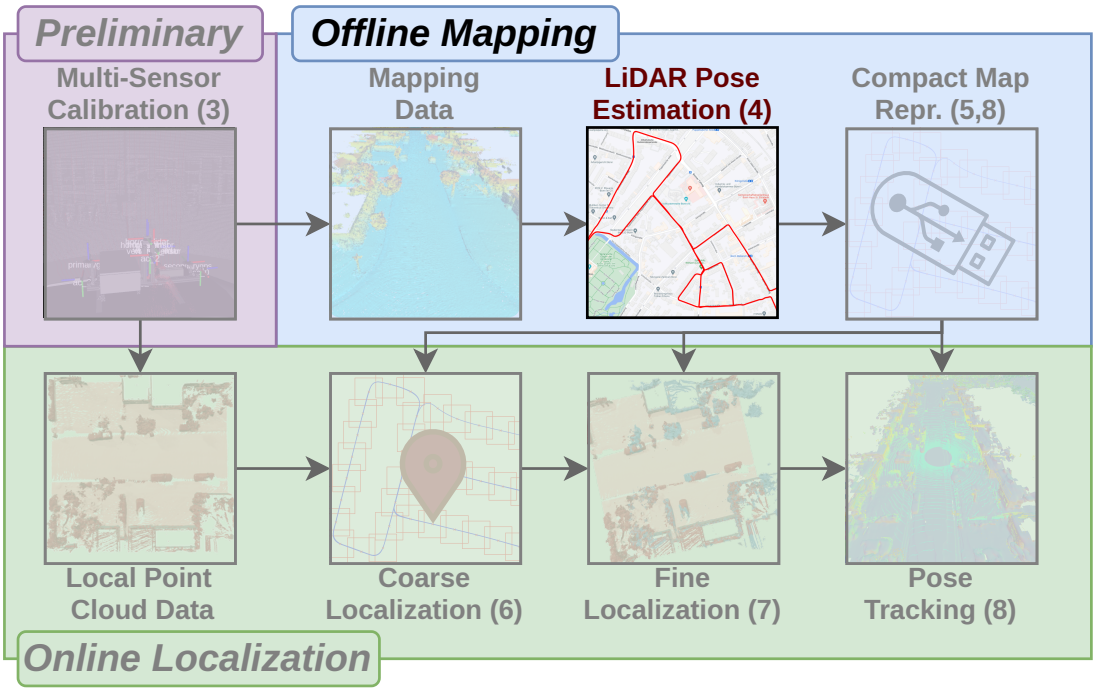


Figure 4.1: Overview of the thesis. In this chapter, we want to process the raw mapping data to generate a globally aligned point cloud map. For this, we need to estimate the trajectory of the LiDAR, which we solve by LiDAR bundle adjustment.

two scans are not sufficient, but rather thousands to millions of scans need to be registered. So instead of doing the registration once, one can apply ICP to successive scans, which is often referred to as LiDAR odometry [206, 243, 247]. This usually provides relatively well-aligned point clouds for short to medium-sized sequences. However, odometry systems suffer from drift, often leading to bad globally aligned maps. To overcome this problem, one can incorporate loop closures, which leads to a simultaneous localization and mapping (SLAM) system, or exploit global positioning information, e.g., GNSS data, using a pose graph. The optimization skews the whole trajectory to reduce the overall alignment error. This usually leads to globally well-aligned maps but may lack local consistency, as the pose graph redistributes the errors along the whole map proportional to the uncertainty of the poses. One can try to increase the uncertainty for the erroneous poses, but this requires knowledge about which pose or through which observations the error was introduced, however, this information is usually not available.

LiDAR bundle adjustment and some of the offline SLAM methods estimate the trajectory jointly with the map, trying not only to reduce the global alignment error but also to keep the map locally consistent. The big advantage over online SLAM and odometry systems is that LiDAR bundle adjustment does not have to work incrementally, but can utilize all the available information at once. LiDAR

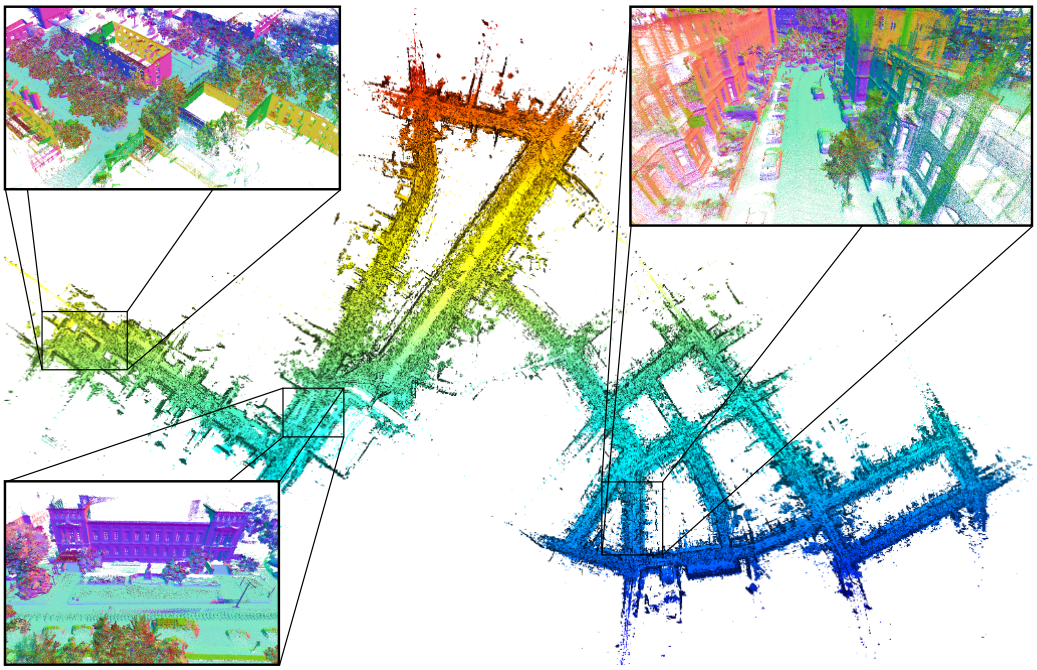


Figure 4.2: We propose a LiDAR bundle adjustment approach for aligning multiple point clouds. As shown here, our approach is able to generate a globally and locally aligned point cloud map. The color of the full point cloud is based on the latitude, while the points of the close-ups are colorized based on the direction of the normals.

bundle adjustment does not operate in real-time due to the large computation needed to find data associations, and for optimizing the map and poses globally. However, for several applications, the quality of the resulting map is the main outcome and far more important than the runtime. In this thesis, we want to have a precise globally and locally consistent map. Since, our goal is to store the map and use it for later for localization, it is less crucial how long the construction takes, and we can focus on obtaining a more precise solution.

> *"Rome was not built in one day."* (Medieval Phrase) – Nor can Rome be mapped in one day.

Scanning large-scale environments can often not be performed in one session. Either requiring to record multiple times or having multiple robots to map the whole area. Processing data from different recordings poses a challenge for many algorithms that try to align multiple point clouds. It leads to a discontinuous trajectory and can lead to local inconsistencies through temporal changes of the environment that need to be considered. Many algorithms exploit the fact that the LiDAR sensor is measuring continuously, and thus the scans are spatially and temporally adjacent. However, this assumption does not hold when having data from multiple recordings. Our goal is to provide a method that can handle data from multiple recording sessions to obtain a joint global map.

The main contribution of this chapter is a LiDAR bundle adjustment approach to globally align a large set of point clouds targeting high-quality reconstructions. The final goal is to obtain a geo-referenced point cloud map as shown in Figure 4.2. Starting from a set of scans and initial pose estimates, our approach tries to align all the point clouds with each other. We jointly optimize the trajectory from thousands of point clouds, resulting in a single big least squares adjustment. Utilizing a continuous-time trajectory allows us to model the motion of the sensor and the motion distortion of the scans without the need for odometry or IMU, nor do we require synchronized triggered sensors. Thus, each individual laser beam in a scan will be treated according to its measurement time. The usage of an out-of-core circular buffer and pruning the search space of correspondences allows us to run our approach on thousands of point clouds. Our approach is able to perform single-session as well as multi-session alignment.

Overall, we propose a method that (1) estimates a continuous-time trajectory of a 3D LiDAR scanner using LiDAR bundle adjustment, (2) which allows for the efficient construction of high-quality large-scale maps, and (3) is able to handle multi-session alignment to obtain a unified global map.

4.1 Related Work

Point cloud alignment is a well-studied problem in literature. Most approaches that tackle the problem are based to some extent on the ICP algorithm [21]. ICP estimates the transformation between two point clouds iteratively by a set of point correspondences, where in each iteration the correspondences are updated by searching for each point the closest point in the other point cloud. The classical point-to-point ICP [21] requires sometimes many iterations to converge, which is why different error metrics [179], starting from point-to-plane [36] up to entity-to-entity [201] relations, have been proposed to speed up the optimization. The closest point assumption relies on a good initial pose estimate to find reliable correspondences, which are necessary to converge to the correct solution. Feature-based correspondence search can provide reliable data associations and can make them more robust against the quality of the initial guess [64, 105, 111, 174]. However, the problem with feature-based matching is finding a suitable feature extractor that works reliably in different environments and under different conditions. Therefore, our approach uses the more classical closest point assumption for all 3D points to be more independent of the sensor data.

Searching for correspondences is often the most time-consuming and also the most difficult part. Acceleration structures, like hash maps [163, 168, 243], voxel grids [255], octrees [216], or KDTrees [291], as well as projective correspondence search [13, 110, 161, 256] are commonly used to speed up the search time.

Mobile mapping systems often utilize modern multibeam LiDAR sensors that can provide point cloud data of the surrounding environment at high frequency. Building a globally aligned map requires registering the LiDAR scans to each other. Sequentially aligning each incoming LiDAR scan to the previous one allows results in an odometry-like estimate, which therefore is often referred to as LiDAR odometry. Obtaining reliable correspondences between successive scans is unlikely due to the sparsity of the measurements. Therefore, it has been proven beneficial to keep some sort of local map [13, 243, 291] of the environment to which the new incoming frames are registered, and by this the poses are estimated. Many different types of map representations have been investigated: point-clouds [243, 291], voxels [161], normal distance transforms [22, 42, 54, 195, 217], surfels [13, 53, 171, 219, 252], and implicit representations [48, 168].

LiDAR odometry estimates only the latest pose based on the previous observations, however, it does not update the former pose estimates. The problem with any kind of odometry source, is that the small estimation errors accumulate and the poses start drifting. Simultaneous localization and mapping (SLAM) tries to tackle the problem by also updating the previous poses, which is especially valuable after revisiting a place, a so-called loop closure, to reduce the impact of the drift substantially [84, 108]. A common representation for the SLAM problem is to formulate it in a pose graph [44, 73, 74, 120, 137] where the poses are nodes and constraints between the poses as edges in a graph. Odometry sources would be edges between consecutive nodes, while loop closure constraints ensure global consistency between nodes of revisited places. The underlying system can be formulated as a least squares problem where a lot of research focuses on solving this efficiently [74, 76, 167]. This is especially important for online SLAM which tries to solve it incrementally, often even directly on the robot, which requires the operation at sensor frame rate. Offline SLAM approaches try to estimate the map and trajectory jointly using all available information [41, 228].

One important thing to consider is that the LiDAR measures while in motion which leads to a distortion in the scans, especially when moving at higher speeds. For deskewing the scans, one can use a motion model [243, 291], measure the sensor's motion using IMUs [205, 272], or estimate a continuous-time trajectory [45, 53, 171, 172].

Classical photogrammetric bundle adjustment [232, 200] utilizes images to jointly estimate poses as well as the 3D points. Similarly, LiDAR bundle adjustment [17, 20, 133, 191] aims at globally aligning a set of point clouds into a consistent map. Our approach belongs to this class of methods and tries to optimize the whole trajectory jointly with the map. In contrast to pose graph-based SLAM approaches, LiDAR bundle adjustment tries to directly minimize the alignment error

between the observed points. The foundation of jointly aligning multiple point clouds with respect to each other dates back several decades [17, 137]. Benjemaa and Schmitt [17] propose a joint alignment of multiple point clouds by minimizing the point-to-point error metric for all overlapping areas. This is the natural extension of the ICP algorithm [21] for registering multiple point clouds and providing the fundamental principles of solving the multi-scan alignment problem. We follow the core ideas of this approach: finding correspondences between overlapping point clouds, minimizing the distances between the correspondences in a joint least squares adjustment, and repeating the process iteratively until convergence. The problem with the original approach is regarding the scalability. It was developed to align around ten point clouds, but not for aligning thousands of scans that nowadays modern LiDAR can easily produce. Since each point cloud is matched with each other, the method scales badly with the number of point clouds. To overcome the problem, we propose to only associate each scan to a few other scans to reduce the matching complexity from quadratic to linear. Furthermore, the approach from Benjemaa and Schmitt [17] was not designed to deal with the motion distortion introduced by the ego-motion of the LiDAR scanner. We tackle the skewing of the moved LiDAR scans by modeling the sensors' ego-motion using a continuous time trajectory.

Recent approaches like BALM [133] are developed to handle larger amounts of scans. Instead of directly matching the scans, BALM [133] utilizes a feature map of planar and line features that are optimized along the scan poses. Having an external map representation bypasses the vast amount of matching, but requires associating the scan points to the features. Similarly, Li et al. [128] also tries to estimate the surface of the scene, but instead of approximating it using planes and lines, they utilize local second-order polynomials. By doing so they can approximate areas with curvature more accurately. The advantage of utilizing such external map representations is that the estimated surfaces can smooth out the noise of the point clouds. We decided against using an external map representation since it always makes assumptions about the environment. Approximating large smooth objects, e.g., streets or walls, with planes or polynomials might be appropriate. But especially in natural environments with fine detailed or uneven objects like trees, it remains an open question how well those local assumptions still hold. Directly trying to align the point clouds does not make any of those assumptions and represents the actual measurements. Another approach that directly tries to optimize the point cloud alignment without any additional map was proposed by Di Giammarino et al. [50]. Their photometric bundle adjustment aims to minimize not only the geometric properties but also a photometric error using the intensity channel of the LiDAR. To find fast correspondences between the point clouds, projective data associations are made.

This exploits the scanning pattern of the multibeam LiDAR sensors by utilizing a range image-based representation. The main drawback of the approach is the amount of RAM needed for larger sequences. To tackle the problem, only a subset of scans, so-called key poses, are used for optimization. We instead utilize out-of-core methods which allow us to align an arbitrary amount of point clouds without running into RAM shortage. A different way to handle the vast amount of data was shown by Liu et al. [129], which tackle the problem in a divide-and-conquer manner. They split the trajectory hierarchically, reducing the large joint optimization problem to aligning multiple smaller trajectories. The main problem with hierarchical splitting is to tune the trade-off between local consistency and global consistency accordingly.

4.2 LiDAR Bundle Adjustment

4.2.1 Problem Definition

Modern 3D LiDAR sensors, e.g., rotating multibeam LiDAR, provide point clouds of the sensor's surroundings. Assuming a dataset $\mathcal{D} = \{(\mathsf{P}_i, \boldsymbol{\tau}_i)\}$ of N_{scans} point clouds $\mathsf{P}_i \in \mathbb{R}^{N_i \times 3}$ with the timestamps $\boldsymbol{\tau}_i \in \mathbb{R}^{N_i}$, where each point cloud P_i consists of N_i points $\boldsymbol{p}_j \in \mathbb{R}^3$ and $\boldsymbol{\tau}_i$ of their corresponding timestamps $t_j \in \mathbb{R}$ at which the points are measured. Each point $\boldsymbol{p}_j$ is located in the local coordinate frame of the LiDAR at timestamp t_j. To globally align the point clouds P_i, we need to find the transformations $\mathcal{T} = \{(\mathsf{R}_{t_j}, \boldsymbol{t}_{t_j}) \mid \forall j\}$ given by the rotations $\mathsf{R}_{t_j} \in SO(3)$ and translations $\boldsymbol{t}_{t_j} \in \mathbb{R}^3$ for each beam that transforms the point $\boldsymbol{p}_j$ from the local coordinate frame at timestamp t_j into a global coordinate frame:

$$\hat{\boldsymbol{p}}_j = \mathsf{R}_{t_j} \boldsymbol{p}_j + \boldsymbol{t}_{t_j}, \tag{4.1}$$

where $\hat{\boldsymbol{p}}_j$ denotes a point in the global coordinate frame. Point clouds from common LiDAR sensors, e.g., Velodyne HDL-64E, Ouster1-128, or similar, contain points measured at different points in time due to their measuring process. This requires either deskewing the scans (e.g., using a constant velocity model [243], or an estimate of the sensor's motion provided by an IMU [266]) or to estimate a continuous trajectory [45, 53] over time to actually model the continuous motion of the sensor in space. We utilize a continuous-time trajectory inspired by Dellenbach et al. [45] and interpolate the poses between the beginning and end of each scan using spherical linear interpolation (SLERP) for the rotation and linear interpolation for the translation. Applying the formulas for the continuous-time

trajectory results the transformations

$$R_{t_i} = \text{slerp}(R_{t_{b(j)}}, R_{t_{e(j)}}, \alpha_j), \tag{4.2}$$

$$\boldsymbol{t}_{t_i} = (1 - \alpha_j)\boldsymbol{t}_{t_{b(j)}} + \alpha_j \boldsymbol{t}_{t_{e(j)}}, \tag{4.3}$$

where α_j is the fraction of time between the start $t_{b(j)}$ and end $t_{e(j)}$ of the corresponding scan for a given point $\boldsymbol{p}_j$:

$$\alpha_j = \frac{t_j - t_{b(j)}}{t_{e(j)} - t_{b(j)}}. \tag{4.4}$$

In contast to Dellenbach et al. [45], we assume the end pose of one scan to be the start pose of the successive scan, which results in a C_0 continuous trajectory. The objective function that we try to minimize is the squared distance of all corresponding points between each scan $\mathcal{D}$

$$E = \sum_j \sum_{\hat{\boldsymbol{p}}_c \in C(\boldsymbol{p}_j)} d(\hat{\boldsymbol{p}}_j, \hat{\boldsymbol{p}}_c)^2, \tag{4.5}$$

where $d(\cdot)$ is a distance function (like point-to-point or point-to-plane), and $C(\boldsymbol{p}_j) = \{\boldsymbol{p}_c \mid \hat{\boldsymbol{p}}_c \equiv \hat{\boldsymbol{p}}_j\}$ is the set of points that correspond to the same location as the point $\boldsymbol{p}_j$. For the point-to-plane distance function, we yield an error function e_j for an arbitrary point $\boldsymbol{p}_j$ as

$$e_j = \hat{\boldsymbol{n}}_c^\top (\hat{\boldsymbol{p}}_j - \hat{\boldsymbol{p}}_c), \tag{4.6}$$

where $\hat{\boldsymbol{n}}_c = R_{t_c}\boldsymbol{n}_c$ is the normal of the point $\boldsymbol{p}_c$ computed based on the local neighborhood within its scan. The derivatives[1] [45] for the transformation of point $\hat{\boldsymbol{p}}_j$ are given by

$$\boldsymbol{J}_{R_{t_{b(j)}}} = -(1 - \alpha_j)\hat{\boldsymbol{n}}_c^\top R_{t_{b(j)}} [\boldsymbol{p}_j]_\times \tag{4.7}$$

$$\boldsymbol{J}_{R_{t_{e(j)}}} = -\alpha_j \hat{\boldsymbol{n}}_c^\top R_{t_{e(j)}} [\boldsymbol{p}_j]_\times \tag{4.8}$$

$$\boldsymbol{J}_{\boldsymbol{t}_{t_{b(j)}}} = (1 - \alpha_j)\hat{\boldsymbol{n}}_c^\top \tag{4.9}$$

$$\boldsymbol{J}_{\boldsymbol{t}_{t_{e(j)}}} = \alpha_j \hat{\boldsymbol{n}}_c^\top. \tag{4.10}$$

The expression $[\cdot]_\times$ is the skew-symmetric matrix of a vector. Note that we estimate the rotation increment as an axis-angle representation. The derivatives of the error by the transformation parameters of the corresponding point

[1] Formulas from the official implementation [45].

$T_{t_c} = \{R_{t_c}, \boldsymbol{t}_{t_c}\}$ can be derived analogously, i.e.,

$$\boldsymbol{J}_{R_{t_{b(c)}}} = (1 - \alpha_c)\hat{\boldsymbol{n}}_c^\top R_{t_{b(c)}} [\boldsymbol{p}_c]_\times \tag{4.11}$$

$$\boldsymbol{J}_{R_{t_{e(c)}}} = \alpha_c \hat{\boldsymbol{n}}_c^\top R_{t_{e(c)}} [\boldsymbol{p}_c]_\times \tag{4.12}$$

$$\boldsymbol{J}_{\boldsymbol{t}_{t_{b(c)}}} = -(1 - \alpha_c)\hat{\boldsymbol{n}}_c^\top \tag{4.13}$$

$$\boldsymbol{J}_{\boldsymbol{t}_{t_{e(c)}}} = -\alpha_c \hat{\boldsymbol{n}}_c^\top. \tag{4.14}$$

Each error equation given by Equation (4.6) contributes to the poses from two scans in the optimization. Since generally only a small set of point clouds overlap, the resulting normal equation system is relatively sparse. This allows us to utilize sparse matrices and operations to solve the otherwise enormous equation system.

In reality, we do not know which points correspond to each other, nor can we even ensure that a corresponding point was measured. Therefore, we use the common assumption as in ICP and search for the closest point in a certain neighborhood $\mathcal{N}(\boldsymbol{p}_j)$. We do not only have two point clouds, thus we have to search for correspondences in multiple, and in the worst case, in all the point clouds. To reduce the compute requirements, we sample a set of scans in which we look for correspondences, as will be further explained in Section 4.2.2. Furthermore, we need a sufficiently accurate initial guess required for solving the non-linear least squares problem. In our experiments, having a LiDAR odometry system combined with loop closures or with a low-cost GPS for global alignment obtained a sufficient initial guess. To deal with outliers, e.g., caused by misalignment or dynamic objects, we can use a Geman McClure-Kernel to reduce their impact in the optimization.

4.2.2 Acceleration Strategies

Aligning all point clouds can be time-consuming. To tackle this problem, we utilize three strategies to speed up operations. First, instead of matching each point cloud P_i to all other point clouds, we randomly sample for each point cloud P_i, N_{matches} different point clouds P_k, within a radius τ

$$\mathcal{C}_i = \{P_k \in \mathcal{D} \setminus P_i \mid \|\boldsymbol{t}_i - \boldsymbol{t}_k\| \leq \tau\}, \tag{4.15}$$

and only search for point associations in this subset of point clouds. This allows us to reduce the complexity of the correspondence search from $\mathcal{O}(N_{\text{Scans}}^2)$ to $\mathcal{O}(N_{\text{Scans}})$.

Second, we utilize a voxel hash map similar to Nießner et al. [163] with a bucket size of one. We store in each voxel the index of the point closest to the voxel center. Saving the index allows us to also access the point attributes without occupying additional memory. The voxel hash map construction time is $\mathcal{O}(N)$,

and performing a radius neighborhood search can be done in $\mathcal{O}(1)$. Note that we have to construct the acceleration structure at each iteration of the optimization, since the local point distribution changes due to the deskewing through the continuous-time trajectory. Therefore, having an efficient acceleration structure like the voxel hash map is crucial. Additionally, the hash map can be implemented efficiently on the GPU to facilitate modern hardware accelerators.

Third, we subsample the point clouds. We utilize a grid-based subsampling, where we can utilize the aforementioned voxel hash map. Subsampling is a simple but efficient way to save on redundant points which does not add a lot of additional information [243, 291].

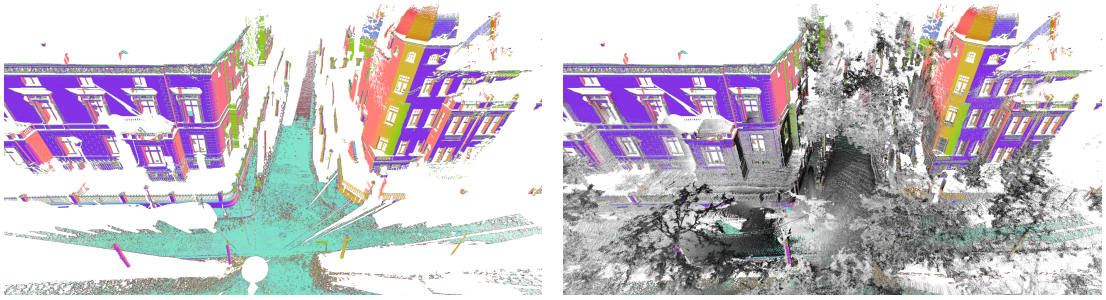
4.2.3 Memory Management

Storing all point clouds in memory is infeasible due to the sheer amount of data when considering realistic setups. Loading each point cloud on demand from disk is memory-efficient but slow since reading from RAM is usually way faster than from disk. We compromise by using a circular buffer, only keeping the last N_{buffer} used point clouds in RAM. Whenever a point cloud is required that is not in the buffer, we delete the scan from the buffer (but still keep it on disk) that was used last and replace it with the new scan. Utilizing such out-of-core processing allows for processing large amounts of data efficiently, without relying on large RAM resources. Due to sampling the point clouds based on the distance τ, it is quite likely to sample point clouds that are also close in time. Therefore, we iterate sequentially through the point clouds to reuse, as much as possible, the point clouds in the buffer before loading new ones in. Additionally, we reverse after each optimization step the iteration order of the point clouds, such that the end of the last iteration is now the beginning of the new one. By this, we exploit that in the end the buffer is filled with point clouds close-by to the start of the following sequence.

4.3 Experimental Evaluation

The main focus of this work is LiDAR bundle adjustment, where we want to optimize from a set of scans and initial poses the continuous-time trajectory that globally aligns the point clouds with respect to each other. We present our experiments to show exactly the capabilities of our method.

In the following experiments, we precompute the normals of the point clouds using the 30 nearest neighbors of each point in the scan. For each point cloud, we search in $N_{\text{matches}} = 10$ other clouds within $\tau = 30\,\text{m}$ radius for matches. We downsample the scans to a resolution of 15 cm while we use for the voxel



(a) TLS scan (b) LiDAR Scans aligned to TLS scan

Figure 4.3: (a) The ground truth for our dataset was generated using high-precision, globally referenced TLS scans. (b) Close-by LiDAR scans (gray) are aligned to the TLS scans to obtain reference poses. Note that we removed moving objects (cars, pedestrians) and movable objects (parked cars, foliage) manually from the TLS to allow precise scan alignment.

hash-map a grid size of 30 cm for finding the correspondences. We only look for the correspondences in the adjacent voxels, resulting in $3^3 = 27$ candidates. Furthermore, we use a buffer size of $N_{\text{buffer}} = 1{,}000$ point clouds, which practically results in the same runtime as loading all point clouds in memory. For the non-linear least squares adjustment, we stop at the latest after $N_{\text{iter}} = 100$ iterations. Usually, after around 10-30 iterations (depending on the quality of the initial alignment), the adjustment converges. All experiments were conducted with an NVIDIA RTX-A5000.

4.3.1 Results in Urban Environment

The first experiment evaluates the performance of our approach on our own collected dataset, which we will refer, from here on, as the IPB-Car dataset. Our setup consists of a car mount equipped with multiple sensors. We use a horizontally mounted Ouster1-128 that provides us with point clouds recorded with 128 laser beams at 10 Hz, and a low-cost GPS receiver to provide global reference. The dataset contains 11,702 scans with each up to $N = 128 \cdot 2{,}048$ points. We obtain the initial guess for all approaches by fusing the poses from KISS-ICP [243] as odometry and GPS as unity factors in a pose graph.

For evaluation, a ground truth for more accurate reference data is needed. For that, we selected key locations in the map, which we measured with a TLS. The TLS scans are globally referenced using precise static dual frequency GPS with long static recording per reference scan and using correction data from SAPOS to mitigate atmospheric errors, resulting in centimeter-accurate global reference poses with millimeter-accurate local accuracy. We align the recorded LiDAR scans to the TLS ground truth scans similar to Nguyen et al. [162] for precise

Table 4.1: Quantiative results on our recorded IPB-Car dataset

Approach	ATE [m] (trans)	ATE [°] (rot)	RPE [m] (trans)	RPE [°] (rot)
KISS-ICP	2.28	1.21	0.023	0.121
KISS-ICP + GPS	1.34	2.79	0.025	0.131
PIN SLAM	1.76	1.43	0.020	0.122
PBA	0.99	3.53	N/A	N/A
HBA	1.29	2.55	0.026	0.133
Our approach	**0.90**	**0.63**	**0.014**	**0.055**

reference poses, see Figure 4.3 for reference. In this dataset, the vehicle passed 8 times through reference locations. All vehicle LiDAR scans that are aligned to a TLS scan will be used for quantitative evaluation. As metrics, we will use the common absolute trajectory error (ATE) to obtain a measure for the global positioning accuracy, as well as the relative position error (RPE) between consecutive frames as a measure for local accuracy.

We also compare our approach against state-of-the-art baselines, here specifically against PIN-SLAM [168] an online SLAM approach, as well as the LiDAR bundle adjustment methods HBA [129] and PBA [50]. For completeness, we provide the results of KISS-ICP [243] solely as an odometry system and in a second variant fused with GPS measurements as an initial guess. The results of our collected dataset are shown in Table 4.1. Our approach is able to consistently outperform all baselines regarding ATE, as well as RPE. Our approach takes on average 45 min per iteration for the 11,702 scans, resulting in around two days of overall optimization time. Note that PBA [50] only estimates the poses for keyframes, which does not allow us to compute the RPE, since the metric is computed between consecutive poses. HBA [129] requires around 160 GB of CPU memory, while PBA [50] requires 43 GB GPU memory even though downsampling the poses. With our out-of-core ring buffer containing the last 1,000 scans, we only require 8 GB of GPU memory. Note that one can further reduce the memory consumption by reducing the buffer size, but trading off slower processing time. Just storing naively the full normal equation system without exploiting the sparsity would alone require 40 GB memory, while solving would take over 150 days.

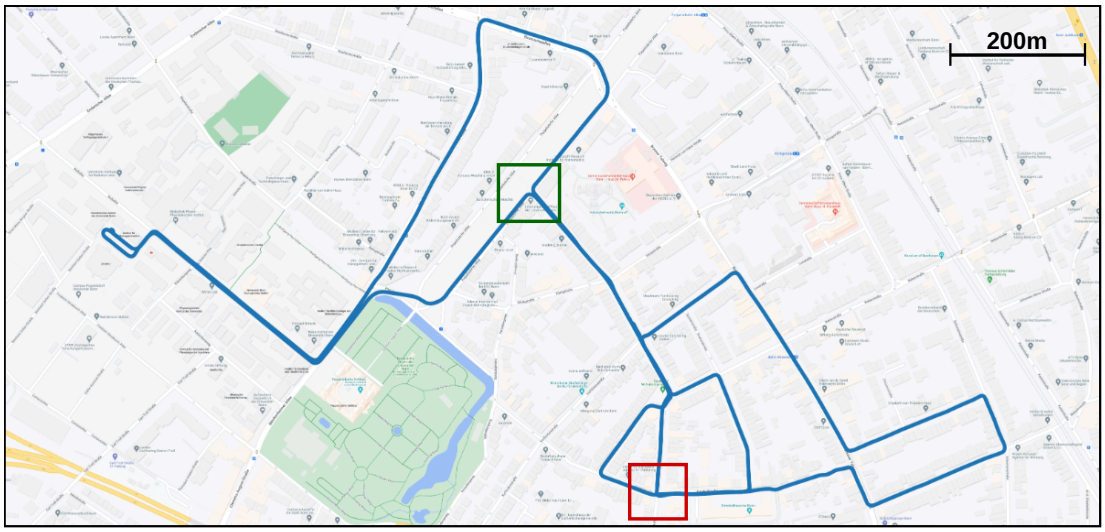


Figure 4.4: The estimated trajectory of our approach on the self-recorded IPB-Car dataset overlayed on Google Maps. The trajectory contains multiple loop closures, where some sections are mapped multiple times. This makes a global alignment necessary to obtain a consistent trajectory. The red and green rectangles indicate the location of the qualitative results in Figure 4.5.

4.3.2 Qualitative Results

Evaluating the accuracy of maps is a challenge, due to not having a ground truth model, nor having known correspondences. Therefore, to convey the quality of the estimated trajectory, we will show the resulting aggregated maps for qualitative evaluation only. In Figure 4.4, we show our estimated trajectory (blue), as well as a red and green rectangle, indicating the scenes shown in Figure 4.5. We visualize the aggregated point cloud map using the estimated poses of the approaches, where the color is coded based on the direction of the normals. In the green area the car is driving through a parkway with a lot of trees, resulting in bad GPS conditions. This leads to a translation error in the initial guess (KISS-ICP + GPS), as can be seen by the walls not being aligned. Starting from this, HBA is not able to resolve the local inconsistencies, while our approach successfully aligns the point clouds.

The area denoted by the red frame is in an urban environment with high, terraced houses, making it susceptible to GPS multipathing. Additionally, misalignment in a sharp turn caused blurry maps. HBA can only reduce the blurring slightly, while our approach is able to correct the pose errors and provide sharp walls.

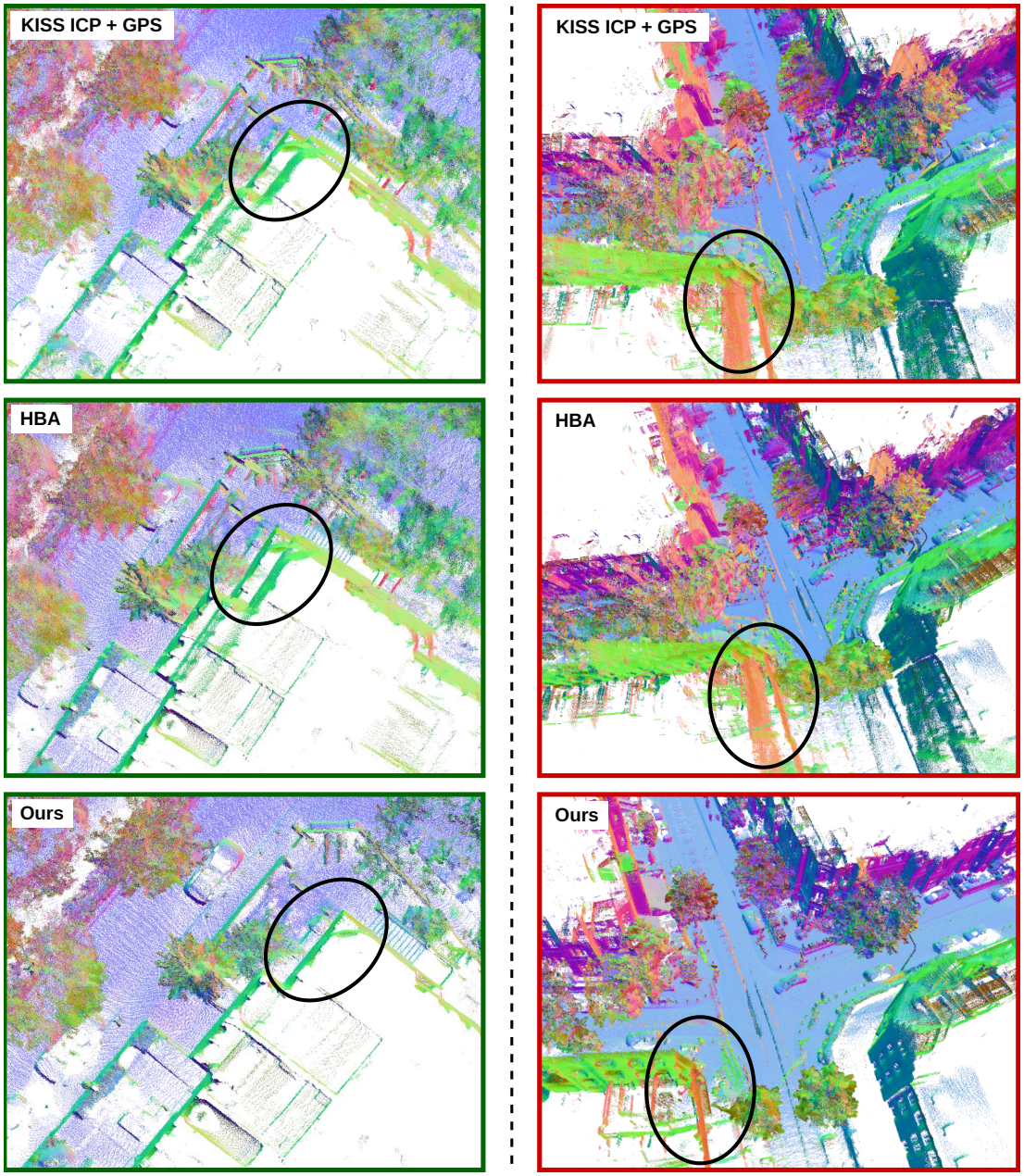


Figure 4.5: Shown are two close-up parts of the IPB-car dataset to allow qualitative evaluation. We obtain the initial guess for the methods using KISS-ICP fused with GPS data (left). Due to bad GPS conditions, the map is misaligned after closing longer loops, resulting in an inconsistent map. HBA can only slightly improve the local map quality, while our approach is able to obtain a crisp point cloud map. The points are colorized based on the direction of the normals.

Table 4.2: Quantitative results on the MCD [162] dataset

	Approach	ATE [m] (trans)	ATE [°] (rot)	RPE [m] (trans)	RPE [°] (rot)
NTU-day-1	KISS-ICP	7.21	3.54	0.133	1.056
	KISS-ICP + Loop	2.08	3.66	0.129	1.059
	PIN SLAM	1.34	2.20	**0.086**	1.069
	HBA	1.79	2.81	0.129	1.060
	Our approach	**1.03**	**1.75**	0.129	**0.599**
NTU-day-2	KISS-ICP	0.27	**0.86**	0.094	0.663
	KISS-ICP + Loop	0.35	1.64	0.092	0.665
	PIN SLAM	0.28	1.29	**0.063**	0.636
	HBA	**0.19**	0.87	0.092	0.665
	Our approach	0.20	0.99	0.083	**0.397**

4.3.3 Results on Campus-Scale Data

To validate our performance in a different environment under different conditions, we evaluate our approach on the Multi-Campus Dataset (MCD) [162] dataset. We use this dataset due to the high-quality ground truth which also utilizes precise TLS reference maps. Since our approach refines poses from, e.g., SLAM/odometry + GPS, comparing against ground truth generated with similar methods [63, 141] would not be significant, therefore using only datasets that have TLS reference. In contrast to our dataset, where the sensor is mounted on a car and the environment is in an urban environment, MCD [162] uses a handheld device in a campus environment. The ntu-day1 sequence of MCD [162] has 6,010 scans, while ntu-day2 is with 2,274 scans substantially smaller. Due to a lack of GPS reception, we use KISS-ICP [243] optimized with loop closures [83] using pose graph optimization as an initial guess. The results are depicted in Table 4.2. On the longer day1 sequence, our approach is able to estimate the most accurate trajectory in terms of ATE (translation and rotation), and RPE (rotation), as well as the second best regarding RPE (translation). On the shorter sequence, the quality of the estimated trajectories is more similar, since the drift is very small and the initial poses are already quite well aligned. Our approach provides throughout competitive results. The overall higher errors with respect to the IPB-car dataset can be explained by the higher motion profile due to holding the sensor in hand. Due to our linear interpolation, we assume constant velocity within the timeframe of one scan. It would be interesting to investigate higher polynomial interpolation in the future.

Table 4.3: Multi-Session alignment MCD-NTU day 1 & 2

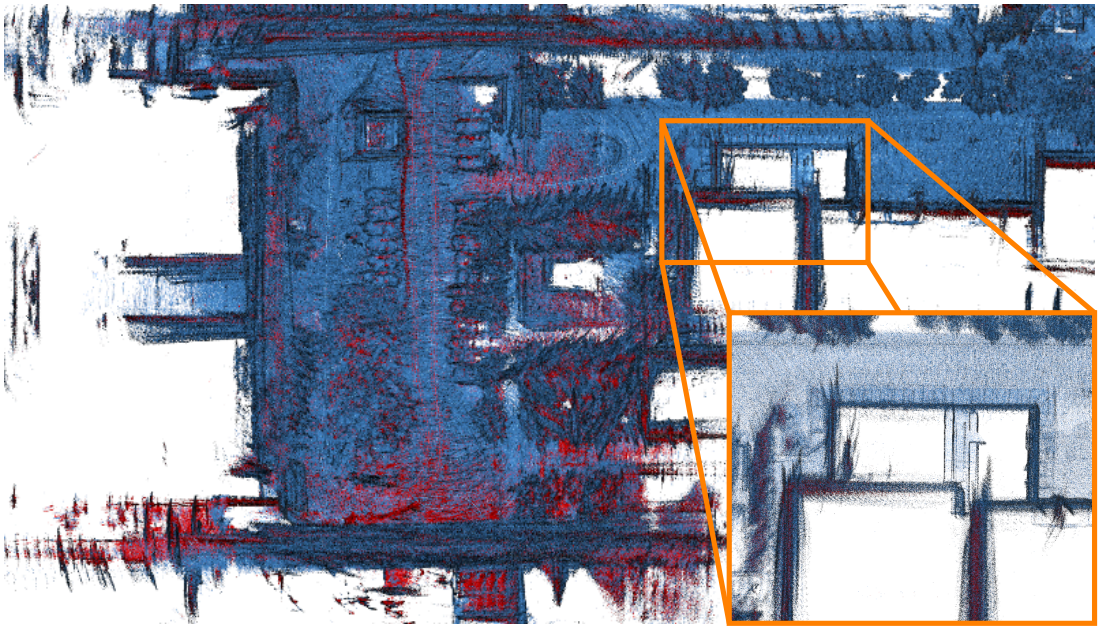
Approach	Inter RPE [m] (trans)	Inter RPE [°] (rot)
KISS-ICP + Loop	0.567	3.20
Our	**0.085**	**0.08**

4.3.4 Mutli-Session Alignment

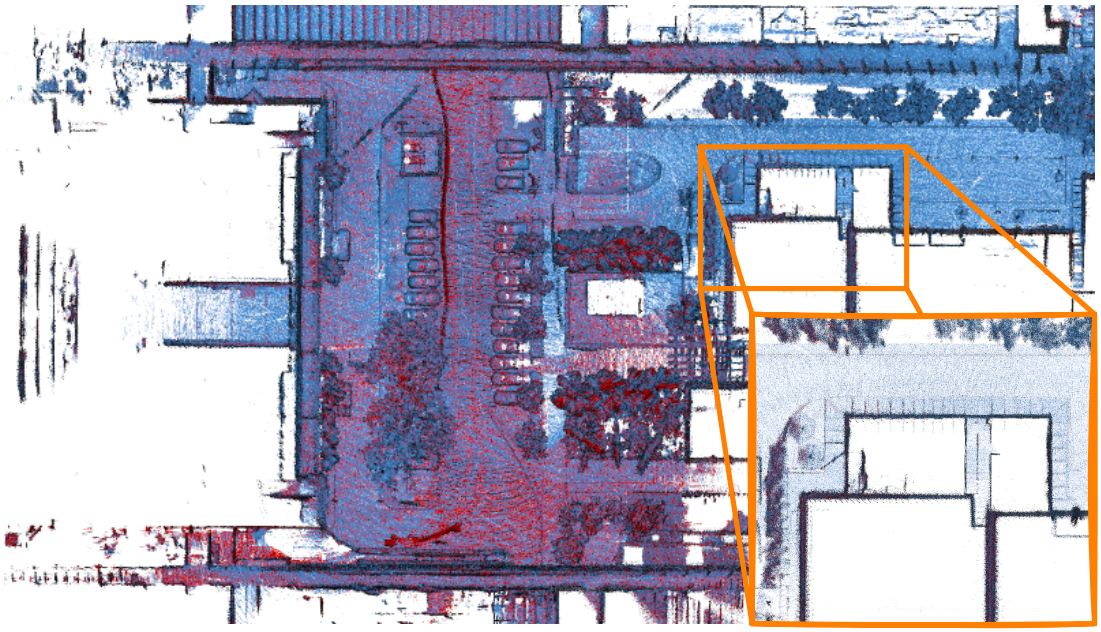
The MCD [162] dataset provides different runs in the same location on different days, enabling us to perform and test multi-session alignment. In this task, we want to align multiple scans from multiple sessions together jointly, such that they form a consistent map. This task is important when building maps of areas that cannot be recorded in one session. For the initial guess, we take the previously separately estimated KISS-ICP [243] poses and fuse them with loop closures between the sequences. In other words, combining LiDAR odometry, plus inter-loop closure within the first sequence and intra-loop closure from the second sequence to the first sequence in a pose graph. Our approach is able to align both trajectories jointly. The only thing one needs to deal with is the time discontinuity between the sessions, i.e., for the trajectory estimation, the end pose of the first session is *not* the start pose of the second session.

For quantitatively evaluating how well both trajectories are aligned to each other, we compute the relative transformation between each pose of the second trajectory to the corresponding pose in the first trajectory. This we do for the ground truth trajectories, as well as for the predicted trajectories, such that we have a set of relative transformations between the trajectories respectively. We compute the root mean squared error between both sets of relative transformations. This is similar to the RPE but between two different sessions instead, therefore we will denote it as inter-RPE. While the classical RPE computes the error between consecutive poses in one session, the inter-RPE computes the error between corresponding poses across two sessions. The correspondences are computed on the ground truth poses by taking for each pose of the second session the closest pose of the first session.

The results for the initial guess and for our approach are provided in Table 4.3. The initial pose-graph-based alignment using KISS-ICP plus loop closures is off by over half a meter and a few degrees. Our approach is able to reduce this initial alignment error to a few centimeters and sub-degree accuracy. For qualitative comparison, we refer to Figure 4.6. Although the trajectories of the initial guess are globally well aligned, the resulting aggregated map is noisy, especially due to errors in rotation. Our approach however is able to align the sessions, resulting in a joint, globally and locally aligned map.



(a) Initial guess



(b) Estimated alignment

Figure 4.6: For the multi-session alignment, we jointly optimize the scans from two sessions (Session 1: Blue, Session 2: red). (a) Depicts the initial guess, where one can see that the sessions are roughly aligned, but overall the map is quite noisy. In (b), we show the aggregated map after our alignment. Seeing that both the red and blue point clouds are well aligned; leading to crisp structures.

4.4 Conclusion

In this chapter, we presented a novel approach for LiDAR bundle adjustment. Our approach processes initial aligned LiDAR scans to compute a globally aligned point cloud map. The proposed method refines the pose estimates from LiDAR odometry or even SLAM systems on a scan level to obtain a more consistent and precise map, as well as trajectory. We estimate a continuous-time trajectory to account for the motion of the LiDAR sensor while scanning. For the optimization, we minimize directly the alignment error between the points of overlapping scans in a least squares fashion. Directly optimizing on a point level allows for generating precise poses and aligned point cloud maps. One of the major challenges for aligning large sets of scans together is the required computations. We tackle this problem by reducing the search space for potential correspondences between the scans within a certain proximity due to the finite range of the LiDAR sensors. Additionally, we utilize hash maps that allow for searching correspondences on a GPU in a reasonable time. The usage of circular buffers allows for scaling up the sequence length without running into memory problems. We implemented and evaluated our approach on different datasets with TLS-based, accurate ground truth information. This allows us to successfully estimate the trajectory of the 3D LiDAR sensor and provide well-aligned point cloud maps. Our approach is able to provide competitive results on different datasets. We show that our approach can provide accurate trajectories, tested on sequences with up to 11,700 scans with centimeter relative accuracy and below a meter global accuracy. Additionally, we show that our approach can be used to jointly align multiple sessions.

In this thesis, we want to localize our robots in preconstructed maps using LiDAR data. The method proposed in this chapter enables us to build these large-scale maps by aligning the raw local point clouds and aligning them into a global point cloud map. This globally aggregated map could theoretically already be used for localization. The main problem with using the whole point cloud as a map is the large amount of memory it requires. A quarter of a city can easily require hundreds of Gigabytes to several Terabytes of memory. If we want to localize a robot within the map, then it also needs access to the map. Storing those maps directly on the robot can exceed the available onboard storage quickly; at the same time the amount of data is also too large for downloading on-demand from a cloud service. Therefore, in the next chapter, we will discuss a way to compress those point cloud maps to obtain a more memory-efficient representation. Afterward, in Chapter 6 and Chapter 7 we investigate how to localize in those compressed maps.

Chapter 5

Deep Compression for Dense Point Cloud Maps

Maps are the foundation for robot navigation, involving tasks such as localization, planning, or scene understanding. Large-scale point cloud maps can be acquired using LiDAR sensors of modern mobile mapping systems as we have demonstrated in the last chapter. For utilizing those maps, the robot needs to somehow access the data: requiring either to store it on-board or downloading it on-demand. However, the pure amount of data poses challenges to both options. On-board storage is limited, costly, and requires physical space, which needs to be considered especially for smaller robotic systems. The other option would be to have the whole data stored on a server and only transfer the required parts to the robot on-demand. Although upcoming 6G networks will enable higher data transfer rates, the amount of data needed will exceed quite fast the available network capacity. Furthermore, the memory consumption is only one side of the medal: we do not only need to store the data, but also use it in our applications and algorithms. Processing these large-scale point clouds for downstream tasks is compute-intensive and algorithmically challenging. Therefore, representing the 3D data in a compact and structured way is key for efficient storage and processing.

In this chapter, we tackle the high memory demand of large-scale point cloud maps. For this, we investigate the usage of deep neural networks to compress the point clouds into a memory efficient representation. Our compact representation can not only be used for decompression to restore the point clouds, but in Chapter 6 and Chapter 7, we will also demonstrate the direct usage for localization. By the end of this chapter, we will have built all the building blocks to construct from the aggregated sensor observations a compact global map as illustrated in Figure 5.1.

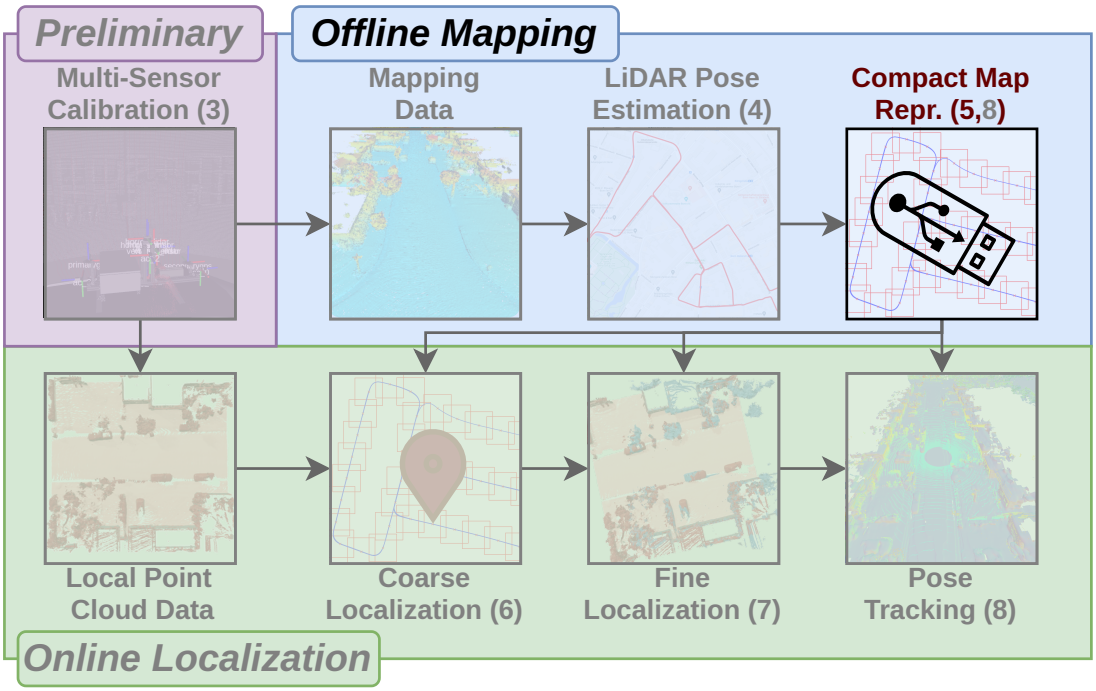


Figure 5.1: Overview of the Thesis. While the last chapter dealt with constructing large-scale point cloud maps, in this chapter we want to compress them, to enable efficient storage. Additionally, this compressed representation will be used in the following chapters for coarse and fine localization.

Representing 3D data in a memory-efficient way is a common problem in robotics and computer graphics and several approaches exist that tackle this challenge. Binary space partition approaches, like Octrees [150] or kD-trees [19], utilize a hierarchical data representation that allows for efficient storage and fast neighbor queries which is needed by many algorithms. Voxel grids [1] discretize the world enabling fast access, however, at the cost of incurring a large memory footprint. To overcome this problem, Octomaps [89] store voxels in a sparse octree structure, thereby reducing the memory requirements.

So-called 2.5 D representations [121, 231] make strong assumptions that often do not hold in natural scenes. Another technique is to use triangular meshes to approximate the environment or use multiple primitives, like planes, cylinders, or spheres [199], but such predefined geometries may not fully exploit the available potential of recurring structures. Learning-based methods enable capturing common characteristics of the scene for further compression [190]. Recent deep neural autoencoder networks [91, 95, 183] provide data compression for different domains and they have also been successfully used to compress 3D point cloud data. This work belongs to this class of approaches and proposes a novel architecture exploiting state-of-the-art representation learning for 3D point clouds to achieve high compression rates.

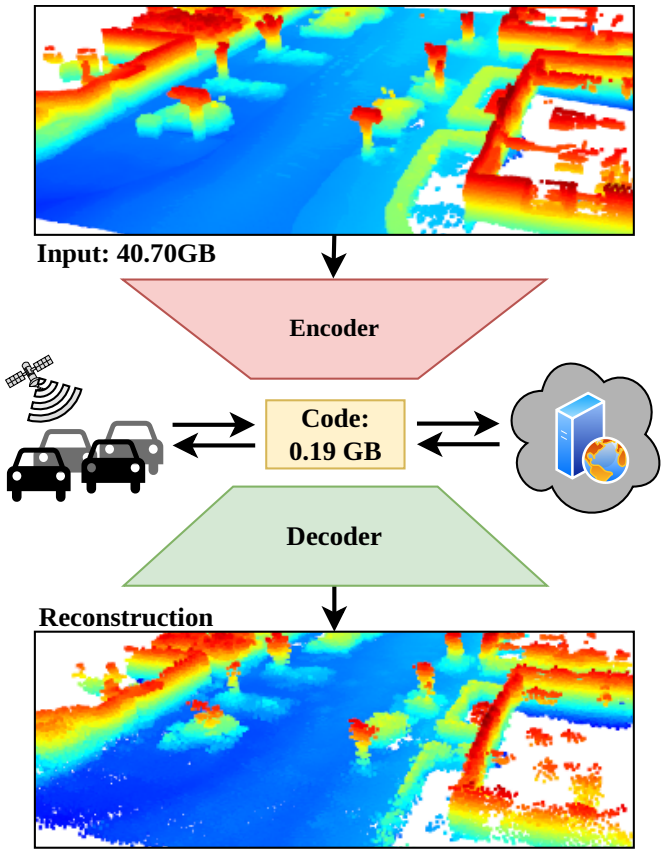


Figure 5.2: 3D point clouds obtained by an autonomous car or robot require a large amount of memory. Transferring this data to a fleet of cars or sending it to some cloud service requires compression of the data. Our approach can reconstruct dense point clouds from a highly compressed representation even when targeting low bit rates. The point cloud color indicates the height above ground ranging from blue (low) to red (high).

The main contribution of this chapter is a novel deep learning-based approach for compressing 3D LiDAR point cloud data in a lossy fashion for large-scale outdoor environments (see Figure 5.2 for an illustration). Our approach exploits the occurrence of common structures through local feature descriptors. It learns a small and compact set of local feature descriptors that allows compressing and reconstructing point cloud data. Our approach is end-to-end learnable and provides dense point clouds even when targeting low bit rate compression. Additionally, we propose a novel deconvolution for point clouds with feature propagation and integration. Our deconvolutional kernel operates directly on a set of points, which makes discretization unnecessary. We compare our approach with state-of-the-art compression techniques such as Draco [70] and the octree-based approach from Mekuria et al. [151]. In brief, we are able to provide higher quality maps after decompression than the state-of-the-art compression approaches at the same bit rate.

5.1 Related Work

In recent years, we witnessed an increasing interest in deep learning using point clouds from the computer vision and robotics community. Guo et al. [82] provide a larger survey of the field, while here we concentrate on representation learning and point cloud compression using neural networks.

5.1.1 Representation Learning on 3D Point Clouds

Point clouds in their rawest form are a set of coordinates, typically sampled from the surface of an object or scene. Representation learning for 3D point clouds tries to estimate useful features for solving specific tasks. Earlier works focus on handcrafted, mostly geometric features like histogram of normal orientations [230], spin images [104], or spectral features [14, 157].

The ability of neural networks to reliably estimate patterns in the data has motivated many researches to investigate how these methods can be applied on point clouds. PointNet [180], one of the earliest works, uses multi-layer perceptrons followed by pooling operations to extract pointwise, but also point cloud-wise descriptors. The biggest shortcoming of this method is that it does not have local awareness. The follow-up work PointNet++ [181] tries to overcome this problem by hierarchically applying the PointNet structure repeatedly to local neighborhoods for different scaling factors. Each layer downsamples the point cloud by iteratively choosing the furthest point from the already collected subset to keep a good coverage of the point cloud. Liu et al. [130] extend this idea by a multi-level feature aggregation to increase the contextual information.

A different way to obtain locally descriptive features is the usage of convolutional neural networks. In the 2D image domain, the convolution is defined on the discrete pixel positions that naturally lie on a grid, which is still the standard for convolutional neural networks that process images [123]. In the point cloud domain, there does *not* exist such a clear standard formulation. The natural extension from the 2D image domain to 3D is the discrete formulation of the convolution on Voxel grids [149, 267]. However, 3D point cloud data differs more from images than just through dimensionality: (1) point clouds are sparse, while images are dense; (2) the coordinates in a point cloud are typically not located on a raster, but arbitrarily in space. To address the sparsity, convolutions on sparse voxel grids have been proposed [38, 39]. Those methods only compute the convolution on occupied voxels, and therefore assume the empty voxels to have a zero feature. Similarly, convolutions can also be defined on octrees [185, 249], which hierarchically partition the space. The hierarchy leads naturally to different abstraction levels. However, both methods, sparse voxel grids, and octrees, need to discretize the scene, which leads to errors.

To overcome this problem Thomas et al. [226] defines a convolution operation for point clouds without discretization. They define a continuous convolution for point clouds with the help of some predefined kernel points, which they call KPConv. These kernel points are distributed in space and each has its own convolutional weight associated. The 3D continuous weight space can be interpolated from these discrete positions allowing to utilize this convolution for arbitrary distributed point clouds. In this thesis, we will mainly utilize KPConv [226] to extract local point features, without having any discretization effects.

Most convolutional networks utilize some sort of downsampling, and upsampling to learn low-level and more abstract features [226]. Our key idea for compression is to utilize these intermediate downsampled, low resolution, but abstract features as compressed representation. The established network architectures [38, 226] do not target compression but are implemented for semantic segmentation instead. This allows them to utilize information from the original input point cloud in their decoder part. We are targeting compression, therefore our decoder should only use the information from our compressed representation to restore the input. Using information from the input point clouds would require to store this data as well, and thus we would not efficiently compress. We address the problem by defining our own deconvolution that is not only computing features, but also upsamples the point coordinates. Our recovering of the point coordinates is a similar problem to the point cloud upsampling task, which tries to estimate a dense point cloud based on a low resolution point cloud. For this, Yu et al. [284] propose a network to upsample point clouds using local feature matrices. In contrast to their hierarchical purely point-based approach, our deconvolution integrates and propagates features. We use multiple deconvolution layers which allows for more flexible upsampling rates at inference time.

Transformer architectures have shown successful results, first in natural language processing [238], and then also in image processing [52]. While convolutions aggregate information based on their spatial distribution, Transformer architectures utilize the attention mechanism to aggregate information based on feature similarity. In the domain of point cloud processing, Transformer methods can be split into two categories. (1) Global Transformers compute the attention mechanism over the full point cloud, but thereby is only feasible for smaller point clouds, as often encountered in computer vision [81], or in robotics for radar data [287, 288]. (2) Local Transformers compute the attention mechanism only locally within the proximity of the points, thus only aggregate features from their local neighbors as for convolutions [227, 296]. In this part of the thesis, we utilize solely convolutional neural networks, while in later chapters we also incorporate Transformer blocks and especially look into more detail in the attention mechanism.

Recently, the use of diffusion models [88, 210] has also been investigated in the context of point cloud generation [124, 164]. Results from diffusion models are often less noisy which makes it interesting also for compression [277]. We will leave the investigation of compressing point clouds with diffusion models as an open future research direction.

5.1.2 Point Cloud Compression

Many works in the field of robotics, computer graphics, and computer vision focus on compressing point cloud data. Octrees [150] can be used to efficiently store three-dimensional data and furthermore is also an efficient map representation in the field of robotics [89, 56]. Binary octrees partition the space recursively into octants until a certain depth is reached, or the octant does not contain any points. The leaf octants describe finally the structure of the scene. A point cloud can be restored by estimating the center points of each occupied leaf node. Schnabel et al. [198] predict for each octant the number of non-empty cells, as well as the cell configuration based on local surface approximations. The imbalanced occurrence of these attributes allows for further compression using arithmetic coding. Huang et al. [96] reorders the bits of the occupancy codes and quantizes the normals for lowering the entropy. OctSqueeze [94] uses neural networks to learn a conditional probability model, followed by entropy coding to compress frequent symbols with fewer bits than rarer symbols. Octrees efficiently store three-dimensional coordinates but require additional methods to compress attributes [290]. In the domain of computer graphics, the similar VDBs [158] are more commonly used for efficient representation. First robotic works [242] have shown success in applying this compact map representation for LiDAR-based mapping.

Other approaches focus on an iterative prediction of neighboring points using spanning trees [80, 153]. Tree structures are very memory efficient in most real-world scenarios but do not exploit the full potential of recurring common objects.

The correlation between point clouds acquired from LiDAR or depth camera streams offers a further possibility to save memory. A range-image-based representation allows for the usage of image or video compression algorithms [106, 152, 160, 221, 235]. However, such a projective representation is not suitable for large dense maps captured from many locations, since it represents well only the environment of the actual viewpoint, while many parts are occluded.

Golla et al. [67] exploit standard image compression for static point clouds by storing oriented compressed height and occupancy images for local patches. Deep convolutional autoencoders can use a rate-distortion loss [182, 183] to ensure a good trade-off between quality and memory consumption. Quach et al. [182] use voxel grids to define the convolution, which is feasible for smaller objects but can be very memory exhaustive for point clouds of outdoor environments like

the ones we operate on. Note, that only the initial and predicted voxel grids are memory exhaustive, while the intermediate results are still compressing the data. In addition to the autoencoder, they utilize differentiable quantization for further compression. The follow-up work [183] replaces the voxel grid with an octree-based structure to reduce the initial memory demands, alongside improved entropy modeling. In contrast, our KPConv-based network does not require to voxelize the whole scene and works directly on the point clouds, making it suitable for larger scale point clouds. For us, quantization after training was sufficient, without the need to specially train for it.

Similar to Huang et al. [95], we also propose an end-to-end learnable point cloud compression autoencoder network. In contrast to their approach embedding the information of the whole point cloud into a single feature, we store a set of local feature descriptors together with positions from which we can then restore the point cloud.

5.2 Point Cloud Compression Using a Convolutional Autoencoder Network

The idea of our novel approach for point cloud compression is to learn for a small subset of points useful features from which we can recover the original point cloud. To this end, we propose an autoencoder structure for point cloud compression comprised of two parts. First, the encoder learns, from a given input data, a reduced and often more abstract representation, which we denote as embedding. The embedding is the input for the second part, the decoder that tries to reconstruct the original data using this compressed representation. By comparing the reconstruction with the original point cloud, the network can learn self-supervised parameters via backpropagation [192]. An illustration of our method is depicted in Figure 5.3. Compression is achieved as long as the embedding is smaller than the original point cloud.

5.2.1 Encoder Blocks

In our case, the input of the encoder is a dense 3D point cloud with the coordinates $P \in \mathbb{R}^{N\times 3}$ and point-wise features $F \in \mathbb{R}^{N\times D}$. The encoder uses multiple convolutional layers to learn local geometric features for each point. In Figure 5.3, the coordinates P are denoted in blue and the learned features F in red. Each layer reduces the number of points while increasing the receptive field. We use kernel point convolutions (KPConvs) by Thomas et al. [226], which directly operate on the features of the points themselves to avoid discretization effects caused by grid-based representations such as voxel grids or octrees. We use the ResNet-

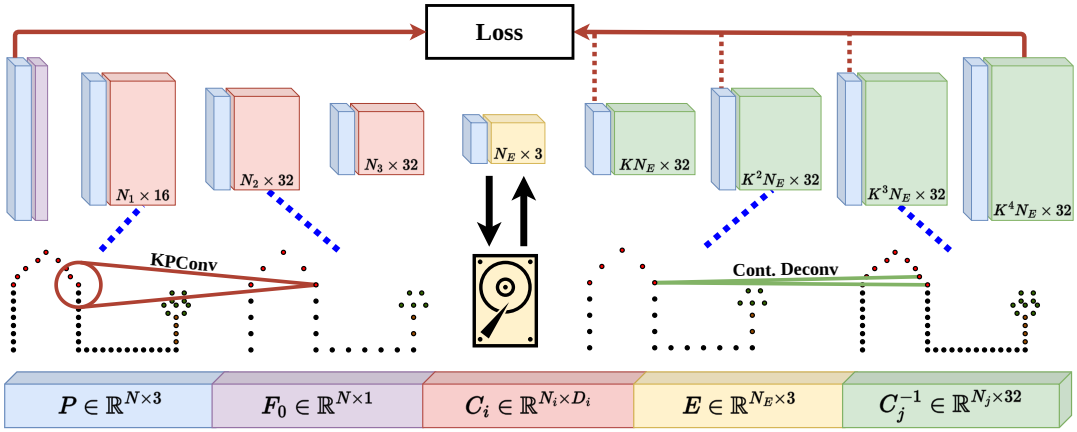


Figure 5.3: Schematic overview of our proposed network. The encoder takes as input a point cloud (P, F_0) and computes subsequentially for each subset a feature descriptor $C_i, i \in \{0, 1, 2\}$ (red) using kernel point convolutions. The last feature descriptor will be mapped using an multi layer perceptron (MLP) to the compact embedding space $E \in \mathbb{R}^{N_E\times 3}$ (yellow) to enable a memory-efficient representation. This embedding E with its associated points can be used for storage or transmission and will later be used by the decoder to decompress the point cloud. The decoder consists of four deconvolutional layers which subsequentially upsample the point cloud (green). We use the feature of the last layer as a refinement of the last coordinates. The loss enforces a similar appearance of input and output. Additionally, we have a regularization term for each upsampled cloud (denoted by the dotted line) to be lower-resolution versions of the input.

style blocks that compute the output features F_o based on the input features F_i and their corresponding coordinates P_i. The identity shortcuts in ResNet blocks enable a more direct gradient flow to earlier layers to reduce the risk of encountering instabilities due to vanishing gradients [86]. For a more detailed explanation, we refer to Section 2.2.2 or the KPConv paper [226]. In each encoding block, we downsample the previous point cloud using grid-based subsampling. This returns for each occupied voxel the point which is the nearest to its center. It provides us with a homogenous point distribution without the risk of losing all points in a certain sparser area (as random sampling) or incur long sampling times (as for furthest point sampling). In contrast to skip connections between the encoder and decoder, no additional information needs to be stored for the decompression. The last layer consists of an MLP to compress the features of size $\mathbb{R}^{N\times D_{\text{out}}}$ to the desired dimension $\mathbb{R}^{N\times D_{\text{emb}}}$.

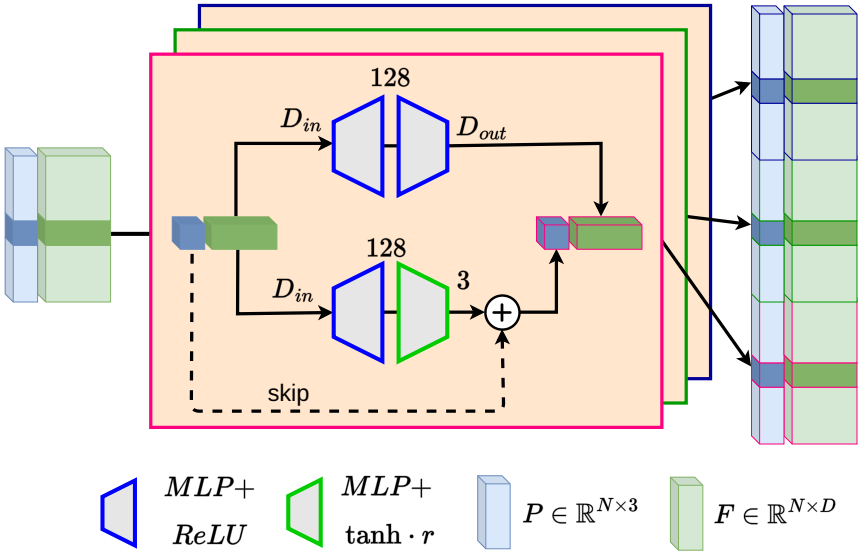


Figure 5.4: Overview of our proposed deconvolution block. Our deconvolutional kernel consists of two small MLP networks which serve as 1D convolutions. The first MLP (upper part) transforms the old feature F into a new feature space. The second MLP (lower part) predicts the offset to the new position $\boldsymbol{p}$ within the unit-square and scales it to the desired size r. Depending on the number of deconvolutional kernels, in this example $K = 3$, we upsample the point cloud by a factor of K.

5.2.2 Decoder Blocks

The task of the decoder is to reconstruct the original data from the embedding. Most encoder-decoder networks use skip connections [156, 187] from the encoder to the decoder to keep the high-frequency information. For compression, skip connections cannot be used, since it would require storing additional data from the encoder blocks for usage in the decoder. Hence, the whole signal must be encoded in the embedding to achieve effective compression. Therefore, we present a decoder block that does not depend on any skip connections but rather estimates the lost coordinates themselves. For a given point $(\boldsymbol{p}_i, \boldsymbol{f}_i)$, we define the deconvolution C^{-1} by a set of K MLP layers, as depicted in Figure 5.4. For each point, we obtain K new points $\{(\boldsymbol{p}_i^k, \boldsymbol{f}_i^k)\}, k \in \{0, \dots K-1\}$ within a given radius r by

$$\boldsymbol{p}_i^k = \boldsymbol{p}_i + r \cdot \Delta_k(\boldsymbol{f}_i), \tag{5.1}$$

$$\boldsymbol{f}_i^k = \Phi_k(\boldsymbol{f}_i). \tag{5.2}$$

Let us call Δ_k an offset block, which consists of an MLP with one hidden layer, uses a ReLU activation after the first layer, and tanh after the second. The offset block Δ_k determines a coordinate increment by a nonlinear mapping of the feature space into the coordinate frame of the kernel $\Delta_k : \mathbb{R}^{D_{\text{in}}} \mapsto [-1, 1]^3$. The feature block $\Phi_k : \mathbb{R}^{D_{\text{in}}} \mapsto \mathbb{R}^{D_{\text{out}}}$ computes new features based on the old descriptor. Similar to the offset block Δ_k, the feature block Φ_k is also an MLP with one hidden

layer but utilizes two ReLU activations instead. K offset and feature blocks form the proposed deconvolution $C^{-1} : \{\mathbb{R}^{N\times 3}, \mathbb{R}^{N\times D_{\text{in}}}\} \mapsto \{\mathbb{R}^{KN\times 3}, \mathbb{R}^{KN\times D_{\text{out}}}\}$ which will be applied to each point $\{(\boldsymbol{p}_i, \boldsymbol{f}_i)\}$ of the current layer. This upsamples the point cloud by a factor of K.

5.2.3 Network Architecture

Figure 5.3 provides an overview of our proposed network architecture. We use 3 KPConv encoding blocks with grid-based subsampling followed by one MLP layer for compressing the embedding. Experiments show that a deeper encoder architecture leads to a lower decompression quality. We explain this behavior by vanishing gradients due to the absence of skip connections between the encoder and decoder. The radius of the convolution is equal to the resolution r_s of the sampling grid. This ensures that all points are part of at least one convolutional operation.

We estimate the relation between the grid resolution r_s and the sub-sampling rate $A(r_s)$ to upsample the point clouds to their original size

$$A(r_s) = \frac{a}{r_s^b}. \tag{5.3}$$

Further derivation and reasoning of the chosen power function (with the parameters a and b) are provided in Section 5.3. We use the grid resolution r_s to control the compression rate. The sparser we sample, the higher the compression will be and the more points we need to upsample. Each KPConv has $3^3 = 27$ kernel points arranged in a grid with an influence radius of $\sigma = r/2$. The feature dimension (the red blocks in Figure 5.3) for the KPConvs is $16, 32, 32$ respectively, and 3 for the embedding (yellow block). We saw that these relatively compact feature descriptors are sufficient to store the geometric information about the neighborhood. If the original point cloud has no feature, we then use the occupancy value ($\boldsymbol{f}_i = 1$) as advocated by Thomas et al. [226]. The decoder consists of 4 deconvolutions to upsample the embedding to its original size. We use 32-dimensional point features and 128-dimensional hidden layer spaces in the deconvolution (expressed through the green blocks in Figure 5.3). The upsampling factor K_i of the deconvolutional kernels is adapted to the sampling rate (see Section 5.3.6.1). Since the feature of the last layer is unused otherwise, we feed it to an MLP ($\mathbb{R}^{32} \mapsto \mathbb{R}^3$) to refine the coordinates.

5.2.4 Loss Function

We want to restore a point cloud, which is as similar as possible to the input. We use the Chamfer distance D_{CD} as a measure of similarity in our loss function $\mathcal{L}$. It is the average symmetric squared distance $\bar{d}^2$ of each point to its nearest neighbor in the other point cloud

$$D_{\text{CD}}(P_{\text{in}}, P_{\text{out}}) = \frac{\bar{d}^2(P_{\text{in}}, P_{\text{out}})}{2} + \frac{\bar{d}^2(P_{\text{out}}, P_{\text{in}})}{2}, \tag{5.4}$$

$$\bar{d}^2(P_i, P_j) = \frac{1}{|P_i|} \sum_{\boldsymbol{p}_i \in P_i} \min_{\boldsymbol{p}_j \in P_j} \|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2^2, \tag{5.5}$$

where P_{in} denotes the input point cloud before compression and P_{out} the decompressed point cloud. Using the symmetric distance prevents the network from flooding the whole space with points but also from leaving out parts that have been present in the original point cloud. We add the Chamfer distances between the input point cloud and the output points $\hat{P}$ of all deconvolutions as a regularization term to ensure valid intermediate results. The loss $\mathcal{L}$ is then given by:

$$\mathcal{L} = D_{\text{CD}}(P_{\text{in}}, P_{\text{out}}) + \beta \sum_j D_{\text{CD}}(P_{\text{in}}, \hat{P}_j), \tag{5.6}$$

where β is a weight to control the impact of the regularization term and we use $\beta = 0.2$ in all of our experiments. For a more detailed analysis of the regularization, see Section 5.3.6.2.

5.3 Experimental Evaluation

In this section, we validate that our proposed algorithm is able to compress point cloud data efficiently. We compare our method to Draco [70] and the octree-based compression algorithm by Mekuria et al. [151], which we will refer to as "MPEG anchor". Additionally, we will show the results for a binary Octree [150] that distinguishes between free and occupied space.

5.3.1 Implementation Details

We use the octree implementation by Behley et al. [15] for an efficient radius neighbor search in the KPConv blocks. Our model is implemented in PyTorch [176] and trained on a GeForce RTX 2080 SUPER and with an Intel CPU with 3.5 GHz. We use the Adam optimizer [114] and the one-cycle learning rate schedule proposed by Smith et al. [209] with a start learning rate of 10^{-6}, which will increase to 10^{-4} in the first 20 epochs and afterward decrease to

10^{-5}. We use the cosine annealing strategy and train for 200 epochs with a batch size of 3. We limit the number of input points to 30,000 points to speed up the training and reduce the memory footprint while training. Nevertheless, we compare at test time our reconstructed point cloud to all available points. We train four network architectures with varying sub-sampling resolutions $r_s \in \{3.0\,\mathrm{m}, 2.0\,\mathrm{m}, 1.2\,\mathrm{m}, 1.0\,\mathrm{m}\}$ for different compression levels. The encoder needs approximately 0.27 MB and the decoder between 0.30 MB and 0.62 MB memory storage (depending on the upsampling rates), which is a one-time investment and not depending on the number of compressed maps or points.

5.3.2 Experimental Setup

Our network is designed for compressing large-scale dense point clouds. We evaluate our method on the SemanticKITTI [12] dataset. For obtaining highly accurate and dense point clouds, we aggregate all scans using the ground truth poses and divide the map into patches of size $40 \times 40 \times 15\,\mathrm{m}^3$. The labels have been used to remove the dynamic objects which otherwise will lead to artifacts in the map. To remove redundant points in the original point cloud, we filter it using a voxel grid with a resolution of $10\,\mathrm{cm}^3$. We use sequences 00 to 10 (except 08) for training. A small subset of the training data serves as a validation set and the complete sequence 08 is used for testing and comparison with the baselines.

The quality of a compression algorithm is a trade-off between compression ratio and reconstruction error. For the compression ratio, we use the average bits per point required for storing the encoding of the point cloud. We use three metrics for measuring the reconstruction error. Symmetric point cloud distances D_d are widely used for measuring the quality of the point cloud reconstruction. These metrics consist of two parts: the distance $\bar{D}_d$ from the ground truth point cloud P_{in} to the reconstruction P_{out} and vise versa

$$D_d(\mathsf{P}_{\text{in}}, \mathsf{P}_{\text{out}}) = \frac{\bar{D}_d(\mathsf{P}_{\text{in}}, \mathsf{P}_{\text{out}})}{2} + \frac{\bar{D}_d(\mathsf{P}_{\text{out}}, \mathsf{P}_{\text{in}})}{2}, \tag{5.7}$$

$$\bar{D}_d(\mathsf{P}_i, \mathsf{P}_j) = \frac{1}{|\mathsf{P}_i|} \sum_{\boldsymbol{p}_i \in \mathsf{P}_i} \min_{\boldsymbol{p}_j \in \mathsf{P}_j} d(\boldsymbol{p}_i - \boldsymbol{p}_j). \tag{5.8}$$

Thereby, the metric is sensitive to false positives (reconstructing points in unoccupied areas) and false negatives (leaving out occupied areas). The Euclidean distance $D_e = D_d(\mathsf{P}_{\text{in}}, \mathsf{P}_{\text{out}})$ with $d = \|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2$ is used as a metric for reconstructing the points itself. However, for some robotics applications (e.g., point-to-plane ICP), it is less important to reconstruct the exact same point than that it lies on the same surface. Therefore, we also show the symmetric plane distance $D_\perp = D_d(\mathsf{P}_{\text{in}}, \mathsf{P}_{\text{out}})$ with $d = |\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{p}_j)|$. Where $\boldsymbol{n} \in \mathbb{R}^3$ denotes the ground truth normal of that point. The normals have been precomputed using

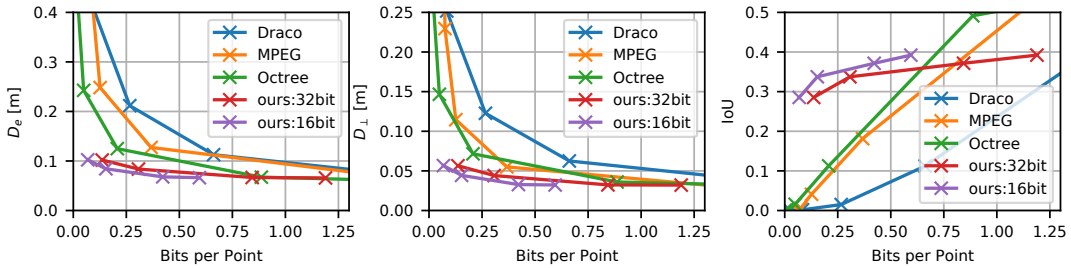


Figure 5.5: Compression results on the test sequence 08 of the KITTI Vision Benchmark dataset. We use Draco [70], the MPEG Anchor from Mekuria et al. [151], and our own binary Octree implementation as baselines. Our approach can reconstruct the point cloud for the same amount of memory at a higher quality level.

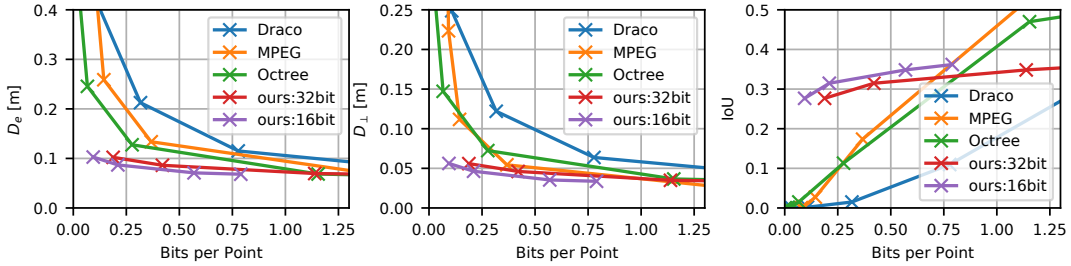


Figure 5.6: Compression results on the nuScenes dataset. The model was trained on SemanticKITTI and evaluated on nuScenes to show the generalizability of our approach. The errors are slightly higher compared to the validation set of KITTI, but we are still able to outperform the baselines.

the Eigenvalue decomposition of the covariance matrix from all points within a 50 cm radius around the query point, for more information we refer to Section 2.1.

The last metric is the intersection-over-union (IoU) between occupancy grids G for both point clouds. The IoU is here defined as

$$IoU = \frac{|G_{\text{in}} \cap G_{\text{out}}|}{|G_{\text{in}} \cup G_{\text{out}}|}. \tag{5.9}$$

The occupancy grids G_{in} and G_{out} have a resolution of $20 \times 20 \times 10\,\text{cm}^3$ as used by Huang et al. [94].

5.3.3 Compression Results

In this first experiment, we compare our compression results to the baselines to quantify the compression performance of our proposed method. Our approach stores the output of the encoder, namely a set of features and points, as the compressed representation. We will show the results for storing the embedding as 32 and 16 bit floating-point values. The compression results of our approach and the baselines on sequence 08 are presented in Figure 5.5. Our proposed method outperforms the baselines in the distance-based metrics D_e and $D_\perp$, as well as

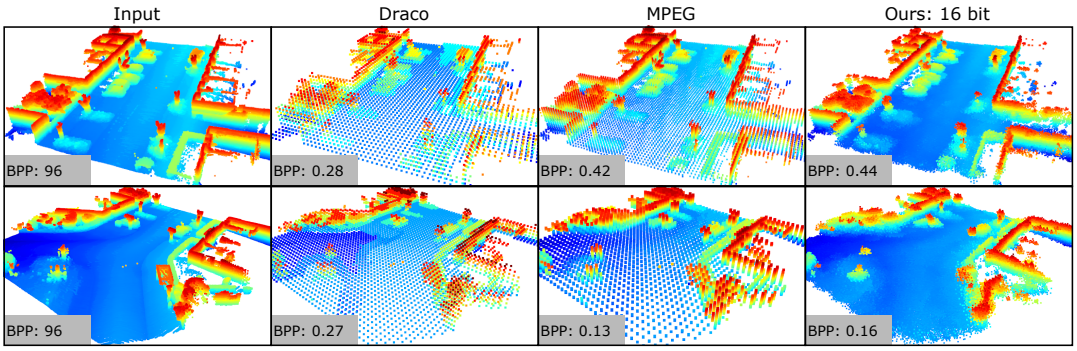


Figure 5.7: Qualitative results of our proposed method and the baselines for different bits per point (BPP) as denoted in the pictures on two example point clouds. The points are colored according to their height. Our approach is able to recover dense point clouds also when targeting low bit rates.

in the IoU for the 16 bit representation. The small quality gain ($< 2\%$) of the 32 bit representation is disproportional low concerning that the memory demand doubles. The reduced density of the baselines leads to substantially higher errors than our approach which can reconstruct for each bit rate the same number of points. Our approach achieves over 2.4 times lower reconstruction errors for bit rates under 0.1 bits per point compared to the baselines. We think that the worse performance of the MPEG Anchor [151] compared to the plain occupancy octree is due to some overhead for attribute compression and the ability to further compress tele-immersive data streams.

5.3.4 Generalization Capability

Learning-based methods often degrade when applied in a different environment due to over-fitting to the specific characteristics of the training set. We claim that our method generalizes well by learning the local geometries rather than remembering the global shape. To support this claim, we test our approach on a completely different dataset without retraining the model. Here, we use the nuScenes dataset [26], which not only has a different sensor setup (different height, field-of-view, and the usage of 32 beams instead of a 64-beam LiDAR) but is also recorded on a different continent which usually changes the appearance of the scenes quite substantially. Note that there are no labels for the nuScenes dataset available so that the dynamics will not be removed from the map, which makes it more challenging due to the presence of new artifacts it has never seen before. Figure 5.6 shows that we still outperform the baselines for the 16 bit representation, even though the margin to the baselines got smaller.

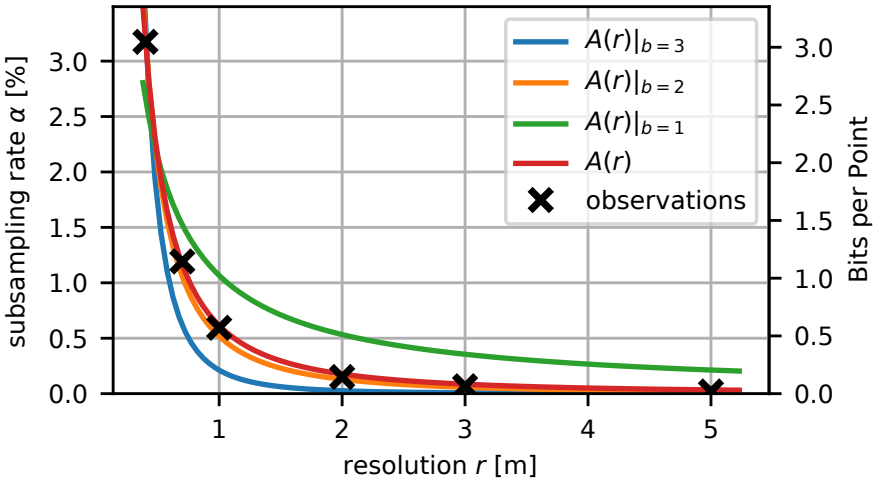


Figure 5.8: We estimate the dependency between the subsampling rate α and the subsampling resolution r to predict the bit rate and to estimate the necessary upsampling rate for the deconvolution. We use least squares fitting to estimate the parameters of a power function $A(r) = a \cdot r^{-b}$ with parameters a and b.

5.3.5 Qualitative Analysis

In this part, we will analyze the decompression results qualitatively. In Figure 5.7, we show the decompressed maps of the baselines and our approach. For the baselines, we have chosen the quality levels with the closest bit rates to our approach. The points are colored according to their height. As we can see, our approach is able to recover comparably dense point clouds even for varying compression rates. The point clouds of the baselines are sparser, especially when targeting low bit rates (see Figure 5.7 second row). When reducing the bit rate, the decompressed maps of our approach get noisier while the baselines show larger quantization errors. Structures like the curbs of the streets are only visible in our denser maps.

5.3.6 Ablation Studies

To validate the choices made and to give a deeper understanding of the behavior of the network, we conducted some ablation studies. We first investigate the adaptive sampling strategy and then the regularization.

5.3.6.1 Adaptive Sampling

Each encoding block reduces the number of points by grid-based subsampling and thus the number of embedding points depends on the resolution of the sampling grid. In this experiment, we investigate the influence of varying grid resolutions r_s and the subsampling rate α of the points. This enables us to adapt the up-

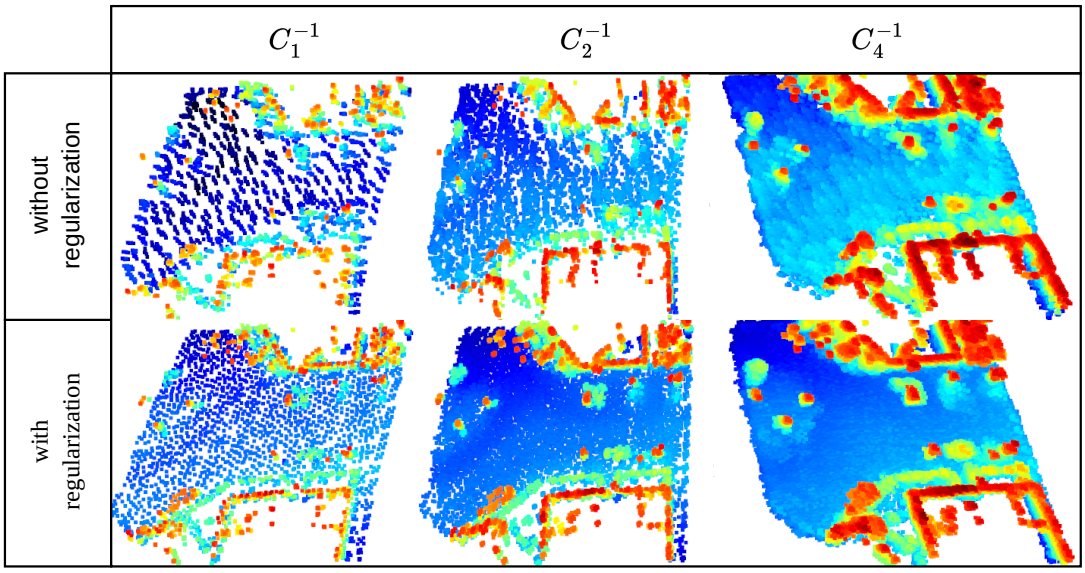


Figure 5.9: Intermediate point clouds after each deconvolutional layer $C_j^{-1}, j \in \{1, 2, 3, 4\}$ for the same networks but with different loss functions. The network of the top is trained without the regularization term, whereas the bottom row is learned with the regularization term. The lack of regularization leads to less homogeneous and more noisy point clouds after each deconvolution. When trained with the regularization, the intermediate point clouds can be used as lower lower-resolution model.

sampling rate to ensure that the input and output point clouds have similar sizes. Additionally, it enables us to predict the bit rate based on the resolution r_s or vice versa. Therefore, we subsample the point clouds of the training set with different grid resolutions r_s and compute the mean subsampling rate α. We fit a power function $A(r_s)$ which is proportional to the density distribution $\rho(r_s)$ of the point cloud

$$A(r_s) = \frac{a}{r_s^b} \propto \rho(r_s). \tag{5.10}$$

The parameter a denotes the magnitude of the density while the parameter b describes the dimensionality of the point distribution. A fixed parameter $b \in \{1, 2, 3\}$ would correspond to a voluminous, planar, or linear distribution, respectively. The result of $A(r_s)$ as well as for fix $b \in \{1, 2, 3\}$ are shown in Figure 5.8. The best-fitted function is given by $a = 0.006$ and $b = 1.80$. The parameter $b = 1.80$ reflects the huge amount of planar surfaces (streets, walls, meadows, etc.) in outdoor environments. Each deconvolutional block i up-samples the point cloud by the factor $K_i = \lceil A(r_e)^{-1/I} \rceil$, where I is the number of deconvolutional blocks and r_e is the sub-sampling resolution of the last encoding layer.

Table 5.1: Quantitative regularization impact

experiment	D_e	$D_\perp$	IoU
without regularization	0.110	0.062	0.327
with regularization	0.077	0.039	0.332

5.3.6.2 Impact of Regularization

In this experiment, we show the importance of the regularization term. We train the same network two times, first without and second with regularization. In Figure 5.9, we can see the qualitative impact on the point clouds after each upsampling step. The distribution of the points is more uniform and less noisy for the point clouds of the regularized network. The regularization term penalizes big point distances between the clouds. This leads implicitly to more homogeneous regions so that we do not see the necessity of an additional repulsion loss as in the PU-Net [284]. The quantitative results in Table 5.1 support our assumption that more meaningful intermediate point clouds help the network reconstruct the data. Additionally, the intermediate point clouds can, with regularization, be used as lower-resolution versions of the point cloud.

5.4 Conclusion

In this chapter, we presented a novel approach for lossy point cloud compression exploiting the pattern recognition ability of neural networks. The main idea is to learn a convolutional autoencoder network that processes point clouds. An autoencoder utilizes first an encoder to obtain an intermediate embedding which the decoder can use to reconstruct the input. The intermediate embedding consists of a low-resolution point cloud with point-wise features that encode information about their local neighborhood. For the decompression we propose a 3D deconvolution that directly operates on the points to avoid discretization effects from using voxel grids. The network is trained such that it reduces the error between the input and the reconstructed point clouds. Since we enforce the intermediate embedding to be substantially smaller than the input point cloud, we yield a compressed representation of the point cloud. The advantage over traditional methods is that we do not only have the compressed representation for encoding the information, but also the decoder. The decoder is trained on a large set of point clouds and therefore can exploit recurrent structures across different point clouds. We have shown that training our network on dense point clouds of outdoor environments can reduce the memory footprint substantially while keeping a relatively good reconstruction quality. Our method can reduce the memory

footprint of dense point clouds by a factor of 1:100, enabling to store significantly larger point cloud maps onboard. This is especially interesting for robots that have to operate in outdoor environments and require a dense 3D map. In the future, it would be interesting to investigate if such compressed 3D maps could be stored on a server and only transmitted on demand due the increased efficiency.

In the previous chapters, we have seen how to generate consistent large-scale point cloud maps. One major problem of those massive point clouds is memory consumption, which we tackled in this chapter. Using our compression method allows us to reduce memory consumption substantially. Potentially allowing for on-board storage or even on-demand transmission. We will use this compact map representation in the following chapters for localization. Notably, our compressed representation consists of a low-resolution point cloud with point-wise features. Those features must contain information about the local neighborhood of the points, since they can be used to recover the input scene for decompression. Utilizing those features directly for finding out if two point clouds are similar will be investigated in the following chapter. The resulting place recognition method will allow us to coarsely localize in a preconstructed compressed map. Afterward, we will do a fine registration to obtain the actual 6 DoF pose of our vehicle with respect to the compressed map. Once we have found our starting position within the map, we can track our position from then on, as will be discussed in Chapter 8.

Chapter 6

Place Recognition in Compressed Point Cloud Maps

The ability to localize in a map is a key ingredient of many robotic systems and autonomous cars. The robots need to know where they are in a map, in order to utilize the map data, e.g., for tasks like path planning, state estimation, or manipulation. Place recognition tries to solve the task by finding the robot's location in a prerecorded map by comparing it with the robots surrounding. It answers the question *"To which part of the map corresponds this place?"*.

In the previous chapters, we have investigated how to build large scale point cloud maps and how we can represent it in a memory efficient way. The resulting compressed representation will now serve as our map, within we want to localize the robots. A realization could look like the following: First, we have our mobile mapping platform, that drives through the environment and scans with the LiDAR sensor through the desired scene. In an offline post-processing step, we can generate our global point cloud using the LiDAR bundle adjustment approach from Chapter 4. Afterward, we split the point cloud in smaller chunks, so called submaps, and compress those with our compression network as discussed in the previous chapter. We now want to drive through the same environments at a different point in time, maybe even with a different robotic system, and localize the robot within this compressed map. Our goal is to find the part of the map that corresponds to the current location by comparing the current observations to the map. An overview of this workflow is depicted in Figure 6.1, in which we currently want to tackle the coarse localization task.

In this chapter, we try to find the submap that corresponds to the current robot's position by using point cloud-based place recognition, which we denote as coarse localization. One common way to tackle this task is to compute for each submap a descriptor that embeds the characteristics of the scene into a single vector. So our map representation will be extended, not only storing the

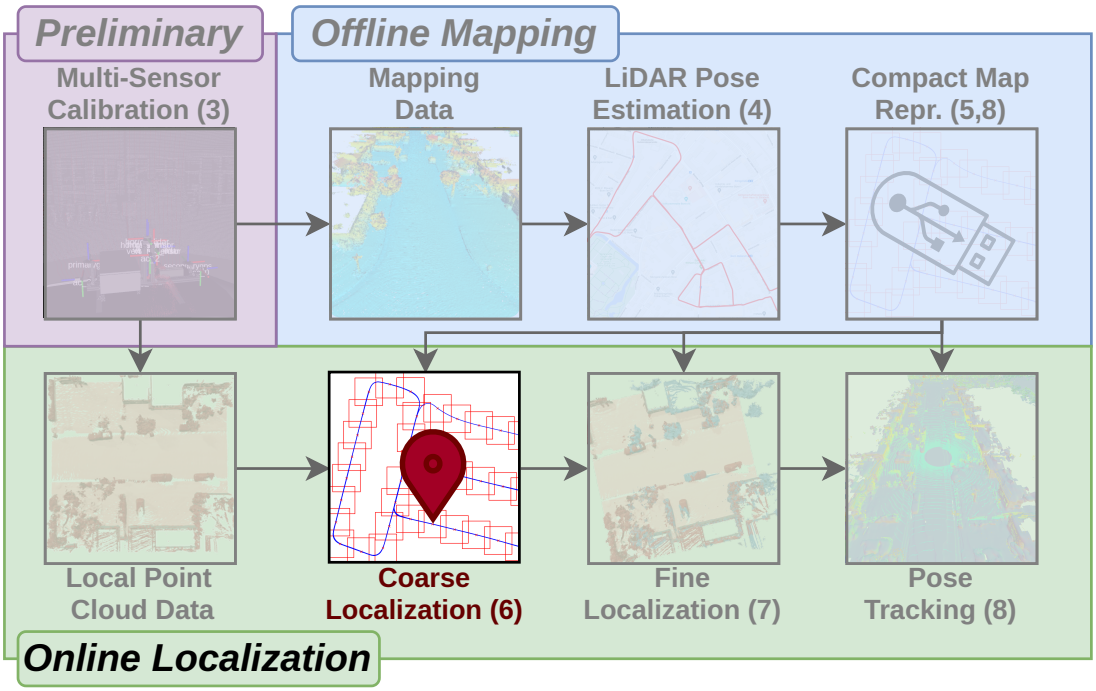


Figure 6.1: Overview of the Thesis. In the last chapter, we have seen how to compress dense point cloud maps. In this chapter, we want to investigate how we can coarsely localize a robot within the compressed maps, by using the current local point cloud of the robot. The goal is to find in which part of the overall map is the robot located. Afterward, we will look into obtaining a more accurate estimate using fine localization.

compressed point clouds, but also a vector that encodes the properties of the scene, respectively. We will call the collection of our submap descriptors the database. To retrieve the position of the robot, one also computes a descriptor for the current scene, the so-called query descriptor, based on the observed LiDAR point clouds. The query descriptor is then compared to the descriptors of the database to retrieve the location. If the learned encoding precisely describes the respective submap, then the most similar descriptor should correspond to the desired region in which the robot is located. Due to ambiguities in the scene, not only the most similar, but often a few possible positions are considered as potential candidates.

Deep neural networks are nowadays often utilized to compute those descriptors from the point clouds. For this, first local point features are computed, which then are aggregated into a single global descriptor. Most works utilize contrastive learning to ensure that the network produces descriptors that allow for finding similar locations. In contrastive learning, the network is shown a lot of positive examples from different point clouds of the same location, as well as negative examples from point clouds of different locations during training. A contrastive

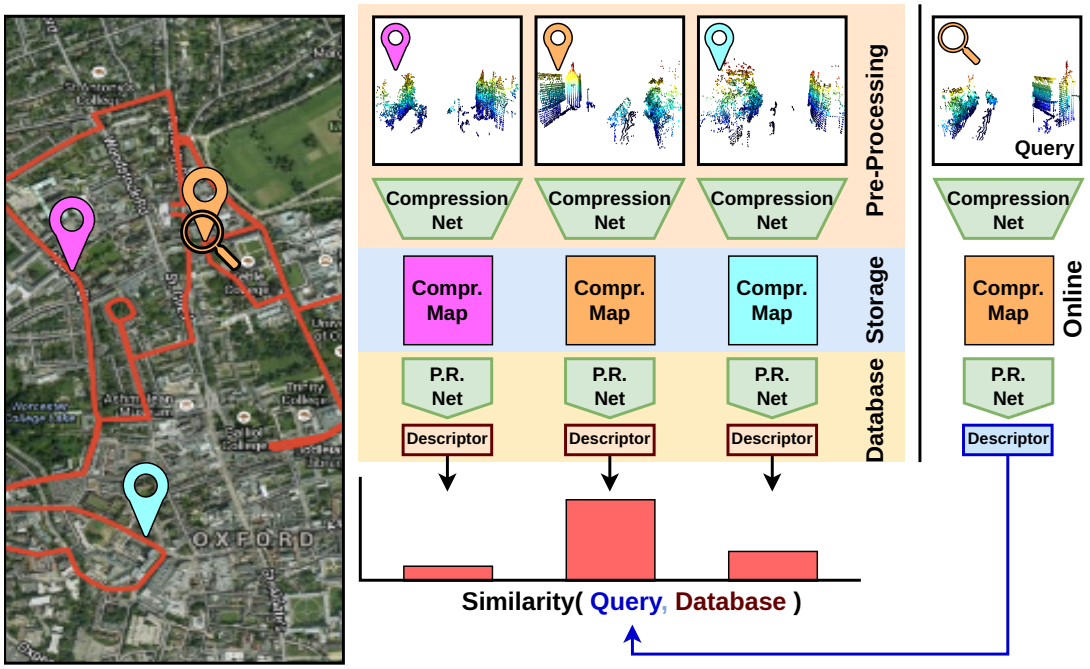


Figure 6.2: Point cloud-based place recognition. We compress the point clouds from a map using a compression network. The resulting compressed encodings can be stored efficiently and later be used for decompression, transmission, or place recognition. For place recognition, we extract descriptors from the compressed representation using our place recognition network (P.R. Net). When revisiting an area, one can retrieve the corresponding map by comparing the descriptors of the current position (query) and the compressed descriptors in a database.

loss function enforces that the positive pairs should produce similar descriptor vectors, while the negative pairs have low similarity.

In this chapter, we are going to exploit our compressed map representation to directly localize in the compressed map. As previously mentioned, the first step for the generation of the descriptors is to compute local point features. But if we recall what our compressed map representation from Chapter 5 actually is, then we see that it consists of a sparse point cloud with local features. In this chapter, we try to leverage directly those compressed point features to obtain the descriptors. By this, we can bypass the otherwise needed decompression, as well as the computation of local features from the decompressed point cloud. The compressed features already contain information about the local neighborhood, since from those the local area can be restored. In Figure 6.2, we have illustrated the localization process using our compressed map representation.

In this chapter, we propose two different place recognition methods, the Retriever and KPPR, to localize in compressed point cloud maps. The Retriever is a more simplistic approach and a proof of concept that our compressed representation can be used for place recognition. With KPPR, we propose a more

sophisticated method that achieves state-of-the-art performance. Both methods follow the descriptor generation paradigm of first computing local features and then aggregating them into a global descriptor. In the following, we will investigate how we can take our compressed features and enhance them to be better suited for place recognition. Additionally, we will look at a new method to aggregate the resulting local features into a global descriptor. Furthermore, we will focus on efficient training. For this, we adapt a training technique from the unsupervised domain to train faster, while increasing the performance at the same time over the conventional training scheme.

6.1 Related Work

Place recognition solves the problem of retrieving the current position based on sensor observations (so-called queries) in a given map (also called database). Most methods tackle this task by trying to find the entry from the database that is most similar to the query. The similarity between positions is often computed based on the similarity of global descriptors that are computed for the queries, as well as for the database entries. Our related work section focuses on those descriptor-based methods that try to estimate only the corresponding entry in the database. For a more complete literature review, we refer to the work by Yin [281] that also covers the areas of projective methods, place recognition with pose estimation, as well as correspondence-based approaches.

Images have often been used to represent the local surroundings of the query position and the entries in the database [136, 159, 203, 245, 246, 250]. However, images are prone to appearance changes caused by illumination conditions or seasonal changes. Point clouds from LiDAR sensors are less affected due to active sensing and 3D geometric information [237].

Generating meaningful descriptors from point clouds is often done in two steps: (1) computing local features, which (2) are aggregated into the global descriptors. Local features can be computed using classical handcrafted methods [11, 135, 189, 214, 215] or can be computed by trained neural networks [29, 32, 57, 99, 117, 268]. Earlier works like PointNetVLAD [237], or PCAN [294] utilize PointNet [180] as a backbone for feature computation. However, the local features lack descriptiveness, due to the PointNet [180] architecture that does not have a reliable notion of the point distribution. Convolutional neural networks provide more descriptive features, and can be defined on representations like graphs [122, 220], sparse voxel grids [29, 116], range images [32], or directly on the points [226]. The use of local features from convolutional neural networks increased the place recognition performance significantly [57, 116, 132]. HiTPR [90] computes local features using short-range Transformer architectures instead.

For computing the final descriptor those local features are aggregated through methods like bag of words [49, 204, 208], vector-of-locally-aggregated-descriptors as commonly referred to as VLAD [103], or its differentiable NetVLAD [3] version for learning-based approaches [237]. While most learning-based approaches train their networks directly for place recognition, some works [62, 222, 285] in the image domain have exploited pretrained networks to provide the necessary features. This motivates us to investigate if the features of our compression network could also be suitable as local features for place recognition. Although the pretrained networks were not optimized for place recognition, the methods were still able to localize successfully. This would allow localizing in the compressed map representation, without the need for decompression. Our method exploits our pretrained compression network for feature generation. We will investigate using those features directly for the feature aggregation, as well as further enhancing those features using an architecture developed based on KPConv [226].

The networks are optimized using contrastive learning to ensure that point clouds from the same location have similar descriptors, and point clouds from different locations are dissimilar. Most commonly, triplet or quadruplet losses are used for place recognition [90, 116, 237]. The triplet loss computes for each query descriptor the distance to a descriptor from the same location (positive) and to a descriptor from a different location (negative). The query and positive descriptor are pulled together, while the negative is pushed apart. The quadruplet loss extends the idea by a second negative to prevent the negatives from collapsing. The challenge for training is to find hard cases, i.e., point clouds that are structurally similar but from different locations, to ensure the network is able to differentiate those from actual positive matches. Most methods try to mine those hard negatives by sampling a lot of negatives to search for the hardest ones [132, 237]. Instead of computing for each query multiple potential negatives, Komorowski et al. [116] searches for hard negatives in the positives of different queries in the same batch. Additionally, they utilize dynamic batching to prevent the descriptors from collapsing. Hui et al. [98] on the other hand focus on efficient inference by training a smaller student network with a bigger teacher model. We on the other hand, exploit feature banks and a momentum encoder from momentum contrast [85] which allows us to use an arbitrary number of negatives that comes at basically no cost. Other contrastive learning approaches focus on large batch size training [31], online clustering [27], or mining strategies [265] to deal with negatives. In contrast, Zbontar et al. [286] does not need any of those techniques, but rather simply introduces a loss based on cross-correlation. A different direction in the unsupervised domain is to train entirely without negatives. To optimize for only positive examples one either uses a momentum encoder [71] or stops certain gradients [35]. Since in the unsupervised domain, the positive ex-

amples are usually only augmented views and the negatives are different images, not pushing away negatives that might actually be structurally similar can be an advantage. In the case of place recognition, which is usually done at least weakly supervised, we know which point clouds correspond (positives) and which ones do not (negatives) using GNSS data. Thus, the risk of having false negatives is substantially smaller, and we even want to distinguish structurally similar negatives for good performance. Therefore, we do not see an advantage in having no negatives at all for place recognition.

Our contribution to the field of point cloud-based place recognition is to provide methods that can directly operate on a compressed map representation. In this chapter, we propose the Retriever and KPPR for tackling place recognition. The most related work to the Retriever is PointNetVLAD [237], where we both use the same backbone for feature computation and the same training methods. The main difference is that we first use our compression network to operate on the compressed point clouds before passing it to the backbone. Besides that, we use our proposed attention-based feature aggregation method for the descriptor generation instead of the standard NetVLAD [3]. MinkLoc3D [116] is most related to our more sophisticated KPPR method. We both utilize a sparse convolutional backbone for the local feature generation. However, we use kernel point convolutions [226] and our compression encoder, while MinkLoc3D operates on a sparse voxel grid. For efficient training, MinkLoc3D is performing hard negative mining over the whole batch. In contrast to that, we exploit the ideas of a feature bank from MoCo [85].

While most approaches tackle one-shot localization, the performance can be improved substantially by taking sequential information into account to resolve ambiguities [127, 142]. We will not exploit sequences of matches and leave this open for future work.

6.2 Place Recognition in Compressed Maps

In this section, we propose different parts to create neural network architectures for point cloud-based place recognition in our compressed map. We focus on the local feature generation, and the effective aggregation of those into a global descriptor, and investigate the training to enhance efficiency. This chapter comprises the content from two of our works, namely the Retriever [258] and KPPR [261]. Our first work, the Retriever is a proof of concept, where we investigate if our compressed representation can be used for localization, thus having a fairly simple architecture. KPPR, our second work on point-cloud-based place recognition, has a more sophisticated network architecture, and by this requires also a more efficient training, to encounter the increased computational demand.

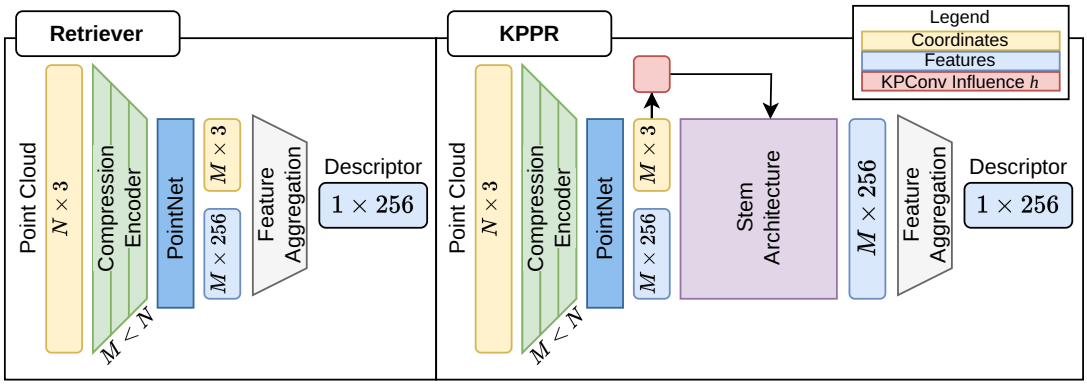


Figure 6.3: Our proposed network architectures first compress the point clouds using our compression network. The Retriever only modifies the resulting features using a PointNet, while KPPR utilizes an additional stem architecture for further feature enhancement. The final stage for both networks is to aggregate the point-wise features into a global descriptor.

However, both network architectures follow the same paradigm: First, we take our compressed representation and compute features that are better suited for place recognition. Afterward, we aggregate the resulting enhanced local features to build the point cloud-wise descriptors. Finally, we will look at the loss functions and training strategies for effective and efficient training.

In the following, we will discuss each part in depth, and point out where and for which method we use it. An overview of the architectures is illustrated in advance in Figure 6.3.

6.2.1 Feature Propagation Network

Our goal is to estimate our position in a compressed point cloud map by exploiting the features of our compressed representation without any further sensor data. For creating this compressed representation in the first place, we use the encoder we proposed in Chapter 5. For a given input point cloud $P \in \mathbb{R}^{N \times 3}$, the encoder $E : \mathbb{R}^{N \times 3} \mapsto \mathbb{R}^{N_c \times D_c}$ samples a small subset of $N_c \ll N$ points, with an expressive feature representation $F_c = E(P) \in \mathbb{R}^{N_c \times D_c}$ from which the decoder can recover the dense point cloud.

We want to use those features to generate descriptors that are usable for place recognition. Having such an expressive representation does, however, not directly mean that it is also the best representation for place recognition. For example, the compressed feature representation cannot be rotational invariant since it must be able to decompress the point cloud in the correct orientation. To compute features that are better suited for place recognition, we utilize a small PointNet [180]

variant, which transforms each point feature into a high-dimensional nonlinear space. For a detailed description of PointNet, see Section 2.2.1.

Note, that PointNet does not contain any convolutions or other means to enhance the features with additional information. The only goal of this small network is to transform the compressed feature representation into a feature space that is better suited for place recognition. The compression encoder plus the small PointNet is the foundation of our networks, and thus is part of the KPPR and Retriever architectures.

6.2.2 Convolutional Stem

The features from the encoder contain local information but lack a broader context due to the small receptive field of the compression encoder. Therefore, we propose to use a convolutional stem to aggregate information from a larger area. The convolutional stem consists of $j = 1, \ldots, J$ residual KPConv blocks

$$\mathrm{RKP}^j : \mathbb{R}^{M \times D_o} \to \mathbb{R}^{M \times D_o}, \tag{6.1}$$

which do not further subsample the already sparse point cloud. In the following, we will first briefly revisit the concept of KPConv [226], to then show how we can disentangle it into a block-dependent and a block-independent part. We will use the superscript to denote block-dependent variables, e.g., $\boldsymbol{F}^j$ are the features in the j^{th} block.

For a point $\boldsymbol{p} \in \boldsymbol{P}_c$, the convolution of the features $\boldsymbol{F}^{j-1}$ with the convolutional kernel g is defined as

$$\boldsymbol{f}^j = (\boldsymbol{F}^{j-1} * g^j)(\boldsymbol{p}) = \sum_{\boldsymbol{p}_i \in \mathcal{N}^j(\boldsymbol{p})} g(\boldsymbol{p}_i - \boldsymbol{p})^j \boldsymbol{f}_i^{j-1}, \tag{6.2}$$

where $\mathcal{N}^j(\boldsymbol{p}) = \{\boldsymbol{p}_i \in \boldsymbol{P}^j \mid \|\boldsymbol{p}_i - \boldsymbol{p}\| \leq r^j\}$ are all the points in the neighborhood within the radius $r^j \in \mathbb{R}$. The kernel g is defined by a linear combination of the weights $\{\boldsymbol{W}_k^j \mid k < K\}$ of the K kernel points $\{\boldsymbol{p}_k^j \mid k < K\}$:

$$h(\boldsymbol{p}_i - \boldsymbol{p}, \boldsymbol{p}_k^j) = \max\left(0, 1 - \frac{\|\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{p}_k^j\|}{\sigma}\right) \tag{6.3}$$

$$g(\boldsymbol{p}_i - \boldsymbol{p})^j = \sum_{k<K} h\big(\boldsymbol{p}_i - \boldsymbol{p}, \boldsymbol{p}_k^j\big)\, \boldsymbol{W}_k^j, \tag{6.4}$$

where its coefficients h decrease linearly with the distances from the neighbors to the kernel points.

In contrast to the original KPConv implementation, we can make the following simplifications to the stem architecture. First, we do not further subsample the points, therefore the coordinates in the point cloud stay the same,

i.e., $P^j = P^{j-1} = P$. Second, the neighborhoods will remain the same by always using the same radius $r \in \mathbb{R}$. Third, we arrange the kernel points always in the same grid structure such that the coefficients are independent of the block. Hence, $h(\boldsymbol{p}_i - \boldsymbol{p}, \boldsymbol{p}_k^j) = h(\boldsymbol{p}_i - \boldsymbol{p}, \boldsymbol{p}_k)$, which allows us to precompute the term for all J blocks. This includes the quite costly kNN search for the neighborhoods.

Consequently, the only variables that are changing in each block $j \in \{1, \dots, J\}$ are the kernel weights W_k^j and the features of the target point cloud F^{j-1}. All KPConv blocks are residual with rectified linear unit (ReLU) activation and layer normalization, similar to Thomas et al. [226].

The goal of the stem architecture is to refine the features and increase the receptive field. In our more lightweight architecture of the Retriever, we do not use the stem architecture and pass the local features directly from the feature propagation network to the feature aggregation that we will discuss next. The stem architecture is used in KPPR for feature enhancement, but comes at the cost of substantial amount of additional compute.

6.2.3 Feature Aggregation

In the previous parts, we have described how to compute a set of local features $F \in \mathbb{R}^{M \times D_o}$ from the compressed representation using the feature propagation network and optionally the stem architecture. Finally, we want exactly one global descriptor $\boldsymbol{d} \in \mathbb{R}^{D_o}$ for each point cloud, which can be used for place recognition based on descriptor similarity. Consequently, we look for an aggregation method $A : \mathbb{R}^{M \times D_o} \to \mathbb{R}^{D_o}$ that takes those local features as input and computes a single global descriptor.

There exist multiple methods to fuse the information from multiple features into a single one. One of the most simplistic ones is global max pooling as used in PointNet [180], while in the place recognition domain many approaches [130, 237, 294] utilize the more sophisticated NetVLAD [3] layer for aggregating the features.

Here, we want to investigate a different way to aggregate local point features. We propose a novel attention-based aggregation method based on Perceiver [101], which is a variant of the Transformer architecture [238]. A short summary of the Transformer architecture and the attention mechanism is given in Section 2.2.3 and is recommended to be read for this section.

The attention mechanism in the Transformers computes features $F_t \in \mathbb{R}^{N_t \times D}$ by a linear combination of a set of value vectors $V \in \mathbb{R}^{N_s \times D}$. Where the weighting $W \in \mathbb{R}^{N_t \times N_s}$ of the value features depend on the outer product of the queries

$\mathbf{Q} \in \mathbb{R}^{N_t \times D}$ and the keys $\mathbf{K} \in \mathbb{R}^{N_s \times D}$

$$\mathbf{F}_t = \mathbf{W}\mathbf{V} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D}}\right)\mathbf{V}, \tag{6.5}$$

where the keys $\mathbf{K} = \mathbf{W}_k\mathbf{F}_s$ and values $\mathbf{V} = \mathbf{W}_v\mathbf{F}_s$ are linear projections of the source features $\mathbf{F}_s$, while the queries $\mathbf{Q} = \mathbf{W}_q\mathbf{F}_t$ are projections from the target feature $\mathbf{F}_t$. The softmax ensures that the weights of the linear combinations sum up to one. For the case of self-attention, where $\mathbf{F}_s = \mathbf{F}_t$ and thus $N_s = N_t$, the weight matrix $\mathbf{W}$ grows quadratically with respect to the number of features. Since point clouds usually have thousands to millions of points, this is too memory-expensive for most modern GPUs. Therefore, instead of doing self-attention on the input feature, Perceiver [101] uses cross attention with a few latent vectors as target feature $\mathbf{F}_t \in \mathbb{R}^{N_t \times D}$ with $N_t \ll N_s$. These latent vectors are learned and optimized while training. Multiple self-attention layers use a cross-attention block between the input feature and the latent vectors on the latent features $\mathbf{F}$. Since N_t is a constant and not depending on the number of points N_s, the computational complexity and memory consumption decreases from $\mathcal{O}(N_s^2)$ to $\mathcal{O}(N_s)$.

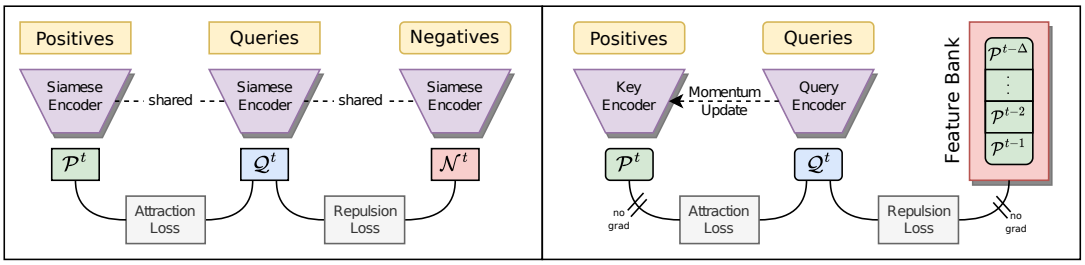
In our application, we use the features coming from our feature propagation network as input sequence for the Perceiver $P : \mathbb{R}^{N_s \times D_{\text{in}}} \mapsto \mathbb{R}^{N_t \times D_{\text{out}}}$. Our Perceiver uses two cross attention blocks for propagating the input features to the latent vectors, where each cross attention block is followed by 4 self-attention blocks working solely on the latent features. All perceiver blocks are also implemented in a residual fashion blocks [86]. A fully connected layer projects the flattened latent features in the end to the desired output dimension of the global descriptor $\boldsymbol{d} \in \mathbb{R}^{D_o}$:

$$\boldsymbol{d} = \mathbf{W}_g\mathbf{f} + \boldsymbol{b}_g, \tag{6.6}$$

with $\mathbf{W}_g \in \mathbb{R}^{D_o \times N_t \cdot D_{\text{out}}}$, and $\boldsymbol{b}_g \in \mathbb{R}^{D_o}$. Note that each operation is permutation invariant, which makes it perfectly suited for the unordered nature of point clouds. As the Perceiver always produces a fixed output feature independent of the number of inputs, we are able to process point clouds of arbitrary sizes.

Both, our Perceiver and the NetVLAD layer aggregate local features relative to a common global context. This feature representation is, in the case of the NetVLAD layer, a set of centroids that are learned while training. For the feature aggregation, they accumulate for each centroid the residual to each input feature weighted by their reciprocal squared distances.

Perceiver uses the latent vectors as a global context. Instead of accumulating the residuals, it recombines the input features using the cross-attention mechanism. NetVLAD stops at this point with the feature propagation and uses a



(a) Training **without** a Feature Bank (b) Training **with** a Feature Bank

Figure 6.4: Training procedures without (a) and with (b) a feature bank. Both methods use an attraction loss to make descriptors from the queries $\mathcal{Q}$ to the corresponding positives $\mathcal{P}$ similar while repulsing the negatives $\mathcal{N}$ away from $\mathcal{Q}$. When using the classical training procedure (a) one computes the queries $\mathcal{Q}$, positives $\mathcal{P}$, and negatives $\mathcal{N}$ all with the same siamese network. The network is trained by backpropagation through all the descriptors. With the feature banks (b) a second network (query encoder) is introduced to compute the positives $\mathcal{P}$. This network gets updated using a momentum update rather than backpropagation. The negatives are not computed per query but are taken from the past positives. The superscript denotes the index at which time t the descriptors are computed.

fully connected layer for aggregating the global descriptor. In contrast to that, our Perceiver uses multiple self-attention and cross-attention blocks allowing it to propagate information also between the latent features for a more refined representation.

While our Perceiver-based aggregation module has shown superior performance for our simplistic Retriever network, the performance does not improve for our KPPR architecture. Therefore, the Retriever uses our Perceiver-based aggregation module, while KPPR uses the basic NetVLAD aggregation, to save compute where possible.

6.2.4 Feature Banks and Momentum Encoder

The success of momentum contrast [85] in the domain of unsupervised representation learning motivated us to apply the ideas of a feature bank and momentum encoder to the field of point cloud-based place recognition.

Traditionally, the positives $\mathcal{P}$ and negatives $\mathcal{N}$ are computed by the same network as the query $\boldsymbol{q}$ (note that we talk here about our query location and not the query features of the Transformer), as visualized in Figure 6.4 (a). The problem is that one requires a lot of negatives to find hard cases. Most locations look completely different, therefore it is easy for the network to generate dissimilar descriptors of locations. The hard part is to distinguish between point clouds which look similar but are actually from different locations. Traditionally, this is solved by mining a lot of negatives and checking which ones are actually

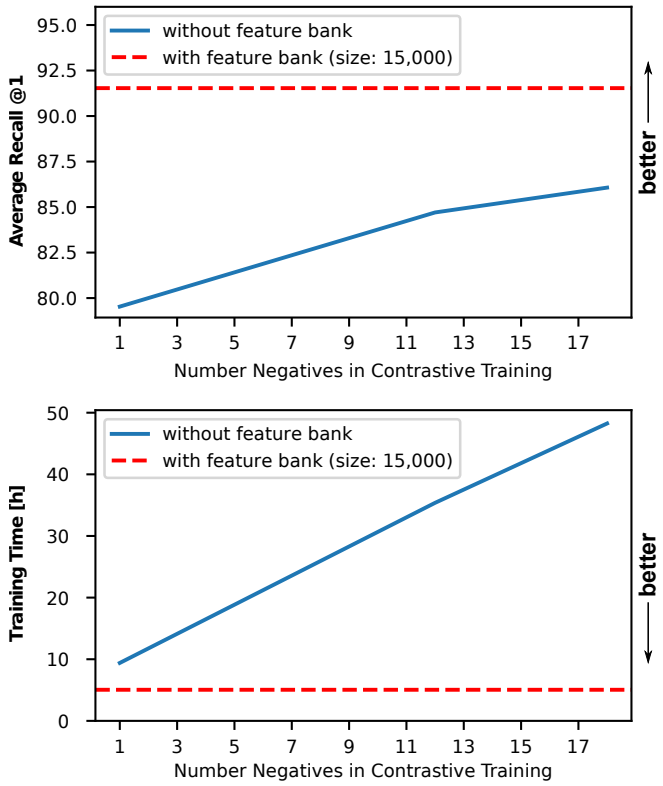


Figure 6.5: Training time with the feature bank versus without at varying numbers of negatives for the hard negative mining. The conventional method, without the feature bank, requires more negatives to increase the performance, but at the same time also increases the training time. Due to the recycling of the old negatives, the feature bank does not need to recompute negatives for the mining and thus can use substantially more negatives (15,000). The training with a feature bank of 15,000 old features is still faster than even recomputing a single negative, thus not only increasing speed but also performance.

hard. More concrete, for each query location, around 20 negative descriptors are computed, while in the end the loss is only back-propagated through the hardest one. So the majority of computations are used to mine those negatives where most of them are not even used. A graphical illustration of the performance and the training time for our network is depicted in Figure 6.5. Generally speaking, the more negatives are mined, the better the performance, but also the slower the training due to the hard negative mining.

In contrast, when using the feature banks, the negatives do not have to be computed online, but rather the former positives get recycled. The feature bank $\mathcal{B}$ is a queue of descriptors, buffering the latest Δ descriptors from the previous positive examples $\mathcal{P}$. By this, the computation of the negatives can be bypassed and therefore saves a lot of computations.

Our loss will try to maximize the similarity between the query and the positives, while minimizing it between query and negatives. By rapidly changing the weights of the network, the positive descriptors can simply diverge from the negatives, just learning that the current descriptors should be different from the negatives. To prevent this, He et al. [85] introduced a second encoder, the so-called key encoder for computing the positives $\mathcal{P}$. The weights w_{key} of the key encoder are not updated by backpropagation but with a momentum update of the weights w_{query} from the query network:

$$w_{\text{key}}^{t} = \gamma\, w_{\text{key}}^{t-1} + (1-\gamma)\, w_{\text{query}}^{t}. \tag{6.7}$$

The superscript denotes the index of the batch. The weights will be updated at the beginning of the forward pass of each batch. Only slowly updating the key encoder shall prevent the divergence of the positives with respect to the negatives [85]. The gradients from the negatives in the feature bank are disabled, which additionally saves computations and memory. Performing backpropagation through tenths of thousands of descriptors would be computationally infeasible, even for most modern accelerators. The pipeline for training with the feature banks is illustrated in Figure 6.4 (b). In contrast to He et al. [85], our approach is supervised, therefore allowing us to only use descriptors of the feature bank as negatives when they are true negatives. For this, we additionally store the indices of the point clouds in the feature bank and check for each query which descriptors belong to point clouds from negative positions, which will then be accounted for in the loss, see Equation (6.10). By using the feature banks, we are able to increase the number of negatives drastically without scaling up in compute and memory. For inference, we only require the query encoder, therefore we can discard the key encoder after training. Remark, that the key encoder was only needed to stabilize the training and prevent the descriptors from diverging.

The relatively small Retriever network can be trained in a reasonable time using the classical hard negative mining method, due to the lightweight architecture. Because of the more sophisticated and compute-demanding architecture of KPPR, which has the convolutional stem, we utilize the feature bank to speed up its training.

6.2.5 Loss Function

The global descriptors that we extract from the point clouds should have the property that descriptors from the same location (positives) should be similar while being dissimilar to descriptors from other locations (negatives). In the following, we will look at two different loss functions to enforce the networks to learn the desired property.

6.2.5.1 Lazy Quadruplet Loss

The most common loss function in point cloud-based place recognition is probably the Lazy quadruplet loss proposed by Uy et al. [237]. Given a query descriptor $\boldsymbol{q} \in \mathbb{R}^{D_o}$ as well as a set of positive $\mathcal{P} = \{\boldsymbol{p}_i\}$, $i = 0, \ldots, N^+$ and negative $\mathcal{N} = \{\boldsymbol{n}_j\}$, $j = 0, \ldots, N^-$ examples, the lazy quadruplet loss is defined as

$$\mathcal{L} = \max(\delta^+ - \delta^- + m_1, 0) + \max(\delta^+ - \delta^* + m_2, 0), \tag{6.8}$$

where $\delta^+ = ||\boldsymbol{q} - \hat{\boldsymbol{n}}||_2$ is the Euclidean distance between the query $\boldsymbol{q}$ and the hardest positive example $\hat{\boldsymbol{n}}$. Consequently, $\delta^- = ||\boldsymbol{q} - \hat{\boldsymbol{n}}||_2$ is the distance to the hardest negative $\hat{\boldsymbol{n}}$ and $\delta^* = ||\hat{\boldsymbol{n}} - \hat{\boldsymbol{d}}^*||_2$ is the distance between the hardest negative and a second negative $\hat{\boldsymbol{n}}^* \in \mathcal{N}$. The second negative $\hat{\boldsymbol{n}}^*$ is not only far away from the query $\boldsymbol{q}$ but also from the hard negative $\hat{\boldsymbol{n}}$. By this, the loss minimizes the distance between positive pairs and tries to maximize the distance to the negative examples. The second negative is used to keep the distance from other negatives that are also structurally dissimilar, therefore helps to distribute the negatives better without letting the negatives collapse. The margins m_1 and m_2 are used to prevent the network from pushing the vectors as far as possible apart. Our goal is to converge towards a feature space that is well distributed such that structural point clouds are located together in the feature space, while dissimilar point clouds are located in different regions of the feature space.

We use this commonly used loss function to train our Retriever network. However, our special feature-bank-based training strategy, which we discussed in Section 6.2.4, can not back-propagate through the negatives. Therefore, we can not use this method for KPPR where we use the feature bank. The next loss function we will look at has this in mind and will be the choice when training KPPR.

6.2.5.2 Additive Supervised Contrastive Loss

For training with the feature bank, we propose to use the additive supervised contrastive loss with differential entropy regularization similar to El-Nouby et al. [55]. For a query descriptor, $\boldsymbol{q} \in \mathbb{R}^{D_o}$, a set of positive descriptors $\mathcal{P}$ and the feature bank $\mathcal{B}$ the contrastive part $\mathcal{L}_c$ is defined as

$$\mathcal{L}_c(\boldsymbol{q}, \mathcal{P}, \mathcal{B}) = \frac{1}{|\mathcal{P}|} \sum_{\boldsymbol{p} \in \mathcal{P}} \left(1 - \boldsymbol{q}^\top \boldsymbol{p}\right) + \frac{1}{\eta} \sum_{\boldsymbol{b} \in \mathcal{B}} \mathbb{1}_{\mathbf{b}} \boldsymbol{q}^\top \boldsymbol{b}, \tag{6.9}$$

where $\mathbb{1}_b$ indicates whether $\boldsymbol{b}$ is in the negatives $\mathcal{N}$ of $\boldsymbol{q}$ and is within the margin β, i.e.,

$$\mathbb{1}_{\mathbf{b}} = \begin{cases} 1 & \text{, if } \mathbf{b} \in \mathcal{N} \text{ and } \boldsymbol{q}^\top \boldsymbol{b} > \beta \\ 0 & \text{, otherwise,} \end{cases} \tag{6.10}$$

and $\eta = \sum_{\boldsymbol{b} \in \mathcal{B}} \|\mathbb{1}_{\mathbf{b}}\|_1$ is the number of times, where $\mathbb{1}_{\mathbf{b}}$ is 1. Intuitively, we try to pull the positives to the query, while pushing the negatives away from the query.

The regularization loss is an entropy-based repulsion loss $\mathcal{L}_r$ to prevent descriptors from collapsing

$$\mathcal{L}_r(\boldsymbol{q}, \boldsymbol{d}^*) = -\log\left(\frac{1 - \boldsymbol{q}^\top \boldsymbol{d}^*}{2}\right), \tag{6.11}$$

where $\boldsymbol{d}^* = \{\text{argmax}(\boldsymbol{q}^\top \boldsymbol{d}) \mid \boldsymbol{d} \in \mathcal{P} \cup \mathcal{B}\}$ is the most similar descriptor. In contrast to El-Nouby et al. [55], we use the cosine distance instead of the Euclidean distance to reuse intermediate results of $\mathcal{L}_c$. The final loss $\mathcal{L} = \mathcal{L}_c + \alpha \mathcal{L}_r$ is the sum of both loss terms with α to weight them accordingly.

6.3 Experimental Evaluation

The main focus of this chapter lies in localization using point cloud-based place recognition. For the evaluation, we investigate the performance of our approach for point cloud-based place recognition in terms of recognition accuracy and training efficiency. Furthermore, we show ablation studies to validate our design choices.

6.3.1 Experimental Setup

The aim of our approach is to reliably retrieve the position in a given map based on the point cloud of the local surrounding. For the evaluation, we use the Oxford Robotcar [143] dataset for training and validation, as well as the three sequences university sector (U.S.), residential area (R.A.), and business district (B.D.) [237] for testing. We follow the common train/test splits and the evaluation metric average Recall R at a specific threshold τ as in [237]. The recall will be denoted as $R@\tau$, e.g., $R@1$ is how often a positive database descriptor is within the top-1 most similar descriptors while $R@1\%$ denotes within the top-1%. Additionally, we will provide the training time for the ablation studies, since one of our key aims is to reduce the training time.

In the previous Section 6.2, we have discussed different building blocks for local feature and global descriptor generation. We can assemble different network configurations with different training strategies and loss functions using those blocks. In the following, we want to summarize the two network architectures and the used parameters, respectively. Retriever [258] is our first more simplistic version, while KPPR [261] is our follow-up work with a more sophisticated network and training schedule.

6.3.1.1 Retriever

Retriever [258] is our first work on place recognition in our compressed map representation and can be seen as a proof of concept. It tries to answer if our compressed map representation can be used for place recognition, and by this bypass decompression. The main idea is to directly use the compressed features for the feature aggregation. The network consists of our feature propagation network, as discussed in Section 6.2.1, followed directly by the attention-based feature aggregation from Section 6.2.3. For the training, we use the standard training technique of hard-negative mining and supervision with the Lazy Quadruplet loss as described in Section 6.2.5.1. Due to the small network architecture and relatively fast training, we do not require the feature bank for training in a reasonable time.

We use AdamW [134] with a learning rate of 10^{-3} and weight decay of 0.01. For the lazy quadruplet loss, we use the margins $m_1 = 0.5$ and $m_2 = 0.2$. We use a batch size of 8 in all our experiments and use $N^+ = 2$ positive and $N^- = 18$ negative examples for the lazy quadruplet loss.

6.3.1.2 KPPR

Our second work, Kernel Point Place Recognition or KPPR, is more sophisticated with a bigger and more complex network architecture, as well as an improved training strategy. For the feature generation, we not only use the small feature propagation network from Section 6.2.1, but also the convolutional stem as discussed in Section 6.2.2 to further enhance the features. For the feature aggregation, we utilize the standard NetVLAD layer. The increased compute demand of the bigger network architecture motivated us to adapt the feature bank idea for place recognition, as discussed in Section 6.2.4. Due to feature banks, we use the additive loss as defined in Section 6.2.5.2.

In more detail, in the experiments, we use the following parameters unless stated differently. The output feature size of the feature propagation network and the stem blocks, as well as for the global descriptor are set to $D_o = 256$, while the intermediate kernel point convolution outputs 128-dimensional features (similar to [226]), to save compute. We normalize the input coordinates to lay within -1 and 1. The radius for the convolution is set to $r = 0.05$. We use $J = 7$ stem blocks and a Feature Bank size of $|\mathcal{B}| = 15,000$. The weights w_q get updated with a momentum of $\gamma = 0.999$ and for the loss, we use $\alpha = 0.3$ as well as $\beta = 0.5$. To enable batched training, we pad the compressed point clouds to always have the same number of points. Padded points get masked out in the blocks accordingly to not affect the training. We use AdamW [134] with a learning rate of 10^{-5}, which will be reduced to 10^{-8} in a cosine annealing

Table 6.1: Average Recall @1% on different datasets

Method	Oxford	U.S.	R.A.	B.D.
PointNetVLAD[1] [237]	85.21	74.80	73.39	71.96
PCAN[2] [294]	83.81	79.05	71.18	66.82
HiTPR[2] [90]	94.64	94.01	89.11	88.31
LPD-Net[2] [132]	94.92	96.00	90.46	89.14
SOE-Net[2] [268]	96.40	93.17	91.47	88.45
SVT-Net[3] [57]	97.80	96.50	92.70	90.70
MinkLoc3D[3] [116]	**97.90**	95.00	91.20	88.50
Retriever (Ours)	92.22	91.88	87.44	85.53
KPPR (Ours)	97.08	**98.01**	**95.10**	**92.09**

schedule. When training with the feature bank, we use a batch size of 32 while only being able to use a batch size of 16 for batch negative mining and a batch size of 3 when training without the feature bank.

6.3.2 Place Recognition Performance

In the first experiment, we analyze the performance of our approach with respect to the baselines. In Table 6.1 are the average recall rates @1% for different datasets shown, while the networks are only trained on Oxford Robotcar [143]. This means the percentage of queries that can find a true positive match within the top 1% of the database size. Our more sophisticated KPPR approach is able to outperform all the approaches on the three test datasets (U.S., R.A. and B.D. [237]). Being between 2.0 and 3.6 percent points better than the second-best approaches on those datasets. Remark that our representation can achieve this on top of our compressed representation, therefore would enable coarse localization in a compressed map. On the other hand, the baselines have access to the full input point clouds. Our rather simplistic Retriever network can outperform other basic techniques like PointNetVLAD [237] and PCAN [294] but lacks behind the methods with more sophisticated network architectures and training strategies. Showing that using our compressed feature representation can even be directly be used for place recognition. In Figure 6.6 are the results from KPPR for the

[1]Approach has been retrained using our own implementation. Due to better results, we report those numbers rather than the original ones from the paper.

[2]Numbers provided by the authors or from the original repositories.

[3]Numbers from the original papers.

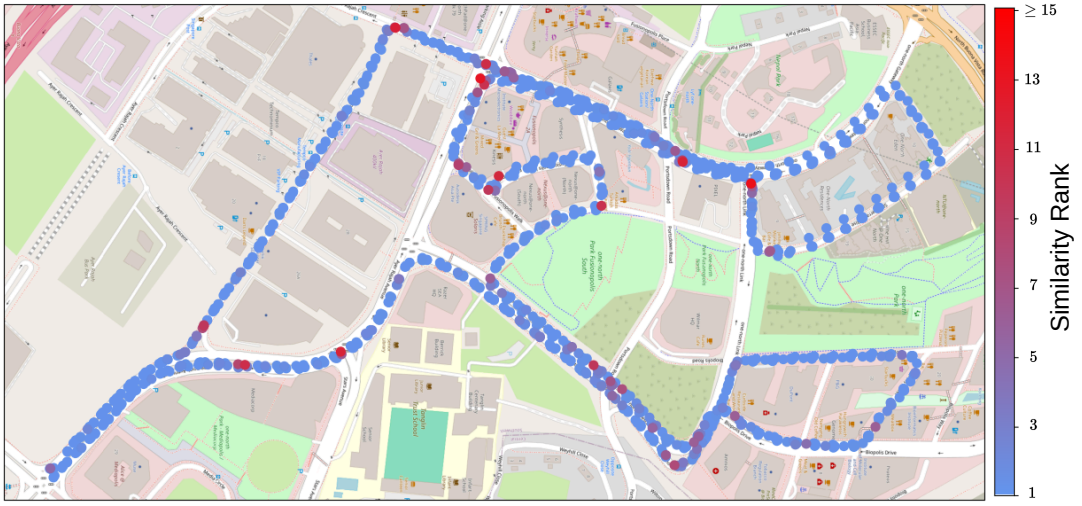


Figure 6.6: Qualitative results for our approach in a business district (B.D.) [237] area. The color of the points denotes at which position the most similar positive was ranked within the database (the brighter the better). Most of the positions are successfully retrieved by only looking at the most similar descriptor.

B.D. [237] dataset visualized. We can see that our approach seems to work very well on straight streets but performs worse on crossings.

For completeness, we also show the validation results on the Oxford Robotcar dataset, where KPPR ranked third. For a more in-depth analysis we can see in Figure 6.7 the recall curves at different thresholds (percentage of successfully finding a true positive in the top N). We can see that the most similar descriptor corresponds 80% of the time to the closest match for the Retriever, and even 90% of the time for KPPR.

6.3.3 Ablation Studies on Retriever

In this section, we validate the choices made in our architecture and evaluate the importance of each part of the Retriever network. First, we look at the impact of using the compressed feature representation produced by our convolutional compression encoder. After this, we compare our Perceiver-based aggregation module with the current state-of-the-art NetVLAD layer, which is typically used in the place recognition domain. For this, we have trained different network architectures. The results on the Oxford Robotcar dataset are shown in Figure 6.8. We either use the original point cloud or the compressed representation (denoted as "Compr") after the encoder as input to the PointNet block or directly to the Perceiver aggregation. For aggregating the local features to a global descriptor, we either use NetVLAD or our proposed Perceiver-based module.

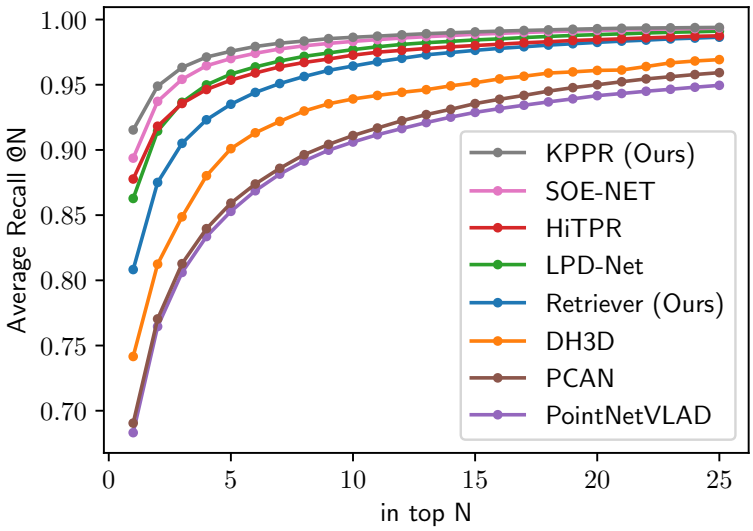


Figure 6.7: Average recall @N on the Oxford Robotcar dataset. Our approach is able to reliably retrieve the position of query point clouds in a given database.

As we can see in Figure 6.8, using the compressed features in both the NetVLAD aggregation and our aggregation module achieves better results than using the raw input points (compare blue to red and orange to purple). This suggests that using the compressed feature representation is not only advantageous for storage or transmission due to the low memory footprint, but also the information of the local neighborhood makes the global descriptors more distinct. Additionally, it mitigates the information loss due to compression.

Exchanging the NetVLAD layer by the Perceiver-based aggregation module increases the performance for the point and compressed feature-based versions (compare Figure 6.8 blue to orange and red to purple). The NetVLAD layer aggregates the local features without considering any relation between the features. The attention mechanisms in the Perceiver allow for suppressing unimportant features and concentrating more on especially descriptive features. Additionally, the self-attention of the latent features incorporates information from the whole sequence and can thus change the features based on the global context. This potentially helps to describe the point clouds more accurately and, therefore, increases the place recognition performance. Directly aggregating the compressed features to a global descriptor without feeding it to the PointNet yields worse results, showing that transforming the compressed features to a task-specific representation is advantageous (Figure 6.8 green and purple).

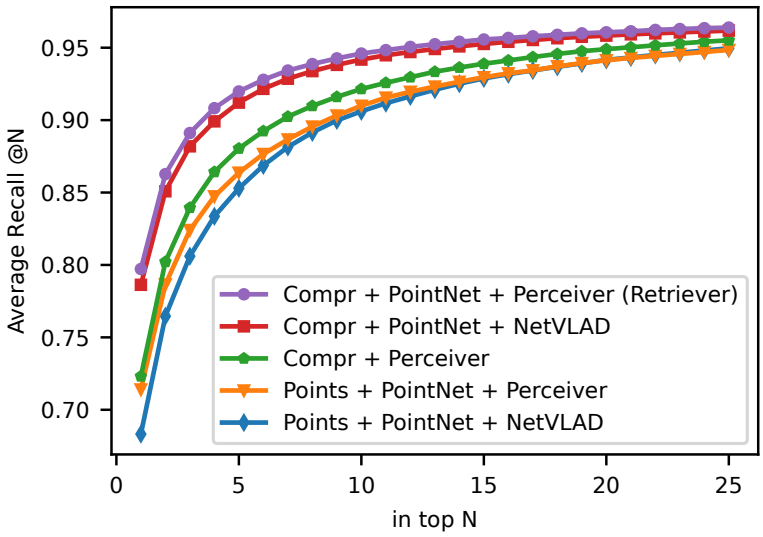


Figure 6.8: Average recall @N on the Oxford Robotcar dataset for different network configurations of the Retreiver. Replacing the raw input point cloud (Points) with our compressed feature representation (Compr) increases the retrieval accuracy. Similarly, exchanging the NetVLAD layer with our Perceiver-based aggregation module allows also for better place recognition.

6.3.4 Ablation Studies on KPPR

In the following, we will present ablation studies to show the impact of the proposed design choices made for our KPPR method. The ablation studies have been evaluated using a second feature bank of size $|\mathcal{B}| = 500$ on the validation dataset Oxford Robotcar [143].

6.3.4.1 Negative Mining and Loss Function

In this experiment, we analyze the impact of different mining strategies. The first part focuses on the performance and how it is affected by the choice of the loss function, while the second part focuses on the impact on the training time. We show in Table 6.2 the results for our network trained with different loss functions and under different mining strategies. Namely, first the classical method of computing for each query 18 negatives ([A] - [C]). The second method is batch negative mining where the negatives are searched within the positives of the batch ([D] - [F]). For a batch size of 16 with two positives per query, we have 32 negatives. False negatives are masked out in the same way as for the feature banks, as in Equation (6.10). Finally, using the feature banks (FB) as described in Section 6.2.4 ([G] - [I]). For the loss functions, we evaluate with the lazy Triplet [237], lazy Quadruplet [237], and additive contrastive loss [55] with (here denoted as Entropy) and without (denoted as Contrastive) entropy

Table 6.2: Ablation: Negative mining and loss function

	Loss	Mining	R@1	R@1%	Time
[A]	Triplet	Classic	85.49%	93.59%	95.36h
[B]	Quadruplet	Classic	84.94%	93.74%	102.06h
[C]	Entropy	Classic	86.07%	94.63%	96.50h
[D]	Triplet	Batch	86.41%	93.96%	6.68h
[E]	Contrastive	Batch	85.69%	93.68%	6.93h
[F]	Entropy	Batch	89.04%	95.74%	7.71h
[G]	Triplet	FB	89.91%	95.71%	5.49h
[H]	Contrastive	FB	88.73%	95.51%	5.50h
[I]	Entropy	FB	**91.53%**	**97.08%**	**5.05h**

regularization. Notably, the Quadruplet loss cannot be trained with the feature bank since the descriptors of the negatives do not have gradients.

Independently of the mining strategy, we can see that the contrastive loss with the entropy regularization ([C], [F], [I]) outperforms the networks trained with different losses. Batch negative mining outperforms the classic method but is worse than using the feature bank. From the training time perspective, we see the huge impact of using a mining strategy rather than computing the negatives classically. Additionally, the performance increases, which is likely due to the higher amount of negatives (FB: 15,000; Batch: 32; Classic: 18) as well as a bigger batch size (FB: 32; Batch: 16; Classic: 3), therefore compensating for the problem of comparing descriptors that got computed at different stages in the training. The training time using the feature banks is lower than the batch negative mining, even though, the substantially larger amount of negatives. The reason for this is that the gradients are not computed for the negatives in the feature bank but are still needed for batch negative mining. When comparing the contrastive loss with entropy regularization [I] against the training without the regularization [H], we can see that the regularization boosts the performance. Throughout our experiments, we did not see that the regularization also helps with the other losses. Training with the exponential contrastive loss [112] was more unstable and led to worse results in our experiments.

6.3.4.2 Feature Bank Size

In Table 6.3 are the results with respect to varying feature bank sizes. The best result can be achieved by a feature bank with 15,000 descriptors [K], which corresponds to 2/3 of the size of the training set. As we can see the training time does not vary a lot for changing feature bank sizes. Throughout the experiments,

Table 6.3: Ablation: Feature bank size

	#FB Size	R@1	R@1%	Time
[J]	10,000	91.05%	96.66%	**4.91h**
[K]	15,000	**91.53%**	**97.08%**	5.05h
[L]	20,000	90.44%	96.15%	**4.91h**

Table 6.4: Ablation on the KPPR Architecture.

	#Blocks	Compr.	R@1	R@1%	Time
[M]	7	✗	88.99%	95.66%	10.87h
[N]	0	✓	68.00%	82.04%	**0.21h**
[O]	5	✓	89.83%	95.92%	3.69h
[P]	7	✓	**91.53%**	**97.08%**	5.05h
[Q]	9	✓	90.73%	96.39%	7.02h

where we do not use the feature banks, i.e., when computing the negatives explicitly, it scales linearly with the number of negatives (as in Figure 6.2). Training with more negatives is not feasible due to the limiting memory resources of the GPU.

6.3.4.3 Architecture

In this section, we analyze different parts of our network architecture. In Table 6.4 are the results for different numbers of blocks in the convolutional stem as well as the network without using the pre-trained compression encoder. Seven convolutional blocks provide the best performance, additionally one can see that the training time increases with the number of blocks. Not having any further convolutional blocks [N] provides the worst results, showing that the convolutional stem is necessary to yield good results. On the other hand, it can be trained in under 13 min and already has a top 1 recall of 68%. We can see that using the compression encoder [P] provides better results and trains faster than without [M]. The slowing down of the training process can be explained by the increasing number of points in the point cloud. The decreasing performance shows that the features computed by the compression encoder are more valuable than the higher number of points. The inference of a single compressed point cloud runs at 95 Hz. Precomputing the neighborhoods $\mathcal{N}$ and h in the convolutional stem speeds up the inference by 44.8%, i.e. being almost twice as fast.

Table 6.5: Ablation: Feature bank training with MinkLoc3D backbone

	Loss	Mining	R@1	R@1%	Time
[R]	Triplet	Classic	71.05%	86.71%	30.07h
[S]	Triplet	Batch	73.94%	88.66%	2.72h
[T]	Triplet	FB	77.47%	89.96%	2.17h
[U]	Entropy	Classic	73.41%	89.14%	30.14h
[V]	Entropy	Batch	73.58%	88.86%	2.75h
[W]	Entropy	FB	**78.07%**	**90.97%**	**2.15h**

6.3.4.4 Backbone

In this section, we investigate if we are able to apply the proposed methodology to a different backbone. For this experiment, we exchange the compression encoder plus the stem architecture with the sparse convolutional feature pyramid network from MinkLoc3D [116]. For the network architecture, we use the same hyperparameters as in the original work [116] and also train for the suggested 50 epochs. Based on a parameter sweep, we use a learning rate of 10^{-4} and a batch size of 32 for these experiments. Additionally, we report the results for different negative mining techniques as in Section 6.3.4.1. The results are depicted in Table 6.5. First of all, we can see the same behaviors as for our network, i.e., using the feature banks ([T], [W]) improves both training time and performance. The entropy loss outperforms the triplet loss as used in the original work [116], also for this network architecture. When comparing the performance of our network Table 6.2 [I] with the feature pyramid network Table 6.5 [W], we see that our network provides substantially better results (Mink: 78.07 % versus our: 91.53 %) at the cost of training time (Mink: 2.15 h versus our: 5.05 h). Our proposed network has roughly twice the amount of parameters, explaining the difference in runtime and performance. The results suggest that the proposed methodology might also increase the performance of other networks. Notably, the results with the feature pyramid network do not perform as well as in Komorowski et al. [116]. Likely due to the lack of data augmentation and using a different feature aggregation head, which we did not apply for comparability to isolate the impact of the backbone.

6.4 Conclusion

In this chapter, we investigated if we can use our previously built compressed map representation for place recognition. For tackling the task of place recognition, we use point cloud descriptors that encode the characteristics of the scene into a single feature vector. To retrieve the vehicles position in the map, one can then compare the query descriptor, that is computed on the current vehicles observations, to the descriptors in the global map. For the descriptor generation, we follow the paradigm of first computing local point features, which are then aggregated to the global descriptor per point cloud. Since our compressed feature-based representation contains already local information, it motivated us to investigate if they could also be used as features for place recognition. As our experiments suggest, those features are not only usable, but can even provide information to obtain state-of-the-arts results. Additionally, we can bypass the otherwise needed decompression and thus save compute. Our first work, the Retriever, showed that even with a minimal network architecture and basic training techniques we are able to retrieve most positions correctly. Encouraged by the promising results, we were motivated to develop the more sophisticated approach KPPR. Enhancing the local compressed features with our stem architecture increases the performance drastically. However, the more sophisticated network architecture comes at a price: more computational demand, and therefore drastically increasing training times. To tackle the problem, we adapted the idea of a feature bank for point cloud-based place recognition. This allows us to search in the training for hard cases without the need to recompute the descriptors. It does not only speed up the training remarkably, but also further increases the performance. For us, this led to a significant reduction of the training time by a factor of up to 17 times, which enabled us to run more experiments and have faster research cycles. The presented experiments with the feature banks took around 47 h of training. If we had done those experiments by computing the classical 18 negatives for each query as in literature [132, 237, 258], it would have taken us around 800 h. Our network architecture is designed for efficient inference and training but does not need to specially account for using the feature banks. Experiments with different network architectures suggest that the advantage of the feature banks (more negatives with less compute) can be applied also for other kinds of place recognition architectures and approaches.

In the previous chapters we: (1) calibrated our mobile mapping systems which enables us to obtain measurements of the desired area; (2) processed the collected data to generate a globally and locally consistent point cloud map; (3) which then was split into smaller submaps, which we compressed to obtain a memory efficient representation. In this chapter, we have tackled how we can localize

withinin our compressed representation. For this, we computed for each submap a global descriptor that can be stored alongside. If a robot, e.g., the same as used for data recording, drives at another time through the same area, then we can also compute a query descriptor for the current robot's observations. The robot can estimate its position in the map by finding the submap descriptor that is the most similar to the query. We call this coarse localization since it can only answer us in which part of the map our robot is located, but does not provide a full 6 DoF pose with respect to the map. In the next chapter, we tackle the problem of fine localization, where we want to exactly find the full metric pose. We again want to operate directly on the compressed feature-based representation. Here we looked if we can use the local compressed features for generating place recognition descriptors, while in the following we want to use the local features to find correspondences for registration. Having those feature-based point correspondences can then be used to estimate the transformation between the robot and the map frame. Thus, at the end of the following chapter, we will have the full 6 DoF pose of the robot in the compressed map. Once we have our global pose within the map, we can use our pose-tracking method to track our movement with respect to the map, as we will discuss in the last chapter.

Chapter 7

Registration of Compressed Point Cloud Maps

Point cloud registration is key for many robotic applications such as map matching, pose tracking, loop closure, or simultaneous localization and mapping (SLAM). Outdoor point clouds, as they are commonly used in the domain of autonomous vehicles, are typically large-scale, contain dynamics, and may vary a lot between different recording times. Thus, robust and reliable methods to handle this challenging data are required.

In this chapter, we finalize our method for localization in a compact map representation. While in the previous chapter, we focused on localizing our robot coarsely, here we are interested in estimating the full 6 DoF transformation between the robot and the map. Our map consists of multiple smaller submaps in the form of compressed point clouds. Additionally, we have for each submap the transformation to the global coordinate system and from the last chapter also a place recognition descriptor associated. We can use the descriptors to find the submap our robot is located in by comparing the descriptor generated from the current robot's observations to the submap descriptors. Starting from there, now that we know in which submap our robot is located, we want to precisely estimate the pose with respect to to this submap. This is what we refer to as fine localization in our overview provided in Figure 7.1.

We try to solve the fine localization part by point cloud registration, which estimates the rotation and translation parameters between two point clouds such that they align. This problem can be solved in closed form when having point correspondences between the point clouds. Finding those correspondences reliably is typically the most challenging part. The classic ICP method resolves this problem by iteratively searching for spatially close correspondences in the target point cloud, as well as using those correspondences to update the estimated transformation. This typically converges to a stable solution. The problem with ICP

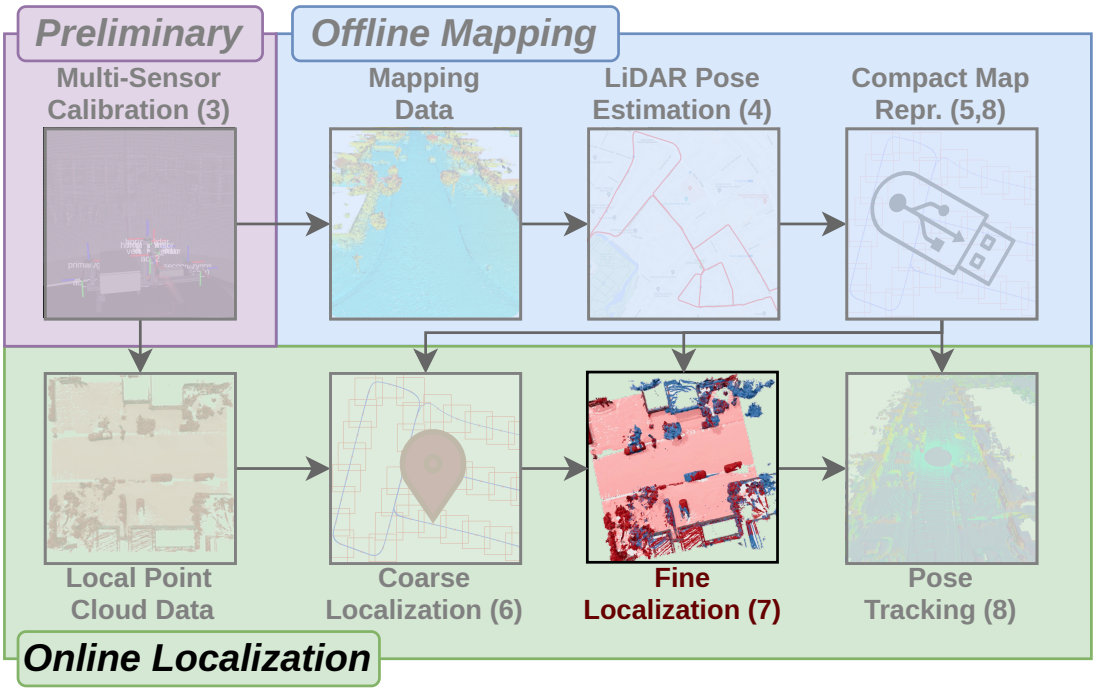


Figure 7.1: In the previous parts, we have investigated how we can build memory-efficient point cloud maps, and how to coarsely localize a robot in those maps. In this chapter, we want to estimate a more precise location of the robot within the map using scan registration.

is that it requires a good initial alignment to converge to the correct solution. In our case, we do not have any prior on the position of the robot in the submap; the coarse localization just provided us with the knowledge that the robot is located in a submap (typically with a size of $40\,\mathrm{m} \times 40\,\mathrm{m}$), but not where in the submap and not the orientation. Thus, the initial alignment can be off by up to 180° and translated several meters within the submap.

To resolve the problem we try to find the point correspondences using feature-based matching. We want to generate distinct features such that points of the same object have a similar feature, while points from different objects are dissimilar. Now instead of matching spatially near points, we would match points that are close in the feature space. As long as the features are transformation invariant, we should be able to find point correspondences reliably even under extreme initial conditions.

As in the previous chapter, we aim to build directly upon the compressed features to avoid decompression. We now want to compute local features for all points that are well-suited for the correspondence search. Here, we can also exploit our compressed representation that already consists of a low resolution point cloud where each point has a feature associated to. We will investigate how we can refine those compression features to be well-suited for feature-based

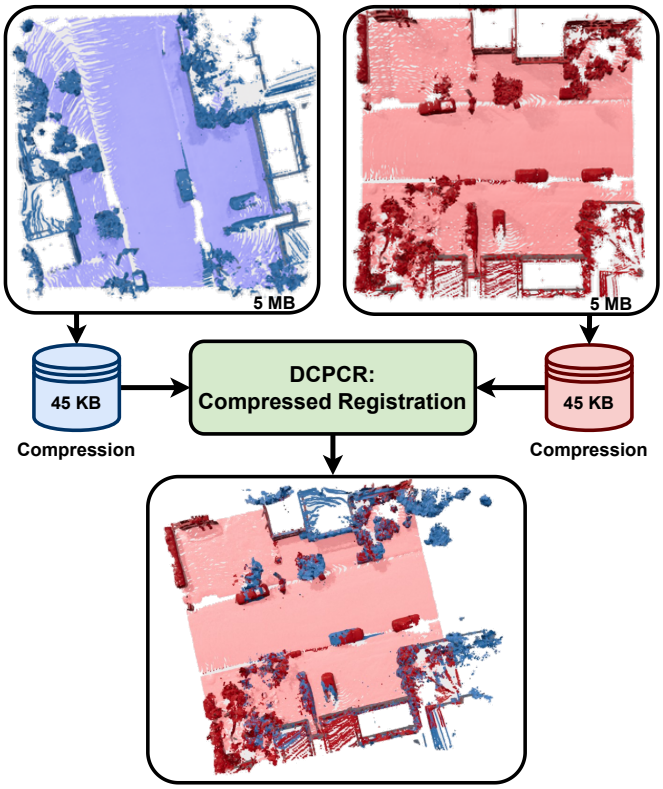


Figure 7.2: In this chapter, we propose our method DCPCR to directly register point clouds on a compressed representation. The point clouds are around 90° rotated and a couple of meters translated to each other. Our method first compresse the point clouds, finally aligns the point clouds directly using the compressed representation. For better visualization, we show the estimated transformation on the input point clouds, rather than the sparse compressed representation.

matching. Additionally, we have to deal with uncertain correspondences due to ambiguities, or certain objects being in one point cloud, but not in the other. Ambiguities are mostly coming from structureless areas, e.g. many points lie on large surfaces like streets or walls. Finding correct point-wise correspondences on those structureless surfaces is very challenging. The absence of parts of one point cloud in the other can be either caused by a physical change (e.g., cars moved, construction of a new building), or simply missing overlap, where both point clouds cover slightly different areas. Therefore, we try to estimate how certain we are with our correspondences to focus on the most distinct matches.

The main contribution of this chapter is an approach to perform global point cloud registration in the context of outdoor environments and which operates on a compressed representation. An example of this can be seen in Figure 7.2. Specifically, we deal with large non-overlapping areas, dynamic scenes, changing environments, while having no prior information about the initial pose. We utilize the compact representation of our prior point cloud compression network to di-

rectly operate on the compact representation and therefore bypass decompression and reduce computations when compression is needed. We build upon the classical SVD-based pose estimation using feature matching with soft assignments. For that, we propose a network architecture that consists of a convolutional backbone with a Transformer [238] head to produce distinct features, which can directly be optimized for point cloud registration. Additionally, the network learns to estimate the quality of the correspondences to focus on points that are well-suited for the registration. We investigate the performance of our approach when targeting memory-constrained registration, as well as point cloud registration in the classical setup without compression.

Our proposed point cloud registration method is able to register dense point clouds without requiring an initial pose estimate, even when the point clouds only partially overlap and were recorded at different points in time. Exploiting our compressed point cloud registration allows us to directly register compressed maps without decompression.

7.1 Related Work

In this chapter, we want to estimate the transformation between two point clouds such that they are aligned. This task is referred to as point cloud registration, which we have seen already in Chapter 4 for aligning the mapping data. The main difference to Chapter 4 is, that we now want to register two dense $40\,m \times 40\,m$ submaps from different points in time, without prior knowledge about initial pose estimates, while previously we registered sequential scans with relatively good initial estimates. For related work on the topics of sequential alignment, and the registration of multiple scans, we refer to Section 4.1. In this section, we will summarize the field of local registration in preparation for the subsequent section on global registration.

7.1.1 Local Registration

The most common approach for aligning two point clouds is ICP [21] with its variants [24, 193, 201]. Here, the main challenge is to find the correct correspondences. Looking only at spatially close points often fails when the initial guess is too far from the correct transformation. Geometric [64] or photometric [105, 111, 174] features are often used to find suitable correspondences and to resolve ambiguities. Projective data association [13, 193, 202] can be applied to speed up the correspondence search, which usually takes a relatively large proportion of the computation time. Point-to-plane [36] and GICP [201] on the other

hand try to relax the assumption of point correspondences for faster convergence and more precise results.

Outdoor environments often change, and therefore, assuming to have a lot of one-to-one correspondences does not necessarily hold. Robust optimization [30, 65, 97] or a careful correspondence selection [193] is often needed to overcome this issue.

Deep learning-based approaches have the advantage of computing more distinct and sophisticated features than their handcrafted counterparts. Some learning-based approaches [140, 253, 279] rely on the SVD-based closed-form solution and soft assignments for end-to-end learning, as also our approach. Other approaches [2, 196] try to estimate the transformation directly in the network based on global features and thereby bypass the need to find suitable correspondences. In general, local registration methods require a quite good initial transformation to converge to the desired solution.

7.1.2 Global Registration

Global registration methods try to estimate the pose between the point clouds without requiring a good initial guess of the transformation.

RANSAC [59] provides the opportunity to deal with large transformations as well as outliers by repeatedly sampling correspondences, estimating the transformation, and returning the most supported solution. The correspondences are found through feature-based matching. For the matching, handcrafted features like spin images [104], FPFH [194], or SGC [223] can be used. Contrastive learning is a powerful tool to train neural networks to generate descriptive features for matching [8, 39, 46]. However, these approaches require ground truth correspondences to learn which features should be similar, and which ones not. While this is especially easy to realize for simulated scenes, on real world data this would require manual annotations or heuristics. One method to estimate correspondences for the training is to utilize 3D reconstruction [278, 289]. A contrastive loss [289] allows for self-supervised training to enforce similar features for similar areas, while dissimilar features to all other areas at the same time. Pose invariant features [46, 113, 178, 211] have the advantage that by design, can be matched under arbitrary misalignment. These methods extract patches from which descriptors are generated. The patches are transformed in a canonical orientation using the distribution of the points in the patch. The main drawback of estimating such transformation is that it is prone to varying densities, noise, and outliers, which limits its application to point clouds outside simulated data. Data from LiDAR sensors has naturally no homogeneous density due to the measuring process, i.e., the further from the sensor the lower the resolution.

Branch-and-bound (BB) methods [275, 276] generate multiple hypotheses in an evolutional manner and prune the search space. The biggest disadvantage of those methods is typically the high computation time.

When having the information about the transformation between the point clouds at training time, one can directly supervise the pose to get features that are well suited for the matching [140, 279]. Ground truth poses can be either artificially generated using data augmentation, or from SLAM [289]. We utilize the Apollo-Southbay [139] dataset, where the ground truth poses are generated by utilizing multiple sensors in their SLAM system. Some learning-based registration approaches let the network directly predict the pose [47]. Instead of directly estimating the transformation in one shot, or iteratively, recently some approaches [138, 248] propose a coarse to fine registration in a hierarchical manner. DCP [253] is a supervised registration approach, and closely related to our method. It utilizes a graph convolutional neural network and a transformer head to compute features, which are later used to obtain soft correspondences for the registration. The follow-up work PRNet [254] utilizes Gumbel-Softmax [102] for sharper matching. DCP and PRNet are designed and successfully applied on synthetic data with 1,024 points and Gaussian noise, but cannot deal efficiently with a large number of points, or dynamics as they are common in the automotive field. We overcome the scalability problem by integrating our compression network from Chapter 5, which subsamples the point clouds significantly but preserves the local information in the feature representation. To deal with points that do not have a corresponding point in the other cloud, we propose a way to weigh the correspondences. By this, we also do not rely on RANSAC to evaluate the quality of the correspondences [289].

7.2 Feature-based Point Cloud Registration

For the alignment of two point clouds, we follow the classical paradigm of first finding point correspondences, which are then used to estimate the relative transformation. While in the classical ICP, correspondences are determined via geometric neighborhood, we follow a feature-based approach. The transformation consisting of a rotation and a translation is estimated using the closed-form solution [107].

In the following, we will first explain in Section 7.2.1 how to incorporate the soft assignments and the feature-based matching of the attention mechanism [238] into the Kabsch algorithm [107]. Afterward, we discuss our network architecture to compute the features for the point matching (see Figure 7.3). In Section 7.2.3, we will address the problem of ambiguities and errors in the matching by introducing a weight for each correspondence.

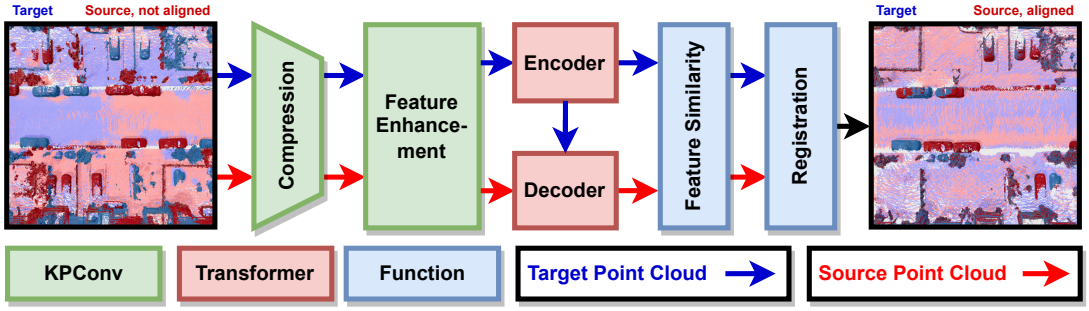


Figure 7.3: Given two unaligned point clouds (target: blue, source: red), we first use a compression encoder to compute local features and to subsample the point clouds. A feature enhancement block increases the receptive field to create more distinct features that serve as input to the transformer heads for global aggregation. The resulting features are used for feature-based correspondence matching. Finally, the soft-assigned source points are aligned to the target point cloud with a point-to-point registration.

7.2.1 Registration of Compressed Point Clouds

Given are two point clouds in different coordinate frames, where each point cloud consists of N_i coordinates $P_i \in \mathbb{R}^{N_i \times 3}$ and their corresponding D_f dimensional features $F_i \in \mathbb{R}^{N_i \times D_f}$. We want to estimate the transformation ${}^zT_s$ from the source frame s to the target frame z, such that both point clouds are aligned. When having a correspondence matrix $W \in \mathbb{R}^{N_z \times N_s}$, we compute the corresponding points $\tilde{P}_z$ of the target points by

$$\tilde{P}_z = WP_s. \tag{7.1}$$

A row in the W matrix represents the corresponding point in $\tilde{P}_z$ by a linear combination of the source points. In the case of one-to-one correspondences, this is a row with 0 everywhere but a 1 at the index of the corresponding source point. The rotation $R \in SO(3)$ and translation $\boldsymbol{t} \in \mathbb{R}^3$ of the transformation ${}^zT_s$ can then be estimated by the Kabsch algorithm [107] using SVD, as follows:

$$C = (P_z^{*T}\tilde{P}_z^*) \overset{\text{SVD}}{=} UDL^\top \tag{7.2}$$

$$R = UL^\top \tag{7.3}$$

$$\boldsymbol{t} = \boldsymbol{x}_z^* - R\tilde{\boldsymbol{x}}_z^*, \tag{7.4}$$

where P_i^* denotes a point cloud shifted by its mean $\boldsymbol{x}_i^*$ such that the point cloud is centered around the origin. When dealing with uncertainties and partially overlapping point clouds one can incorporate a weight $\boldsymbol{w} \in \mathbb{R}^{N_z}$ for each correspondence by computing weighted means in Equation (7.4) and a weighted cross-covariance C in Equation (7.2).

Since the correspondences W are usually unknown, one estimates them by, e.g., nearest neighbors, feature-based matching, etc., as in classical ICP meth-

ods [193]. In this chapter, we use feature-based matching by utilizing the attention mechanism [7] as used in the Transformer [238]. For an explanation of the attention mechanism and Transformer architecture we refer to Section 2.2.3. It is defined by three matrices, namely the queries Q, keys K and values V, where the attention A is computed by

$$A = \text{attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{D_f}}\right)V. \tag{7.5}$$

When choosing the target features as queries $Q := F_z$, the source features as keys $K := F_s$ and the source coordinates as values $V := P_s$, we can rewrite Equation (7.5) resulting in the desired correspondences in Equation (7.1) with $A := \tilde{P}_z$ as

$$\tilde{P}_z = \text{softmax}\left(\frac{F_z F_s^\top}{\sqrt{D_f}}\right)P_s = WP_s. \tag{7.6}$$

In other words, we are computing a soft assignment between the target points P_z and the source points P_s, based on the scaled cosine similarity between their features F_z and F_s. In the following, we present our network architecture for computing the point features.

7.2.2 Feature Generation

Our network architecture consists of three parts, namely a compression encoder for memory and computational efficiency, a convolutional block to increase the receptive field, and a transformer head for global feature aggregation.

7.2.2.1 Compression Network

Point clouds obtained with modern LiDAR scanners easily contain multiple hundred of thousands of points, which is for most networks infeasible to process. Our in Section 7.2.1 described attention-based registration relies on cross attention between the source and target point cloud and therefore grows quadratically with the number of points. To overcome this issue, we use our compressed map representation from Chapter 5, which substantially reduces the number of points and provides locally-aware features. We use the memory-efficient representation produced from the convolutional encoder as input for our network, which allows us to run the full registration procedure on the compressed point clouds without the need for decompression.

Additionally, we use a reduced PointNet [180] to transform the compressed point representation to a localization-specific feature space, which is better suited for matching than for reconstruction. We use a similar feature propagation network as in the previous chapter. For more details, we refer to Section 2.2.1.

7.2.2.2 Feature Enhancement Network

The features from the compression network contain local information about the close neighborhood of the points but lack broader context, which might be useful to create more distinct features for better matching and resolving ambiguities. We use additional B_k KPConv [226] blocks in a ResNet-like fashion [86].

To compute richer features we can use the stem architecture which we introduced in Section 6.2.2 to increase the receptive field. Note that we use the same network architecture, but due to the training with a different loss the weights, and thus the resulting features will change. Here, we learn features that are well-suited for matching while in the previous chapter, we optimized the features to generate meaningful global descriptors.

7.2.2.3 Transformer

We utilize a small Transformer head [238] consisting of an encoder and decoder. The encoder operates on the target point features F_z and utilizes multihead self-attention for global feature aggregation.

$$F_z^t = \text{MultiHead}(Q := F_z, K := F_z, V := F_z) \quad \text{with} \tag{7.7}$$

$$\text{MultiHead}(Q, K, V) = [\text{attn}(QW_j^Q, KW_j^K, VW_j^V)]W^O \quad | \, j \in \{1, N_h\}, \tag{7.8}$$

where W^Q, W^K, and W^V are projection matrices of the queries, keys, and values, respectively. W^O projects the N_h concatenated heads to the desired feature dimension.

The decoder uses multi head cross-attention between the source and the target, as follows:

$$F_s^t = \text{MultiHead}(Q := F_s, K := F_z^t, V := F_z^t). \tag{7.9}$$

This decoder transforms the features of the source points into the feature space of the target to increase the feature similarity and therefore should lead to better matching. More information about the Transformer blocks are provided in Section 2.2.3. We do not explicitly add any positional encoding since our generated features already contain some positional information and we do not want to add any bias to the Transformer, which could increase the influence of the positions. Defined by our task, we know that the point clouds are each in their respective coordinate frame, therefore trying to infer information based on the coordinates is meaningless. The final features F_z^t and F_s^t are used for the correspondence matching in Equation (7.6).

7.2.3 Weighting Scheme

The problem with point-to-point correspondences is the ambiguity in the scene. For example, estimating the correct location of a point on a wall or the street is quite challenging. However, we also do not need to find all the correspondences, having a few reliable matches is better than having many, but partially also wrong correspondences. To account for this, we estimate for each correspondence a weight $\boldsymbol{w} \in \mathbb{R}^N_z$.

Intuitively, we have already some kind of measure to see how precise or uniquely a point can be matched by our weight matrix $\boldsymbol{W}$. A row in the weight matrix $\boldsymbol{W}$ tells us for a specific point, how similar its point feature is to the point features in the other point cloud. Note that we normalized these similarities to add up to one. Having a point that can be matched uniquely would correspond to a row with only zeros except a one for the corresponding point. Points which have similar features to a lot of points would have similar but substantially smaller weights for a lot of points. In the end, we want the points that can be matched uniquely, therefore looking for points with a peaky distribution in the weight matrix $\boldsymbol{W}$. In the following, we will discuss different methods to estimate those weights.

7.2.3.1 Maximum

Formulating the idea of a peaky distribution can be done by looking at the weights of the weight matrix $\boldsymbol{W}$. Having a good match would correspond to a high value, i.e., close to one, for one of the candidates. Therefore, the first method we propose is to simply use for each correspondence the maximum value in the respective row of $\boldsymbol{W}$ as weight k_w.

7.2.3.2 Entropy

We can also look at it from the opposite way. This would mean that when having a uniform distribution of the weights, it would correspondend to an ambiguous match. To quantify this, we can use the relative entropy, i.e., Kullback-Leibler divergence, between each row of $\boldsymbol{W}$ and the uniform distribution as a weight k_w. The less our distribution is uniform, the higher will be the weight of the corresponding match.

7.2.3.3 MLP

As the last option, we follow the idea by Tiziano Guadagnino, who proposes to compute the weights based on the correspondence matrix $\boldsymbol{W}$ as follows: "For each correspondence, we feed the top k_w features into an MLP: $\mathbb{R}^{N \times k_w} \mapsto \mathbb{R}^{N \times 1}$

to obtain the respective correspondence weight. The MLP consists of three linear layers with ReLU activation in the first two layers and a sigmoid activation after the last one, to ensure that the weights are between 0 and 1." [257], [79, p.44 f.]

7.2.4 Loss Function

The registration procedure described in Section 7.2.1, the feature generation from Section 7.2.2, and the computation of the weights in Section 7.2.3 are fully differentiable. This allows us to directly optimize for the correct pose to learn features that are well suited for matching and therefore estimating the transformation. We split the loss $\mathcal{L}_T$ into a rotational part $\mathcal{L}_R$ and translational part $\mathcal{L}_t$. For the rotation, we want to minimize the angle between the ground truth $\boldsymbol{R}_{gt}$ and estimated rotation $\boldsymbol{R}_{est}$

$$\mathcal{L}_R = \text{trace}(\boldsymbol{I}_3 - \boldsymbol{R}_{gt}^\top \boldsymbol{R}_{est}), \tag{7.10}$$

and for the translational part the Euclidean distance between the ground truth $\boldsymbol{t}_{gt}$ and estimated $\boldsymbol{t}_{est}$ translation vectors

$$\mathcal{L}_t = \|\boldsymbol{t}_{gt} - \boldsymbol{t}_{est}\|. \tag{7.11}$$

The overall loss $\mathcal{L}_T$ is the weighted sum of both

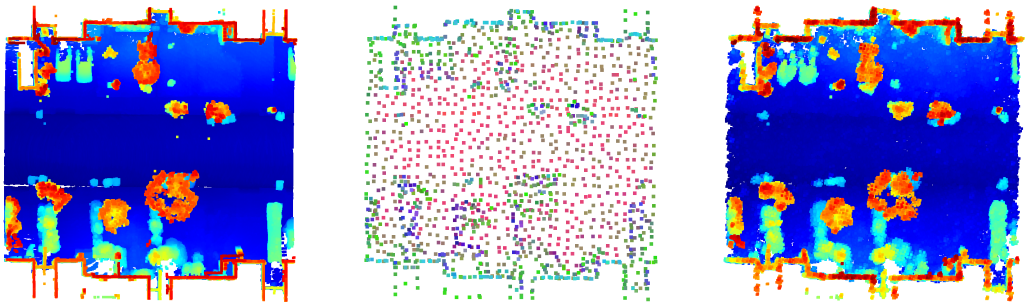
$$\mathcal{L}_T = \alpha_1 \mathcal{L}_R + \alpha_2 \mathcal{L}_t, \tag{7.12}$$

where α_1 and α_2 are the respective weight factors.

7.3 Experimental Evaluation

Our goal is to estimate the transformation between point clouds, as they for example are used in map matching or loop closure detection. For this, we want to estimate the transformation from a local source point cloud to the target point cloud which was recorded at a different point in time. Note that we do not utilize the previous datasets, like KITTI [63] since the sessions do not overlap, or Oxford Robotcar [143] which is the standard for point cloud-based place recognition, but due to the profile scanner not commonly used for scan registration. Instead, for registering point clouds across sessions, we evaluate our approach on the Apollo-Southbay-ColumbiaPark dataset [139], which provides multiple runs, namely a mapping, training, and testing run, as well as consistent ground truth poses.

By this, we can register point clouds from the training or testing runs with the ones from mapping. Registering point clouds recorded at completely different points in time is especially challenging due to objects which moved slightly or miss



(a) Input Point Cloud (b) Compressed Point Cloud (c) Decompressed Point Cloud

Figure 7.4: Visualization of the different point cloud representations in different stages of the network. (a) represents the original data like it is used as input for our network and the baselines for the registration without memory constraints. In (b) the sparse compressed representation is visualized, and colorized by the feature space. In (c) one can see the decompressed point cloud, used for the baselines to register under memory-constrained conditions. The color in (a) and (b) represents the height of the points for visualization purposes.

completely, e.g., cars in parking lots exchanged or moved which often leads to wrong associations. We generate local map patches as done in other works [237, 260] by aggregating the scans within a 2 s timeframe and a bounding box of size [40 m×40 m]. We homogenize the patches using a voxel grid with 10 cm resolution, resulting in point clouds with around 300,000 points, see Figure 7.4 on the top left. We consider maps for registration that are within 10 m horizontal range for a sufficient amount of overlap but still have a large non-overlapping area. This can practically be accomplished even with cheap GPS sensors or with place recognition as described in the previous Chapter 6. The initial rotations differ up to 180° due to different driving directions, therefore making it pretty challenging for local registration methods.

We compare our approach with respect to two geometric approaches: RANSAC-based coarse registration with finetuning using GICP [201] as well as Teaser [275]. For the comparison with learning-based baselines, we retrained PCRNet [196] and HRegNet [138] on the Apollo data. We exchanged the loss function of the PCRNet from the earth mover distance to our loss function in Equation (7.12), which led to better performance for this data.

For the quantitative evaluation, we will evaluate the approaches based on the mean absolute error MAE(·) in terms of the rotation MAE(R) and translation

MAE($\boldsymbol{t}$), as follows

$$\mathrm{MAE}(R) = \frac{1}{|I|} \sum_{i \in I} \arccos\left(\frac{\mathrm{trace}(R_{gt(i)}^{\top} R_{est(i)}) - 1}{2}\right), \tag{7.13}$$

$$\mathrm{MAE}(\boldsymbol{t}) = \frac{1}{|I|} \sum_{i \in I} \|\boldsymbol{t}_{gt(i)} - \boldsymbol{t}_{est(i)}\|_2. \tag{7.14}$$

To measure the success rate of the registration, we always provide additionally for each metric the recall rate of how often it is below a certain threshold, e.g., MAE(R)@5°. We compute the mean absolute errors only for the inlier I, which are below the threshold.

7.3.1 Implementation Details

For our network, we utilize our compression encoder from Chapter 5 which achieves a compression ratio of around 1:100. For the feature enhancement network in Section 7.2.2.2, we use $B_k = 7$ KPConv blocks with radius $r = 2\,\mathrm{m}$. We use the transformer blocks with pre-layer normalization [270] for faster convergence and more stable training. In our experience, we saw that more than 2 transformer layers require substantially more training time and do not improve the performance significantly. The final feature dimension of the PointNet in Section 7.2.2.1, the ResNet blocks in Section 7.2.2.2, and the transformer in Section 7.2.2.3 is set to 256. We use Layer normalization [6] in all blocks due to the relatively small batch size of 8. Further, we use gradient accumulation over 4 batches and a learning rate of $5 \cdot 10^{-5}$ with the AdamW optimizer [134].

During developement, we have seen our feature-based registration method aligns the point clouds generally very well, but can sometimes still be slightly misaligned. To fine register, we use standard GICP [201] with a Geman-McClure kernel [65] on the compressed point clouds. The normals for the GICP fine registration are computed based on the 25 nearest neighbors. Registering a compressed point cloud requires around 0.04 s for feature extraction and pose estimation, as well as 0.02 s for the fine registration, i.e., in sum 60 ms for our approach. The compression of a dense point cloud of around 300 thousand points requires an additional 0.173 s but can be done in a preprocessing step and might be needed anyway for operation or storage. All experiments and the runtime are evaluated on an i7 @ 3.50 GHz with 8 cores and a GeForce RTX 2080 SUPER with 8 GB GPU memory.

7.3.2 Compressed Point Cloud Registration

This experiment investigates the registration quality of the aforementioned approaches when only the compressed representation is available, for example, due to memory constraints on the vehicle. For a fair comparison with the baselines, we will either provide them with the compressed (Figure 7.4b) or the decompressed point clouds (Figure 7.4c), whatever works best for them. We will evaluate our approach directly operating on the compressed point clouds. We want to note that for our approach, we also do the fine registration with the compressed point clouds (see Figure 7.4b) and *not* with the denser decompressed ones. In Table 7.1 the results for our approach and the baselines are shown. Our approach is the only one that is able to consistently (over 95% of the time) estimate the transformation within 1° and 0.3 m. The decompression of the point clouds leads to artifacts or additional noise, which seem to deteriorate the performance of the baselines. Our approach computes the initial guess in both cases on the compressed representation, and therefore the recall does not drop significantly when compared to the registration without memory constraints (compare Table 7.2). The mean absolute errors on the other hand drop by a factor of 2 to 3 when doing the fine registration on the compressed representation. The very low point resolution (around 1.5 m) does not allow for accurate normal estimations, limiting the performance of the GICP in the fine registration.

7.3.3 Comparison to General Point Cloud Registration Techniques

In this experiment, we evaluate the accuracy of the point cloud registration without aiming at compression. All methods have access to the original input data (see Figure 7.4a). Note that our approach still uses the compression network for feature extraction and subsampling, but uses the original input data (as in Figure 7.4a) for more precise fine registration. The results are depicted in Table 7.2. Our approach is able to outperform both, the classical, as well as the other learning-based approaches. The RANSAC-based approach provides quite similar results to ours. In Figure 7.5 we visualize some registration results of our approach. Our approach shown in the middle is able to recover from the bad initial guess depicted on the left. The fine-registration shown on the right allows fixing the small remaining pose error.

Table 7.1: Compressed registration

Approach	MAE(R) @0.5°	MAE(R) @1.0°	MAE(R) @5.0°	MAE(t) @0.1m	MAE(t) @0.3m	MAE(t) @0.5m
Teaser [275]	0.317	0.545	1.194	0.071	0.179	0.239
	(19.90%)	(45.04%)	(79.84%)	(6.63%)	(46.23%)	(65.03%)
PCRNet [196]	0.353	0.672	2.117	0.073	0.200	0.318
	(2.91%)	(13.32%)	(64.88%)	(0.87%)	(9.24%)	(21.98%)
HRegNet [138]	0.291	0.390	0.557	0.067	0.115	0.126
	(56.99%)	(77.31%)	(86.55%)	(37.87%)	(79.89%)	(83.66%)
RANSAC + GICP	0.209	0.299	0.691	**0.062**	0.123	0.159
	(51.69%)	(63.44%)	(78.39%)	(26.50%)	(57.43%)	(66.58%)
Ours	**0.178**	**0.207**	**0.216**	0.064	**0.109**	**0.113**
	(91.95%)	**(98.01%)**	**(98.55%)**	**(49.45%)**	**(96.61%)**	**(98.26%)**

Quantitative Results on the Apollo-Southbay dataset using the compressed point clouds as input to all methods. Presented numbers are the mean absolute errors of the translation in meters and the rotation in degrees, as well as the success rate in brackets.

Table 7.2: Classical registration

Approach	MAE(R) @0.5°	MAE(R) @1.0°	MAE(R) @5.0°	MAE(t) @0.1m	MAE(t) @0.3m	MAE(t) @0.5m
Teaser [275]	0.254	0.369	0.577	0.060	0.122	0.141
	(62.02%)	(84.12%)	(97.80%)	(39.21%)	(87.30%)	(94.61%)
PCRNet [196]	0.366	0.673	2.071	0.072	0.198	0.310
	(3.25%)	(14.57%)	(66.17%)	(1.18%)	(11.96%)	(26.62%)
HRegNet [138]	0.243	0.296	0.402	0.060	0.081	0.085
	(74.90%)	(85.19%)	(90.86%)	(66.30%)	(88.07%)	(89.37%)
RANSAC + GICP	0.066	0.067	0.088	0.048	0.056	0.057
	(97.52%)	(97.64%)	(98.43%)	(88.28%)	(97.46%)	(97.82%)
Ours	**0.045**	**0.045**	**0.059**	**0.047**	**0.048**	**0.048**
	(98.95%)	**(98.95%)**	**(99.35%)**	**(96.86%)**	**(98.69%)**	**(98.69%)**

Quantitative Results on the Apollo-Southbay dataset with the regular, not compressed point clouds, as input to all methods. Remark that our approach is still compressing for the feature computation. Presented numbers are the mean absolute errors of the translation in meters and the rotation in degrees, as well as the success rate in brackets.

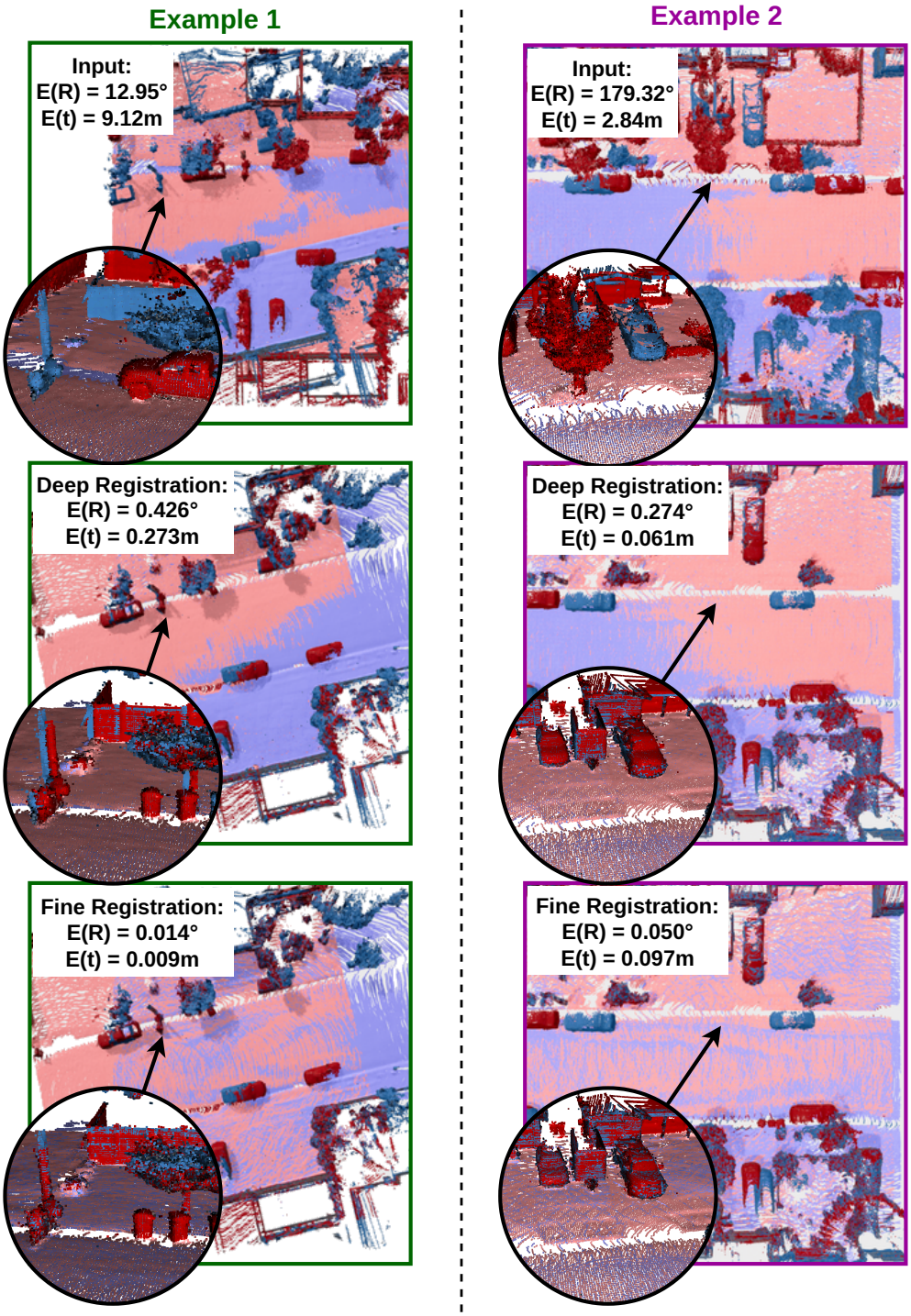


Figure 7.5: Qualitative results of our point cloud registration method on two example scenes. In blue, the target point cloud is shown, while in red the source is transformed by either the initial guess (top), the registration solely based on our network (middle), and after the fine registration (bottom). Even under extreme conditions like 180° of wrong rotation, our network is able to align the point clouds properly. The fine registration is there to fix the last slight misalignment. $E(R)$ and $E(t)$ denotes the error in rotation and translation, respectively.

Table 7.3: Ablation: Architecture

	KPConv	Transf.	GICP	MAE(R) @5.0°	MAE(t) @0.5m
[A]	✗	✗	✓	0.503 (48.89%)	0.120 (13.33%)
[B]	✗	✗	✗	2.990 (40.13%)	0.304 (1.70%)
[C]	✓	✗	✗	0.425 (98.69%)	0.101 (94.38%)
[D]	✗	✓	✗	1.285 (89.80%)	0.252 (54.51%)
[E]	✗	✓	✓	0.253 (94.64%)	0.121 (92.16%)
[F]	✓	✓	✗	0.405 (99.74%)	0.099 (96.21%)
[G]	✓	✓	✓	0.187 (99.87%)	0.115 (99.61%)

Mean absolute error for different network architecture configurations for translation in meters and rotation in degrees, as well as the success rate in brackets.

7.3.4 Ablation on the Network Architecture

In this section, we will take a look at different parts of the network to provide deeper insights into the approach. All the following results are evaluated on the validation set.

In Table 7.3 we investigate the performance of the architectural choices by enabling and disabling certain parts of the network. We can see that only using the compressed features [B] is not sufficient to reliably estimate the transformations, leading to the worst performance. Increasing the receptive field by using the proposed feature enhancement increases the performance drastically [C]. An additional transformer head can slightly improve the performance [F], while only using the transformer degrades the performance significantly [D]. These results are in line with results from different domains, showing that the locally inductive bias of convolutions speeds up the training and is especially helpful when having smaller-sized datasets [269]. The best results can be achieved when enabling all parts of the network and finetuning the results using GICP [G]. For completeness, we show the results of only using the GICP without an initial guess from our network [A]. ICP cannot deal well with large transformations, which therefore shows the need for a good initial guess.

7.3.5 Ablation on the Weighting Schemes

In this ablation study, we will investigate the performance of the different weighting schemes introduced inSection 7.2.3. Additionally, we compare against no adaptive weighting, which results in a constant weight ($\boldsymbol{w} = \mathbf{1}_N$) as baseline.

Table 7.4: Ablation: Weighting schemes

Metric	Constant	Entropy	Maximum	MLP
MAE(R) @5.0°	1.648 (89.28%)	0.725 (97.78%)	0.620 (98.04%)	0.405 (99.74%)
MAE(t) @0.5m	0.304 (10.85%)	0.186 (82.22%)	0.156 (86.93%)	0.099 (96.21%)

Mean absolute error for different weighting of the correspondences for translation in meters and rotation in degrees, as well as the success rate in brackets.

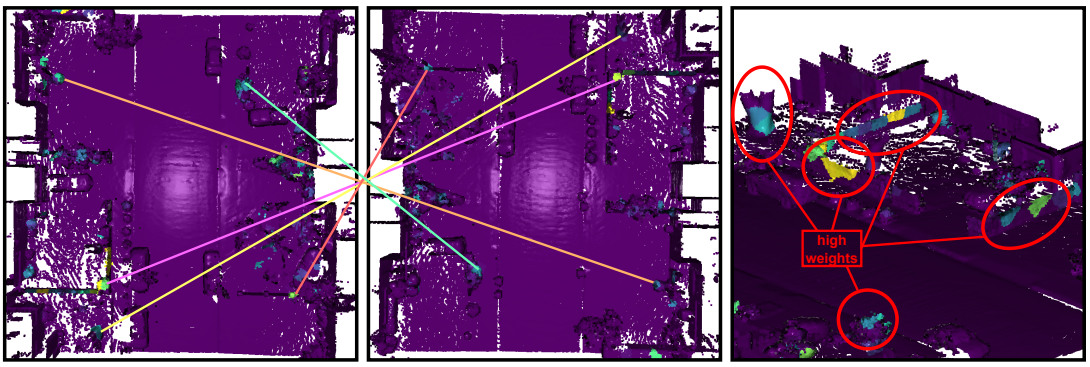


Figure 7.6: Point clouds (target: left, source: middle) are visualized by the estimated weights $\boldsymbol{w}$ to show which points contribute to the estimated transformation. The brighter the color the higher the impact. On the right is a close-up view of the target. Our network only considers static, structured areas for registration. Weights are upsampled on the input point cloud for better visualization.

The results are depicted in Table 7.4. The MLP-based method shows superior performance over the max-pooling and entropy-based versions, which both perform quite similarly. Using for every correspondence a constant weight performs substantially worse (with a 80 percent-point drop in the translation), showing the importance of computing individual weights. In Figure 7.6, we additionally visualize the weights for the correspondences to illustrate, which points the network uses for the registration. For each target point, we have exactly one correspondence weight and therefore can directly colorize the points based on the weight magnitude. For the source, we compute the mean activation in W weighted by the correspondence weight $\boldsymbol{w}$ to colorize the points. Bright colors indicate a high weight and dark colors indicate a low weight. For better visualization, we show the weights on the input point cloud based on its nearest neighbor in the compressed point cloud. Big areas with a low structure like ground, and huge walls have a very low weight and therefore are not considered for the registration. Even though cars provide a lot of structure, they also have a low weight. The

network did maybe learn that cars often move and therefore are not reliable for the pose estimation. Only a few distinct and stable areas like trunks, small walls, or poles have a high weight and therefore are used for the registration. Since our approach relies on point-to-point correspondences, these results are in line with our expectations.

7.4 Conclusion

In this chapter, we presented a novel architecture for point cloud registration under arbitrary initial estimates. We exploited the point-feature-based representation from our former defined point cloud compression approach. This allows us to directly estimate the transformation on the compressed point clouds without the need for decompression. We used the representation as a basis for our feature enhancement network to estimate features that are well-suited for matching. The network consists of a stem architecture for increasing the receptive field, as well as a Transformer network for propagating information between the point clouds. We have reformulated the scaled dot product attention from the Transformer architecture to estimate point correspondences. Having the set of correspondences allows use to utilize standard closed form solutions for estimating the desired transformation that aligns both point clouds. Our proposed method is completely differentiable, such that we can directly optimize the whole network end-to-end to find the best parameters for the pose estimation. The network needs to learn during training to generate features that lead to correct feature-based matching of the points to estimate a correct pose. By this, we can directly optimize the network to learn correct matches, without directly supervising the matching, which would require ground truth correspondences. Due to the presence of dynamic objects, non-overlapping areas and changed scenes because of the time difference in recording and mapping, we have to estimate for each correspondence a weight. Therefore, our network also estimates a confidence weight for each correspondence such that the network can focus on the unambiguous parts which are well suited for matching. Our experiments have shown, that the weighting is necessary for reliable registration. The proposed method can register compressed point clouds over 95% of the time below 30 cm and 1°. Additionally, we have shown that our method can also be used for regular point cloud registration, when not targeting compression. There, we achieve over 95% of the time registration errors even below 10 cm and 0.5°. By this, our method is able to outperform the baselines. Working solely on compressed representations, which is two orders of magnitudes smaller than the raw point cloud data has the potential to scale mapping-based systems substantially, without compromising the systems' performance.

Even though our method showed very promising results, the main drawback of this method is the runtime. The feature computation is quite time-intensive even though the compressed features are already reused. Future work could investigate how to speed up the method without compromising too much on the registration quality. One possible direction would be to look into knowledge distillation networks, where a teacher network supervises a smaller light-weight student network. In the end, the significantly more compute-efficient student network can often perform relatively well with substantially fewer resources. For now, we exploited aggregated point clouds which have the advantage that they cover bigger areas, are denser and potentially see objects from multiple sides.

With this work, we completed the full localization pipeline leveraging a memory-efficient representation. From the initial system calibration, over the map construction, to the coarse localization using place recognition, and finishing now with the fine localization using place recognition. While in the coarse localization, we have found in which part of the map our robot is located, here we estimated the full 6 DoF transformation of the robot to the map.

The network architecture is quite compute-intensive and takes a couple of seconds to compute the features. Additionally, we operate on aggregated point clouds to obtain a denser and more complete representation. Therefore, this method is not suitable to run at the sensor frame rate. That is not really problematic since often systems have not only the global relocalizer but also an online odometry estimation to cover the time between two global estimates. However, it might be advantageous to track the position of the robot directly in the map once the initial position is computed. For the initial localization we require global registration that can deal with arbitrary initial position estimates, but once the position is found we could exploit the movement of the car. Estimating the motion of the car, even with a simple model, can improve our initial position estimate to a centimeter level. Thus, from then on we could rely on basic ICP-based techniques to track the movement of the robot to the map, without the need for compute-intensive feature estimation. In the next chapter, we want to investigate how can we track our robots position within a map, when already having a quite good initial pose estimate, as could be obtained by our coarse to fine localization. Instead of relying on our compressed feature-based representation, we will explore the neural implicit representations for mapping the environment. As done so far, we will focus on memory efficiency to enable large-scale mapping and efficient data storage.

Chapter 8

Pose Tracking in Neural Distance Field Maps

The ability to track the position of a robot within a given map is important for localization, planning, and state estimation. When moving through an already mapped area, the map can be used to estimate the ego motion of the robot. For this, one can align the observations of the robot's surroundings to the map. The transformation needed for the alignment can then be used to estimate the robot's position within the given map. Modern robots often perceive the environment at high frequency, which enables the tracking of the robot's pose in the map by successive alignment. This so-called map matching has a big advantage over relative pose estimates, e.g., from inertial navigation, or odometry estimation, by having a global reference. Relative estimates typically accumulate errors which leads to a drift in position over time.

In this chapter, we want to investigate how to track the pose of a robot in a given map. For clarification, we refer to pose tracking as the task of estimating the movement of a robot or sensor with respect to a map using sequential sensor observations. By this, it is a special form of localization, where we assume an initial position given and from there on iteratively estimating the robot's pose. In the last two chapters, we have localized a robot based on a local dense point cloud of the robot's proximity in our compressed map representation. We first computed in Chapter 6 in which part of our map the robot is located, while in Chapter 7 we estimated the full 6 DoF pose. Starting from such an initial pose estimate, we now want to track the movement of the robot in a map iteratively using the current LiDAR scans. By operating on a scan level, but still exploiting sequential data, we can estimate the motion without inducing latency for point cloud aggregation as needed for our global registration methods. Therefore, our proposed global registration methods can find the initial position in the map, and

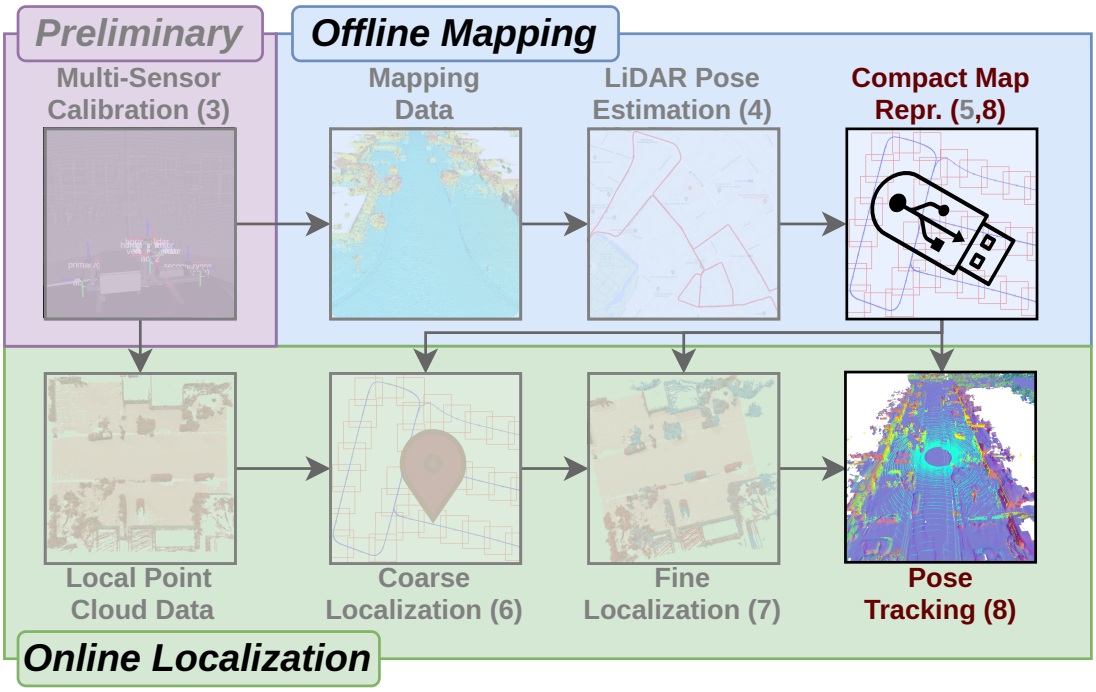


Figure 8.1: With the last chapter, we have localized our robot precisely in a given compressed map. In this chapter, starting from a known pose, we want to now track the movement of the robot within the map by successively aligning local scans to a neural distance field.

once the position is found, pose tracking can take over to estimate the motion from then onwards.

Common map representations in mobile robotics are occupancy maps [89], surfels [13, 219], distance fields [72, 188], NDTs [218], or raw point clouds [243, 260]. These representations are often combined with data structures for efficient management: octrees, kD-trees, grid maps, or hash maps. In this work, we will take a closer look at representing the scene with a distance field. The advantage of this representation for mobile robotics is that the distance it provides can be used for common robotic tasks like scan registration, Monte Carlo localization (MCL), path planning, and even reconstruction.

In line with the previous chapters, one key focus is to have a compact representation of the map to enable localization in city-scale environments. However, instead of again relying on our compressed point-feature-based map from Chapter 5, we will explore the usage of a neural implicit map representation for this task. Neural implicit representations do not explicitly model the environment, e.g., through points, triangles, or primitives, but rather implicitly as a function of space. For this neural networks are learned such that they can predict for every point in space certain attributes like density, color, or the distance to the closest surface. Recent works in computer vision train neural networks to learn

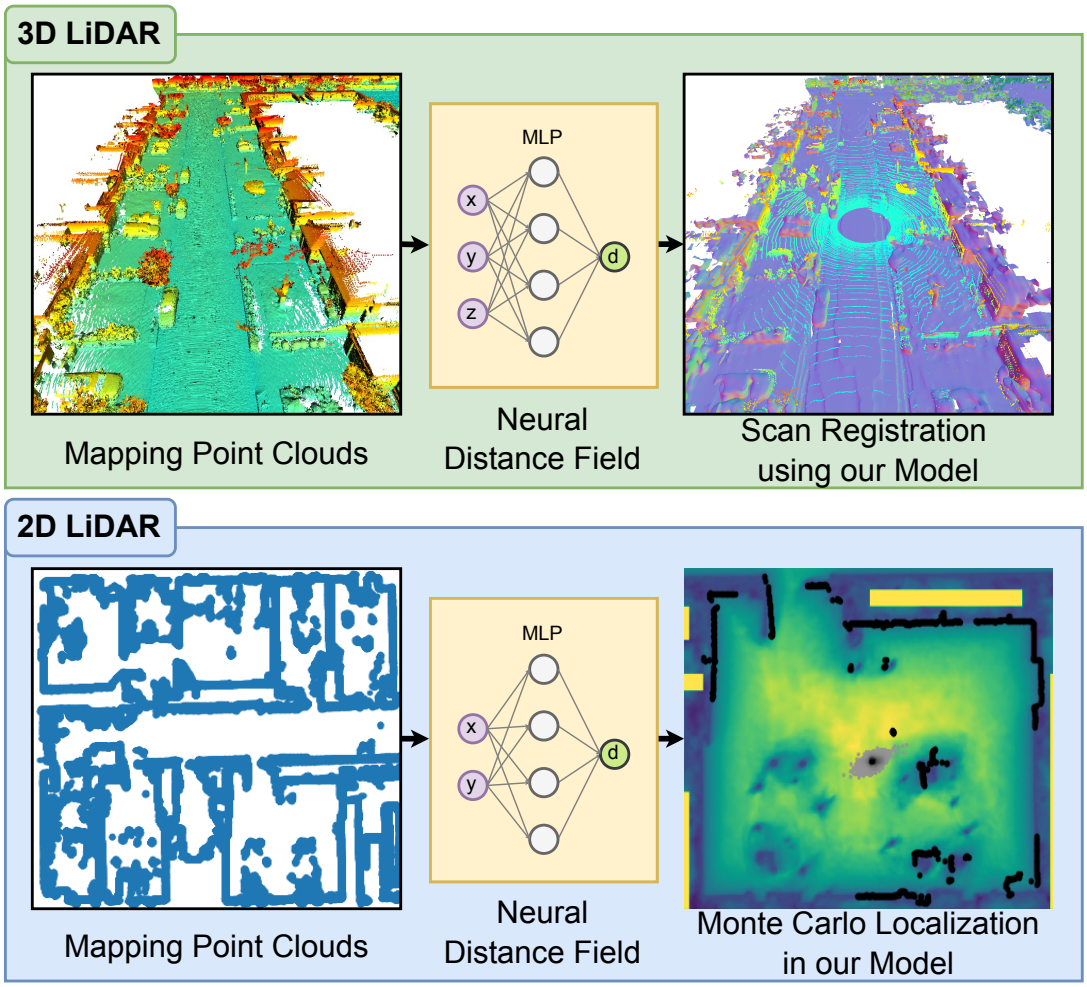


Figure 8.2: In this chapter, we will show how to learn a neural distance field from point cloud data. Using this neural distance field as a map representation enables scan registration and localization.

the distance field for a given scene. This so-called neural distance field (NDF) is usually modeled by a simple multi-layer perceptron, which returns for each point in space the distance to the closest surface. The advantage over the common grid-based distance fields is the continuity of the prediction, therefore, it is not limited by grid resolution. Additionally, a network has the ability to represent low-dimensional information efficiently, whereas grid-based representations spend a lot of grid cells to store low-dimensional objects like walls or a ground plane. However, neural implicit representations learn a function that describes the scene instead. Since, the network only has a certain amount of parameters and DoF, it needs to use them efficiently. In computer vision, the training of the NDFs is usually done by either using high-resolution ground truth meshes or normal information or is directly supervised by a given distance field. So far, it has been rarely investigated how the NDF can be trained from raw sensor data, such as

raw LiDAR data. Additionally, it is not clear how well those NDFs are suited to tackle mobile robotic tasks.

In this chapter, we propose our approach for learning NDFs based on point cloud data as commonly obtained from real LiDAR sensors. Additionally, we show how to localize in a given NDF, as visualized in Figure 8.2. In the domain of autonomous driving, we look at pose tracking for odometry estimation in a given map. As an additional experiment, we also look at indoor navigation where we use MCL in combination with NDFs. In sum, we propose a method to learn a neural distance field, which is well suited for scan registration and allows for localization using MCL.

8.1 Related Work

8.1.1 Map Representations

Many different map representations have been used in the robotic context. The point cloud representation is quite common since it can represent directly the sensor observations. Naturally, a point cloud is an unordered list of coordinates, which makes operations like finding neighbors, or computing distances to the map computationally demanding. Therefore, point clouds are often combined with acceleration structures like octrees [15, 150], hash maps [163, 243], or kD-trees [19] to speed up such operations.

Although point clouds can be dense by having many points, the representation can not produce close surfaces. Surfels [13, 219] approximate the underlying surface by planar patches, thereby approximating surfaces usually better. These patches are usually treated independently of each other which allows for easy deformation, as it is needed for correcting the map, e.g., after a loop closure in SLAM [13]. However, surfels usually do not form a watertight surface as often present in the world. Triangle meshes, on the other hand, can be constructed in a watertight fashion and can smooth out noise effectively, e.g., using Poisson surface reconstruction [109, 240].

In the field of robotics, so-called 2.5D representations are often used for path planning or localization. Elevation maps [9, 121, 177] store the height of each cell in a 2D grid. To overcome the limitation of representing only a single height per cell, multi-level surface maps [231] extend the idea by storing multiple levels.

Occupancy fields have the notion of free and occupied space and can even be extended to represent unknown space in a probabilistic fashion [89]. These occupancy fields are often stored in image-like grids for 2D [75] and classically in octrees for 3D [89]. Neural radiance field (NeRF) [10, 155] methods use most commonly an MLP to learn the occupancy of a scene. While the original

work [155] learns the underlying representation from image data, recent works utilized LiDAR data either to complement the training [292] or directly to replace the images [224]. Neural occupancy fields learn for each position in space to predict the respective occupancy value [154, 273].

Instead of storing the occupancy of a scene, another common representation is to store for each position the distance to the closest surface [40]. A Euclidean distance transform can be used to transform an occupancy grid [188] into a distance field. Traditionally, the distance values are stored in a grid. Nowadays, more learning-based approaches emerge, which use neural networks to learn the so-called NDF. Where instead of storing the distances into a grid, a network shall learn for each position in space the distance to its closest surface. Such NDFs are usually supervised either by density fields [236], normals [207, 264], or directly the ground-truth distance [37, 175]. Learning them directly from sensor observations is just rarely exploited [5, 297]. Azinovicć et al. [5] learn NDFs from RGB-D camera data. The availability of color information is used in the training such that the rendered images look like the observed images. While their approach is refining the poses of the training data, our approach is focusing on estimating the pose from data that is not used for the training. Their poses are directly refined in the training using ADAM optimization. We instead, combine classical pose estimation methods with the usage of NDFs. Azinovicć et al. [5] as also we are representing a scene by one MLP for a compact representation. However, some approaches learn local MLPs or embedding vectors that divide the space in smaller chunks [184, 297]. This has the advantage that the individual MLPs or embeddings are significantly smaller and thus faster to train. The cost for it is that you require more MLPs and due to the locality, the network can not exploit low-frequency information in the data. Similarly to SHINE-Mapping [297], we are trying to learn NDFs directly from LiDAR data. However, the goals are different; our approach aims at having a compact representation for localization, while they focus on high fidelity reconstruction. Besides the architectural differences, we propose to use the neural distance field for correcting the observed distances while training.

8.1.2 Scan Registration

Scan registration is a common problem in robotics, that we encountered already in Chapter 4 and Chapter 7. For a more extensive review of the field, we refer to the respective parts, i.e., Section 4.1 and Section 7.1. Here, we only point out the major works to set our work into perspective.

The most common method for aligning two point clouds is the ICP algorithm [21]. It usually, consists of two steps: first finding correspondences between the two point clouds, which can be then used to estimate the pose by optimiz-

ing an error metric. For finding correspondences, one can search for the closest point [21], use projective data associations [13, 161], or do feature-based matching [78, 257] (as we have also done in Chapter 7). Point-to-point [21], point-to-plane [36], or even plane-to-plane [201, 202] metrics are minimized for computing the alignment. Our proposed point-to-NDF-based registration is conceptually quite similar to classical registration to a distance field [161]. The potential over the grid-based representation is that the neural-based representation is free of discretization and does not require truncating the distance fields due to higher memory efficiency.

Robust kernels and correspondence thresholds are used to reduce the impact of outliers, low overlap, and dynamic objects [30, 65, 97].

8.1.3 Monte Carlo Localization

MCL is a method to localize a mobile robot in a given map using a particle filter [43, 60]. Its key idea is to represent the posterior belief about the robot's pose by a set of weighted samples, so-called particles. Each particle weight represents the likelihood of the corresponding pose hypothesis, and can be computed by the sensor observations. The basic idea is to compare the current sensor reading at the particle location with the map, using the so-called observation model [229]. In the context of 2D global localization, the map is often represented as an occupancy grid [43] which can be efficiently constructed from LiDAR data. Coverage maps [212] extend the idea and not only distinguish between free and occupied space but store a posterior of the coverage instead. Different methods address the problem of changing or dynamic environments [68, 213, 300]. One way is by not relying only on one version of the map, but storing patches of the map and incorporating a map transition model [213]. Nowadays semantic cues in the form of text [302], or fully semantic masks [68, 300, 301] are used to focus on the static and stable parts. Precise plans from building information modeling (BIM) are sometimes used for localization to exploit existing data [87, 282]. As demonstrated by Boniardi et al. [23], or Zimmermann et al. [300], MCL does not always require accurate maps, but can even work in imperfect floor plans.

The performance of MCL-based systems heavily relies on the number of particles used. The more particles, the higher the chance to converge to the right position, but also the slower the method. Often methods have an initialization phase where they use substantially more particles for global localization, and a tracking phase with fewer particles once the system is converged [118]. Fox [61] proposes a method for adaptively regulating the number of particles based on their distribution. The most common observation model for LiDAR based localization is the classical beam-end model [18, 43, 118, 213]. With the rise of neural

networks, a semantic observation model can complement the geometric beam-end model [68, 300] for faster convergence.

Besides the potential of using neural networks for the observation model, learning based map representation have recently shown advantages over classical maps. Standard grid-based maps are bound by the resolution of the underlying grids, while implicit neural representations can estimate a continuous field. Kuang et al. [118] therefore propose a neural occupancy field to overcome the discretization limitation to improve localization. However, their approach requires computationally demanding ray casting-based rendering to evaluate the observation model and thus has to reduce the usable number of beams per scan for online localization. Conversely, using our NDF, we predict the distances to the closest surface directly for using it in the observation model. By this, we do not require compute-heavy rendering. In the image domain, NeRFs have been used to localize [144] by rendering images at the particles' position from the neural radiance field (NeRF) and compared to observed images to estimate the particles' likelihood.

One major problem for particle filter-based methods is the scalability. Scaling from a 2D map with typically 3 state parameters (2 for translation and 1 for rotation) to 3D with 6 state parameters requires exponentially more particles to have comparable particle coverage, also known as the curse of dimensionality. Maps in the domain of autonomous driving cover usually large areas, that also increases the amount of particles with respect to the classically smaller indoor areas. To localize in a 3D large-scale map, methods estimate with the particle filter only a 2D pose by using either point cloud descriptors, or overlap-based observation models [33, 280, 283]. By this, they assume a mostly planar movement and only little rotation in roll and pitch, which usually holds in the automotive domain. A final 6 DoF pose can later be estimated using classical scan registration methods.

8.2 Learning Neural Distance Fields for Robot Localization

In this chapter, we aim at learning a neural distance field from point cloud data, acquired by sensors, like LiDAR sensors, or RGB-D cameras as a representation to explicitly support localization. We do not rely on point cloud normals, since normal estimation heavily relies on the type of sensor and is prone to errors. The above-mentioned range sensors have in common that they measure the distance from the sensor origin to the surface. The only assumption we make is that the space between the sensor and the measured point on the surface is free space.

In the following, we will first explain how we train the NDF from the sensor data. Second, we show two common localization methods for point cloud data within the NDF, namely scan registration using ICP and global localization using a particle filter.

8.2.1 Learning Neural Distance Fields from Sensor Data

Our goal is to be able to query the neural distance field D at an arbitrary point in space $\boldsymbol{p} \in \mathbb{R}^{\mathcal{D}}$ to obtain the distance d to the closest surface. In this work, we focus on applications in outdoor robotics with $\mathcal{D} = 3$, which covers localization and registration using 3D point clouds produced by commonly employed automotive LiDAR sensors or terrestrial laser scanners, but also indoor environments, where we often use $\mathcal{D} = 2$, for commonly equipped 2D LiDARs for localization and navigation.

The representation of our map is a multi-layer perceptron $D : \mathbb{R}^{\mathcal{D}} \mapsto \mathbb{R}$ which maps the input coordinates to the Euclidean distance space. We use a positional encoding $\pi : \mathbb{R}^{\mathcal{D}} \mapsto \mathbb{R}^{2I_\omega}$ with periodic activation functions,

$$\pi(\boldsymbol{p}) = (\boldsymbol{p}, \sin(\omega_1 \boldsymbol{p}), \cos(\omega_1 \boldsymbol{p}), \ldots, \sin(\omega_{I_\omega} \boldsymbol{p}), \cos(\omega_{I_\omega} \boldsymbol{p})), \tag{8.1}$$

where ω_i is the i^{th} frequency. The positional encoding helps to retain high-frequency information in the distance field [155].

In contrast to previous approaches which supervised the training process either by ground truth distances [37, 175], occupancy fields [236], or given normals [207, 264], we exploit the measurement process of the LiDAR sensors similar to truncated signed distance field (TSDF) fusion pipelines [161, 242]. However, we are not directly supervising using the TSDF values like Zhong et al. [297], but rather use an approximated distance as explained in the following.

Laser sensors measure the distance from the sensor origin to the surface, which we will call the ray distance d_p. Inspired by NeRFs, we sample points $\{\boldsymbol{p}_i \in \mathbb{R}^{\mathcal{D}} \mid i = 1, ..., N\}$ along the LiDAR beam, i.e., $\boldsymbol{p}_i = (1 - \lambda_i)\boldsymbol{o}_i + \lambda_i \boldsymbol{e}_i$, between the sensor origin $\boldsymbol{o}_i \in \mathbb{R}^{\mathcal{D}}$ and the end of the beam $\boldsymbol{e}_i \in \mathbb{R}^{\mathcal{D}}$. We sample more points near the surface by sampling log-linearly along the ray, i.e.,

$$\lambda_i = \frac{1 - 10^{\frac{i}{N-1}-1}}{0.9}. \tag{8.2}$$

Consequently, the ray distance d_i for each sampled point to the surface is given by $d_i = \|\boldsymbol{e}_i - \boldsymbol{p}_i\|$. Note that the ray distance d_i does not necessarily correspond to the distance to the closest surface. Instead of searching for each sampled point for the closest point on a surface, which is computationally expensive for large-scale maps and requires determining first the surface by reconstruction, we can

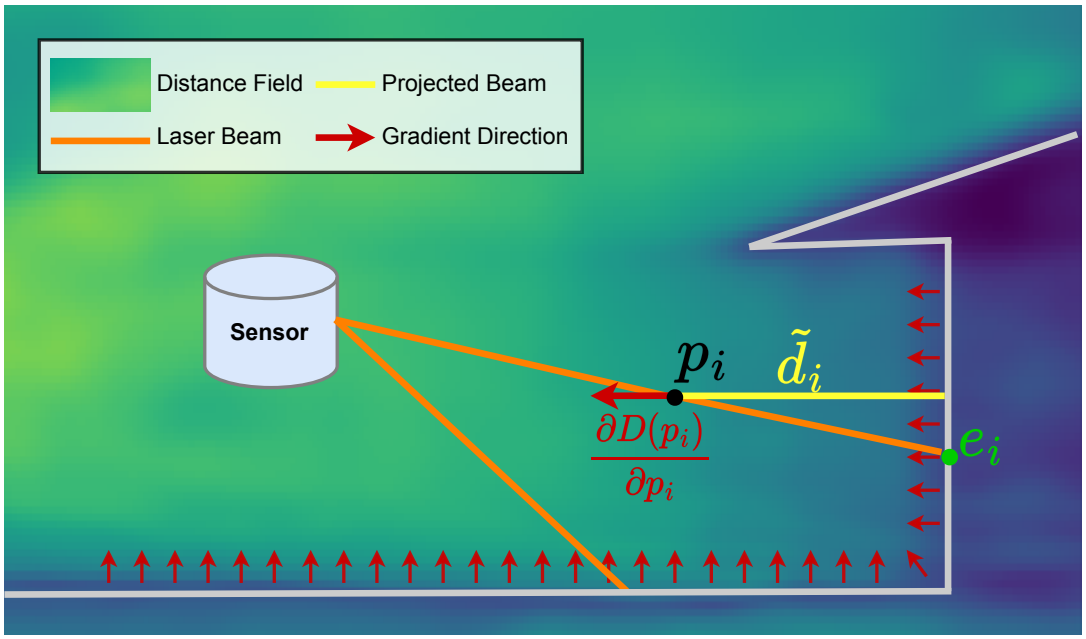


Figure 8.3: Instead of learning the distance between the point $\boldsymbol{p}_i$ and the measured point on the surface $\boldsymbol{e}_i$ (red), e.g., as for TSDF, we project the beam along the direction of the gradient (grey arrow) to supervise by the approximated distance $\tilde{d}_j$ to the closest surface. The color of the background corresponds to the distance field; the brighter the color, the higher the distance.

approximate the direction $\boldsymbol{n}_i$ to the closest surface analytically using the NDF D by using the gradient:

$$\boldsymbol{n}_i = -\frac{\partial D(\boldsymbol{p}_i)}{\partial \boldsymbol{p}_i}. \tag{8.3}$$

A visualization of this process is depicted in Figure 8.3. The gradient provides us with the direction of the steepest increase of the distance field, therefore the negative gradient points toward the closest surface. In practice, we can use automatic differentiation to compute $\boldsymbol{n}_i$. We project the distance d_i along the direction $\boldsymbol{n}_i$ to approximate the distance to the closest surface

$$\tilde{d}_i = \frac{(\boldsymbol{e}_i - \boldsymbol{p}_i)^\top \boldsymbol{n}_i}{\|\boldsymbol{n}_i\|}. \tag{8.4}$$

We use the approximated distance $\tilde{d}_i$ as supervision signal during training. Note that this is a circular problem: The better the approximated distance $\tilde{d}_i$, the better we can supervise the NDF. At the same time, the better the NDF, the better we can estimate the direction to the surface, which finally should result in a better-approximated distance. This circular dependency might raise the question if the training is stable, especially when we initialize the NDF D

with random weights. Practically, we did not notice any instabilities due to this approximation in the training. We reason that this might be due to the fact that the approximation error, $\epsilon_i = |\tilde{d}_i - d_i|$, is smaller, the closer the query point $\boldsymbol{p}_i$ is to the surface. Thus, for surface points or points close to the surface, i.e., $d_i \approx 0$, we approach $\epsilon_i \approx 0$ regardless of the gradient $\boldsymbol{n}_i$. Therefore, the correct distance propagates from the surface to the free space while training the NDF.

Similar to TSDF fusion pipelines [242], we prioritize measurements $\boldsymbol{e}_i$ with lower distance d_i. For this, we introduce a weight w_i given by

$$w_i = (d_{max} - d_i)^{\gamma}, \tag{8.5}$$

where d_{max} is the largest distance in a batch and γ is a hyperparameter that regulates the impact of measurements from higher distances, i.e., the higher γ, the lower the impact of far points. We supervise the NDF by minimizing the weighted L1 loss of our approximated distances

$$\mathcal{L}_{\text{dist}} = \sum_i \frac{w_i |D(\pi(\boldsymbol{p}_i)) - \tilde{d}_i|}{\sum_j w_j}. \tag{8.6}$$

Additionally, we have an additional loss to enforce that the endpoints lie on the surface

$$\mathcal{L}_{\text{end}} = \sum_i |D(\boldsymbol{e}_i)|. \tag{8.7}$$

Similar to Zhong et al. [297], we add a regularization loss to enforce the Eikonal equation $\|\nabla D\| = 1$, which needs to hold for being a valid distance field

$$\mathcal{L}_{\text{Eik}} = \sum_i |\|\boldsymbol{n}_i\|_2 - 1|, \tag{8.8}$$

as well as a loss to enforce that neighboring points have similar normals

$$\mathcal{L}_{\text{n}} = \sum_i |\angle(\boldsymbol{n}_i, \hat{\boldsymbol{n}}_i)|, \tag{8.9}$$

where $\angle(\cdot)$ is the cosine distance and $\hat{\boldsymbol{n}}_j$ is the gradient of the neighbor of $\boldsymbol{p}_j$. The neighbors $\boldsymbol{p}_j$ are sampled within a distance τ of $\boldsymbol{p}_i$, i.e., we randomly select from all radius neighbors $\{\boldsymbol{p} \mid ||\boldsymbol{p}_i - \boldsymbol{p}|| < \tau\}$ an arbitrary point $\boldsymbol{p}_j$.

The final loss is a linear combination of the aforementioned losses

$$\mathcal{L} = \mathcal{L}_{\text{dist}} + \alpha_{\text{end}}\mathcal{L}_{\text{end}} + \alpha_{\text{Eik}}\mathcal{L}_{\text{Eik}} + \alpha_{\text{n}}\mathcal{L}_{\text{n}}. \tag{8.10}$$

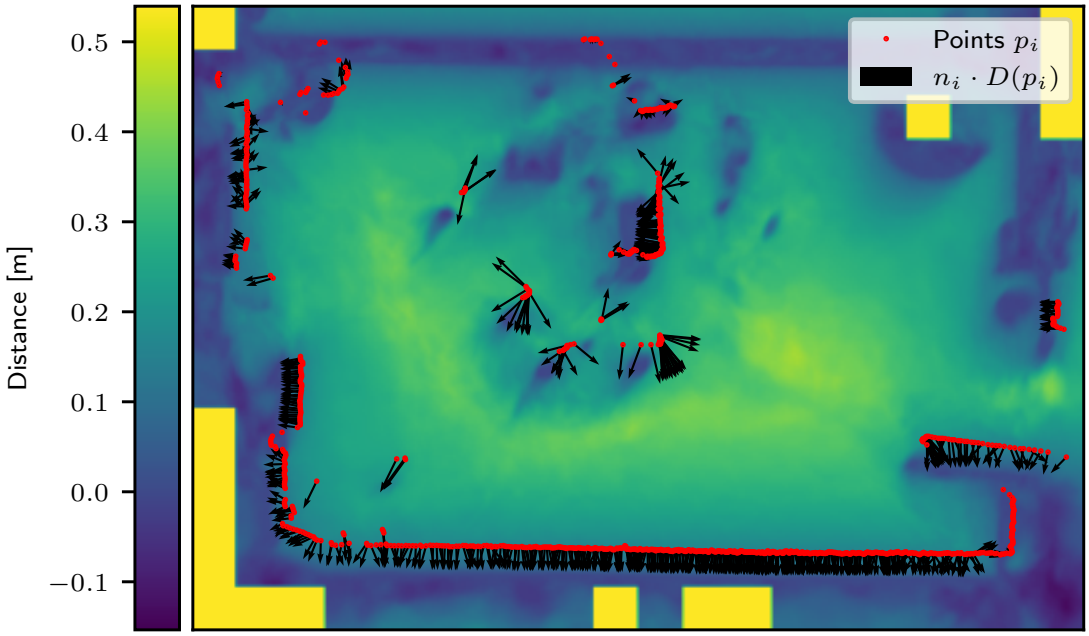


Figure 8.4: Principle of scan registration in an NDF. Querying the NDF at the positions of the input scan provides us with the distance and by differentiation also with the direction we have to go, to align the scan to the NDF. This procedure can be solved iteratively in an ICP fashion. The color of the background corresponds to the distance field; the brighter the color, the higher the distance.

8.2.2 Scan Registration to a Neural Distance Field Map

In this section, we show how to leverage the learned NDF to register point clouds to the map using ICP effectively. The objective is to find the rotation $R \in SO3$ and translation $\boldsymbol{t} \in \mathbb{R}^3$ that aligns the point cloud P to the NDF map D, i.e., that reduces the distance between the point cloud and the surface

$$R^*, \boldsymbol{t}^* = \underset{R,\boldsymbol{t}}{\operatorname{argmin}} \sum_{\boldsymbol{p}_i \in P} D(R\boldsymbol{p}_i + \boldsymbol{t})^2. \tag{8.11}$$

We solve the problem using non-linear least squares optimization, where the Jacobians for the i$^{\text{th}}$ point are

$$J_i = \left[\frac{\partial D(R\boldsymbol{p}_i + \boldsymbol{t})}{\partial \boldsymbol{t}}, \frac{\partial D(R\boldsymbol{p}_i + \boldsymbol{t})}{\partial \boldsymbol{\Theta}} \right] = \left[\boldsymbol{n}_i, \boldsymbol{p}_i \times \boldsymbol{n}_i \right], \tag{8.12}$$

where $\boldsymbol{\Theta}$ is the axis-angle parameterization of R.

As we can see in Equation (8.11) and Equation (8.12), we do not rely on corresponding points. This has the advantage of not needing to search for correspondences in contrast to classical ICP-based methods. We can solely solve the problem by knowing in which direction ($\boldsymbol{n}$) and how far (given by $D(\boldsymbol{p})$) we have to go. This information is directly encoded in the weights of the network and

learned in the generation of the map. A visualization of this for the 2D case can be seen in Figure 8.4.

We know from theory [77] that the distance field needs to fulfill the Eikonal equation $\|\nabla D\| = 1$, or intuitively: if we move one meter away from the surface, the distance needs to increase by one meter. Since we only approximate the NDF by a neural network, this does not necessarily hold. If the norm of the gradient is either larger or smaller, we would over or underestimate the distance accordingly. To counter this phenomenon, we normalize the distance by the norm of the gradient $D(\boldsymbol{p}_i)/\|\boldsymbol{n}_i\|$. If for example, the gradient would be $\|\nabla D\| > 1$, one would overestimate the distance and overshoot, ending up behind and not at the surface. Therefore, by shortening the step to only move $D(\boldsymbol{p}_i)/\|\boldsymbol{n}_i\|$ towards the surface, one would end up on, or at least closer to the surface, which enabled us a more reliable registration.

8.2.3 MCL-based Localization in a Neural Distance Field-based Map Representation

In this section, we explain how to globally localize within an NDF using Monte Carlo localization. The belief $bel(\boldsymbol{x}_t)$ about the robots position $\boldsymbol{x}_t$, at time t is represented by as set of particles $\{(\boldsymbol{x}_t^i, w_t^i) \mid i = 1...I\}$ each with a corresponding weight w_t^i. A motion model $p(\boldsymbol{x}_t) \sim p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_t)$ updates the particles based on their previous position $\boldsymbol{x}_{t-1}$ and the control commands $\boldsymbol{u}_t$. The weight of the particles is updated by the observation model $w_i^t \propto p(\boldsymbol{z}_t \mid \boldsymbol{x}_t, D)$ which depends on the observations $\boldsymbol{z}_t$, the pose $\boldsymbol{x}_t$, and the NDF D. Assuming we observe the local surrounding with a LiDAR sensor, we can evaluate the distance field at the observed point cloud $\{{}^j\boldsymbol{z}_t^i \mid j = 1, ..., J\}$ around the particle position ${}^j\boldsymbol{x}_t^i$ with the classical beam-end model

$$ {}^j d_t^i = D\big(\boldsymbol{R}_t^i \, {}^j\boldsymbol{z}_t^i + \boldsymbol{t}_t^i\big). \tag{8.13} $$

The terms $\boldsymbol{R}_t^i$ and $\boldsymbol{t}_t^i$ are the rotation matrix and translation of the particle's position $\boldsymbol{x}_t^i$ respectively. Assuming a Gaussian noise model, we can compute the weight for a particle by

$$ p(\boldsymbol{z}_t \mid \boldsymbol{x}_t^i, D) \propto w_t^i = \exp\left(-\frac{\beta}{J}\sum_{j=1}^{J} {}^j d_t^i\right) + \eta. \tag{8.14} $$

The parameters β and η are commonly used for robustness against outliers. In other words, the lower the average distance between the observations and the surface, the higher the weight (see Figure 8.5). The particles are resampled after each observation step based on their weight w_t^i to focus on the most likely positions.

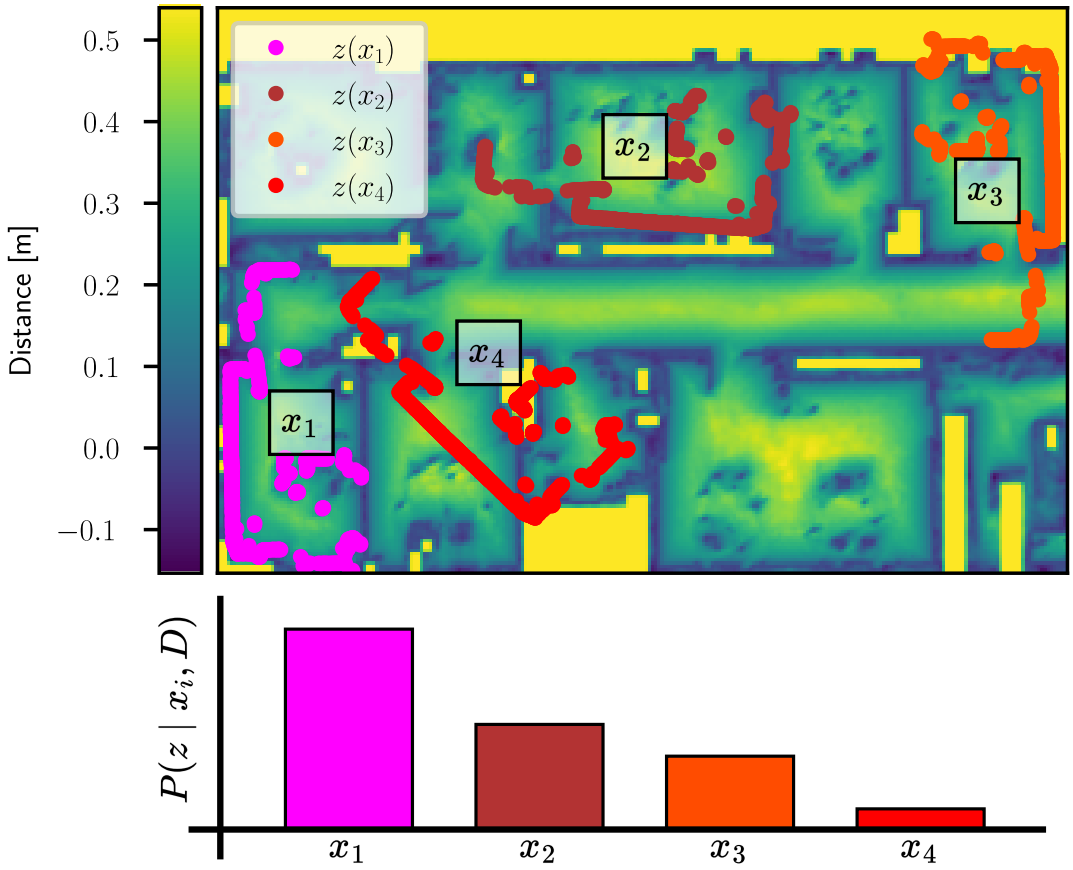


Figure 8.5: The neural distance field can be used in a particle filter to evaluate the likelihood of a measurement for each particle in the given NDF. The better the scans are aligned with the zero level of the NDF, the higher the likelihood.

8.3 Experimental Evaluation

In this section, we evaluate the localization performance to validate our proposed method. We show that our proposed training strategy can provide neural distance field maps which are well suited for scan registration, as well as global localization using MCL. To evaluate the scan registration performance, we will track the pose of a LiDAR sensor mounted on a car in the given maps using our point-to-NDF ICP. For global localization, we evaluate our method on a 2D indoor dataset of our office environment. Eventually, we provide our ablation studies to validate our proposed training methodology.

8.3.1 Training Setup

In this section, we provide the hyperparameters used to conduct the experiments, which, unless stated differently, are used throughout all experiments. We transform the coordinates of the point clouds to be in the range of [0,1] before passing them into the positional encoding. For the positional encoding, we sample $I_\omega = 30$ different frequencies. The default network uses a SIREN [207] backbone with a hidden feature dimension of size 128. For the training, we sample $N_i = 40$ points between the sensor origin and the endpoint to supervise the distance field using Equation (8.5), as well as additional 20 pairs of points with a distance up to 10 cm, which are distributed randomly in space for the regularization terms in Equation (8.8) and Equation (8.9). The coefficients between the different loss terms are $\alpha_{end} = 10^{-1}$, $\alpha_{Eik} = 10^{-4}$, and $\alpha_n = 10^{-3}$, as well as $\gamma = 3$. We optimize using AdamW with a start learning rate of 10^{-4}, which gets decreased with a cosine annealing scheduler to 10^{-7} over around 25,000 steps. The experiments have been evaluated on a desktop PC with i7 @ 3.5 GHz×8 CPU and an Nvidia RTX A5000.

8.3.2 3D Pose Tracking in Outdoor Scenes

For the 3D localization, we want to estimate the current vehicle pose by aligning local LiDAR point clouds with the map. We assume a rough initial location to be given, which is usually provided by a low-cost GPS sensor and track the vehicle's position using scan registration. The initial guess for ICP of the first timestamp is provided by a rough GPS position, whereas, for the following scans, we use a constant velocity model as the initial guess for ICP. We evaluate the registration performance on the Apollo-Southbay [141] dataset, which has multiple runs through the same areas recorded at different points in time. Since the environments changed substantially between the different points in time, points cannot always be matched, and therefore a robust kernel, here a Geman-McClure kernel with the parameter $k = 0.3$ m, is used.

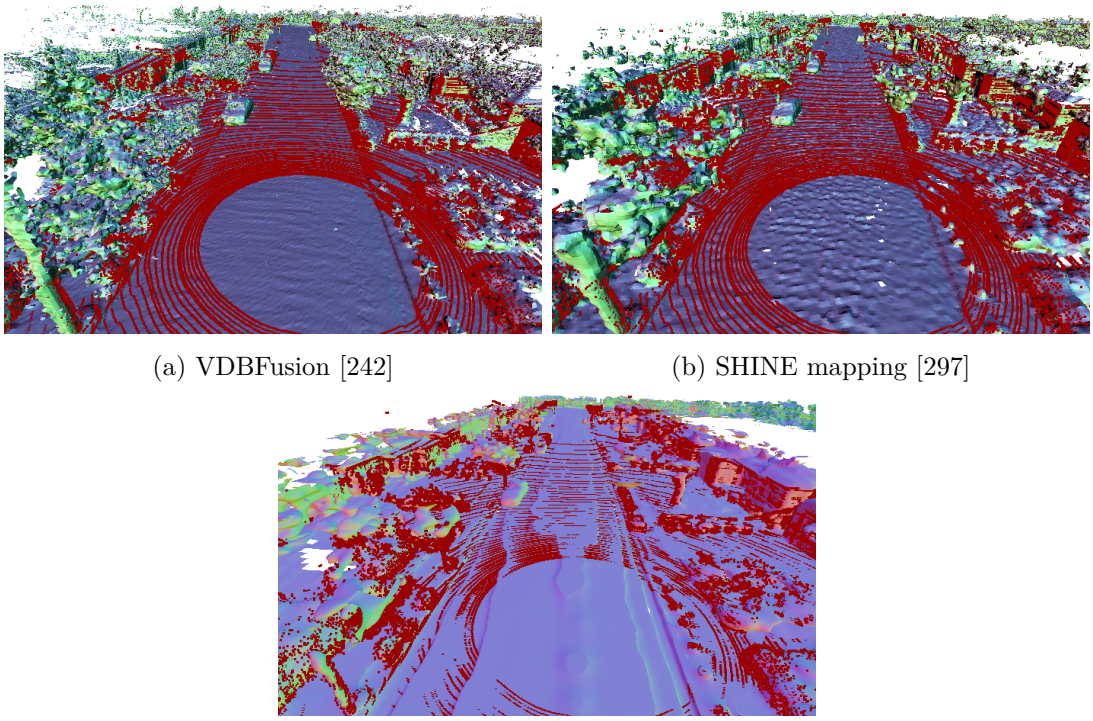
For the map generation, we use the first 800 scans of the ColumbiaPark-3 mapping run and the provided poses, which were obtained using a combination of GPS, IMU, and a SLAM system. Instead of training one big network for the whole scene, we found it beneficial to follow the key pose paradigm. For this, we use bounding boxes of 50 m size and 20 % overlap along the trajectory. We assign to each bounding box an NDF and train them incrementally based on the weights of the previous key pose. We fine-tune the maps for 10 epochs with the standard parameters. The evaluation will be done on 700 scans in the same area (starting at scan 5280) from the test run. Hyperparameters are tuned on 800 scans of the training set (starting at scan 6880). We compare against VDBFusion [242] as a

Table 8.1: Scan registration results

Approach	MAE($\boldsymbol{t}$) [m]	Memory [MB]	Runtime [s]
VDBFusion (KDTree)	0.072	170.8	0.87
VDBFusion (Projective)	0.072	170.8	1.52
LOAM (KDTree)	0.0714	6.3	0.23
SHINE (KDTree)	0.070	154.1	0.83
SuMa (Projective)	0.085	240.7	**0.03**
Ours	**0.059**	**5.1**	0.42

highly effective, traditional TSDF fusion pipeline, as well as against the recent learning-based SHINE mapping system [297], that utilizes also neural distance fields for mapping. For these baselines, we do the registration based on point-to-plane ICP using the same robust kernel using their meshes. Additionally, we compare against the surfel-based method SuMa [13] and the grid-based method LOAM [291], both in localization mode, i.e., first constructing the map on the mapping run, and use the second run from a different point in time solely for registration without updating the map. For the evaluation, we use the mean average translation error MAE($\boldsymbol{t}$), the memory consumption of the maps, as well as the average runtime for aligning a scan. The baselines either use a kD-tree or projective data associations to find correspondences.

The results are presented in Table 8.1. Our approach is able to outperform the baselines in mean average translation error MAE($\boldsymbol{t}$) while requiring also the least memory. All the approaches have a mean average rotation error lower than 0.1°. In Figure 8.6, the registration of a scan with respect to the map representations is depicted. Note that we use the mesh obtained using marching cubes [126] only for visualization and not for registration. We can see that the points are clipping inside our triangle mesh (Figure 8.6c), showing that the point cloud is well aligned. The meshes from the other approaches are more detailed, but the aligned point clouds seem like floating in the scene. This is due to the acquisition process of the TSDF, where for each voxel a weighted average over the ray distances is stored. This leads to an overestimation of the distance to the closest surface, resulting in slightly smaller objects and the aligned point cloud seems to be always in front of the surface. This effect is mitigated for our approach, due to projecting the ray distance along the surface normal and having a special loss term given in Equation (8.7) to enforce that the measured surface actually is the zero level of the distance field. SHINE mapping [297] is also supervising using the ray distance leading to similar effects as for TSDF.



(a) VDBFusion [242] (b) SHINE mapping [297]

(c) LocNDF (Ours)

Figure 8.6: Qualitative results of registering a scan (red points) to the maps generated by different distance field-based methods. The meshes are generated using the marching cubes algorithm. We use the mesh only for visualization purposes, the registration is done directly on the distance field.

8.3.3 2D Monte Carlo Localization

In this experiment, we evaluate the global localization performance using the MCL in our office environment. The robot is equipped with a 2D laser scanner (Hokuyo UTM-30 LX) and wheel odometry. We use directly the measurements $\boldsymbol{u}_t = (\Delta x, \Delta y, \Delta\theta)$ of the wheel odometry as a motion model with a Gaussian noise model of

$$\Sigma_{\boldsymbol{u}_t} = \begin{bmatrix} 5 \cdot 10^{-1} & 2.5 \cdot 10^{-2} & 1 \cdot 10^{-2} \\ 2.5 \cdot 10^{-2} & 5 \cdot 10^{-3} & 5 \cdot 10^{-3} \\ 5 \cdot 10^{-2} & 5 \cdot 10^{-4} & 2.5 \end{bmatrix}. \tag{8.15}$$

We use the same sequences for mapping and evaluation used by Kuang et al. [118] as well as the same setting using 100,000 particles for initialization for a fair comparison. We assume the particle filter to be converged when the standard deviation of the particle's position is below 30 cm to switch into the pose tracking mode with 10,000 particles. We reweight and resample the particles if the robot moved by at least 5 cm or 0.1 rad. The hyperparameters for the observation model are set to $\beta = 100$ and $\eta = 10^{-8}$.

Table 8.2: MCL results

	Approach	RMSE($\boldsymbol{t}$) @ 5cm	RMSE($\boldsymbol{t}$) @ 10cm	RMSE($\boldsymbol{t}$) @ 20cm
Seq 1	AMCL	- (0.0%)	- (0.0%)	- (0.0%)
	SRRG	0.034 (57.1%)	0.047 (88.6%)	0.049 (90.3%)
	IR-MCL	0.033 (60.3%)	0.047 (92.5%)	0.052 (95.7%)
	Ours	**0.031** (**76.6**%)	**0.041** (**96.2**%)	**0.047** (**99.2**%)
Seq 2	AMCL	0.037 (26.5%)	0.061 (56.2%)	0.089 (80.6%)
	SRRG	0.034 (41.4%)	0.059 (**87.4**%)	0.063 (92.6%)
	IR-MCL	**0.029** (60.1%)	0.048 (**87.4**%)	0.054 (**93.8**%)
	Ours	**0.028** (**78.0**%)	**0.032** (83.0%)	**0.042** (86.7%)
Seq 3	AMCL	0.038 (20.0%)	0.066 (58.7%)	0.099 (81.3%)
	SRRG	0.033 (36.5%)	0.050 (59.0%)	0.075 (71.0%)
	IR-MCL	0.033 (**68.2**%)	0.043 (**84.4**%)	0.054 (**89.8**%)
	Ours	**0.028** (54.4%)	**0.034** (62.0%)	**0.053** (70.0%)
Seq 4	AMCL	0.034 (**63.1**%)	0.048 (**88.7**%)	0.059 (**98.8**%)
	SRRG	0.035 (54.1%)	0.052 (83.7%)	**0.058** (88.2%)
	IR-MCL	0.033 (33.7%)	0.054 (59.9%)	0.091 (85.2%)
	Ours	**0.031** (42.4%)	**0.046** (64.5%)	0.069 (73.7%)
Seq 5	AMCL	- (0.0%)	- (0.0%)	- (0.0%)
	SRRG	0.035 (**48.8**%)	0.051 (**86.8**%)	**0.055** (89.0%)
	IR-MCL	0.032 (41.4%)	0.057 (81.3%)	0.064 (**89.8**%)
	Ours	**0.027** (41.5%)	**0.032** (45.0%)	0.071 (55.6%)

For constructing the map, we train for 15 epochs on the 31,608 training scans with the default parameters for our network. The NDF has no notion about unknown space by itself, i.e., areas that did not get supervised. Therefore, we also store a low-resolution bitmap of size $[100 \times 100]$ voxels to know roughly, which spaces are not supervised. Points that lie in an unknown area have the maximum distance assigned, rather than querying the network. We use the standard root mean squared error metric (RMSE) between the ground truth and estimated positions. The RMSE is evaluated for converged positions at certain thresholds (5 cm, 10 cm, 20 cm). Additionally, we provide the percentage of poses below the given thresholds. The results are averaged over 5 runs where the RMSE is only reported if all runs at least converged once, otherwise denoted as "-". We compare against the standard ROS1 localizer AMCL [61], the MCL system by Grisetti [72], as well as the recent, learning-based IR-MCL [118] approach.

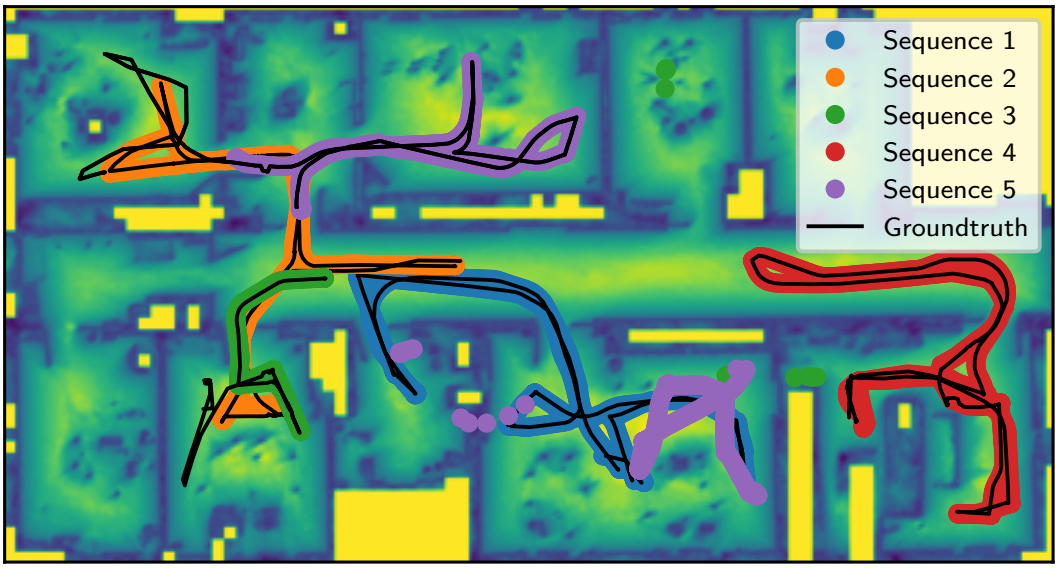


Figure 8.7: Qualitative localization results of our approach on the five sequences. The estimated poses are mostly aligned with the ground truth. In sequence 5 our approach converged to the wrong location but could recover later to the correct position. Sequence 4 took a long time to resolve ambiguities to finally converge to the correct position.

The results are depicted in Table 8.2. As can be seen, our approach is able to outperform the baselines in terms of RMSE on most sequences showing that our approach can provide reliable pose information once it is converged. We believe this is due to the continuous map representation and therefore we are not limited by the grid resolution. The convergence rate of our approach to the correct position is highly competitive with respect to baselines. Our approach runs at an average frame rate of around 2.6 Hz. A visualization of the localization is depicted in Figure 8.7.

8.3.4 Ablation Studies

In this section, we will provide ablation studies on certain hyperparameters to validate our choices and provide a deeper insight in the behaviour of the approach. In the following, we will first look at the different loss terms, and later the influence of the used backbone. We conduct the ablation studies on the scan registration task, as described in Section 8.3.2. All numbers provided in the following are evaluated on the validation set.

8.3.4.1 Loss Function

In this experiment, we validate the choice of our loss function. For this, we enable (✓; taking the default α) or disable (✗; set $\alpha = 0$) certain parts of the loss

Table 8.3: Ablation: Loss function

	project. dist.	α_{Eik}	α_{end}	α_n	MAE($\boldsymbol{t}$) [m]	MAE($\boldsymbol{R}$) [deg]
[A]	✓	✗	✗	✗	0.103	0.269
[B]	✓	✗	✓	✓	0.063	0.103
[C]	✓	✓	✗	✓	0.198	0.449
[D]	✓	✓	✓	✗	-	-
[E]	✓	✓	✓	✓	0.062	0.100
[F]	✗	✓	✓	✓	0.313	0.886

functions. In Table 8.3 are the results of this experiment shown. Disabling all the additional losses [A] increases the error by a factor of around 2 with respect to enabling them [E], showing the importance of the regularization losses. Disabling the regularizations of the gradients [B], [D] deteriorates the performance slightly to completely. While only disabling α_{end} [C] results in even worse performance than disabling all [A], suggesting the loss is useful to mitigate undesired effects of the other terms. Lastly in [F], we supervise by the ray distance d_i instead of the projected distance $\tilde{d}_i$ where the performance substantially degrades. For a better understanding of this phenomenon, we show a slice of the distance fields in Figure 8.8. The gradient for the ray distance looks toward the LiDAR sensor, pulling the points not toward the closest surface but along the ray. The projected distance n_i points more towards the closest surface.

8.3.4.2 Backbone and Feature Size

In this experiment, we investigate the impact of the backbone and the network size on the localization ability. The first backbone we use is the classical NeRF [155] architecture with a positional encoding and an MLP with 8 layers, layer norm, and leaky ReLU. There is a skip connection from the positional encoding to the 6^{th} layer. The second network is a SIREN [207], an MLP with 5 layers, layer norm, and sine nonlinearity. The results of this experiment are depicted in Table 8.4. With both backbones, we can see that a bigger network size does not necessarily mean a better localization performance. For the SIREN network, the results look more stable with the best performance for a hidden feature dimension of 128 ([N]). The classical NeRF network has less consistent but similar results ([G]-[K]). The results are in line with Sitzmann et al. [207] stating that the supervision of derivatives works better for SIRENs than for ReLU-based networks.

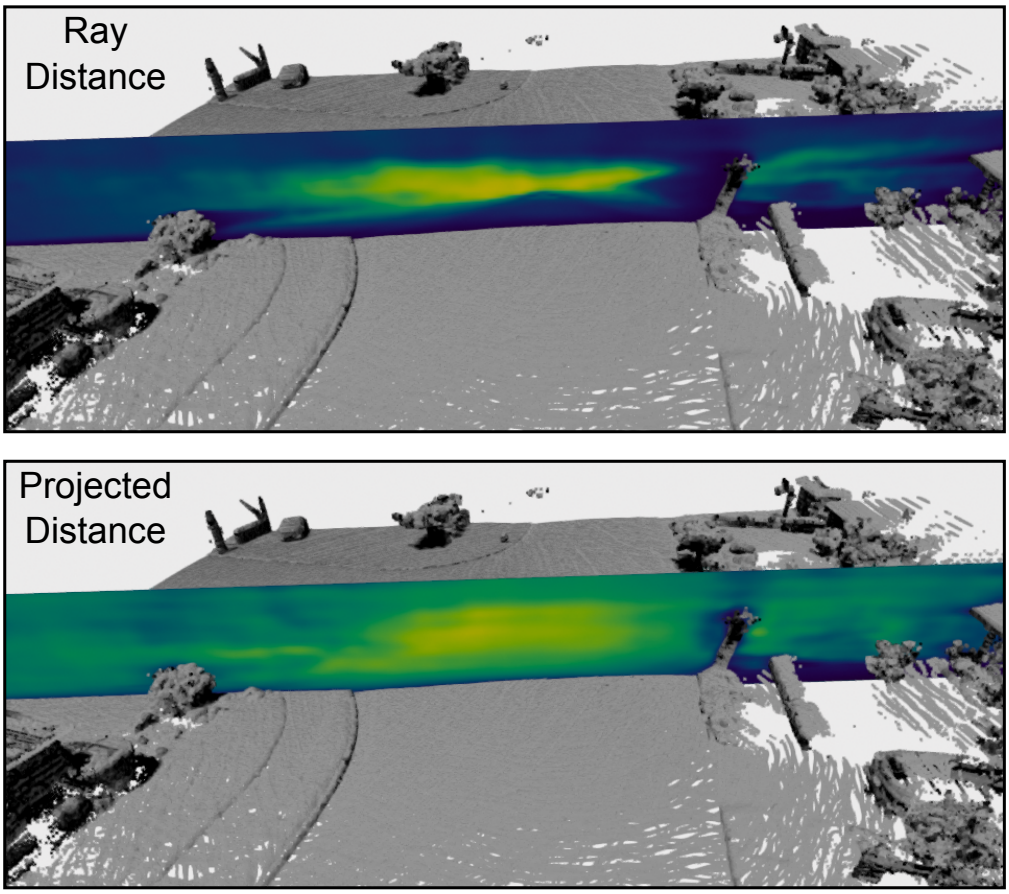


Figure 8.8: A visualization of the distance fields when supervising the distance field with the ray distance (top), as well as with the projected distance (bottom). The distance fields are evaluated on a slice in the middle of the scene and depicted by the color. When using the ray distance, one can see the field of view of the LiDAR sensor, leading to a gradient from the surface to the LiDAR center. For the projected distance on the other hand it looks more like a true Euclidean distance field where the gradient ascents radialy from the surface. Note, that the NDF is only supervised for the regions within the field of view of the sensor, leading to wrong values in unobserved areas.

8.3.5 Limitations

Despite these encouraging results, there is further space for improvement. The training time at each key pose takes around 20 min, which makes it only suitable for offline mapping but prohibits building the maps on the fly as it would be needed for online SLAM applications. In our case, we only optimized for obtaining the distance to the surface, but it would be interesting to regress point attributes such as semantics or colors as obtained from RGB-D sensors.

Table 8.4: Ablation: Backbone & feature size

	Backbone	Feature Size	MAE($\boldsymbol{t}$) [m]	MAE(R) [deg]	Memory [MB]
[G]	NeRF	32	0.215	0.402	**0.934**
[H]	NeRF	64	**0.061**	0.102	2.648
[I]	NeRF	128	0.104	0.257	8.989
[J]	NeRF	256	**0.061**	**0.100**	31.677
[K]	NeRF	512	0.084	0.122	125.289
[L]	SIREN	32	-	-	-
[M]	SIREN	64	0.126	0.395	1.541
[N]	SIREN	128	**0.062**	**0.100**	5.051
[O]	SIREN	256	0.068	0.101	17.732
[P]	SIREN	512	0.068	**0.099**	69.835

8.4 Conclusion

In this chapter, we investigated the usage of neural distance field (NDF) maps for robot localization. We showed how to directly learn the NDF from range sensor observations by projecting the measurements along the gradients of the network. The NDF provides us with a discretization-free and highly memory-efficient distance field, that allows us to compute directions to the closest surface elegantly through the Jacobian. As a result of that, ICP can be used to register point clouds directly to the NDF without the need to search for data associations. We have tested the point-to-NDF based registration in the context of pose tracking where we want to estimate the robot's motion with respect to the prebuild NDF map. Our system requires an initial guess that can be acquired using global localization methods or GNSS-based navigation solutions. From there on, the system successively registers the scans to the NDF. A constant velocity model predicts the motion of the robot such that the ICP only needs to correct for deviation to this model. Our method is able to register the scans with up to a few centimeters of accuracy.

With our described pose-tracking method, we propose an alternative way to estimate the robot's motion within the map after initial global localization. The full framework for obtaining a compressed map to estimate the motion in this map could look like the following: (1) as a prerequisite step all the robots are calibrated to obtain the sensor characteristics using our approach described in Chapter 3. Step (2) is to build the map where the robot drives through and perceives the desired scene using LiDAR sensors. From this raw data, we can

obtain a global point cloud map using our global LiDAR-bundle adjustment from Chapter 4. To have a memory-efficient representation that can be stored onboard or even transmitted over the network, we (3) split the map into smaller submaps that we compress using the compression approach from Chapter 5. In addition to those submaps, we can compute global descriptors that can be stored alongside our place recognition network as described in Chapter 6. To then (4) localize a robot at a later point in time in the given map, we can first estimate the submap in which the robot is located using the estimated global descriptors. Once we know our coarse location, we (5) can use our compressed point cloud registration method in Chapter 7 to obtain the 6 DoF of the robot with respect to the map. Once we have found our position, we can (6) switch our system into tracking mode. For this, we utilize precomputed NDFs for the submaps, in which we can track the robot's motion using point-to-NDF-based registration as described in this chapter.

Additionally, we have investigated briefly localization in indoor environments. Being spatially substantially smaller, and often only requiring a 2D position makes the use of particle filters an excellent choice to solve localization in those scenarios. A particle filter can track multiple potential paths until all ambiguities are resolved. We have investigated how to combine particle filters with NDFs for indoor localization. We have shown that using the NDF for the observation model in a particle filter can yield precise position estimates. Unfortunately, those methods scale badly with respect to environment size and estimation in higher dimensions. Limiting it to indoor environments and 2D estimates.

Chapter 9

Conclusion

Autonomous operating robots have the potential to take over tasks that people do not want to do, or even cannot do. Robotic systems are already driving automation in industrial production lines. However, robots are also becoming more and more part of our personal daily lives. Autonomous lawnmower and vacuum-cleaner robots operate in many households on a daily basis. However, letting robots operate in larger scale and open environments like autonomous driving cars remains a challenge. Many of those robotic systems rely on a map of the environment for navigation, planning, and interaction. One prerequisite for the robot to utilize a map is the capability to localize within the map. If you know where you are, then, you can plan where and how to move next to fulfill your tasks. For localizing a robot within a given map, one can compare the current surroundings of the robot with the map.

For this thesis, we developed multiple methods targeting LiDAR-based localization and mapping of outdoor environments. However, before one can go out to map the environment, one requires a well-calibrated sensor system to reliably perceive the environment. Therefore, our first work aims at calibrating different perception systems. Once our system is calibrated, we can start mapping the desired environment. Our LiDAR bundle adjustment method processes the local sensor data to obtain a global point cloud map. Due to the memory footprint of those maps, we investigated a compression algorithm to learn a more memory-efficient representation. Afterward, we looked into two methods for localizing a robot into those compressed maps using only LiDAR data. First, we developed a place recognition approach that coarsely finds the position of a robot in the map. Second, once we find our rough location, we can use our point cloud registration method to precisely estimate the pose with respect to the map. At last, we looked into a method for tracking the pose of the robot within a given map assuming we found the initial position already. We conducted experiments on publicly available datasets for validation and compared them with the state-of-the-art.

9.1 Our Key Contributions to LiDAR-based Localization and Mapping

In this thesis, we investigated multiple methods for large-scale localization and mapping of outdoor environments. Our first contribution tackles the calibration of different perception systems. Irrespective of localization, mapping, or any other task, the sensors of the robots need to be calibrated to reliably perceive the environment. We developed a calibration method that utilizes a calibration environment to estimate the calibration parameters of different multi-sensor systems. While other methods often rely on overlap between the sensors, our method utilizes a high-precision point cloud map from a TLS instead. Therefore, our sensors have no need of overlapping FoV, they just need to perceive parts that have been mapped with the TLS. We designed our method and calibration environment to exploit the characteristics of our sensors as well as possible. The calibration parameters are estimated using least squares adjustment.

Our second contribution is a LiDAR bundle adjustment method that can align thousands of point clouds into a consistent point cloud map. We estimate a continuous-time trajectory to account for the ego-motion of the sensor platform while scanning. The objective function we try to minimize is the alignment error between corresponding points. Due to the sheer amount of scans we sample for each point cloud a small amount of scans for the correspondence search. Due to the subsampling and the usage of an out-of-core buffer, we are able to process thousands of scans in a reasonable time. Our method is even able to jointly align the scans from multiple sessions.

Our third contribution targets the compression of point cloud maps. Point cloud maps can very quickly reach hundreds of gigabytes. To deal with large-scale point cloud maps, we first divide them into smaller submaps. Those submaps are compressed using our convolutional compression encoder which generates a sparse point cloud that has for each point a compression feature associated. Our specially designed compression decoder can then be used to recover the point cloud densely.

Our fourth contribution exploits our compressed representation to localize directly in the compressed maps. For this, we build upon the compression features to estimate for each generated compressed submap a small descriptor vector using a second convolutional neural network. To localize a robot a second time in the compressed map, one can compute such a descriptor vector also for the current robot's observations and search for the most similar submap, based on the feature similarity. To efficiently learn the weights of the neural network, we adjusted the idea of a feature bank and momentum encoder from the unsupervised learning domain to our supervised point cloud-based place recognition approach.

While the previous method only estimates in which submap the robot is located, our fifth contribution estimates the precise position and orientation of the robot in the map. To solve this task, we take the compressed point cloud map in which we are supposed to be and register the point cloud map from the current robot's observations. To register within a compressed map we estimate features for both point clouds. Those features can be used to find correspondences between the point clouds that can be used to estimate the pose. We investigated weighting schemes for the estimated correspondences to deal with ambiguities and missing correspondences. To ensure that the estimated features are well suited for matching, we directly optimized the features such that the matching will lead to correct pose estimates. For this, we reformulated the attention mechanism of the Transformer architecture to estimate corresponding points in a differentiable manner.

Our last contribution targets tracking the movements of a robot within a map. For this, we investigated the usage of implicit neural representations as an alternative memory-efficient map representation. We have shown how to consecutively estimate the robot's position in a neural distance field. Additionally, we investigated how to learn a neural distance field directly from sensor observations without requiring the ground truth Euclidean distance.

By this, we have made several contributions towards mapping and localization in large-scale outdoor environments. Overall, this thesis provides a full pipeline to realize mapping and localization from raw LiDAR sensor data. For the mapping, we have investigated sensor calibration, the construction of consistent maps, and representing the scenes in a compressed format. We contributed to localization, especially by exploring if robots can be localized directly in compressed map representations. Our proposed methods enable consistent mapping of larger areas, as well as precise localization in those maps. With this thesis, we have not only shown that it is possible to localize within compressed maps but even that we can do it more precisely than methods without memory constraints.

9.2 Open Source Contributions

The work discussed here in the thesis led to certain open-source software packages that implement the methods described here in this thesis. Additionally, they contain the optimized parameters of our learned neural networks, as well as data that we recorded or processed. The main software packages that were developed as part of this thesis are the following:

- IPB-Calibraiton from Chapter 3: `https://github.com/PRBonn/ipb_calibration`
- Deep Point Map Compression from Chapter 5: `https://github.com/PRBonn/deep-point-map-compression`
- Retriever from Chapter 6: `https://github.com/PRBonn/retriever`
- KPPR from Chapter 6: `https://github.com/PRBonn/kppr`
- DCPCR from Chapter 7: `https://github.com/PRBonn/DCPCR`
- LocNDF from Chapter 8: `https://github.com/PRBonn/locndf`

9.3 Future Work

In this work, we have investigated different localization and mapping methods for outdoor environments. We have shown how to construct and store point cloud maps efficiently, as well as a way to localize in our proposed map representations. In the following, we want to discuss shortly potential new research directions that could build upon our advances.

In this thesis, we have conducted a lot of experiments on different publicly available datasets to test and develop our methods. However, none of those datasets can really be used for evaluating every task. Therefore, it would be beneficial to have a dataset that suits many different tasks. This dataset should cover large areas, provide accurate ground truth, has repeated sessions for localization, and different benchmarks for a unified evaluation.

In order to operate autonomously, robots need to know what the environment looks like and require the ability to localize within the scene. However, these are usually just the first steps towards solving a particular task. For example, if we want a robot that shall pick up and deliver a package, the robot surely requires knowledge about the world, and where it is. But once this is resolved, it needs to plan a route to the destination, move there, and finally interact with the desired object. Hence, it would be interesting to investigate if and how our methods can be used for tasks like path planning, control, and manipulation.

In this thesis, we focused on geometric maps which describe a scene structurally. We have utilized LiDAR sensors which provide us with points sampled from the scene's surface. Although the pure geometric description of a scene is important, many tasks require additional information. A semantic understanding of the world is crucial for robots to operate autonomously. Estimating and incorporating semantics in a compact, and efficient map representation could be useful for solving many other tasks. Since geometry and semantics are often correlated, it has the potential to be stored together efficiently. On the other hand, also the here proposed methods could benefit from semantic information.

When speaking of semantic information, one might want to explore different types of scene representations. We as humans often think in terms of objects. This raises the question if an object-based representation might also be beneficial for robotic applications. This is especially relevant when robots have to interact and communicate with humans. For this, it would be interesting to investigate the usage of language models for robots.

Bibliography

[1] T. Akenine-Möller, E. Haines, and N. Hoffmann. *Real-time Rendering*. A K Peters/CRC Press, 4th edition, 2018.

[2] Y. Aoki, H. Goforth, A.S. Rangaprasad, and S. Lucey. PointNetLK: Robust & Efficient Point Cloud Registration Using PointNet. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[3] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN Architecture for Weakly Supervised Place Recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] M. Arora, L. Wiesmann, X. Chen, and C. Stachniss. Static Map Generation from 3D LiDAR Point Clouds Exploiting Ground Segmentation. *Journal on Robotics and Autonomous Systems (RAS)*, 159:104287, 2023.

[5] D. Azinović, R. Martin-Brualla, D.B. Goldman, M. Nießner, and J. Thies. Neural RGB-D Surface Reconstruction. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[6] J.L. Ba, J.R. Kiros, and G.E. Hinton. Layer Normalization. *arXiv preprint*, arXiv:1607.06450, 2016.

[7] D. Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint*, arXiv:1409.0473, 2014.

[8] X. Bai, Z. Luo, L. Zhou, H. Fu, L. Quan, and C.L. Tai. D3feat: Joint learning of dense detection and description of 3d local features. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[9] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. Whittaker. Ambler: An Autonomous Rover for Planetary Exploration. *Computer*, 22(6):18–26, 1989.

[10] J.T. Barron, B. Mildenhall, D. Verbin, P.P. Srinivasan, and P. Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

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

[12] 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.

[13] 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.

[14] J. Behley, V. Steinhage, and A. Cremers. Performance of Histogram Descriptors for the Classification of 3D Laser Range Data in Urban Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2012.

[15] 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.

[16] J. Beltrán, C. Guindel, A. De La Escalera, and F. García. Automatic Extrinsic Calibration Method for LiDAR and Camera Sensor Setups. In *Proc. of the IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*, 2022.

[17] R. Benjemaa and F. Schmitt. A solution for the registration of multiple 3d point sets using unit quaternions. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 1998.

[18] M. Bennewitz, C. Stachniss, S. Behnke, and W. Burgard. Utilizing Reflection Properties of Surfaces to Improve Mobile Robot Localization. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2009.

[19] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[20] R. Bergevin, M. Soucy, H. Gagnon, and D. Laurendeau. Towards a general multi-view registration technique. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 18(5):540–547, 1996.

[21] 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.

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

[23] F. Boniardi, T. Caselitz, R. Kümmerle, and W. Burgard. Robust LiDAR-based localization in architectural floor plans. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[24] S. Bouaziz, A. Tagliasacchi, and M. Pauly. Sparse Iterative Closest Point. *Computer Graphics Forum*, 32(5):113–123, 2013.

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

[26] H. Caesar, V. Bankiti, A.H. Lang, S. Vora, V.E. 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.

[27] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[28] D. Casado Herraez, L. Chang, M. Zeller, L. Wiesmann, J. Behley, M. Heidingsfeld, and C. Stachniss. SPR: Single-Scan Radar Place Recognition. *IEEE Robotics and Automation Letters (RA-L)*, 9(10):9079–9086, 2024.

[29] M. Chang, S. Yeon, S. Ryu, and D. Lee. SpoxelNet Spherical Voxel-Based Deep Place Recognition for 3D Point Clouds of Crowded Indoor Spaces. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[30] 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.

[31] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2020.

[32] 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.

[33] 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.

[34] 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.

[35] X. Chen and K. He. Exploring Simple Siamese Representation Learning. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[36] 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.

[37] J. Chibane, A. Mir, and G. Pons-Moll. Neural Unsigned Distance Fields for Implicit Function Learning. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[38] C. Choy, J. Gwak, and S. Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[39] C. Choy, J. Park, and V. Koltun. Fully convolutional geometric features. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[40] 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.

[41] J. Dai, L. Yan, H. Liu, C. Chen, and L. Huo. An Offline Coarse-To-Fine Precision Optimization Algorithm for 3D Laser SLAM Point Cloud. *Remote Sensing*, 11(20):2352, 2019.

[42] A. Das, J. Servos, and S. Waslander. 3D Scan Registration Using the Normal Distributions Transform with Ground Segmentation and Point Cloud Clustering. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2013.

[43] 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.

[44] F. Dellaert. GTSAM. `https://github.com/borglab/gtsam`, May 2022.

[45] 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.

[46] H. Deng, T. Birdal, and S. Ilic. PPFNet: Global Context Aware Local Features for Robust 3D Point Matching. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[47] H. Deng, T. Birdal, and S. Ilic. 3d local features for direct pairwise registration. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

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

[49] L. Di Giammarino, I. Aloise, C. Stachniss, and G. Grisetti. Visual Place Recognition using LiDAR Intensity Information. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2021.

[50] L. Di Giammarino, E. Giacomini, L. Brizi, O. Salem, and G. Grisetti. Photometric LiDAR and RGB-D Bundle Adjustment. *IEEE Robotics and Automation Letters (RA-L)*, 8(7):4362–4369, 2023.

[51] W. Dong and V. Isler. A novel method for the extrinsic calibration of a 2d laser rangefinder and a camera. *IEEE Sensors Journal*, 18(10):4200–4211, 2018.

[52] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2021.

[53] D. Droeschel and S. Behnke. Efficient Continuous-Time SLAM for 3D Lidar-Based Online Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[54] E. Einhorn and H.M. Gross. Generic ndt mapping in dynamic environments and its application for lifelong slam. *Journal on Robotics and Autonomous Systems (RAS)*, 69:28–39, 2015.

[55] A. El-Nouby, N. Neverova, I. Laptev, and H. Jégou. Training Vision Transformers for Image Retrieval. *arXiv preprint*, arXiv:2102.05644, 2021.

[56] J. Elseberg, D. Borrmann, and A. Nüchter. One billion points in the cloud – an octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 76:76–88, 2013.

[57] Z. Fan, Z. Song, H. Liu, Z. Lu, J. He, and X. Du. SVT-Net: Super Light-Weight Sparse Voxel Transformer for Large Scale Place Recognition. In *Proc. of the Conf. on Advancements of Artificial Intelligence (AAAI)*, 2022.

[58] C. Fang, S. Ding, Z. Dong, H. Li, S. Zhu, and P. Tan. Single-shot is enough: Panoramic infrastructure based calibration of multiple cameras and 3D LiDARs. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2021.

[59] M. Fischler and R. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[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] D. Fox. KLD-sampling: Adaptive particle filters. In *Proc. of the Conf. Neural Information Processing Systems (NIPS)*, 2001.

[62] S. Garg, N. Snderhauf, and M. Milford. Don't Look Back: Robustifying Place Categorization for Viewpoint and Condition-Invariant Place Recognition. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[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] N. Gelfand, N.J. Mitra, L.J. Guibas, and H. Pottmann. Robust Global Registration. In *Proc. of the Symp. on Geometry Processing*, 2005.

[65] S. Geman and D. McClure. Bayesian image analysis: An application to single photon emission tomography. In *Proc. of the American Statistical Association*, 1985.

[66] E. Giacomini, L. Brizi, L. Di Giammarino, O. Salem, P. Perugini, and G. Grisetti. Ca2Lib: Simple and Accurate LiDAR-RGB Calibration Using Small Common Markers. *Sensors*, 24(3), 2024.

[67] T. Golla and R. Klein. Real-time point cloud compression. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[68] C. Gomez, A. Hernandez, R. Barber, and C. Stachniss. Localization exploiting semantic and metric information in non-static indoor environments. *Journal of Intelligent & Robotic Systems*, 109(4):86, 2023.

[69] X. Gong, Y. Lin, and J. Liu. 3D LIDAR-Camera Extrinsic Calibration Using an Arbitrary Trihedron. *Sensors*, 13(2):1902–1918, 2013.

[70] Google. Draco 3d data compression. https://github.com/google/draco, 2017.

[71] J.B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[72] G. Grisetti. srrg-localizer2d (1.6.0). `https://gitlab.com/srrg-software/srrg_localizer2d`, 2018.

[73] 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.

[74] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2010.

[75] G. Grisetti, G. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi. Fast and Accurate SLAM with Rao-Blackwellized Particle Filters. *Journal on Robotics and Autonomous Systems (RAS)*, 55(1):30–38, 2007.

[76] G. Grisetti, R. Kümmerle, and K. Ni. Robust Optimization of Factor Graphs by using Condensed Measurements. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[77] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman. Implicit Geometric Regularization for Learning Shapes. *arXiv preprint*, arXiv:2002.10099, 2020.

[78] 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.

[79] T. Guadagnino. *Enhancing Spatio-Temporal Scalability in Graph-based SLAM systems*. PhD thesis, Sapienza University of Rome, 2022.

[80] S. Gumhold, Z. Kami, M. Isenburg, and H.P. Seidel. Predictive point-cloud compression. In *ACM SIGGRAPH Sketches*, 2005.

[81] M.H. Guo, J.X. Cai, Z.N. Liu, T.J. Mu, R.R. Martin, and S.M. Hu. PCT: Point Cloud Transformer. *Computational Visual Media*, 7:187–199, 2021.

[82] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep Learning for 3D Point Clouds: A Survey. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 43(12):4338–4364, 2021.

[83] S. Gupta, T. Guadagnino, B. Mersch, I. Vizzo, and C. Stachniss. Effectively Detecting Loop Closures using Point Cloud Density Maps. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2024.

[84] J.S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE Intl. Symp. on Computer Intelligence in Robotics and Automation (CIRA)*, 2000.

[85] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[86] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[87] R. Hendrikx, P. Pauwels, E. Torta, H. Bruyninckx, and M. van de Molengraft. Connecting Semantic Building Information Models and Robotics: An application to 2D LiDAR-based localization. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[88] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[89] 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.

[90] Z. Hou, Y. Yan, C. Xu, and H. Kong. HiTPR: Hierarchical Transformer for Place Recognition in Point Cloud. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[91] Y. Hu, W. Yang, Z. Ma, and J. Liu. Learning End-to-End Lossy Image Compression: A Benchmark. *arXiv preprint*, arXiv:2002.03711, 2020.

[92] J.K. Huang and J.W. Grizzle. Improvements to Target-Based 3D LiDAR to Camera Calibration. *IEEE Access*, 8:134101–134110, 2020.

[93] K. Huang and C. Stachniss. On Geometric Models and Their Accuracy for Extrinsic Sensor Calibration. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[94] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun. OctSqueeze: Octree-Structured Entropy Model for LiDAR Compression. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[95] T. Huang and Y. Liu. 3D point cloud geometry compression on deep learning. In *Proc. of the Intl. Conf. on Multimedia*, 2019.

[96] Y. Huang, J. Peng, C.C. Kuo, and M. Gopi. Octree-Based Progressive Geometry Coding of Point Clouds. In *In Proc. of the Symp. on Point-Based Graphics (PBG)*, 2006.

[97] P.J. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[98] L. Hui, M. Cheng, J. Xie, J. Yang, and M.M. Cheng. Efficient 3D Point Cloud Feature learning for Large-Scale Place Recognition. *IEEE Trans. on Image Processing*, 31:1258–1270, 2022.

[99] L. Hui, H. Yang, M. Cheng, J. Xie, and J. Yang. Pyramid Point Cloud Transformer for Large-Scale Place Recognition. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[100] K. Irie, M. Sugiyama, and M. Tomono. Target-less camera-lidar extrinsic calibration using a bagged dependence estimator. In *Proc. of the Intl. Conf. on Automation Science and Engineering (CASE)*, 2016.

[101] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver: General Perception with Iterative Attention. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2021.

[102] E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. *arXiv preprint*, arXiv:1611.01144, 2016.

[103] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating Local Descriptors into a Compact Image Representation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[104] A. Johnson and M. Hebert. Using spin images for effcient object recognition in cluttered 3D scenes. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 21(5):433–449, 1999.

[105] A.E. Johnson and S.B. Kang. Registration and Integration of Textured 3D Data. *Journal on Image and Vision Computing (IVC)*, 17(2):135–147, 1999.

[106] G. Jóźków. Terrestrial laser scanning data compression using JPEG-2000. In *Proc. of the Conf. of the German Society for Photogrammetry, Remote Sensing and Geoinformation (DGPF)*, 2017.

[107] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32(5):922–923, 1976.

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

[109] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. of the Eurographics Symposium on Geometry Processing*, 2006.

[110] 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.

[111] C. Kerl, J. Sturm, and D. Cremers. Robust Odometry Estimation for RGB-D Cameras. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2013.

[112] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised Contrastive Learning. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[113] M. Khoury, Q.Y. Zhou, and V. Koltun. Learning Compact Geometric Features. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2017.

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

[115] K. Koide, S. Oishi, M. Yokozuka, and A. Banno. General, Single-shot, Target-less, and Automatic LiDAR-Camera Extrinsic Calibration Toolbox. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2023.

[116] J. Komorowski. MinkLoc3D: Point Cloud Based Large-Scale Place Recognition. In *Proc. of the IEEE Winter Conf. on Applications of Computer Vision (WACV)*, 2021.

[117] X. Kong, X. Yang, G. Zhai, X. Zhao, X. Zeng, M. Wang, Y. Liu, W. Li, and F. Wen. Semantic Graph based Place Recognition for Point Clouds. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[118] H. Kuang, X. Chen, T. Guadagnino, N. Zimmerman, J. Behley, and C. Stachniss. IR-MCL: Implicit Representation-Based Online Global Localization. *IEEE Robotics and Automation Letters (RA-L)*, 8(3):1627–1634, 2023.

[119] J. Kümmerle and T. Kühner. Unified Intrinsic and Extrinsic Camera and LiDAR Calibration under Uncertainties. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[120] 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.

[121] I.S. Kweon, M. Hebert, E. Krotkov, and T. Kanade. Terrain Mapping for a Roving Planetary Explorer. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 1989.

[122] L. Landrieu and M. Simonovsky. Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[123] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[124] J. Lee, W. Im, S. Lee, and S.E. Yoon. Diffusion Probabilistic Models for Scene-Scale 3D Categorical Data. *arXiv preprint*, arXiv:2301.00527, 2023.

[125] V. Lepetit, F. Moreno-Noguer, and P. Fua. EPnP: An Accurate O(n) Solution to the PnP Problem. *Intl. Journal of Computer Vision (IJCV)*, 81:155–166, 2009.

[126] T. Lewiner, H. Lopes, A.W. Vieira, and G. Tavares. Efficient implementation of marching cubes cases with topological guarantees. *Journal of Graphics Tools*, 8(2):1–15, 2003.

[127] C. Li, F. Yan, and Y. Zhuang. Sequence matching enhanced 3D place recognition using candidate rearrangement. *IET Cyber-Systems and Robotics*, 4(3):189–199, 2022.

[128] Y. Li, Z. Kuang, T. Li, G. Zhou, S. Zhang, and Z. Yan. ActiveSplat: High-Fidelity Scene Reconstruction through Active Gaussian Splatting. *arXiv preprint*, arXiv:2410.21955, 2024.

[129] J. Liu, L. Kong, J. Yang, and W. Liu. Towards better data exploitation in self-supervised monocular depth estimation. *IEEE Robotics and Automation Letters (RA-L)*, 9(1):763–770, 2023.

[130] X. Liu, M. Yan, and J. Bohg. MeteorNet: Deep Learning on Dynamic 3D Point Cloud Sequences. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[131] X. Liu, C. Yuan, and F. Zhang. Targetless Extrinsic Calibration of Multiple Small FoV LiDARs and Cameras using Adaptive Voxelization. *IEEE Trans. on Instrumentation and Measurement*, 71:1–12, 2022.

[132] Z. Liu, S. Zhou, C. Suo, P. Yin, W. Chen, H. Wang, H. Li, and Y.H. Liu. LPD-Net: 3D Point Cloud Learning for Large-Scale Place Recognition and Environment Analysis. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[133] Z. Liu and F. Zhang. Balm: Bundle adjustment for lidar mapping. *IEEE Robotics and Automation Letters (RA-L)*, 6(2):3184–3191, 2021.

[134] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint*, arXiv:1711.05101, 2017.

[135] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Intl. Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

[136] S. Lowry and H. Andreasson. Lightweight, Viewpoint-Invariant Visual Place Recognition in Changing Environments. *IEEE Robotics and Automation Letters (RA-L)*, 3(2):957–964, 2018.

[137] F. Lu and E. Milios. Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4:333–349, 1997.

[138] 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.

[139] 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.

[140] W. Lu, G. Wan, Y. Zhou, X. Fu, P. Yuan, and S. Song. DeepVCP: An End-to-End Deep Neural Network for Point Cloud Registration. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[141] 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.

[142] J. Ma, X. Chen, J. Xu, and G. Xiong. SeqOT: A Spatial-Temporal Transformer Network for Place Recognition Using Sequential LiDAR Data. *arXiv preprint*, arXiv:2209.07951, 2022.

[143] 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.

[144] D. Maggio, M. Abate, J. Shi, C. Mario, and L. Carlone. Loc-NeRF: Monte Carlo Localization using Neural Radiance Fields. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2023.

[145] R. Marcuzzi, L. Nunes, L. Wiesmann, J. Behley, and C. Stachniss. Mask-Based Panoptic LiDAR Segmentation for Autonomous Driving. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1141–1148, 2023.

[146] R. Marcuzzi, L. Nunes, L. Wiesmann, E. Marks, J. Behley, and C. Stachniss. Mask4D: End-to-End Mask-Based 4D Panoptic Segmentation for LiDAR

Sequences. *IEEE Robotics and Automation Letters (RA-L)*, 8(11):7487–7494, 2023.

[147] 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.

[148] E. Marks, M. Sodano, F. Magistri, L. Wiesmann, D. Desai, R. Marcuzzi, J. Behley, and C. Stachniss. High Precision Leaf Instance Segmentation in Point Clouds Obtained Under Real Field Conditions. *IEEE Robotics and Automation Letters (RA-L)*, 8(8):4791–4798, 2023.

[149] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[150] 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.

[151] R. Mekuria, K. Blom, and P. Cesar. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Trans. on Circuits and Systems for Video Technology (TCSVT)*, 27(4):828–842, 2016.

[152] P. Merkle, A. Smolic, K. Muller, and T. Wiegand. Multi-view video plus depth representation and coding. In *Proc. of the IEEE Intl. Conf. on Image Processing (ICIP)*, 2007.

[153] B. Merry, P. Marais, and J. Gain. Compression of dense and regular point clouds. In *Proc. of the Intl. Conf. on Computer graphics, virtual reality, visualisation and interaction in Africa*, 2006.

[154] 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.

[155] B. Mildenhall, P. Srinivasan, M. Tancik, J. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[156] 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.

[157] D. Munoz, J.A. Bagnell, N. Vandapel, and M. Hebert. Contextual Classification with Functional Max-Margin Markov Networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[158] 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.

[159] T. Naseer, W. Burgard, and C. Stachniss. Robust Visual Localization Across Seasons. *IEEE Trans. on Robotics (TRO)*, 34(2):289–302, 2018.

[160] F. Nenci, L. Spinello, and C. Stachniss. Effective Compression of Range Data Streams for Remote Robot Operations using H.264. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[161] 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. Symp. on Mixed and Augmented Reality (ISMAR)*, 2011.

[162] T.M. Nguyen, S. Yuan, T.H. Nguyen, P. Yin, H. Cao, L. Xie, M. Wozniak, P. Jensfelt, M. Thiel, J. Ziegenbein, and N. Blunder. MCD: Diverse Large-Scale Multi-Campus Dataset for Robot Perception. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[163] 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.

[164] L. Nunes, R. Marcuzzi, B. Mersch, J. Behley, and C. Stachniss. Scaling Diffusion Models to Real-World 3D LiDAR Scene Completion. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[165] L. Nunes, L. Wiesmann, R. Marcuzzi, X. Chen, J. Behley, and C. Stachniss. Temporal Consistent 3D LiDAR Representation Learning for Semantic Perception in Autonomous Driving. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[166] E. Olson. AprilTag: A Robust and Flexible Visual Fiducial System. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2011.

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

[168] Y. Pan, X. Zhong, L. Wiesmann, T. Posewsky, J. Behley, and C. Stachniss. PIN-SLAM: LiDAR SLAM Using a Point-Based Implicit Neural Representation for Achieving Global Map Consistency. *IEEE Trans. on Robotics (TRO)*, 40:4045–4064, 2024.

[169] G. Pandey, J. McBride, S. Savarese, and R. Eustice. Extrinsic Calibration of a 3D Laser Scanner and an Omnidirectional Camera. In *Proc. of the Conf. on Advancements of Artificial Intelligence (AAAI)*, 2010.

[170] G. Pandey, J. McBride, S. Savarese, and R. Eustice. Automatic Targetless Extrinsic Calibration of a 3D Lidar and Camera by Maximizing Mutual Information. In *Proc. of the Conf. on Advancements of Artificial Intelligence (AAAI)*, 2012.

[171] C. Park, P. Moghadam, S. Kim, A. Elfes, C. Fookes, and S. Sridharan. Elastic LiDAR Fusion: Dense Map-Centric Continuous-Time SLAM. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[172] C. Park, P. Moghadam, J.L. Williams, S. Kim, S. Sridharan, and C. Fookes. Elasticity Meets Continuous-Time: Map-Centric Dense 3D LiDAR SLAM. *IEEE Trans. on Robotics (TRO)*, 38(2):978–997, 2022.

[173] C. Park, Y. Jeong, M. Cho, and J. Park. Fast Point Transformer. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[174] J. Park, Q. Zhou, and V. Koltun. Colored Point Cloud Registration Revisited. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2017.

[175] 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.

[176] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner,

L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2019.

[177] P. Pfaff, R. Triebel, and W. Burgard. An Efficient Extension to Elevation Maps for Outdoor Terrain Mapping and Loop Closing. *Intl. Journal of Robotics Research (IJRR)*, 26(2):217–230, 2007.

[178] F. Poiesi and D. Boscaini. Learning general and distinctive 3D local deep descriptors for point cloud registration. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 45(3):3979–3985, 2022.

[179] 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.

[180] C.R. Qi, H. Su, K. Mo, and L.J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[181] C. Qi, K. Yi, H. Su, and L.J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proc. of the Conf. Neural Information Processing Systems (NIPS)*, 2017.

[182] M. Quach, G. Valenzise, and F. Dufaux. Learning convolutional transforms for lossy point cloud geometry compression. In *Proc. of the IEEE Intl. Conf. on Image Processing (ICIP)*, 2019.

[183] M. Quach, G. Valenzise, and F. Dufaux. Improved Deep Point Cloud Geometry Compression. In *Proc. of the IEEE Intl. Workshop on Multimedia Signal Processing (MMSP)*, 2020.

[184] C. Reiser, S. Peng, Y. Liao, and A. Geiger. KiloNeRF: Speeding Up Neural Radiance Fields With Thousands of Tiny MLPs. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[185] G. Riegler, A. Ulusoy, and A. Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[186] S.A. Rodriguez F., V. Fremont, and P. Bonnifait. Extrinsic calibration between a multi-layer lidar and a camera. In *Proc. of the IEEE Intl. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, 2008.

[187] 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.

[188] A. Rosenfeld and J.L. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13(4):471–494, 1966.

[189] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2011.

[190] M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of compact 3d models based on the detection of recurrent structures. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[191] M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Highly Accurate 3D Aurface Models by Sparse Surface Adjustment. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2012.

[192] D. Rumelhart, G. Hinton, and R. Williams. *Learning Representations by Back-Propagating Errors*, pages 696–699. MIT press, 1988.

[193] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. of Intl. Conf. on 3-D Digital Imaging and Modeling*, 2001.

[194] R. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2009.

[195] 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.

[196] 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.

[197] D. Scaramuzza, A. Harati, and R. Siegwart. Extrinsic Self Calibration of a Camera and a 3D Laser Range Finder from Natural Scenes. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[198] R. Schnabel and R. Klein. Octree-based Point-Cloud Compression. In *In Proc. of the Symp. on Point-Based Graphics (PBG)*, 2006.

[199] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for point-cloud shape detection. In *Computer Graphics Forum*, 2007.

[200] J. Schneider, F. Schindler, T. Läbe, and W. Förstner. Bundle adjustment for multi-camera systems with points at infinity. In *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2012.

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

[202] 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.

[203] M. Shakeri and H. Zhang. Illumination Invariant Representation of Natural Images for Visual Place Recognition. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[204] T. Shan, B. Englot, F. Duarte, C. Ratti, and D. Rus. Robust Place Recognition using an Imaging Lidar. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[205] 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.

[206] 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.

[207] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit Neural Representations with Periodic Activation Functions. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[208] J. Sivic and A. Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2003.

[209] L.N. Smith and N. Topin. Super-convergence: Very fast training of neural networks using large learning rates. *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, 11006:369–386, 2019.

[210] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2015.

[211] R. Spezialetti, S. Salti, and L.D. Stefano. Learning an Effective Equivariant 3D Descriptor without Supervision. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[212] C. Stachniss and W. Burgard. Exploring Unknown Environments with Mobile Robots using Coverage Maps. In *Proc. of the Intl. Conf. on Artificial Intelligence (IJCAI)*, 2003.

[213] C. Stachniss and W. Burgard. Mobile Robot Mapping and Localization in Non-Static Environments. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 2005.

[214] B. Steder, G. Grisetti, and W. Burgard. Robust Place Recognition for 3D Range Data Based on Point Features. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2010.

[215] 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.

[216] 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.

[217] T. Stoyanov, M. Magnusson, H. Andreasson, and A.J. Lilienthal. Fast and accurate scan registration through minimization of the distance between compact 3D NDT representations. *Intl. Journal of Robotics Research (IJRR)*, 31(12):1377–1393, 2012.

[218] T. Stoyanov, J. Saarinen, H. Andreasson, and A. Lilienthal. Normal Distributions Transform Occupancy Map Fusion: Simultaneous Mapping and Tracking in Large Scale Dynamic Environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[219] 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.

[220] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.H. Yang, and J. Kautz. SPLATNet: Sparse Lattice Networks for Point Cloud Processing. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[221] X. Sun, H. Ma, Y. Sun, and M. Liu. A novel point cloud compression algorithm based on clustering. *IEEE Robotics and Automation Letters (RA-L)*, 4(2):2132–2139, 2019.

[222] N. Sünderhauf, S. Shirazi, F. Dayoub, B. Upcroft, and M. Milford. On the performance of convnet features for place recognition. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[223] K. Tang, P. Song, and X. Chen. Signature of Geometric Centroids for 3D Local Shape Description and Partial Shape Matching. In *Proc. of the Asian Conf. on Computer Vision (ACCV)*, 2017.

[224] T. Tao, L. Gao, G. Wang, Y. Lao, P. Chen, H. Zhao, D. Hao, X. Liang, M. Salzmann, and K. Yu. LiDAR-NeRF: Novel LiDAR View Synthesis via Neural Radiance Fields. In *Proc. of the ACM Intl. Conf. on Multimedia*, 2024.

[225] The APRIL Robotics Laboratory at the University of Michigan. Apriltag 3. `https://github.com/AprilRobotics/apriltag`.

[226] 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.

[227] H. Thomas, Y.H.H. Tsai, T.D. Barfoot, and J. Zhang. KPConvX: Modernizing Kernel Point Convolution with Kernel Attention. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[228] S. Thrun and M. Montemerlo. The graph SLAM algorithm with applications to large-scale mapping of urban structures. *Intl. Journal of Robotics Research (IJRR)*, 25(5-6):403, 2006.

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

[230] R. Triebel, K. Kersting, and W. Burgard. Robust 3D Scan Point Classification using Associative Markov Networks. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2006.

[231] 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.

[232] B. Triggs, P.F. McLauchlan, R.I. Hartley, and A.W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proc. of the Intl. Workshop on Vision Algorithms: Theory and Practice*, 1999.

[233] D. Tsai, S. Worrall, M. Shan, A. Lohr, and E.M. Nebot. Optimising the selection of samples for robust lidar camera calibration. In *Proc. of the IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*, 2021.

[234] Y.X. Tsai, R. Shao, P. Gui, B. Li, and L. Wang. Infrastructure Based Calibration of a Multi-Camera and Multi-LiDAR System Using Apriltags. In *Proc. of the IEEE Intelligent Vehicles Symposium*, 2018.

[235] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda. Point cloud compression for 3D LiDAR sensor using recurrent neural network with residual blocks. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[236] I. Ueda, Y. Fukuhara, H. Kataoka, H. Aizawa, H. Shishido, and I. Kitahara. Neural Density-Distance Fields. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2022.

[237] A. Uy and G. Lee. PointNetVLAD: Deep point cloud based retrieval for large-scale place recognition. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[238] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. In *Proc. of the Conf. Neural Information Processing Systems (NIPS)*, 2017.

[239] S. Verma, J. Berrio, S. Worrall, and E. Nebot. Automatic extrinsic calibration between a camera and a 3D Lidar using 3D point and plane correspondences. In *Proc. of the IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*, 2019.

[240] 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.

[241] 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.

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

[243] 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.

[244] 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.

[245] O. Vysotska and C. Stachniss. Lazy Data Association For Image Sequences Matching Under Substantial Appearance Changes. *IEEE Robotics and Automation Letters (RA-L)*, 1(1):213–220, 2016.

[246] O. Vysotska and C. Stachniss. Effective Visual Place Recognition Using Multi-Sequence Maps. *IEEE Robotics and Automation Letters (RA-L)*, 4(2):1730–1736, 2019.

[247] 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.

[248] H. Wang, Y. Zhu, H. Adam, A. Yuille, and L.C. Chen. MaX-DeepLab: End-to-end panoptic segmentation with mask transformers. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[249] P. Wang, Y. Liu, Y. Guo, C. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Trans. on Graphics (TOG)*, 36(4):1–11, 2017.

[250] R. Wang, Y. Shen, W. Zuo, S. Zhou, and N. Zheng. TransVPR: Transformer-Based Place Recognition with Multi-Level Attention Aggregation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[251] W. Wang, K. Sakurada, and N. Kawaguchi. Reflectance Intensity Assisted Automatic and Accurate Extrinsic Calibration of 3D LiDAR and Panoramic Camera Using a Printed Chessboard. *Remote Sensing*, 9(8):851, 2017.

[252] 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.

[253] Y. Wang and J.M. Solomon. Deep Closest Point: Learning Representations for Point Cloud Registration. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[254] Y. Wang and J.M. Solomon. PRNet: Self-Supervised Learning for Partial-to-Partial Registration. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2019.

[255] 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.

[256] 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.

[257] 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.

[258] L. Wiesmann, R. Marcuzzi, C. Stachniss, and J. Behley. Retriever: Point Cloud Retrieval in Compressed 3D Maps. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[259] L. Wiesmann, E. Marks, S. Gupta, T. Guadagnino, J. Behley, and C. Stachniss. Efficient LiDAR Bundle Adjustment for Multi-Scan Alignment Utilizing Continuous-Time Trajectories. *arXiv preprint*, arXiv:2412.11760, 2024.

[260] 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.

[261] L. Wiesmann, L. Nunes, J. Behley, and C. Stachniss. KPPR: Exploiting Momentum Contrast for Point Cloud-Based Place Recognition. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):592–599, 2023.

[262] 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.

[263] L. Wiesmann, T. Läbe, L. Nunes, J. Behley, and C. Stachniss. Joint Intrinsic and Extrinsic Calibration of Perception Systems Utilizing a Calibration

Environment. *IEEE Robotics and Automation Letters (RA-L)*, 9(10):9103–9110, 2024.

[264] F. Williams, M. Trager, J. Bruna, and D. Zorin. Neural Splines: Fitting 3D Surfaces with Infinitely-Wide Neural Networks. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[265] C.Y. Wu, R. Manmatha, A.J. Smola, and P. Krahenbuhl. Sampling Matters in Deep Embedding Learning. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2017.

[266] Y. Wu, T. Guadagnino, L. Wiesmann, L. Klingbeil, C. Stachniss, and H. Kuhlmann. LIO-EKF: High Frequency LiDAR-Inertial Odometry using Extended Kalman Filters. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2024.

[267] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[268] Y. Xia, Y. Xu, S. Li, R. Wang, J. Du, D. Cremers, and U. Stilla. SOE-Net: A Self-Attention and Orientation Encoding Network for Point Cloud Based Place Recognition. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[269] T. Xiao, P. Dollar, M. Singh, E. Mintun, T. Darrell, and R. Girshick. Early Convolutions help Transformers see Better. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2021.

[270] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu. On Layer Normalization in the Transformer Architecture. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2020.

[271] Y. Xiong and K. Turkowski. Creating image-based VR using a self-calibrating fisheye lens. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1997.

[272] 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.

[273] D. Yan, X. Lyu, J. Shi, and Y. Lin. Efficient Implicit Neural Reconstruction Using LiDAR. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2023.

[274] G. Yan, F. He, C. Shi, P. Wei, X. Cai, and Y. Li. Joint Camera Intrinsic and LiDAR-Camera Extrinsic Calibration. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2023.

[275] H. Yang, J. Shi, and L. Carlone. TEASER: Fast and Certifiable Point Cloud Registration. *IEEE Trans. on Robotics (TRO)*, 37(2):314–333, 2020.

[276] J. Yang, H. Li, D. Campbell, and Y. Jia. Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 38(11):2241–2254, 2015.

[277] R. Yang and S. Mandt. Lossy Image Compression with Conditional Diffusion Models. *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 36, 2024.

[278] Z.J. Yew and G.H. Lee. 3DFeat-Net: Weakly Supervised Local 3D Features for Point Cloud Registration. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2018.

[279] Z.J. Yew and G.H. Lee. RPM-Net: Robust Point Matching using Learned Features. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[280] H. Yin, Y. Wang, X. Ding, L. Tang, S. Huang, and R. Xiong. 3D LiDAR-Based Global Localization Using Siamese Neural Network. *IEEE Trans. on Intelligent Transportation Systems (TITS)*, 21(4):1380–1392, 2019.

[281] H. Yin, X. Xu, S. Lu, X. Chen, R. Xiong, S. Shen, C. Stachniss, and Y. Wang. A Survey on Global LiDAR Localization: Challenges, Advances and Open Problems. *Intl. Journal of Computer Vision (IJCV)*, 132:1–33, 2024.

[282] H. Yin, Z. Lin, and J.K. Yeoh. Semantic Localization on BIM-generated Maps using a 3D LiDAR Sensor. *Automation in Construction*, 146:104641, 2023.

[283] H. Yin, L. Tang, X. Ding, Y. Wang, and R. Xiong. Locnet: Global localization in 3d point clouds for mobile vehicles. In *Proc. of the IEEE Vehicles Symposium (IV)*, 2018.

[284] L. Yu, X. Li, C. Fu, D. Cohen-Or, and P. Heng. Pu-net: Point cloud upsampling network. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[285] J. Yue-Hei Ng, F. Yang, and L.S. Davis. Exploiting Local Features from Deep Networks for Image Retrieval. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[286] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2021.

[287] M. Zeller, J. Behley, M. Heidingsfeld, and C. Stachniss. Gaussian Radar Transformer for Semantic Segmentation in Noisy Radar Data. *IEEE Robotics and Automation Letters (RA-L)*, 8(1):344–351, 2023.

[288] M. Zeller, V. Sandhu, B. Mersch, J. Behley, M. Heidingsfeld, and C. Stachniss. Radar Instance Transformer: Reliable Moving Instance Segmentation in Sparse Radar Point Clouds. *IEEE Trans. on Robotics (TRO)*, 40:2357–2372, 2024.

[289] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[290] C. Zhang, D. Florencio, and C. Loop. Point cloud attribute compression with graph transform. In *Proc. of the IEEE Intl. Conf. on Image Processing (ICIP)*, 2014.

[291] J. Zhang and S. Singh. LOAM: Lidar Odometry and Mapping in Real-time. In *Proc. of Robotics: Science and Systems (RSS)*, 2014.

[292] J. Zhang, F. Zhang, S. Kuang, and L. Zhang. Nerf-lidar: Generating realistic lidar point clouds with neural radiance fields. In *Proc. of the Conf. on Advancements of Artificial Intelligence (AAAI)*, 2024.

[293] Q. Zhang and R. Pless. Extrinsic Calibration of a Camera and Laser Range Finder (improves camera calibration). In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2004.

[294] W. Zhang and C. Xiao. PCAN: 3D Attention Map Learning using Contextual Information for Point Cloud based Retrieval. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[295] Z. Zhang. A Flexible New Technique for Camera Calibration. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 22(11):1330–1334, 2000.

[296] H. Zhao, L. Jiang, J. Jia, P.H. Torr, and V. Koltun. Point Transformer. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[297] X. Zhong, Y. Pan, J. Behley, and C. Stachniss. SHINE-Mapping: Large-Scale 3D Mapping Using Sparse Hierarchical Implicit Neural Representations. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2023.

[298] L. Zhou, Z. Li, and M. Kaess. Automatic Extrinsic Calibration of a Camera and a 3D LiDAR using Line and Plane Correspondences. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[299] Q. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv preprint*, arXiv:1801.09847, 2018.

[300] N. Zimmerman, T. Guadagnino, X. Chen, J. Behley, and C. Stachniss. Long-Term Localization using Semantic Cues in Floor Plan Maps. *IEEE Robotics and Automation Letters (RA-L)*, 8(1):176–183, 2023.

[301] N. Zimmerman, M. Sodano, E. Marks, J. Behley, and C. Stachniss. Constructing Metric-Semantic Maps using Floor Plan Priors for Long-Term Indoor Localization. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2023.

[302] N. Zimmerman, L. Wiesmann, T. Guadagnino, T. Läbe, J. Behley, and C. Stachniss. Robust Onboard Localization in Changing Environments Exploiting Text Spotting. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2022.

List of Figures

List of Tables