Autonomous 3D Robot Navigation on Volumetric Map Representations

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Marvin Große Besselmann

Tag der mündlichen Prüfung: 04.12.2024

1. Referent: Prof. Dr. Rüdiger Dillmann
2. Referent: Prof. Dr. Karsten Berns

Credits

The research presented in this thesis was accomplished during my time as a research scientist at the FZI Forschungszentrum Informatik in the Department of Interactive Diagnostic and Service Systems (IDS). I would like to thank all the people who have accompanied, helped, and supported me on this long journey and have positively influenced this work.

First and foremost, I extend my deepest gratitude to my supervisor, Prof. Rüdiger Dillmann, for his continuous support throughout my thesis journey and for providing me with the opportunity to work in this research lab. I am especially thankful for his invaluable guidance, eternal patience, and for always encouraging me to push the boundaries of my knowledge. I would also like to express my sincere gratitude to Prof. Karsten Berns for his genuine interest in my research, valuable insights, and especially for his essential last-minute feedback that greatly improved the presentation of my work.

I owe a huge thanks to my department manager, Arne Rönnau, for allowing me to engage in various exciting projects and for often placing more faith in my research ideas than I had myself. His ongoing support, insightful scientific guidance, and challenging questions were crucial to the success of this work. Similarly, I like to thank Georg Heppner, whose northern charm has always given me a sense of home, no matter where I am.

My heartfelt thanks to the entire IDS-Team for the incredible journey and the cherished memories that will stay with me forever. I am particularly thankful to Felix Exner, whose influence has profoundly shaped who I am today, including all those sleep-deprived French excursions. I am thankful to Tristan for countless valuable discussions and for joining me in exploring extraordinary workplaces, from abandoned mines over deserts to lunar surfaces. My gratitude also extends to Stefan Scherzinger for his relentless optimism and encouragement during times of doubt. I would also like to express my appreciation to Timothée Büttner, David Timmermann, Nicolas Hügel, Carsten Plasberg, Johannes Mangler, Niklas Spielbauer, Christian Eichmann and David Oberacker for the great times, fun trips and working together on so many amazing projects. Furthermore, I want to express my gratitude to several exceptional students: Ramona Häuselmann, Samuel Mauch and Max Schik. I also like to thank Jens Doll for the many deep discussion we had in our office on literally every topic. Further thanks are extended to Sonja Göttl and Nicole Rösch for their constant professional as well as emotional support and invaluable assistance in countless circumstances.

I would particularly like to thank Lennart Puck, Genessis Pérez, Lea Steffen and Fabian Steffen for their continuous friendship, amazing trips up and down the wrong mountains and countless joyful moments.

Additionally, I want to thank my former flatmates who supported me during this journey: Christoph Semperowitsch, Florian Többen, and Julia Schymalla for welcoming me when I was new and without a home in a faraway city, and for all the wonderful feasts we shared. I would also like to thank Julian Lorenz and Alissa Müller for their company and support, which helped keep me sane during pandemics and lockdowns. Last but certainly not least, my heartfelt gratitude goes to Lisa Weis, Andreas Schmid, Fabian Schreiner, Kerstin Bauer, and Sarah Schink who have become more like family than merely flatmates. You all made our shared living space a home filled with laughter and support. I gladly look forward to both reminiscing about our past adventures and to the many future trips we will embark on together.

Additionally, I want to express my gratitude to Joscha Stelljes, who not only supported me during my studies but also discovered the job opportunity that led me to this thesis. His role has been instrumental in my academic journey, and I am profoundly thankful for his continued friendship.

Pursuing the long-term goal that this thesis represents also necessitates emotional support, for which I am profoundly thankful to my family. My deepest gratitude goes to my parents, Theo and Angelika, for their constant support and belief in my abilities; and to my brother Denis, who has always looked out for me and helped me better myself.

Abstract (English Version)

The task of autonomous navigation is an omnipresent topic in robotic applications. Autonomous mobile robots have gained immense popularity, especially in controlled, structured environments such as warehouses. Recent advances in mobile robotics have brought forth more complex walking robots that are able to traverse even challenging terrain with ease. Despite this, current navigation algorithms usually reduce the path-planning problem to a planar projection of the environment. This loss of dimensionality greatly diminishes autonomous robots' capabilities. It either restricts their movement capabilities or increases their risk of collision since not all of their operation space is considered. The presented research aims to alleviate this shortcoming by introducing a navigation framework that utilizes the entire three-dimensional search space. To enable a fast processing of the volumetric data, in this work the fast and memory efficient VDB-Mapping framework was developed. Its highly efficient data integration capabilities enable mobile robots to create high-resolution maps of large-scale environments in real-time during operation. Based on this detailed map representation, this thesis introduces efficient path-planning approaches to navigate robots autonomously through the full three-dimensional space.

One of the most crucial parts of every navigation framework is the underlying map representation on which it is based. This work proposes a novel volumetric mapping framework to build consistent maps of arbitrary environments. The framework is focused on fast and efficient data integration to handle various challenges during the operation in unknown areas. Due to this extremely fast data integration, new sensor data can be integrated into the map in real-time. Contrary to similar mapping frameworks, VDB-Mapping is able to keep up with current sensors' data rates. As a result, no sensor data has to be dropped during processing, resulting in overall more complete and accurate volumetric maps. The fast integration speed can be additionally leveraged by mobile robots to cope quickly with complex dynamic changes in the environment. Furthermore, it offers the capability to handle spatial displacements in large-scale environments caused by pose estimation errors. This is achieved by splitting the mapping process into a network of locally consistent sub-maps. It enables the mapping algorithm to re-align the individual sub-maps after pose estimation errors have been corrected. As a result overlapping map sections can be correctly aligned to generate a more accurate map representation. The last key feature of the proposed mapping framework is its capability to exchange partial or complete maps efficiently over bandwidth-restricted networks. As network bandwidth is often

strictly limited, the approach focuses on reducing the necessary memory footprint of exchanged maps as much as possible. This capability to exchange maps between different systems can be used in versatile applications, such as remote map streaming or collaborative mapping in multi-robot setups.

Based on this map representation, an efficient three-dimensional path-planning framework is developed. The thesis proposes multiple optimizations that enable especially mobile ground robots to generate paths through arbitrary, complex, three-dimensional environments. Despite the increased search space, computational practicability is ensured by introducing various adjustments to the basic path-planning algorithms. Even though the concepts were developed to work on graph-based planners such as A*, this thesis also investigates how the same concepts can be easily transferred to sampling-based approaches such as RRT. The developed skills to utilize the entire three-dimensional search space during autonomous navigation significantly increase the overall capabilities of mobile robots in unstructured and dynamically changing environments.

Abstract (German Version)

Autonome Navigation stellt eine allgegenwärtige Aufgabe im Anwendungsgebieten der mobilen Robotik dar. Vor allem in kontrollierten und strukturierten Umgebungen, wie zum Beispiel Lagerhäusern, haben Roboter mittlerweile einen signifikanten Zuwachs an Verbreitung zu verzeichnen. Aktuelle Fortschritte in mobilen Robotern haben eine Reihe komplexer Laufroboter hervorgebracht, die in der Lage sind selbst herausforderndes Gelände mühelos zu überwinden. Trotz dessen reduzieren aktuelle Navigationsalgorithmen das Pfadplanungsproblem größtenteils noch auf eine planare Projektion der Umgebung. Diese Dimension-Reduktion führt jedoch dazu, das komplexe mobile Roboter ihr volles Potenzial nicht ausschöpfen können da entweder ihre Bewegungsmöglichkeiten stark eingeschränkt werden oder zusätzliches Risiko für Beschädigungen in Kauf genommen werden da nicht der gesamte Arbeitsraum berücksichtigt wird. Um dieser Problematik entgegenzuwirken, behandelt diese Thesis die Entwicklung eines Navigationsframeworks, welches den gesamten drei-dimensionalen Suchraum während jedes Planungsschrittes in Betracht zieht. Diese Arbeit präsentiert die Entwicklung des schnellen und gleichzeitig speichereffizienten VDB-Mapping Kartierungsframeworks. Dessen extrem effiziente Datenintegration ermöglicht es mobilen Robotern während einer Operation hochauflösende Karten von weiträumigen, komplexen Arealen in Echtzeit zu generieren. Aufbauend auf dieser detaillierten, volumetrischen Karte werden in dieser Thesis zudem effiziente Pfadplanungsansätze vorgestellt, um Roboter autonom unter Berücksichtigung des vollen drei-dimensionalen Raumes zu navigieren.

Eine der wichtigsten Komponenten jedes Navigationsframeworks ist die Kartenrepräsentation auf der alle nachfolgenden Planungsalgorithmen aufbauen. In dieser Thesis wird hierfür ein effizientes, volumetrisches Kartierungsframework entwickelt welches es ermöglicht konsistente Karten von komplexen und dynamischen Umgebungen zu erstellen. Das Framework setzt einen starken Fokus auf die schnelle und effiziente Integration neuer Daten um die verschiedenen Herausforderungen die im Einsatz in unbekannten Gebieten lauern zu bewältigen. Aufgrund der extrem schnellen Datenintegration ist es möglich, neue Sensordaten in Echtzeit in die Kartenstruktur zu integrieren. Im Gegensatz zu ähnlichen Kartierungsframeworks, ist VDB-Mapping dazu in der Lage mit der Datenraten von heutigen Sensoren mitzuhalten. Dadurch ist es möglich, sämtliche Daten zu integrieren, ohne Sensormessungen zu verwerfen, wodurch eine insgesamt akkuratere Kartenrepräsentation erstellt wird. Dies erlaubt dem Roboter schnell in komplexen Umgebungen auf dynamische Veränderungen zu reagieren. Zudem

bietet es die Möglichkeit nachträglichen räumliche Verschiebungen der Kartendaten in weiträumigen Arealen auszugleichen. Dies wird erreicht, indem der Kartierungsprozess in einem Graphen kleiner, lokal konsistenter Kartenausschnitte aufgeteilt wird. Dies ermöglicht der Kartierung, die einzelnen Kartenausschnitte bei einer Posenoptimierung nachträglich neu zueinander auszurichten. Zusätzlich wird eine Methode präsentiert, Karten partiell oder im Gesamten effizient über ein beliebiges Netzwerk zwischen mehreren Systemen auszutauschen. Da die Netzwerkbandbreite jedoch in Einsatzgebieten häufig sehr begrenzt ist, wird ein starker Fokus auf die Reduzierung des benötigten Speicherbedarfs der ausgetauschten Karten gelegt. Basierend auf dem entwickelten Kartenformat wird zudem ein effizientes, drei-dimensionales Pfadplanungsframework entwickelt. In dieser Thesis werden hierzu mehrere Erweiterungen von traditionellen Pfadplanungsalgorithmen vorgestellt, um mobilen Robotern das Berechnen komplexer Pfade durch beliebige drei-dimensionale Umgebungen zu ermöglichen. Durch die entwickelten Optimierungen wird, selbst trotz der erhöhten Dimensionalität, die praktikable Laufzeit der Planungsalgorithmen erhalten. Obwohl die Konzepte zur Nutzung mit Graphbasierten Pfadplanungsalgorithmen wie zum Beispiel A* entwickelt wurden, zeigt diese Thesis das die gleichen Konzepte ebenso auf andere Planungsparadigmen wie zum Beispiel sampling-basierte Planer transferierbar sind. Somit können die in dieser Arbeit entwickelten Ansätze den gesamten drei-dimensionalen Suchraum zur autonomen Navigation zu nutzen. Dadurch kann die allgemeine Autonomie von mobilen Robotern in komplexen und dynamischen Umgebungen signifikant gesteigert werden.

Contents

Acronyms

AGV Automated Guided Vehicle. 1, 4, 56, 92, 117, 119, 126, 165

AUV Autonomous Underwater Vehicle. 1, 56, 123, 126, 160, 161

BFS Breadth First Search. 29

CGI Computer Generated Images. 35

CPU Central Processing Unit. 55

CSG Constructive Solid Geometry. 35, 41

DB-Grid Dynamic Blocked Grid. 35

DDA Digital Differential Analyzer. 67, 119, 145

DFS Depth First Search. 96

EKF Extended Kalman Filter. 13

ESDF Euclidean Signed Distance Field. 55

GPS Global Positioning System. 12

GPU Graphics Processing Unit. 54, 55, 166

HDDA Hierarchical Digital Differential Analyzer. 44, 45, 145

IPS Indoor Positioning System. 12

MAV Micro Aerial Vehicle. 55

MLS Multi-Level-Surface. 53

NDT Normal Distribution Transform. 56

OMPL Open Motion Planning Library. 111, 148, 159

PRM Probabilistic Roadmap. 30, 31, 33, 34, 58, 59, 111

ROS Robot Operating System. 127

RRT Rapidly-Exploring Random Tree. 30, 33, 34, 59, 60, 111, 113

SDF Signed Distance Field. 19, 55

SLAM Simultaneous Localization and Mapping. 8, 11–15, 49, 55, 56, 75

STM Stochastic Triangular Mesh. 19

TSDF Truncated Signed Distance Field. 55

UAV Unmanned Aerial Vehicle. 1, 4, 53, 56, 63, 82, 86, 93, 104, 118, 119, 123, 126, 143, 160, 161

UKF Unscented Kalman Filter. 13

1 Introduction

Robotic systems are becoming increasingly prevalent and accessible across various sectors of society. Mobile robots, in particular, have seen a significant increase in popularity, both in public and industrial applications. By now, they are involved in a growing number of real-world scenarios. For example, automated warehouses are becoming increasingly common, where fleets of simple Automated Guided Vehicles (AGVs) transport the goods. Due to the increased demand and the resulting affordability, these types of robots have also found their way into typical households in the form of vacuum cleaning robots. Furthermore, simple robots are already used in education to teach the next generation the concepts of computer science. Additionally, this will decrease preconceptions and raise society's general acceptance of robots.

To increase their usefulness while performing tasks in everyday life without further assistance from a human supervisor, the current main focus of robotics is set towards enhancing the autonomy of robots. One of the main capabilities of mobile robots required to perform these tasks is to navigate safely from one point of the environment to another. In simple, structured environments, tremendous progress has been made toward truly autonomous navigation of mobile robot systems.

Not only AGVs but also the fields of Unmanned Aerial Vehicles (UAVs) and Autonomous Underwater Vehicles (AUVs) are gaining in popularity. Privately used, remote-controlled drones have become so popular that multiple governments have even started to regulate their use to ensure the safety and privacy of their populations. Furthermore, plans to utilize them for efficient automated parcel delivery are being actively researched. On the other hand, AUVs are valuable for inspection tasks in hostile environments, such as monitoring the structural integrity of struts in deep-sea offshore oil rigs. Due to their increased motion prowess, and more specifically, their ability to move freely in the three-dimensional space, more complex navigation capabilities are required in these scenarios.

For flying and diving robots fully capable of three-dimensional motion, these advanced navigation skills are crucial to further their autonomous capabilities. Nevertheless, the field of mobile ground robots would also greatly benefit from utilizing the entire three-dimensional space. Recently, complex walking machines such as Spot from Boston Dynamics or ANYmal [44] of ANYbotics have transitioned from prototype research robots to product-level ones. Since they can freely traverse over rubble and complex terrains, the restrictions presented in a planar setting would diminish their autonomous abilities. However, they should be

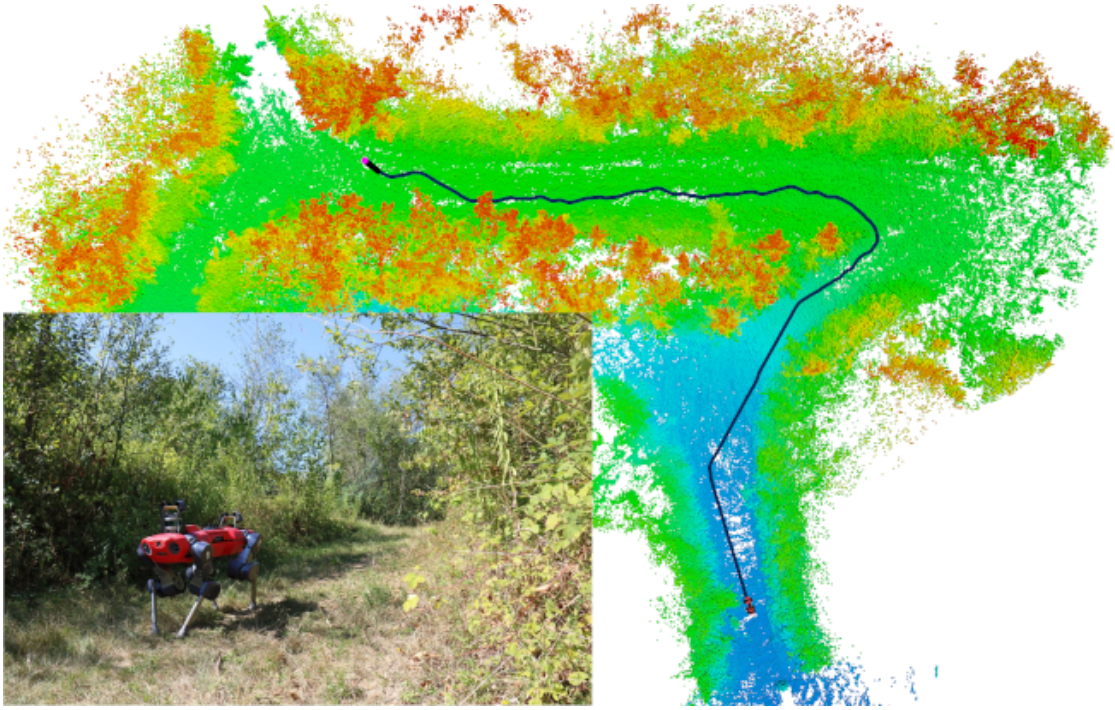


Figure 1.1: Autonomous navigation through a search and rescue facility of the THW in AAchern. An ANYmal-C [44] equipped with a Velodyne VLP-16 [45] and an array of RGB-D sensors was used to create the map. The area covered an area of around $62\,\mathrm{m}$ x $53\,\mathrm{m}$ x $8\,\mathrm{m}$. The blue path took the robot through highly overgrown areas and multiple inclines on a path of around $55\,\mathrm{m}$.[Grosse Besselmann et al., 2024]

able to understand their current environment better in order to plan and act accordingly. Thus, a higher focus should be placed on their navigation capabilities to enable them to perform to their fullest extent. Advancing their navigational prowess and the ability to perceive the three-dimensional world would therefore result in higher autonomy and increased safety during operation.

1.1 Problem Statement and Research Questions

Due to the increasing popularity and general acceptance, the number of mobile robots will increase further in the future. To keep the operator-to-robot ratio scalable, further effort should be put into making robots more autonomous. The three critical components depicted in Figure 1.2 need to be solved to enable them to operate in unknown and challenging terrain.

Mapping: The multitude of areas of applications paired with often unknown areas provides a particular challenge for mobile robots. The robot requires a deeper understanding of its environment to achieve an autonomous operation. This knowledge might be known beforehand in simple structured environments

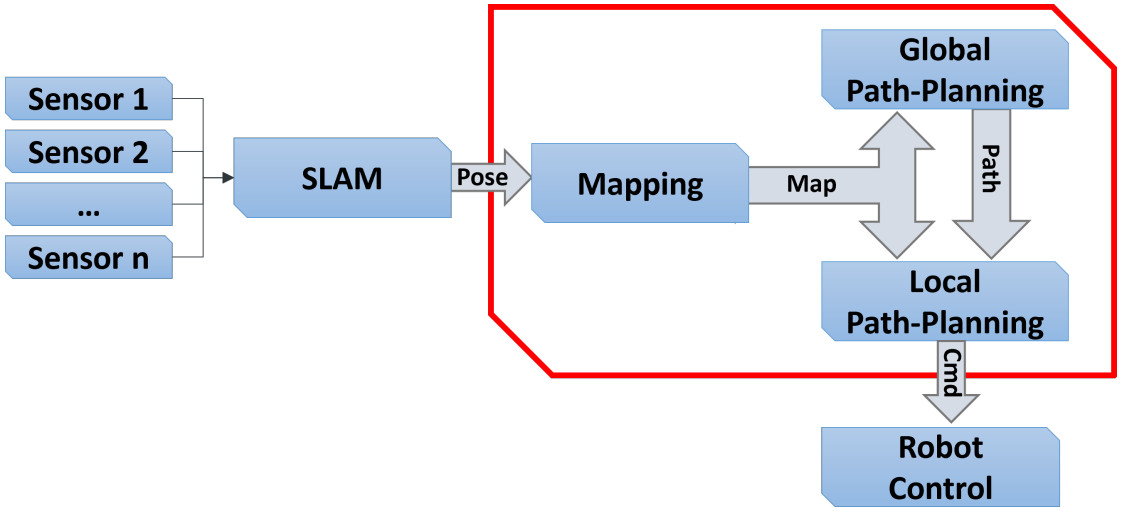


Figure 1.2: Structural architecture diagram of the topics covered in this thesis. The work is divided into three major parts. First, a volumetric mapping framework is presented. Based on this map representation, the following chapters present global and local path-planning concepts that utilize the full three-dimensional space.

like storage facilities. However, problems arise when the robot faces an unknown, complex environment. To enable autonomous navigation, it is first necessary to generate a map of the present environment using the robot's sensor setup. Given the extension in movement capabilities of current mobile robots, a higher dimensional volumetric representation is required to provide a better understanding of the area of operation.

Path Planning: Path planning is the most crucial part of autonomous robot navigation. Due to historical reasons, this task is commonly reduced to a two-dimensional space. More complex navigation was unnecessary since early robots were mostly restricted to planar surfaces. Furthermore, they lacked the computational power to perform the path planning tasks on a higher dimensional level. However, the desire for more complex path-planning algorithms increases as robots become more sophisticated and powerful. This way, the robot can take advantage of the full range of its motion capabilities.

Collision-free Path Execution: Planning a path from a start to a goal position is only the first step in a successful navigation task. Subsequently, the robot has to perform a collision-free execution of the calculated path. Depending on its movement capabilities, here as well a projection onto a simple planar surface would impede the performance of the robot. Instead, observing the entire three-dimensional space enables a mobile robot to prevent collisions with more complex obstacles that cannot be safely projected onto their two-dimensional footprint.

Research Goal. This work aims to develop a holistic navigation framework utilizing the entire three-dimensional search space. The navigation framework should be independent of the robot's specifications, kinematics, and sensor setup. It should provide the means to ultimately build up the autonomous navigation capabilities of any mobile robot in unknown and complex environments. The developed algorithms should be easily adaptable and exchangeable to deploy them on a large range of autonomous robots.

The following research questions can define the challenges derived from this research goal:

Research Question 1. How can the world be efficiently transformed into an accurate environmental representation suitable for robot navigation in the full volumetric space?

Research Question 2. How can ground-based mobile robots autonomously calculate efficient and collision-free paths through challenging and complex terrain?

Research Question 3. How can mobile robots efficiently traverse arbitrary three-dimensional paths through unstructured, dynamic areas without harming people, the environment, or themselves?

1.2 Concept Overview

This thesis proposes a holistic framework for autonomously navigating robots through complex environments. Contrary to available approaches, in this work, the entire three-dimensional space is considered and utilized in all components. This enables mobile robots to perform to the fullest extent of their movement capabilities. The framework is designed to be quickly adaptable to different robot models to prevent an over-specialized solution. As different robots possess different computational capabilities and sensor setups, they often apply different approaches to localize themselves within the world. For example, AGVs often rely on relatively heavy LIDAR technology to estimate their locations in space. In contrast, UAVs are bound to crucial weight restrictions to fly efficiently and thus rely on lighter sensors such as cameras. Therefore, from a program flow perspective, the developed navigation framework is located after the robot's pose estimation to achieve independence from the localization approach. This way, the localization systems used can be easily exchanged without affecting the performance of the navigation framework. On the other hand, from a low-level point

of view, each robot is controlled differently. Complex walking machines, for example, are required to meticulously plan each of their leg trajectories. At the same time, UAVs must account for parameters, such as wind strength, while controlling their rotors. The presented framework uses a generalized control interface to abstract this highly complex level of control to remain usable for a more extensive range of robot types. This is realized as a three-dimensional twist interface. This interface is widely applied in robotics and specifies the desired linear and angular velocity at which the robot should move. As a result, the developed concepts can be used transparently on a range of heterogeneous robotic systems. An overview of the entire system architecture, how the components interact with each other, and its delimitation is presented in Figure 1.2.

The concept developed in this thesis can be divided into three main components: mapping, global planning, and local path execution. The core part of the entire navigation stack is the efficient three-dimensional mapping structure. Its goal is to generate a detailed volumetric representation of the robot's surrounding environment, which can be subsequently used to perform various navigational tasks. In order to create this representation, the output of the robot's sensor readings is accumulated within a highly efficient tree structure. As additional input, the robot receives the pose estimation of an external localization system. This is necessary since all sensor data is initially represented in its respective reference system. Therefore, each reading must first be transformed into a common global reference system to combine them. Based on the generated volumetric map representation, the robot should be able to traverse from its current pose to an arbitrary target pose. This process is further divided into two steps.

First, a global path planning algorithm calculates a viable, collision-free path from the start toward the goal pose. To achieve this, information about the structure of the environment and present obstacles encoded in the aforementioned map are utilized. Contrary to common approaches, which reduce the space to a two-dimensional plane, here, the full three-dimensional space is considered and utilized. This enables the robot to plan and traverse even through complex environments such as multi-level buildings and rough terrain.

However, the map does not always cover all dynamic objects, such as moving people or structure changes like opening doors. Furthermore, parts of the map the robot has not visited for a while might be outdated. Thus, strictly speaking, the path is only semi-collision-free. Therefore, the resulting path is subsequently passed down to the local path-planning process. Its primary purpose is to execute the previously calculated path and avoid all previously unconsidered obstacles during its traversal. Instead of using the entire, possibly huge, map for the planning process, here only a small local sub-area is used. This comes with the advantage that multiple sensors cover the robot's adjacent space. Therefore, local execution can integrate current sensor data into its planning process. Furthermore, by restricting the considered area to a local subspace, the performance and reactiveness of the system can be drastically increased. In this case, the three-dimensional map can be utilized to provide improved and more sophisticated

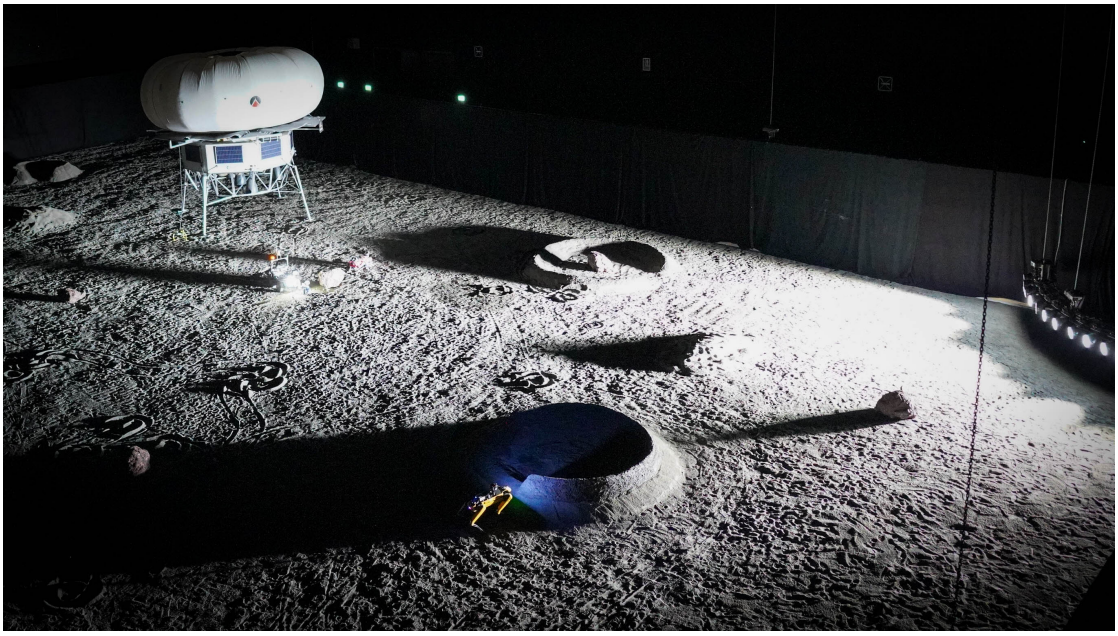


Figure 1.3: Image of a field test recorded during the ESA/ESRIC Space Resources Challenge [20]. The goal was to search for valuable resources on the lunar surface autonomously. A team of three robots controlled from a remote base station used the presented navigation framework during a five-hour mission.

capabilities to avoid volumetric obstacles.

The concepts developed in this thesis have been deployed and evaluated on multiple robot systems. The applied robots range from simple planar differential drive platforms over off-road wheel-based vehicles to complex walking robots. Even though the concepts were initially developed within safe laboratory environments, they have been tested in multiple field tests, facing challenging, rough terrain. It was evaluated during the ESA/ESRIC Space Resource Challenge [20], in a realistic replica of the lunar surface depicted in Figure 1.3 Here, multiple robots had to navigate through craters and between spacecrafts to find rare resources on a Moon-like surface. Furthermore, the concepts were evaluated and improved during an analog mockup of a Mars mission in the Tabernas Desert in southern Spain. There, the framework was applied to multiple heterogeneous robots to overcome challenging, uneven terrain, rock formations, and dried-out river beds. These real-world field tests of the presented approaches have demonstrated their ability to efficiently handle challenging terrain while simultaneously avoiding collisions with arbitrary, unforeseen obstacles.

1.3 Scientific Contribution

Within this thesis, a holistic framework for navigating mobile robots through complex terrain is proposed. The developed concept leads to an increased autonomy, which will enhance the ability of mobile robots to efficiently perform tasks in arbitrary environments. The primary contributions of this research are:

- A volumetric mapping framework to create and store environmental data of the robots' surrounding area. The framework puts considerable focus on the efficient integration of new sensor data. By decreasing the necessary runtime, new information can be directly used in the subsequent navigation framework, leading to a decreased reaction time for the robot. The value of each voxel in the hierarchical tree structure represents the occupancy state of the corresponding area in the environment. Additionally, to this basic state, it is possible to store further information within each voxel individually, thus increasing the robots' knowledge and understanding of the world. The mapping framework also provides various interfaces, such as handling time-variant data in large-scale environments and efficient map exchange between different systems in a multi-robot scenario.

- An approach for planning three-dimensional, collision-free paths through complex unstructured environments. Due to the additional search space dimension, the complexity of volumetric path planning increases significantly. Multiple optimizations to the path-planning process on volumetric maps have been developed to counter this issue and provide an efficient runtime. To ensure the safe traversal of the robot and the ability to compute a collision-free path, extensive, specialized ground and collision checks have been developed and integrated.

- A local path planning component that considers the entire three-dimensional space during its execution. The concept includes a highly configurable navigation pipeline, which can be easily adapted to support different robot types and kinematics. Despite the already integrated surface checks of the global path planning process, local dynamic obstacles are often not considered. Therefore, an extensive obstacle avoidance system is integrated into the local navigation pipeline. By considering the entire volumetric space, the safety of the robot and its surrounding environment can be assured, even in complex terrain.

Different analyses and experiments in various scenarios support the validation of the proposed concept. The contribution shall widen the distribution of three-dimensional navigation approaches for mobile robots. This will further enable mobile robots to reach previously unavailable locations, thus significantly increasing their application area. Due to increased autonomy, mobile robots can become more common in areas where human supervision is considerably more challenging. This includes use cases such as planetary exploration, where remote

operators face difficulties due to an increasing communication delay. Furthermore, it becomes easier to perform tasks in hostile environments like contaminated areas where communication is impeded due to severe radiation. In order to provide access to the developed concepts to a wider research community, individual parts have already been released as open-source software [1]. It is also intended to continue to release missing components to spread the developed algorithms to a broader community, thus extending the availability of freely available state-of-the-art research.

1.4 Thesis Outline

The remainder of this work is structured as follows: **Chapter 2** provides the theoretical background to the topics discussed in this thesis. It covers the foundations of robot mapping and relevant topics. First, an introduction to robot localization and different SLAM approaches is given. Afterward, various possible representations used to store maps are presented. Subsequently, an introduction to multiple path planning concepts, such as graph-based and sampling-based planners, is provided. The chapter finishes with an overview of the utilized OpenVDB framework. Here, the general concepts and how the properties of the data structure are leveraged in the developed navigation framework are highlighted.

Chapter 3 reviews related work relevant for this thesis. The chapter is focused on the closely related volumetric mapping and path planning approaches.

Chapter 4 introduces the developed volumetric mapping approach. First, a general overview of the proposed VDB-Mapping framework and how it efficiently integrates new data into the map is given. Subsequently, the various developed extensions and improvements to the mapping framework are introduced. The Sub-Mapping in Section 4.2 provides a specialized framework based on small, locally consistent sub-maps, which helps to prevent spatial displacements in large-scale environments. The Remote-Mapping module described in Section 4.3 is utilized to exchange maps between multiple agents in the network. This way, a collaborative mapping between individual systems becomes possible. Lastly, the Map-Augmentations introduced in Section 4.4 enables the robot to store arbitrary additional data in the map. This can be used to enrich all available map data to help the robot in its decision process during subsequent tasks.

Chapter 5 details the developed three-dimensional path planning approach. The chapter is divided into two main parts. First, in Section 5.1, global path planning, which provides a rough path from start to end goal, is introduced. Section 5.2 covers the developed local path planning pipeline used to execute the previously calculated global path.

[1] https://github.com/fzi-forschungszentrum-informatik/vdb_mapping
https://github.com/fzi-forschungszentrum-informatik/vdb_mapping_ros
https://github.com/fzi-forschungszentrum-informatik/vdb_mapping_ros2

Chapter 6 provides an in-depth evaluation of all the concepts proposed in this thesis. The performance of the proposed mapping framework is shown by extensive experiments using both artificial benchmarks and applications on real-world data. Based on these maps, the performance of the developed three-dimensional path planning approaches, compared to several state-of-the-art planners, is given. The section concludes with a discussion about the achieved results.

Chapter 7 concludes with a summary of this thesis. It reviews the initial research questions and highlights the core contributions of this work. Furthermore, it sheds light on open aspects and future research questions.

2 Foundations of Robot Navigation

The task of autonomous navigation for mobile robots requires multiple interconnected systems. This chapter introduces the foundation of robot navigation, on which this work was built.

The concepts used here can be roughly divided into three aspects. In order to operate in any given environment, a robot needs to sense its own internal state, its surroundings and act based on this information. Thus, the first aspect covered in this chapter is a general introduction to automated robot mapping. Furthermore, crucial approaches and concepts are covered to build, represent, and operate on map structures. Based on these maps, the goal of navigation is to support finding and executing viable paths from its location to a goal position. Therefore, path planning concepts and their respective subclasses are introduced. A crucial component in most aspects of this work is the underlying efficient OpenVDB data structure. It functions as a backend of the mapping framework, storing all necessary information to efficiently encode the present environment into the respective volumetric map format. Therefore, this chapter concludes with a detailed description of the data structure's concepts. Furthermore, an insight into the details of the implementation and how they are leveraged in the context of mobile robot navigation is given.

2.1 Autonomous Robot Mapping

Since ancient times, the art of cartography has been a crucial part of the navigation process. For example, in the context of seafaring, maps were an integral tool to plan safe journeys and to avoid shallow waters or other possibly lethal obstacles. Similar requirements are necessary for mobile robots to operate autonomously in previously known or unknown environments. This section introduces the fundamentals of automated robot mapping and corresponding topics. One of the core problems for autonomous robots is the task of generating an accurate map while simultaneously estimating its position within this environment. This problem is commonly referred to as the Simultaneous Localization and Mapping or short SLAM problem. Section 2.1.1 introduces the problem and gives an overview of various solutions to solve it. Given the possibility of estimating maps of an environment, the next crucial topic covered in this section is in which data representation of these maps can be stored. Section 2.1.2 briefly overviews commonly used mapping concepts and representations. A deeper focus is set on

geometric representation and even more specific grid-based approaches, which will be discussed in depth throughout the remainder of this work.

2.1.1 SLAM

One of the core problems during the mapping process is to align each acquired sensor measurement in a spatial relation to each other. Since the sensor itself commonly has no knowledge about its current position, the gathered data is usually represented in the sensor's local coordinate system. Although the sensor can be referenced in correlation to the robot coordinates system, mobile robots also move through the environment, resulting in a dynamically moving set of coordinate systems. Therefore, the robot requires an accurate localization within its current environment. This problem is tightly coupled to the mapping process and provides a kind of *chicken-egg-problem*. In classical robot localization algorithms like particle filter localization (also known as Monte Carlo localization) [14], the robot requires an accurate map to estimate its position. Conversely, the robot requires the position and orientation of the robot relative to the global reference frame to generate an accurate map from acquired sensor data. Therefore, a good localization of the robot is indispensable. This circular problem could be avoided by using external referencing systems such as a Global Positioning System (GPS) or an Indoor Positioning System (IPS). However, these systems are lacking in accuracy and are only suitable in certain situations. GPS, for example, requires a line of sight to multiple positioning satellites, which is not given in indoor scenarios. On the other hand, indoor position systems require a fixed setup of the workspace, making them infeasible in unknown environments. Alternatively, the robot could utilize blueprints of the buildings to generate an initial map of the area. However, most of the time, these plans are not available or lack the necessary accuracy. Furthermore, structural changes are often not covered, leading to an imprecise localization of the entire system. To alleviate these problems and restrictions, in current mobile robotic systems, so-called Simultaneous Localization and Mapping (SLAM) algorithms have been developed. This type of algorithm aims to tightly couple the pose estimation and mapping process of the robot.

Probabilistic Formulation

Smith, Self, and Cheeseman [112][111] pioneered the basic concept of the SLAM problem in 1986. They focused their work on estimating the position and orientation of obstacles relative to the robot. In 1991, Leonard and Durrant-Whyte [64] proposed its first formal definition as a probabilistic function. In order to solve the SLAM problem, it is necessary to estimate the robot's trajectory and the state of the map while the robot moves throughout the environment. This is achieved by processing the sensor measurements captured by the robot. However, due to the inherent sensor noise of these measurements, the SLAM problem is commonly defined as a probabilistic formulation. It is assumed that no knowledge

about the environment is available a priori. In this formulation, the trajectory the robot traverses is defined by a sequence of random variables $x_{0:T} = x_0, \ldots, x_T$, where T denotes the last point in time. The movement between two random variables is represented by the robot controls $u_{1:T} = u_1, \ldots, u_T$. Commonly, these estimates of the robot's movement are obtained using odometry (e.g., visual odometry or rotary encoders), which tracks the movement of the robot from one time step to the next. Additionally, while the robot moves along the trajectory $x_{0:T}$, it obtains a set of sensor measurements $z_{1:T} = z_1, \ldots, z_T$ of the environment. For the probabilistic SLAM problem, the task is now to estimate the posterior probability of the robot trajectory represented by the random variables $x_{0:T}$ as well as the state of the map m, given the set of all obtained measurements $z_{1:T}$ and controls $u_{1:T}$. The posterior probability for the Full-SLAM problem is given by:

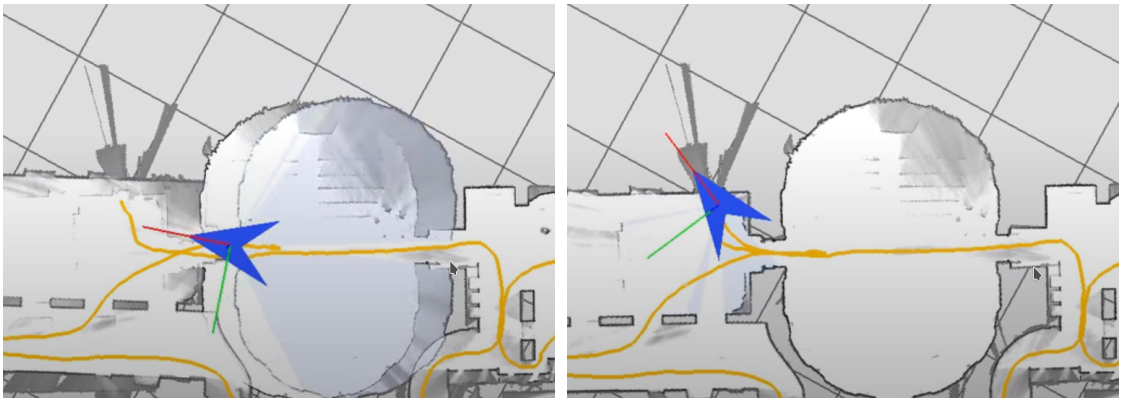
$$p(x_{0:T}, m | z_{1:T}, u_{1:T}) \tag{2.1}$$

Contrary to the Full-SLAM problem, in online SLAM, only the current position is estimated using all previously available data. This posterior probability is represented by:

$$p(x_t, m | z_{1:t}, u_{1:t}) \tag{2.2}$$

where t denotes the current time stamp. However, estimating this posterior requires operating in a high-dimensional space. In order to decrease the complexity of this problem, two critical assumptions are commonly made in SLAM. First is the static world assumption, which, as the name states, assumes that observation of the environment remains constant. More specifically, this means that all observed objects and obstacles remain in the same position. The second utilized assumption is the Markov assumption [74], which was named after the Russian mathematician Andrey Markov and is an integral idea in probabilistic models and Markov decision processes. It says that the future state of a process solely relies on the most recent state, ignoring all states that lead to the current one. Even though both assumptions might not accurately represent all real-world effects of the process, they still significantly reduce the complexity of this high-dimensional problem. However, despite their flaws, these two assumptions made the first approaches to solving the SLAM problem possible.

The first practical implementation of a SLAM system solved the problem by applying an Extended Kalman Filter (EKF) to estimate the posterior probability described in Equation (2.1). Wan and van der Merwe [126] proposed the exchange of the EKF for an Unscented Kalman Filter (UKF) to improve the handling of highly non-linear systems. In order to further improve the efficiency and accuracy of the pose and map estimation, Murphy et al. [80] proposed a grid-based approach based on Rao-Blackwellized particle filters. Subsequently, Montemerlo et al. [78] proposed FastSLAM, which combined the Rao-Blackwellized particle filters with a tree structure for efficient map building.



(a) Before loop closure (b) After loop closure

Figure 2.1: Due to inaccuracies in the robot pose estimation process, the map slowly drifts. If a previously observed location is re-visited, a SLAM algorithm can re-adjust the map using so-called loop closure constraints. The same area is re-entered in the first image, but the map sections do not overlap exactly. After a short period, a loop closure is triggered, resulting in the right, re-aligned map.

Graph SLAM

In 1997, Lu and Milios proposed a different approach. The idea was to exchange the probabilistic formulation in favor of a graph representation[72]. This graph-based SLAM provided a method to construct and represent the SLAM problem in a more graspable representation. In this approach, each node represents a robot pose from the trajectory or an observed landmark. The edges, on the other hand, represent constraints between the corresponding nodes. These constraints are derived from observations of landmarks or the estimated control movements between two robot poses.

The main intention of the graph-SLAM approach is to find the spatial configuration of all nodes that best satisfies all available constraints. This is achieved by continuously optimizing the spatial location of all nodes within the graph. During the adjustment of all poses and landmarks, the overall error is minimized, thus providing increased accuracy and global consistency compared to the local update performed in EKF and particle filter approaches. As a result, graph-based SLAM methods inherently focus on global consistency. Furthermore, the graph structure can be easily adapted for dynamic environments. Changes in the environment can be handled by updating the respective nodes in the graph structure.

One of the main problems in SLAM is handling the so-called loop closure appropriately. In this instance, a loop closure specifies the moment the robot returns to a previously observed area. Using only incremental updates like odometry, for example, the pose estimate will drift over time, resulting in a distorted, non-consistent map as seen in Figure 2.1. To overcome this problem, the SLAM al-

gorithm has to detect such loops and re-evaluate all previous state estimations accordingly. In the context of graph-SLAM, this issue is addressed using special loop-closure constraints that connect pose estimates, similar to the movement constraints. However, compared to movement constraints, the loop closure constraints possess an inherently higher weight during the graph optimization process. As a result of the higher weight, the loop closure constraints are able to re-adjust large portions of the graph to reduce drift and thus improve the spatial map consistency. In 1999, Gutman and Konolige [33] first proposed an efficient way to handle the construction of such a graph. The additional loop closure constraints were detected while the robot pose was incrementally estimated.

The overall optimization problem of such graph structure presented a considerable challenge regarding the required computational complexity. Even though multiple optimizations were presented, such as graph relaxation of Howard et al. [41], or multi-level relaxation of Frese et al. [27], the graph SLAM formulation struggled for years to become a viable option compared to filtering approaches. However, in 2006, Dellaert and Kaess [15] first proposed exploiting sparse matrix factorization to solve the linearized problem presented in offline graph-SLAM. This approach was further built upon in iSAM proposed by Kaess et al. [46], which provides an online version of the basic algorithm. To achieve this, the algorithm leverages partial re-ordering to generate a sparse matrix factorization. With the increasing number of implementations, multiple insights into the graph-SLAM approach led to a restructuring of the problem. As a result, the optimized information matrix in graph-SLAM, which is the inverse of the covariance matrix, is always sparse (e.g., most entries are zero). This sparsity can be further exploited to significantly reduce the problem's computational complexity. Thus, the optimization problem is less memory-consuming and, in turn, more efficient. Building upon these insights, current approaches are able to utilize efficient graph optimization. Efficient non-linear least square optimizations such as Gauss-Newton or Levenberg-Marquardt [65][75] can be utilized to gain real-time graph-SLAM capabilities.

Furthermore, the sparse representation also increased the scalability of the graph-SLAM compared to more classical filtering approaches. This is especially the case when facing a large environment with many observations. The computational complexity of EKF-based methods, for example, scales quadratically with the number of poses and landmarks. Compared to this for sparse graph-SLAM systems, efficient solvers can achieve near-linear complexity for each iteration.

Additionally, to these advantages, the graph-SLAM formulation presents more flexibility when defining the problem and its constraints. The spatial constraints between different robot poses and landmarks represented by the edges of the graph allow the modeling of multiple types of constraints. This includes various types of measurements and uncertainties, including non-Gaussian and non-linear constraints. As a result, the constraints can be used to perform a multi-sensor fusion by integrating constraints for each individual sensor measurement to the same connected node pair.

2.1.2 Map Representations

The ability to generate a consistent map of the environment while simultaneously localizing within this map allows mobile robots to gain a detailed understanding of the area in which they operate. Thus, it expands the robot's capabilities to plan and make decisions based on this understanding. However, the representation used for the map to gather all data is equally crucial. An accurate and simultaneously efficient model of the environment is paramount during the successful operation of autonomous mobile robots. This is even more true while facing complex, challenging, and continuously changing environments.

The representations of the gathered data serves multiple crucial purposes during operations. It enables the robot or a remote operator to understand the present environment better and make decisions based on it. Furthermore, it allows an operator discover and identify unknown areas. This knowledge can be subsequently used to explore new areas in a specific direction to gain a better understanding of the environment. Most importantly, it enables a robot to plan viable trajectories between arbitrary positions within the map representation. Therefore, a sophisticated map representation presents a critical element of every mobile robot system and significantly influences its autonomous capabilities.

To achieve this, a viable map representation has to be able to handle the following aspects:

- **Dynamic Environments:** Since the environment in which robots operate is often subject to continuous change, the map representation must be able to handle dynamic changes. These dynamic changes can have multiple reasons. In multiple instances, the structural properties of an environment are not fixed permanently. This could include objects like opening or closing doors or repositioned shelves in an industrial warehouse context. Additionally, moving objects, such as other robots or people traversing the area, must be covered during mapping. The ability to handle dynamic objects significantly influences the mapping process's performance. To still satisfy the requirements of an ever-changing environment, the map representation should be able to integrate and update new data quickly into its data structure. This way, the map can be updated in real-time, allowing dynamic changes in the environment during operations such as navigation tasks to be regarded.

- **Data Access:** Most operations performed on a map require access to the stored data. Therefore, the reading access for the map is paramount for the efficiency of all subsequent operations it is used for.

- **Sensor Noise and Errors:** Current sensors are still prone to inherent sensor noise. Additionally, errors during the pose estimation introduce an additional error for the relative position of each measurement. Thus, the representation must be robust against minor deviations in pose estimates.

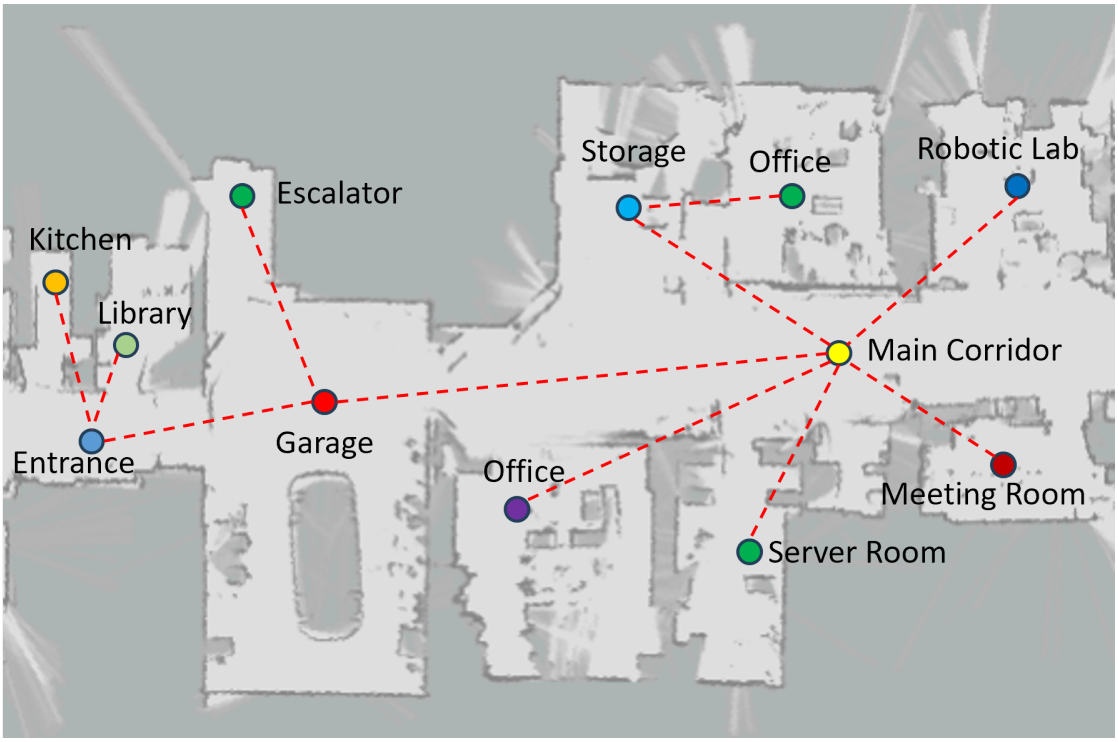


Figure 2.2: Map representation of a laboratory environment. The underlying gray and black areas represent the geometric map generated from LIDAR data. The graph overlay, on the other hand, represents a semantic map with annotations about important locations within the map.

- **Computational Resources:** In general, the map representation should be able to store the gathered data efficiently with respect to their memory size. This is especially true while mapping large, complex environments, which could result in huge maps. Thus, a balance between the desired grade of detail and the manageability of the data has to be considered.

The field of mobile robot mapping can be roughly divided into topological and metric maps of the environment. Additionally, metric maps can be further distinguished into continuous and discrete mapping approaches. While topological maps describe the environment in a context-specific way, such as landmarks, metric maps store the geometric properties of the environment. Figure 2.2 shows an exemplary map of a laboratory environment. The background depicts the geometric map of the environment. The graph overlay, on the other hand, represents the topological map.

Topological Map Representations

Topological maps offer a simplified yet powerful type of representation. This is especially the case when precise geometric details are not strictly necessary. Instead of a detailed representation of the physical and geometric features, the en-

vironment is encoded by its connectivity and layout. One of the most established representations are Voronoi Diagrams [125]. Here, the space is divided into regions based on the distance between a set of predefined points. The nodes of the graph represent the central points of these regions. Edges, on the other hand, are formed where different regions meet. As a result, the edges represent a path that maximizes the distance from all obstacles.

Another commonly used form of topological maps are landmark-based maps. Landmark-based maps model their environment as abstract graph-like maps, in which the nodes represent significant locations or landmarks such as rooms or corridors. The edges, on the other hand, encode the connectivity and relationship between these significant locations and denote the relative path between the landmarks. This concept is further improved in hierarchical topological maps, where the environment is represented as a multi-level graph. It combines multiple layers of individual topological maps. Each of those layers corresponds to a different resolution of the environment. The higher levels represent, for example, large regions such as buildings. The lower levels, on the other hand, provide a more detailed description of these regions.

Due to their high abstraction level, topological maps have the advantage of requiring a relatively low memory footprint, even in large-scale environments. Furthermore, due to their high efficiency, they are especially useful for fast path calculations on a high-level planning level [119]. In order to achieve this lower complexity, topological maps sacrifice the concept of environment-specific data, such as the geometrical features of an environment. However, this characteristic is often essential in more complex and changing environments [67]. As a result, this type of map becomes cumbersome and difficult to handle in the context of low-level motion planning. Instead, they are effective for tasks such as high-level mission planning, where an overall plan to achieve a multistep task is required. A more detailed low-level path planner can subsequently handle the explicit execution of such a mission plan.

Continuous Metric Mapping

Metric maps are often fundamental while controlling mobile robots. They provide a detailed geometrical representation of the environment, including the precise coordinates of objects and obstacles. As a result, a robot can obtain exact spatial knowledge to efficiently and safely perform tasks in the actual environment. This can include localization, path planning, or obstacle avoidance tasks. The class of continuous map representations is especially useful for tasks that require a highly detailed understanding of the environment. They offer a precise description of the environment at a high, non-discrete resolution. These types of maps are typically helpful for advanced navigation or high-precision manipulation tasks.

The most intuitive representations are point cloud maps. They aim to represent the environment using an accumulated collection of three-dimensional points in space. Usually, these maps are generated with the help of 3D ranging sensors such as LIDAR or depth cameras. In addition to their coordinates in space, each point is able to store additional data, such as the color of the corresponding pixel or the intensity of the LIDAR beam's signal reflection. The PCL library proposed by Rusu and Cousins [105] offers an efficient framework that can store arbitrary templated point types in its data structure. Contrary to other map formats, point cloud maps do not require any pre-processing, such as grid discretization. Instead, they store the unfiltered continuous data, which provides a high level of detail and precision. However, this precision is computationally expensive. Due to the large amount of 3D points, the map structure quickly requires a huge memory footprint. Furthermore, the large quantities of data are often not manageable during subsequent operations in larger environments. Thus, this type of map representation is suitable to be used on a local, restrictive scale.

An efficient way to represent continuous environmental data are triangulated mesh maps. In this kind of representation, surfaces of objects detected by the robot are encoded through simplified geometric shapes, like polygons. By representing the environment as a network of connected triangles, the efficiency of the representation can be greatly optimized [67]. The level of detail of the resulting continuous map surfaces can subsequently be dynamically changed by adjusting the number of polygon vertices. This spatial modeling approach is widely applied in computer graphics. However, even though they present an efficient data structure, they often fail to explicitly model uncertainties in the data, thus making them prone to sensor noise. This issue is addressed in Stochastic Triangular Mesh (STM) [69] maps, which offer the possibility to deal with uncertainties in continuous mapping. They represent the environment using a set of collections of stochastic processes and can manage uncertainties in measurements and robot poses through incremental map updates.

Even though they can be seen as continuous, the Signed Distance Field (SDF) maps have more in common with grid-based discrete map structures. Here, each grid cell stores the Euclidean distance from the nearest surface. Reasoning from this, a highly detailed map of the environment can be reconstructed in real time. However, this type of map also does not handle probabilistic properties. As a result, they can not model the statistical dependencies between mapped elements, rendering them impractical in highly dynamic environments.

Discrete Metric Mapping

The second type of metric map representation are discrete or grid-based maps. Contrary to continuous metric mapping, they provide a much more manageable amount of data. As the name states, they store only a discretized subset of all available data. To achieve this, the environment is divided into a grid of individual cells of a specific resolution. The size of the map is directly proportional

to the operating area's size. Each grid cell represents the discretized area's state at the respective coordinate. Such states could, for example, describe whether the area is occupied, its distance to the closest object, or the corresponding area's height. Due to their intuitive usability, the discrete metric mapping approaches have risen in popularity and are widely used for autonomous robot navigation.

One of the earliest formulations of the problem are the Occupancy Grid Maps introduced by Moravec and Elfes[79] in 1984. The occupancy grid maps provide a crucial tool, especially in 2D mapping for mobile robots [10]. They divide the space into a regular grid in which each cell represents a predefined area of the environment. The size of these cells can be freely adjusted using a global grid resolution. The state of the area is given by a single probability value, which defines whether the cell is seen as occupied by an obstacle, free or unknown [121]. The occupancy probability value $P(n|z_{1:t})$ of each cell n is updated each time a sensor measurement z_t intersects its spatial locations. Since the probability update is prone to numerical instabilities around the 0 and 1 probabilities, often the log-odds notation is used for the state estimation:

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + \log\left[\frac{P(n|z_t)}{1 - P(n|z_t)}\right] \tag{2.3}$$

In order to determine whether a grid cell is occupied or not, the occupancy probability is compared to a fixed, user-specified threshold. This probability can be retrieved at any time from the log-odds representation with:

$$P(n|z_{1:t}) = 1 - \frac{1}{1 + e^{L(n|z_{1:t})}} \tag{2.4}$$

Since its introduction, the Occupancy Grid Maps have become quite popular and are still widely used to support robotic applications.

Another often applied close derivative of occupancy grid maps are the costmaps [22]. Contrary to occupancy grids, which store the probability of an occupied space, costmap stores an arbitrary cost within each grid cell. They are often generated by processing occupancy maps with additional environmental information in order to create a traversability map. The stored costs can be subsequently directly incorporated during a path-planning process. In Lu et al. [71], the approach was extended towards a Layered Costmap approach to overcome the limitations of a single monolithic costmap, rendering the map more efficient and extensible. In the approach, each layer is semantically separated and tracks one individual type of obstacle or constraint. Subsequently, all layers can be fused to regain a single monolithic costmap directly applicable to the path-planning problem.

Apart from the map size and resolution, the complexity and memory requirements of metric maps scales directly with the represented dimensionality of the environment. Reducing the three-dimensional world into a two-dimensional plane is mostly sufficient in simple, structured environments such as interior areas. Here, the volume of any object is projected downward onto its footprint to reduce the complexity. However, for more complex terrain, additional information

about all dimensions is crucial. It can be used for tasks such as detecting slopes or planning traversable paths over stairs. In single-level environments, the two-dimensional search space is sometimes retained using topographic 2.5D maps. This kind of map still represents the world as a two-dimensional grid. However, each grid cell additionally encodes the average height within the area. This can be especially useful for outdoor environments where knowledge about terrain variations significantly impacts the robot's navigation capabilities. However, in the case of more complex multi-level environments and overhanging obstacles, a full three-dimensional representation of the environment becomes necessary.

Historically, discretized metric maps were encoded as a regular grid of a fixed dimensionality. However, storing values in regular grids often poses the thread of a memory bottleneck. This is especially the case when facing large-scale environments or when a high resolution is required for the map. To solve this issue, Kraetzschmar et al. [57] propose to exchange the concept of using regular grids in favor of sparse grids, which group areas with identical values. This is achieved by encoding the map in a more efficient data structure, such as quadtrees or octrees in the 2D and 3D cases, respectively.

Hybrid Mapping Approaches

The last type of map can be seen as a combination of the map representations mentioned before. While operating an autonomous mobile robot, the basic geometric knowledge of whether a space is traversable is often insufficient. At the same time, sparse topological information is often not enough to fully operate a robot during tasks such as generating a collision-free path towards a goal. For example, could a robot be required to behave differently in specific areas of the map, such as steep slopes or staircases. Another primary example is the use case in which specific objects must be identified and marked within a map. Thus, a more sophisticated, context-rich representation that does not neglect geometric details is required.

To achieve this, hybrid approaches such as Semantic Mapping [93] are applied, which combine both the metric and topological map concepts. Semantic maps often come in the form of an augmented metric map. However, they not only describe the geometrical properties of the environment but also assign semantic labels to certain spatial elements. This could include identifying and labeling objects, relevant features, or even entire regions. As a result, semantic maps offer an additional layer of meaning and understanding to the representation, providing the means to make context-aware decisions in complex robotic tasks.

2.2 Path Planning and Navigation

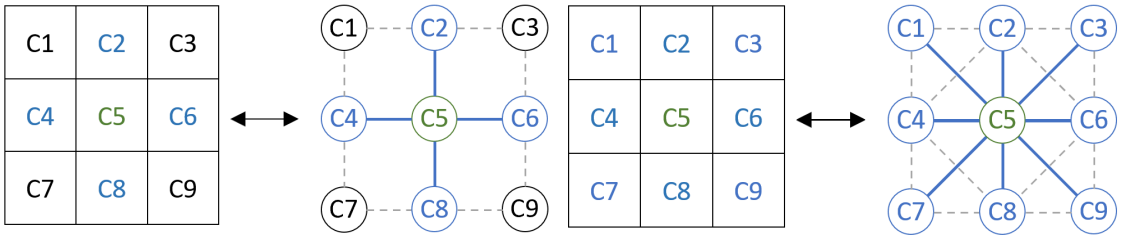
This section covers the foundations necessary for the navigation of autonomous robots through arbitrary environments. The process can be divided into two distinct steps. The first step is global path planning, which is a core part of the navigation of autonomous robots. Its main goal is to find an optimal global path from a starting point to an arbitrary destination. This path is calculated based on the overall information presented in a pre-recorded map. It should avoid any potential obstacles along the way in order to provide a collision-free path. One of the most established ways to calculate a global plan is to convert the mapped environment into a graph representation. Subsequently, proven graph search algorithms can be applied to find a valid path to the target destination. During the creation of the graph representation, a rough distinction between graph-based and sampling-based methods can be made. Graph-based algorithms require that the map representation is already graphed or can be directly transformed into one. Sampling-based algorithms, on the other hand, implicitly create their graph while exploring the environment by selecting random points from the map representations search space. This section gives a brief overview of the basic concepts for each of those planner classes.

2.2.1 Graph Based Planner

One of the most established approaches for the path-planning problem is to solve it on a graph representation of the environment. In this case, the path planning problem can be transferred to the shortest path problem in graph theory. Given a set of nodes and edges, the class of the shortest path algorithms is used to find the best, viable path from a start towards a goal node. However, in this case, the path is not necessarily the shortest path through the graph, as the name would suggest. Instead, it can relate to different criteria like the path length, the cost of the path, or the traversal speed. Since graph theory represents a well-researched field, it provides multiple algorithms to solve the problem of finding the best viable path within a graph [97]. In the following sections the fundamentals of creating a graph representation and the most commonly used optimal search strategies relevant to this work are discussed.

Neighborhood

For topological maps, which are already stored conceptually as a graph representation, this class of algorithms can be directly applied. In the domain of grid-based algorithms, however, initially, a transformation into a graph structure has to be performed. Each pixel or voxel of the grid structure is assumed to represent a single node within the graph. The edges between the nodes, on the other hand, are defined by the neighborhood of the grid cell. This neighborhood is defined



(a) 4-connectivity (b) 8-connectivity

Figure 2.3: The two most commonly used connectivity functions for two-dimensional path planning. The connectivity function determines which neighboring pixels/voxels are considered neighbors of the currently expanded cell. [Mau23]

by the connectivity function, which determines whether a traversal between two grid cells is possible. As a result, the density of the grid is highly dependent on the selection of neighboring cells the algorithm is allowed to traverse. Therefore, a greater movement capability (e.g., vertically, diagonally, and horizontally) leads to more considered nodes and a more extensive graph representation. Accordingly, this also increases the complexity of the graph search problem.

Figure 2.3 depicts the two most commonly used connectivity functions used in a typical two-dimensional path search. The orthogonal- or 4-connectivity connects all directly adjacent horizontal and vertical neighbors to the central grid cell. The diagonal- or 8-connectivity, on the other hand, also includes the four diagonal grid cell elements to improve the range of possible motions. Similar to the more simple two-dimensional case, these neighborhood functions can be accordingly extended to a three-dimensional search space. The basic orthogonal connectivity function becomes a 6-connectivity in which all six orthogonal movements, corresponding to the six faces of a die, are included. The diagonal connectivity, on the other hand, splits up into two distinct neighborhood functions. They can be roughly defined by their distance from the central grid-element. The 18-connectivity contains all grid cells, requiring only a single diagonal movement. The 26-connectivity, on the other hand, similar to the diagonal connectivity in the two-dimensional case, contains the set of all directly neighboring grid cells. Formally, the neighborhood function can be described as follows: Let $C(x, y, z)$ denote the current central grid cell and $N(x, y, z)$ a neighboring grid cell. In both cases, x, y, and z represent the spatial grid coordinates. The neighboring cells are defined as those sharing at least one contact surface with the central cell. A valid move from C to N on the grid is defined by the maximum allowed changes in each index. This corresponds to limiting the $l1$ norm of the vector v as defined by:

$$\mathbf{v} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{cases} |N_x - C_x|, \Delta x \in \{0, 1\} \\ |N_y - C_y|, \Delta y \in \{0, 1\} \\ |N_z - C_z|, \Delta z \in \{0, 1\} \end{cases} \tag{2.5}$$

Depending on the allowed movement distances, Table 2.1 shows the standard connectivity level for 2D and 3D grids.

Table 2.1: Graph connectivity conditions for 2D and 3D grids

Dimension	Connectivity	Condition
2D	4-Connectivity	$\|\mathbf{v}\|_1 \leq 1$
	8-Connectivity	$\|\mathbf{v}\|_1 \leq 2$
3D	6-Connectivity	$\|\mathbf{v}\|_1 \leq 1$
	18-Connectivity	$\|\mathbf{v}\|_1 \leq 2$
	26-Connectivity	$\|\mathbf{v}\|_1 \leq 3$

Dijkstra's Algorithm

One of the most fundamental search algorithms for finding the shortest path within a graph representation is Dijkstra's algorithm [16], which belongs to the class of greedy search algorithms. Dijkstra's algorithm is guaranteed to find the optimal path given the restriction that all edge weights within the graph are positive. This is achieved by iteratively selecting and exploring the node with the lowest cost starting from the initial node. Subsequently, all distances to neighboring nodes are updated if a shorter path is found. In its most basic form, this process is repeated until all nodes connected by vertices have been checked. In the case of a grid-based search, this would result in checking all available nodes which are not blocked entirely. However, in a more efficient form, the search is only continued until all possible routes have a higher cost than the shortest one found so far. As a result, the algorithm provides the path with the lowest sum of total costs between the start and end nodes.

However, the algorithm comes with a major drawback which makes it often infeasible in the context of a grid-based structure. This drawback is the necessary time complexity, especially when facing large or very dense graph representations. This is mainly because the algorithm will often explore in opposing directions to the target, resulting in multiple unnecessarily expanded nodes. The problem can be observed in Figure 2.4, which highlights the cells the Dijkstra algorithm explores during its search on a rather small map of an environment. It is evident, that the considered search space quickly expands. This problem becomes even more significant if the number of available grid cells increases. This can have multiple reasons, like the necessity for a higher map resolution or the extension of the path-planning problem to the three-dimensional case. As a result, the basic formulation of Dijkstra becomes quickly infeasible in real-world applications [32]. However, it provides the basis for more complex and sophisticated search paradigms.

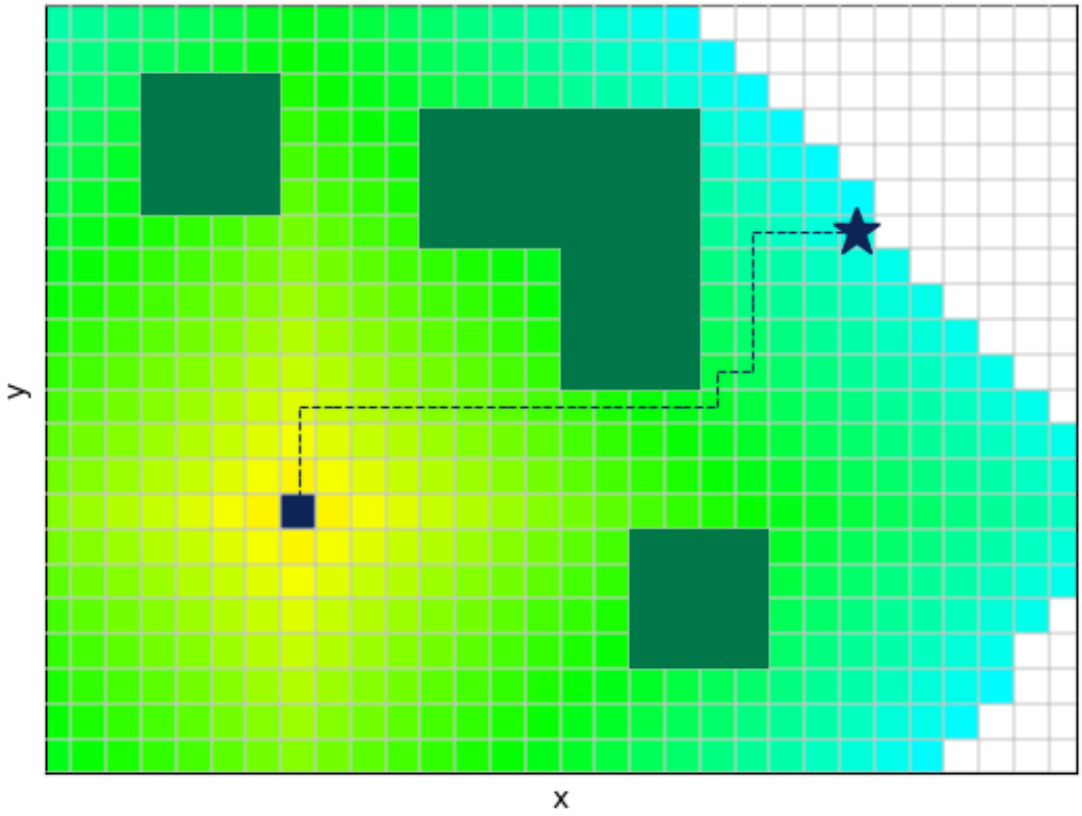


Figure 2.4: Dijkstra's search algorithm during a path-planning process. The black square defines the start node, the star defines the goal node, and the dashed line the resulting path. All expanded nodes and their corresponding costs are depicted as green to light blue grid cells. [Mau23]

A*

In order to reduce the increased computation time, it is advisable to avoid the unnecessary exploration of non-optimal solutions. To achieve this, cost heuristic functions as described in Section 2.2.1 are used to steer the algorithm's search pattern towards the direction of the target node. One of the most prominent members of this class of heuristic search algorithms is the A* algorithms introduced by Hart et al. in 1968 [34].

The A* is, at its core, an extension of Dijkstra's algorithm. However, instead of optimizing only the already accumulated cost toward a specific node, here, an estimate of the remaining cost towards the goal is included in the process. For this, the heuristic estimates the minimum cost between the currently expanded node and the target node. This approach leads to generally faster search results than an unfocused algorithm like Dijkstra's algorithm. As the heuristic steers the exploration towards the goal node, fewer nodes must be expanded. This advantage over the basic Dijkstra search becomes apparent in the results shown in Figure 2.5. It can be observed that using a heuristic during the search focuses the search di-

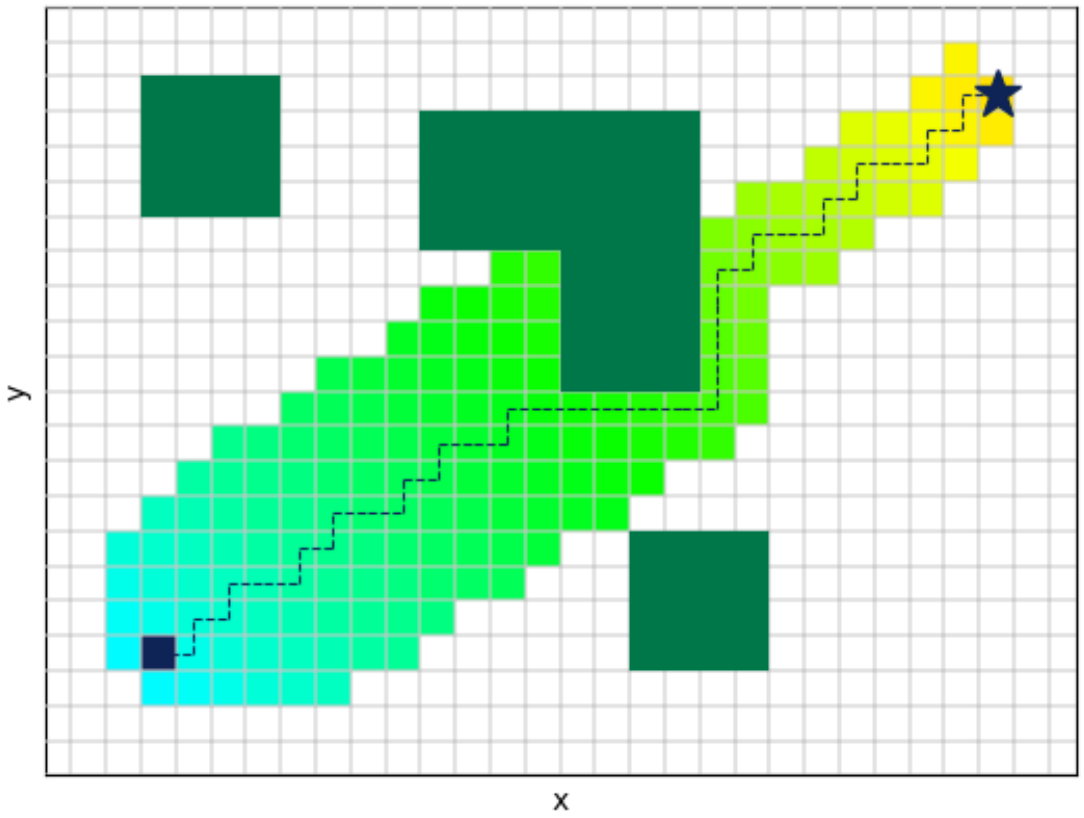


Figure 2.5: A* search algorithm during a path-planning process. The black square defines the start node, the star defines the goal node, and the dashed line the resulting path. All expanded nodes and their corresponding costs are depicted as green to light blue grid cells. [Mau23]

rection toward the goal, leading to a significant decrease in considered grid cells. Although the algorithm explores only a subset of the grid, it still is guaranteed to provide an optimal solution. However, to achieve this, the heuristic function is not allowed to overestimate the actual cost to reach the goal [62].

In detail, the A* algorithm explores one node at a time. Starting from the initial node, it continues until it reaches its final destination node or if no additional nodes are in its search queue. During the exploration process, the neighboring nodes, defined by the connectivity function presented in Section 2.2.1, are evaluated to find the most cost-effective successor node. For each expanded node, the overall cost that should be minimized during the search is defined by the equation:

$$f(n) = g(n) + h(n) \tag{2.6}$$

In this instance, $g(n)$ denotes the accumulated cost of the path from the starting node to reach the current node n defined by:

$$g(n) = \sum_{i=0}^{n-1} c(n_i, n_{i+1}) \tag{2.7}$$

where $c(n_i, n_{i+1})$ denotes the cost to reach node n_{n+1} from n_i. The term $h(n)$ represents the heuristic. It is defined by the estimated remaining cost from the current node to the target node. Often used heuristics in the context of spatial problems include models such as the Euclidean or Manhattan distance [58]. The resulting sum, denoted by $f(n)$, approximates the overall cost necessary to reach the destination via the current node [18].

The algorithm maintains two separate lists to track its progress during the exploration process. Those two lists are the *open list* and *closed list* [23]. The closed list stores all nodes that were already explored. The open list, on the other hand, contains all nodes that still need to be expanded and evaluated. For efficiency's sake, this second list is usually implemented as a priority queue, which is ordered by its combined costs $f(n)$. In each iteration, the node with the lowest cost is picked from the open list and evaluated. During the expansion of the exploration phase, each possible neighbor is checked against the open and closed list. If a visited node is already present in the closed list, it will be skipped. As the node was expanded beforehand, the algorithm already found a shorter path towards this node. Thus, it can be safely ignored. The same procedure and reasoning can be applied if the node is already present in the open list but with a lower cost $f(n)$. In this case the expanded neighbor can also be omitted since a more efficient path toward this node has already been found. If, on the other hand, the $f(n)$ value is smaller, the neighbor might represent a potentially more efficient path and is consequently added and ordered into the open list. After all neighbors are exhausted, the explored node is added to the closed list. Subsequently, the process is repeated in the next iteration by expanding the node with the lowest value of $f(n)$ in the open list. The algorithm terminates if either the open list is empty, meaning there are no more nodes to explore, or if the target goal was found. If a valid path is found, it can be reconstructed by back-tracing through the closed list. More specifically, starting from the initial node, the path can be followed by always moving to the node with the smallest value for $f(n)$.

Although the path is generally optimal, due to the use of the grid representation, it might still not be the shortest path from the start to the goal node. Using grid cells as search points combined with restricted neighbor selection limits the movement along the path. Figure 2.6 highlights this problem, in which an 8-connectivity was used for the neighborhood selection. It can be seen that the individual segments of the calculated path always have an angle that is a multiple of $45°$. These restrictions in the path's movement result in an accumulated longer path. This problem is further addressed by extensions such as Theta* [88] and Phi* [89], which perform so-called any angle path planning.

Heuristic

The A* algorithm applied here belongs to the class of heuristic search algorithms and is an informed variation of the Dijkstra search algorithm. It greedily chooses

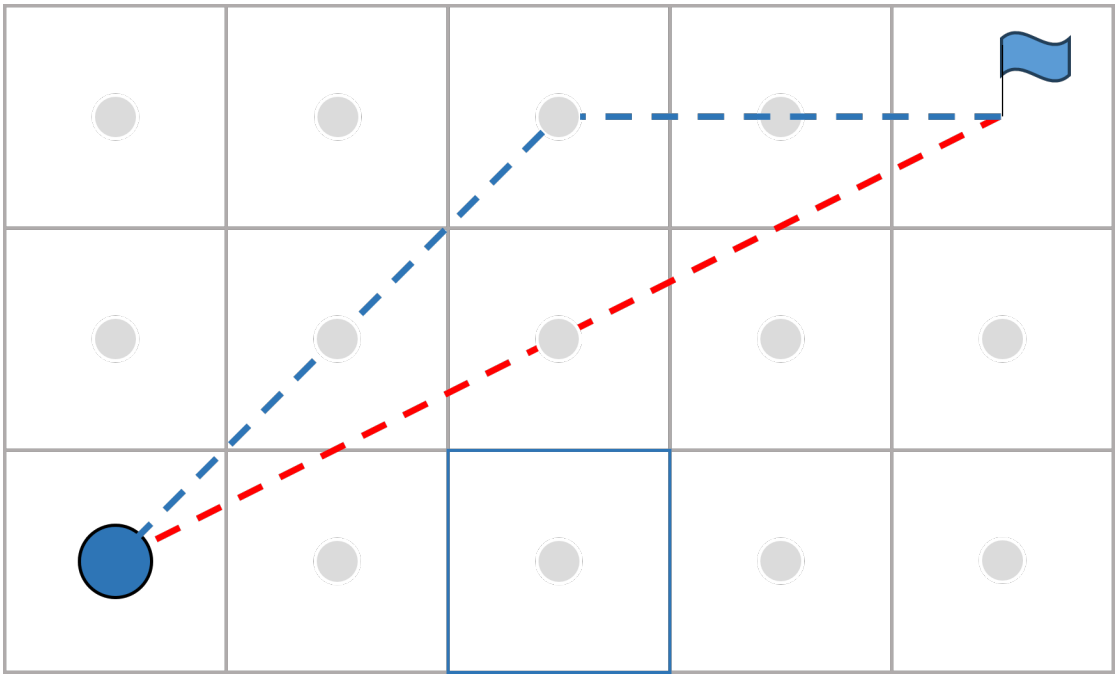


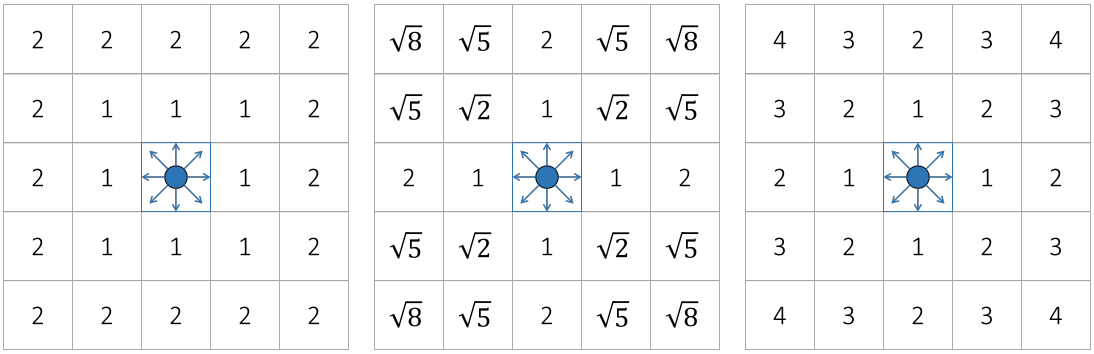
Figure 2.6: Concept of any-angle path planning. In grid-based approaches, the angle of the path is usually restricted to multiples of $\frac{\pi}{2}$ even though the direct path would result in a shorter path.

the vertex to explore next, given the value of $f(n) = g(n) + h(n)$, where $g(n)$ is the cost to reach this node, whereas $h(n)$ represents the heuristic.

The term heuristic initially stems from the ancient Greek word εὑρίσκω ('heurískō'), which roughly translates to 'method of discovery' [103]. A heuristic approach is applied in problem-solving when finding an optimal solution in a search space is impossible or impractical (i.e., too slow). This estimated guess is neither optimized, perfect, nor rationalized but nevertheless sufficient as an approximate solution to the problem. Instead, optimality, completeness, accuracy, and precision are exchanged for a faster achievable result. These heuristic strategies are often derived from previous encounters with a similar problem.

In the context of path planning algorithms, the heuristic estimates the remaining distance from the current expanded cell to the target. This technique supports the algorithm by guiding the search toward the goal in a promising direction. Hart et al. [34] state that the path calculated by an A* algorithm is always optimal, given that the heuristic is both admissible and monotonic. An admissible heuristic function is never allowed to overestimate the cost calculated from the current to the target node. Conversely, the monotonic constraint specifies that the estimated cost must fulfill the triangle inequality. More specifically, this means, given a point s and g, the estimated cost $c(s, g)$ must always be less or equal to the cost of moving from s to any neighboring cell n plus the heuristic estimate of the cost between n and g

$$c(s, g) \leq c(s, n) + h(n, g) \tag{2.8}$$



(a) Chebyshev distance (b) Euclidean distance (c) Manhattan distance

Figure 2.7: Heuristic distances for the two-dimensional use-case. The presented number indicates the resulting distance from the green central node.

If one of the constraints is violated, it can cause impairments to the search algorithm. Given that the heuristic is not correctly scaled relative to $g(n)$ and severely underestimates the remaining cost, it would degrade the A* more towards a Breadth First Search. In the extreme case of a value of $h(v) = 0$, the exact same solution as the Dijkstra search algorithm would be provided. On the contrary, given an overestimating heuristic, especially at large distances, the traveled cost $g(v)$ would no longer contribute significantly to $f(v)$. This would degrade it to a greedy Best First Search algorithm, preventing it from finding an optimal solution to the path planning problem.

In the context of path planning on grid structures like occupancy or volumetric voxel maps, the geometric properties of the data structure can be utilized to calculate the heuristic function. Different heuristic models might be applied depending on the specific constraints and allowed movement capabilities imposed by the underlying search problem. The correct heuristic in the context of path planning depends mostly on the cost function dictated by the movement restrictions of the search strategy. In the 2D case, some of the most commonly used heuristic models in path planning are the Euclidean distance, Manhattan distance, and Chebyshev distance. The heuristic model must fit closely with the connectivity function to fulfill both the constraints mentioned above. In the most restrictive case, given that the robot is only allowed to move directly north, east, south, and west, the 4-connected neighborhood is used. Given this neighborhood function, the Manhattan distance would be the obvious choice since it provides the exact same movement capabilities. This way, the heuristic model directly reflects the cost function of the robot's supported movement as accurately as possible. Without restricting the dimensionality, the distance is given by the equation:

$$D_{Manhatten} = |p_1 - q_1| + |p_2 - q_2| + \ldots + |p_n - q_n| \tag{2.9}$$

, where p and q are the current and goal positions given to the heuristic, which provide the index coordinates within the n-dimensional grid.

Using less restricted movement capabilities, in which the robot is allowed to

move diagonally between grids, an 8-connected neighborhood function is used. In these cases, the Chebyshev distance would provide the most fitting heuristic to mimic the expected cost. These heuristics are defined by:

$$D_{Chebyshev} = max(|p_1 - q_1|, |p_2 - q_2|, \ldots, |p_n - q_n|) \tag{2.10}$$

If the search algorithm belongs to the every-angle search strategies and permits movements in each direction, the Euclidean distance would be the obvious choice, which is defined by:

$$D_{Euclidean} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} \tag{2.11}$$

2.2.2 Sampling Based Planner

One of the main disadvantages of the previously introduced optimal search algorithm is its associated high computation time. The problem is even amplified in the face of high-dimensional planning search space. A high-dimensional search space can have multiple reasons. In this work, it might be desirable to utilize the full volumetric space of an environment, which naturally increases the dimension during the path planning problem. Another prominent area in which optimal path planners quickly reach the boundaries of their feasibility is calculating trajectories for robotic arms and manipulators. Here, each joint can be individually controlled according to its movement capabilities. However, to define a specific pose for the entire kinematic chain of an arm, all joints have to be considered in combination. The motion planning of those arms is commonly done in the so-called configuration space. This space represents all possible positions and orientations of each joint. More specifically, each joint contributes an additional dimension to the state space. As a result, each point in the configuration space corresponds to a different state of the robot. Since even simple arms contain four or more joints, the amount of possible states in the configuration space explodes quickly during the motion planning process. Thus, the guarantee of calculating an optimal solution quickly, given a limited time, becomes nearly infeasible. Besides the complex movement capabilities that have to be considered, the solution space is further restricted due to numerous obstacles in the configuration space.

To address this issue, probabilistic sampling-based planners are often applied in such instances. The main intention behind this class of algorithms is to decrease the complexity of the search graph in high-dimensional spaces where traditional grid-based methods are impractical. In comparison to optimal path planning algorithms, the most significant advantage of probabilistic planning algorithms is that they are able to find a path in complex environments given a limited time, even in the face of high-dimensional problems [130]. Two of the most commonly used classes of probabilistic sampling algorithms for simplified graph representations are the Probabilistic Roadmaps (PRMs) [49] and the Rapidly-Exploring Random Trees (RRTs) [61]. In both cases, the main idea is to randomly sample

points from the configuration space. These samples are used to determine possible configurations the robot could occupy. Subsequently, the states are verified and connected in order to construct a graph representation. Nevertheless, both methods differ significantly in the way this graph is constructed.

Sampling-based path planning approaches generally do not guarantee the shortest or optimal path. Instead, they provide a feasible path, given the time constraints imposed on them. The quality of the resulting path also heavily depends on the chosen sampling strategy, as it may require a large amount of samples to cover an adequate portion of the sampling space.

Probabilistic Roadmap

Probabilistic Roadmaps were first introduced by Kavraki et al. [49] in 1996. The PRM and all its derivates are multiple-query methods, meaning they are meant to perform multiple path planning operations on the same configuration space. Since the roadmap has to be only constructed once, it can then be reused for multiple queries. As a result, this class of algorithms is especially useful for path planning in static environments. For this, they first construct a graph or the eponymous roadmap. This graph representation contains a large set of collision-free trajectories between randomly sampled points. Subsequently, the path queries calculate the shortest path along the grid, which connects the initial and final states. Due to the reduction of the configuration space towards a more sparse graph representation, the PRM algorithm has been proven to perform well in a high dimensional state space [49]. Furthermore, the PRM algorithm is probabilistically complete since its probability of failure decays exponentially towards zero given a large number of samples during the construction of the roadmap [48]. Due to its usefulness in the presence of high-dimensional search problems, it is widely applied in the field of robotic manipulation, autonomous vehicles, or even to animate objects in computer graphics.

The overall algorithm can be split into two phases: the roadmap construction and the query phase. In the first phase, the roadmap is constructed by connecting the randomly sampled points to form a graph representation. For this, a set of points is randomly sampled from the configuration space. Each of these samples is verified in order to ensure that it is in a free configuration and not in collision with any obstacle. If a sampled pose is valid, it is added as an additional node to the graph representation. Subsequently, for each added node, it is tried to connect it to its nearest neighbors. For this, the path between both points is validated to be collision-free. This is usually done by checking simplified connections like straight lines or simple curves for which it is checked whether they intersect with any obstacle. In case the connection is collision-free, an edge is added to the graph between these configurations to represent a feasible path. Once the graph is constructed, it can be utilized for multiple path-planning queries. In this query phase, the goal is to find a path connecting the roadmap's initial and final state. To achieve this, well-established graph-search algorithms like Dijkstra or

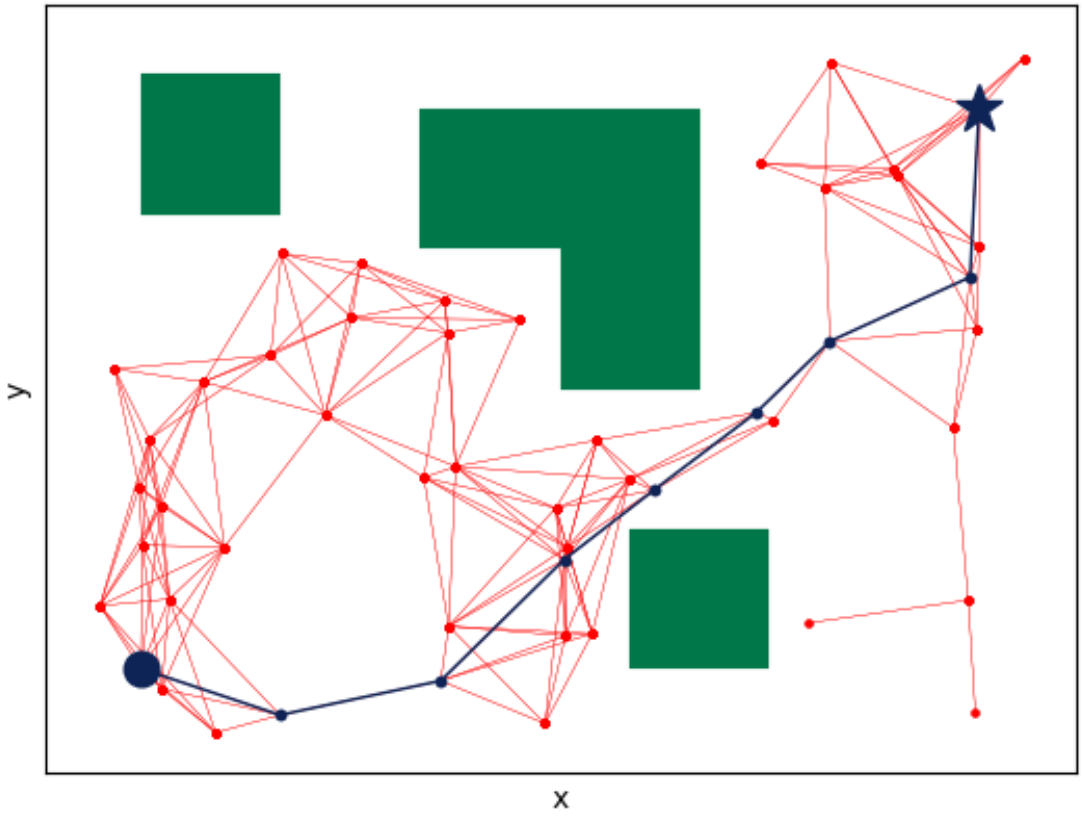


Figure 2.8: The PRM search algorithm during a path-planning process. The black square defines the start node, the star defines the goal node, and the dashed line the resulting path. The red graph depicts all randomly sampled nodes and the built roadmap. [Mau23]

A* can be used to find a valid path. Figure 2.8 shows a small exemplary roadmap constructed using PRM. In this case, the shortest path visualized in blue was calculated using Dijkstra's algorithm.

Rapidly-exploring Random Trees

Multiple-query algorithms are instrumental in highly structured and, thus, static environments. However, in the context of online path planning for mobile robots, they show severe disadvantages. Usually, multiple queries on the same graph are unnecessary since the robot traverses the environment, which continuously changes over time. As a result, the roadmap can not be reused and has to be generated from scratch each time. However, the computation of the entire roadmap during each planning process might be computationally difficult in some applications.

In order to alleviate these issues within the online path and motion planning process, incremental sampling-based planning algorithms are often used. One of the

most prominent kinds of these algorithms is the Rapidly-Exploring Random Tree [61, 42] search. This algorithm presents a single-query counterpart to the previously described Probabilistic Roadmap. Contrary to the aforementioned PRM, the RRT is primarily used for single-query operations and incrementally solves the motion planning problem. Here it is not necessary to sample a previously defined amount of random points in order to build a complete graph before finding a path. Instead, it samples new random points until the generated tree contains enough trajectories to provide a sophisticated solution to the problem. Due to its incremental nature, RRTs are predominantly suitable for real-time applications and enable the implementation of online path planning in dynamic environments. Similar to PRMs, the RRTs have also been proven to be probabilistically complete as their probability of failure exponentially decays toward zero [26].

In its basic form, the algorithm generates a tree built from a set of feasible trajectories to different configurations. Initially, the algorithms grow a tree T with only a single vertex V and no edges within the graph. This node represents the starting configuration and also functions as the root of the random tree. Subsequently, the algorithm iteratively samples a random point $x_{rand} \in x_{free}$ from the configuration space. In order to integrate the newly sampled point into the tree, the algorithm tries to establish a valid connection to the tree T. For each of those samples, the algorithm tries to find the node $x_{nearest}$ within the set of vertices V of the tree closest to x_{rand}. This is usually supported by a distance metric to find the most promising connection. Subsequently, the RRT steers a new node x_{new} from $x_{nearest}$ towards x_{rand} by a calculated input. The goal of this input is to minimize the distance from $x_{nearest}$ to x_{rand} while still ensuring that x_{new} remains in the collision-free space. This is usually done by iteratively moving x_{new} towards x_{rand} by a predefined step size in order to extend the tree. After each step, it is verified whether the node and the path leading to it are free from collisions with any obstacles. In case the path from $x_{nearest}$ towards x_{new} remains collision-free, x_{new} is added to the set of Vertices V and $(x_{nearest}, x_{new})$ to the set of edges E. As a result, the tree tends to extend toward unexplored areas, hence its name of "rapidly-exploring".

This process is repeated until one of two conditions is met:

- The algorithm has reached its maximum number of iterations and is not able to find a viable path from the start toward the goal configuration.
- The algorithm stops when the tree contains a node that is in the region close to the goal state.

In Figure 2.9, an example RRT path planning process is shown in the previously used environment. In this instance, the red nodes depict how the tree grows from the start toward the goal node. As soon as a new point is sampled close enough to the target, the tree connects it and returns the path.

Multiple extensions to the initial algorithm have been proposed to reach the goal configuration faster and more consistently. A prominent variant is, for example, RRT-connect [59], which proposes to generate two trees instead of a single one.

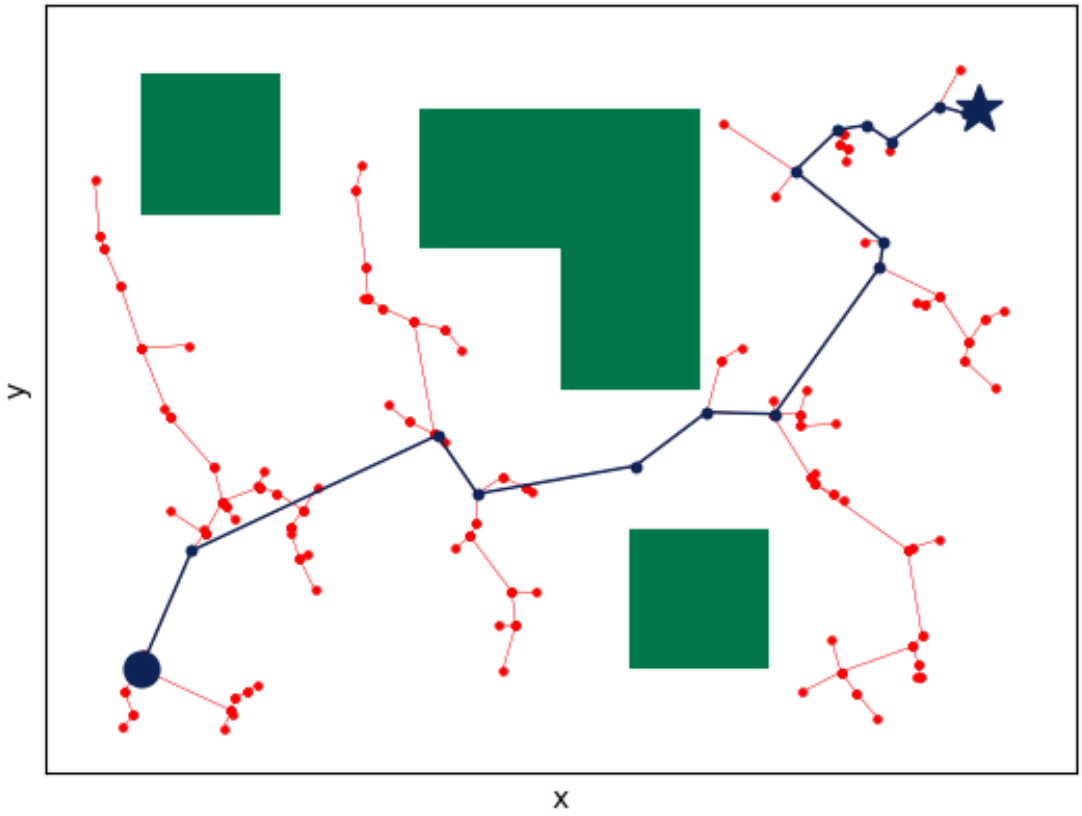


Figure 2.9: The RRT search algorithm during a path-planning process. The black square defines the start node, the star defines the goal node, and the dashed line the resulting path. The red graph depicts all randomly sampled nodes and the random tree. [Mau23]

The first tree is rooted like before at the starting configuration, while the second extends from the goal configuration. This way, both trees try to build a connection with each other, improving the chances that the randomly generated nodes create a valid path.

Optimality of Sampling-Based Planner

Since sampling-based planners are inherently probabilistic, the PRM and the RRT are not guaranteed to generate an optimal path. This is mostly due to the problem that the nodes necessary to achieve an optimal path are simply excluded as they were never sampled. To overcome this issue, Karaman and Frazolli [47] proposed PRM* and RRT* to extend both algorithms towards optimality. Once the first feasible solution is found, these algorithms optimize the tree by continuously creating the shortest connections. This is achieved by adding further samples in order to rewire the tree in a way that ensures that vertices are connected by a minimum-cost path. These algorithms are proven to return an optimal path

given infinite samples. However, in reality, they provide near-optimal paths far sooner than that.

2.3 OpenVDB

One of the basic building blocks of the proposed navigation framework presented in this work is the efficient OpenVDB data structure. It is utilized as the underlying data storage of the proposed map representation. For a detailed description of all concepts and further information on implementation details, the corresponding OpenVDB paper by Museth [83] should be considered.

Volumetric data is often stored within spatially uniform, regular 3D grids. While these dense regular 3D grids provide an easy and convenient representation, they suffer from one major problem. Since they have to encompass the entire bounding box of the volumetric data, their memory footprint grows proportional to the volume of the embedding space. As a result, even at moderate sizes, these dense regular grids are prone to becoming a memory bottleneck. To circumvent this issue, the solution is to employ a sparse volumetric data structure. In contrast to densely populated, regular grids, only relevant voxels are added to the data structure. As a result, the memory footprint scales only relatively to the number of voxels that contain actual information instead of the volume of the dense embedding space.

These sparse volumetric data structures are commonly implemented as a hierarchical tree structure, in which the individual tree layer represents the varying resolutions of the grid. Besides being more memory efficient, hierarchical data structures also offer additional advantages like improved performance during volumetric applications such as ray marching, CSG, or flood filling. Even though numerous sparse volumetric data structures have been proposed, most provide only slow or restrictive data access. Furthermore, they do not scale well while facing extreme grid resolutions or large-scale environments. Contrary to this, the OpenVDB data structure offers a memory-efficient data structure that provides fast and unrestricted data access. It can further support the representation of volumes with time-varying data and encode an arbitrary topology in a compact representation. Even though its primary intention is the representation of narrow-band level sets, VDB is also suitable to represent generic compact spatial data.

The titular VDB stems from its main attributes. It provides a sparse Volumetric and Dynamic grid structure that shares several similarities with B+trees. The initial application that drove the development of the VDB framework was calculating volumetric effects in Computer Generated Images (CGI) using high-resolution sparse level sets [96] in animation movies. The framework is based on the closely related Dynamic Blocked Grid (DB-Grid) proposed by Museth et al.

[87] in 2007. These grids were applied in Museth et al. [86] to calculate the fragmentation of solid geometries. Further development led to the highly improved hierarchical DB+Grid [81, 82], which combine DB-Grids with B+Trees. This grid structure was subsequently renamed into VDB and first applied to model high-resolution volumetric cloud representations in Miller et al. [77]. In 2012, it was released as an open-source library as OpenVDB during SIGGRAPH 2012 [83] and is currently maintained by the Academy Software Foundation.

Even though it has its roots in computer animation, it provides multiple characteristics and synergies that are equally useful in the context of robot mapping. OpenVDB presents a hierarchical compact data structure to encode data value and grid topology. It provides fast (average O(1)) random data access while inserting, deleting, or retrieving data. Apart from this, the framework offers a toolbox of efficient algorithms to operate on the data structure in order to utilize it to its fullest extent.

2.3.1 Characteristics

This Section provides an overview of the individual characteristics inherent in OpenVDB grids. Furthermore, it is inspected in which way each of these particular characteristics relates to the task of robot mapping and how it can be leveraged to improve this process.

Memory Efficiency Compact nodes are only dynamically and hierarchically allocated for corresponding areas that contain relevant data. This way, instead of allocating the entire embedding space, only a small portion of nodes is added to the tree. As a result, a memory-efficient sparse data structure is achieved. This characteristic is especially relevant for robot mapping of large-scale environments. It enables the robot to create a map with an extreme grid resolution for use cases such as precise manipulation tasks. To further increase memory efficiency, the framework provides efficient topology-based compression techniques. These compression techniques are combined with bit-quantization to optimize the memory footprint of stored or transferred maps.

Fast random and sequential data access One of the most important features of the VDB data structure is its fast, constant-time random data access. This includes looking up stored data, inserting new data, and deleting values from the grid. On average, the random access patterns support an O(1) lookup. This characteristic makes it highly suitable to represent dynamic maps, which are required to add new data within the sensor's data rate. New data can be quickly integrated and accessed to react to dynamic objects during navigation. Furthermore, VDB offers fast, constant-time, sequential iterators, which can quickly access multiple voxels in close vicinity to each other. These can be used to efficiently verify collisions of the robot during path-planning

Virtually infinite Each voxel within an OpenVDB grid can be addressed using a three-dimensional integer coordinate (x, y, z) within the index space to provide cache-coherent and fast data access. In concept, this 3D index space models an unbounded grid that is virtually infinitely large. As a result, it is possible to represent sparse volumes of equally infinite size as well as high-resolution grids. In reality, it is clear that the accessible coordinate index space is limited by the bit-precision of the voxel coordinates as well as the available memory. By design, the grid coordinate indices are unrestricted and can be potentially negative. The ability to extend the grid into the negative index space is highly desirable for applications with dynamic grid topology. Often, mapping frameworks circumvent this issue by using an offset to virtually move the grid into negative areas. However, this often requires realigning and resizing of the data structure when new areas that were previously out of the grids' bounds are explored.

Variable Node Types The data structure supports to store arbitrary data types within each node. Even though the data type can be chosen freely, it is important to note that for all functionalities to operate as intended, the node type should be restricted to static objects. In general, the data structure can also hold dynamic objects such as pointers, which only point toward other data somewhere within the memory. However, the fact that they do not store the relevant data directly would cause problems while saving or serializing the grid structure. Despite this restriction, the feature can be especially useful to generate maps of an environment that need to be augmented with additional arbitrary information for each voxel.

Separation of Values and Topology OpenVDB utilizes spatial correlation of its data to separately encode values and grid topology. This means that the topology of the tree and the corresponding values are handled completely individually from each other. Due to this distinction, the performance of all access operations to the VDB-Grid is still ensured, even if large additional data is stored within each node. However, the memory footprint is not directly influenced by this distinction.

Configurable generalized topology The tree structure of OpenVDB is highly configurable and can be adjusted for arbitrary use cases. All three relevant characteristics, namely the tree depth, branching factors, and the individual node sizes of each tree layer can be freely configured. This allows the tree to be adjusted to specific applications. Adjusting these settings can be used, for example, to optimize properties such as the memory footprint, cache utilization, and expected data access patterns.

Adaptive resolution The hierarchical structure of a VDB grid allows values to be stored at any level of the underlying tree structure. However, it should be highlighted that the values of different levels conceptually do not overlap in index space. As a result, the same coordinate in index space can represent different values at multiple levels of detail.

2.3.2 Data Structure

In this section, a detailed description of the VDB tree structure is given. Furthermore, it highlights relevant implementation details that are utilized during the course of this work. As stated earlier, the tree itself presents a hierarchical data structure that shares multiple features with B+tree [3]. This tree type is often used in performance-critical applications such as file systems and relational databases like NTFS and Oracle.

By design, the tree's height is balanced, meaning that the tree has a fixed depth in which all leaf nodes reside at the same layer. As a result the tree has a relatively wide but shallow structure. Similar data structures, such as octrees, adopt a much taller structure due to their small branching factor in each spatial dimension. This allows the allocation of fewer nodes and improves the capability to prune redundant nodes, resulting in a smaller memory footprint. However, this comes at the cost of reduced access times to elements in the lower tree levels since more traversals of subtree branches have to be performed to reach the desired node. Conversely, OpenVDB adopts larger branching factors, which effectively increase the representable domain while simultaneously reducing the depth of the tree. As a consequence of the fixed depth, the number of I/O operations necessary to traverse the tree from its root to any leaf node is also constant. The tree depth and branching factors can be altered using different configurations of the individual node types and their parameters. The configuration of these tree parameters has, in turn, a high impact on the different characteristics of the tree. This includes traits such as the available resolution of the grid, the performance while accessing or modifying values of the tree, and the required memory footprint to store it. Depending on the desired use case, the following correlations can be summarized:

- The resolution of the grid scales directly with the chosen branching factors as well as the tree depth.
- Shallow trees result in generally faster random access since the length of tree traversals is reduced.
- In order to increase the adaptivity of the tree, small branching factors and the resulting height of the trees are favorable.
- The memory scales directly with the amount and size of allocated nodes.
- Smaller nodes, combined with deep trees, lead to an increase in cache hits.

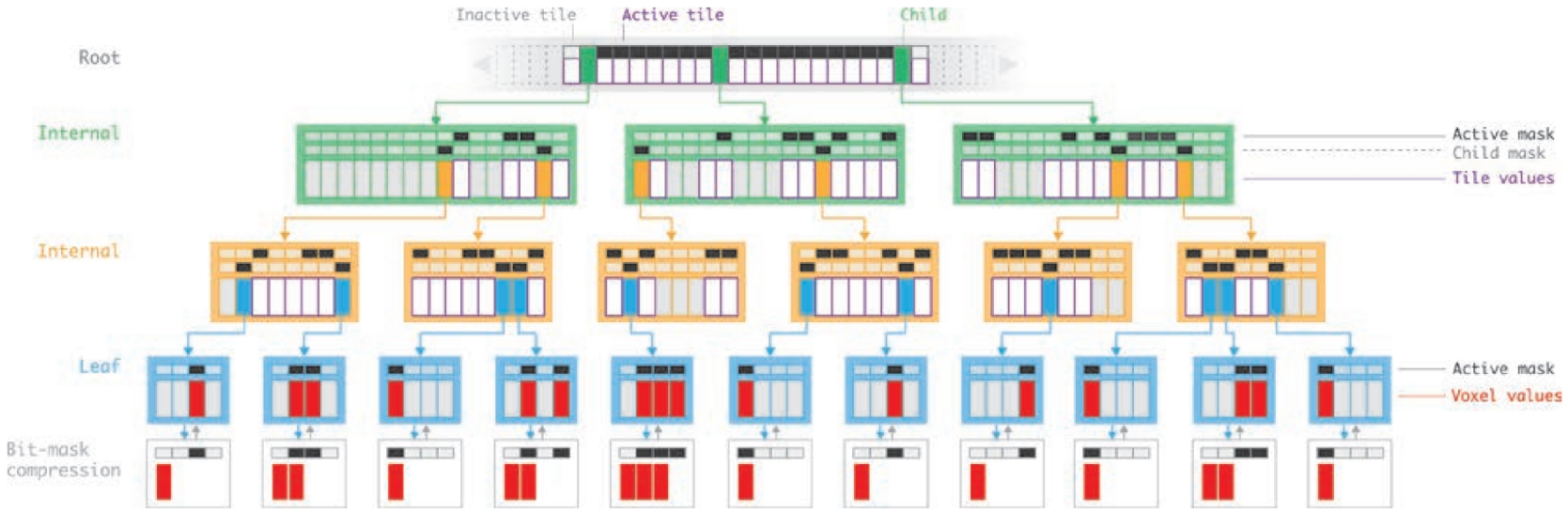


Figure 2.10: One-dimensional abstraction of the default VDB structure. The setup contains four layers: one unbounded root node, two internal layers, and one leaf node layer. The fixed branching factors of the internal and leaf layers become increasingly smaller to improve access performance.[83]

Figure 2.10 shows a 1D abstraction of a configuration which balances most of these factors for a wide range of sparse volume applications. It encodes a short and wide height-balances B+tree with small leaf nodes. It contains four tree levels (i.e., a dynamic root node, followed by two levels of internal nodes and a layer of leaf nodes) with increasing branching factors from the bottom up. In general, this tree configuration shares multiple similarities with the design of modern CPUs. It is inspired by the setup of Multi-Level-Caches, where a fixed number of nodes of decreasing size are used to increase the random-access performance.

The structure of a VDB grid is encoded entirely in the tree topology, which is stored in efficient direct access bit masks residing in each node. The values of the tree, on the other hand, are stored explicitly in buffers on each level of the tree. Each value and node is accessible using its corresponding discrete index (x, y, z), which specify its spatial location.

The tree structure itself is built from three distinct node types: the root, internal, and leaf nodes. Conceptually, the root node of the tree node covers the entire index space. Each of its child nodes covers a predefined subset of the same space. The dimensions of all nodes are fixed at compile time and effectively distribute the index space into non-overlapping subregions. Each of those regions contains $2^{\log_2 w}$ voxels along each coordinate dimension, where $\log_2 w = 1, 2, 3, ...$ For example, a typical configuration of $\log_2 w = 3$ would correspond to 8x8x8 voxels in each leaf node. As a result, the dimension of each node is strictly restricted to powers of two. This crucial property enables fast bit shift operations on the grid to significantly increase its overall performance. Each node type is highly templated, which allows arbitrary values to be stored within the grid. To provide an adaptive multilevel grid, these values do not only reside at the leaf node level. Instead, each non-leaf node is also able to hold a value. In case an internal- or root node contains a so-called tile value, it is assumed that each child of this node represents the same value. This way, redundant information can be stored efficiently in higher hierarchy layers of the tree.

In the following sections, a brief overview of the individual components and how they interact with each other is given.

Direct Access Bit Masks

One of the most fundamental aspects leveraged to achieve the efficiency of the data structure is the separation of tree topology and data values. In OpenVDB, efficient and compact direct access bit masks are used to encode this topology. These bit masks are embedded as two types into each node at all levels of the tree. The *value masks* specify whether a leaf or tile value is active or inactive. This information denominates, whether a subjacent node contains meaningful data. Leveraging this compact representation, it is possible to obtain fast sequential iterators for all available data. Contrary to similar data structures where the entire

data structure has to be considered, here only the compact bit mask must be iterated to find nodes that contain valuable data. The *child masks*, on the other hand, contains information about the child nodes of each node. For this, the child mask specifies if and where the current node contains further child nodes. This way it can be quickly determined if and where children are present without actually accessing the data structure.

Reducing the topology to a compact binary representation allows fast, direct access to the tree structure. Furthermore, the spatial index space of the corresponding node can be directly derived from each bit's position, which enables encoding the entire topology using only these two mask types.

An additional advantage of these bit masks is that they can be used for lossless compression. Each of the bits in the direct access bit masks corresponds to a single specific spatial location. Since the spatial coordinate of each subjacent node and value can be unambiguously recovered using the bit mask, the node itself can be stored in an ordered block in memory. This way, saving not-allocated nodes can be omitted without loss of information during the reconstruction of the tree. Furthermore, the bit masks can be utilized to improve operations on the grid structure. Since the bit mask is built up of individual bits for each voxel, efficient boolean and logical operators can be directly applied to the mask. This enables multiple efficient applications, such as using a logical *AND* on the value mask to extract a subset of tree nodes. Another area in which these operations result in efficient algorithm implementations is Constructive Solid Geometry (CSG). Here, multiple grids can be easily combined, extracted, or differentiated, which can be especially useful when merging individual map sections. Similar to the boolean operations, the bit mask can be efficiently used for topology algorithms on the voxels. This includes applications that only require the binary representation of the index space, such as dilation, erosion, narrow-band rebuild, or flood-filling of volumes.

Node Types

Even though all nodes of the VDB tree are quite similar, a few characteristics distinguish them from each other. They are mainly grouped by their use case and, thus, the depth in which they are placed within the tree. Each tree consists of three types of nodes: the root node, followed by multiple layers of internal nodes, and finally, a layer of leaf nodes. The branching factors of each node of the tree, even though fully configurable, are restricted to powers of two. This allows efficient bit-shift operations and enables fast tree traversals.

Similar to other data structure, the leaf node, defines the lowest-level blocks of the grid. By design, each leaf node resides at the same tree depth. In most tree structures, this lowest leaf-level stores a single value for this spatial location. Contrary to this, the VDB stores multiple value buffers as well as an additional direct access bit mask in each leaf node to perform efficient grid operations.

The internal nodes are used in all intermediate levels of the tree and reside between the root and leaf nodes. They are used to define the tree's overall structure, meaning the depth and shape of the VDB tree. Contrary to the leaf nodes, which only store data, internal nodes encode both values and the tree's topology. For this, each internal node holds pointers to other subjacent internal or leaf nodes. The topology itself is compactly encoded inside the direct bit access child mask, which specifies the spatial index space in which a subjacent node resides. The value mask, on the other hand, indicates whether a stored tile value of the internal node is active or not. The configuration of each internal node layer of the tree can be chosen individually. As a result, each internal tree layer can have a different branching factor to provide more flexibility in shaping the tree to a specific use case.

The root node of the tree represents the topmost node of the tree on which each operation commonly begins. There is only a single root node that covers the complete spatial area of the index space. Usually the domain of a volumetric data structure is defined by the size of the root node. However, in order to obtain an unbounded grid domain, which simultaneously requires only a small memory footprint a dynamically resizable root node is necessary. To achieve this, OpenVDB uses a sparse root node containing a dynamic hash-map that can be easily resized during runtime. This hash map is kept sparse and contains very few entries. This is primarily due to the vast index domain represented by each node within it. For example, the default VDB layout is a four-level tree in which after the root node, two internal and one leaf layers have a branching factor of $\log_2 w =< 5, 4, 3 >$ respectively. This results in the following index domains for each dimension of a single node:

$$internal_1 = 2^5 = 32 \tag{2.12}$$
$$internal_2 = 2^4 = 16 \tag{2.13}$$
$$leaf = 2^3 = 8 \tag{2.14}$$
$$root = internal_1 * internal_2 * leaf = 4096 \tag{2.15}$$

This sums up an index space of 4096^3 voxels for each root node hash table entry. Given a grid resolution of $5\,\mathrm{cm}$ per voxel, this covers an area of roughly $(204.8m)^3$ or $8589934.592m^3$ As a result, the root node is not required to frequently reallocate nodes. Therefore, despite being, by design, the bottleneck of the data structure, the root nodes do not pose an efficiency problem. Nevertheless, it is generally the case that accessing a dynamic sparse data structure, such as a map or hash table, is slower than performing a lookup into a dense direct access bit mask of fixed size. From this, it could be reasoned that the root node suffers a major disadvantage in access times compared to the fast internal nodes. To address this issue, the root node contains a registry of accessors, significantly improving spatially coherent grid access. This is achieved by reusing cached node pointers from previous access operations as described in Section 2.3.3.

2.3.3 Data Access

One of the most common but simultaneously most challenging data access patterns is the random access to arbitrary voxels within the tree structure. In the worst case, this requires to perform a complete top-down traversal of the tree for each new access. Starting from the root node, the accessor has to traverse all intermediate nodes until possibly terminating at the specific leaf node. Since this imposes high computational costs, truly uniform random access should always be avoided.

Despite this, most non-sequential access patterns still contain some exploitable spatial coherence. This coherence can be leveraged to improve the seemingly random access significantly. In VDB, the main idea to improve the random access is by inverting the tree traversal order during each grid access operation. Typically, during a lookup, the search would start at the tree's root node, traversing down along the path of subjacent nodes until the desired spatial location is reached.

Since each VDB tree is, by design, height-balanced and has a fixed depth, each traversal from the root node to one of the leaf nodes is equally long. From this, it can be argued that each random lookup of a leaf value poses the same worst-case time complexity.

During traversal, both the internal and leaf nodes utilize the previously described direct access tables. This enables them to provide a worst-case random access time of O(1). The root node on the other hand utilizes an optimized hash map and efficient hash function. This results in an average time complexity of O(1) with a worst-case time complexity of O(n). Thus, it becomes clear that the overall time complexity of each random access is given by the performance of the root node. If, instead, an internal node is used as the top-level search entry point, the random access lookup would remain O(1) even in the worst case.

Thus, instead of initializing each random access at the root node, the tree is traversed in an inverse order. For this, a node accessor residing in each root node element is used to cache the sequence of nodes visited during the last access operation. This way, the access operation can traverse the tree from the bottom up until a common ancestor for both the previous and current lookup is reached. This common ancestor can subsequently be used as a starting point for the current random access. Consequently, greater spatial clustering of the access pattern will, on average, result in a more significant speedup. More specifically, this means that if multiple entries of a leaf node or the node above are accessed in a random pattern, the traversal paths to a common ancestor and back below become shorter. This, in turn, leads to a drastic increase in performance during random lookup operations. On a larger scale, the increased access time of the root node can be fully amortized if its child nodes are sufficiently large to maintain average spatial coherence for all practical access patterns.

Contrary to this, sequential access patterns are more efficient. In multiple algorithms, it is required to access or modify all active values of a grid. These algo-

rithms are invariant to the specific sequence in which they process each voxel of the grid. By defining a sequential access pattern, this invariance can be leveraged to improve the performance compared to random access. Given the structure of each VDB node, this task is surprisingly easy to achieve by leveraging the direct access bit masks. The sequential access can be simply defined by iterating over the efficient, binary child and value mask. Since each node contains the bit masks for its subjacent nodes and values, they can be used to combine bit mask iterators at multiple levels of the tree. This way, it is possible to derive iterators over active voxel values, active tile values, or leaf nodes. The inherent compactness makes these bit masks cache-friendly and enables algorithms to process multiple entries or bits simultaneously. The linear offset between the spatial coordinates of two subsequent bits of a mask can be efficiently retrieved using a DeBruijn sequence as described in Leiserson et al. [63]. This way, efficient sequential iterators over bit masks can be obtained.

2.3.4 Accelerated Ray Tracing

Ray tracing originated in the 16th century when Albrecht Dürer used it to project 3D scenes onto an image plane [39]. In 1968, it was first used in the context of image processing to generate shaded pictures by Arthur Appel [1]. This was achieved by tracing a ray for each point in the image to determine the closest surface with which it would intersect. Even now, rendering the global illumination of computer-generated imagery is based on similar but more sophisticated principles. Today, ray tracing is one of the most commonly used techniques in image computer graphics and animations. Despite its primary use in these areas, it also offers high synergies in the context of automated robot mapping. During the mapping process, each sensor ray has to be traced through the map structure to determine whether a voxel along the way should be considered free or occupied. More specifically, each passed voxel along the ray is updated toward a free cell, while the endpoint represents the obstacle from which the sensor measurement was reflected. Thus, this last voxel is considered an obstacle and will be updated accordingly in the map representation. To achieve this, it is necessary to find all the individual voxel cells through which each of the beams of a sensor measurement traverses. This process, depicted in Figure 2.11 is commonly referred to as rasterization. One of the oldest published algorithms for this was proposed by Jack Bresenham [6] in 1963. It was used to control digital plotter devices in which the plotting pin could only move in fixed horizontal, vertical, or diagonal spacing along a grid.

Conceptually related to this algorithm is the class of Digital Differential Analyzer algorithms, which can also be used in the volumetric context. Like the Bresenham algorithm, they can successfully determine all voxels or tiles intersected by any given ray. Additionally, the hierarchical tree structure can be leveraged to accelerate the ray marching process further. To achieve this, Ken Museth [84] proposed Hierarchical Digital Differential Analyzer (HDDA), which takes full advantage

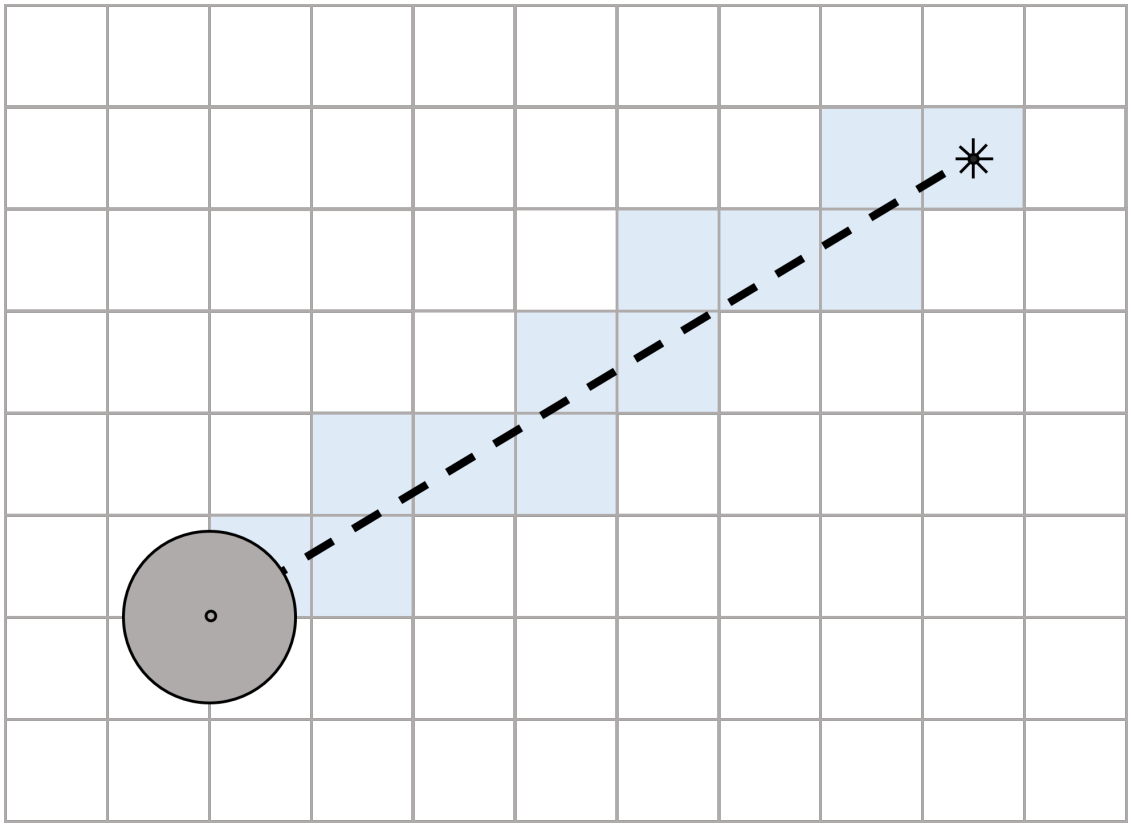


Figure 2.11: Rasterization of a single laser beam. The algorithms provide all discretized grid elements which are traversed by the sensor ray.

of an underlying B+tree structure. The main advantage of HDDAs is their ability to quickly traverse along the ray by skipping over free spaces in lower hierarchical levels of the tree. This can be used to skip large portions of the index space by recursively intersecting the cast ray against the tree nodes at any level of the tree structure. Given that a higher-level node is completely empty, the ray casting does not necessarily need to traverse down on the leaf level. Instead, these free nodes can be entirely omitted during the ray casting operation, significantly reducing the amount of traversed grid cells. To further increase the performance of the ray casting capabilities, the properties of the tree configuration fixed at compile time are also leveraged at this point. Given the semi-sequential access pattern of ray casting combined with the efficient hierarchical caching mechanism, the overhead of vertically traversing the tree can be reduced significantly.

2.3.5 Morphology

Mathematical morphology is a theoretical model based on set theory, lattice theory, and topology. Its initial use case was based on the image processing of binary images. This non-linear image processing method is commonly used to analyze and influence the size, shape, convexity, and connectivity of geometri-

cal structures. It was first introduced in Jean Serra's PhD Thesis [108] in 1963 when it was used to quantify the mineral characteristics from thin cross-sections. Subsequently, he and his supervisor, Georges Matheron, founded the Centre de Morphologie in Fontainebleau to further their research and investigate integral geometry and topology advancements. Even though mathematical morphology was originally developed only to process binary images, it was later extended to grayscale functions, images, and complete lattices [109]. In this work, mathematical morphology operations are used to pre-process sparse voxel volumes to improve their quality for easier use in subsequent processing steps.

Basic Operators

In the context of image processing, mathematical morphology is based on two basic operators, namely dilation and erosion. These two operators are used on an input image in combination with a structural element to create an adjusted output image. In this regard, the structural element performs the role of a 2D convolution kernel similar to other image processing operations, such as a band pass filter in image smoothing. Common filters are, for example, the 4- or 8-connectivity:

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & \otimes & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \otimes & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.16}$$

or more complex structural elements like the approximation of a circle with a radius of 2:

$$B = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & \otimes & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.17}$$

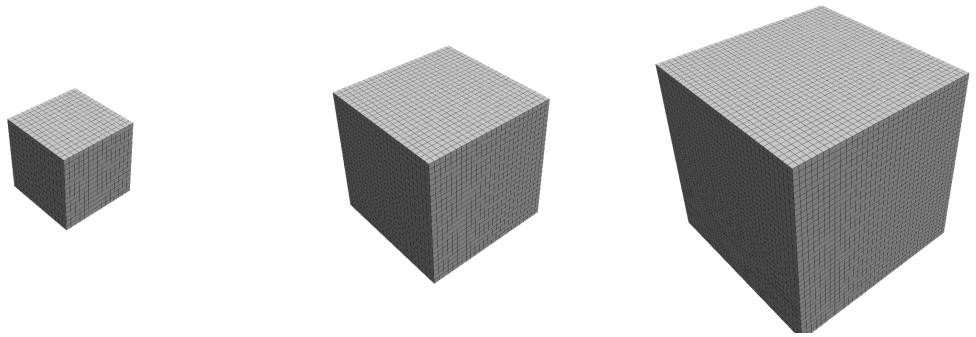
In all three cases, the origin pixel denoted by $\otimes$ is the point of reference around which the convolution is performed. Given an input image A, the erosion is defined by

$$A_{erosion} = A \ominus B \tag{2.18}$$

This operation is intended to remove the outer boundary or layer of an object. As a result, the given objects are decreased in size, and previously connected elements might be separated by the operation. Dilation, on the other hand, provides the opposite effect and is defined by:

$$A_{dilation} = A \oplus B \tag{2.19}$$

Contrary to the erosion, in the dilation process, an additional layer is added to all geometrical shapes. The actual shape of the peeled or added layer is, in both cases, defined by the previously introduced structural element. Figure 2.12 show



(a) Erosion (b) Input volume (c) Dilation

Figure 2.12: The two basic morphological operators. In the middle, the input cube volume is depicted. On the left, the volume was eroded, meaning the outer layers of the volume were morphologically removed. The right figure, on the other hand, depicts the dilation process where additional layers have been added to the volume.

the results of both basic operations on a simple cube volume. Here, the input volume depicted was eroded and dilated. It can be observed that the process either peels away or adds voxel layers to the volume while the initial cube structure is retained. In this instance, the process was performed multiple times to enlarge or reduce the object further. The operations can be executed multiple times to achieve a large extent of the added or scraped layers around the shape.

By combining both of them in different orders, two more operations can be realized, namely the opening and closing operators. The opening of A and B is defined by:

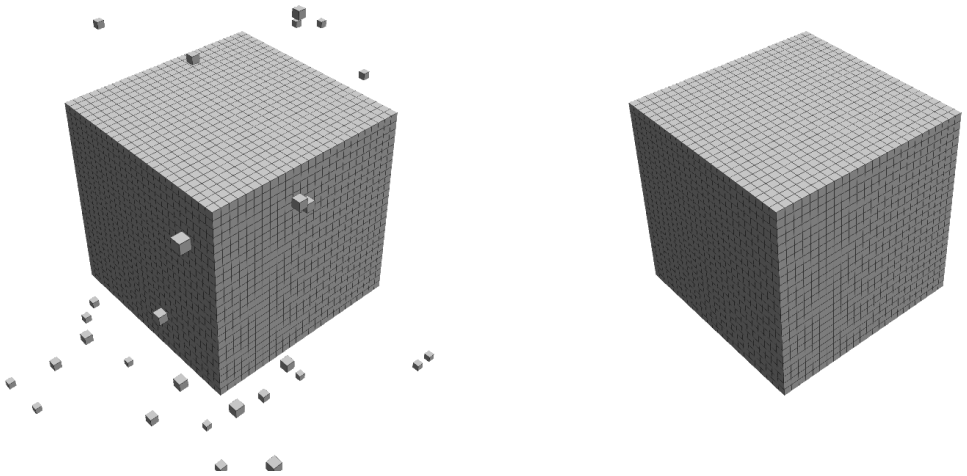
$$A_{opening} = A \circ B = (A \ominus B) \oplus B \tag{2.20}$$

Here, first a layer is eroded from the shape and subsequently another layer is added. At first glance, this seems to be an operation that results in the same image as before since the removed and re-added layer is both defined by the same structural element. However, this process can, for example, be efficiently used to remove noise from an image, as depicted in Figure 2.13. First, all individual freestanding elements which are not part of a larger shape are removed. During this process, the outer layer of the initial volume is also reduced by one voxel layer. Therefore, the removed layer of the shape is added again, resulting in the input shape without ambient noise.

The closing operation of A and B, on the other hand, is defined by:

$$A_{closing} = A \bullet B = (A \oplus B) \ominus B \tag{2.21}$$

As the name states, this operator's main application is to close the infill of shapes, as can be seen in Figure 2.14. This time, first, an additional layer is added to the object to close off all holes or missing infills. Subsequently, the outer layer is removed again to regain the initial shape of the object.



(a) Input volume (b) Opening

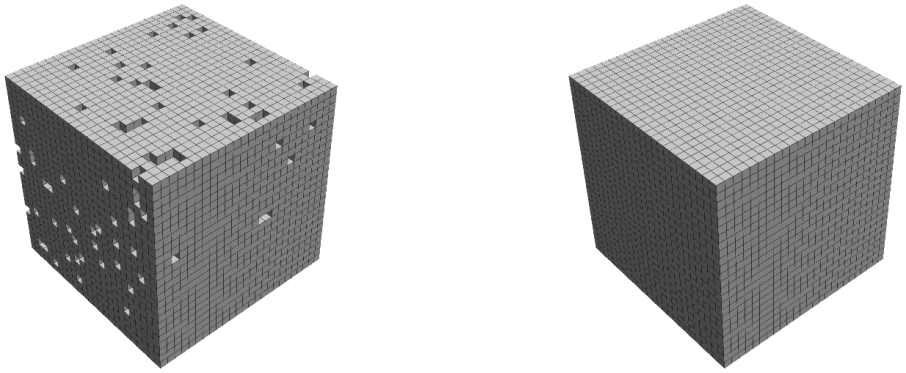
Figure 2.13: The morphological opening process. As input, the volume depicted on the left was used. It can be seen that it is cluttered with random noise voxels. The picture on the right shows the volume after the morphological opening. It can be seen that all random noise is removed by the initial erosion operation. Subsequently, the initial structure of the cube is reconstructed by the dilation.

Volumetric Morphology

Even though it originated in 2D image processing, mathematical morphology is not strictly restricted to this domain. In this case, it can also be used to deform three-dimensional volumes. Instead of adding or deleting only a band of pixels around the object, an entire layer of voxel is added around the entire volume.

For efficiency reasons, the volumetric morphological operators are performed purely on the grid topology but not on the voxel values. As a result, it only applies to the tree's topology without adjusting the grid's individual values. To achieve this, the topology algorithms can be directly applied to the value mask, providing much faster processing compared to employing the morphology to the entire data structure.

The approach provides the significant advantage that the operation can be executed extremely fast on the bit masks. This efficiency is mainly achieved by a scatter algorithm that scatters a projection of each access into neighboring bit masks. For the dilation, this is achieved by employing simple bit-wise OR and bit-shift operations. The erosion, on the other hand, is defined similarly and uses only bit-wise AND and bit-shift operations. This way the efficiency of each mathematical morphology operation could be significantly improved since it updates all bits of a mask at a time while completely avoiding random access to voxel values.



(a) Input volume (b) Closing

Figure 2.14: This figure depicts the morphological closing process. As input, the volume depicted on the left was used. It can be seen that it contains random holes on the volume surface. The picture on the right shows the volume after the morphological closing. It can be seen that all random holes are filled by the initial dilation operation. Subsequently, the initial structure of the cube is reconstructed by the erosion.

2.4 Conclusion

In this chapter, a rough overview of the necessary foundations of robot navigation on which this work was based has been given. This overview focused on three primary topics. First, the basics of robot mapping were covered. One of the most crucial questions is how robots localize themselves in unknown environments. This is usually solved using a Simultaneous Localization and Mapping approach. Therefore, this chapter gives an introduction to problem formulation as well as various approaches to solving it. Additionally, in the localization process, the algorithm also has to store the gathered sensor data efficiently. Thus, a rough overview of viable map representation formats is given.

The second topic that was addressed in this chapter is the fundamentals of path planning. Here, the two prevalent paradigms of graph-based and sampling-based path planning approaches are covered.

The third part of this chapter gave an introduction to the highly utilized VDB data structure. It gave a detailed overview of how the framework is conceptually designed. Furthermore, it explained how these properties can be leveraged and synergized with the field of autonomous robot mapping. The framework offers highly efficient grid operations. Thus, an in-depth examination of the data structure and its implementation details was given. Even though many existing data

structures possess a similar subset of these characteristics, OpenVDB combines and implements all of these in an efficient and performant way. As a result, the OpenVDB data structure offers excellent capabilities and synergies to create an efficient large-scale mapping framework capable of handling high resolutions.

The chapter finishes with an introduction to two core concepts highly relevant to the task of robot mapping. First, the basic concepts of ray casting and how it can be used to trace sensor measurements are explained. This was followed by a brief introduction to mathematical morphology. It was shown how it can be used to improve map data in the initial two-dimensional as well as three-dimensional context.

3 Related Work

This chapter gives a brief overview on scientific achievements highly relevant to this work. First, in Section 3.1, an introduction to the process of automated robot mapping is given. It inspects insights into how robots perceive their environment and provides a summary of multiple mapping approaches in various dimensionality. Following this, Section 3.2 provides an overview of commonly closely related path planning concepts.

3.1 Mapping

The field of environmental representations can be roughly divided into topological and geometric maps [121]. Topological mapping focuses more on encoding an environment's semantic information. The topological mapping approach represents the environment by reducing it to significant landmarks and paths connecting one another. As a result, a sparse topological graph of the environment is created. The nodes in this graph contain information about the position of the landmarks, whereas the edges connecting them are used to represent the relative path between them. Topological maps can be efficiently used for high-level planning since tasks can be easily describable using semantics (e.g., telling the robot to move into the kitchen and open the green cupboard). However, from this information, it is not possible to generate a precise motion plan for the robot. Geometrical maps, on the other hand, focus, as the name states, on the geometric features of an environment. This predestines them to be used as an underlying data representation to plan collision-free paths toward distant locations. Therefore, the main focus of this thesis lies more within these geometric map representations.

3.1.1 Grid-Based Map Representations

Generally, two-dimensional geometric mapping algorithms are widely used and perform well in simple, structured environments such as storage facilities. In this instance, present obstacles are usually projected down on their planar footprint to reduce the complexity of the map. Due to this restriction, planning on these map representations is quite efficient and thus has been extensively researched in the robotic community. One of the first and most prominent map representations in robotics is the Occupancy Grid Map proposed by Moravec and Elfes [79]. Here, the environment is divided into a two-dimensional grid. Each individual cell

represents a corresponding area in the real world. The size of the area is defined by the grid resolution whereas the state is encoded as an occupancy probability. Generally, it can take on one of three states: free, occupied, or unknown. In Konolige [56], this method was improved by introducing the so-called MURIEL (MUltiple Representations, Independent Evidence Log) method. This method is focused on handling specular reflections and discount redundant readings during sensor fusion to improve the performance of the occupancy grid with sensors such as radar. Later, Thrun [120] introduced forward sensor models to further improve the occupancy probability calculations. More specifically, it utilizes the expectation-maximization algorithm to estimate the map while uncertainty is determined by Laplacian approximation. In Kraetzschmar et al. [57], the concept is extended by using sparse quadtrees instead of a regular grid. This way, multiple cells or areas that contain the same occupancy value can be grouped together to create a more lightweight representation.

An adaption especially tailored to the needs in a navigation context is the costmap. Instead of simply tracking the state of a corresponding grid cell, these maps can contain more specialized information. These costs can then be directly leveraged during the path planning by minimizing the cost along the desired path. This could, for example, contain the costs derived from a traversability analysis of the initial occupancy grid map. These relevant costs can then be used directly during path planning operations. Gerkey and Agrawal [30] improved upon this and used it to store information about different types of terrain as well as their traversability with different costs. Sisbot et al. [110] and Kirby et al. [50], on the other hand, used the costmap to model personal space and estimated fields of vision of persons moving through the environment Since the amount of information stored within the costmap continuously increased, Lu et al. [71] proposed to use a layered approach to overcome the limitations of a single monolithic map. Here, the information of multiple instances can be stored in individual costmap layers. In Forouher et al. [Forouher et al., 2016], this was leveraged to fuse the input of multiple different sensor types, such as ultrasound and RGB-D cameras. Lu et al. [70], on the other hand, proposed to use it for tracking people during socially aware navigation. All individual layers can then subsequently be merged by a policy to generate a single monolithic costmap again, which can then be directly used during the path planning process.

All these grid mapping approaches operate on the assumption that all obstacles can be projected down onto their surface polygon. This assumption holds well for easily structured environments. However, even lightly more complex environments can no longer be fully described using the planar projection restriction. As even small, surmountable objects are considered obstacles, the map format quickly reaches its limits. In these instances, additional height-related information is often necessary to plan viable paths in non-planar environments, such as areas with slopes or stairs. Since the projection onto a two-dimensional grid is no longer sufficient, Herbert et al. [36] proposed elevation maps. Contrary to occupancy grids, elevation maps encode additionally the height of the corresponding area in each cell. This way, simple inclinations and obstacles can be included

without increasing the complexity of the underlying data structure. This principle was first used in Bares et al. [2], where a prototype legged rover was built to traverse rugged terrain such as the Martian surface.

In Fankhauser et al. [21], the concept is picked up and extended by adding multiple robot-centric layers of information to the grid. This way, the height and additional information, such as surface normals or traversability, can be stored within the grid cells. With the PlexMap, Heppner et al. [35] proposed a similar approach. Instead of using a regular 2D grid, a quadtree is applied as an underlying data structure to store data more efficiently. Miki et al. [76] proposed a similar elevation mapping framework focused on GPU acceleration. This work included details suitable for the navigation of legged robots, such as plane segmentation and traversability estimation. Based on this, Erni et al. [19] introduced Multi-modal Elevation Mapping. This framework enables robots to store and fuse different sensor modalities, such as color information or segmentation masks, directly inside the elevation map. Elevation maps provide the possibility to describe the world more thoroughly while retaining all the advantages of a two-dimensional representation, such as a small memory footprint and efficient search algorithms. This makes them especially useful while mapping slightly uneven outdoor environments. In Wolf et al. [129] and Wellhausen et al. [127], these elevation maps are used to account for different orientations of the terrain during path- and footstep-planning. Despite their popularity, elevation maps only provide a single surface layer. There is still no information about areas above or below this surface layer. However, these 2D and 2.5D representations often reach their limitations while navigating through complex terrains, for example, multiple-level pathways (e.g., bridges) or multiple-story buildings. In these cases, breaking the environment down into a single map plane is no longer sufficient. Triebel et al. [122] proposed the Multi-Level-Surface (MLS)-Maps to address this limitation. They allow the storage of multiple surfaces in each individual grid cell. In this instance, it becomes possible to model environments containing complex structures like bridges.

Recent advancements in the capabilities of versatile robotic platforms, coupled with their increased accessibility, have driven substantial progress in this field, exposing the limitations of elevation maps. This is especially relevant when operating Unmanned Aerial Vehicles (UAVs) and walking robots during search and rescue missions [102][44], where a comprehensive 3D representation often becomes imperative. Due to the extended movement capabilities, it becomes necessary to use three-dimensional maps to fully describe complex terrain. Even though the MLS-Maps provide a first step in this direction, they reach their limit in more complex applications. As they only describe layers of surfaces, they cannot explicitly represent the free space necessary while navigating Unmanned Aerial Vehicles (UAVs). Furthermore, complex structures often contain many possible surface layers, quickly breaking the maintainability of this map representation. To address these challenges, full volumetric mapping is employed. This modeling technique discretizes the environment into equally sized cubic volumes known as voxels, extending the grid-based approach to three dimen-

sions. However, the additional dimension of the grid introduces a significant drawback regarding the required memory to store a map. Although it is possible to describe the environment in more detail in volumetric maps, these approaches usually come with significant drawbacks compared to the more traditional two-dimensional maps. Due to their increased dimensionality, volumetric maps introduce increased complexity during the creation of maps as well as subsequent navigation tasks. In this instance, it is not only necessary to store the already large quantity of three-dimensional obstacle points but also the surrounding free space. However, contrary to two-dimensional grids, this ambient space, defined by the bounding box of the mapped area, increases cubically for volumetric maps. Thus, mapping large-scale areas or high-resolution maps imposes significant challenges on memory boundaries. As a result, using a conventional three-dimensional voxel grid becomes quickly infeasible. To address this issue, Hornung et al. [40] introduced the OctoMap. Instead of encoding the world as a regular grid, OctoMap leverages octrees to reduce the required memory footprint. Here, the world is split into increasingly smaller octile grid cells until the desired resolution is reached. This way, areas with relevant data can contain high-resolution grid cells while other octiles remain at low resolutions. As a result, the required memory footprint significantly decreases. Similar to the Occupancy Grid Mapping, each of the cells contains an occupancy probability to determine the state of the corresponding area. It is further possible to merge and prune high-resolution cells back to a lower-resolution octile level. This way, free space can be encoded significantly more efficiently in a sparse data structure. With the SkiMap [13], De Gregorio and Di Stefano propose a similar approach. By utilizing a SkipList as an underlying data structure, they could outperform the OctoMap during the integration of dense RGB-D sensor data. The UFOmap proposed by Duberg and Jensfelt [17] presents an extension to the OctoMap. Here, free and unknown space is explicitly modeled to increase the performance of the data structure. More specifically, access times are accelerated by introducing indicators that denote what child nodes are present at each layer. This process is similar to the direct access bit masks of OpenVDB proposed by [83] and enables efficient iterating over the available data, especially while facing large areas of free or unknown spaces. Funk et al. [29] propose a multi-resolution mapping approach that explicitly models free space to enable fast and accurate motion planning for mobile robots. In Hermann et al. [37] GPU-Voxel are proposed. This framework unifies different data structures for direct usage on a Graphics Processing Unit (GPU). Depending on the specific application, it can switch between a voxel map, an octree, or voxel lists to increase the algorithm's performance. A closely related mapping framework is the Spatiotemporal Voxel Layer proposed by Macenski et al. [73], which also utilizes OpenVDB as the underlying data structure. The framework can quickly integrate camera-based range data by utilizing optimized frustum iterators. It introduces voxel decay to handle the dynamic properties of the mapping process. As a result, the consistency of the map is compromised since it constantly discards parts of the map.

3.1.2 Signed Distance Fields

In computer graphics, a technique called Signed Distance Field (SDF) has long been used for representing three-dimensional volumes [31][28]. It describes the distance between a point in space and the nearest surface. If the point is outside the object, the distance is positive and negative when it is inside. The value becomes zero when the point is directly on the object's surface. This way, it is possible to reconstruct continuous distance values from a discretized voxel representation, leading to more accurate position estimates. Curless and Levoy [12] first used this distance function to build offline reconstructions of objects using sensor data. In order to handle computational complexity, they introduced Truncated Signed Distance Field (TSDF), which represents the distance to the closest surface along any given sensor ray. Contrary to the normal SDF, TSDF focuses only on the distances close to obstacles. All distances farther away are, as the name states, truncated. In 2011, Newcombe et al. [92] proposed an approach dubbed KinectFusion. The approach enabled accurate, high-resolution, three-dimensional reconstruction from RGB-D data. It included a SLAM algorithm, which provided a pose estimation of the camera while simultaneously creating the reconstruction. In this instance, the data was directly processed on a GPU in real-time using a TSDF as underlying data representation. In Whelan et al. [128], this approach was extended to allow the reconstruction of significantly larger spaces. Furthermore, Steinbrücker et al. [113] introduced additional extensions that provided the means for online reconstruction based on the CPU rather than on GPU. In Vizzo et al. [124] VDBFusion is presented. The approach utilizes TSDFs modeled in OpenVDB to reconstruct volumetric surfaces.

A closely related data representation is the Euclidean Signed Distance Field (ESDF) Despite also being a SDF representation, it differs significantly in how the distance is calculated. Contrary to the TSDF, ESDFs calculates the Euclidean distance to the nearest occupied voxel instead of the directional distance along a ray. In Lau et al. [60], an approach was proposed to efficiently reconstruct ESDF out of occupancy maps. It leverages small section updates of the environment to significantly outperform previous batch reconstructions of ESDFs in robotic applications. Zucker et al. [134] have used this data representation in motion planning for collision avoidance with complex shapes. Here, the ESDFs are specifically used to find free space areas and to infer distance gradients from present objects. Klingensmith et al. [52] proposed a system to reconstruct dense three-dimensional ESDFs on a mobile device in real-time. In Oleynikova et al. [94] TSDFs are combined with ESDFs to improve the usability of the map in the context of robotic applications. Based upon this, they propose Voxblox [95], which introduces a method to efficiently build ESDFs directly out of TSDFs. By exploiting the already existing distance information present in the TSDF, they were able to build ESDF faster than with Octomap [40]. Furthermore, it could be shown that the resulting ESDFs build from TSDFs was more accurate than when building them from occupancy maps. The framework was optimized to directly work on a CPU to enable On-Board planning on low-powered Micro Aerial Vehicles.

In Reijgwart et al. [101] Voxgraph, a sub-mapping-based derivative of Voxblox is presented. Similar to contributions presented in this work, this was used to gain a globally consistent alignment of sub-maps.

3.1.3 Normal Distribution Transform

An alternative method to model the geometric space is the Normal Distribution Transform (NDT). It was first introduced by Biber and Strasser [5] in the context of 2D scan matching used in relative position tracking and SLAM Takeuchi and Tsubouchi [118] extended this approach to also work for three-dimensional scan matching. However, as stated in Chan et al. [7], this approach is highly prone to numerical errors. Furthermore, it does not incorporate an occupancy update. Instead, Yguel and Aycard [132] propose an approach for updating each cell's distribution by including error-refinement. However, only the occupancy probability of cells containing a Gaussian component is tracked in this approach. As a result, the free space is disregarded. Saarinen et al. [106] introduced the NDT Occupancy Maps to circumvent this issue, enabling accurate real-time 3D mapping in large-scale dynamic environments. One of the main advantages of NDT maps compared to occupancy grid maps is their ability to obtain a similar accuracy while using a much lower cell resolution [116] However, this comes at the cost of increased computational complexity, which can impact crucial real-time capabilities.

3.2 Path Planning

The task of autonomous path planning for mobile robots is a crucial field in current research. The topic has been highly discussed in multiple surveys such as Qin et al. [99], Liu et al. [66], and Sanchez et al. [107]. Furthermore, a higher research interest in high-dimensional spaces and cluttered environments has emerged in recent years. Yang et al. [131] present a thorough study of currently available three-dimensional path planning approaches. They classified these methods into five categories based on their capabilities. These methods include various algorithms directly applied for AGV, UAV, and AUV. A study specialized in path planning and obstacle avoidance of underwater robotics was presented by Cheng et al. [8]. They provide an overview of recent advances in the field of three-dimensional path planning under the influence of marine environments. On the other hand, Loganathan and Ahmad [68] focus their study more on autonomous mobile robots. They classify planners based on their capabilities and divide the search space into 2D versus 3D, single versus multi-objective planning, and dynamic versus static environments.

3.2.1 Graph Based Methods

Graph-based path planning presents a fundamental component in robot navigation. It is one of the oldest and most researched fields in autonomous navigation of mobile robots. Here, the path-planning problem is transformed into a graph representation. The goal is then to find the shortest or most cost-efficient path from a starting node of the graph towards a goal node. As this is closely related to graph theory in general, an excessive number of possible search algorithms exist. One of the most fundamental approaches is Dijkstra's search algorithm [16]. The algorithm performs an exhaustive iterative search in the graph until the goal node is reached, or all present nodes have been visited. Since the algorithm is quite similar to a breadth-first search with included edge weights, it is guaranteed that an optimal solution will be found if it exists.

In Hart et al. [34], the A* algorithm was introduced. Here, the principle of Dijkstra's algorithm was extended by including a heuristic function. In Dijkstra's algorithm, the goal was only to minimize the path already traveled toward the goal. Contrary to this, A* minimizes the sum of the path and an additive heuristic function. In this instance, this heuristic estimates the remaining path toward the goal. As a result, the search direction is more focused towards the goal, leading to a faster but still optimal result.

In order to handle the path-planning problem for partially known environments, Stentz [114] introduced the D* algorithm. The algorithm resembles an A* but is able to handle dynamic cost changes within the graph representation. This way, it becomes possible to incrementally determine the shortest path from the current position toward the goal while the robot traverses the path. In D*, the path is conversely calculated from the goal node toward the robot. This way, it is possible to repair paths to the robot once new information is included in the map. As a result, the robot is able to cope with unknown, partially known, and dynamically changing environments. Similar to the A*, it provides an efficient and cost-optimal result. To improve upon this in Stentz [115], the focussed D* was proposed. This extension to the initial D* focussed on improving the path reparation process to significantly reduce the initial path calculations as well as all subsequent replanning operations.

Similar to this, Koenig et al. [54] proposed Lifelong Planning A*, which presents an incremental version of A*. It repeatedly searches for the shortest paths from any given start vertex towards a goal given changing graph edge costs. For searching the initial plan on which the subsequent replanning operations are based, a traditional A* is used. The resulting costs of this initial search are then updated and, where applicable, re-used. In Koenig et al. [53], this concept was further improved and applied to the domain of goal-directed navigation in unknown terrains.

Graph-based path-planning approaches in robotics are often tightly coupled to grid-based representations. Here, the individual cells are used as nodes, and the

edges connect them to all neighboring nodes. However, due to the grid-based structure, the angle between these neighborhood connections is restricted to multiples of $\frac{\pi}{2}$. Therefore, the calculated path is not necessarily the most cost-efficient path. Despite being optimal in a grid-based sense, the truly shortest path would require the plan to take intermediate angels. This problem is solved by the class of any-angle path-planning algorithms [89]. An example of this is Field D* proposed by Ferguson and Stentz [25], which is an improved version of the initial D* capable of any-angle angle path planning. Theta* by Nash et al. [88] also provides these abilities by propagating information along grid edges without constraining the generated path to these edges. However, despite being able to find short paths, it does not guarantee to find the optimal path. Lazy Theta* proposed by Nash et al. [91] offers an improved version of the Theta* capable of handling 2D and 3D environments. This was achieved by restricting the line-of-sight check to a single operation per expanded vertex. As a result, paths can be found faster than the original Theta* without any increase in path length. Phi* [90] provides an incremental version of the basic Theta* algorithm. The approach utilizes the free space assumption proposed by Koenig et al. [55], which states that the robot can move from any start vertex to a goal cell without knowing the occupancy state of all traversed cells beforehand. This way, it was possible to speed up the Theta* by one order of magnitude.

3.2.2 Sampling Based Methods

Sampling-based planners have gained huge popularity and a wide distribution in planning tasks in high-dimensional spaces or cluttered environments [131]. This type of planner is based on randomly sampling points from the planning space and verifies that they and the path leading towards them are not in a collision state. One of the first sampling-based methods was the Probabilistic Roadmap (PRM) proposed by Kavraki et al. [49]. This planner builds a roadmap in the configuration space of collision-free trajectories between randomly sampled points. After building this roadmap, graph-based search methods such as Dijkstra or A* can be used to find the optimal path within the roadmap. As this graph has to be built only once, this approach is predominantly used for multiple-query tasks, where the same configuration space is used for multiple path requests. Analysis of the algorithm presented in Kavraki et al. [48] even showed that the approach is probabilistically complete given a large enough set of samples. However, due to its sampling-based nature, the algorithm is not guaranteed to produce an optimal path. To alleviate this issue, Karaman and Frazzoli [47] proposed PRM* an optimal version of the basic PRM. However, in tasks such as online path planning of mobile robots, multiple-query methods come with severe disadvantages. In this instance, constructing an exhaustive roadmap for multiple queries is unnecessary, as the environment constantly changes over time. Therefore, the roadmap can no longer be re-utilized for subsequent planning requests. To solve this issue, multiple incremental probabilistic algorithms have been proposed. In 1997, Hsu

et al. [43] proposed Expansive Space Trees (EST), which only sample the portion of the configuration space that is related to the current query. As a result, the roadmap generation could be avoided. Hsu et al. [42] propose to generate a new roadmap for each query by connecting the initial and goal state points via short admissible trajectories.

One of the most prominent incremental sampling-based path-planning algorithm is the Rapidly-Exploring Random Tree (RRT) proposed by LaValle [61]. Contrary to the PRM, RRT solves the motion planning problem incrementally as a single-query operation. It continuously samples new points from the state space and verifies if they can be connected via a collision-free state transition. This way, a rapidly exploring tree is built from these randomly sampled points, hence the name of Rapidly-Exploring Random Trees.

This original version of the RRT algorithms has been extensively used in various robotic domains. Frazzoli et al. [26] proposed to use it for real-time motion planning of autonomous vehicles. The work focused on the dynamic constraints on vehicle motion during path planning in the presence of fixed and moving obstacles. In Bhatia and Frazzoli [4], it was used for reachability analyses for continuous and hybrid systems. Cortes et al. [11] even used a derivative of RRT to calculate ligand passageways inside of proteins during molecular disassembly. Despite this approach's biological background, the authors state that the concepts behind the algorithm can also be applied to disassembling articulated mechanical parts. Furthermore, Zucker et al. [133] proposed multipartite RRT, which supports planning in unknown or dynamic environments. By restricting the sampling distribution and re-using partial trees of previous planning iterations, the dynamic motion planning capabilities of the RRT could be improved further.

Due to its wide distribution, RRT gained much attention in the research community and was therefore extended in several directions. Plaku et al. [98] proposed the Sampling-based Roadmap of Trees(SRT), which fuse the core properties of multiple-query approaches (e.g., PRM) and single-query algorithms (e.g., RRT and EST). To decrease the necessary runtime to find a solution, Kuffner and LaValle proposed RRT-Connect [59]. In this derivative, two RRTs are built up individually. One is rooted at the start- and one at the goal configuration. During the search process, both trees explore the space around them and simultaneously advance toward each other. This is achieved by applying an additional greedy heuristic bias during the random exploration of new samples.

Despite the fact that sampling-based approaches provide fast path-planning solutions in high-dimensional problems, their results are still not guaranteed to be optimal. However, the overall quality regarding path length or execution time of the solution is, in most instances, crucial. Urmson and Simmons [123] proposed to use heuristics during the creation of the RRT. This way, the generated path has a higher bias toward low-cost solutions. In Ferguson and Stentz [24], the RRT was executed multiple times. This way, the solution was progressively improved. Even though this approach was not guaranteed to converge against

an optimal solution, it still resulted in lower path costs. To gain an optimal path from an RRT, Karaman and Frazzoli [47] proposed RRT*. The algorithm has been proven to asymptotically converge against the optimal solution. In Klemm et al. [51], the concepts of RRT-Connect is combined with the optimality of RRT*. This combined algorithm dubbed RRT*-Connect is able to find solutions faster than RRT* while still converging towards the theoretical optimum.

4 Volumetric Mapping

Mobile robots are becoming increasingly versatile and capable of executing various tasks autonomously. As a result, there is also an increased interest in achieving a higher level of autonomy in all their areas of expertise. One of the most essential skills an autonomous mobile robot must possess is its navigation capability. More specifically, the skill to find and execute a collision-free path from its current position to a target location in which it has to perform a task.

The most crucial requirement to achieve this is for the robot to be aware of its surrounding area. Most robots are equipped with multiple active and passive sensors to record their operating environment. However, relying only on the most recent measurements of these sensors is prone to failure, as current sensor readings only cover a small portion of the operation area. As a result, large-scale

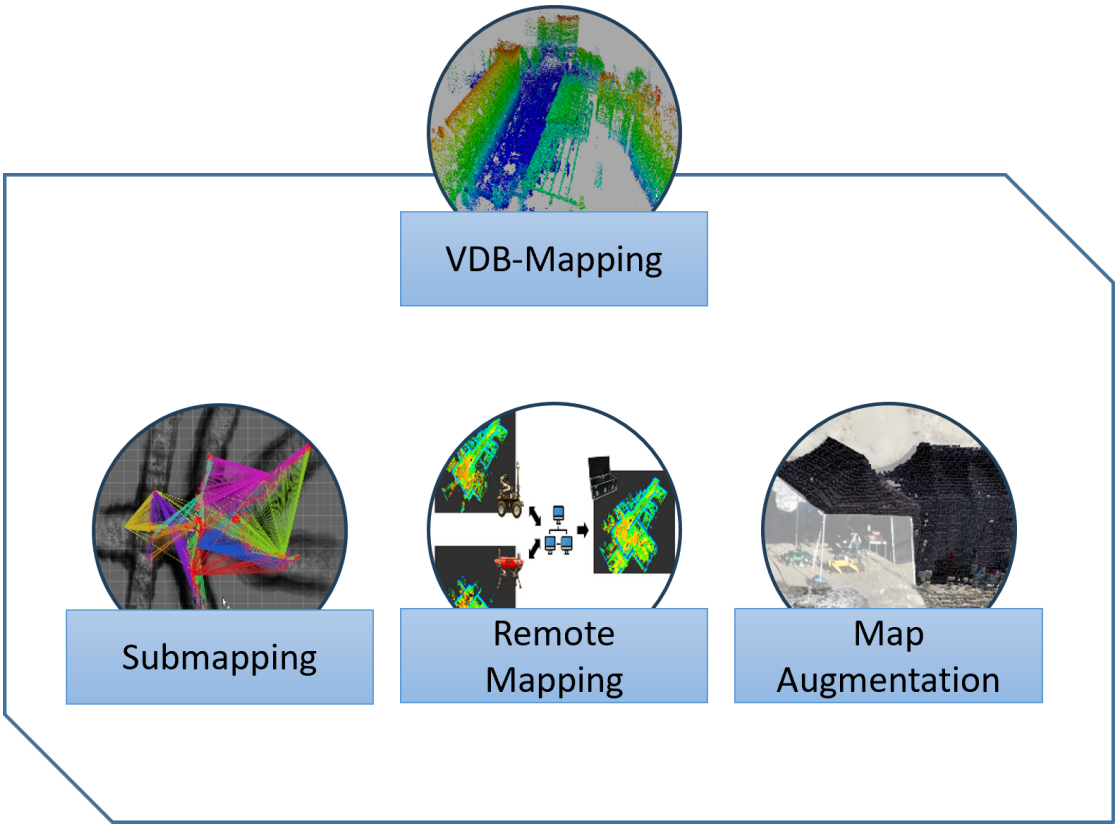


Figure 4.1: The different contribution to volumetric mapping presented in this Thesis.

operations like planning a path to a far-off target become infeasible only on local data. Furthermore, the sensor data is prone to noise and occlusions, which might lead to faulty decisions during path planning and execution. Instead, the navigation is commonly performed on a map representation, accumulating multiple sensor readings. Depending on the intended use case, these representations can come in various forms and dimensionality. Current navigation approaches commonly project the environment onto a 2D or 2.5D plane. However, this reduction is no longer applicable as the terrain gets more complex. Instead, all the information of the three-dimensional world has to be considered. Recent developments in versatile mobile robots have also made volumetric map representations covering three-dimensional space more enticing. However, due to the massive amount of data the sensors produce, it quickly becomes infeasible to simply accumulate all generated data points. Instead, it becomes necessary to abstract the data into a more usable and efficient format. A suitable data structure has to satisfy the following requirements to be used for navigation purposes:

- **Memory efficiency:** The map representation must be able to store and merge vast amounts of data to cover large-scale environments.
- **Integration speed:** To generate an up-to-date map representation, the data structure must be able to process and incorporate the incoming sensor data immediately. Otherwise, measurements might be dropped, and the navigation would be required to operate on incomplete or stale data. Consequently, this might lead to unintended collisions or faulty decisions.
- **Data handling:** The data must be in a representation that enables path planning algorithms to quickly access and perform efficient calculations.

This chapter gives an in-depth overview of the volumetric mapping developed during this work. In Section 4.1, a detailed description of the developed mapping process and utilized data structure is given. The subsequent Sections provide insights and explanations about the various extensions to the basic mapping structure. First, the sub-mapping module is introduced in Section 4.2, which enables the handling of time-varying position data. Afterward, the remote mapping capabilities are described in Section 4.3, which can be utilized to exchange maps between various systems efficiently. Section 4.4 covers how the map can be augmented with arbitrary data to help store more information in each voxel. The chapter concludes with a brief summary of the presented approach in Section 4.5.

4.1 VDB-Mapping

To enable the robot to use all the volumetric information the world offers, the first step is to provide an efficient 3D map representation of the environment in which it operates. A first naive approach would be to simply accumulate all gathered

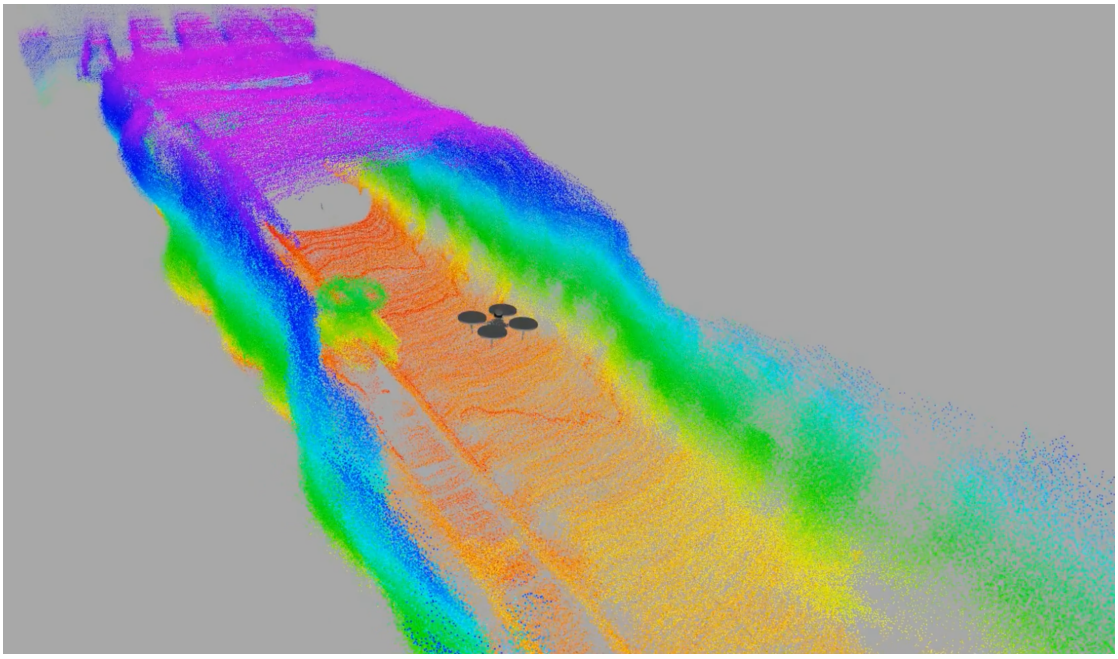


Figure 4.2: Raw sensor point cloud data generated by a LIDAR. The map was recorded using a UAV inside an old iron mining complex. The sensor produces around 300.000 data points per second, which quickly accumulates to an infeasible amount of data, both memory and handability-wise.

sensor data and stitch them together into a single high-resolution map of the environment. As stated in Section 2.1.2, this process entails significant disadvantages, especially in the context of robot navigation. First and foremost, typical 3D sensors generate vast amounts of data. For example, a Velodyne VLP-16 [45] generates around 300.000 points per second. A resulting raw sensor data map recorded on a UAV can be seen in Figure 4.2. Accumulating them all without further consideration would quickly break the data memory boundaries of the system. This effect is even further reinforced, given that the sensor scans most surfaces multiple times, leading to a vast amount of redundant data points. Additionally, the unordered nature of the accumulated points would not provide a suitable representation in the context of path-planning algorithms, where it is often necessary to check neighboring data points for traversability. Since this neighborhood is not directly apparent from the accumulated data, an additional ordering or search step would be necessary. Additionally, the amount of neighbors to even consider would be computationally exhaustive. The main goal of an efficient map representation in the context of robot navigation is to collect all recorded sensor data in a more manageable format. Therefore, it is necessary to abstract the gathered information into an efficient data structure. To achieve this goal, in the course of this thesis, the VDB-Mapping [Grosse Besselmann et al., 2021] framework was developed, which uses OpenVDB [83] as an underlying data structure.

This section introduces the utilized data structure and how its properties were

used to create an efficient volumetric mapping framework for subsequent navigation tasks. At first glance, it might be sufficient to integrate each recorded sensor measurement directly into the respective grid cell of the tree data structure to generate a map. This way, all grid cells corresponding to an area of the environment get a state assigned, and any obstacles the sensors observe are correctly integrated into the map. In a simplified static environment without dynamic obstacles or sensor noise, this approach would indeed be adequate. However, during operation in a real-world scenario, the environment often continuously changes over time. These changes range from dynamic parts of the environment, such as opening and closing doors, to moving obstacles, such as persons or other robots operating in the same area.

Integrating sensor data of a dynamic object would result in the correct insertion of the corresponding voxel into the map. However, using this simplified insertion scheme, the data point can no longer be cleared out of the map if the state or position of the obstacle changes. A visualization of this problem is observable in the map depicted in Figure 4.3, in which a person has moved through the environment. It is evident that a trail of its former positions follows the person, leading to falsely occupied areas considered impassable for a navigating robot. Thus, instead of just plainly inserting all acquired map data, keeping track of all free space between the obstacles is also necessary to modulate dynamic obstacles. Given the physical properties of commonly used range sensors in robotics, such as LIDAR or RGB-D sensors, it can be assumed that the entire trace from sensor origin to the first reflected obstacle is free space. Therefore, by using ray casting throughout the data structure for each sensor data point, all free voxels can be easily discerned and reset if they are already marked as an obstacle in the map. To efficiently gain this information, the presented mapping approach relies heavily on the optimized ray casting capabilities of the OpenVDB data structure presented in Section 2.3.4. This process is combined with an underlying probability framework for each voxel to determine its occupancy state as a probability function. As a result, the generated map is less susceptible to sensor noise and better capable of handling dynamic objects. The entire integration of new sensor data is split into a two-step process. The first step is the creation of a temporary update grid, which is co-aligned with the actual map. This grid is intended to contain an accumulation of all voxels marked for an update by the last few sensor measurements, whether it is free or occupied. After each measurement or a specific timeframe, depending on the desired usage, the update grid is subsequently integrated into the actual map.

4.1.1 Data Structures

As mentioned above, the entire mapping framework is built around the VDB data structure introduced in Section 2.3. This data structure highly synergizes with the proposed volumetric mapping approach, as it combines high performance with

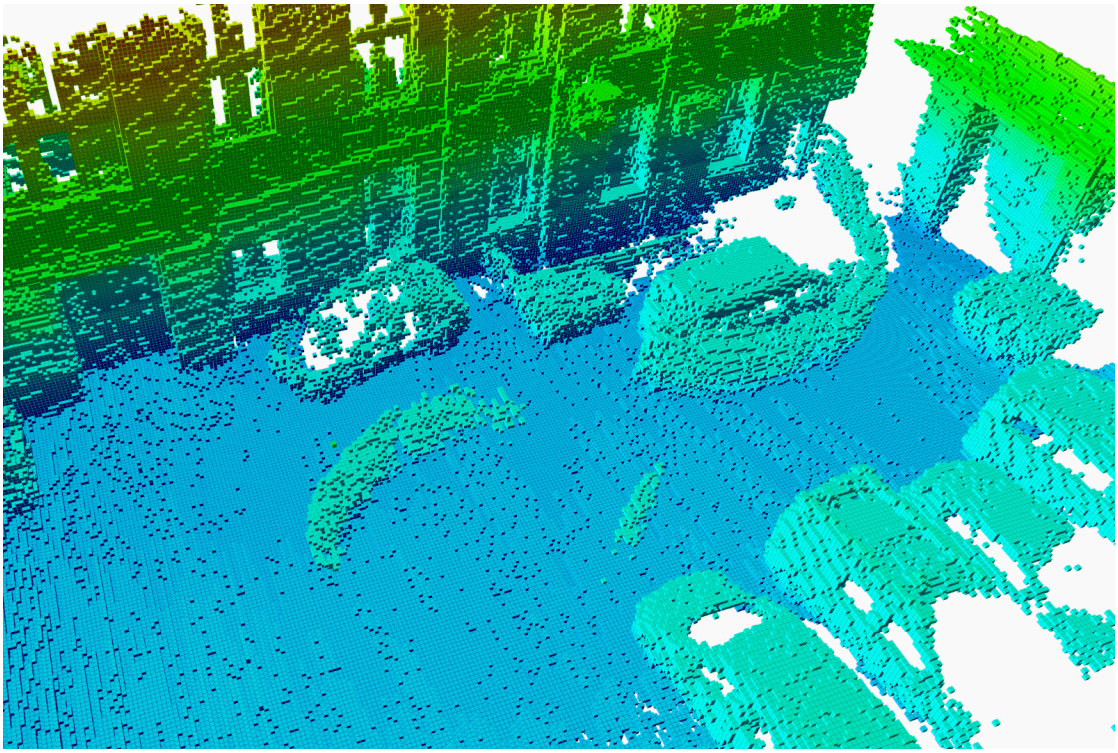


Figure 4.3: Map of a parking lot in which all data points are simply inserted without ray-tracing the sensor data. It can be seen that this process is unusable to handle dynamic map properties. Since all voxels are only inserted into the map, the human walking through the area can be clearly observed by the trace his body leaves on the map. Without ray-casting, these now free voxels are never removed from the map representation.

easy adaptability to various use cases. The efficient data handling and integration is one of the key features, as it prevents sensor data from being discarded. Given a slower framework, sensor data might not be processed at the same time the sensor captures it. This would lead to dropped sensor data, which will not be incorporated into the map. Resulting from this, the generated map would exhibit inaccuracies such as holes in walls, incomplete surfaces, or missing sections. Therefore, a fast integration speed is a crucial requirement for autonomous robot mapping. The second leveraged property of the data structure is the capability to configure it in detail. By adjusting the data type, height, and branching factors of the tree, it can be tailored to multiple specific use-cases.

In the developed framework, two distinct types of VDB-Grids are used during the mapping process. The first one is the map grid, which stores all values accumulated in the actual map. In its basic form, the map stores a single 32-bit floating point value on its leaf layer. This value typically encodes the occupancy probability of the corresponding area. However, as described in Section 4.4 using an adapted version, it is possible also to store arbitrary additional information. The tree's branching factors are configured in the default OpenVDB setup

$log_2 w =< 5, 4, 3 >$ described in Section 2.3.2. More specifically, this means each level from the bottom up has a higher branching factor to cover a higher spatial index area. The tree has four layers and is built up from a single root node, followed by two internal and one leaf layer. Given the specified branching factors, the internal and leaf layers have the following configuration:

$$internal_1 = 2^5 = 32 \tag{4.1}$$
$$internal_2 = 2^4 = 16 \tag{4.2}$$
$$leaf \quad = 2^3 = 8 \tag{4.3}$$
$$\tag{4.4}$$

By multiplying each layer of the tree, each node in the root contains 4096^3 individual voxel. This configuration was chosen as it combines fast processing capabilities with a small memory footprint.

Contrary to this, the update grid has a different purpose. As described below in Section 4.1.2, its intention is to update only the small area currently covered by sensor updates. Additionally, it is later utilized in Section 4.3 as a transferable update grid. In both instances, a high focus on speed and memory efficiency is paramount. To increase the grids' performance three specific configurations have been chosen. First, the data type of the update grid was reduced to a boolean value to reduce the size of each individual voxel. Additionally, as described in Section 4.1.2, the grid only contains value nodes on occupied leaf voxel. All other updates are encoded only in the direct access mask of the update grid. The last adjustment to the update grid is the reduced branching factors. For the update grid, the following $log_2 w =< 1, 4, 3 >$ configuration has been selected:

$$internal_1 = 2^1 = 2 \tag{4.5}$$
$$internal_2 = 2^4 = 16 \tag{4.6}$$
$$leaf \quad = 2^3 = 8 \tag{4.7}$$
$$\tag{4.8}$$

This leads to a root node covering 256^3 voxels. At a resolution of $0.05m$, this results in a root node size of $12.8m$, which was chosen as it covers most close-range sensor measurements. It also provides a high speed and suitable memory usage since the overhead of a larger first internal layer is reduced.

4.1.2 Update Grid Generation

As stated before, it is not sufficient to simply insert the end-points of each sensor measurement into the map since it would not be possible to represent dynamic obstacles this way. Modulating the free space between the sensor origin and the obstacle is also necessary, especially for the subsequent path planning. In order to achieve this, the physical properties of ranging sensors such as LIDAR or RGB-D cameras are leveraged. Commonly, they return the distance of the first reflected

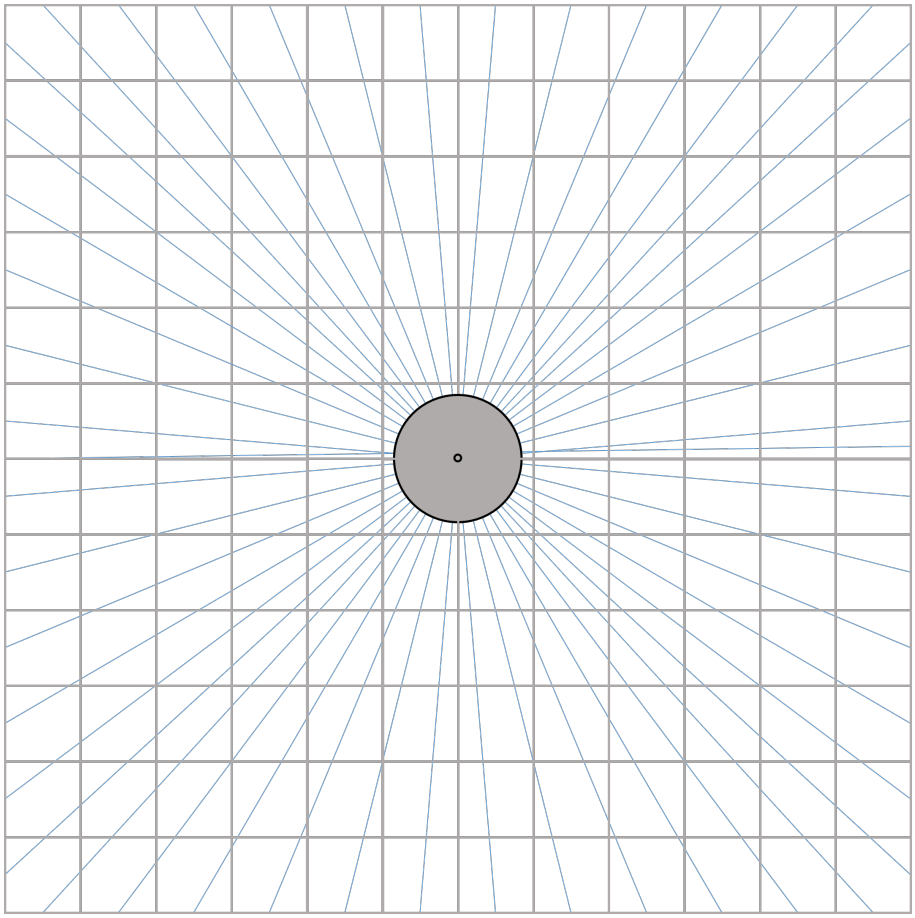


Figure 4.4: LIDAR beams passing through a discretized grid structure. It can be seen that close to the sensor origin, multiple beams pass through the same voxel. This, in turn, has negative consequences for the probability framework underlying the mapping process.

obstacle they encounter in a specific direction. Therefore, it can be reasoned that all space along the straight line created by the sensor's origin and the range measurements endpoint is unobstructed. Given this information, the initial step in the data integration process is to perform a ray casting for each 3D data point within the map data structure. This way, all voxels relevant to each individual sensor measurement can be discerned. To find each discretized voxel along the line within the grid, a technique called Digital Differential Analyzer (DDA) [84] is applied. The intuitive approach would be to directly update the occupancy state of each traversed voxel as the DDA moves along the ray. However, this leads to two severe problems.

The first problem is that all rays close to the sensor origin are positioned relatively close to each other, as can be seen in Figure 4.4. As a result, all voxels near the sensor origin are traversed significantly more often than the ones farther away. If each time the full probability update of the voxel in the map is performed, the probability of these close voxels would be directly pushed firmly to either free

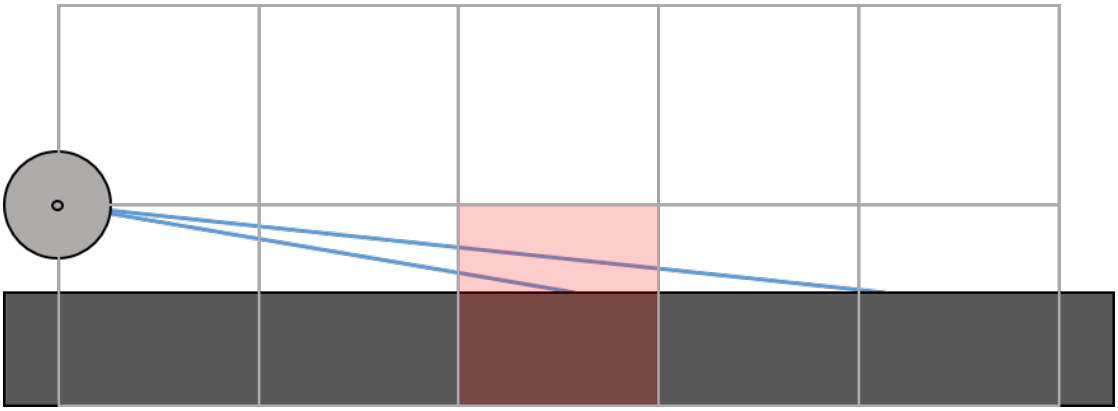


Figure 4.5: Due to the discretizing of the environment, the LIDAR produces contradicting information. The two depicted range measurements traverse the same voxel highlighted in red. However, the lower beam states that the cell is occupied while the upper beam passes through it and states that it is a free cell. This contradicting information has to be resolved during the mapping process.

or occupied. As each sensor measurement quickly adds more certainty to the voxel's occupancy state, the overall probability dynamics of the map structure would be impaired. This is mainly due to the fact that an equal amount of contradicting measurements would be required to push the voxel occupancy back to the opposite state. As a result, dynamic changes in this area would take longer to affect the state of the corresponding voxels, leading to a slower integration of new sensor information.

The second problem stems from discretizing the space into a grid of voxels with a fixed resolution. Due to the necessary discretization, two adjacent sensor measurement rays sometimes provide contradicting information about the same voxel. An example of this issue is evident in Figure 4.5. In this scenario, both rays traverse the same voxel. However, the first passes right through, whereas the second hits an obstacle on the floor. This results in two contradictory states, which are both true due to the discretization. Therefore, the algorithm has to decide which of both statements to trust and how this information will be applied to the map. However, if all rays were directly inserted and updated into the map, this effect would not be detectable since each sensor ray is processed separately.

To circumvent these two issues, in the developed framework, the data of all sensor rays is gathered prior to integrating them into the map structure. Instead of directly inserting the data into the actual map, a temporary update grid is created beforehand. The spatial location of the index space of this grid is co-aligned with the grid of the actual map This enables the usage of the exact same coordinates without the necessity of spatial transformations between both grids. The main idea is that the update grid contains a reference to each voxel that needs to be adjusted during the next update step of the mapping process. As before, a ray casting is performed for each sensor ray to find all traversed voxels along the

ray's path. However, instead of directly updating an occupancy probability, the update grid just tracks whether the voxel is marked to be freed or occupied.

To optimize the generation of the update grid, specific properties of the OpenVDB data structure and its direct access bit mask described in Section 2.3.2 are leveraged. These bit masks provide a fast overview of the topology of the underlying tree structure in which the data is held. By modifying only the direct access bit masks, efficiency is magnified, resulting in savings in both computation time and memory efficiency. As a first step, the bit entry in the value mask of each traversed cell is marked as active. The marking process ensures the ability to quickly distinguish all relevant voxels from the embedding space not currently within the sensor coverage. Using this technique, the update grid creates a grid subspace in which each relevant voxel is directly accessible through fast iterators.

However, until this point, it is not directly possible to differentiate between occupied or free cells. To make the entries in the update grid distinguishable, for each endpoint of the ray (i.e., occupied voxel), a new leaf value node has to be added to the VDB-tree, instead of just modifying the bit mask. This way, minimal memory overhead is introduced into the system. Despite this, the amount is negligible since the number of occupied cells is generally significantly smaller than that of free cells. However, being able to clearly distinguish free from occupied cells during the batch processing of all beams enables the algorithm to prevent contradicting data and remove it from the update grid. This is done by checking if a visited node is already marked as occupied and has a value assigned to it. If this is the case, the ray cast is no longer able to overwrite this voxel's state with a free state. In conclusion, each node visited by the batch ray casting triggered by the sensor measurement is set to active, which enables traversing them using fast iterators. Furthermore, it is possible to clearly determine from its value, or its absence, whether a node should be updated as free or occupied. After the batch processing of each sensor beam is finished, the actual map can be updated.

4.1.3 Data Accumulation

Parallel to the contradiction of individual beams, similar problems arise when the integration of individual sensor measurements is considered to be temporal-independent. This effect is often caused by the structure of standard LIDAR sensors, as seen in Figure 4.6. The data pattern consists of multiple concentric circles around the origin, which is caused by the placement of the rotating laser mirrors inside the sensor. Depending on the angle of departure and the reflecting surface, the angle towards a surface might be relatively flat. At the same time, the space between the individual concentric circles is not covered by the sensor and, therefore, is not marked as an obstacle.

While the problem for individual rays can be mitigated using batch processing described above, similar problems arise when considering each complete measurement individually. The main problem occurs when facing flat incidence an-

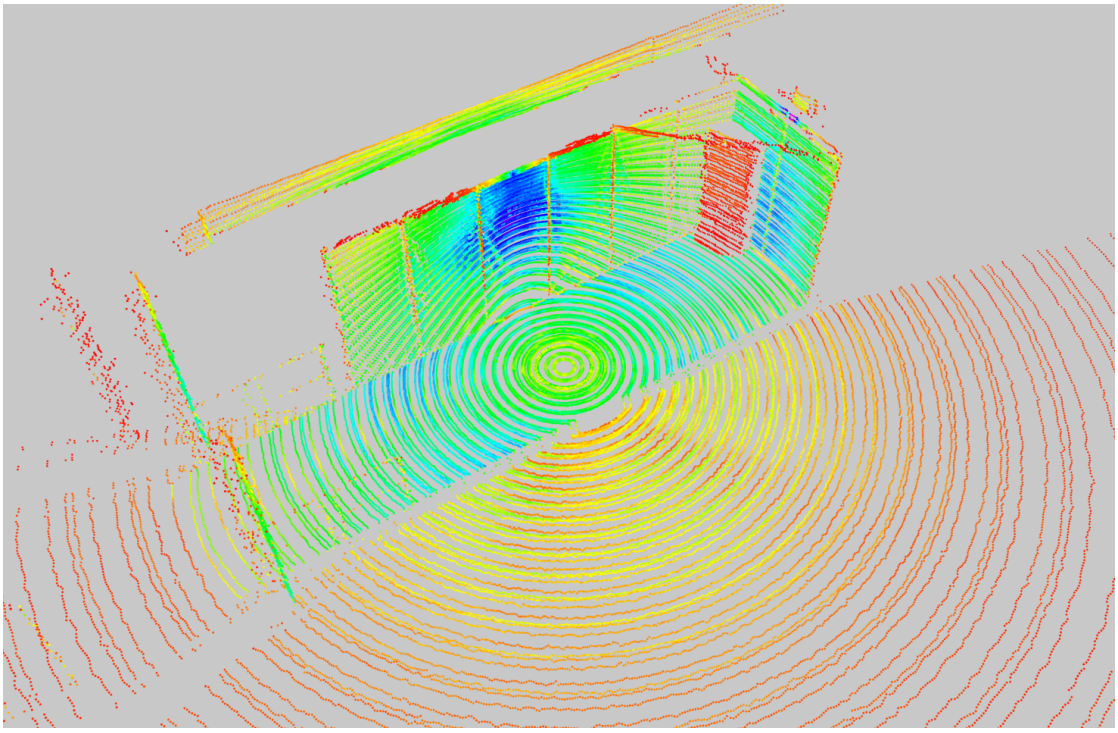


Figure 4.6: The typical geometric structure of a LIDAR scan. The concentrically spaced circles can reduce the mapping performance.

gles where two adjacent segments of the concentric circle are far apart. As can be seen in Figure 4.7, the left and right voxels are correctly marked as occupied. Due to the gap in the sensor measurement, the middle voxel, on the other hand, is falsely considered free space. As a result, voxels within the gaps of the concentric circles are cleared out of the map, leading to an inaccurate map representation.

To counteract this problem, the developed VDB-Mapping framework provides the alternative of not directly integrating each update grid into the actual map. Instead, measurements are accumulated over a user-specified time frame. During this time, the update grid will be filled using multiple sensor measurements from the same or different sensors. This way, given that the robot moves in-between measurements, a more dense update grid is created, in which the gaps between concentric circles are filled with actual information. After each accumulation period, the update grid will be integrated into the map as before and subsequently reset.

Even though this fixes the problems caused by the sensor data structure, it introduces a trade-off between the accuracy and responsiveness of the map. This is caused by the slower integration speed due to the delay introduced by the accumulation period. Therefore, the accumulation period can be freely configured during runtime to adjust the mapping to the current use case.

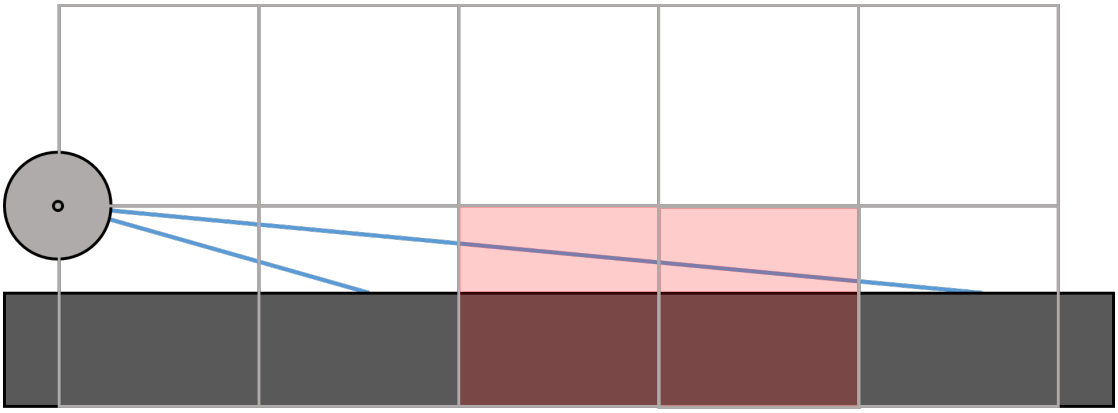


Figure 4.7: The geometric structure of modern LIDAR often consists of several concentric circles around the sensor's origin. This structure, combined with the discretization, produces false state estimates of individual voxels. In this instance, the upper beam states that both red-marked grid cells are free since they can be passed. Since the next closest line is farther away due to the gap between the LIDAR lines, no contradicting data is generated. As a result, the surface area would be cleared even though there is ground present.

4.1.4 Probability Update

After generating the update grid, the last step of the mapping process is to integrate the update grid into the map. For this, each update grid's accumulated voxels are iterated, and the probability of the corresponding map voxel is updated accordingly. To efficiently iterate over all relevant voxels of the update grid, the direct access bit mask of the VDB-grid is utilized. As described above, within the update grid, each voxel can have one of three states The first state is present when the voxel position in the direct access bit mask is inactive. In this case, the voxel was not traversed by any sensor beams. Therefore, it is irrelevant during the update step and can be safely ignored. For the second state, the voxel is set to active in the bit mask but has no associated value pointer for the voxel's location. This means that the voxel was free in all last measurements, and thus, its occupancy probability should be decreased during the update process. The last state is present if the voxel is active and has an assigned value leaf node in the VDB-tree. In this case, the occupancy probability has to be increased to mark the map's corresponding area as an obstacle. This reduced, memory-efficient bit mask encoding enables the algorithm to quickly find all active voxels that must be updated towards either a free or occupied state. Integration efficiency is also increased by the fact that no spatial transformation between the grids is necessary. Since both grids are co-aligned, all necessary indices can interchangeably be used in each grid. To update the probability, the occupancy grid mapping formulation in logit representation described in Section 2.1.2 is used.

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + \log\left[\frac{P(n|z_t)}{1 - P(n|z_t)}\right] \tag{4.9}$$

The advantage of this formulation is that it is not restricted to a range from zero to one. Instead, it can grow indefinitely, thus preventing numerical instabilities close to a one and zero occupancy probability. The concrete probability can be retrieved using:

$$P(n|z_{1:t}) = 1 - \frac{1}{1 + e^{L(n|z_{1:t})}} \tag{4.10}$$

Next to the probability, the occupancy state is also tracked for each voxel to ease the use of the map. For this, a simple thresholding of the probability is done to determine the current state. Each voxel can have one of three states: free, occupied, or unknown. The occupied and free state is self-explanatory and will be set depending on whether the occupancy probability lies outside the upper or lower threshold boundaries. The unknown state represents the case in which insufficient data about the voxel was gathered since not enough sensor measurements traversed it. Thus, it is assigned if the occupancy value neither exceeds nor subceeds the upper or lower threshold.

To efficiently encode the state in the map, the direct access bit mask is leveraged here as well. If the occupancy probability exceeds the specified upper threshold, the corresponding voxel is set to active in the bit mask. This accessible representation of the voxel states makes it easier for subsequent navigation algorithms to distinguish between free and occupied space in the environment. On the other hand, the differentiation between free and unknown space is not directly distinguishable since only a single value can be efficiently encoded in the bit mask. Therefore, in this case, the probability has to be manually accessed and checked against both thresholds.

4.2 Sub-Mapping

To utilize the generated map in a navigation framework, the representation of the surrounding area has to remain precise, even in large-scale environments. During the mapping process of these large-scale environments, small sensor and position inaccuracies accumulate over time, resulting, over time, in a more significant error. This problem is commonly compensated using a sophisticated SLAM system on top of low-level position estimation techniques, such as wheel odometry. As described in Section 2.1.1, all sensor and position estimates are gathered in a graph representation. This graph is continuously optimized to find the most consistent spatial configuration to adhere to all constraints given by the estimates. However, not all of these constraints are of equal importance. The so-called loop closure presents a special and far more critical constraint. These constraints are used if a known area of the environment is revisited and consequently recognized. While arriving at the previously explored area, these constraints are added to the graph, connecting the old and new observations by an edge. During the graph optimization, these constraints possess a significantly higher edge weight, which increases their influence on the spatial alignment of graph nodes. After

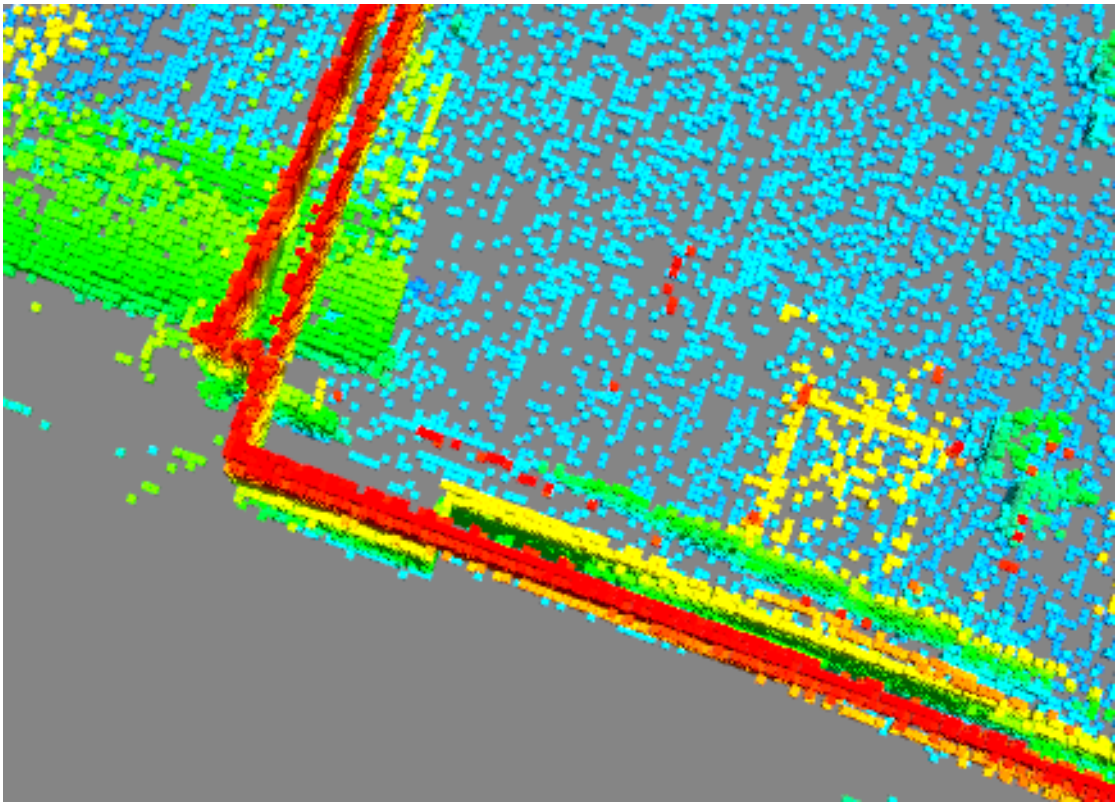


Figure 4.8: Effect of retrospective pose corrections on a monolithic map representation. As the pose estimate of the revisited area no longer overlaps with the first visit's estimate, errors are introduced to the map. In this instance, the wall measurements no longer overlap correctly, leading to problems with subsequent navigation tasks. [Grosse Besselmann et al., 2023a]

adding them to the graph, the SLAM tries to align both spatial locations over each other, resulting in a possibly massive shift of all in-between graph nodes. As a result, the poses of the robot and all recorded sensor data are changed in retrospect.

In the developed mapping approach, all sensor data is usually gathered into a monolithic map of a fixed resolution. All available sensor data is discretized, merged into individual voxels, and tightly integrated into the map grid. This process leads to a small memory footprint since not all raw sensor data must be stored during the mapping process. Even though this approach provides multiple advantages, it has a significant drawback. During the discretization and merging process, the original reference frame in which the sensor data was collected is lost. This loss of information, in combination with the retrospectively adjusted sensor poses, represents a considerable problem for the accuracy of large-scale maps. Since the initial pose error accumulates over time, new sensor measurements are integrated at possibly incorrect locations in relation to the actual map coordinate system. As a result, the subsequent pose corrections of

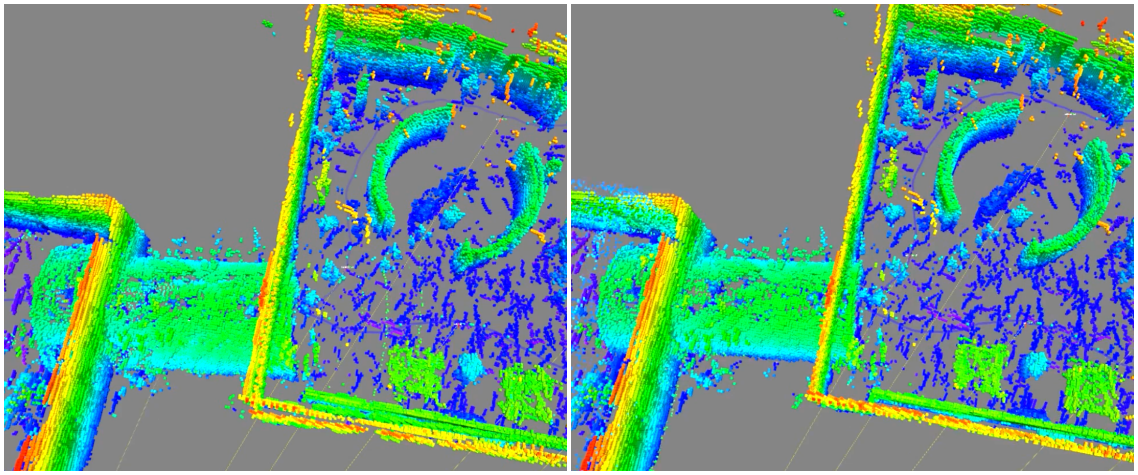
the SLAM loop closure introduce significant inaccuracies in the discretized map. This problem becomes especially apparent while revisiting already explored areas, where previously observed structures such as walls might no longer align or overlap due to position estimate inaccuracies, as seen in Figure 4.8. Since, at this point, the data is already tightly integrated and interdependent, the sensor data cannot be shifted after correcting the robot's location through loop closure. As a result, a monolithic mapping formulation is not well suited when confronted with continuously changing pose estimates.

4.2.1 Sub-Map Integration

To address the issue of misaligned maps in large-scale environments, the developed mapping framework provides the ability to utilize smaller, locally consistent sub-maps for the mapping process. These small sub-maps are dynamically repositionable and interconnected by the SLAM pose graph. During the SLAM optimization process, the spatial re-alignment of the graph nodes implicitly influences the alignment of all tightly interlaced sub-maps as well. Given all individual sub-maps and their corresponding relative position to each other, a complete representation of the map can be reconstructed.

For the sub-mapping module, it is assumed that the estimated robot pose is at least locally consistent. More specifically, this means that the accumulated pose estimation error does not exceed the abstraction error of the underlying grid structure within a small area of the environment. As a result, it is possible to generate small, locally consistent sub-maps unaffected by the SLAM optimization and retrospective pose corrections. While this assumption helps on a small scale, large-scale drift and the subsequent pose shift still present a challenge during the mapping process. Due to the discretization process, it is not easily possible to retrospectively change each sensor data point after it has been integrated into the map. Instead, the entire network of locally consistent sub-maps is corrected after a graph optimization of the SLAM system has been performed. In contrast to a single large monolithic map, it is possible to retrospectively shift around each submap's spatial position in this network. This enables the sub-mapping module to adhere to coordinate system shifts and to create a more accurate map representation of large-scale environments.

To achieve this goal, each submap is generated independently and contains an additional rigid transformation matrix linked to the SLAM pose graph. This transformation represents the global position at which the submap should be placed relative to a common global reference frame. As a result, the relative positions between all sub-maps can be reconstructed. Since each submap is directly attached to the pose graph, pose corrections are implicitly handled without additional computational costs. Each linked map transform is synchronously updated as soon as the SLAM system encounters a significant position shift due to graph



(a) Before loop closure (b) After loop closure

Figure 4.9: The Figure shows the loop closure process using the proposed Sub-Mapping module. The non-overlapping sub-maps before the loop closure are shown on the left image. It can be observed from the misaligned walls at the bottom of the Figure that the entire room is skewed, leading to a degraded map. As soon as the loop closure of the SLAM system happens, the entire network of sub-maps is re-aligned, leading to the map in the right-hand Figure. It can be seen that all walls and the entire room are now correctly aligned. [Grosse Besselmann et al., 2023a]

optimization. As a result, the whole network of sub-maps is spatially restructured to generate a more precisely aligned representation of the environment.

This process can be seen in Figure 4.9. In Figure 4.9a, the robot reentered a previously known environment and added the new sensor data according to its current pose estimate. Given a monolithic map, these falsely marked voxels would remain within the map. However, in this case, the new data was added to a new sub-map independent of the one recorded during the previous visit. As a result, as soon as the loop closure is detected, both maps re-align accordingly, creating a more accurate map as seen in Figure 4.9b.

Even on this small scale, the problem of misplaced walls or objects that are inserted more than once could cause significant problems during the navigation process. However, the problem becomes even more significant considering accumulated drift in large-scale environments. In Figure 4.10, an exemplary map of the Deutsches Museum in Munich is depicted with and without sub-map extension. On the left in Figure 4.10a, the raw monolithic map of the environment is shown. It is evident that entire sections of the building are misaligned due to pose estimation errors. As the data set contained multiple loop closure points, the internal SLAM graph representation was corrected retrospectively. However, this was not directly transferred to the monolithic map of the environment. Contrary to this, Figure 4.10b depicts a version of the map which utilizes the developed

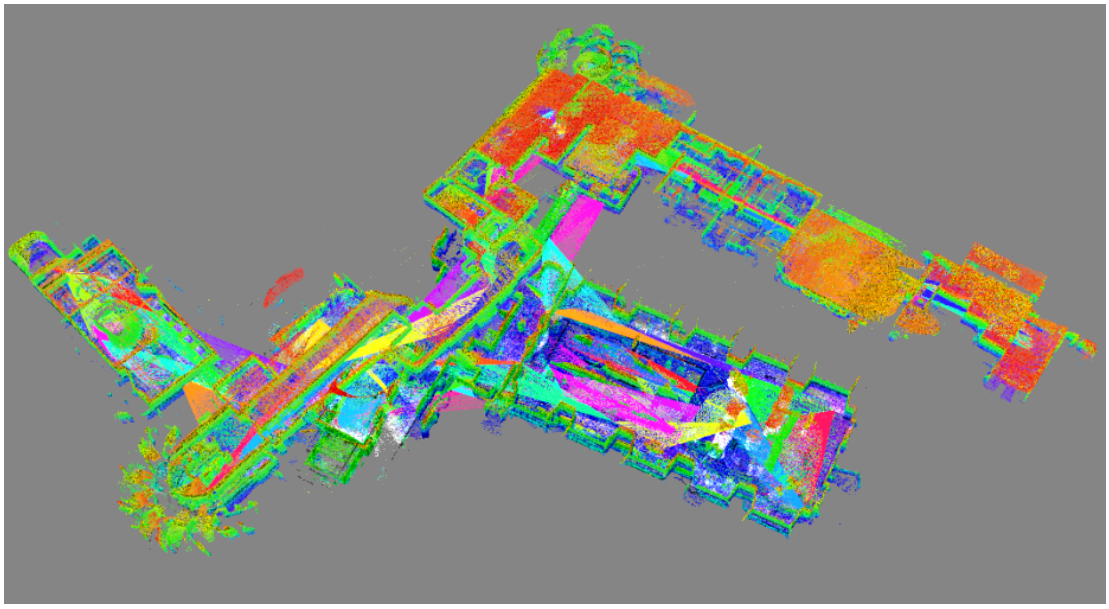
sub-mapping module. Each submap is directly coupled to the SLAM graph, resulting in a correctly aligned, globally consistent map.

Compared to the monolithic method, the essential integration of new sensor data is not altered significantly. An update grid is first created for each sensor measurement using ray casting of all available data. It is, however, not necessary to create an update grid for each sub-map. Instead, the update grid can be integrated into a single submap or reused to fill multiple sub-maps with the same data to create map overlap. The main difference is that prior to the integration step, a transformation of the grid indices is necessary. In the initial VDB-Mapping, the coordinates of the update grid could be directly used since it co-aligned with the map frame. For the sub-mapping approach, however, each coordinate must first be transformed into the specific submap's local reference frame. Furthermore, there are two deciding factors to consider during the submap generation: The first factor is how long new incoming sensor data should be integrated into an individual submap to uphold the local consistency constraint. Given a short integration period of a submap, it will remain more locally consistent due to fewer accumulated pose errors during its lifetime. Conversely, short-lived sub-maps mean less data and less information in each submap. Furthermore, each additional submap introduces small memory and management overhead into the mapping system.

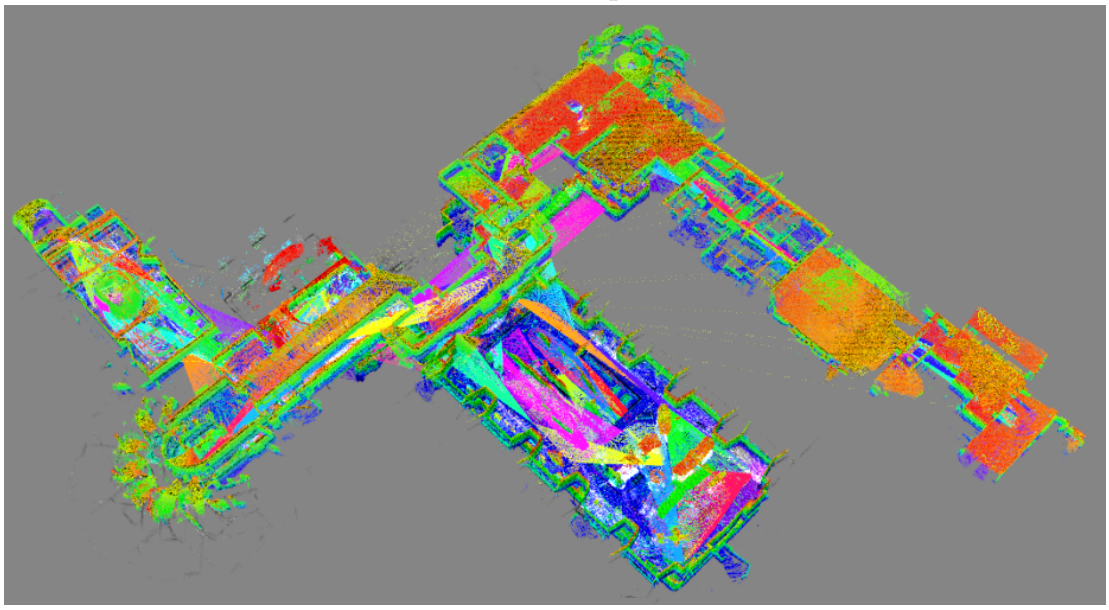
The second factor to consider is how many sub-maps are updated simultaneously. Updating multiple adjacent sub-maps with the same data leads to more overlap between the individual sub-maps, which is beneficial during map merging and navigation. On the other hand, integrating the sensor data into any additional submap costs additional valuable computation time. As a result, the system's performance is impaired, leading to fewer dynamic map updates and possibly dropped data. Even though this problem is reduced by reusing the update grid, the integration step and the additional coordinate transformation must be done individually for each submap.

Both factors are highly interdependent and should be adjusted to the desired use case. In order to control both aspects, the mapping framework tracks the state of each sub-map. Each submap can be either in an open or closed state. In this context, open sub-maps represent the maps that are still updated using new sensor data. Closed sub-maps, on the other hand, are no longer updatable and remain constant forever. The number of sub-maps and how long they should be updated can be freely configured, giving the user maximum control over the sub-mapping system. The moment when a sub-map is considered finished and thus closed can depend on multiple factors. It can, for example, be closed after a specific time or if its size exceeds a certain threshold. Alternatively, it can also depend on the accumulated position error reported by the SLAM system.

During longer mapping processes, the network of sub-maps continuously grows over time. In case the network often revisits previous locations, it also contains multiple redundant sub-maps covering the same area at different points in time.



(a) Monolithic map version



(b) Map utilizing the presented Sub-Mapping module

Figure 4.10: Large-scale map of the Deutsche Museum in Munich. The top image was generated using the default monolithic mapping version. Due to pose estimation errors, huge spatial displacements have been introduced into the map. The bottom image shows the improved mapping approach utilizing the Sub-Mapping module. Here, the pose corrections of the SLAM system are directly integrated into the submap alignment process. As a result, a correctly aligned, consistent map representation is obtained. [Grosse Besselmann et al., 2023a]

To reduce the required memory footprint, it is further possible to prune old overlapping sub-maps from the system. This way, an upper bound to the allocated memory space can be guaranteed. Despite the increased memory footprint, the presented sub-map integration offers advanced capabilities to handle spatial displacement while mapping large-scale environments.

4.2.2 Map Merging

In contrast to one huge monolithic map, a network of locally consistent sub-maps presents multiple advantages. First, creating a more accurate representation of the environment is possible since retrospective pose shifts can be easily adjusted. Furthermore, in the context of local path planning, only a small portion of the environment is necessary for the planning process. Usually, this includes extracting a small area around the robot from the map, which is subsequently used inside the local planning pipeline. Instead, it is possible to take the most recent submap for the process using the sub-mapping system. In the context of map visualization, the submap network also provides further advantages. Each submap can be visualized individually, including its correct pose, to generate the feeling of one coherent map of the environment. Additionally, it is no longer necessary to re-visualize the entire map. Instead, only the open, changing sub-maps must be considered after each update step. All closed sub-maps remain constant and only have to be visualized if a change occurs in the pose graph. The only downside is that overlap between the individual maps is shown redundantly. However, this caveat is negligible considering the vast amount of overall data points.

Nevertheless, some tasks during the navigation process still require a complete monolithic and, thus, continuous map representation. The global path planning task is one core functionality in which this requirement is mandatory. Here, the robot has to consider the entire map to find the optimal path from the start to a goal position. As a result, path planners are commonly unable to work on a network of small fractured maps.

To re-create a complete, monolithic map, each submap has to be merged into a single representation. This entails transforming each sub-map's points into the global map coordinate system and subsequently integrating them into the merged map. Due to the massive amount of data points, this process represents a time-consuming task. However, the complete map does not have to be continuously merged for the global path planning task. It is merely necessary when the global planner requests the calculation of a new path, which happens rather infrequently.

The map can be generated in two forms depending on the desired use case. In the first, more lightweight form, only the active voxels of each submap are considered during the merging operation. This process can be performed quite efficiently using the direct access bit mask of the data structure. As all active voxels represent obstacles, the resulting map enables a robot to distinguish between free

and occupied areas. However, the third state of unknown space is no longer distinguishable in this representation. Additionally, the entire underlying probability framework is lost in this process. This restriction works sufficiently in the context of basic path planning in perfectly mapped environments.

However, in more challenging terrain, the robot often has to cope with incomplete maps and needs to be able to plan through unknown or incomplete areas. For traditional planners used in a planar two-dimensional environment, this does not pose a problem since it can be assumed that all unknown areas are equally flat. Therefore, the additional differentiation between unknown and free or occupied space is unnecessary. It can be assumed that the planner is either allowed or prohibited from traversing the unknown space in the planning process. However, this information becomes crucial while planning a path into unknown areas on a three-dimensional map. This is especially the case for ground robots, which are required to have traversable surfaces below them. If the planner handled free and unknown spaces equally, this might result in invalid paths, as the robot could fly above or dive through the map.

To prevent this from happening, it is necessary to distinguish between free and unknown space in the merged map to enable the global path planner to calculate a path into unknown regions. As a result, only copying all occupied voxels from each submap is not sufficient. Instead, the entire information of the submap network needs to be merged. This includes all occupied, free, and unknown voxels, as well as their respective probability information. Since the amount of ambient free and unknown voxels is by magnitudes larger than only the occupied ones, this process requires a massive amount of computations. As a result, the required time to merge all maps increases accordingly.

The actual merging process is, in both cases, the same. Since the focus of the VDB-Mapping framework also lies in correctly modeling dynamic objects, newer sub-maps are prioritized. To create the merged map, the list of all sub-maps is iterated backward from the most recent to the oldest. The order is chosen to better handle dynamic objects of old sub-maps, which might have vanished over time. For each voxel of a submap, it is verified whether an entry in the merged map is already present at the same spatial index location. If this is the case, it was inserted by a more recent submap, which consequently means the already present data is preferable. As a result, the merged map tends to have a high bias toward more recent data points, reflecting the state of the world more accurately.

4.3 Remote Mapping

Up until now, it was only considered storing the large-scale maps locally on the robot that recorded them. However, it is usually the case that the map is not only required on the local system. Instead, different agents might also require access to the gathered information. Often, robots are controlled from afar by a remote

operator in a control center. This person must receive as much data as possible to make the correct decision. To be able to do this, the operator requires, among other things, the most recent environment data on the controlled robot and its surroundings. Another scenario in which it is necessary to share maps is in a multi-robot setup. Here, multiple robots work together as a collaborative team in the same environment. During operation, each of them creates an individual map of their surroundings. To increase operation performance and prevent redundant mappings, it is desirable that all robots exchange their observed results. Therefore, sharing the data between the individual systems must be possible.

In Figure 4.11 as an example, the topological setup during the ESA/ESRIC-Space-Resource-Challenge [20] is shown. Here, a team of three heterogeneous robots was deployed on a moon-like exploration mission, during which rare resources should be found. In the setup, all robots were connected over a local network and able to exchange their individual maps. An additional remote base station from which the mission was controlled also received the map data. To provide a realistic scenario, the communication capabilities between the moon and the remote network were reduced and suffered from a three-second delay. Even though exchanging maps between different systems provides multiple advantages, it also comes with multiple challenges and drawbacks. One of the main problems is the massive amount of data, which, most of the time, has to be transferred over a restricted wireless network.

In general, there are two approaches to exchanging environmental data between systems. Each robot could share its raw sensor data with all other nearby systems. However, even though data structures, such as point clouds, efficiently encode the data, this approach would lead to an infeasible amount of network traffic due to the high sensor data rate. Alternatively, it is possible, to send the already generated map over the network when changes occur. This process might prove equally infeasible, if not even worse. The required memory of the generated map data structure scales directly with the size of the environment, the enclosing space, and the desired resolution. Since the amount of free space is by magnitudes larger than the number of obstacle voxels, this bottleneck becomes especially apparent in the case of volumetric maps. As a result, it is virtually unbounded and can grow indefinitely, which becomes too taxing for a wireless network and its current limitations. The developed VDB-Mapping framework provides a remote mapping module to exchange map updates as efficient and compact VDB-grids to alleviate this problem. The memory footprint can be reduced significantly by compressing the updates to cope with the network bottleneck. This, combined with the efficient encoding of map update grids, creates a viable solution to transfer data reliably over any given network.

The basic idea is to run a full mapping instance on each system, which generates its own map. However, each mapping instance can receive and process the map updates described in Section 4.1.2 from arbitrary sources. This includes its own sensor data and external map updates from different machines. As a result, it is

Figure 4.11: Schematic system setup of a multi-robot use case. Here, the map data is exchanged between two robots and a mobile base station from which the robots are controlled. [Grosse Besselmann et al., 2022]

possible to rebuild the exact same map as the initial system from which the map data was initially produced.

The choice of how the data shall be transferred and how the remote instance shall integrate it can be adjusted to different use cases. For this, the remote mapping module offers four configurations, each with different advantages and disadvantages:

- Full update-grid
- State-transition grid
- Occupied map section
- Full map section

The full upgrade grid is closely related to the default integration of new data into the VDB-Mapping structure. As described in Section 4.1.2, the first part of integrating new data into the map is to ray cast all its data points and store the results in the co-aligned update-grid. This sparse grid contains all the necessary information to update the map according to the accumulated recent sensor data. Therefore, the obvious choice is to transfer each of these compact update grids over the network. Subsequently, the transferred update-grid can be directly applied to update the map of the remote instance. This way, performing the time-consuming ray casting step again on the remote side is unnecessary, as would be the case if the raw sensor data had been exchanged between the systems. One advantage of removing the need to perform ray casting is that the mapping can be easily applied to systems with lower computation power, such as UAVs. However, this approach still has several drawbacks that must be considered when choosing a suitable remote mapping scheme. Since each free and occupied voxel of the current scan is included, depending on the resolution, the overall number of data points far surpasses those of the input point cloud. However, the discretization within the grid structure and exhaustive compression techniques lead to a similar network bandwidth. Depending on the chosen resolution of the grid, it even requires less bandwidth.

The state-transition grid contains significantly fewer data than the full update-grid. While integrating new update-grids on the host system, it is simultaneously tracked which voxel has changed its current occupancy state. This is done for each state transition into one of the three states (e.g., free, unknown, and occupied). The corresponding current occupancy probability is stored within the state-transition grid if a change occurs. Subsequently, this information can be seamlessly applied to the current map of the remote instance. The resulting grid is sparse since these transitions seldom happen in mostly static environments. As a result, this approach provides, by far, the least amount of necessary network bandwidth.

Both of the first two methods operate directly on the latest input data. This way, the most recent data is also directly integrated into the remote map. However,

this comes with a significant drawback. In case of network instabilities, transferred grids might get dropped, resulting in diverging maps. For the full update grid, this would only result in altered occupancy probabilities depending on the amount of dropped data. For the state-transition grid, on the other hand, the effect would be even more severe since it can miss transition events of voxels. If the state does not change again afterward, it will subsequently never be applied to the remote mapping instance. To make things worse, in a temporary network outage, retrieving any data gathered during this time would also be challenging. Instead, a full synchronization with the map of the host system would be necessary, which is possible but requests a large amount of bandwidth.

The last two methods focus on higher reliability while facing an unstable network connection. For both section grids, the general idea is that, instead of reacting to each new measurement, the system publishes new updates in a fixed interval. In this case, the transmitted grid contains an entire map section. This section is extracted from the current map using a predefined bounding box around the robot's location. As the name states, the occupied map sections only store active, occupied voxels in the transmitted update grid. Since no ambient space is included, the update-grid remains sparse. As a result, the amount of data that has to be transferred is significantly reduced. On the receiving side, however, this has to be considered when integrating new data. Since all free space is omitted, it is not directly evident from the update grid when a transition from an occupied to a free state occurs. Therefore, all occupied voxels within the bounding box of the section will be reset to an unknown state. After this process, the updated-grid's occupied voxels are inserted into the now-cleared map to generate an unambiguous representation of the environment. However, one drawback of this method is that the map no longer contains all three states (e.g., free, occupied, and unknown). Since only occupied voxels are transmitted, only free and occupied cells are distinguishable. This could cause problems for tasks such as navigating unknown areas. Conversely, it offers excellent robustness for visualizing remote maps using restricted network capabilities.

The full map section, on the other hand, contains all free, occupied, or unknown voxel as well as their occupancy probability. This results in a more extensive bandwidth requirement during the data transfer. However, this method provides the advantage of generating the exact same map on the receiving remote mapping instance. Contrary to the occupied map section, a prior clearing of the map is unnecessary since each voxel in the section is overwritten either way.

All of these remote mapping schemes have their uses and can be helpful in various scenarios. As both the full-update and state-transition grid are merely byproducts of the mapping process, they produce no additional computational load. Therefore, they are always provided by the mapping framework. The sections, on the other hand, create additional overhead and can, therefore, be enabled or disabled by the user. Nevertheless, using the Remote-Mapping module makes it feasible to collaboratively create arbitrary maps from multiple systems and exchange them efficiently.

4.4 Map Augmentation

The previously described mapping is focused on generating a map that can store and represent the occupancy state of an area in the environment. This information is usually sufficient for the pure three-dimensional navigation use case. However, tracking additional data about the environment might also be beneficial in multiple situations. Possible scenarios in which additional data would be useful are, for example:

- **Semantic Data:** Each obstacle or voxel could have an additional label from a semantic segmentation. This could include cases in which special objects are present in the environment relevant to the mission. These obstacles could range from unconscious persons to rocks containing precious resources. Therefore, the robot should be able to tag and handle these particular objects on the map. Alternatively, information about dynamic objects, such as doors, could be used to change the motion parameters during navigation.

- **Environmental Risk:** While performing a mission, the robot might encounter a multitude of terrains, each with different surfaces and materials, which should be regarded differently [Puck et al., 2023]. This is especially the case in the context of planetary exploration. In this case, protecting and conserving the used hardware as much as possible is crucial since no spare parts are available, and exchanging them is often impossible. Therefore, detecting and storing the surrounding area's risk factors might be beneficial before planning a valid path. By applying techniques such as anomaly detection, segments of the map can be annotated with additional information about the environment. As a result, the augmented map can be utilized to find less risky paths through the environment.

- **Surface Gradients:** Due to the discretization and resolution of the voxel grid map, an area's exact numerical gradient or surface normals can no longer be recovered. Instead, the calculated value is discretized as well. However, this information can benefit path planning to better cover the robot's motion capabilities. This information can, for example, be extracted from the input point cloud data and subsequently stored within the corresponding voxel.

These are only a few use cases in which storing additional data within the map representation is helpful. As a result, the mapping should be able to handle and easily integrate this information into the data structure. Therefore, the developed VDB-Mapping framework allows for the storage of arbitrary data within each integrated voxel. This includes storing information about an obstacle in any active voxel. Furthermore, it enables the storage of ambient information in free-space voxels to track other environmental properties, such as ambient radiation or the connectivity of a wireless network.

Figure 4.12: Augmented Map of the Tabernas Desert located in southern Spain. It covers an area of around $83\,\mathrm{m} \times 57\,\mathrm{m}$. The volumetric data structure was enriched with additional RGB color information for each voxel. This way, a more natural representation of the environment can be achieved.

As described in Section 2.3.2, the node type of the VDB data structure is templated and highly configurable. This property is utilized in the presented VDB-Mapping framework to store arbitrary data nodes within each node of the tree. This characteristic can be used in two forms. Either individual nodes contain additional information, which can be, for example, used to store semantic annotations within the map. Additionally, storing an entire VDB-Grid layer containing information for each occupied voxel is possible. This way, it is, for example, possible to store an additional RGB color value in each cell. As a result, a colorized map of an environment can be generated. Figure 4.12 shows an exemplary colorized map recorded in the Tabernas Desert. Here, the generated VDB-Map recorded by a walking robot was augmented with UAV pictures taken from the air in the same area. Contrary to the default mapping process, the entire ray-casting process is omitted during the augmented data integration. Instead, it is only necessary to insert the data directly into the corresponding location.

4.5 Conclusion

This chapter introduced the novel, efficient, volumetric VDB-Mapping framework developed during this work. It utilizes the underlying OpenVDB data structure for fast map processing, data integration, and grid access operations. This enables arbitrary robots to generate consistent three-dimensional maps by processing new, incoming sensor data in real-time. Contrary to existing mapping approaches, this fast processing enables the framework to keep up with current sensor data rates. Thus, no data is dropped, resulting in more precise and complete map representations.

In the default variant, each grid cell of the VDB-Mapping framework stores an occupancy value that determines whether the corresponding area is free or occupied. Nevertheless, this thesis provided extensions that enable the framework to store arbitrary data. As a result, it can generate maps of environments augmented with additional sensor data, such as color values or risk estimates.

Next to the basic mapping framework, this chapter also covered how to handle multiple aspects and problems that occur when transferring robot mapping to real-world applications. First and foremost, while facing large-scale environments, mapping systems often suffer from spatial displacements due to pose estimation errors. This issue is usually alleviated using SLAM loop closure detections, which subsequently correct all estimated robot poses. However, this is often not covered in monolithic map representations. This thesis aimed to develop approaches to integrate these retrospective changes seamlessly. Therefore, a submapping module was introduced, which operates on small, locally consistent sub-maps. It was shown how they can be linked to the pose-graph optimization problem. This enables the framework to synchronously re-adjust map alignments to generate consistent large-scale maps without spatial displacements. As subsequent tasks such as path planning often require a monolithic map, the chapter

provided methods for efficiently recreating a single map from the entire sub-map network.

The second major issue with large-scale map representation lies in their usage in a multi-agent system. Since multiple instances rely on the map data, exchanging maps between the different agents is often necessary. However, the network connection between those systems is often limited by the physical restrictions of wireless networks. To mitigate this problem, this chapter provided a RemoteMapping module capable of efficiently exchanging map data between different systems. Furthermore, this enables multi-robot teams to generate collaborative maps. As each robot only has to explore a partial area, robot teams can quickly cover huge areas.

5 Path-Planning on Large-Scale Volumetric Maps

The tasks of mobile robots usually include traversing autonomously from one place to another between different work steps. These tasks often have to be performed in only partially known or even entirely unknown environments. Additionally, while performing the task, the robot has to regard changing environments, such as dynamic obstacles or other robots and humans passing by. Therefore, one of their most essential skills is to navigate autonomously through uncontrolled environments. To achieve this, mobile robots require the ability to plan and execute a path between their current and a target position. Simultaneously, they must avoid collisions with their environment to prevent harm to others, themselves, and their surroundings. The navigation process can be divided into two distinct phases: global and local path planning. The main task of the global path planning phase is to initially calculate a path from the start position towards a goal. This global path is calculated based on the most recent version of the generated volumetric map. During this process, the planning algorithm has two primary criteria to consider.

- **Optimality:** The path should represent the best possible path between the start and goal position.

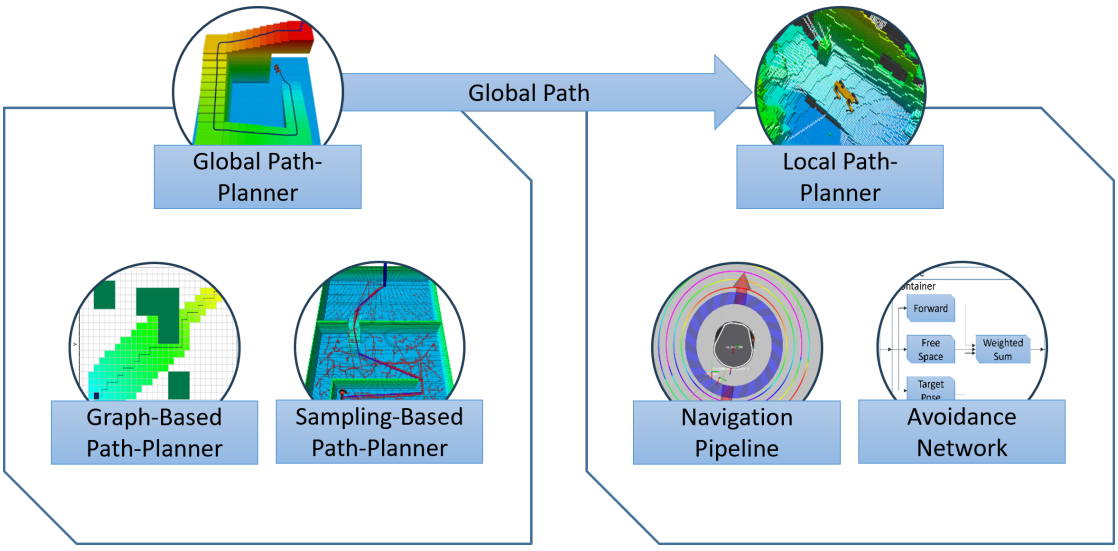


Figure 5.1: The different contribution to path planning presented in this Thesis.

- **Collision-free:** The path should consider and avoid all obstacles in the map along the way.
- **Feasibility:** The path should correspond to the robots' movement capabilities, such as maximum traversable slopes or step size.

During the mapping process, local areas of the map are only updated when the robot is around to fill it with gathered sensor data. However, the map data might be partially outdated since the world is full of dynamic objects and, thus, constantly changing. Therefore, environmental changes that happened while the robot was elsewhere are not considered during the global path planning process. Furthermore, dynamic obstacles are only regarded as static obstacles recorded at the point in time when the planning was performed. As a result, the robot should not directly execute the calculated global path without further consideration. Instead, local path planning is performed to execute the previously calculated global path. This local path-planning process has two primary tasks. First, it should avoid all unforeseen obstacles that were not considered during the global path calculation. For this purpose, only a local map section centered around the robot is used. This local map is constantly updated with the most recent sensor data to adhere to dynamic changes in the environment. Based on the global path, the local path planner guides the robot along the individual waypoints while simultaneously avoiding all previously unconsidered obstacles.

The second task of the local planner is to translate the calculated trajectory into an understandable format for the robot. Since the calculated path usually consists of a list of ordered waypoints, the robot is not directly able to execute it. Instead, the robot is controlled by commandeering a motor velocity for each possible movement dimension. The de facto standard to represent these robot velocity commands in most robotic applications is a twist command. This command contains both the three translational and rotational velocities in which the robot can move. During the execution of the path, the most recent local map and sensor information is considered in the planning process. This way, the local planner can guide the robot along the individual waypoints while avoiding all previously unconsidered obstacles.

The complete navigation process proposed in this work is performed on the three-dimensional space represented by the volumetric map described in Chapter 4. This efficient mapping framework lays the foundation for advanced mobile robots like walking machines to traverse even the most challenging terrains. However, the increased search space, introduced by the additional third dimension, provides multiple challenges regarding their computational complexity. Multiple optimizations to traditional path-planning approaches are necessary to alleviate these issues and provide an optimal solution without exceeding runtime performance boundaries. Furthermore, the used data structure makes it possible to efficiently perform local and global planning in large-scale areas despite the increased search space. The following sections present a deeper insight into the applied algorithms and how properties of the map were leveraged to optimize their overall runtime.

5.1 Global Path Planning

One of the most crucial steps for operating a mobile robot autonomously within an uncontrolled environment is the global path-planning process. Given a map of the robot's current environment, global path planning aims to calculate an optimal, collision-free path between its current and a target position where, for example, a task must be performed. However, as stated in Section 3.2, the global path planning problem for mobile robots is often limited to the two-dimensional case. Two-dimensional planners usually assume that each volumetric obstacle, independent of its height, can be reduced to a single point in the map. To achieve this, the environment is projected onto a planar surface. This restriction and the resulting search space reduction significantly decrease computational complexity. However, this assumption breaks in the face of more complex and challenging terrain with multiple obstacles on different heights. Instead, current navigation algorithms should exceed their limits and utilize the entire three-dimensional search space of the environment. Nevertheless, by transferring traditional two-dimensional algorithms to the three-dimensional search space, they might become infeasible from a calculation time perspective. Due to the additional dimensionality of the search space, the resulting performance degrades drastically. Furthermore, the ability to check the entire robot model for collisions with the volumetric space leads to additional performance impairments. Conversely, the ability to determine viable, collision-free movement positions for the robot more accurately increases its overall safety in challenging terrain. Due to recent advancements in the field of reinforcement learning [104], modern walking robots have become extremely capable of traversing complex terrains. As these robots can be used in more challenging terrain, the necessity for more advanced path-planning strategies is also raised. These approaches should be able to adhere to restrictions such as steady, walkable ground or consideration of the maximal climbable inclination. Therefore, one of the main goals of this work is to develop path-planning approaches specialized for autonomous ground vehicles in challenging terrain, which utilize the entire three-dimensional search space.

The core idea of this approach is to build upon a traditional A* path planner and extend it with various optimizations to directly work on the fast volumetric map structure presented in Chapter 4. By leveraging the fast map representation and its advanced data processing capabilities, the overall efficiency of the presented path-planning approaches could be increased significantly. These combined achievements enable mobile robots to efficiently generate optimal paths through arbitrary environments despite their increased search space dimensionality.

The remainder of this section is structured as follows. In Section 5.1.1, necessary map pre-processing steps are introduced, reducing the path-planning approach's complexity on incomplete or imperfect map data. Section 5.1.2 described details about the transition from the map to graph representation. The selective heuristic cost estimation is presented in Section 5.1.3. To ensure a collision-free and

simultaneously traversable path, Sections 5.1.4 and 5.1.5 present the collision and surface checks necessary for the safe operation of AGVs. The section concludes in Section 5.1.6 with an introduction to how the developed concepts can be integrated into various path-planning approaches.

5.1.1 Map Pre-processing

As stated above, the path generated by the global path-planning algorithm is based on a map that is either available in advance or, more commonly, built from recent sensor data while the robot explores an area. Since the resulting path depends highly on the data structure's accuracy and quality, the input map's grade is crucial. However, in real-world scenarios, creating a perfect map is often not possible. Sensor noise, limited coverage, and discretization errors during mapping often lead to minor inaccuracies. These inaccuracies can range from smaller holes to uneven or non-existing surfaces in the map representation. Furthermore, depending on the chosen technique and parameters, grid cells might require multiple traversing sensor readings before their occupancy state can be clearly determined. As a result, occupied voxels might not get correctly inserted if too few sensor measurements detected the corresponding obstacle.

Unfortunately, the planner is not able to easily differentiate these imperfections from real gaps, pits, or minor obstacles. As a result, more complicated methods to accurately determine the traversability state would be necessary, leading to an impaired performance during path planning. To circumvent this issue, in this work, multiple grid pre-processing steps are performed before the path planning. These steps rely highly on efficient mathematical morphology operations to improve the generated map and simplify certain aspects of the subsequent planning process. By applying morphological operators such as erosion, dilation, and the combined closing operation, inaccuracies during the map construction are compensated. The resulting processed maps are stored separately in additional VDB grids to preserve the integrity of the input map.

The path planning approach presented in this work uses two additional pre-processed maps to perform efficient ground and collision checks. The first one is the *planning grid*, which is intended as a smoother and more complete representation of the environment. For this grid, the goal of the first pre-processing step is to close smaller holes and smooth out the surfaces of the map. Due to inaccuracies of the sensor measurements combined with minor pose estimation errors, some voxels representing an obstacle in the real world are cleared away or never recorded. To circumvent the issue of incomplete maps, a morphological operator is used on the entire map to reduce these occurrences. More specifically, the closing operator is used to smooth the surfaces and close off smaller gaps. Figure 5.2 depicts an exemplary input map created using the mapping approach presented in Chapter 4. From this, it is evident that the non-processed map is incomplete and affected by noise. It can be observed that the floor area is full of

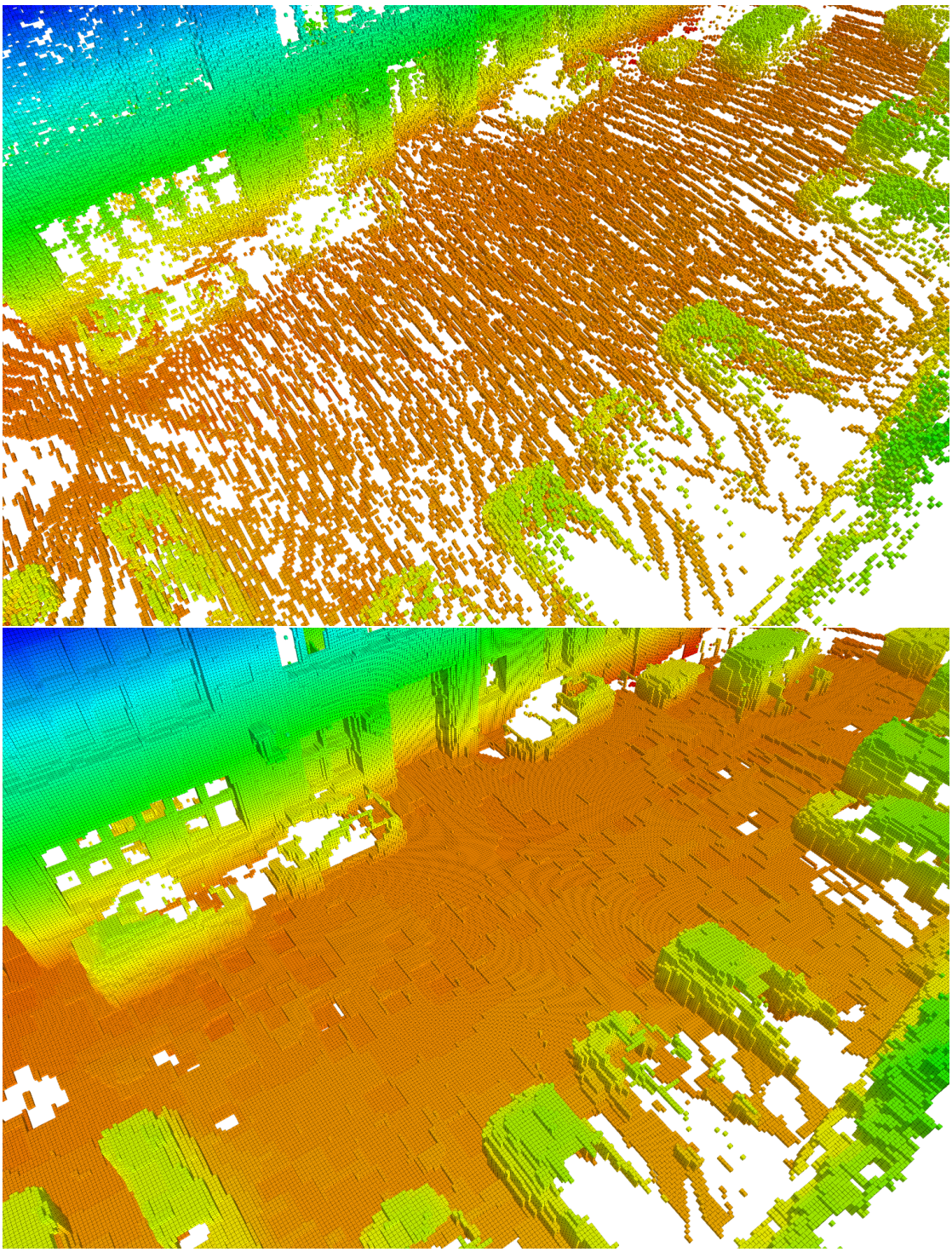


Figure 5.2: Pre-processing of the map during path planning. The upper Figure shows a map of a parking log and office building recorded during a UAV flight. Due to pose estimation inaccuracies, the map contains multiple holes and incomplete areas. The lower map shows the morphologically pre-processed map of the same environment. It can be observed that all surfaces are smoothed and holes are closed, leading to improved path-planning capabilities.

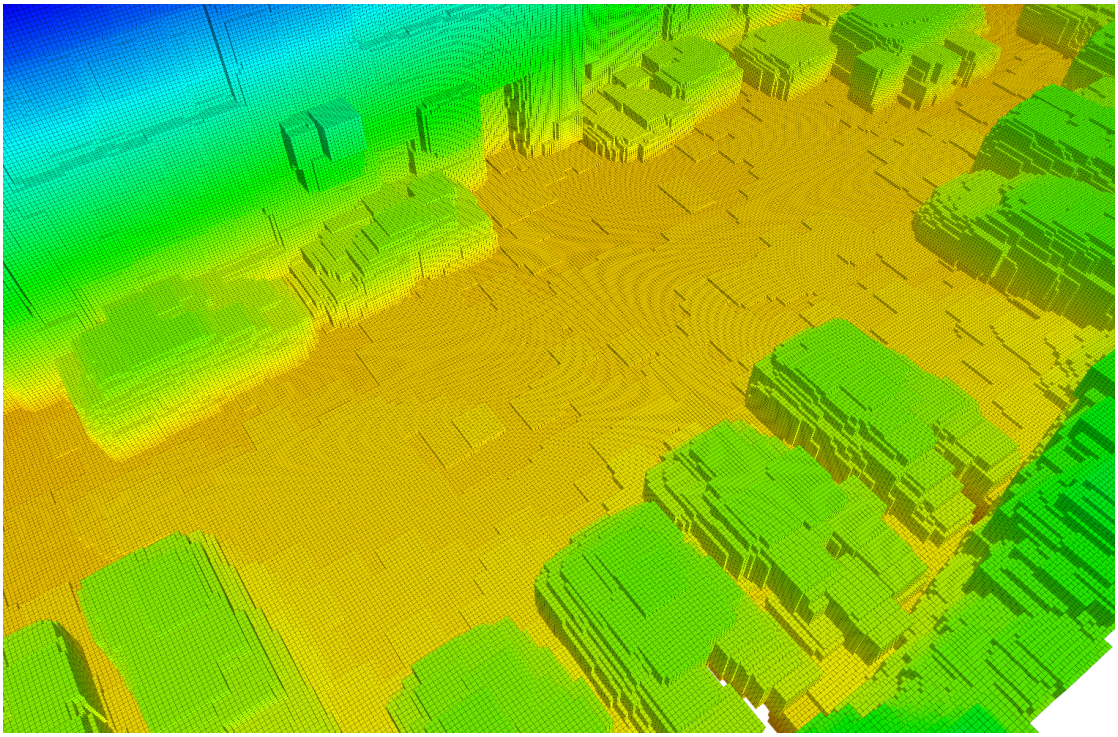


Figure 5.3: Inflated collision grid of the previous parking lot. Here, each surface is dilated by the robot's size. This way, collision checks can be directly performed on this grid, implicitly encoding the dimensions of the robot on the map.

small holes, even though it should be an even, continuous surface. The resulting planning grid of the pre-processing step is shown on the lower part of Figure 5.2. It can be seen that it provides much smoother surfaces and a more natural representation of the environment. As the planning grid provides a clean, continuous representation of the environment, it is heavily utilized during all traversability checks of the planning process.

The second processed grid, the *collision-space* grid, is intended to support the collision check described in Section 5.1.4. The already adjusted planning grid is used as a base for this grid to reduce the number of artifacts. The collision-space grid provides an expanded version of the input map, in which all surfaces are inflated. Thus, ceilings are lower, passageways narrower, obstacles wider, and the ground level is increased. This gives the planning process a better safety margin around obstacles and is crucial for the efficiency of all collision checks. To gain this representation, an additional morphological dilation step is applied to the input grid. This process is performed multiple times, whereas the number of iterations is defined by the robot's dimensions. The goal of the collision space is to inflate the grid by the size of the robot's bounding box. By increasing the size of all obstacles, the collision check can be trivialized. This way, it can be assumed that the robot is the size of a single voxel, resulting in only a single map lookup to determine traversability. Figure 5.3 shows the resulting collision-space grid based

Table 5.1: Selective Heuristic for multiple connectivity functions

Connectivity	Heuristic
4-Connectivity	Manhattan Distance
8-Connectivity	Octile Distance
6-Connectivity	3D Manhattan Distance
18-Connectivity	Fast 3D Octile Distance
26-Connectivity	3D Octile

on the processed grid of Figure 5.2. It can be observed that each voxel has been inflated by multiple layers of voxel equal to the robot's size.

5.1.2 Search Graph Creation

Given the smoothed and improved map, the next step is to translate the volumetric tree structure into a representation, which can be used to plan the optimal path from the start to a goal position. In order to reduce the complexity of finding the shortest path, path-planning algorithms are often based on sparse graph representations. Within this graph, the nodes represent locations in the environment, whereas the edges define the way from one place to another. For planar path-planning approaches, this is commonly achieved by utilizing the local cell neighborhood of the map grid as described in Section 2.2.1. This cell neighborhood is a fundamental concept of grid-based search algorithms. It can be used to determine which neighboring might be considered next while calculating a path toward the goal position. In the two-dimensional space, the planar 4- or 8-connected neighborhood definitions are commonly used. Corresponding to this neighborhood function, each neighboring node is connected to the current node by an edge in the graph. However, this concept needs to be extended to cover the whole three-dimensional search space in the proposed volumetric approach. As described in Section 2.2.1 for higher dimensional neighborhoods, the graph is constructed using volumetric connectivity functions such as 6-, 18-, and 26-connectivity. In the approach presented here, all three types of connectivity are accommodated for graph creation to offer increased flexibility and adaptability during the path-planning process. Even though higher connectivity seems more beneficial, opting for lower connectivity levels also has advantages. A lower connectivity results in a search graph with an overall lower density, resulting in a more sparse search problem and, in turn, shorter average planning time. However, this reduction of the search graph comes at the cost of path quality and optimality. Therefore, more post-processing is necessary to refine the generated paths.

The connectivity is also tightly coupled to the chosen heuristic function of the A*.

The heuristic has to be chosen according to the rules for movement allowed in the grid. The Manhattan distance, for instance, would be suitable for a 6-connectivity grid. However, it tends to overestimate diagonal distances in the case of an 18-connectivity grid. Consequently, in such cases, an adjusted Octile heuristic is applied to ensure the optimality of the path. This heuristic, commonly used only in a two-dimensional context, was extended by an additional height term to provide a more fitting heuristic with respect to the connectivity function. However, in the context of a 26-connectivity, this simplified height term would not be sufficient. Therefore, the 3D-Octile distance accounts for changes in all three dimensions, including spatial diagonals. This heuristic captures the possible movement in a 26-connectivity grid without overestimating the cost toward the goal. As a result, it provides the ability to generate more precise plans while reducing the search space that needs to be expanded. The full connectivity to heuristic correspondences can be seen in Table 5.1.

In addition to defining the connectivity relationship between nodes, the edges of the graph serve another important purpose. While planning a path between a start and a goal node, usually, the path with the optimal cost is calculated. This cost can take on multiple forms, such as the shortest path, the path with the most efficient energy consumption, or the path that provides the quickest travel duration. Furthermore, it can also include more complex concepts that consider other factors, such as the risk the robot has to take during its traversal [Mau23]. These costs between two neighboring nodes are encoded in the edges connecting them and can take on arbitrary values, which are subsequently incorporated into the graph-search process. However, it should be noted that the corresponding heuristic distance has to be adjusted to retain the optimality criteria described in Section 2.2.1. During the path planning process, this graph is iteratively built up, beginning from the node in which the robot is currently present. This process is concluded in two possible ways. Either it is not possible to expand to any neighbor, or the goal node is reached, and no shorter path is available.

5.1.3 Heuristic

As stated in Section 2.2.1, a correct heuristic function tailored to the path planning problem has to be found to ensure the optimality of the generated path. This heuristic has to be adjusted to the robot's movement capabilities and reflect the estimated cost as accurately as possible. Depending on the chosen neighborhood function, the heuristic function has to be adjusted as well in order to prevent the degradation of the search algorithm. The presented approach chooses the correct heuristic function according to the current movement model of the robot. In order to correctly model the estimated cost, the heuristic has to be adjusted to the three-dimensional case. This process is straightforward for most functions presented in Section 2.2.1 since they are already defined in an n-dimensional fashion. In the case of the 6-connected neighborhood or any-angle path planning,

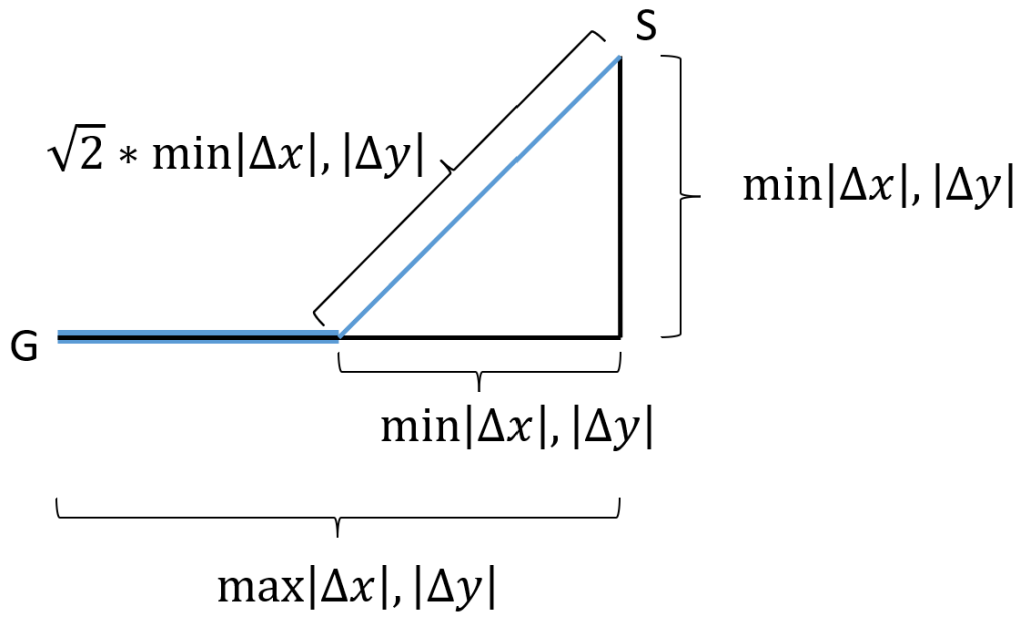


Figure 5.4: Principle of the octile heuristic distance.[Mau23]

the Manhattan and Euclidean heuristics can be used, respectively, in combination with a three-dimensional input vector. However, given a higher connectivity, such as a 26-connectivity, the Manhattan distance would constantly overestimate the actual spatial cost of $\sqrt{3}$ with 3. As a result, the A* algorithm would deteriorate more into a Depth First Search (DFS), leading to suboptimal paths. Therefore, additional adaptations to the heuristic function are necessary for the three-dimensional space given the 18- or 26-connected neighborhood.

As a basis for both heuristics, the octile distance is used. In the two-dimensional case, the octile distance is often used in combination with a fully connected 8-neighborhood. The basic idea of this distance function is to traverse along one axis until a shorter, diagonal movement directly to the target is possible, as depicted exemplarily in Figure 5.4. It can be described by the equation:

$$D_{Octile} = D_1 \cdot max(|x_2 - x_1|, |y_2 - y_1|) + (D_2 - D_1) \cdot min(|x_2 - x_1|, |y_2 - y_1|) \quad (5.1)$$

with

$$D_1 = 1 \qquad \text{(unit distance along an axis)} \quad (5.2)$$

$$D_2 = \sqrt{2} \qquad \text{(diagonal distance across a unit square)} \quad (5.3)$$

This approach can be extended to work in the more complex three-dimensional search space as well In the case of the less intertwined 18-connected neighborhood, a simplified *Fast-3D-Octile* distance has been introduced. Similar to the normal Octile distance, it tries to move along a single axis until a straight diagonal connection is possible. However, it is extended for the three-dimensional case

by including the distance in the z-direction. Formally, it is defined by:

$$D_{Fast-3D-Octile} = D_1 \cdot max(|x_2-x_1|, |y_2-y_1|) + (D_2-D_1) \cdot min(|x_2-x_1|, |y_2-y_1|) + |z| \tag{5.4}$$

For the more complex 24-connected neighborhood, the Octile distance is completely extended to the three-dimensional case to offer the more closely fitting heuristic. For this, the minimal and maximal one-dimensional distances are calculated as before with:

$$d_{max} = max(|x_2 - x_1|, |y_2 - y_1|, |z_2 - z_1|) \tag{5.5}$$
$$d_{min} = min(|x_2 - x_1|, |y_2 - y_1|, |z_2 - z_1|) \tag{5.6}$$

Additionally, the medium distance is calculated by:

$$d_{med} = |x_2 - x_1| + |y_2 - y_1| + |z_2 - z_1| - d_{min} - d_{max} \tag{5.7}$$

Given these three distances, the 3D-Octile distance can be calculated in the following way:

$$D_{3D-Octile} = D_1 \cdot d_{max} + (D_2 - D_1) \cdot d_{mid} + (D_3 - D_2) \cdot d_{min} \tag{5.8}$$

with D_1 and D_2 defined as in Equations (5.2) and (5.3).

$$D_3 = \sqrt{3} \text{ (diagonal distance across a unit cube)} \tag{5.9}$$

The general principle of the 3D-Octile distance function is similar to the two-dimensional case. As before, the distance first travels along a single dimension until it can reach the median dimension using diagonal movement across a planar square. This trajectory is followed until, again, it is possible to directly reach the goal in all dimensions using a diagonal movement across the volumetric cube.

Tie-Breaking

It is worth noting that while using heuristics such as Manhattan, Octile, and 3D Octile distance, their value often remains unchanged in the immediate vicinity of a node. Only if the current node approaches the goal by at least one dimension does the heuristic value decrease. As a result, multiple neighbors of the currently expanded node might provide the exact same path costs. Since the A* is based around a priority queue that prioritizes the next expanded node by the estimated path costs, this behavior causes significant problems. As multiple nodes contain the same estimated costs and, therefore, the same priority, this leads to the expansion of redundant nodes. Next to producing suboptimal paths, this also significantly increases the computational workload and the required processing time. An exemplary instance of this problem is depicted in Figure 5.5, where a Manhattan distance is used as the heuristic estimation. In this instance, both depicted paths have the exact same costs. Consequently, more voxels than are

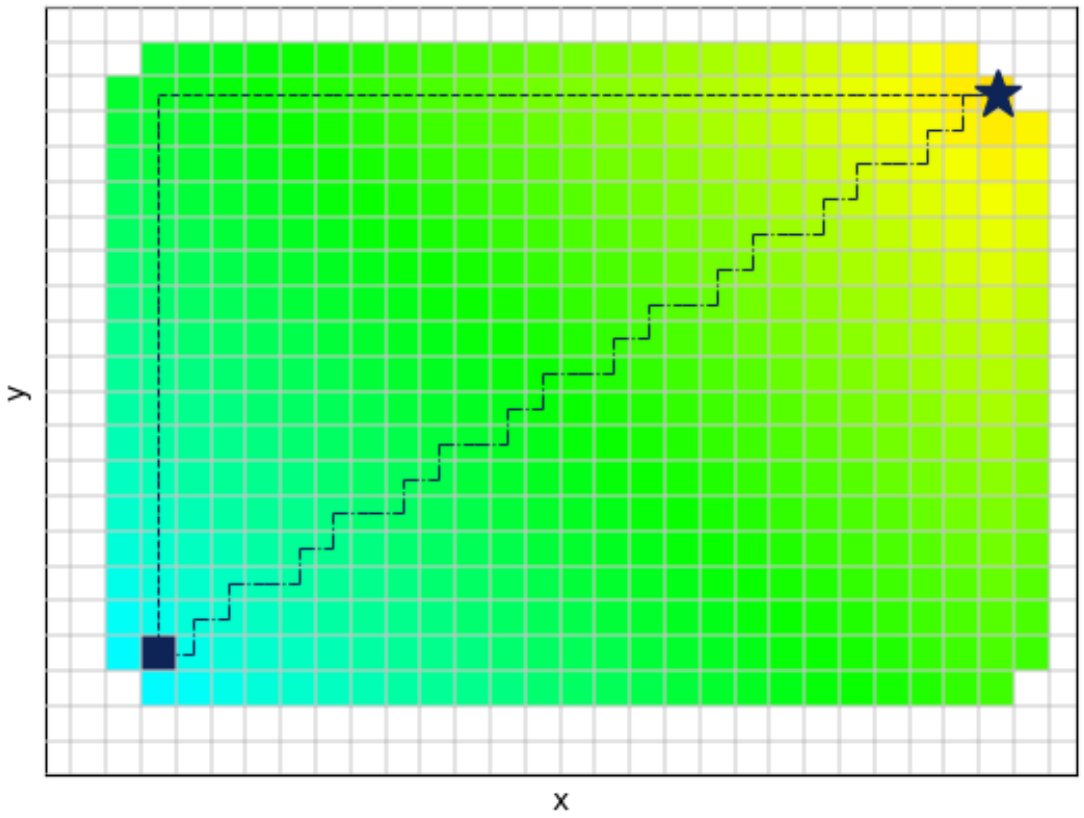


Figure 5.5: The image shows the cost generated using a Manhattan distance. Both depicted paths result in the exact same costs. This also applies to each path traversing within the constructed triangle. However, the direct central path would produce the shortest path. This issue can be resolved by applying a tie-breaking to the heuristic calculation. [Mau23]

strictly necessary are explored during the node expansion, which significantly impacts the overall runtime. This problem is even more severe, given a volumetric planner's increased search space dimensionality, as the number of unnecessarily expanded nodes will increase cubically. Next to the unnecessary buildup of the search space, an additional problem is that these paths are often not optimal. Even though all these paths have the same cost, it is evident that the direct diagonal path would be more favorable as it comes closer to the shortest path after being smoothed.

To alleviate these issues, the presented path planning framework utilizes a tie-breaking strategy to decide which path to choose and which to neglect. The main idea behind tie-breaking is to marginally adjust the heuristic values and give more promising nodes a slight edge. As a result, the priority queue is able to distinguish between those nodes and choose accordingly. For this, an additional yet minimal term is added to the heuristic value h(n) to differentiate it from paths of the same heuristic length. As this value is magnitudes smaller than the actual heuristic values, the optimality criteria presented in Section 2.2.1 is not

infringed. The term consists of a fraction of the orthogonal distance between the current node and the vector connecting the start and goal nodes. This orthogonal distance becomes smaller at the same time as the angle between the two vectors decreases and vice versa. Generally, the utilized cross-product between the vector of the currently expanded node and the target node is minimized, given that the path follows close to the line of sight. Thus, direct movements toward the target are preferred when the priority queue decides on the next target.

The concept can be formalized using three nodes of the three-dimensional search graph.

- $\mathbf{v_s}$: Starting node from which the planning origins.
- $\mathbf{v_g}$: Target node that should be reached.
- $\mathbf{v_i}$: Currently expanded node for which the heuristic is calculated

These three nodes are used to obtain the two vectors used for the cross-product:

$$\mathbf{v_{gs}} = \mathbf{v_g} - \mathbf{v_s} \tag{5.10}$$
$$\mathbf{v_{is}} = \mathbf{v_i} - \mathbf{v_s} \tag{5.11}$$

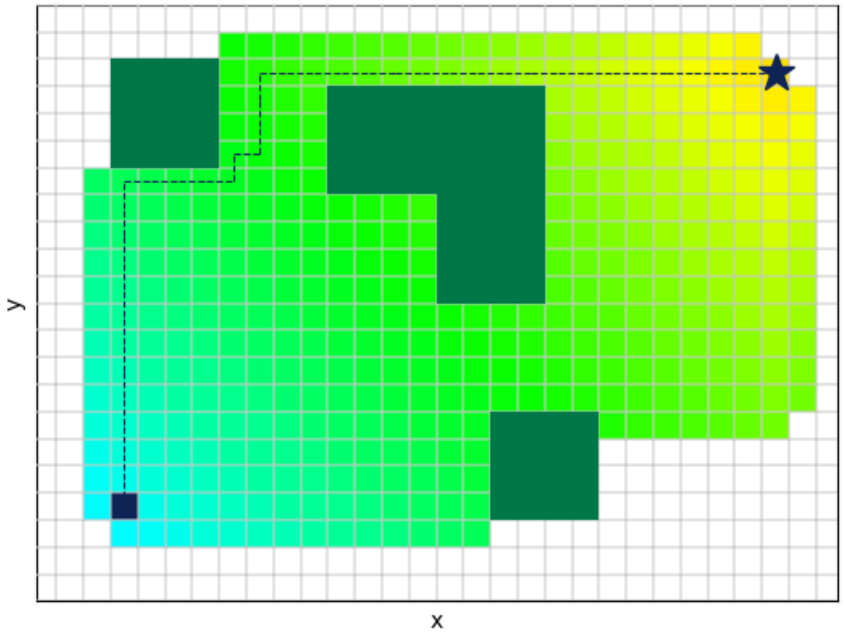
, where $\mathbf{v_{gs}}$ denotes the vector from the start to the goal node and $\mathbf{v_{is}}$ the vector from the start to the currently expanded node. The idea is to calculate the orthogonal distance of the currently evaluated node in relation to the direct vector between the start and goal node.

$$N = \frac{\|\mathbf{v_{gs}} \times \mathbf{v_{is}}\|}{\|\mathbf{v_{is}}\|} \tag{5.12}$$

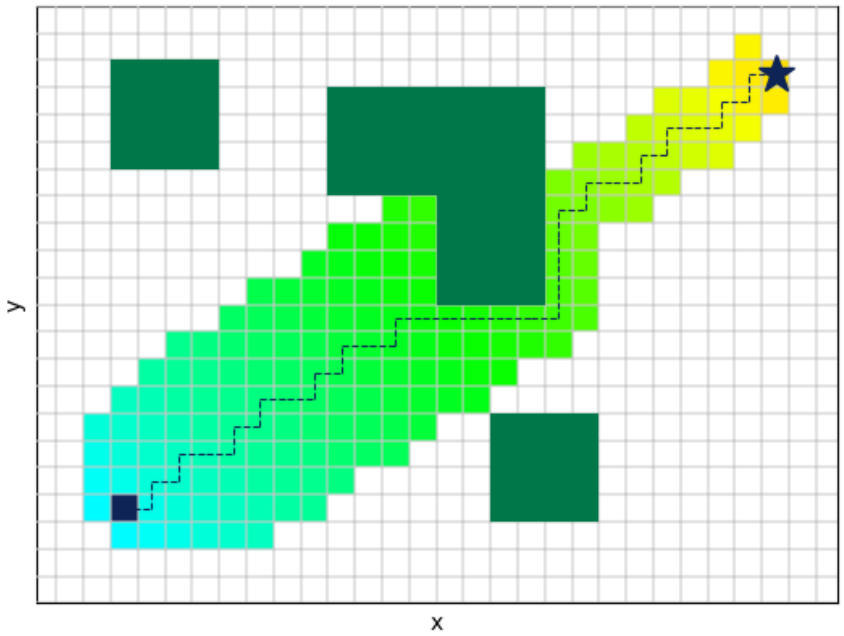
Subsequently, this metric is used to increase the heuristic value $h(\mathbf{v_i})$ by a fraction of N

$$h'(\mathbf{v_i}) = h(\mathbf{v_i}) + \frac{N}{1000} \tag{5.13}$$

As this term only presents a tiny fraction of the calculated orthogonal distance, the heuristic is only marginally adjusted. Therefore, the heuristic does not overestimate the costs, which ensures adherence to the A* optimality criteria described in Section 2.2.1. Conversely, the priority queue is still able to differentiate between the different nodes, leading to a more focused search and a faster calculation. In Figure 5.6, the difference between a search query with and without the tie-breaking is visualized. Evidently, the modification prioritizes its search along the direct line of sight between the start and goal node. Consequently, the tie-breaker heuristic adjustment enables the planner to generate paths closer to the optimum significantly faster.



(a) A* without tie-breaking



(b) A* with tie-breaking

Figure 5.6: On the upper Figure, an A* without tie-breaking is shown. All expanded nodes and their corresponding costs are depicted as green to light blue grid cells. Compared to the lower Figure, where tie-breaking is applied, far too many cells are explored. Despite this increase in search space, it still generates the worst path. It can be seen that by applying tie-breaking, a more direct and simultaneously faster path planning is possible. [Mau23]

5.1.4 Collision Check

In graph-based path planning, the search algorithm expands all viable nodes along the possible path. For each of those expanded nodes, it has to check, among other properties, whether the robot can traverse it safely. Commonly, these collision checks are done at the grid-cell level. This means it uses the occupancy information of each voxel of the map to determine the traversability of the corresponding area of the environment. Nevertheless, depending on the map resolution, the robot might take up more space than the area or volume a single voxel provides. Thus, considering only a single voxel for the traversability of the area might lead to collisions with the environment. In order to address this issue and consequently guarantee a collision-free path, the robot's dimensions have to be regarded during the path-planning process. An exhaustive collision check would require the planning algorithm to check each voxel of the robot for collisions with parts of the environment. Since this check has to be performed for each expanded voxel, this results in a considerable amount of grid collision checks As this would significantly impair the approach's overall performance, the map pre-processing steps described in Section 5.1.1, specifically the collision-space grid, is utilized to decrease the computational complexity.

Using the pre-processed collision-space grid, the search algorithm can neglect the robot model and implicitly check whether a robot with specific dimensions would fit within a part of the map. The idea is to modify the input map, so the algorithm can treat the robot as a single point instead of a three-dimensional object. This is achieved by reducing the size of the robot model to the size of a dot while simultaneously expanding each object within the map according to the robot's dimensions. More specifically, the map is inflated by a size corresponding to half the robot's largest extent. As a result, the entire bounding box of the robot is encoded implicitly in the collision space grid. Therefore, the search algorithm can assume that the robot fits within a single grid cell while the collision check with the environment is implicitly handled.

However, this process has a slight drawback while path-planning. As the exact pose of the robot is not known when the collision check for an expanded cell is performed, the general orientation is neglected during the inflation process. Instead, each obstacle is expanded into all directions equally. Thus, the robot is assumed to be an axis-aligned cube in the transformed space. However, since robots are often not cubic, this approach leads to faulty estimations of the robot's dimensions. Depending on which dimension of the robot is used for the inflation of the map, the assumption might over or underestimate the size of the real robot. Therefore, the chosen dilation size parameter has to be carefully chosen. This choice proves to be a trade-off between enabling the robot to operate at the boundaries of its movement capabilities and ensuring that it does not collide with its environment. For example, in Figure 5.7, the body of an ANYmal-C robot is shown. It can be observed that the body is much longer than it is wide. In case the user prefers safety over reachability, the larger dimension of the robot should be

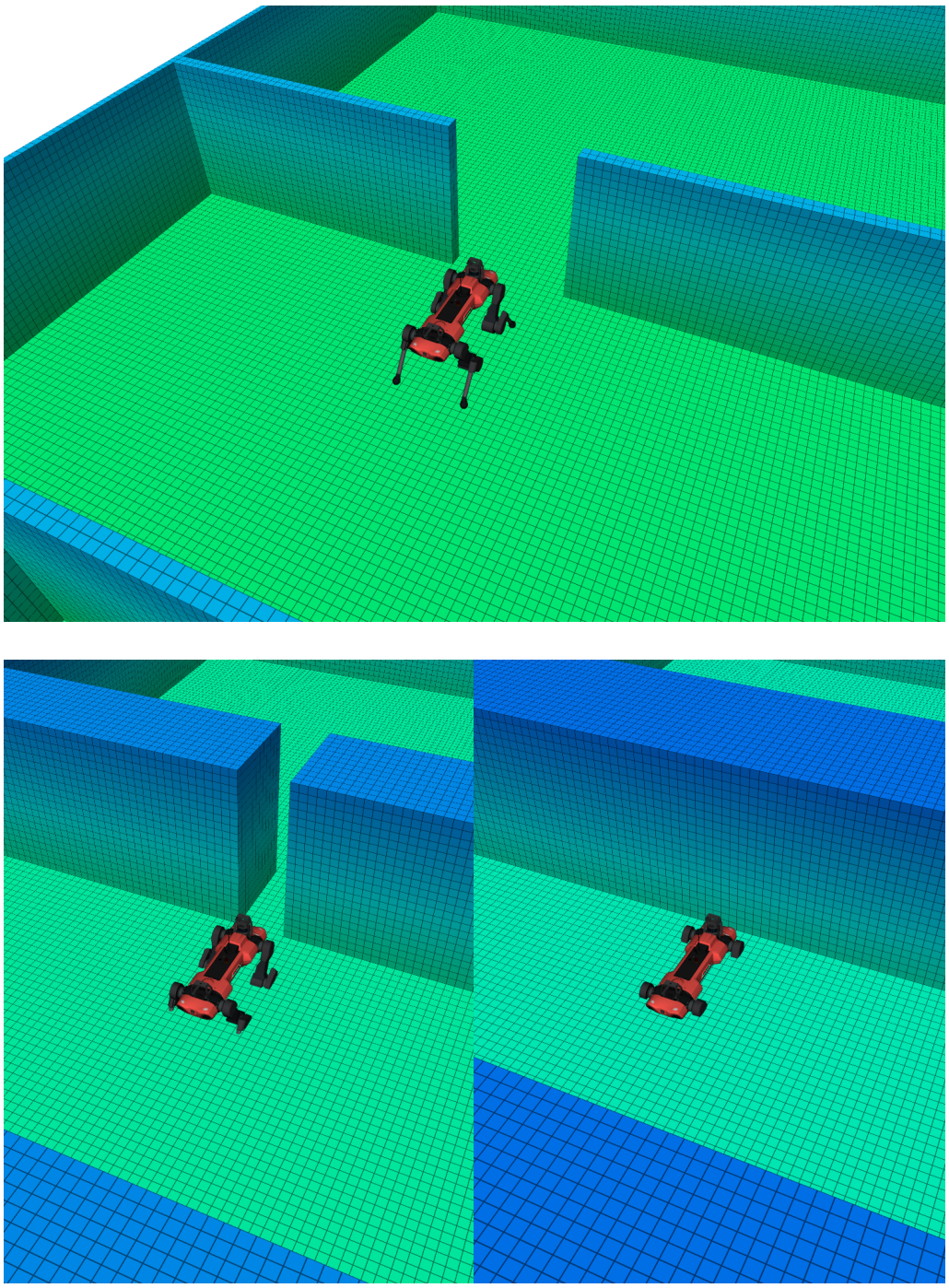


Figure 5.7: Influence of the selected inflation radius. The input map shows a small door through which the robot should plan a path. The left map is obtained if the smaller side of the robot is chosen as the inflation radius. Here, the robot would still be able to pass through the door but has an increased risk of collisions. If the larger side of the robot is used, possible collisions with the environment are reduced. Despite this, the robot would no longer be able to pass through the door as it was closed in the collision map.

used to inflate the map during the path planning as can be seen in Section 5.1.4. As a result, it is ensured that the robot does not collide with any object in any direction, making the planned path safer. However, the path planner would assume a much larger robot despite one side being narrower than the bounding box cube. As a result, it would be unable to find a valid path through tighter passages, even though the real robot could fit, given a specific orientation. Contrary to this, the user could prefer that it is assured that the robot reaches its target by passing through obstacles in a specific orientation. This might be, for example, the case in areas with frequent small doors or corridors. In a normal case, the robot would try to pass through the door in a fitting manner without rotating its body. Thus, it would only be necessary to inflate the door to the smallest dimension of the robot, as can be seen in Section 5.1.4. This decision, however, strongly depends on the user's preferences, geometric conditions, and mission safety criteria. However, it only influences the global path-planning process. Depending on the execution of the local path planning algorithm, the safety of the robot might still be ensured, given that the robot finds the right orientation during the execution of the calculated path.

5.1.5 Surface Check

Given a robot without movement restrictions like UAVs, the previously described collision check would be sufficient to plan a collision-free path the robot can execute. However, additional constraints must be considered when calculating a collision-free and feasible path for autonomous ground vehicles. In addition to the collision check, adjustments are necessary to handle the increased complexity introduced by the additional search space dimension. The path planning algorithm has to account for the limitations imposed by the kinematics of mobile ground robots (i.e., the necessity for steady ground and the limited capability to climb slopes and steps). Therefore, an additional surface check is necessary to ensure that each point on the calculated path provides sufficient walkable ground underneath the robot.

In this case, the simplification of assuming the robot to be dot-sized and inflating the map layer is not feasible. One major problem can be observed in Figure 5.8. Due to the inflation process, the size of the ramp is extended. Using this grid would result in additional surface area, which is, in truth, empty space. This map representation might cause the robot to assume it can walk on these surfaces, resulting in a falling robot. Therefore, the surface check is performed on the non-inflated planning grid. This way, the use of artificially enlarged surface area, which would cause a false ground estimate, is prevented. Figure 5.9 depicts multiple conditions (e.g., stable ground, stairs/steps, inclinations) that need to be covered by the surface check. This surface check has to be executed multiple times at every step of the planning process. More specifically, it has to be performed for each cell neighborhood candidate of all expanded nodes along the

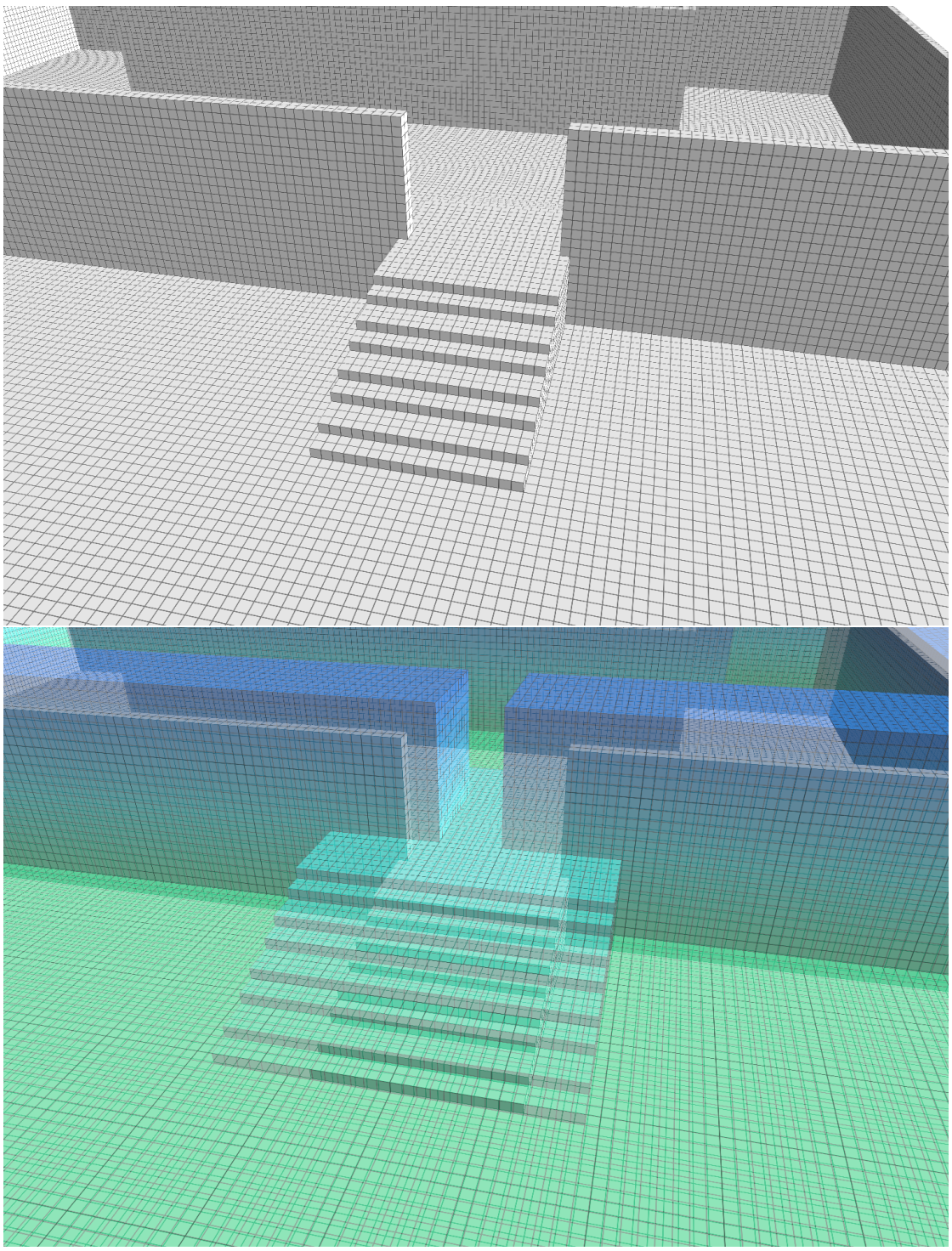
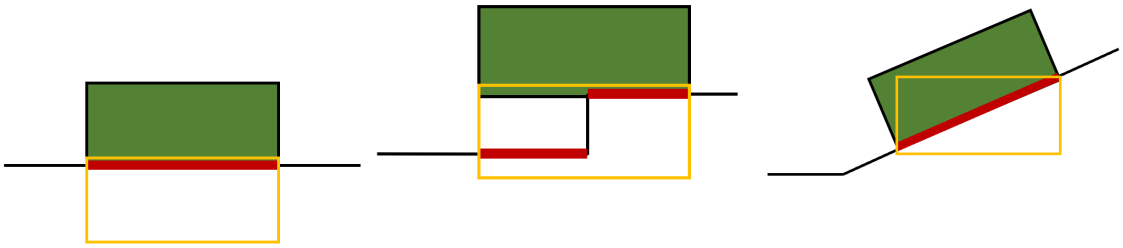


Figure 5.8: Original and inflated version of a ramp. It can be seen that the walkable ground of the ramp is increased during the inflation. If the inflated grid were used during the surface check, it would result in falsely validated ground points. Therefore, all surface checks have to be performed for the entire robot bounding box on the input volumetric map.



(a) Flat ground (b) Stairs/steps (c) Slopes

Figure 5.9: Multiple scenarios the surface check has to handle. The green box represents the robot, black is the overall surface, and red is the surface required for the robot to stand on. The yellow box depicts the surface box that detects walkable surfaces for each use-case.[Hä22]

searched path. Thus, its efficiency is crucial to the overall runtime of the path-planning process.

The surface check is divided into two phases with increasing complexity to decrease its overall runtime. In the first phase, a simple reduced model of the required surface area is used, which allows faster processing. However, it can only handle flat and slightly uneven surfaces and steps or slopes with limited inclination. The second phase is consequently only executed if the first fails to find a viable ground plane. In this phase, a more advanced slope detection is performed, which can handle slopes of arbitrary inclination. If the check is successful in either one of those phases, the current expanded voxel is considered a valid and traversable point on the map.

Phase 1: Basic Model

First, a simplified model for the surface check is initially applied to determine whether an area of the map is traversable. The advantage of this simplified model, in contrast to more complex surface checks, is its ability to find suitable ground areas quickly. Within the basic model, a horizontal box is fitted onto the map area around the robot's current position, as depicted by the yellow box in Figure 5.9 Since this model shall also already be able to overcome uneven ground or steps, the chosen dimensions of the box have to reflect the robot's capability and physical properties. The length and width of the box are defined by the robot's footprint. The box's height, on the other hand, corresponds to the maximum step height the robot can ascend. Since the checked surface box is both axes aligned and in a horizontal orientation, the maximal allowed incline is deducible from Figure 5.10 and can be calculated by:

$$\tan\alpha = \frac{\text{step height}}{\text{robot extent}} = \frac{\text{cell step height}}{2 \cdot \text{inflation level} + 1} \tag{5.14}$$

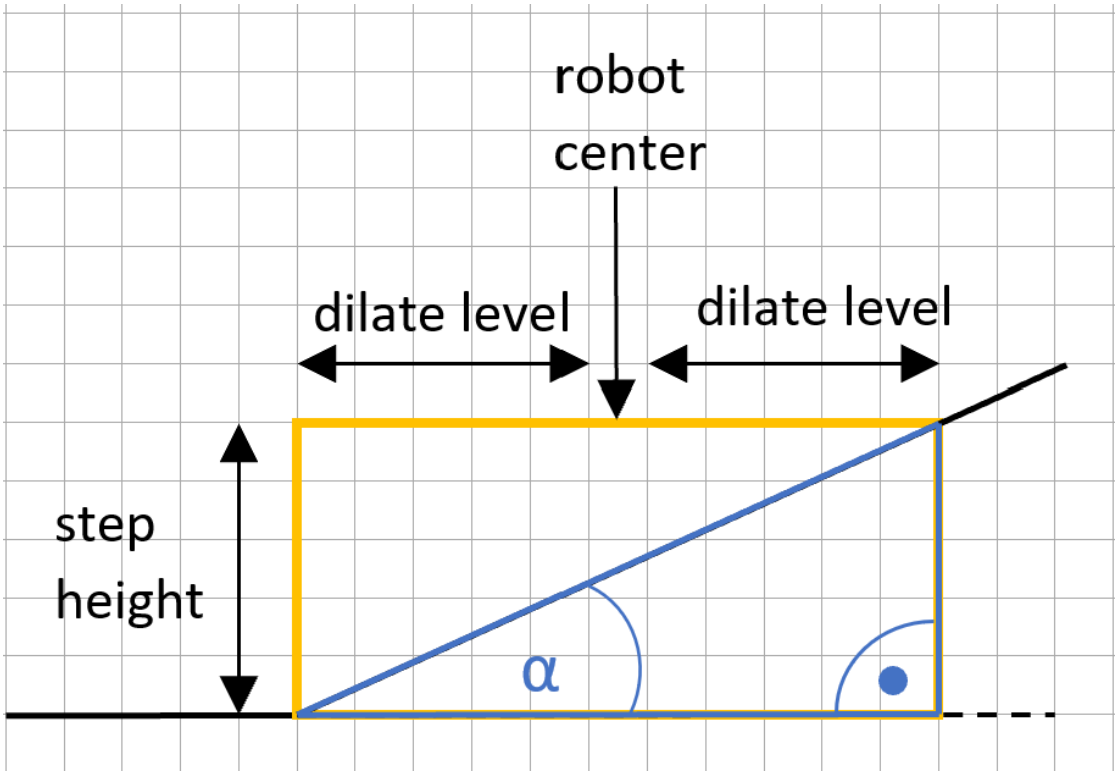
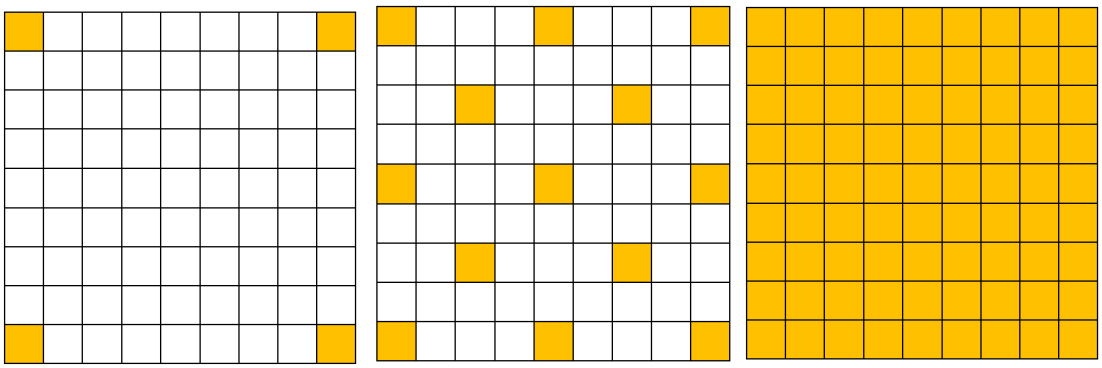


Figure 5.10: Concept to calculate the maximum possible slope of the robot.[Hä22]

In this instance, both the step height and robot extent can be freely chosen with respect to the robot's capabilities and the desired safety factors. However, it is evident that due to the simplicity of this model and the fixed horizontal alignment, possible slopes the robot is able to overcome are restricted by those parameters. The actual surface check is performed by sampling positions within the box to check whether they are currently colliding with objects on the map. The algorithm searches for valid surface points, i.e., a position in which multiple free cells(depending on the height of the robot) are placed above an occupied cell. The simplified surface check is considered successful if each test position yields a positive result.

However, the check for each valid surface point involves multiple grid accesses. Due to the robot's size, grid resolution, and feasible step height, this process can quickly become computationally expensive. Therefore, the proposed surface check can be performed on three levels of detail. Depending on the desired computation time and accuracy of the result, the number of checked positions in the box can be varied as shown in Figure 5.11. The level of detail offers a configurable trade-off between computational complexity and accuracy. On Level 1, only the corners of the box are checked, which provides very fast but also the most inaccurate and possibly unsafe results. In this case, it is assumed that if there is a surface at the corner points of the robot, there is also a surface in between. This assumption might hold in structured environments but quickly fails in case of uneven or fractured terrain.

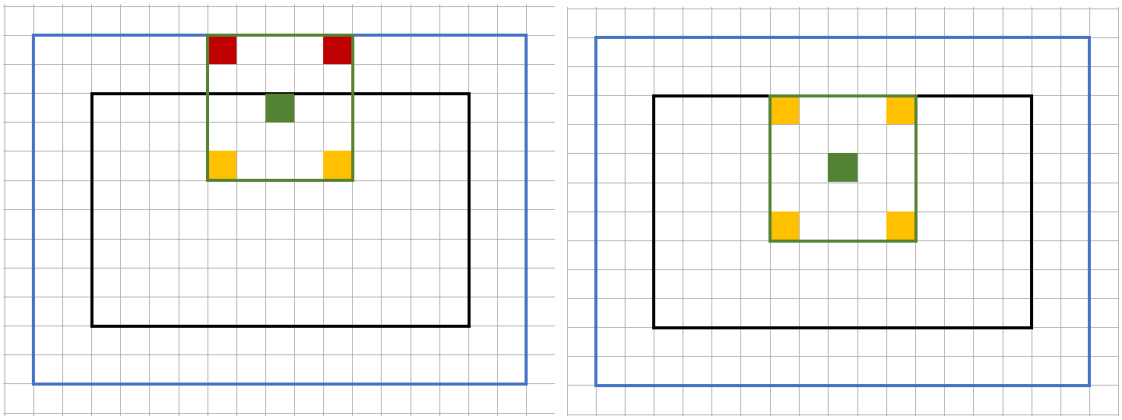


(a) Level 1 (b) Level 2 (c) Level 3

Figure 5.11: Different ground search levels used during the surface check. The yellow tiles represent the search positions. [Grosse Besselmann et al., 2024]

However, the basic idea of checking each corner point is crucial and included in each level of detail to enable the robot to calculate a safe, traversable path. During the planning process, the robot is considered a single cell-sized block. As seen in Figure 5.12, this might tempt the robot to plan a path close to the edges without considering its own extent. During the collision check, this problem is avoided by inflating each obstacle in the map. However, as mentioned before, the inflation process increases the size of obstacles and inflates ledges in objects such as stairs, cliffs, or bridges. As a result, the planner would consider these expanded free cells as traversable ground, leading the robot to fall down these ledges. Therefore, the surface check is performed on the pre-processed, non-inflated map, where edges can be exactly identified. This, however, requires at least checking all corner points of the robot to ensure that the surface provides enough stable ground for it to stand on. The second level adds additional safety constraints to the surface estimate by increasing the number of sampled positions. However, the number of samples is still limited so as not to impair the path planning runtime too much. The sampling of the check first divides the surface box into four quadrants. Afterward, a sample point is set in all four corners and the center of each quadrant. As this surface check still only samples a limited number of points, it offers a compromise between safety and efficiency. Level 3 provides the most detailed surface check. Here, each location of the surface grid is checked individually. However, since this involves a multitude of grid lookups, this increased safety comes at the cost of the algorithm runtime. The performance cost is, therefore, directly influenced by the chosen robot size, grid resolution, and step height.

Due to the fixed horizontal alignment of the checked area, limitations on the inclination angle of possible slopes the robot is able to overcome are introduced. However, given the model's simplicity, comparatively fast initial checks to determine the surface stability are possible.



(a) Invalid surface check result (b) Valid surface check result

Figure 5.12: Representation of the ground check process near edges with respect to the inflated and non-inflated grid. The outer blue box represents the inflated version of the environment, which is represented by the smaller black box. The green cell represents the currently evaluated grid cell in the center of the robot, whereas the larger green box represents the robot's bounding box. In order to find a viable, walkable surface, the ground check has to be performed on the non-inflated version of the map. Otherwise, this would cause the ground check to provide infeasible solutions. [Hä22]

Phase 2 - Slope Detection

If finding a viable ground estimate within the first phase of the surface check is impossible, the second phase is applied. In this phase, a more complex approach is used to cover cases that are not correctly handled by the first check. The overall concept is shown in Figure 5.13 in a simplified two-dimensional projection. The main goal is to estimate the current inclination of the surface. For this, a plane is fitted into the volumetric map around the robot. The idea is to search for anchor points at the edges of the robot body bounding box. These anchors are chosen in a way that provides the ability to detect slopes of a given maximum angle, defined by the slope detection range.

The search positions are the same corner points defined during the Level 1 surface check of the first phase. In order to fit a plane into the map, the algorithm must find at least three valid surface points to span a surface. Subsequently, the plane's inclination in all directions is calculated and checked against a robot-specific threshold. This way, it can be assured that the inclination of the fitted plane does not exceed the robot's capabilities. If the fitted plane does not exceed the maximum allowed inclination angle, a similar check as in the first phase is applied. Again, a bounding box is fitted around the robot for which surface checks in a pre-defined pattern are performed. In contrast, during Phase 2 verification, the box used to check the surface area is tilted according to the plane's orientation. By doing so, the path-planning algorithm is able to overcome uneven surfaces of

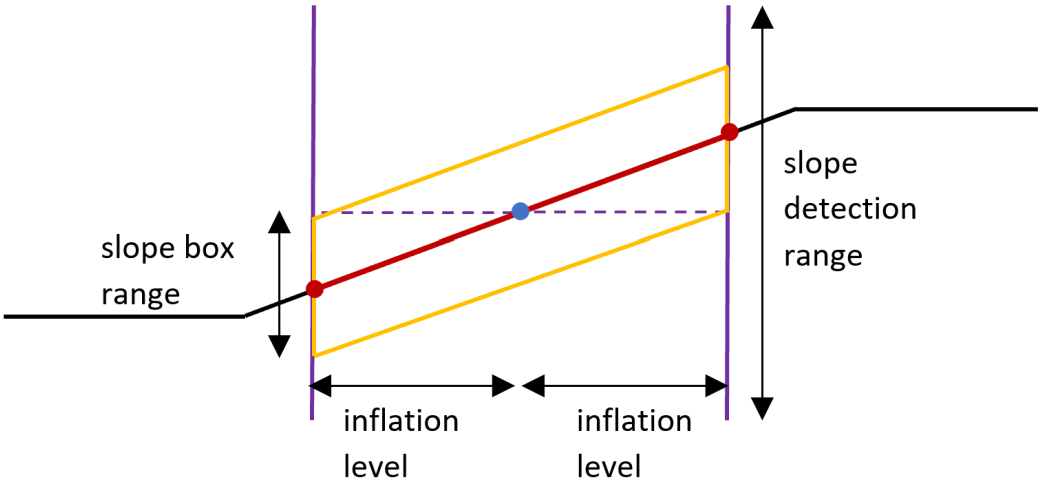


Figure 5.13: Principle idea of the slope detection concept in a two-dimensional projection. The algorithm searches for viable anchor points in the slope detection range depicted by the violet bars. When at least three viable anchor points depicted in red are found, a plane is created from them. Subsequently, it is verified if the currently checked position, depicted as a blue dot, lies within the slope range box to ensure the pose is close to the plane.[Grosse Besselmann et al., 2024]

varying inclinations.

Similar to the first phase, verifying the surface points within the enclosed area of the bounding box can be done in various levels of detail. This way, finding a suitable trade-off between computation time and safety constraints is again possible.

Surface Caching

Due to the vast amount of voxels it has to verify, the previously described surface check algorithm is rather time-consuming. The check is required to determine whether each cell has enough viable ground floor underneath to support the robot. This process has to be performed for each voxel considered during the path planning process. This also includes each possible neighboring candidate of currently expanded voxels. Combining these two factors results in a huge amount of grid access requests. To mitigate the issue's influence on the computational time, an additional cache structure is introduced in this approach. This surface cache stores information about the presence of sufficient ground surface for all voxels that have already been checked. Here as well, the VDB-grids are utilized to provide an efficient yet fast caching structure. An efficient constant-time random access cache is achieved by using an additional co-aligned grid that

stores the results of previously checked voxels. To further reduce the memory footprint a lightweight boolean grid is used to store the cached values. As soon as a voxel checked, the corresponding cache entry is set to active in order to quickly determine if the costly surface check can be avoided. If, at a later stage, the same cell s visited again, the cached value is reused instead of performing the expensive surface check and slope detection. Otherwise, the entire check is performed and subsequently added to the surface cache for future use.

Planning into the Unknown

The application of robots in real-world environments often involves multiple challenges. The data generated by the sensors is often incomplete or noisy. Consequently, the recorded map is also full of holes due to occlusions or unexplored areas. During the path-planning process, this problem is mainly mitigated by the morphological pre-processing step described in Section 5.1.1. However, in some instances, the missing data is too significant to be handled entirely by the map pre-processing. On the other hand, the planner has to be able to handle completely unknown and unexplored areas of the environment. To represent this within the map, each voxel has three states, which are handled differently by the planner. The voxels are either free, occupied, or, most important, in this case, unexplored. The class of unexplored voxels represents all voxels that have none or very few observations. As a result, the neighborhood function regards these kinds of voxels differently during the path-planning process.

In order to overcome these areas, all search-space restrictions described in Section 5.1.5 are negated within the area's bounds. More specifically, unknown voxels are considered all free during the collision check and simultaneously seen as walkable surfaces in the ground estimation process. As a result, the planner is able to overcome the unknown space in all directions, including diving below or flying above the map. This is necessary as there is no information on whether a connecting surface or the goal node is at the same height as the last free or occupied surface area. However, this leads to a largely increased search space as more neighbors are considered as a successor node. To alleviate this problem, the planner is configured to prioritize known areas to prevent the algorithm from deteriorating. Nevertheless, as the unknown space often does not contain any obstacles, the heuristic function combined with the tie-breaker usually results in taking the straight path toward the goal.

5.1.6 Integration of Different Planners

The concepts presented in this chapter are generally optimized to work with an A* or similar graph-based approaches such as D*. However, in order to further spread volumetric mapping and three-dimensional navigation within the science

community, the concepts can be transferred to different three-dimensional planners as well. Therefore, the crucial collision and surface checks are kept independent of the underlying planning framework.

Exemplarily, for the evaluation, the concepts were used in combination with various sampling-based path planning algorithms such as RRT [61], RRT-connect [59], or PRM [49]. In this case, the Open Motion Planning Library (OMPL) [117] was chosen to integrate different path-planning approaches. Sampling-based planners are able to generate non-optimal but fast near-optimal solutions, even for complex problems. Thus, the OMPL is often applied in high-dimensional search spaces such as calculating trajectories for robotic arms with six or more degrees of freedom. It provides multiple interfaces to set up and exchange different sampling-based planner approaches. As the name states, sampling-based planners generate a set of samples based on the underlying map representation. In the context of path planning during robot navigation, the sampling space is built from the set of all possible three-dimensional poses the robot could occupy. As the necessary time to find a valid solution is highly dependent and scales proportionally to the number of checked samples, it is beneficial to restrict the sampling space beforehand. Since the underlying map representation already discretizes the entire space into a voxel-index space, sampling poses in a continuous space would provide no additional information gain. Consequently, the sampling process can be directly restricted to the discretized voxel-index space of the VDB-grid. Sampling directly from the index space of the map simultaneously restricts the sampler to areas in which map data is available. As a result, each of the sampled coordinates remains within the boundaries of the map representation without further consideration.

Nevertheless, this still includes vast ambient free space within the mapped area. In the graph-based approach, invalid movements were mostly restricted by the neighborhood selection. This is not given in sampling-based approaches, as the state space is sampled randomly. To prevent the robot from flying above the map and further restrict the number of sampling points, a similar approach to the calculation of valid nodes described in Section 5.1.5 is applied.

Subsequently, a random uniform distribution continuously generates sampling points from the index search space. The goal is then to verify or reject sampled points as quickly as possible.

Due to their three-dimensional nature, each sampled point has a high probability of floating at an unusable height for ground robot traversal. Therefore, it is first necessary to project it onto the closest ground voxel to generate a possibly valid sample. The definition of ground is, in this case, somewhat simplified. For performance's sake, it is only checked whether a single occupied voxel is available below. If no viable ground voxel for the sample can be found, the sample is immediately discarded, effectively pruning the search space. This way, costly surface checks are prevented, which consequently improves the runtime.

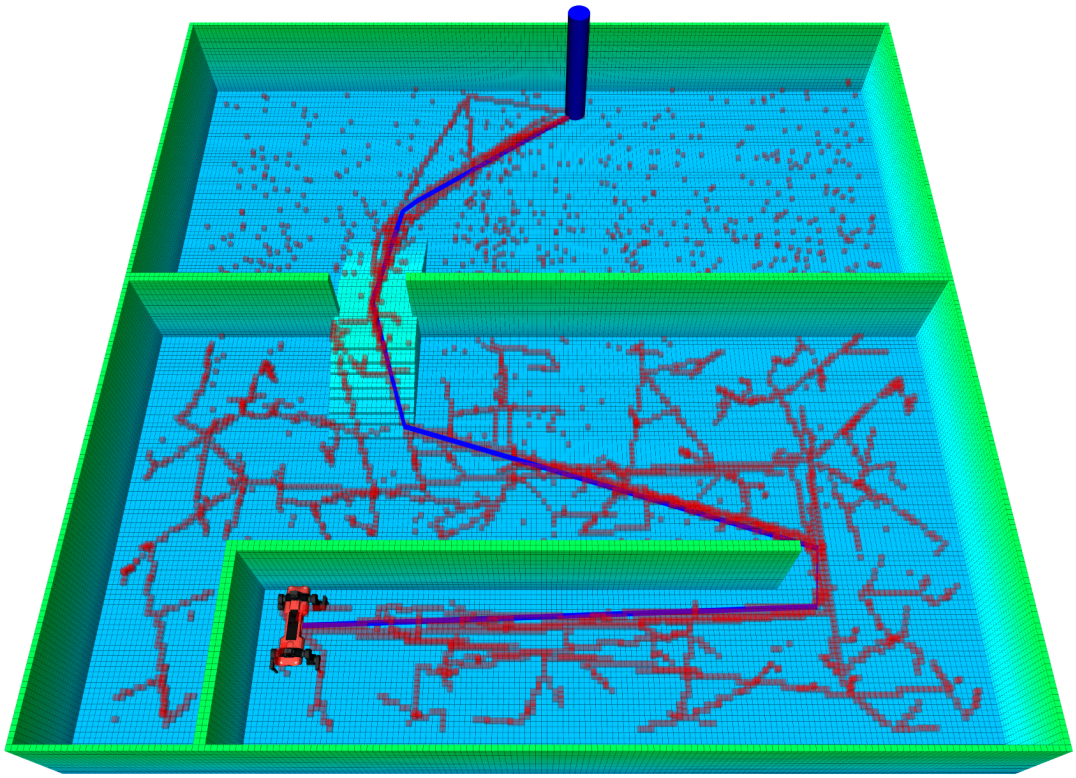


Figure 5.14: Path planning process using an RRT search algorithm. The red voxels represent sampled points and the collision-free paths in between them. The blue line presents the path extracted from the random tree.

If a sampled position meets this minimal requirement, more sophisticated checks will be performed. For this validation step, the developed concepts in the above sections are re-used. Therefore, the set of sampling-based planners also utilizes the map pre-processing steps described in Section 5.1.1.

It is first verified whether the sampled point is in collision with the collision-space grid as described in Section 5.1.4. Subsequently, the extensive surface verifications proposed in Section 5.1.5 are applied here as well. Here again, a caching of previously conducted surface checks as described in Section 5.1.5 can be performed to increase the performance in case the same position is sampled multiple times. Figure 5.14 depicts the sampled space as well as the resulting path from an exemplary RRT-Connect planner. It shows that the concepts developed in this chapter can be easily adapted to various path-planning algorithms.

5.2 Local Path Planning

Section 5.1 presented the process of calculating an initial three-dimensional path from a start to a goal position. Given this path, the robot's next step in the overall navigation process is to execute this path. More specifically, to translate the

path into commands that the robot is able to understand. The global plan is, as described above, calculated on the entire map of the environment. However, the robot can only update the map around its current location since it requires sensor readings to do so. As a result, the map on which the global path is based might be outdated or stale at places the robot has not visited for a longer time. Furthermore, the current environment might also contain various unforeseen circumstances, such as dynamic obstacles or structural changes. If the environment has changed in the meantime, this information is not considered during the global path planning step. Although the global plan is essentially collision-free and optimal, it might lead through obstacles that were not present in the current state of the map. Therefore, the robot cannot simply execute the calculated plan. Instead, a local planner is required to include and react to the most recent information available. Since it operates only within a local scale, it is able to incorporate all locally constrained sensor data into the planning process. This way a more safe path execution can be made possible.

In classical approaches, local path planning is also commonly performed on a projected 2D or 2.5D representation. Since these maps, for example, do not reflect circumstances such as overhangs or ceilings, the local planner can not react to this additional information. Furthermore, obstacles such as holes or ledges in the environment are not covered. Therefore, in order to increase the robot's capabilities to operate autonomously in challenging terrain, it is also beneficial for the local path planning algorithm to utilize the entire three-dimensional state space.

To achieve this, here as well, the volumetric map representation introduced in Chapter 4 is applied to provide fast access to information about obstacles within the robot's vicinity. The core part of the local planning algorithm presented in this work is the plugin-based NavPi structure. The proposed structure is highly adaptable and enables the robot to exchange individual components during runtime based on the current situation. As a result, it can adapt quickly to changing environments and different scenarios. In general, the pipeline architecture has been developed to be used for both two- and three-dimensional use cases. However, the plugins described here are all tailored to be used in combination with the presented volumetric VDB-Mapping framework. The following section will give a short introduction to the presented pipeline and how the different parts of it interact with each other. Afterward, a detailed description of each three-dimensional navigation plugin is given.

5.2.1 Navigation Pipeline

One of the main challenges of autonomous navigation is that the robot has to handle various terrains to reach its goal. Especially when traversing previously unknown terrain, the robot does not know which obstacles are expected in what quantity. This raises considerable problems since different obstacles require different behaviors of the robot to pass them safely.

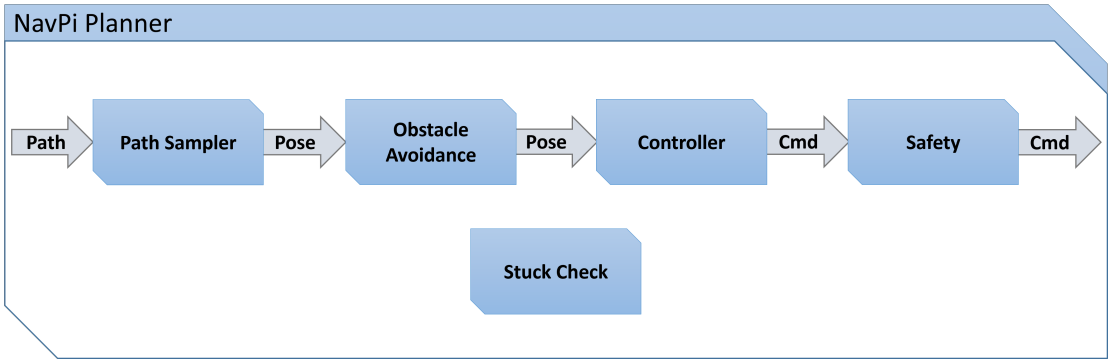


Figure 5.15: Navigation pipeline and its individual components.

For example, assuming the robot has to traverse a crowded environment with multiple people moving dynamically through the area. In this case, the robot should focus more on safety and obstacle avoidance to keep other persons in its vicinity safe (i.e., not collide with them). This could be achieved by having the robot keep a higher distance from obstacles in its environment. However, given these more restrictive settings, the robot might no longer be able to pass smoothly through narrow corridors and doors since it wants to keep its distance from both sides of the frame. To enable the robot to handle this and other scenarios, the proposed navigation framework provides a highly configurable and interchangeable pipeline structure that can be tailored to different environments. A processing pipeline comprised of multiple independent steps is proposed as a basic structure.

Figure 5.15 provides an overview of the structure developed for this local path-planning approach. In general, the pipeline is divided into four main steps and one additional controlling step:

- **Sampler:** Samples the global path and returns the next target pose.
- **Obstacle Avoidance:** Re-adjusts the sampled target pose to avoid collisions with obstacles.
- **Controller:** Translates the adjusted target pose into commands the robot understands.
- **Safety:** Verifies and prevents a collision if the calculated command is invalid.
- **Stuck Check:** Gathers the results of each individual pipeline step and verifies whether the robot is stuck.

Each individual part of the pipeline can be exchanged and configured to fit the present scenario to increase the flexibility of the navigation framework. However, given that the extent of the challenges the robot has to face might not be known previously, simply exchanging parts beforehand might not be enough. Additionally, the proposed navigation pipeline allows freely exchanging parts of it during

runtime. This is achieved by implementing each of the individual steps as a dynamically loadable plugin. This enables the robot to load and switch between multiple plugins, each covering a different challenge or scenario. As a result, the pipeline can be easily adapted to various conditions and situations. The following sections give a detailed insight into the basic idea behind each component. Furthermore, a description of each default behavior is provided.

5.2.2 Path Sampler

As an input, the local path planning pipeline receives the previously calculated global path as input. This path consists of a set of consecutive poses representing the trajectory the robot has to traverse to reach a certain goal. However, this data format is not directly executable for the robot. Instead, the local planner has to extract intermediate goals, which closely follow the traversed path. Therefore, the main goal of the path sampler is to extract the next valid point from the input path. The basic path sampler plugin the proposed navigation framework provides is split into two steps. First, the pose segment of the path closest to the robot has to be determined. Naively thinking it would suffice just to choose the point closest to the robot. However, in the case of tightly intertwined paths, like traversing winding roads or executing meander paths, this behavior might not yield the desired result. Due to effects like avoiding obstacles, the robot does not necessarily follow the path exactly. As a result, the robot might skip huge parts of the path which is not always beneficial. A simple example can be seen in Figure 5.16. The robot should follow the entire U-turn structure to avoid the wall splitting the area. However, as it avoids a previously unknown obstacle, it moves closer to the global path segments on the other side of the wall. If the path sampler simply chooses the closest point, which lies on the other side of the upper wall, the planner would try to pass through the obstacle. This would result in the best case with a stuck and, the worst case, a crashed robot. To circumvent the issue, the planner remembers the last closest robot pose and iterates from this point through the remainder of the path. For each pose considered this way, it is verified that within this pose's coordinate system, the robot still lies in front of it. As soon as one of the poses overtakes the current robot position, the iteration is stopped, and the pose is returned to the next phase of the path sampler.

After finding the correct pose of the path segment closest to the robot, the sampler has to decide the next pose the robot should try to reach. In the most basic version, the sampler chooses the pose by simple means of a look-ahead. The user can specify a distance, and the robot calculates the amount of poses it has to skip within the global path list and returns the corresponding element. A more complex behavior includes the consideration of the path's curvature. In this case, the distance the robot samples ahead is dynamically adapted by a maximum curvature that cannot be exceeded. This way, corners are prevented from being cut if a far sampling distance is chosen.

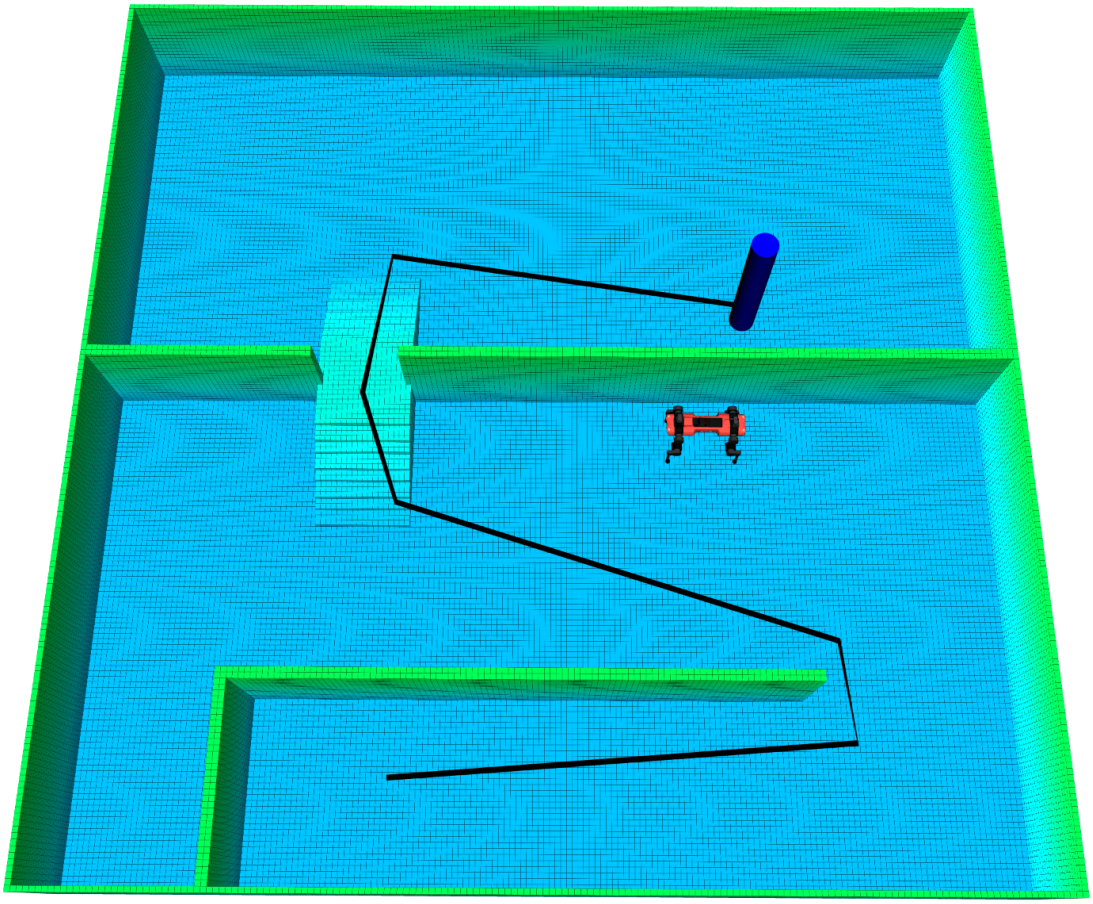


Figure 5.16: Specific use case where the local planner has to handle a U-turn correctly without shortcutting through the wall.

5.2.3 Obstacle Avoidance

In general, the generated global path can be, by definition, considered collision-free. Thus, the same can be assumed for the sampled pose of the path sampler. However, as mentioned before, the global planner's plan is based on a pre-recorded map of the environment. Therefore, this map might contain outdated information as it has only been updated in areas the robot has visited recently. Furthermore, fast dynamic objects might not be covered correctly. As a result, on a local scale, the sampled pose might not be in a collision-free state. Thus, it becomes necessary to implement additional checks to prevent the robot from colliding with unforeseen obstacles while traversing the global path.

To achieve this, an additional local obstacle avoidance component is added to the navigation pipeline. As an input, the obstacle avoidance receives the previously sampled goal pose of the path sampler. Its main objective is to adjust this pose to provide the robot with a collision-free trajectory to reach its target. Next to preventing collisions with possible dynamic or previously unobserved obstacles, it can also fulfill additional constraints. It could, for example, ensure that a certain

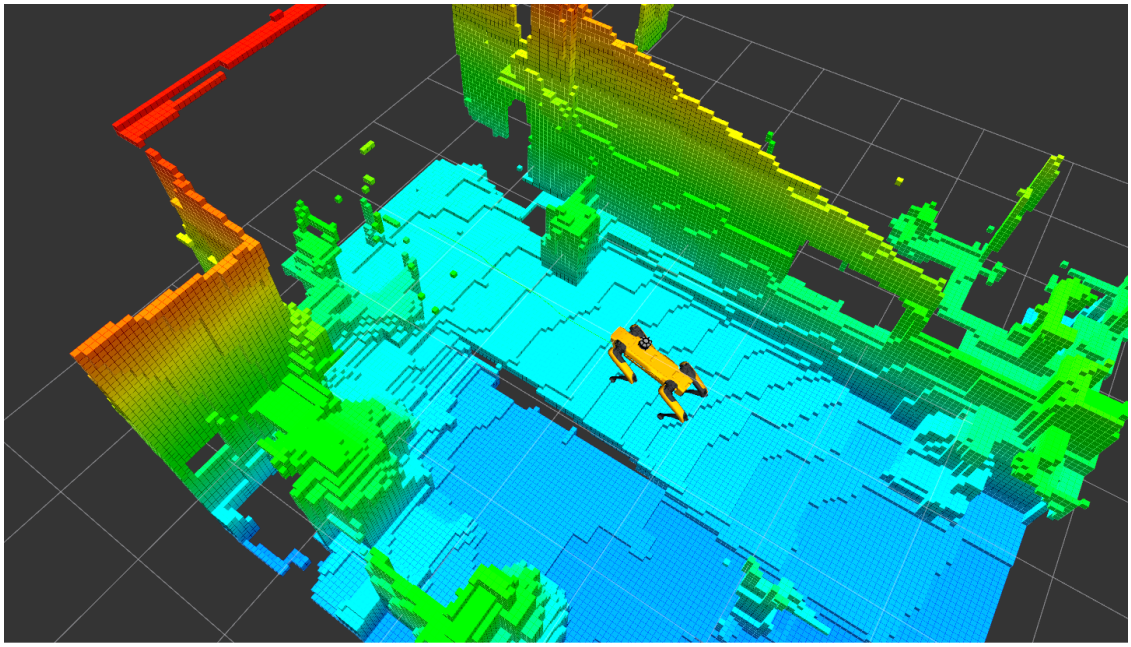


Figure 5.17: Extracted local sub-map used in the local path planning pipeline. Here, morphological pre-processing is also applied to improve obstacle extraction and surface analysis.

distance from all objects in the environment is kept. To support this process, the component is provided with additional inputs, including a local section of the current map, the paths' final target, and additional sensor data. Based on this information, the best-fitting adjusted collision-free pose is calculated and passed on to the next step of the pipeline. The following sections provide a closer introduction to the individual parts of the obstacle avoidance component.

Obstacle Extraction

In order to successfully avoid collisions of the robot with the environment, the first and most crucial step is to identify each obstacle in the environment. For this, it is first necessary to define what specifies an object as an obstacle for the robot. Since the scope of this work focuses mostly on Automated Guided Vehicles, the definition of obstacles is tailored to this use case. Overall, obstacles are defined as occurrences in the environment that the robot is unable to pass. In the most basic sense, an obstacle can be an object that is too large for the robot to traverse. Conversely, the absence of objects can also be considered as an obstacle. This includes negative space present when facing ledges or holes. The last class of obstacles considered here are slopes. Given a too-steep inclination, either upward or downward, also poses an insurmountable obstacle for the robot

The process of extracting relevant obstacles is freely configurable and can also be tailored to different robot models. An UAV, for example, would require no restrictions concerning the walkability of the surface below. Instead, it would be,

due to their increased movement capabilities, required to include the ceiling as a common obstacle As the obstacle avoidance highly relies on detecting interferences in the robot trajectory, their extraction is a crucial part of this component. Here again, the volumetric VDB-Mapping structure described in Chapter 4 is utilized to improve the overall performance. Due to the efficient data structure, the robot is able to perform exhaustive collision checks in the full three-dimensional state space.

The process itself is split into four consecutive steps. First, a small local subsection of the map is extracted from the complete map. Since the local planner only considers a few meters around the robot, the entire rest of the map can be disregarded. As only a small subspace is considered, the computational load of all subsequent steps is significantly decreased. Similar to the approach described for the global planner in Section 5.1.1, a morphological pre-processing is performed, as can be seen in Figure 5.17. This way, a smoother map less affected by sensor noise is generated, which can be used to further improve the ability to extract obstacles.

Based on this pre-processed map, an efficient ground estimation is performed. Its main goal is to extract a viable navigation corridor through which the robot can traverse. For this, the map is sampled in multiple directions using a Digital Differential Analyzer (DDA) [84]. The sampling is performed circularly around the robot on multiple heights, checking a cylindrical pattern. In general, AGVs are only able to traverse in a more or less planar fashion (i.e., in x and y direction). Since they are not able to directly move up or down, the ray casting in those directions is omitted to reduce the computational load. However, given the UAV use case, this pattern can also be extended towards a spherical form. While the DDA advances in each direction, two verification steps are performed for each traversed voxel to ensure that the position is valid. Here again, the concepts of collision and surface checks developed in Sections 5.1.4 and 5.1.5 are re-applied. First, it is checked whether the robot is in collision with any objects in the collision-space grid. Subsequently, the surface check verifies that enough walkable surface and enough space above this is available for the robot. After each successful check, the current extent of the cast ray is advanced, and the process repeated with the next position. Each valid point is subsequently stored in a list of surface points for further consideration and validation.

However, if any of the checks fail, it determines that an obstacle in this direction is present. The ray casting process is aborted as soon as this happens, or the DDA reaches its maximum extent, and the list of surface candidates is returned. In Figure 5.18, a visualization of these surface candidates is depicted. After this process is done for each direction, the next step is to extract a viable navigation corridor from the set of surface candidate lists. To achieve this, multiple additional validation constraints are applied to the surface lists. These checks are performed for each direction independently and highly depend on the robot's motion capabilities.

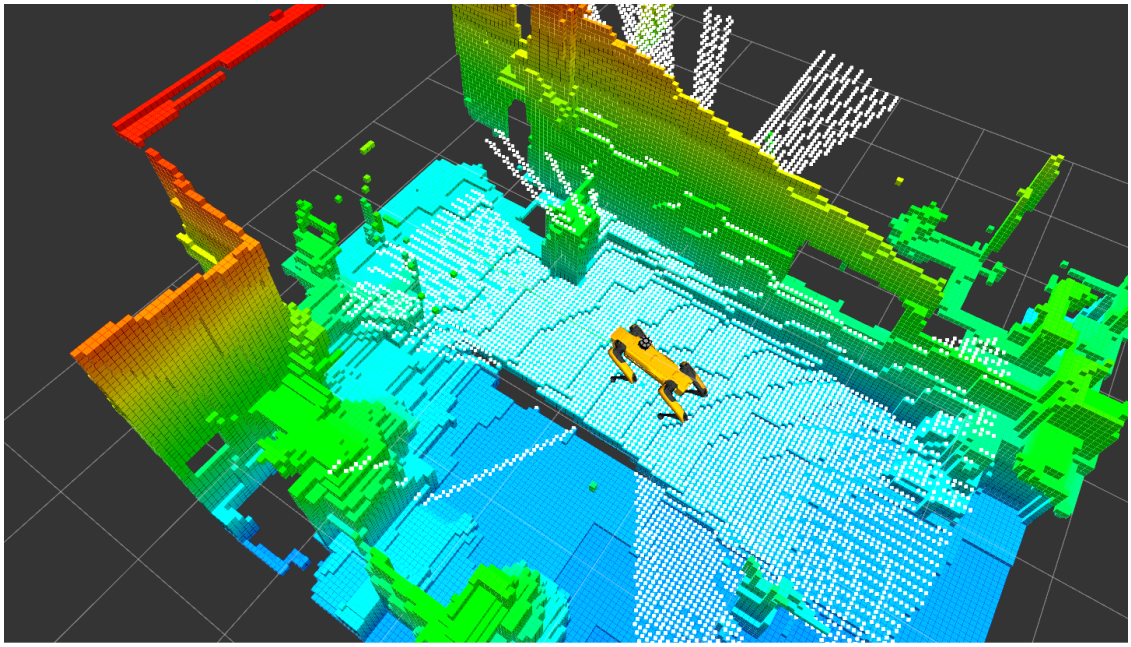


Figure 5.18: Extracted navigation corridors within the sub-map. A ray-casting is performed for multiple directions to find walkable surfaces and obstacles within the environment.

The first validation step is to verify compliance with the maximum possible step height. The extracted surface is not necessarily continuous since non-planar surfaces might contain ledges, holes, or steps. Therefore, an essential additional step is to verify the height difference between two consecutive points of the extracted surface. If the height difference of two consecutive points exceeds the maximum step height in positive as well as negative directions, the corresponding position is also considered an obstacle.

An equally important consideration is the maximum slope the robot is able to handle. Due to the nature of the discretized grid, using the slope between two consecutive points along the cast ray is not directly applicable. Here, the smallest possible incline would already be at 45°. Instead, a convolution over multiple consecutive elements is performed to gain a more reliable slope estimation without possessing the continuous height difference. Afterward, the calculated slope is thresholded against a configurable maximum slope angle. Both checks are iteratively performed, starting at the closest point to the robot and successively moving outwards. As soon as one of the constraints is violated, the process is stopped, and the offending position is considered an obstacle in this direction. This way, extracting the closest obstacle position for each direction is possible, as can be seen in Figure 5.19. Subsequently, this point is added to the list of obstacles the obstacle avoidance component considers. As the process is quite time-consuming, the generated result is not only used for the obstacle avoidance component. Instead, it is also passed down to the later described safety component to prevent redundant calculations.

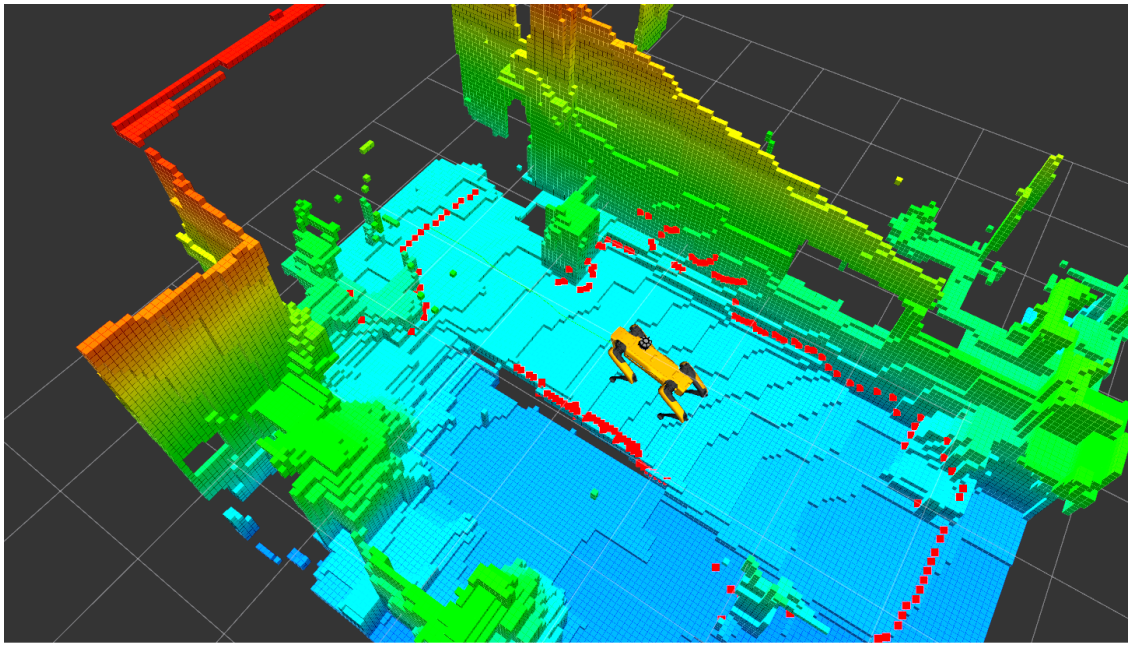


Figure 5.19: Extracted obstacle points based on the pre-calculated navigation corridors. These points are directly used in the remainder of the obstacle avoidance as well as the safety component.

Component Network

The proposed obstacle avoidance module is based on a network of smaller components. This section provides an overview of the component network used for the obstacle avoidance module. Each of these component receives an input pose, influences it individually in a certain way, and passes on the result to the next one. The main idea is to use and combine multiple individual simplistic components. This way, more sophisticated obstacle avoidance systems can be composed by intelligently combining multiple components in a complex network structure.

The components can be divided into control-flow and pose-adjustment components. Control-flow components are intended to build the network and combine different pose-adjustment components. Each control-flow component receives an input pose and returns, after adjustments, an output pose. The two main types of control-flow components are sequences and containers. As the name states, a sequence executes multiple components consecutively, one after the other. Here, the output pose of the previous component is always passed on to its successor for further adjustment. A container, on the other hand, executes multiple components in parallel. When each component has finished its calculations, all results are merged into a combined output. The combination is achieved using different merge policies for all individual outputs. Both sequences and containers can be composed of control and pose-adjustment components. By nesting control-flow components, a multi-layer network can be constructed to realize a more complex obstacle avoidance system.

The pose-adjustment components are intended as simple atomic modules that optimize the robot's trajectory in a certain way. In addition to the input pose of the sampler or other components, they can receive various inputs. This can range from inputs like map data to additional raw sensor measurements. The pose-adjustment components can be further divided into pose and direction components. Similar to the control-flow components, the pose components receive an input pose and return an optimized output pose based on the other input. However, in some instances, this simplified structure is not sufficient. A simple example is when the robot should traverse in the direction that offers the most free space. As multiple directions might be valid, a single pose adjustment can no longer represent the state space. To handle this problem, this work introduces direction components. Based on their inputs, these direction components generate a directional distribution around the robot. The distribution value represents the reward the robot receives for moving in the corresponding direction. Multiple of these distributions can subsequently be combined using a container component and specialized merge policies. After merging all distributions, the container chooses the direction in which it should travel based on a reward policy (e.g., minimum, maximum, mean, or median of the distribution). Subsequently, a pose is calculated from the directional vector, and the input pose is accordingly adjusted towards this pose.

Figure 5.20 displays an exemplary component network. A sequence composed of two nested components is used as an overall control-flow structure. First, three simple direction components are combined in a container structure. The forward component produces a distribution centered around letting the robot continue to drive in its frontal direction. Therefore, its distribution has its peak in front of the robot and gradually decreases from there on. This way, unnecessary turns on the platform are prevented, which is especially useful for non-omnidirectional platforms. Similar to this, the target pose component has its peak in the direction of the initially sampled pose of the global planner. This way, the robot is steered toward the actual target destination. The free space component, on the other hand, is more complex and utilizes the information gained by the obstacle extraction. The distribution is tightly coupled to the distance to obstacles in certain directions. As obstacles get farther away, the reward value of the distribution also increases. After merging all three distributions using a weighted sum, the maximum reward is used to adjust the input pose. Subsequently, the output pose is given to the next pose component of the sequence. Contrary to distribution components, this component aims to adjust the input pose directly. Here, the already adjusted target is additionally pushed away from obstacles in its near vicinity by using a potential field approach. Even though each of these components is, by itself, rather simplistic, combining them can achieve a more powerful obstacle-avoidance system.

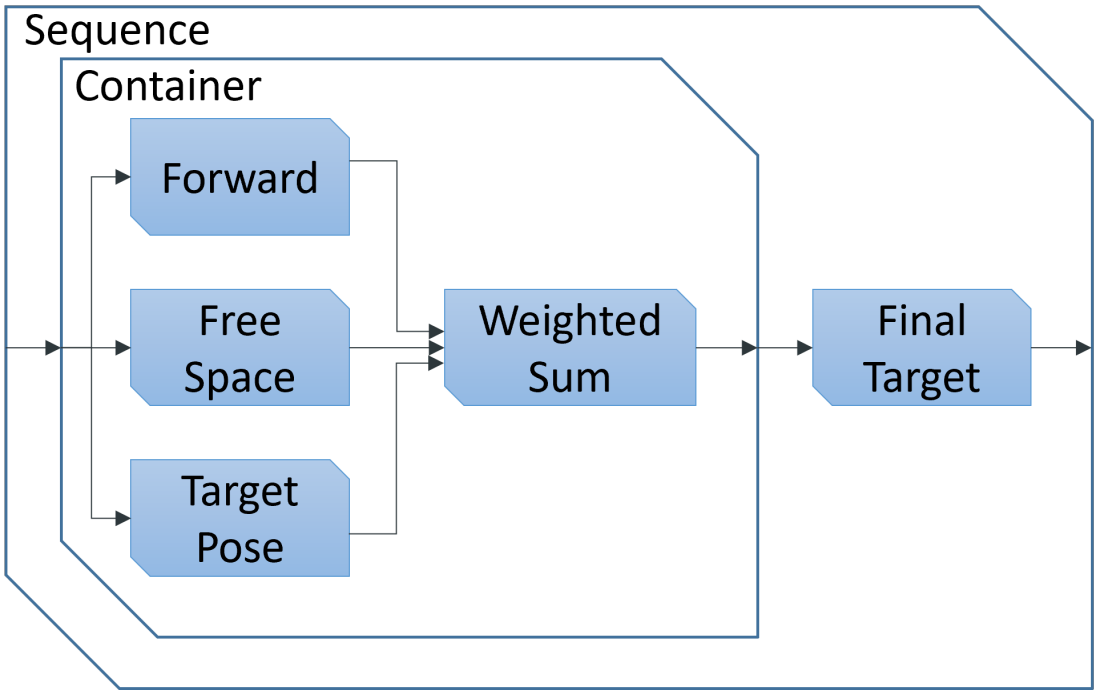


Figure 5.20: Simple component network that is able to avoid dynamic obstacles while moving toward a target. It consists of a component container with tree pose adjustment components: One that rewards driving forward, one that tries to drive toward the target, and one that drives toward the largest free space. The distributions of each of these components are merged and transformed into a goal pose. Subsequently, the last component tries to push the adjusted target further away from narrow obstacles using a potential field.

5.2.4 Controller

The final re-adjusted and collision-free pose calculated by the obstacle avoidance is subsequently passed to the controller module. One of the main goals of this navigation pipeline is to keep the framework reasonably independent of the applied robotic platform. Therefore, the in robotics commonly applied low-level twist interface is used. A twist is split into two parts: the linear and angular movement commands. The linear command consists of the three dimensions of velocities in the x, y, and z directions. The angular command comprises the angular velocity around the roll, pitch, and yaw axes. However, as most robots use this standard low-level interface, the output pose of the obstacle avoidance can not be passed directly to the robot. Therefore, the main task of the controller is to translate between a pose and a twist interface. In this instance, a multi-modal PID controller is used to achieve this. As a control error, the angular and linear distance between the target and robot pose vectors is used.

The controller itself can be further configured to closely fit the desired robot kinematics. For example, a walking robot can only walk in the x-y direction and rotate

around its yaw axis. A UAV, on the other hand, is able to control the additional z direction to adjust its height. Lastly, AUVs can freely control all three translational and rotational axes within the water. Therefore, it can be configured to cover only a subset or all six dimensions of the twist command interface individually. To fine-tune the controller, it is possible to further configure the ramp-up phase of each of the six velocities. This can be controlled by adjusting the maximal velocity and acceleration in each dimension. As a result, smooth acceleration and deceleration can be achieved independently of the robotic platform and its kinematic constraints.

5.2.5 Safety

After being processed by the obstacle avoidance, the adjusted target should result in a smooth collision-free trajectory. However, given the reactive nature of the presented obstacle avoidance and controller component, not all dynamics of the robot kinematics are covered. This could result in collisions, as the robot is, for example, not able to rotate fast enough to reach the desired orientation in time to avoid certain obstacles.

Nevertheless, the control and obstacle avoidance should not be too tightly tailored to a single robot kinematic to enable multiple robots to use the same navigation framework. To circumvent this issue and mitigate the influence of dynamics, an additional safety layer is added to the pipeline, performed after the controller calculates the necessary twist command. The main goal of this safety component is to verify and, if necessary, adjust the twist command calculated by the controller to ensure a collision-free execution of the calculated global path. As an input, the component receives all information gained by the obstacle extraction described in Section 5.2.3. Extracting these obstacles is time-consuming, so the results can be efficiently reused in the safety component without further computational overhead. Similar to the obstacle avoidance, it is also possible to integrate additional information like raw external sensor input to increase the system's reliability.

In the default configuration of the pipeline, the safety plugin generates multiple virtual fields centered around the robot's body, as seen in Figure 5.21. Each field is checked individually to verify if any obstacle currently violates its boundaries. The front, rear, left, and right fields safeguard each direction of the linear velocities. The angular velocities in clock and anticlockwise direction are regulated by combining verification of the front-left and rear-right or the front-right and rear-left field, respectively. To fit various robot models, the size and dimensions of each safety field can be freely configured and adapted. Each of the fields is present twice and divided into two categories: the safety and emergency fields. The emergency field should be configured to fit tightly around the robot's body. The safety fields, on the other hand, should contain an additional spatial margin around the robot. This division is performed to offer two escalation stages to the

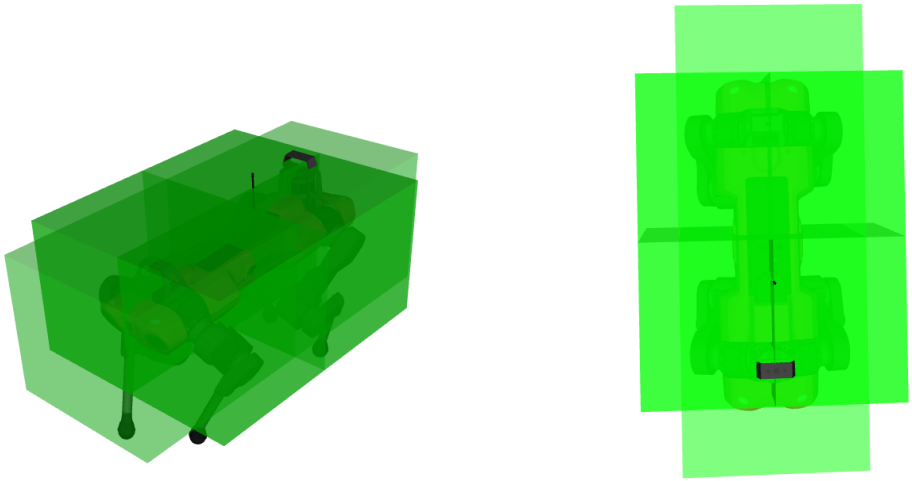


Figure 5.21: All safety fields around an ANYmal-C robot. The fields are distributed in a forward, right, left, and rear field for the directional component. A combined front-left/rear-right and front-right/rear-left fields are used for the rotation. Each of the fields is present in two escalation stages. The safety field reduces the velocity, whereas the emergency fields cut the motion in the corresponding direction entirely.

safety component. First, the safety field reduces the platform's overall velocity in the specified direction if a collision is imminent. However, at this stage, it is not intended to stop completely. Instead, the robot is allowed to use its current velocity to leave the collision trajectory. If any obstacle violates the boundaries of a safety field, the corresponding angular or linear velocity is clamped down to the reduced safety speed. All other linear and angular dimensions are untouched by the clamping operations. The clamping itself is performed as a signed operation. For example, if the frontal field is violated, only the positive velocities in the x-direction are clamped. Therefore, the robot is still able to command a negative velocity to drive backward from the obstacle. The same procedure is also performed for each of the much narrower emergency fields around the robot. However, contrary to the safety fields, in this case, the velocity in the respective dimension is strictly reduced to zero to prevent any imminent collisions with an obstacle.

5.2.6 Stuck Check

The last component of the local path planning pipeline is the stuck check. Although strictly speaking, it is not entirely a part of the pipeline structure. Instead, it is intended as a high-level control instance for each of the individual components. Its primary purpose is to verify the results of each step of the planning

pipeline and, if necessary, recover the robot if it enters a fail-state. Therefore, it receives the accumulated inputs and results of the entire navigation pipeline. From this, it has to determine whether the robot is currently stuck in its execution and thus unable to execute the global path. This includes, for example, states in which the robot cannot pass an object in its way. Another common problem in local navigation is when the robot oscillates between two objectives. In case any of these discrepancies in the robot's motion are recognized, the stuck check can abort the execution of the local path planner. Subsequently, it is able to perform pre-defined recovery behaviors to re-enable the robot to reach its initial goal. This might include behaviors such as calculating a new global path based on a more recent map. This way, it might be possible to circumvent the cause of the robot's problems to continue the traversal toward its goal. Prior to restarting the navigation process, it might also attempt to free the robot from its current stuck situation. This could include operations to clear parts of the map that might be inaccurate or perform specified movements to re-enable the robot movement.

5.3 Conclusion

The problem of robot path planning is usually reduced in dimensionality to retain efficient computation. However, this loss of dimensionality often restricts mobile robots to less complex or planar terrains. Furthermore, working on projected map representation limits their ability to correctly check for possible collisions.

To solve these issues and provide more extensive navigation capabilities, this chapter introduced concepts for efficient path-planning on volumetric data structures. The approach presented in this chapter shows the capability to find viable paths in complex unstructured environments. For this, the entire three-dimensional search space was utilized to enable each robot to perform to its fullest extent. Even though it had to consider an additional search space dimension, the developed path planner shows great potential and is able to find fast, optimal paths efficiently.

This chapter provided insights into how the initial formulation of an A* search algorithm can be combined with multiple optimizations to retain efficient computability. It showed how volumetric morphology could be utilized during preprocessing to gain a more sophisticated map representation. Additionally, multiple efficient surface and collision checks were proposed. These checks were primarily necessary to adhere to the ground restrictions of Automated Guided Vehicle. Nevertheless, the proposed approach offers high re-configurability when adjusted to various robotic platforms. Furthermore, it can be easily adapted to the use case of UAV or AUV by simply omitting extensive surface restrictions.

An alternative paradigm to handle higher dimensionality is the loss of optimality in path quality. In this approach, the optimality of the A* search algorithm is retained using a selective heuristic approach that covers multiple search strategies.

However, if the optimality of the path is not paramount, the chapter provided insights on how the concepts developed for the A* can be seamlessly transferred to alternative path-planning concepts. This was exemplarily shown and evaluated in the class of sampling-based path-planning approaches.

Even though generating this global path is essential, it is only the first step in a navigation pipeline. The second part covered in this chapter is the subsequent execution of the generated path. During this process, it is necessary to consider and adhere to previously unforeseen or dynamically moving objects in the environment. To achieve this, this thesis proposes a highly configurable and adaptable local path planning pipeline. This chapter described how the developed plugin-based structure can be utilized to adaptively react to dynamic changes. It provides the capabilities to adjust the framework to arbitrary robot models and gives insights into how it can be easily adapted to support different changing environments.

6 Evaluation

This chapter provides a detailed experimental evaluation of the main contributions presented in this work. Each of the following sections covers a different aspect of the navigation framework and all its essential parts. A combination of different use cases from artificial and real-world data was used to evaluate the developed algorithms. In Section 6.1, the performance of the mapping framework is closely evaluated. Section 6.2, on the other hand, covers a detailed evaluation of the proposed path planning concepts.

The experiments described in this chapter were all performed on a standard consumer laptop containing an Intel Core i7-6820HQ with 2.70GHz and 32 GB RAM. This system was running an Ubuntu 20.04, and all experiments were performed in the Robot Operating System (ROS) ecosystem.

6.1 Volumetric Mapping

This section takes a closer look at the developed VDB-Mapping framework. First, in Section 6.1.1, the basic VDB-Mapping framework and all its components are inspected. This is followed by a detailed evaluation of the Sub-Mapping and Remote-Mapping add-ons in Sections 6.1.2 and 6.1.3.

6.1.1 VDB-Mapping

This first section of the experimental evaluation focuses on the efficiency of the developed VDB-Mapping framework. The evaluation is supplemented with a comparison to the widely employed OctoMap [40] framework to set the results into a context. Multiple test cases were derived for this, each covering a different aspect of the mapping process. All experiments presented in this section have been performed single-threaded exclusively on the CPU. By utilizing a multithreaded approach, the efficiency of the proposed mapping framework can be further improved. However, since this would lead to less comparable results during the evaluation, this improvement was disabled. Both the VDB-Mapping and the OctoMap are set up similarly to keep them as comparable as possible. Thus, each data structure uses the same underlying data type, a 32-bit floating point, making their memory footprint directly comparable.

Real-world data from any given LIDAR sensor comes in various forms and is highly dependent on the scanned environment. Each measured point is more or less invariant of all other points, meaning it can take on an arbitrary range. However, the performance of the mapping framework is highly influenced by the distance of each individual sensor measurement. This is especially the case when comparing the computational time a framework needs for the ray casting operations. Consequently, this kind of data is not well suited to generate comparable results concerning the runtime of a mapping framework. Since the data structure performance should generally be independent of the present environment, artificial data was used in this initial evaluation. This way, the relevant variables, such as ray casting distance, resolution of the grid, and the number of integrated data points, can be precisely controlled. More specifically, this part of the evaluation used a synthetic cylindrical point cloud based around the sensor origin. The synthetic data, exemplarily depicted in Figure 6.1, was chosen this way to closely emulate current LIDAR systems. While the vertical angular resolution is fixed at 1°, the horizontal angular resolution θ of the scan depends on the chosen number of data points and number of scan lines. It is defined by:

$$\theta = \frac{360^\circ}{\frac{number\ of\ points}{number\ of\ lines}} \tag{6.1}$$

If not otherwise specified, each point is placed at a distance of $10\,\mathrm{m}$ around the sensor origin point. Simultaneously, they should still be regular enough to give an unbiased, comparable runtime estimation.

Point Insertion

The first test case was focused on the general performance of the underlying data structure. Initially, the simple point insertion into the data structure was tested to evaluate its performance. For this, a point cloud containing a static number of 40.000 data points was inserted into both data structures. The idea behind this process is that it provides a general indicator of the time necessary to insert new data into the map. In this test, the whole ray casting process was discarded to focus only on the plain point insertion. Instead, only the nodes containing relevant data were added to the map structure. This process includes generating the underlying tree structure as well as adding the corresponding nodes to the tree. Furthermore, the influence of different grid resolutions was investigated. Therefore, each test was performed multiple times while simultaneously increasing the resolution of both grids iteratively.

The resulting influence of varying grid resolutions on the runtime during point insertion is depicted in Figure 6.2. This data shows that both data structures suffer a large performance loss toward higher resolutions. This effect is caused by the necessary finer grid and additional allocated grid cells within each tree structure. It is evident that even in this case, the proposed usage of the VDB-Mapping framework provides substantial runtime improvements. However, both data

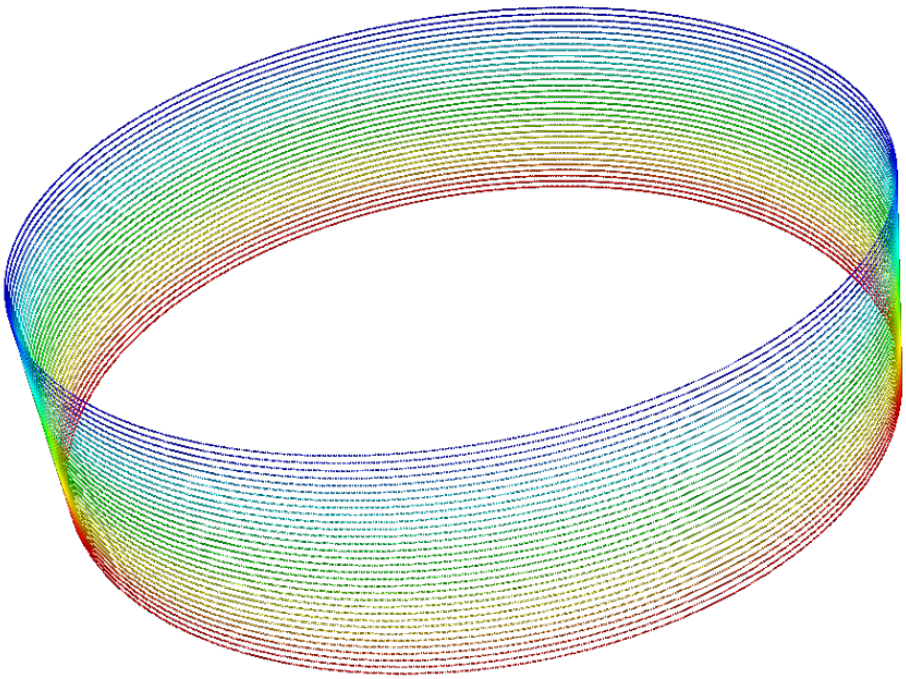


Figure 6.1: The figure depicts the artificial LIDAR test data used for the framework's basic performance tests.

structures show an exponential decay when decreasing the grid resolutions, leading to a more feasible runtime for the data insertion process. Nevertheless, the data insertion of the proposed mapping framework only requires a fraction of the time compared to the widely used OctoMap on all resolutions. Additionally, the runtime of the proposed VDB-Mapping remains almost constant for resolutions above $12\,\mathrm{cm}$.

In the second test, the influence of varying amounts of inserted points was inspected. Contrary to the first test, the resolution is fixed to $10\,\mathrm{cm}$ per voxel, while the number of inserted data points of the point cloud was gradually increased. The resulting effects on the runtime are visualized in Figure 6.3. It can be observed that both data structures scale linearly in terms of the number of integrated data points. However, it is again evident that the proposed VDB-Mapping framework is faster during the plain point insertion. Overall, the OctoMap data structure requires more time during the point insertions. Compared to the here proposed data structure, a steeper rise in runtime is noticeable while increasing the number of data points. In general, it can be concluded that the OpenVDB data structure is highly advantageous during the plain point insertion. Compared to the widely applied OctoMap framework, it provides a substantial runtime improvement over all initial tests.

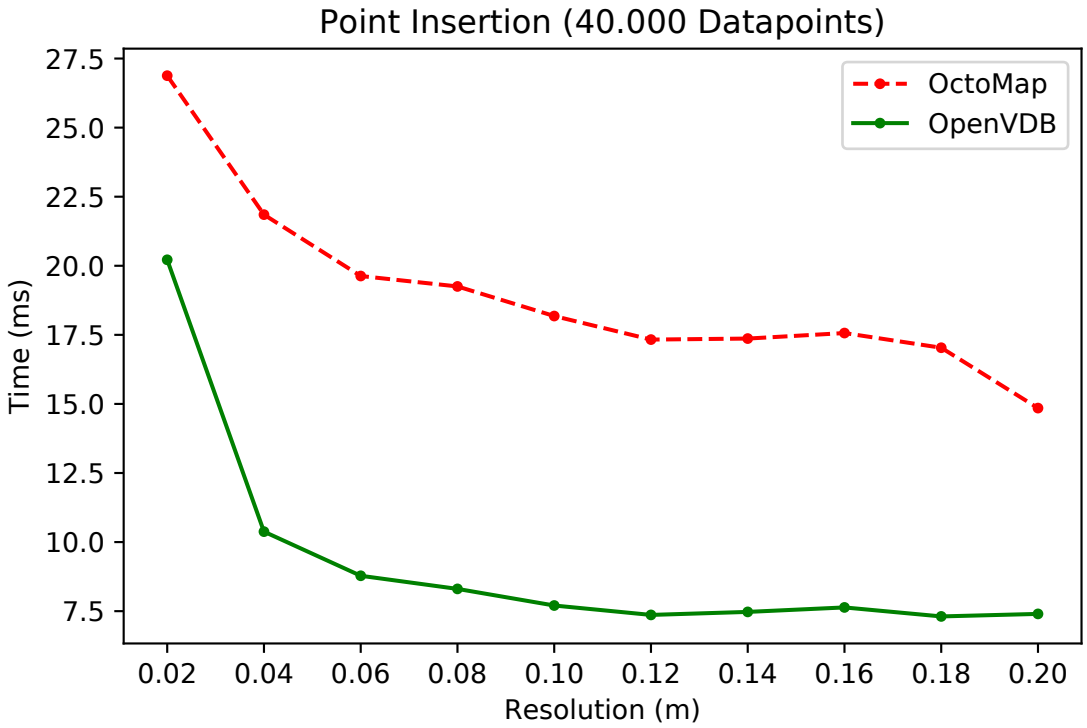


Figure 6.2: Average time to insert a set of data points into the structure. Here only the data points are updated while the ray casting is omitted. The experiments are performed using variable grid resolutions. [Grosse Besselmann et al., 2021]

Ray Casting

After evaluating the basic grid access runtimes of the underlying data structures, the following evaluation aims to investigate the ray casting capabilities of both frameworks. For this, the performance of a single cast ray was evaluated. This end point of the ray was placed at a distance of $100\,\mathrm{m}$ from the origin. During the ray casting, each traversed cell along the corresponding beam was updated according to each framework's probability update function. The number of voxels the ray had to pass during processing highly influences the general performance of the ray casting operation. Therefore, the experiment was performed multiple times with varying amounts of traversed voxels to examine how this affects the computational time. To achieve this, the resolution of both grids was increased in each new test iteration, which consequently led to a higher amount of traversed voxels. It should be noted that the same effect would be achieved if the grid maintained a fixed resolution while the range of the ray was slowly increased instead. Both operations have the same effect on the evaluation since both approaches change the number of necessary voxels that must be traversed to reach the goal voxel. Therefore, from both, it can be concluded how each of the two ray castings performs for a different amount of voxel.

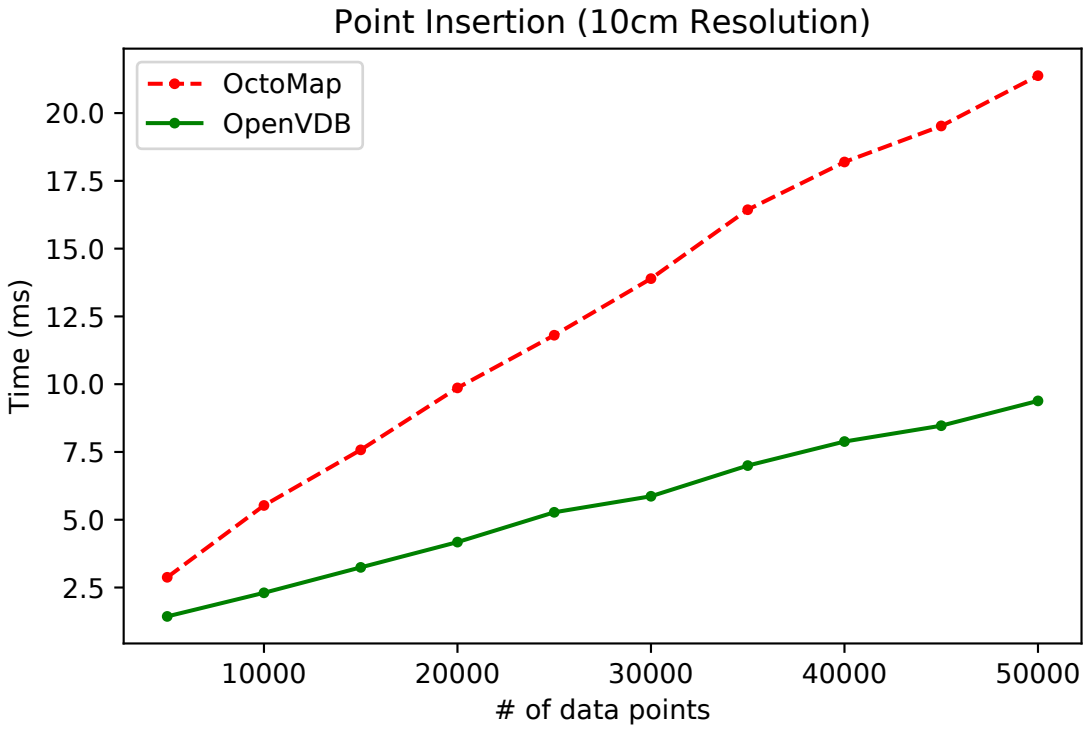


Figure 6.3: Average time for inserting data points into the data structures. The experiment is performed using a variable amount of points during each insertion operation. For both maps a linear increase is evident for larger amount of points. The VDB-Mapping structure takes less time and shows a smaller rise in computational time for larger data sets. [Grosse Besselmann et al., 2021]

The results of the ray casting runtime experiments can be observed in Figure 6.4. Here, the average time necessary to integrate a single data point into both data structures, including all probability updates, is shown. Generally, an exponential increase of the runtime towards higher resolution is observable for both data structures. This curve flattens out at a resolution of about $10\,\mathrm{cm}$ per voxel, and the gain of decreasing the resolution becomes less significant. Even though both data structures experience an exponential increase in the runtime, it is evident that the VDB-Mapping framework outperforms the OctoMap approach by far. These improved results are a direct consequence of the constant time iterators as well as the optimized ray-casting capabilities of OpenVDB. However, at lower resolutions, the curves of both data structures become more and more asymptotic towards each other. At this stage, the amount of traversed voxel is so reduced that the differences between both data structures become rather insignificant for a single ray.

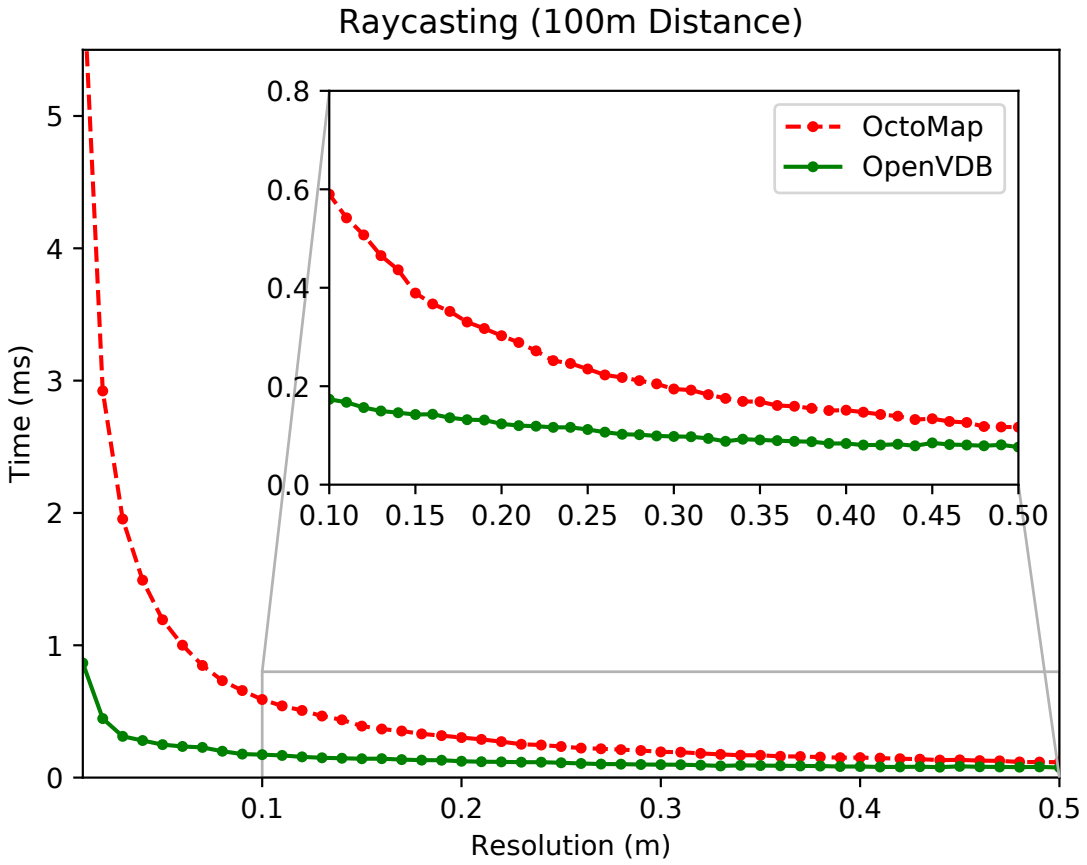


Figure 6.4: Average time to add single data points to the data structures. The ray casting includes updating all cells along the sensor beam. The target point is $100\,\mathrm{m}$ away from the sensors' origin. It shows the influence, a variable resolution has on the performance. [Grosse Besselmann et al., 2021]

Full Framework

Until now, only the basic capabilities necessary for the mapping process have been inspected. The next part of the evaluation covers runtime tests of the entire combined framework. For this, complete point clouds are inserted into the map structure. The first test examines the influence of the number of data points inserted into the data structure. In this instance, a fixed resolution of $10\,\mathrm{cm}$ for both girds is used, while the number of ray cast points is iteratively increased. The results of this experiment and the effect of a variable number of data points can be observed in Figure 6.5. It is evident that the proposed mapping framework outperforms the commonly used OctoMap during the full integration of new data points into the map. The OctoMap, experienced a steep incline in runtime during the first part of the experiment. Only after an amount of around 15000 points, the runtime curve flattened towards a linear ascent. This effect is mostly caused by the initial allocation of the data structure. Here, the world is continuously

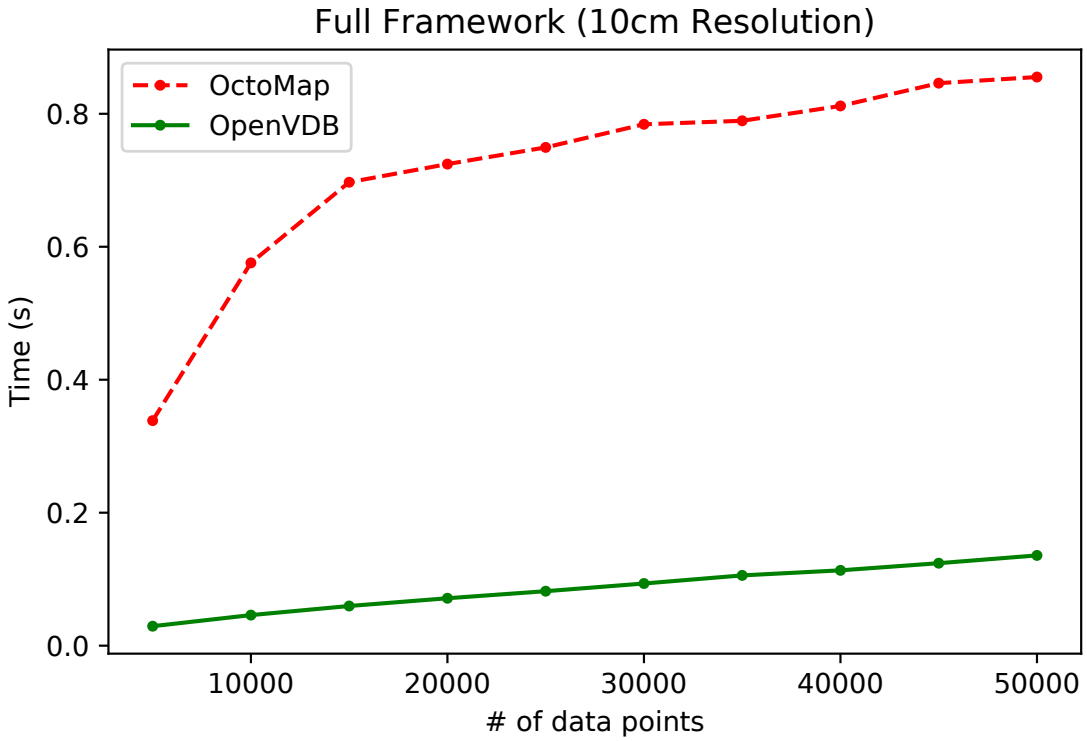


Figure 6.5: Average runtime for fully integrating new data points into the structures. In this evaluation, the update steps of all cells which are traversed during ray casting are considered. The experiments are performed using a variable amount of points. [Grosse Besselmann et al., 2021]

split into decreasingly smaller octiles. However, this effect does not occur as often above a certain amount of already integrated data, as the points more often lie within the same octiles. Conversely, the runtime of the VDB-Mapping framework increases linearly relative to the number of inserted points over the whole experiment. Even while adding 50000 data points to the map, the required time stayed close to 0.1 seconds. Compared to this, commonly applied LIDAR, such as a Velodyne VLP-16, generates around 300.000 points per second at a rotation speed of 10Hz. This means, the sensor generates around 30,000 points each 0.1 seconds. Consequently, the presented VDB-Mapping framework is capable of keeping up with the amount of data the sensor generates. In fact, the experiment even shows that it also provides a buffer of around 20000 additional points per measurement, which could be integrated in the same step. Given that VDB-Mapping can be operated in a multithreaded mode, inserting multiple sensors simultaneously on different threads becomes also possible, increasing the performance even more.

Following this, the influence of a variable resolution on the runtime was evaluated. For this, a fixed amount of 40.000 points is inserted into the map. Simultaneously, the resolution of both data structures gradually decreased. The results of this experiment are presented in Figure 6.6. Similar to the basic ray

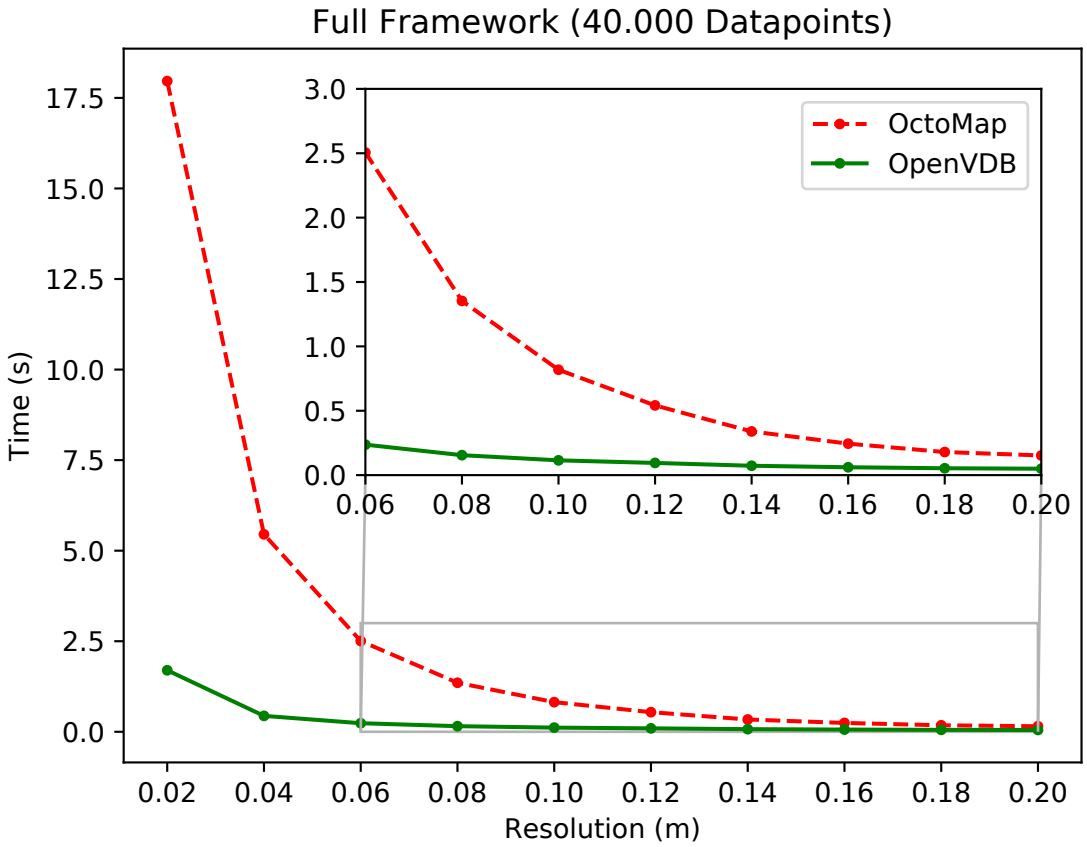


Figure 6.6: Average time for fully integrating new data points into the structure. The evaluation is performed using a variable resolution and 40.000 data points. Furthermore, the update steps of all cells which are traversed during ray casting are included. The insertion time for OpenVDB stays nearly constant, rising with resolutions higher than $10\,\mathrm{cm}$. The OctoMap experiences an exponential increase starting at resolutions higher than $20\,\mathrm{cm}$. [Grosse Besselmann et al., 2021]

casting tests, an exponential decay, and the subsequent asymptotic convergence are again evident for both frameworks. However, the proposed VDB-Mapping framework performs significantly better and shows a slower runtime while operating on high-resolution grids.

Memory Footprint

The last part of the basic framework evaluation covers the memory efficiency of the proposed framework. For this, the memory footprints of the grids generated in the previous test were evaluated. More specifically, here again, 40.000 points are integrated into a grid while continuously decreasing both grids' resolutions. This gives an insight into the influence of a variable resolution on the required memory footprint. The required allocation of memory can be observed in Fig-

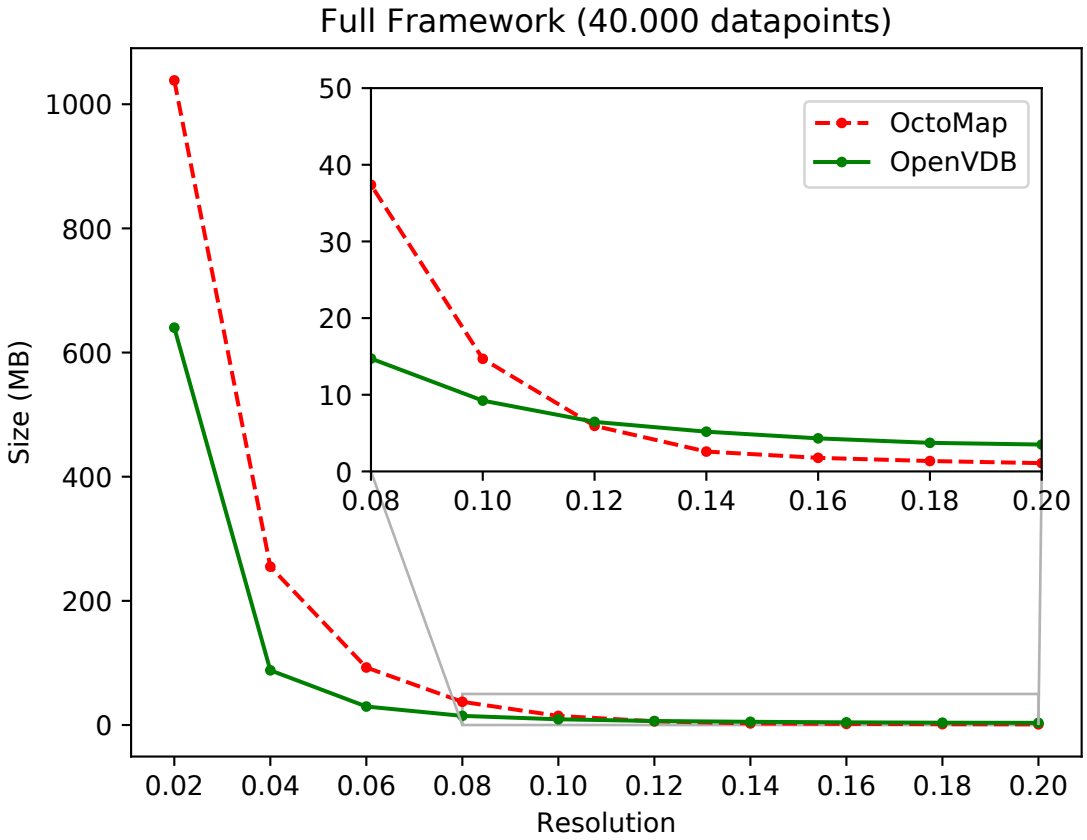


Figure 6.7: Comparison of the memory footprint using decreasing grid resolutions. The tests are conducted with the full mapping representation, includes ray casting through the entire data structure. [Grosse Besselmann et al., 2021]

ure 6.7. For both structures, the progression of the memory footprint curves looks quite similar. It can be observed that the proposed VDB-Mapping framework performs better in a high-resolution scenario. However, at a resolution of about $12\,\mathrm{cm}$ the OctoMap data structure gains the lead over the proposed approach. In conclusion, the memory footprint is for both approaches comparable. Depending on the chosen resolution, one or the other approach might be favorable.

6.1.2 Submapping

The first part of the mapping evaluation covered the basic VDB-Mapping framework using an Occupancy Grid Mapping approach. The following sections will cover the performance of various extensions made to the framework. In this section, an in-depth evaluation of the proposed sub-map extension is provided. The approach is compared against the base version of VDB-Mapping, to investigate the influence of the applied changes.

The evaluation focuses on multiple test cases, each covering a different aspect of the mapping process. Those include the integration speed of new data as well as the memory efficiency of both systems. The main advantage of the sub-mapping module described in Section 4.2 is the correct mapping and data alignment of large-scale environments. In order to evaluate the effect of these large-scale environments on the framework, the 3D dataset presented in Hess et al. [38] was used. The data set was recorded at the Deutsches Museum in Munich. It provides individual point cloud sensor scans obtained using two Velodyne VLP-16. The walked trajectory covers multiple floors and includes various loop closures at different points of the environment.

Data Integration

In the first test, the general performance of the sub-mapping module during the data integration is evaluated. The original version of VDB-Mapping without additional extensions was used as a comparable baseline. This first part of the sub-mapping evaluation covers the average time necessary to insert new point cloud data into the map. The process is performed multiple times, using different map resolutions to inspect the influence of varying voxel sizes. Each aligned point cloud, taken from the abovementioned dataset, contained about 30.000 data points. This test setup also evaluates how the runtime is influenced when updating multiple sub-maps in order to create overlapping sub-maps.

The resulting runtime curves are shown in Figure 6.8. It is apparent that the original VDB-Mapping performs nearly equal to the sub-mapping module using a single updated sub-map. Only for high-resolution maps was a small deviation in the runtime observable. This is mostly due to the additional coordinate transform necessary while working with sub-maps. As described in Section 4.2.1, the ray casting for single or multiple submaps is only performed once per submap to save computation time. However, when integrating the ray cast update grid into the respective sub-map, an additional transformation is necessary to translate the coordinate into the respective submap's reference frame. As a higher resolution results in a higher density of voxels for the same area, this, in turn leads to an increased integration time. However, the additional transformation poses only a minimal overhead. Conversely, the influence of simultaneously updating multiple sub-maps is more significant. It can be plainly observed that the required time for updating additional sub-maps is reduced compared to the first inserted submap. As stated before, the ray casting is performed only once for each input point cloud. Depending on the resolution, this expensive operation takes up roughly half the integration time. The other half is taken up by the update process for each individual submap. However, contrary to the ray casting, this step has to be performed for each additional submap individually, resulting in an increased runtime.

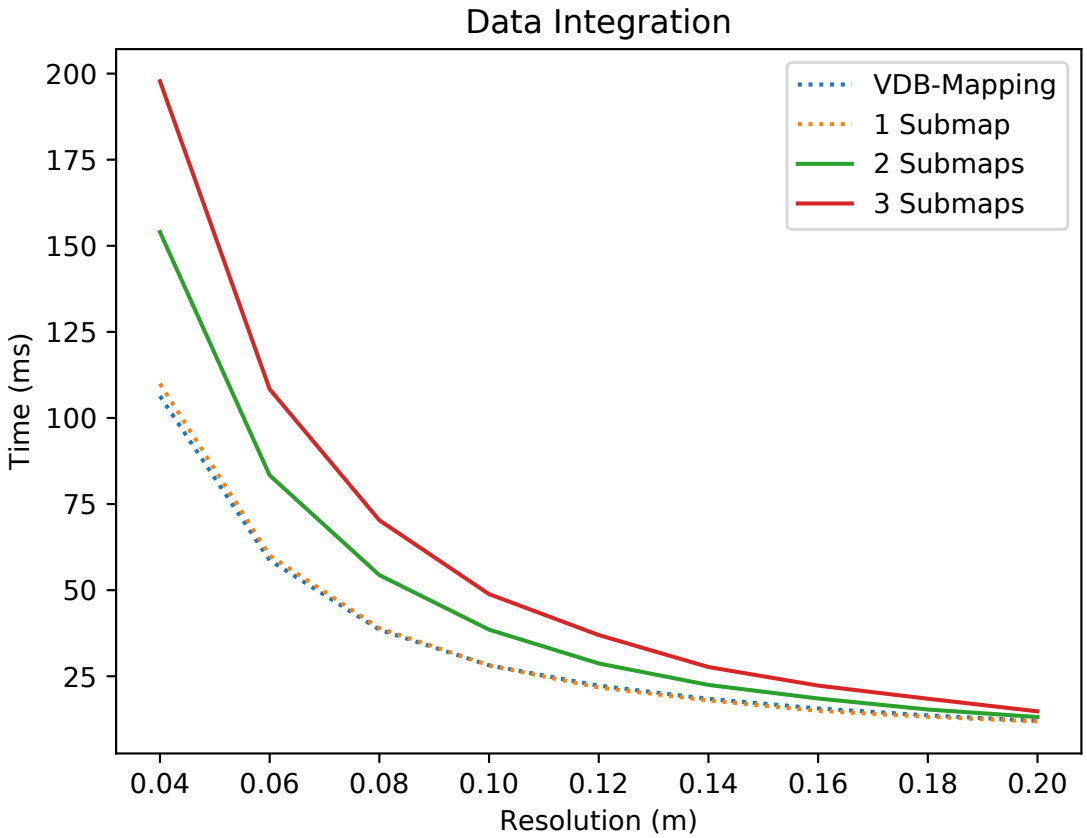


Figure 6.8: Average required time to insert a single point cloud into the map structure. The process is evaluated for multiple resolutions and compares the submapping approach to the original VDB-Mapping. It is shown how the integration time increases if multiple sub-maps are simultaneously updated. [Grosse Besselmann et al., 2023a]

Map Merging

In the second part of the sub-mapping evaluation, the merging process of the individual sub-maps is closely inspected. More specifically, the required time to merge all individual sub-maps into a single monolithic map, used, for example, during global path planning, is investigated. The experiment is split into two parts, covering both merging policies described in Section 4.2.2. Again, this process was repeated for multiple map resolutions. This is done to highlight the influence of the number of existing voxels on the overall merging process.

The first part covers the performance evaluation of merging the full map structure. This includes all occupied, free, and unknown values as well as their respective probability value. The corresponding results can be observed in Figure 6.9. They indicate that each additional submap added to the merging process requires roughly the same amount of time, leading to a linear increase per submap. However, it can be further seen that the chosen map resolution significantly influences

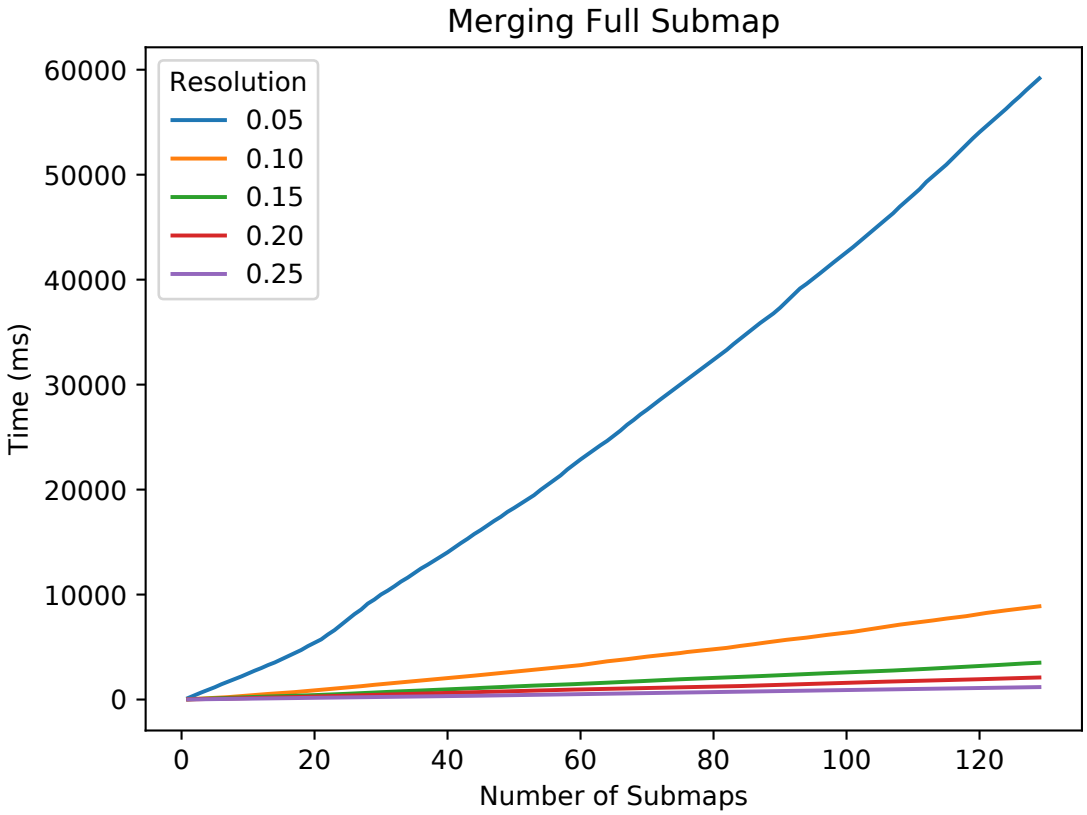


Figure 6.9: Average time necessary to merge a full map from a sub-map network depending on the number of submaps. This process is performed for variable resolutions to investigate the effect of how the amount of voxel impairs the merging performance. In this case, all information, including all free and unknown voxels, is merged. [Grosse Besselmann et al., 2023a]

the required merging time. This is especially evident in the $5\,\mathrm{cm}$ resolution curve. The main cause is that each voxel allocated in the structure has to be transformed and merged. As the number of voxels scales cubically relative to the resolution, the amount of necessary copy operations increases drastically. This is even more severe as this merge-policy also includes all ambient free space encompassing the map structure, which far outweighs the number of occupied voxels.

Subsequently, the second merge-policy was inspected. In this instance, only the active or occupied voxels were merged into the combined monolithic map. Conversely, the huge amount of ambient free and unknown space was ignored. The results of these experiments are shown in Figure 6.10. Compared to the previous merge-policy, a significant improvement in merging performance is observable. Since only the occupied voxel, representing a small subset of all voxels, needed to be merged, the process could be accelerated significantly. Even at the lowest tested resolution of $5\,\mathrm{cm}$, the entire map of the Deutsches Museum, consisting of more than 120 sub-maps, could be merged in under a second. However, even

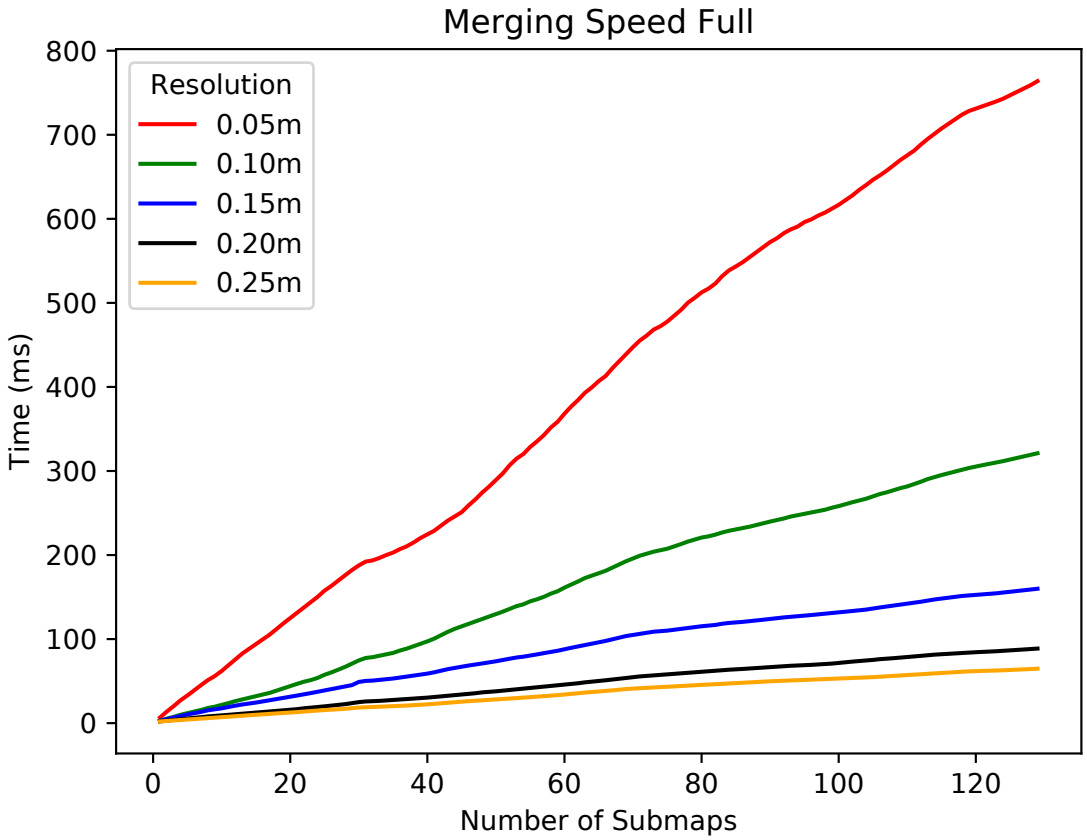


Figure 6.10: Average time necessary to merge a full map from a sub-map network depending on the number of submaps. This process is performed for variable resolutions to investigate the effect of how the amount of voxel impairs the merging performance. In this case, only the binary information, whether a voxel is occupied or not, is merged into the full map. [Grosse Besselmann et al., 2023a]

though no ambient free space had to be included during the merging operation, it can be seen that the performance influence of the chosen resolution still scales cubically.

Both of the above evaluations show that a linear increase of the merging runtime per additional submap is evident. The resolution, on the other hand, has a more significant impact on the overall performance of the merging process. To further investigate the actual influence of the resolution on the entire system, an additional test with a higher focus on the chosen resolution was performed. For this, the merging process of a single submap into the map structure is examined over multiple varying resolutions. The resulting merging process runtime over various resolutions can be seen in Figure 6.11. It can be observed that the necessary runtime to merge the entire probability framework of a single submap increases exponentially relative to the resolution. As stated before, this can be attributed to the exponential growth of the ambient free and unknown space around obstacles.

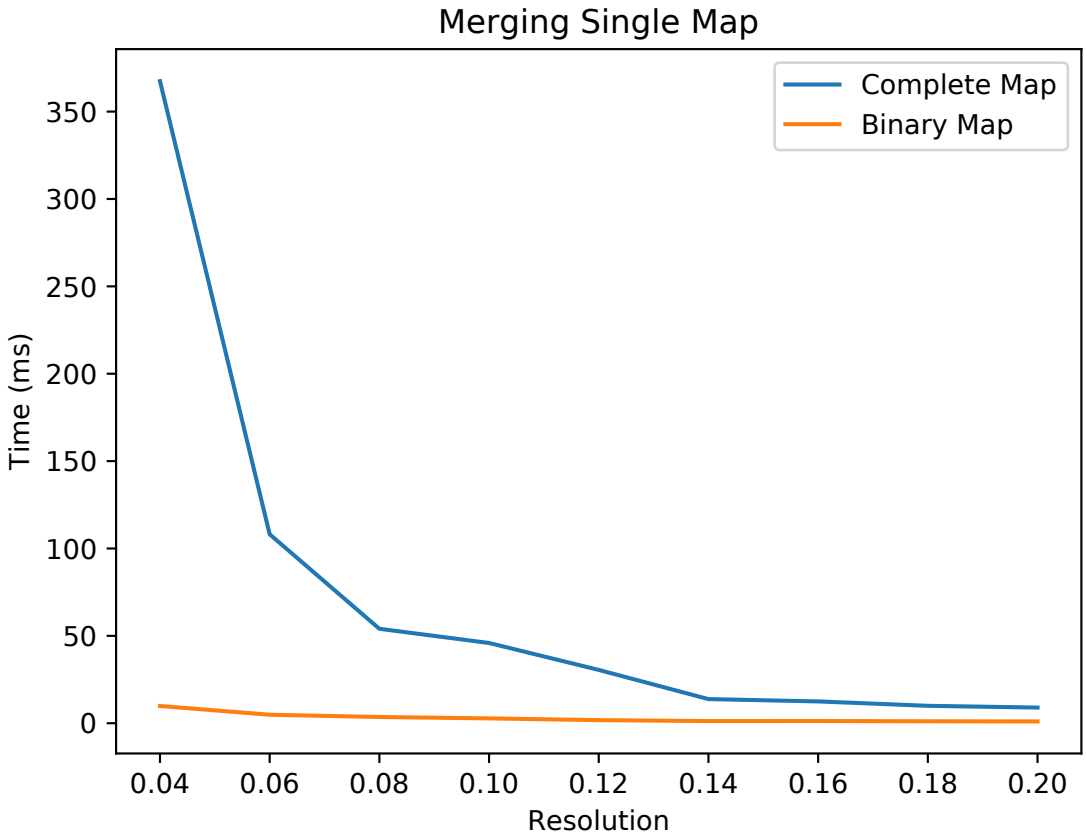


Figure 6.11: Influence of the resolution while merging a single submap. The process is repeated for both the full and fast merge-policy. [Grosse Besselmann et al., 2023a]

Conversely, the runtime of the faster merging of the active occupied values shows only a nearly linear ascent for higher resolutions. This is due to the fact that regardless of the increased resolution, the number of integrated obstacles does not rise drastically. The evaluation shows that the proposed sub-mapping approach can efficiently recreate a monolithic map, with the merging performance heavily depending on the chosen map resolution and merge-policy.

Memory Footprint

The last evaluated part of the sub-mapping is the difference in regard to the required memory footprint. Here again, the influence of varying resolutions on the memory necessary to store the entire map structure is investigated. The experiments were also performed on the indoor dataset of the Deutsches Museum. In order to calculate the required memory footprint of the entire submap structure, the accumulated memory usage of all 129 generated submaps was summed up.

The corresponding results of this experiment are shown in Table 6.1. It can be seen that, compared to the original VDB-Mapping, the submapping approach re-

Table 6.1: Memory footprint comparison (in megabytes) of the initial VDB-Mapping and the proposed submapping approach. The memory footprint of the submap network is calculated using the sum of all 129 individual submaps.

	Memory Footprint (MB)	
Resolution	VDB-Mapping	Submapping
5 cm	1714.20	3190.62
10 cm	258.31	663.72
15 cm	90.66	337.67
20 cm	45.12	245.51
25 cm	26.83	205.57

quires a way higher peak memory usage. This problem has multiple causes. First, keeping track of multiple grid structures within the map introduces general overhead. Besides the actual occupancy data, each additional submap has to store its tree structure as well as additional metadata, such as its transformation within the map coordinate system. Additionally, multiple voxels are tracked simultaneously in different sub-maps to create the desired overlap. As a consequence of this redundant data, additional memory has to be allocated. This is especially evident in the ambient free space of each map. In this instance, the number of data points is magnitudes larger than the number of occupied voxels. As a result, each individual sub-map requires massive amounts of additional memory space.

6.1.3 Remote Mapping

This section covers the evaluation of the last extension of the developed mapping framework. In this instance, the two main properties of the Remote-Mapping module introduced in Section 4.3 are covered. The first part of this evaluation focuses on a closer inspection of the necessary bandwidth to transfer the map updates over the network. The second part takes a closer look at the module's performance. More specifically, in this case, the necessary runtime during the integration of map updates is evaluated.

In order to inspect the data reduction capabilities provided by the remote updates, the previously used data set of the Deutsches Museum was used here as well. Since both the amount of transferred data and the integration time highly depend on the number of voxels within the update grid, grids of varying resolutions are used during this evaluation.

Table 6.2: Resulting bandwidth usage for grids of different resolutions.

Resolution	Used Bandwidth(KB/s)			
	State Transition Grid	Full Update Grid	Occupied Map Section	Full Map Section
2 cm	162.81	4631.24	299.45	23 552.52
4 cm	92.55	3529.62	218.69	1819.93
6 cm	83.76	2682.39	102.33	587.96
8 cm	59.92	1390.07	53.94	277.12
10 cm	45.12	714.57	36.68	164.47
12 cm	32.51	308.23	25.81	101.17
14 cm	25.50	201.72	19.29	68.35
16 cm	20.85	140.05	15.38	51.76
18 cm	17.27	105.21	12.05	40.19
20 cm	14.99	78.29	9.60	32.31

Update Bandwidth Usage

This first experiment covers the bandwidth required to transfer the generated map updates. In this evaluation, each of the different map update schemes is inspected individually for multiple resolutions. The input point clouds' average bandwidth can be considered as a comparable reference for the presented value. Each individual point cloud contains around 30.000 data points at a sensor rate of 10Hz. This results in roughly 300.000 measured data points per second, creating a network load of about 2.79 MB/s. In Table 6.2, the resulting bandwidth usage for each remote mapping update method is shown.

Given this data, it becomes evident that the grid's resolution significantly influences the necessary bandwidth of all map updates. The transferred update grids are co-aligned and have the same resolution as the map. Consequently, the reduction of the resolution over all experiments led to a gradually smaller amount of data being transmitted over the network. Compared to the baseline data rate of the input point cloud, the full update-grid requires even more data rate on higher resolutions. This is due to the fact that not just the endpoints of each ray but also all ambient voxels in between are transmitted over the network. However, it comes with the advantage that the ray casting does not have to be performed on the receiving side again. Furthermore, it offers a data reduction up from a resolution of $6\,\mathrm{cm}$ per voxel. The state transition gird, on the other hand, directly outperforms the given input point cloud data rate. A large data reduction is achieved as the amount of state-transition occurrences is significantly lower and less frequent. Furthermore, an additional data reduction compared to the input cloud is achieved due to the discretization of the grid.

The update sections, on the other hand, are not coupled to the sensor data rate.

Instead, it is transferred in a fixed interval set to 1Hz for this test case. The reduced occupied map section directly outperforms the input point clouds data rate. Since only a minimal compressed subset of voxel is transferred here, a massive improvement in the necessary data rate could be achieved over all resolutions. However, as mentioned in Section 4.3, this comes at the cost of losing the underlying probability framework of the map. Instead, only the binary occupancy states are retained. The full map section, on the other hand, required a higher bandwidth for higher resolution. This is due to the fact that the entire probability framework of all occupied as well as ambient free and unknown space is integrated into this grid.

Extraction and Integration Time

The second part of this evaluation covers the required computational time to extract grids and integrate them into a remote mapping instance. In this experiment, the effect of various resolutions combined with the different update schemes is inspected.

First, Table 6.3 shows the required computational time to extract sections from the initial VDB-Map. In this case the State Transition and Full Update grids are omitted since they are a byproduct of the mapping process and thus do not have to be extracted separately. It can be observed, that higher resolution lead to significantly higher extraction times. This is especially apparent for the Full Map Sections and should be considered when deciding for a remote update scheme. As this kind of section has to extract vast amounts of ambient free space, the remote mapping scheme is nearly infeasible for high-resolution applications. However, it is also evident that the required computational time has an exponential decline leading to feasible runtimes from a resolution of $6\,\mathrm{cm}$ and below.

Generally, a higher integration rate is always beneficial, as it allows the robot to integrate data from various remote sources. Table 6.4 shows the result of the integration time experiments on various grid resolutions. Similar to the previous evaluation, the chosen resolution directly influences the performance of each remote updating scheme. Since a higher resolution also includes a higher voxel count, an increased amount of update operations is necessary. It is evident that both the state transition grid and occupied map sections provide a significantly faster data integration. In both cases, updating only a small subset of voxels is necessary, leading to increased performance. Contrary to this, both the full update and the full map section contain the entire probability framework of the host map. Consequently, the computational load required to re-integrate them into a remote map is significantly increased. However, it still comes with the advantage of not having to perform the costly ray-casting operation on the remote system. This is especially useful for use cases, such as usage on UAVs, where computational power is sparse.

Table 6.3: Impact of grid resolution and data reduction levels on the extraction time of map sections.

Resolution	Extraction Time(ms)	
	Occupied Map Section	Full Map Section
2 cm	46.72	906.69
4 cm	10.24	269.46
6 cm	3.86	83.22
8 cm	1.67	27.58
10 cm	1.01	15.08
12 cm	0.63	7.83
14 cm	0.49	5.34
16 cm	0.37	4.36
18 cm	0.28	2.77
20 cm	0.23	2.21

In conclusion, it could be shown that the Remote-Mapping module can significantly reduce the necessary network load. Furthermore, the different update schemes enable the user to closely tailor the bandwidth to various use cases and network restrictions. Depending on the resolution and chosen scheme, it was able to outperform the input data rates by far.

6.1.4 Discussion

The evaluation presented in this section showed that the developed VDB-Mapping framework is able to efficiently generate precise high-resolution maps of large-scale environments. It was compared against the closely related Octomap framework, which is still applied in many robotic applications and is the de facto standard in volumetric map representations. The presented results highlighted that the developed VDB-Mapping far outperforms the OctoMap. Depending on the chosen use case and resolution of the map, an up to ten times faster runtime could be achieved. The fast integration of new data enables the presented VDB-Mapping approach to keep up with current sensors without discarding any data.

One of the key features compared to other frameworks is the ability to generate virtually infinite maps without resizing the underlying structure. However, this feature should be used with caution. Even though it enables the mapping of large-scale environments, there is currently no verification that enough memory space is available. Therefore, the map could run out of bounds, resulting in

Table 6.4: Impact of grid resolution and data reduction levels on the integration time on remote maps.

Resolution	Integration Time(ms)			
	State Transition Grid	Full Update Grid	Occupied Map Section	Full Map Section
2 cm	5.70	359.62	11.31	492.89
4 cm	0.77	82.69	6.36	99.69
6 cm	0.39	35.24	3.56	33.77
8 cm	0.15	14.19	2.08	17.76
10 cm	0.09	8.09	1.15	10.42
12 cm	0.06	4.55	0.86	7.08
14 cm	0.04	3.22	0.74	5.16
16 cm	0.03	2.26	0.52	4.15
18 cm	0.03	1.69	0.47	3.42
20 cm	0.02	1.27	0.34	2.75

a critical failure for the robot. One way to alleviate this issue would be to create system-specific thresholds that prevent further allocating new grid cells. The second way, at least, to mitigate or postpone this issue would be to reduce the required memory footprint further. For this, the inherent hierarchical structure of the underlying OpenVDB framework could be utilized more. It generally allows the combination and pruning of voxel into higher tree layers. In this case, only the high level node stores the single value which is then assumed to all of its children. This way all children can be deleted since they no longer contain necessary information. The pruning requires that each of the child voxel contain the same value so that no crucial information is lost. However, as the probability framework generally produces highly different occupancy values, directly combining them into a high-level node is often not possible. To combine them, an additional clustering of free and occupied values would still be necessary. Since the surface areas are often heterogeneous or not uniform it would most likely have no effect on occupied areas. Therefore, there would be no benefit in merging these voxels. The free space on the other hand could be efficiently pruned away. This way massive amounts of redundant ambient space could be removed resulting in an overall reduced memory footprint.

Setting a higher focus on the hierarchical structure would additionally enable the mapping to utilize advanced ray casting concepts such as Hierarchical Digital Differential Analyzer (HDDA) [84]. The improved form of the DDA used during the mapping is able to efficiently skip over nodes at various levels of the hierarchical tree structure. This, in turn, would drastically reduce the amount of traversed voxels during a ray casting operation. As a result, a further improved runtime during data integration and grid access operations could be achieved.

In general, it could be shown that the presented volumetric mapping framework provides an efficient alternative compared to existing 2D and 2.5D map representations. It offers multiple significant advantages over these planar map formats. Most importantly, it enables mobile robots to generate a representation of their entire operating space without any data loss due to dimensional reduction. However, it should be noted that volumetric mapping is still computationally significantly more expensive in almost all regards. Depending on the intended use case, this additional complexity might not be worth the gain of using the entire three-dimensional space. For example, applications such as autonomous robots in structured environments such as warehouses would gain few additional benefits. Therefore, it highly depends on the desired environment if a volumetric map representation or more traditional approaches should be used.

The second part of the evaluation covered the proposed sub-mapping module extension of VDB-Mapping. It enables mobile robots to generate precise and accurate maps of large-scale environments. One of the core problems in this scenario is that the estimated position of the robot is always prone to drift and inaccuracies. As a result, spatial drift is introduced into the system, which is often alleviated by using an SLAM system to re-arrange generated maps. However, this process is usually overlooked in monolithic, volumetric mapping frameworks. This approach addresses this issue by allowing the re-aligning of a network of small, consistent sub-maps after spatial misalignments have occurred. The evaluation presented in this section showed that the proposed network of sub-maps introduced only minimal memory overhead compared to the original VDB-Mapping approach. Furthermore, it became evident that the creation of these locally consistent sub-maps included nearly no additional computational load.

However, it is also apparent that the main drawback of this approach is the merging process. This process is often necessary when subsequent operations, such as global path planning, require a single monolithic map representation. While generating this monolithic map, all voxels of each generated sub-map must be considered and, if necessary, merged into the complete map. This, however, poses a huge challenge, as the number of all voxels entails a massive amount of copy operations. To optimize this process, only sub-maps with changed positions could be considered during the merging operation, as all other data is still correct. However, this would require the monolithic map to track which map provided which voxel, which could ultimately prove equally complex. More beneficial would instead be to tackle the problem from another perspective. Instead of optimizing the complex merging process, the subsequent path-planning process could be adjusted. Planning directly on the network of aligned sub-maps would make the merging process unnecessary.

The merging process aside, the sub-mapping extension provides significant advantages without severe drawbacks. It only induces minimal memory and computational overhead, which is a reasonable trade-off for the benefit of increased data handling during retrospective spatial pose corrections.

The last topic covered in this section was the Remote-Mapping module. The evaluation showed that the presented map update schemes could significantly reduce the required network bandwidth compared to sending the input point clouds of each sensor. Depending on the desired use case and network restriction, the approach could be adjusted into four levels, each providing its own sets of advantages and drawbacks.

Even though all of these schemes are viable in some regard, the first two presented approaches are highly dependent on a more or less stable network connection. However, they struggle when individual map updates are dropped due to network outages. This is especially severe for the state transition update grid. Here, each occupancy state transition is only triggered once it occurs. Even if a single update was lost during transmission, the corresponding state transitions would never be included in the map unless the state switches back again. Despite offering the most drastically decreased transmission rate, this scheme is highly situational, making it nearly infeasible for most real-world applications. This problem is less severe for the full update grid, as it contains all continuous probability updates. As a result, the remote map would still converge toward the map of the host system after a while.

The problem of network outages is less critical for the section updates. Since, in this instance, an entire sub-set of the map is transmitted and consequently overwritten on the remote side, it is less affected by network outages or single transmission losses. However, there is generally no deliberate synchronization between the two systems. As a result, the relative state between the host and remote map can not be distinguished. This issue might be alleviated by integrating the sub-map system into the remote mapping approach. As each sub-map has implementation-wise a fixed ID and version number, the remote system could request specific parts of the map. This way, the remote system could verify the integrity of its own sub-mapping network by checking if all sub-maps are present in the correct version. This would also enable the handling of complete network outages over longer periods of time as the sub-map network can then be reconstructed in post-processing.

Overall, it is evident from this evaluation that the presented VDB-Mapping framework offers multiple advantages and furthers the current state of the art in volumetric mapping. Even though there are multiple possible improvements, the mapping framework enables the efficient generation of high-resolution maps of large-scale environments.

6.2 Path Planning

This section gives an in-depth evaluation of the developed three-dimensional path-planner approach developed in this work. The presented algorithm is evaluated on both artificial data as well as on real-world scenarios. Each of the pre-

sented scenarios covers different challenges for the path-planning algorithms, highlighting their ability to handle different terrains. For each of the maps, a fixed grid resolution of 7 cm per voxel was used. This way, the individual tests had the same basic setup, making the results more comparable. To validate the viability of the presented path-planning approach, the developed volumetric path-planning approach was compared against various existing algorithms. In this instance, the metrics used to compare all approaches were the following:

- **Computation time:** How long did the algorithm require to find a valid path toward the goal node?
- **Path length:** How long was the resulting solution of the path-planning algorithm?
- **Explored Nodes:** How many grid cells have been visited during the search process?

To make the proposed volumetric approach more comparable to traditional path-planning algorithms, a two-dimensional A* was included in this evaluation. More specifically, the implementation included in the ROS [100] Navigation Stack was used. This planner is based on occupancy grid maps from which it builds a costmap [71] as can be seen in Figure 6.14a. Similar to the concept presented in Section 5.1.1, this costmap also includes an inflation layer, depicted in light blue, to decrease the computational complexity of the robot collision check. In this evaluation, the required parameters were set to closely emulate the presented three-dimensional A* inflation behavior.

For the 2.5D case, the ART-Planner proposed by Wellhausen et. [127], which is a sampling-based approach operating on a grid map [21]. The map includes an additional layer containing heights for each pixel. Both the 2D and 2.5D maps were generated from the three-dimensional VDB-Map in post-processing. In the case of the 2D occupancy grid, each obstacle higher than two voxels was projected down onto its planar footprint. For the 2.5D map, on the other hand, the height of the highest voxel in the 3D map in the corresponding x-y-coordinate was used for the elevation encoded in the map.

To offer comparable three-dimensional path planning algorithms, an efficient map interface that is usable with the OMPL [117] was developed as described in Section 5.1.6. This library offers a wide range of sampling-based planners that can be adapted to various search spaces and map formats. In this work, a PRM [49], RRT [61], RRT-connect [59], and RRT*-connect [51] are used during the evaluation. The developed interface also works directly on the underlying VDB-Mapping framework. Similar to the proposed three-dimensional A* algorithm, here as well, the collision checks and surface evaluation on the pre-processed grids as described in 5.1 are re-used. As described in Section 2.2.2, the field of sampling-based path-planning algorithms can be divided into two classes: optimizing and non-optimizing planners. The PRM, RRT, and RRT-connect are set up so that each generates a random tree until it finds a valid path from the start toward the goal position. This process is repeated ten times and integrated into

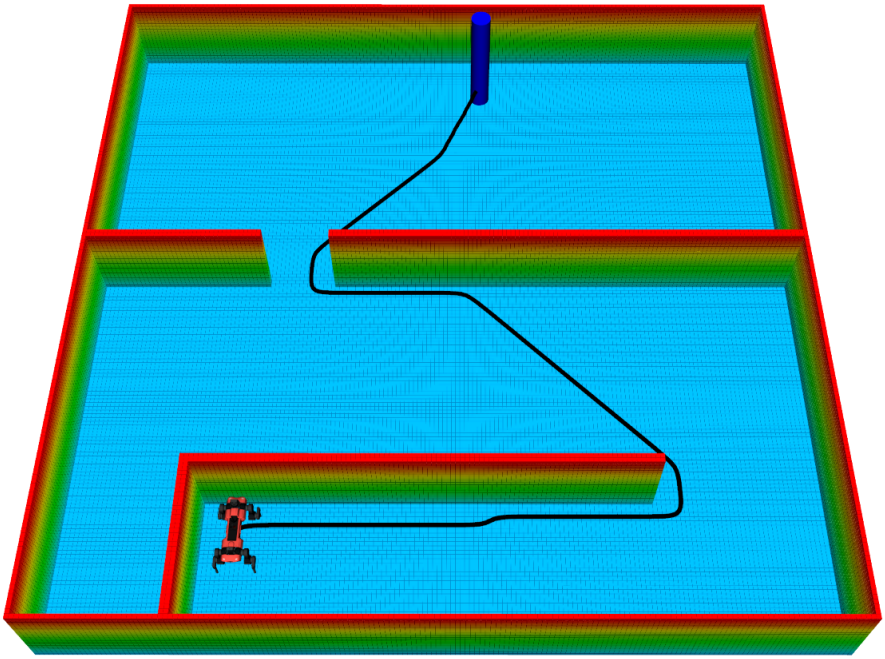


Figure 6.12: Simple artificial map to test the baseline of the planner. [Grosse Besselmann et al., 2024]

the tree of the previous run. This way, it is possible to generate a result closer to the optimal path compared to a single random path query. The RRT*-connect, on the other hand, is set up to optimize a found path for a user-specified time frame. In this instance, it is configured to search for an optimal path for five seconds. Therefore, the runtime entries in Tables 6.5 to 6.10 always stay near this fixed amount of time. The additional fraction of time stems from the setup and path extraction, which is not included in the optimization step.

Each of the proposed three-dimensional path-planning algorithms operates directly on the generated VDB-Map. This, however, required the additional map pre-processing described in Section 5.1.1. This process must be included in the path planning pipeline each time the map changes significantly. However, if it is required to calculate a path towards multiple goals on the same map, the process is only necessary once since it is independent of the start and goal pose. Thus, all planning time results in the previous tables were cleared from this pre-processing time in order to decrease the fluctuation it introduces into the system. The same goes for the pre-processing necessary to generate the 2D and 2.5D projection.

The initial scenarios presented in this evaluation are performed on maps generated from artificial environments. The primary intent of this scenario is to provide a baseline test of the presented algorithm and the underlying data structure. They cover different challenging situations to evaluate the proposed path planning algorithm's basic performance and viability. The first test case in Figure 6.12 represents a small slalom parkour. The area spans multiple rooms but does not

contain any complex obstacles. Instead, this test is intended as a general verification of the basic path-planning capabilities. The corresponding 2D and 2.5D maps are shown in Figure 6.14. Table 6.5 presents the results of this first baseline test case. The results show that the two-dimensional A* path planner performs best in this simplified environment. Compared to the other algorithms, it only requires a fraction of the necessary time to generate a path from the start to the goal position. Both the 2D and 3D A* implementations generate a generally optimal path. However, it should be noted at this point that some of the evaluated sampling-based planners generate even shorter paths. This is due to the differences in the calculation concept (i.e., allowing any-angle planning), heuristics, and restrictions of each planning algorithm. Both A* derivates are bound to move along a grid, which restricts their movement capabilities to a multiple of 45 degrees, as it is only allowed to move straight or diagonally. Conversely, sampling-based planners are free of these restrictions and can move in a straight line over longer distances. Even though this path is statistically longer, these slight deviations become insignificant since they got smoothed out during the local path execution. All other higher dimensional planners on the other hand require way more time, to generate a similar result.

The second scenario depicted in Figure 6.13 is based on a similar environment as the first test. However, in this case, the second room is connected via a small ramp between the passageway. The resulting 2D and 2.5D maps are depicted in Figure 6.14. From this, it becomes evident that the two-dimensional occupancy grid is no longer usable by the planner. Since the ramp extends above the ground plane's height threshold, the intersection becomes impassible within this projected representation. As a result, the 2D planner could no longer find a viable way through the environment, which is also visible in the results presented in Table 6.6. The 2.5D planner, on the other hand, continues to find a viable path since it is able to handle non-planar, single-layered obstacles with ease. Nevertheless, due to its sampling-based nature, its resulting path still exceeded the optimal path by more than two meters. The results of all other three-dimensional planners, on the other hand, performed nearly identical. Only the PRM's runtime increased significantly due to the more complex environment the second scenario presented. However, the length of all generated paths only showed minor devia-

Table 6.5: Planner results for the baseline test

Baseline	Planning Time	Path Length	Node Expansions
2D A*	8.41 ms	20.74 m	-
ART-Planner	276.45 ms	20.52 m	-
3D A*	215.82 ms	20.57 m	133898
RRT	104.32 ms	21.13 m	52358
RRT-connect	97.57 ms	21.07 m	58158
RRT*-connect	5028.55 ms	20.44 m	5071209
PRM	437.28 ms	20.60 m	499779

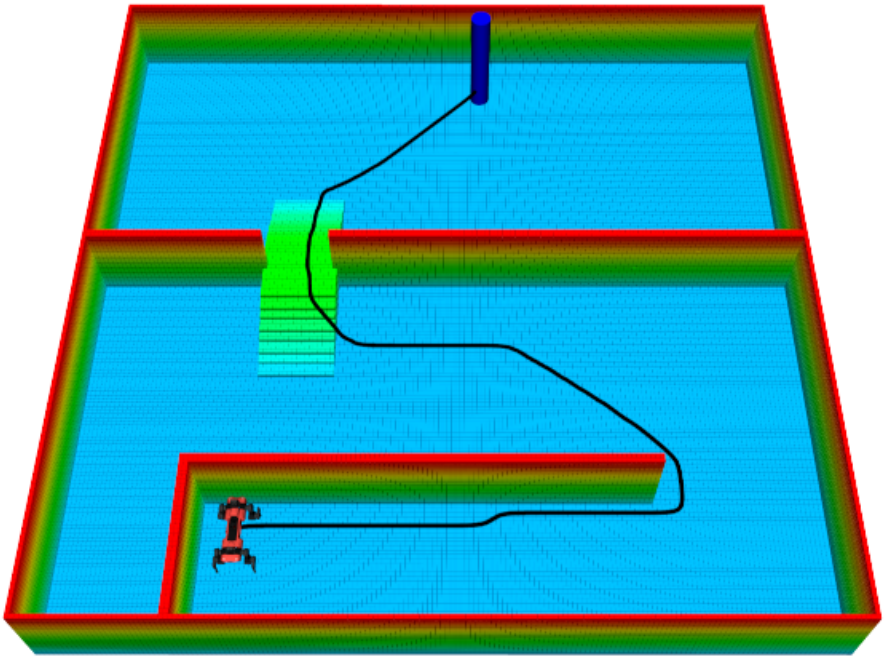


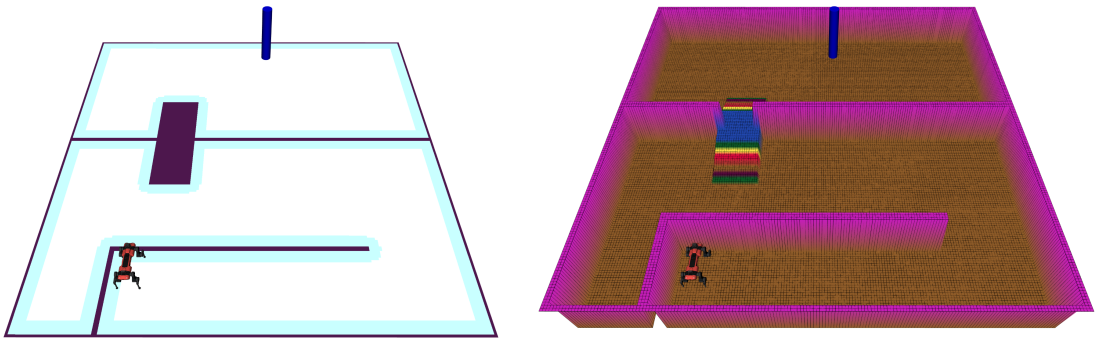
Figure 6.13: Artificial map, which contains multiple walls and a ramp connecting both rooms. [Grosse Besselmann et al., 2024]

tions.

For the third scenario depicted in Figure 6.15, the parkour map was adjusted by adding an area of unknown ground right after the ramp connecting both rooms. It is meant to showcase the capability of the different planners to handle unknown terrain within the map structure. This is especially useful when exploring previously unvisited places on a map, for which, at this point, no map data is present. As is evident from Table 6.7, only two of the evaluated planners were able to handle the unknown terrain. It should be noted that the 2D A* planner could ultimately handle unknown terrain. It fails, however, again due to the downward projection of the ramp. The only two planners who could manage this scenario were the ART-Planner and the presented three-dimensional A* algo-

Table 6.6: Planner results for the ramp test case

Ramp	Planning Time	Path Length	Node Expansions
2D A*	-	-	-
ART-Planner	225.55 ms	23.55 m	-
3D A*	241.23 ms	21.38 m	139404
RRT	159.44 ms	21.71 m	74984
RRT-connect	176.80 ms	21.93 m	86638
RRT*-connect	5024.31 ms	21.07 m	3908630
PRM	732.43 ms	21.29 m	829664



(a) Costmap (b) Elevation GridMap

Figure 6.14: Planar projections of the second test environment. It can be seen that the ramp in the Costmap is seen as an unsurpassable obstacle.

rithm. From a runtime perspective, the ART-Planner performed way better than the A*, which came at the cost of an exceeding path length to reach the desired goal. However, while comparing the necessary expanded node between this and the last scenario, it can be observed that the amount of expanded cells has nearly quadrupled. This is due to neglecting all movement restrictions within unknown spaces as described in Section 5.1.5. Evidently, the required amount of processing time directly scales with the amount of expanded cells since it involves multiple costly surface checks. This is also the reason why the integrated caching strategy introduced in Section 5.1.5 is crucial for the viability of the presented algorithm.

The sampling-based planners had severe problems finding a path, even though they applied similar techniques like the three-dimensional A*. The root of this problem is their sampling-based nature combined with the necessary restrictions for surface checks. In general, the sampling-based planner is allowed to sample random points from the entire bounding box around the current map. This includes multiple points floating above the map in free and unexplored cells. While the points in free space are easily rejected during the ground check, unknown voxels are a more complicated matter. During the A*, the movement restrictions were lifted, and unknown cells were considered walkable ground. This, in combination with the necessity to only traverse to neighboring points, leads to a uniform exploration of unknown space if encountered. In the sampling-based approach, however, the neighborhood restriction is not present, which causes severe problems when all movement restrictions are neglected. As a result, randomly sampled points above and below the explored free cell are seen as walkable and added to the tree. This leads, for example, to paths that skip walls by jumping over them. Therefore, this neglect of unknown space becomes infeasible, which results in the sampling-based planner being unable to overcome unknown ground areas.

The next test case covers the first scenario in which a 2.5D map is no longer viable no matter how the map is build up. In a staircase a three-dimensional path planning is crucial for multi-floor planning. Therefore, the maps in this next scenario

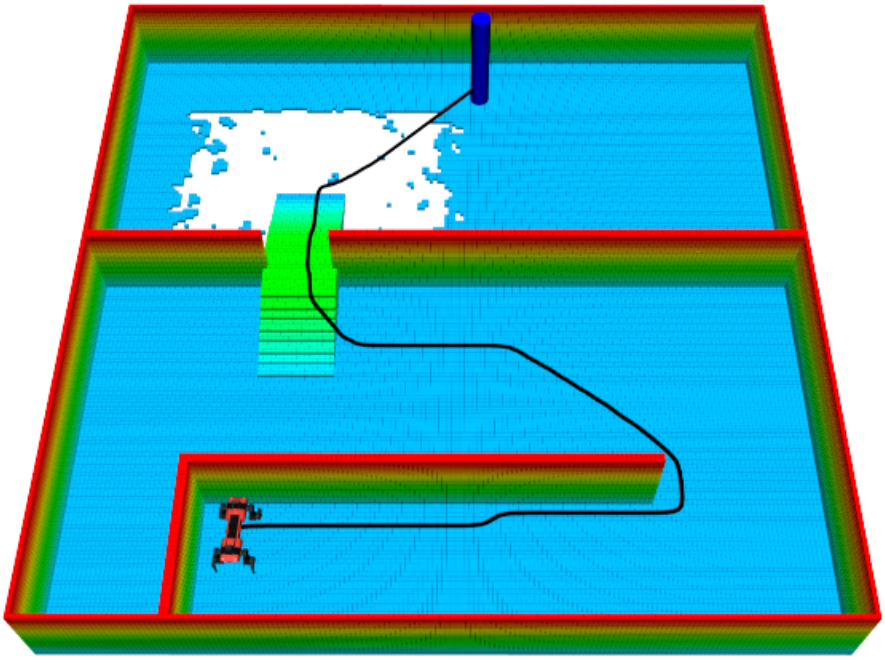


Figure 6.15: Similar map as used during the first test cases. This time, though, an area of unknown space was included after the ramp pathway. [Grosse Besselmann et al., 2024]

consist of increasingly complex artificial staircases. These maps include the one-, three- and ten-story staircases depicted in Figure 6.16. This scenario demonstrates the capability of three-dimensional path planning algorithms to plan a valid path over multi-level areas. While doing so, the path avoids obstacles and ledges with sufficient clearance to provide a safe and collision-free traversal. The multi-level staircases could no longer be projected onto a 2D or 2.5D elevation map. Therefore, the ART-Planner, as well as the 2D A*, were no longer applicable. All other three-dimensional planners were able to find viable paths through all three staircases. However, it is evident from the results presented in Table 6.8 that with increasing complexity of the scenario, the performance of the sampling-based

Table 6.7: Planner results for the unknown area test case

Unknown Area	Planning Time	Path Length	Node Expansions
2D A*	-	-	-
ART-Planner	242.27 ms	26.23 m	-
3D A*	1118.01 ms	21.17 m	448201
RRT	-	-	-
RRT-connect	-	-	-
RRT*-connect	-	-	-
PRM	-	-	-

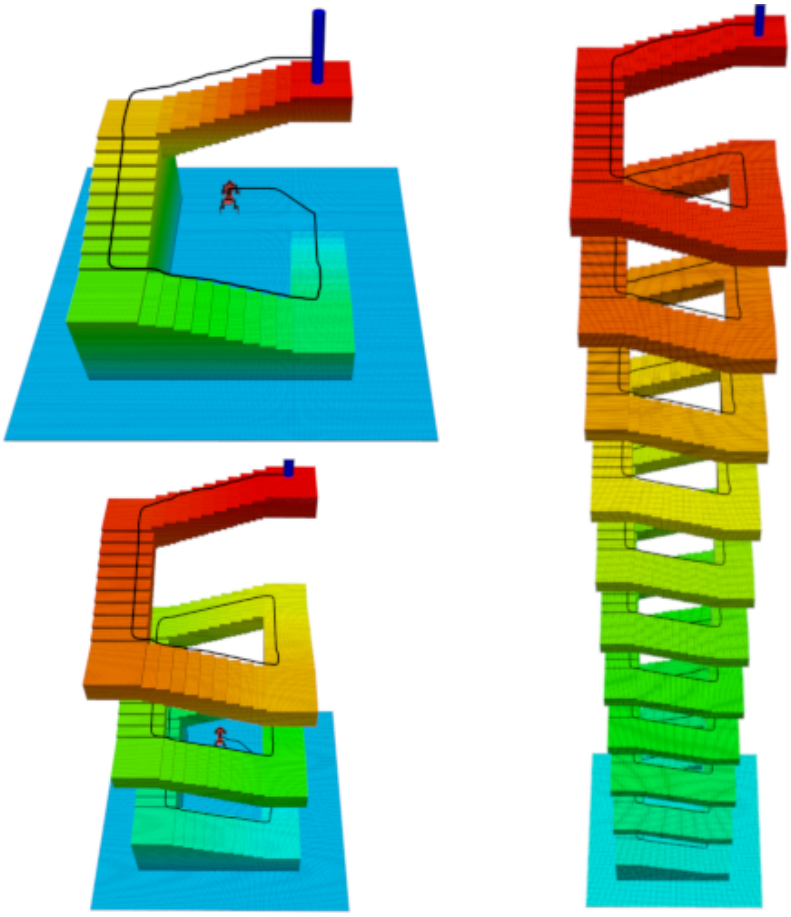


Figure 6.16: Three artificial staircases, with one, three, and ten windings. [Grosse Besselmann et al., 2024]

planner deteriorates. More specifically, this strongly correlates to the map's size. The sampling space of the planner is represented by the index bounding box of the input map. From this entire sampling space, the planner generates uniformly sampled coordinates. As the sampling space becomes large, so does the amount of invalid voxel, which might not lead to the target. This problem is even further aggravated since the amount of walkable surface in the staircase scenario is proportionally small compared to the invalid free space. Since the size of the sampling space increases cubically, so does the amount of sampled points. Therefore, the sampling-based performance deteriorates in complex, larger-scale areas. The A*, on the other hand, scales rather linearly since its exploration is bounded by the neighborhood, which is restricted by the surface area of the stairs.

All these artificially generated environments prove that the proposed planner is able to generate valid paths through different challenging scenarios. However, it still needs to be shown that the developed algorithms perform well in a real-world scenario. Compared to artificial maps, these kinds of scenarios provide multiple additional challenges. This includes incomplete maps, inaccuracies due to noise, and, in general, more unstructured surfaces. Two highly different envi-

Table 6.8: Planner results for the multiple staircase test

1 Floor	Planning Time	Path Length	Node Expansions
2D A*	-	-	-
ART-Planner	-	-	-
3D A*	439.92 ms	21.53 m	228497
RRT	659.22 ms	21.37 m	187682
RRT-connect	460.34 ms	21.45 m	137771
RRT*-connect	5028.86 ms	20.65 m	2554886
PRM	937.28 ms	21.11 m	960376
3 Floor	Planning Time	Path Length	Node Expansions
2D A*	-	-	-
ART-Planner	-	-	-
3D A*	891.37 ms	58.34 m	390042
RRT	3892.80 ms	61.98 m	556580
RRT-connect	2282.39 ms	62.19 m	438881
RRT*-connect	5033.70 ms	56.66 m	1879636
PRM	2393.43 ms	57.95 m	1939449
10 Floor	Planning Time	Path Length	Node Expansions
2D A*	-	-	-
ART-Planner	-	-	-
3D A*	2616.05 ms	207.07 m	943532
RRT	44405.41 ms	231.54 m	3753338
RRT-connect	30804.19 ms	231.86 m	3506627
RRT*-connect	5029.34 ms	205.81 m	1300827
PRM	11304.90 ms	209.29 m	7087769

ronments in challenging outdoor terrain were chosen to evaluate the performance on non-artificial maps. The maps were generated during robot deployment during outdoor missions, in which the proposed algorithms were thoroughly used and evaluated. The first real-world map was recorded on a search and rescue testing facility of the THW located in Aachern. It was generated using an ANYmal-C walking robot. The sensor setup consisted of a Velodyne VLP-16 LIDAR as well as four RealSense RGB-D cameras. All data was fused into a single VDB-Map as described in Chapter 4. In order to reach the goal, the robot had to traverse the path seen in Figure 6.17. The generated path included multiple inclines and had to travel through overgrown environments.

The corresponding results can be seen in Table 6.9. Even though the area would have been passable without giving the entire third dimension too much thought, the 2.5D ART-Planner was not able to find a viable path through the environ-

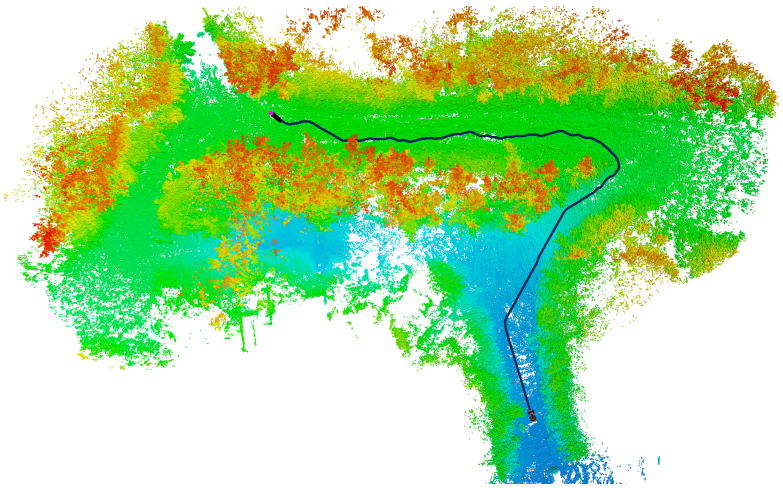
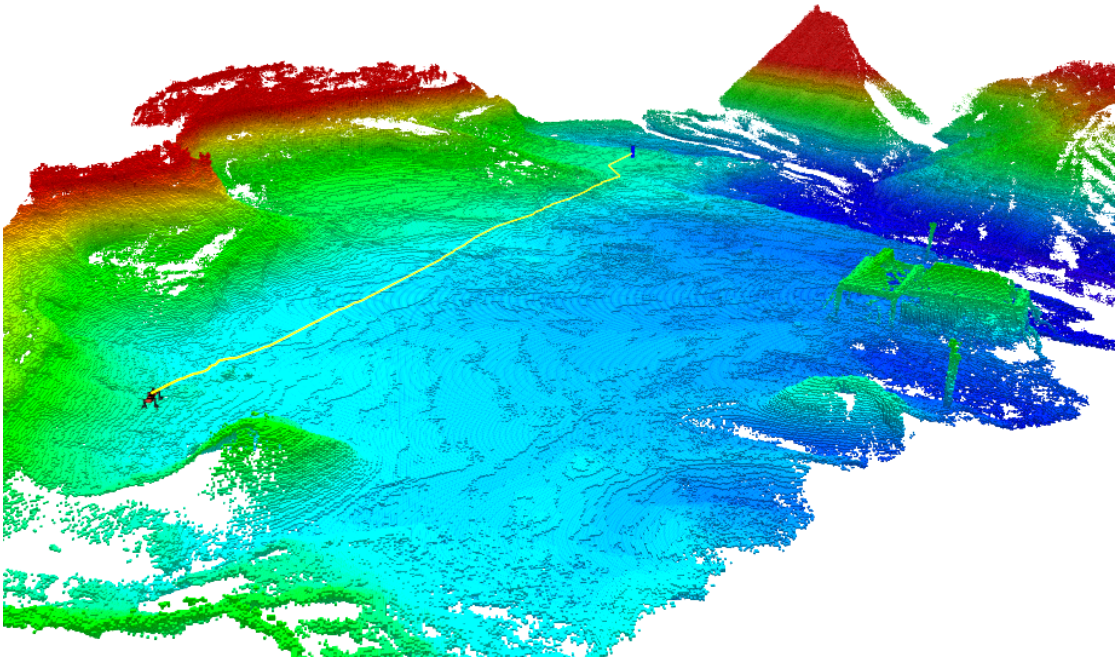


Figure 6.17: Real-world data set of a search and rescue facility of the THW in Aachern, on which the planning was evaluated. It was recorded on an ANYmal-C robot and consisted of multiple inclines and overgrown narrow passages.

ment. This had mainly two different reasons. The first one was caused by the overgrowth in the presented environment. Here, overhanging branches of trees and bushes reached into the walkable surface area. As they were projected downwards, entire areas of the map were no longer passable. This problem could be reduced by fine-tuning the parameters of the elevation map for each specific use case. The second problem of the ART-Planner is that it is more optimized towards finding shorter paths on a semi-local scale. As the size of the map increases, the runtime restrictions of the planner prevented it from finding better results. The other evaluated three-dimensional planners, on the other hand, were able to generate viable paths through the complex environment, as can be seen in Table 6.9. However, sampling-based planners generally have problems when required to find a path through small passages like corridors or door-frames. This is due to the fact that enough samples have to lie within this constricted area to be able to cross it. As the overgrowth around the main path highly constrict the environment, it basically represented a long but relatively narrow corridor. Conversely, this leads to better planning results for the more iterative three-dimensional A* algorithm. Since the small passages highly restrict the neighborhood and, in turn, the number of expanded nodes, this environment is highly advantageous for this kind of algorithm. Even though all sampling-based planners were able to find a viable path, their runtime was significantly higher despite finding only non-optimal paths.

The second real-world outdoor test case was taken during an analog planetary exploration mission. The map itself was recorded on a Boston Dynamics Spot robot. This robot was also equipped with a Velodyne VLP-16 LIDAR and five RealSense RGB-D cameras. The environment provided multiple challenges like craters, mountain ridges, dried river beds, and rough, difficult-to-pass, rocky terrain. As stated before, the ART-Planner could not find a viable path through

(a) Drone shot of the Tabernas Desert



(b) VDB-Map of the same environment

Figure 6.18: Second real-world data set on which the planner was evaluated. It was recorded in the Tabernas Desert in southern Spain during an analog Mars mission of Project IntelliRisk. It consisted of multiple hills, rocky, sandy terrain, and dried-out riverbeds. [Grosse Besselmann et al., 2024]

Table 6.9: Planner results for the first outdoor test

Aachern	Planning Time	Path Length	Node Expansions
2D A*	-	-	-
ART-Planner	-	-	-
3D A*	1388.07 ms	55.05 m	558430
RRT	6497.10 ms	56.66 m	1003178
RRT-connect	4294.15 ms	57.92 m	636973
RRT*-connect	5022.80 ms	55.26 m	1904943
PRM	5314.67 ms	56.06 m	2484131

Table 6.10: Planner results for the second outdoor test

Tabernas	Planning Time	Path Length	Node Expansions
2D A*	-	-	-
ART-Planner	-	-	-
3D A*	2275.65 ms	60.86 m	430461
RRT	1300.59 ms	62.42 m	293689
RRT-connect	1138.18 ms	62.57 m	212862
RRT*-connect	5035.11 ms	61.29 m	2780266
PRM	1359.46 ms	62.29 m	778263

the presented environment due to the area's size and its own planning time restrictions. All other three-dimensional planners were again able to find viable paths through the environment. Table 6.10 gives a detailed overview of the results. Given the presented area, the sampling-based planners had a huge advantage when considering the necessary runtime to find a solution. The terrain mainly consisted of huge freestanding areas, so the sampler could quickly find connectable sampling points on walkable surfaces. This way, the exploration tree could be constructed fairly easy, resulting in an overall better runtime. However, this comes at the cost of an optimal path length. In this case, even the RRT*-connect, which usually tries to optimize the path towards an optimal result, chose the easier traversable terrain, leading to an exceeding path length of a few meters. At this point, the choice of the planner concept becomes a trade-off between the additional time required to generate an optimal path and the additional time required to execute the additional length of the exceeding, non-optimal path. Therefore, it should be noted that, in general, each planner has its advantages and disadvantages, and it is highly dependent on the current situation which particular planner should be preferred.

Over all experiments, the results show that the planning time of each planner directly scales with the complexity of the scenario and the planned distance. From the data, it can be reasoned that the planning time directly corresponds to the number of expanded nodes necessary to find a path. This is mainly due to the

Table 6.11: Ground cache results for all test cases

Ground Caching	Ground Checks	Cache Hits	Cache Hit Rate
Baseline	335968	311216	92.63%
Ramp	351334	325265	92.58%
Unknown Area	966521	916115	94.78%
Stair 1 Floor	595693	548625	92.09%
Stair 3 Floor	1040283	954833	91.79%
Stair 10 Floor	2577979	2356077	91.39%
Aachern	1338101	1242874	92.88%
Tabernas	953164	877415	92.05%

costly ground checks performed for each expanded or sampled node. To mitigate this problem, efficient ground caching was introduced using the underlying VDB-Grid presented in Section 5.1.5. Table 6.11 lists the amount of performed ground checks, cache hits, and the resulting cache hit rate for all presented scenarios while using the three-dimensional A*. In each case, the hit rate was well above 90%, leading to a huge increase in performance. While the initial ground check involves several complex operations, the cache lookup becomes, thanks to the OpenVDB data structure, a simple $O(1)$ lookup.

6.2.1 Discussion

This section took a closer look into the feasibility and performance of the developed three-dimensional path-planning concepts based on the VDB-Mapping framework. The presented data shows that the presented optimizations to the planning process significantly increase the performance of three-dimensional path planning, leading to viable computation times.

Despite the additional computational load, the presented planning concepts could generate optimal paths through complex, large-scale environments. Compared to traditional planar path-planning algorithms, the presented volumetric concepts enabled robots to overcome unstructured terrain easily. Thus, it becomes a trade-off between the robot's capabilities to handle unstructured terrain and the present runtime restrictions.

It could be shown that the developed concepts work on the initially intended graph-based A* algorithm and are also extendable to all sampling-based planners in the OMPL. During the evaluation, it became evident that each presented planning approach provided different advantages and use cases. The graph-based A*, provides a cost-optimal path by design, whereas the sampling-based approach only provides near-optimal solutions. This optimality, however, comes at the cost of the necessary runtime. Due to their sampling-based nature, the OMPL planner provided a path faster than the comparable A* in most scenarios. As they are

able to quickly step over large areas of free terrain in a straight line, the number of considered grid cells is reduced, leading to a shorter overall planning period. Nevertheless, there are instances in which the sampling-based nature also proves to be a huge disadvantage. In general, the sampling-based approach uses the entire bounding box of the map as sampling space. As this bounding box can be virtually infinite, so is the sampling space. However, this leads to a drastic loss in performance as too many invalid samples are generated. Contrary to this, the graph-based A* provides faster results in this scenario. As the search space is bound to the neighborhood function, the searched areas slowly increase iteratively. This results in a smaller and more locally constrained search space than the sampling-based approaches. Additionally, due to its iterative nature, it has significant advantages in constricted areas. In contrast, the sampling-based planners depend on sampling the exact position within this navigation corridor in order to pass it. Due to the random nature of sampling-based approaches, this might lead to a drastic increase in sampled points, leading to decreased performance.

One of the most significant advantages of using the proposed A* derivative is its capability to seamlessly plan paths through unknown environments. This is especially useful in an exploration context where no map data is available beforehand. Here, the planner is strictly required to plan outside its constricted borders. The sampling-based planners, on the other hand, had massive problems in this domain.

Contrary to this, the sampling-based planners provided significantly faster processing and were better suited for tasks such as high-level mission planning. During the sampling based path planning, the pre-processing took a substantial part of the runtime in relation to the complete process. Since the pre-processing step must only be performed once for each map, these planners are even more helpful for multiple-query operations. This provides a large opportunity to use them in the context of high-level mission planning as described in Schnell et al. [Schnell et al., 2023]. Here, the goal is to find the most cost-efficient route between multiple waypoints. In this instance, the sampling-based planner concept could pre-process the map once and subsequently handle a batch of path requests efficiently and fast in parallel.

Ultimately, the choice between these two paradigms is a trade-off between optimality and computational time. On the one hand, a reduction in computational time is always desirable. However, if the path is significantly longer and thus the resulting path execution takes longer, no advantage is gained. This is especially true if the present terrain is rough and unstructured, as traversing additional ways takes even more time. Contrary to this, the robot can most likely compensate for the additional planning time by using an optimal path with lower costs.

This work and the presented evaluation were primarily focused on ground robots. However, it is also possible to utilize the presented concepts directly for robots with higher movement capabilities, such as Unmanned Aerial Vehicles or Autonomous Underwater Vehicles. All presented concepts were designed to also

adhere to their motion restrictions. However, these robots were still not considered further during the course of this work. One of the main conclusions of the presented evaluation is that the amount of voxel considered during a planning space scaled directly with the necessary computation time. In this regard, the extensive ground checks provided the main perpetrator. As UAVs are not bound to the necessity of walkable ground, these verification checks could be simply skipped. Instead, only the collision check would remain necessary, resulting in a massive reduction of considered voxels. As a result, the presented planning concepts would scale significantly better when used on UAVs or AUVs. This advantage could be even further increased by concentrating more on the hierarchical nature of the map structure. If no detailed, per-voxel ground check is necessary, the collision check could be performed on a higher tree level. As a result, the number of necessary validations during path planning would significantly drop, leading to a massive increase in performance. However, all these considerations only apply to the more straightforward use-case of flying or diving robots, where no ground restrictions are necessary.

Lastly, it should also be noted that even though the three-dimensional approach provides significant advantages, it is not beneficial in all scenarios. If the robot is used in simple, structured, mildly uneven terrain, there would be no additional benefit to adding the full volumetric space. Instead, it would be beneficial if the robot would continue to operate using simpler 2D or 2.5D methods. Nevertheless, the evaluation presented in this chapter proved that using the developed optimizations to the traditional A* algorithm enabled mobile ground robots to generate optimal paths through dynamically changing, rough terrain. These improved capabilities significantly increase the domain in which autonomous operation of mobile ground robots is possible. It was further shown that the developed concepts could be easily and efficiently transferred to other path-planning algorithms, such as sampling-based planners. This gives the presented approach the flexibility to handle different situations and use cases in various ways.

7 Conclusion

Autonomous navigation is a crucial prerequisite in the overall process of making robots independent of human supervision. The robot must be capable of self-reliantly moving between various locations where parts of an overall task must be performed. Huge advances have been made in this research area for structured, planar environments such as warehouses. However, the utilization of the full three-dimensional space is mostly overlooked and neglected. To enable robots to be truly autonomous, they should be able to overcome any given terrain and situation. This thesis focused on the three integral problems posed by autonomous robot navigation:

- **Volumetric Mapping:** The ability to generate a three-dimensional map from sensor data, which can be utilized in the subsequent path-planning tasks.
- **Global Path Planning:** The ability to calculate an optimal, collision-free path through arbitrary environments over unstructured terrain.
- **Local Path Execution:** The ability to execute the previously calculated path while simultaneously handling dynamic changes of the map and preventing any collisions with the environment.

This work combines these three features in an end-to-end navigation framework that utilizes the entire three-dimensional search space. The presented navigation framework is generally robot agnostic, meaning that it can be easily deployed on arbitrary robots, independent of their specifications, kinematics, or sensor setup. Ultimately, it provides the means to build autonomous navigation capabilities for mobile robots in unknown and complex environments.

Chapter 1 gives a general overview of the challenges of three-dimensional navigation. Furthermore, the relevant research questions and goals of this thesis are presented. Chapter 2 provides a deeper introduction to the foundations of autonomous robot navigation related to this work. The chapter first provided deeper insight into the topics of mapping and path planning. Following this, an introduction to the highly utilized OpenVDB data structure underlying the mapping framework is given. Here, multiple concepts and implementation details, as well as their implications in the context of navigation, are discussed.

Chapters 4 and 5 present the main contributions of volumetric mapping and path-planning. First, in Chapter 4, the developed VDB-Mapping framework and its additional extensions are described in detail. Section 4.1 presents the basic

structure of the mapping framework and focuses on how the data could be efficiently added to the representation. Subsequently, the Sub-Mapping module is described in Section 4.2. This extension enables robots to build maps of large-scale environments and handle spatial displacements caused by pose estimation errors. Section 4.3 covers the framework's capability to efficiently transfer partial map updates over restricted networks. Chapter 5 focused on the developed path planning concepts based on the presented VDB-Map representation, both on a global and local scale. These presented algorithms enable robots to plan paths efficiently, even through rough, unstructured terrain. The experimental evaluation of the proposed approaches is detailed in Chapter 6. Here, multiple experiments were used to prove the feasibility and quality of the three-dimensional navigation concept. The initial part of the evaluation tested the developed algorithms thoroughly in artificial and safe laboratory environments. The second part focuses on real-life scenarios by testing it in multiple outdoor missions. These include, for example, an analog mission in the Tabernas Desert and a mock-up mission on a moon-like surface. This last chapter gives an overall summary and discusses the initial research questions presented in Section 1.3. Subsequently, it concludes with the current limitations of the presented approach and derives further research opportunities.

7.1 Contribution

The scientific contribution within this thesis is primarily focused on three open research questions. In this section, each of these questions is revisited and discussed in the context of the concepts proposed in this work.

Research Question 1: How can the world be efficiently transformed into an accurate environmental representation suitable for robot navigation in the full volumetric space?

The mapping algorithm and representation constitute one of the most crucial features in the operations of autonomous robots. Since it builds the foundation of all subsequent path-planning and execution processes, it is of the utmost importance that it contains as much information as possible. Therefore, the mapping process should focus on including the full three-dimensional range of the environment.

In the course of this work, the VDB-Mapping framework was developed in order to generate precise maps of dynamic, unstructured environments. One of its main features is the fast and efficient data integration. Compared to currently available volumetric mapping frameworks, the developed approach is able to integrate data in real-time. More specifically, this means that it can easily keep up with the data rates of modern sensors. This way, no sensor data is dropped, which results in more accurate and complete maps of the environment.

One of the main challenges while mapping large-scale environments are spatial pose estimation errors, which accumulate over time. An accompanying SLAM

system can mainly reduce these from a pose estimation point of view. However, these spatial corrections are not directly transferable to monolithic map representations. Instead, the proposed mapping framework offers the capability to perform the mapping on a network of locally consistent sub-maps. This sub-map network is only loosely aligned and can be re-arranged after larger position shifts occur. As a result, the mapping is able to handle retrospective pose changes of any underlying SLAM representation.

These volumetric maps are, however, not only necessary for the robot itself. Instead, it is often necessary to exchange map data between different systems. This includes tasks such as visualizing the data or creating a combined map from a heterogeneous team of robots. To provide the capability to do so, VDB-Mapping offers the possibility to exchange and transmit partial maps efficiently. In this instance, a significant focus is set on data reduction capabilities to handle highly restricted networks.

To enrich the map representation with additional information, it is also possible to store arbitrary data within each individual node of the tree structure. This enables robots to include pieces of information, such as specific risks or semantic annotations, into the map.

Research Question 2: How can ground-based mobile robots autonomously calculate efficient and collision-free paths through challenging and complex terrain?

The second large part of autonomous robot navigation is the capability to safely plan a path from one point to another. In currently applied systems, this process is usually reduced to an easier planar use case. However, in order to be able to travel through unstructured environments, it is necessary to utilize the entire three-dimensional search space of a robot. To achieve this, a global path planning algorithm based on a traditional A* was developed. This well-researched path-planning algorithm was adjusted to work directly on the presented VDB-Mapping data structure. However, one of the main problems of operating on volumetric maps is the increased search space dimensionality. As the available search space increases, so does the necessary time to find an optimal path between poses. To overcome this issue, this thesis proposed multiple optimizations leveraging properties of the underlying map structure to increase search performance. Furthermore, a high focus was set on the movement restriction of Automated Guided Vehicles and how the necessary ground validation steps can be efficiently performed on the VDB-Mapping structure. Despite being developed initially for the A* search algorithm, it was shown that the basic concepts can be easily adapted to different planner paradigms. In summary, the developed path-planning concept enabled mobile robots to generate efficient, collision-free paths over arbitrary, unstructured terrain.

Research Question 3: How can mobile robots efficiently traverse arbitrary three-dimensional paths through unstructured, dynamic areas without harming people, the environment, or themselves?

Given this optimal global path, the last part of the navigation process is executing the robot's path. As the global path is usually calculated on stale data, dynamically changing objects or persons are mostly not covered. As a result, the global path is not strictly collision-free. To handle this problem, in this thesis, a modular local path planner was presented. This local planner was also based on the highly efficient VDB-Mapping framework. Utilizing the fast grid access and efficient ray casting operation, the planner is able to cope with highly dynamic environments. Since the planner uses the entire three-dimensional volumetric space, it is able to plan paths through complex, challenging terrain. To efficiently cope with dynamic and unforeseen obstacles, the planner provides a complex network of obstacle avoidance components that can be configured for current use cases or scenarios. This way, the presented approach is highly adaptable to provide a safe and reliable path execution on arbitrary robots in various environments.

7.2 Further Research

In the thesis, multiple concepts were presented and discussed advancing the current state-of-the-art in autonomous robot navigation. The accompanying evaluation underlined the feasibility of the proposed methods and showed improvements to the current state of the art that were achieved. However, multiple open research questions remain to improve these results even further.

Mapping: One of the most crucial parts of the entire volumetric mapping process presented in this work is the underlying OpenVDB data structure. This data structure enabled the fast and efficient integration of new sensor data. Recently, Museth developed in combination with NVidia NanoVDB [85], which presents an adjusted OpenVDB derivative, which is able to directly utilize the computational power of GPUs. This enables the framework to leverage the advanced ray-casting abilities present in current graphical processors. Utilizing this new version of OpenVDB could further accelerate and parallelize the ray casting and improve the data integration capabilities.

One drawback of the presented remote mapping capabilities is the missing synchronization between the host and remote map. As a result, both maps can diverge significantly over time, depending on the amount of lost data due to network outages. Instead, the Remote- and Sub-Mapping modules could be tightly coupled. As the Sub-Mapping adds an ID and version number to each submap, this information could be leveraged to request only specific maps on the remote side. As a result, it would be possible to keep track of the map's current state and request missing maps to generate the exact same map on both sides.

The proposed mapping framework is not only relevant in the context of mobile robot navigation. Instead, it could also be used for tasks such as motion planning for robotic arms. Therefore, it could be integrated into different planning frameworks such as MoveIt [9].

Path Planning:

The main drawback of the Sub-Mapping module is the necessity to merge the entire network of sub-maps back into one continuous map of the environment, as it is required for path planning. To prevent this issue, the problem could be tackled from a different perspective. Instead of creating a merged map, the planner could be adjusted to work on the network of locally consistent sub-maps.

Even though the sampling-based planner was initially only intended as a comparable baseline during the evaluation, it performed exceedingly well in multiple scenarios. Therefore, the next step would be to adjust the concepts presented in this work further to work better on this class of problems. One significant problem would be, for example, traversing unknown areas using the sampling-based approach.

The last possible future research opportunity focuses on the local planner. Even though its reactive behavior provides multiple advantages due to its fast processing, it still comes with some drawbacks and is prone to oscillating between local minima. Therefore, it could be explored how well trajectory-based local planners, such as a Time Elastic Band planner, work directly on the VDB-Mapping representation.

List of Figures

List of Tables

Bibliography

[1] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45, 1968.

[2] John Bares, Martial Hebert, Takeo Kanade, Eric Krotkov, Tom Mitchell, Reid Simmons, and William Whittaker. Ambler: An autonomous rover for planetary exploration. *Computer*, 22(6):18–26, 1989.

[3] Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, pages 107–141, 1970.

[4] Amit Bhatia and Emilio Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 142–156. Springer, 2004.

[5] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 3, pages 2743–2748. IEEE, 2003.

[6] Jack E Bresenham. Algorithm for computer control of a digital plotter. In *Seminal graphics: pioneering efforts that shaped the field*, pages 1–6. 1965.

[7] Tony F Chan, Gene H Golub, and Randall J LeVeque. Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician*, 37(3):242–247, 1983.

[8] Chunxi Cheng, Qixin Sha, Bo He, and Guangliang Li. Path planning and obstacle avoidance for auv: A review. *Ocean Engineering*, 235:109355, 2021.

[9] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[10] Thomas Collins and JJ Collins. Occupancy grid mapping: An empirical evaluation. In *2007 mediterranean conference on control & automation*, pages 1–6. IEEE, 2007.

[11] Juan Cortés, Léonard Jaillet, and Thierry Siméon. Molecular disassembly with rrt-like algorithms. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3301–3306. IEEE, 2007.

[12] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[13] Daniele De Gregorio and Luigi Di Stefano. Skimap: An efficient mapping framework for robot navigation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2569–2576. IEEE, 2017.

[14] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, volume 2, pages 1322–1328. IEEE, 1999.

[15] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006.

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

[17] Daniel Duberg and Patric Jensfelt. Ufomap: An efficient probabilistic 3d mapping framework that embraces the unknown. *IEEE Robotics and Automation Letters*, 5(4):6411–6418, 2020.

[18] František Duchoň, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico, and Ladislav Jurišica. Path planning with modified a star algorithm for a mobile robot. *Procedia engineering*, 96:59–69, 2014.

[19] Gian Erni, Jonas Frey, Takahiro Miki, Matias Mattamala, and Marco Hutter. Mem: Multi-modal elevation mapping for robotics and learning. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11011–11018. IEEE, 2023.

[20] European Space Agency. Esa space resources challenge. `https://www.spaceresourceschallenge.esa.int/`, 2024. Accessed: 2024-09-17.

[21] Péter Fankhauser, Michael Bloesch, Christian Gehring, Marco Hutter, and Roland Siegwart. Robot-centric elevation mapping with uncertainty estimates. In *Mobile Service Robotics*, pages 433–440. World Scientific, 2014.

[22] Dave Ferguson and Maxim Likhachev. Efficiently using cost maps for planning complex maneuvers. *Lab Papers (GRASP)*, 5, 2008.

[23] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. A guide to heuristic-based path planning. In *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, pages 9–18, 2005.

[24] Dave Ferguson and Anthony Stentz. Anytime rrts. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5369–5375. IEEE, 2006.

[25] Dave Ferguson and Anthony Stentz. Field d*: An interpolation-based path planner and replanner. In *Robotics Research: Results of the 12th International Symposium ISRR*, pages 239–253. Springer, 2007.

[26] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of guidance, control, and dynamics*, 25(1):116–129, 2002.

[27] Udo Frese, Per Larsson, and Tom Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Transactions on Robotics*, 21(2):196–207, 2005.

[28] Sarah F Frisken, Ronald N Perry, Alyn P Rockwood, and Thouis R Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254, 2000.

[29] Nils Funk, Juan Tarrio, Sotiris Papatheodorou, Marija Popović, Pablo F Alcantarilla, and Stefan Leutenegger. Multi-resolution 3d mapping with explicit free space representation for fast and accurate mobile robot motion planning. *IEEE Robotics and Automation Letters*, 6(2):3553–3560, 2021.

[30] Brian P Gerkey and Motilal Agrawal. Break on through: Tunnel-based exploration to learn about outdoor terrain. In *ICRA Workshop on Path Planning on Costmaps*, 2008.

[31] Sarah FF Gibson. Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 23–30, 1998.

[32] TAN Guan-Zheng, HE Huan, and Aaron Sloman. Ant colony system algorithm for real-time globally optimal path planning of mobile robots. *Acta automatica sinica*, 33(3):279–285, 2007.

[33] J-S Gutmann and Kurt Konolige. Incremental mapping of large cyclic environments. In *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA'99 (Cat. No. 99EX375)*, pages 318–325. IEEE, 1999.

[34] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[35] G Heppner, A Roennau, J Oberländer, S Klemm, and R Dillmann. Lauropesix legged walking robot for planetary exploration participating in the spacebot cup. *WS on Advanced Space Technologies for Robotics and Automation*, 2(13):69–76, 2015.

[36] M. Herbert, C. Caillas, E. Krotkov, I.S. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 997–1002. IEEE Comput. Soc. Press, 1989.

[37] Andreas Hermann, Florian Drews, Joerg Bauer, Sebastian Klemm, Arne Roennau, and Ruediger Dillmann. Unified gpu voxel collision detection for mobile manipulation planning. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4154–4160. IEEE, 2014.

[38] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[39] Georg Rainer Hofmann. Who invented ray tracing? a historical remark. *The Visual Computer*, 6(3):120–124, 1990.

[40] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34:189–206, 2013.

[41] Andrew Howard, Maja J Mataric, and Gaurav Sukhatme. Relaxation on a mesh: a formalism for generalized localization. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, volume 2, pages 1055–1060. IEEE, 2001.

[42] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

[43] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Proceedings of international conference on robotics and automation*, volume 3, pages 2719–2726. IEEE, 1997.

[44] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 38–44. IEEE, 2016.

[45] Velodyne Lidar Inc. Velodyne lidar system. `https://velodynelidar.com/`, 2024. Lidar sensor.

[46] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. isam: Fast incremental smoothing and mapping with efficient data association. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 1670–1677. IEEE, 2007.

[47] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[48] Lydia E Kavraki, Mihail N Kolountzakis, and J-C Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and automation*, 14(1):166–171, 1998.

[49] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[50] Rachel Kirby, Reid Simmons, and Jodi Forlizzi. Companion: A constraint-optimizing method for person-acceptable navigation. In *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 607–612. IEEE, 2009.

[51] Sebastian Klemm, Jan Oberländer, Andreas Hermann, Arne Roennau, Thomas Schamm, J Marius Zollner, and Rüdiger Dillmann. Rrt*-connect: Faster, asymptotically optimal motion planning. In *2015 IEEE international conference on robotics and biomimetics (ROBIO)*, pages 1670–1677. IEEE, 2015.

[52] Matthew Klingensmith, Ivan Dryanovski, Siddhartha S Srinivasa, and Jizhong Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: science and systems*, volume 4, 2015.

[53] Sven Koenig and Maxim Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings 2002 IEEE international conference on robotics and automation (Cat. No. 02CH37292)*, volume 1, pages 968–975. IEEE, 2002.

[54] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning a*. *Artificial Intelligence*, 155(1-2):93–146, 2004.

[55] Sven Koenig, Craig Tovey, and Yuri Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*, 147(1-2):253–279, 2003.

[56] Kurt Konolige. Improved occupancy grids for map building. *Autonomous Robots*, 4:351–367, 1997.

[57] Gerhard K Kraetzschmar, Guillem Pages Gassull, and Klaus Uhl. Probabilistic quadtrees for variable-resolution mapping of large environments. *IFAC Proceedings Volumes*, 37(8):675–680, 2004.

[58] Eugene F Krause. Taxicab geometry. *The Mathematics Teacher*, 66(8):695–706, 1973.

[59] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[60] Boris Lau, Christoph Sprunk, and Wolfram Burgard. Improved updating of euclidean distance maps and voronoi diagrams. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 281–286. IEEE, 2010.

[61] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[62] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[63] Charles E Leiserson, Harald Prokop, and Keith H Randall. Using de bruijn sequences to index a 1 in a computer word. *Available on the Internet from http://supertech. csail. mit. edu/papers. html*, 3(5), 1998.

[64] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IROS*, volume 3, pages 1442–1447, 1991.

[65] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[66] Lixing Liu, Xu Wang, Xin Yang, Hongjie Liu, Jianping Li, and Pengfei Wang. Path planning techniques for mobile robots: Review and prospect. *Expert Systems with Applications*, 227:120254, 2023.

[67] Iker Lluvia, Elena Lazkano, and Ander Ansuategi. Active mapping and robot exploration: A survey. *Sensors*, 21(7):2445, 2021.

[68] Anbalagan Loganathan and Nur Syazreen Ahmad. A systematic review on recent advances in autonomous mobile robot navigation. *Engineering Science and Technology, an International Journal*, 40:101343, 2023.

[69] Clint D Lombard and Corné E Van Daalen. Stochastic triangular mesh mapping: A terrain mapping technique for autonomous mobile robots. *Robotics and Autonomous Systems*, 127:103449, 2020. Publisher: Elsevier.

[70] David V Lu, Daniel B Allan, and William D Smart. Tuning cost functions for social navigation. In *Social Robotics: 5th International Conference, ICSR 2013, Bristol, UK, October 27-29, 2013, Proceedings 5*, pages 442–451. Springer, 2013.

[71] David V Lu, Dave Hershberger, and William D Smart. Layered costmaps for context-sensitive navigation. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 709–715. IEEE, 2014.

[72] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4:333–349, 1997.

[73] Steve Macenski, David Tsai, and Max Feinberg. Spatio-temporal voxel layer: A view on robot perception for the dynamic world. *International Journal of Advanced Robotic Systems*, 17(2):1729881420910530, 2020.

[74] Andrei Andreevich Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta Imeni VA Steklova*, 42:3–375, 1954.

[75] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[76] Takahiro Miki, Lorenz Wellhausen, Ruben Grandia, Fabian Jenelten, Timon Homberger, and Marco Hutter. Elevation mapping for locomotion and navigation using gpu. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2273–2280. IEEE, 2022.

[77] Brett Miller, Ken Museth, Devon Penney, Nafees Bin, Zafar, and DreamWorks Animation. Cloud modeling and rendering for "puss in boots". 2012.

[78] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598:593–598, 2002.

[79] Hans Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 116–121. IEEE, 1985.

[80] Kevin Murphy and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Sequential Monte Carlo methods in practice*, pages 499–515. Springer, 2001.

[81] Ken Museth. An efficient level set toolkit for visual effects. In *SIGGRAPH 2009: Talks*, pages 1–1. 2009.

[82] Ken Museth. Db+ grid: A novel dynamic blocked grid for sparse high-resolution volumes and level sets. In *ACM SIGGRAPH 2011 Talks*, pages 1–1. 2011.

[83] Ken Museth. Vdb: High-resolution sparse volumes with dynamic topology. *ACM transactions on graphics (TOG)*, 32(3):1–22, 2013.

[84] Ken Museth. Hierarchical digital differential analyzer for efficient ray-marching in openvdb. In *ACM SIGGRAPH 2014 Talks*, pages 1–1. 2014.

[85] Ken Museth. Nanovdb: A gpu-friendly and portable vdb data structure for real-time rendering and simulation. In *ACM SIGGRAPH 2021 Talks*, pages 1–2. 2021.

[86] Ken Museth and Michael Clive. Cracktastic: fast 3d fragmentation in" the mummy: Tomb of the dragon emperor". In *ACM SIGGRAPH 2008 talks*, pages 1–1. 2008.

[87] Ken Museth, Michael Clive, and Nafees Bin Zafar. Blobtacular: surfacing particle system in" pirates of the caribbean 3". *SIGGRAPH sketches*, 10(1278780.1278804), 2007.

[88] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. In *AAAI*, volume 7, pages 1177–1183, 2007.

[89] Alex Nash and Sven Koenig. Any-angle path planning. *AI Magazine*, 34(4):85–107, 2013.

[90] Alex Nash, Sven Koenig, and Maxim Likhachev. Incremental phi*: Incremental any-angle path planning on grids. In *IJCAI*, pages 1824–1830, 2009.

[91] Alex Nash, Sven Koenig, and Craig Tovey. Lazy theta*: Any-angle path planning and path length analysis in 3d. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 147–154, 2010.

[92] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*, pages 127–136. Ieee, 2011.

[93] Andreas Nüchter and Joachim Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008.

[94] Helen Oleynikova, Alexander Millane, Zachary Taylor, Enric Galceran, Juan Nieto, and Roland Siegwart. Signed distance fields: A natural representation for both mapping and planning. In *RSS 2016 workshop: geometry and beyond-representations, physics, and scene understanding for robotics*. University of Michigan, 2016.

[95] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373. IEEE, 2017.

[96] Stanley Osher, Ronald Fedkiw, and K Piechor. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.*, 57(3):B15–B15, 2004.

[97] Madhumita Panda and Abinash Mishra. A survey of shortest-path algorithms. *International Journal of Applied Engineering Research*, 13(9):6817–6820, 2018.

[98] Erion Plaku, Kostas E Bekris, Brian Y Chen, Andrew M Ladd, and Lydia E Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005.

[99] Hongwei Qin, Shiliang Shao, Ting Wang, Xiaotian Yu, Yi Jiang, and Zonghan Cao. Review of autonomous path planning algorithms for mobile robots. *Drones*, 7(3):211, 2023.

[100] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[101] Victor Reijgwart, Alexander Millane, Helen Oleynikova, Roland Siegwart, Cesar Cadena, and Juan Nieto. Voxgraph: Globally consistent, volumetric mapping using signed distance function submaps. *IEEE Robotics and Automation Letters*, 5(1):227–234, 2019.

[102] Arne Roennau, Georg Heppner, Michal Nowicki, and Rüdiger Dillmann. Lauron v: A versatile six-legged walking robot with advanced maneuverability. In *2014 IEEE/ASME international conference on advanced intelligent Mechatronics*, pages 82–87. IEEE, 2014.

[103] Marc HJ Romanycia and Francis Jeffry Pelletier. What is a heuristic? *Computational intelligence*, 1(1):47–58, 1985.

[104] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[105] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE, 2011.

[106] Jari Saarinen, Henrik Andreasson, Todor Stoyanov, Juha Ala-Luhtala, and Achim J Lilienthal. Normal distributions transform occupancy maps: Application to large-scale online 3d mapping. In *2013 IEEE international conference on robotics and automation*, pages 2233–2238. IEEE, 2013.

[107] Jose Ricardo Sanchez-Ibanez, Carlos J Pérez-del Pulgar, and Alfonso García-Cerezo. Path planning for autonomous mobile robots: A review. *Sensors*, 21(23):7898, 2021.

[108] Jean Serra. *Image analysis and mathematical morphology*. Academic Press, Inc., 1983.

[109] Jean Serra. Mathematical morphology for complete lattices. *Image analysis and mathematical morphology*, 2:13–35, 1988.

[110] Emrah Akin Sisbot, Luis F Marin-Urias, Rachid Alami, and Thierry Simeon. A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 23(5):874–883, 2007.

[111] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Machine intelligence and pattern recognition*, volume 5, pages 435–461. Elsevier, 1988.

[112] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986.

[113] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Volumetric 3d mapping in real-time on a cpu. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 2021–2028. IEEE, 2014.

[114] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE international conference on robotics and automation*, pages 3310–3317. IEEE, 1994.

[115] Anthony Stentz et al. The focussed d^* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[116] Todor Stoyanov, Martin Magnusson, and Achim J Lilienthal. Comparative evaluation of the consistency of three-dimensional spatial representations used in autonomous robot navigation. *Journal of Field Robotics*, 30(2):216–236, 2013.

[117] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[118] Eijiro Takeuchi and Takashi Tsubouchi. A 3-d scan matching using improved 3-d normal distributions transform for mobile robotic mapping. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3068–3073. IEEE, 2006.

[119] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial intelligence*, 99(1):21–71, 1998.

[120] Sebastian Thrun. Learning occupancy grids with forward models. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, volume 3, pages 1676–1681. IEEE, 2001.

[121] Sebastian Thrun et al. Robotic mapping: A survey. 2002.

[122] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 IEEE/RSJ international conference on intelligent robots and systems*, pages 2276–2282. IEEE, 2006.

[123] Chris Urmson and Reid Simmons. Approaches for heuristically biasing rrt growth. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pages 1178–1183. IEEE, 2003.

[124] Ignacio Vizzo, Tiziano Guadagnino, Jens Behley, and Cyrill Stachniss. Vdbfusion: Flexible and efficient tsdf integration of range sensor data. *Sensors*, 22(3):1296, 2022.

[125] Georges Voronoi. Nouvelles applications des parametres continusa la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *J. reine angew. Math*, 133(97-102):9, 1908.

[126] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 adaptive systems for signal processing, communications, and control symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[127] Lorenz Wellhausen and Marco Hutter. Rough terrain navigation for legged robots using reachability planning and template learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6914–6921. IEEE, 2021.

[128] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous: Spatially extended kinectfusion. 2012.

[129] Patrick Wolf, Thorsten Ropertz, Moritz Oswald, and Karsten Berns. Local behavior-based navigation in rough off-road scenarios based on vehicle kinematics. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 719–724. IEEE, 2018.

[130] Fei Yan, Yi-Sha Liu, and Ji-Zhong Xiao. Path planning in complex 3d environments using a probabilistic roadmap method. *International Journal of Automation and computing*, 10:525–533, 2013.

[131] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, 2016(1):7426913, 2016.

[132] Manuel Yguel and Olivier Aycard. 3d mapping of outdoor environment using clustering techniques. In *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pages 403–408. IEEE, 2011.

[133] M Zucker, J Kuffner, and M Branicky. Multiple rrts for rapid replanning in dynamic environments. In *IEEE Conference on Robotics and Automation*, 2007.

[134] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International journal of robotics research*, 32(9-10):1164–1193, 2013.

Publications by the author et al.

[Forouher et al., 2016] Forouher, D., Grosse Besselmann, M., and Maehle, E. (2016). Sensor fusion of depth camera and ultrasound data for obstacle detection and robot navigation. In *2016 14th international conference on control, automation, robotics and vision (ICARCV)*, pages 1–6. IEEE.

[Grosse Besselmann et al., 2024] Grosse Besselmann, M., Häuselmann, R., Mauch, S., Puck, L., Schnell, T., Rönnau, A., and Dillmann, R. (2024). 3D Global Path Planning for Mobile Ground Robots on Volumetric Maps. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5430–5437.

[Grosse Besselmann et al., 2021] Grosse Besselmann, M., Puck, L., Steffen, L., Roennau, A., and Dillmann, R. (2021). VDB-Mapping: A high resolution and real-time capable 3D mapping framework for versatile mobile robots. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 448–454. IEEE.

[Grosse Besselmann et al., 2023a] Grosse Besselmann, M., Roennau, A., and Dillmann, R. (2023a). VDB-Submapping: Efficient Handling of Retroactive Pose Changes During Large-Scale Volumetric Mapping. In *2023 20th International Conference on Ubiquitous Robots (UR)*, pages 327–332. IEEE.

[Grosse Besselmann et al., 2022] Grosse Besselmann, M., Rönnau, A., and Dillmann, R. (2022). Remote VDB-Mapping: A Level-Based Data Reduction Framework for Distributed Mapping. In *Climbing and Walking Robots Conference*, pages 448–459. Springer.

[Grosse Besselmann et al., 2023b] Grosse Besselmann, M., Rönnau, A., and Dillmann, R. (2023b). NavPi: An Adaptive Local Path-Planning Pipeline for 3D Navigation in Difficult Terrain. In *Climbing and Walking Robots Conference*, pages 224–235. Springer.

[Puck et al., 2023] Puck, L., Mauch, S., Grosse Besselmann, M., Schnell, T., Buettner, T., Roennau, A., and Dillmann, R. (2023). RISK VDBMAPPING – ENRICHED VOLUMETRIC INFORMATION FOR RISK-AWARE MISSIONS. In *2023 17th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*. ESA.

[Schnell et al., 2023] Schnell, T., Oberacker, D., Exner, F., Puck, L., Grosse Besselmann, M., Spielbauer, N., Plasberg, C., Roennau, A., and Dillmann, R. (2023).

An Efficient Scalable Autonomy Approach for Teams of Heterogeneous Mobile Robots. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–7. IEEE.

Related student work

[Hä22] Ramona Häuselmann. Path planning for mobile ground robots in a dynamic 3D environment. Bachelorarbeit, KIT Karlsruher Institut für Technologie, Karlsruhe, Germany, 2022.

[Mau23] Samuel Mauch. Risk-aware path planning for planetary surfaces. Bachelorarbeit, KIT Karlsruher Institut für Technologie, Karlsruhe, Germany, 2023.