TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG

The University of Resources. Since 1765.

Sampling-Based Exploration Strategies for Mobile Robot Autonomy

Von der Fakultät für Mathematik und Informatik
der Technischen Universität Bergakademie Freiberg

genehmigte

Dissertation

zur Erlangung des akademischen Grades

Doktor-Ingenieur
(Dr.-Ing.)

vorgelegt von **M.Sc. Marco Steinbrink**

geboren am 16. Januar 1993 in Wuppertal

Gutachter: Prof. Dr.-Ing. Bernhard Jung
Prof. Dr. rer. nat. Stefan May

Tag der Verleihung: Freiberg, den 26. Juli 2023

Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistungen von folgenden Personen erhalten:
Bernhard Jung und Stefan May, die diese Dissertation betreut haben, gaben Hinweise zur Überarbeitung der Arbeit. Philipp Koch, der im UNDROMEDA Projekt beteiligt war, gab Anmerkungen zum Inhalt, die in das Manuskript eingeflossen sind. Weitere Personen waren an der Abfassung der vorliegenden Arbeit nicht beteiligt.

Die Hilfe eines Promotionsberaters habe ich nicht in Anspruch genommen. Weitere Personen haben von mir keine geldwerten Leistungen für Arbeiten erhalten, die nicht als solche kenntlich gemacht worden sind. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

26. Juli 2023 M.Sc. Marco Steinbrink

Acknowledgement

I would like to thank my advisers Bernhard Jung and Stefan May for their support and advice throughout this work. They provided extraordinary insights and experiences to guide my research as well as refine it and helped me to not lose my focus throughout the years.

Furthermore, I am grateful for the opportunity to join the AutonOHM team in various robotic competitions. The team members, especially Philipp Koch, shared valuable experience in robotics with me which helped me in this work.

I am thankful for my partner Dakota who supported me through this work and kept me motivated to go on. My parents and grandmother also were there for me, for which I am grateful. Last but not least, I want to thank my friends who always encouraged me.

Contents

Previous Publications

[1] Marco Steinbrink, Philipp Koch, Stefan May, Bernhard Jung, and Michael Schmidpeter. "State Machine for Arbitrary Robots for Exploration and Inspection Tasks". In: *Proceedings of the 2020 4th International Conference on Vision, Image and Signal Processing*. New York, NY, USA: ACM, Dec. 2020, pp. 1–6. DOI: 10.1145/3448823.3448857.

[2] Marco Steinbrink, Philipp Koch, Bernhard Jung, and Stefan May. "Rapidly-Exploring Random Graph Next-Best View Exploration for Ground Vehicles". In: *2021 European Conference on Mobile Robots (ECMR)*. IEEE, Aug. 2021, pp. 1–7. DOI: 10.1109/ECMR50962.2021.9568785.

[3] Philipp Koch, Marco Steinbrink, Stefan May, and Andreas Nuechter. "Traversability Analysis for Wheeled Robots using Point-Region-Quad-Tree based Elevation Maps". In: *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, Apr. 2022, pp. 192–197. DOI: 10.1109/icarsc55462.2022.9784803.

This work focuses on the development of an autonomous exploration approach for Unmanned Ground Vehicles which utilizes the robot autonomy state machine proposed in the author's first article [1] that is described in Chapter 4. The proposed exploration is published in article [2] which is detailed in Chapters 5 and 6. In Chapter 8, the traversability analysis from [3] is used in combination with this work's proposed exploration approach.

Nomenclature

Symbol	Description
state	State of an object
`ClassName`	Class introduced in this work
PROCEDURENAME	Function introduced in this work
c	Scalar
S	Set
$\mathcal{T}$	Tuple
$F(x)$	Function
$\mathbf{p}$	Point $(x_p, y_p, z_p)^T \in \mathbb{R}^3$
$\mathbf{p}^m$	Grid map point $\left(x_p^m, y_p^m\right)^T \in \mathbb{N}_0^2$
$\mathbf{X}$	Matrix in $\mathbb{R}^3$
$\lfloor c \rfloor$	Round down a scalar $c \in \mathbb{R}$ to the nearest value $z \in \mathbb{Z},\ z \leq c$
$\lceil c \rceil$	Round up a scalar $c \in \mathbb{R}$ to the nearest value $z \in \mathbb{Z},\ z \geq c$
$\lfloor c \rceil$	Round a scalar $c \in \mathbb{R}$ to the nearest value $z \in \mathbb{Z}$
φ	Discrete orientation in degrees with $0 \leq \varphi < 360,\ \varphi \in \mathbb{N}_0$, 0 is aligned with the x-axis and it is measured counter-clockwise

For calculations in degrees, absolute differences are defined as stated in Equation (0.1) which always returns the shorter difference between φ_1 and φ_2 around the circle. Furthermore, the opposite orientation $\overline{\varphi}$ of an orientation φ is defined in Equation (0.2).

$$|\varphi_1 - \varphi_2| = 180 - ||\varphi_1 - \varphi_2| - 180| \tag{0.1}$$

$$\overline{\varphi} = \begin{cases} \varphi - 180, & \varphi \geq 180 \\ \varphi + 180, & \varphi < 180 \end{cases} \tag{0.2}$$

List of Abbreviations

AEDE Autonomous Exploration Development Environment

AEP Autonomous Exploration Planner

DARPA Defense Advanced Research Projects Agency

DoF Degrees of Freedom

DSVP Dual-Stage Viewpoint Planner

DWA Dynamic Window Approach

EIT European Institute of Technology

EKF Extended Kalman Filter

ESDF Euclidean Signed Distance Fields

FE Frontier Exploration

FoV Field of View

GBPlanner Graph-Based exploration path Planner

GPS Global Positioning System

GUI Graphical User Interface

IMU Inertial Measurement Unit

NAI Node Area Inflation

NBV Next-Best View

NN Neural Network

PRM Probabilistic Roadmaps

RGB-D Red Green Blue - Depth

RH-NBVP Receding Horizon Next-Best-View Planner

RNE Random-Sampling-Based Next-Best View Exploration

ROS Robot Operating System

RRG Rapidly-exploring Random Graph

RRL Rescue Robot League

RRT Rapidly-exploring Random Tree

RSM Robot Statemachine

SD Standard Deviation

SLAM Simultaneous Localization and Mapping

sPRM simplified Probabilistic Roadmaps

SRC Sparse Ray Casting

SRP Sparse Ray Polling

SubT Subterranean

TARE Technologies for Autonomous Robot Exploration

TSDF Truncated Signed Distance Fields

TSP Travelling Salesman Problem

UAV Unmanned Aerial Vehicle

UGV Unmanned Ground Vehicle

UML Unified Modeling Language

UNDROMEDA Underground Robotic System for Monitoring, Evaluation and Detection Applications

URDF Unified Robot Description Format

List of Figures

List of Tables

List of Algorithms

1. Introduction


Fig. 1.1.: The robotic platform developed in the UNDROMEDA project operating in an underground mine using its sensor payload to create a detailed map of the hazardous and Global Positioning System (GPS)-deprived surroundings.

Autonomous robotics are employed in ever more diverse fields and became everyday consumer appliances like e.g., lawnmower and vacuum robots. Industrial robots are utilized in production facilities and warehouses and take over repetitive and monotonous tasks. But in the fields of disaster response and the exploration of unknown environments, robots are still controlled by human operators. The formerly mentioned applications take place in mostly similar, human-constructed environments which allow for comparably simple approaches. For example, the navigation of lawnmower, vacuum and warehouse robots is solely based around two dimensions on predominantly flat surfaces. In comparison, the latter applications feature rough, hazardous, multi-level environments which cannot be navigated using a simple 2D approach.

More recently, Unmanned Aerial Vehicles (UAVs) gained popularity in disaster response, exploration and inspection research because they are not restricted by rough floors and stairways like Unmanned Ground Vehicles (UGVs). But in comparison, they have a very limited run time and are not able to hold the payloads a UGV can carry. At least there is a major trade-off between run time and payload which is not as drastic for UGVs.

Therefore, this work focuses on the research of an exploration approach for UGVs in GPS-deprived environments such as disaster sites and underground mines. It should enable a UGV to autonomously explore a previously unknown environment while identifying where it is able to move to and where it is not. The exploration's goal is to produce a 3D map of the observed environment that can be utilized by the operators to assess further actions without putting human lives in danger.

Even though the UGV is still bound to move on the ground which is simplified to 2D planning, it should observe the map volume in 3D and utilize it for its navigation and exploration planning. Therefore, this work proposes a 3D exploration approach that is more complex but also more versatile compared to simple 2D mapping and exploration which can solely be applied on predominantly flat surfaces.

Autonomous exploration's largest issue is the balance between speed and thoroughness. Because time can be critical in disaster scenarios or because the run time of the robot is a restriction regarding the size of the area to be explored, it is not efficient to rigorously explore every part of the area up close. A sufficient strategy must be found that explores large volumes of space fast enough but also allows a detailed mapping to make out the objects of interest.

Autonomy is required as communication is mostly limited to line-of-sight in underground environments because thick walls block all radio waves. Therefore, manual operation of a robot is only possible using a connecting cable or an external communication infrastructure which is not available in most environments.

The first section discusses the motivation for this work, followed by the utilized methodology and a listing of the scientific contributions of this work.

1.1. Motivation

This work is motivated by a research project called UNDROMEDA whose goal is the autonomous inspection and exploration of underground mines using a self-developed robotic platform and sensor payload.

Because the current state-of-the-art solutions in the field of 3D exploration research focus on using UAVs or are tailored for a specific robotic platform used in the particular research, a flexible approach has to be developed. It should be usable with a variety of different robots to explore large underground environments based on 3D map coverage.

Due to the project's robot being under development with an unknown size and sensor payload, an approach for exploration that is not bound to a specific robot, is required.

Furthermore, the approach should not be bound to a specific robotic platform or sensor setup to remain flexible and usable in versatile scenarios.

In Figure 1.2, the robot Schrödi can be seen on which the proposed exploration should also be deployed. It is a tracked robot constructed by the Nuremberg Institute of Technology and is used by its AutonOHM team in the RoboCup Rescue Robot League (RRL) Competition.

The possibility to release this approach as an open-source repository and make it available for the scientific and robotics community is also a motivation to keep it flexible.

In the following, details about the UNDROMEDA project and RoboCup RRL are described which are the major motivation for this work. Furthermore, the decision to focus on a sampling-based exploration is explained and the influence of the DARPA Subterranean (SubT) Challenge is shown.

1.1.1. UNDROMEDA

This work was financed by the European Institute of Technology (EIT) RawMaterials' Underground Robotic System for Monitoring, Evaluation and Detection Applications (UNDROMEDA) project. The project's goal was to develop a robotic platform that can execute exploration and inspection tasks in an underground environment. A cooperation of companies, universities and mine operators from the EU wanted to combine existing products and research to technology readiness level 7 as defined by the

Fig. 1.2.: Robot Schrödi from the AutonOHM team during the RoboCup 2019 in Sydney, Australia. It is using the proposed state machine for autonomous navigation in the left image and can be seen in an exploration challenge in the middle as well as manipulating pipes for a dexterity challenge on the right.

EU [4]. A ground-based vehicle was chosen for its longer range and run time compared to UAVs.

To enable a variety of different applications, a sensor platform was developed that is able to mount different sensors and can be attached to different robotic ground vehicles. This sensor platform is attached to a UGV and is deployed in underground mines to autonomously inspect appliances or explore and map certain areas of the mine. It can be seen in Figure 1.1. For this, a high-level control of the autonomous operation is required and a framework to efficiently explore free space is needed.

1.1.2. RoboCup Rescue Robot League

The RoboCup RRL's goal is to advance robotics research for disaster scenarios by holding annual international events where different teams compete in the following four categories:

- Mobility
- Dexterity
- Exploration
- Autonomy

The mobility category is about the robot's ability to traverse obstacle courses repeatedly and as fast as possible including balancing on bars, driving over curbs and climbing up stairs with added debris.

In dexterity, challenges feature the manipulation of small and hard-to-reach objects mimicking valves or opening doors. This requires a mobile manipulator with an extended range which limits the space for additional sensors on the robot.

The exploration category focuses on exploring courses while identifying as many artifacts as possible. Artifact locations must be correctly marked in a map of the explored environment. They feature e.g., visual markers, hazard signs and fire extinguishers.

Autonomy does not include challenges on its own but is judged throughout all other categories. Competitors gain bonus points for executing autonomous runs and a partial bonus for semi-autonomous behaviors like an inverse kinematic for a manipulator arm. Certain exploration tasks can only be participated in by using fully autonomous operation of the robot.

Each event is concluded with a final in which all categories are combined by a search and identification of specific artifacts in a large arena including the previous challenges' course areas [5].

1.1.3. Sampling-Based Exploration

A thorough analysis of existing exploration approaches which can be seen in Section 2.4, reveals three major research directions. These are frontier, sampling-based and information-theoretic exploration.

Frontier exploration's effort increases significantly the larger the map grows compared to sampling-based approaches. An information-theoretic exploration is generally even more computationally complex than frontier exploration. Therefore, a sampling-based approach is selected for the exploration in the UNDROMEDA project as the exploration of large underground mines is one of its goals.

A fourth direction is learning-based exploration which emerged recently and utilizes Neural Networks (NNs) to decide on the next exploration goal. But the proposed approaches are trained on specific environments and are therefore not suitable for arbitrary environments.

Recent work focuses heavily on exploration using UAVs [6, 7, 8, 9, 10] which motivated the development of a novel sampling-based exploration approach designed specifically for UGVs. It has an emphasis on the ability of the robot to traverse certain terrain for exploration. This ability is referenced as traversability.

1.1.4. DARPA Subterranean Challenge

The DARPA SubT Challenge was introduced in 2017 by the Defense Advanced Research Projects Agency (DARPA) to alleviate research regarding autonomous underground exploration with the goal to create robotic capabilities for disaster sites [11]. A virtual and a physical contest with 20 international teams were held which finished in October 2021 with a total prize money of 5 million dollars distributed between the top five teams in both categories.

Three preliminary rounds were held, completed by a grand finale in which each team had to find and correctly spatially reference as many artifacts in the environment in a limited amount of time, e.g., manikins, fire extinguishers and backpacks. A set of heterogeneous robots could be used for the exploration but the communication was restricted due to the topology of the underground environments.

A lot of research from the teams consisting of universities and industry partners was published during and after the SubT Challenge focusing on autonomous exploration, localization and communication [12, 13, 14, 15]. This research also served as an inspiration and comparison for the later iterations of the proposed exploration algorithm. All exploration approaches used in the SubT Challenge split exploration into a local and a global algorithm which motivates the introduction of this work's hybrid approach in Chapter 8.

1.2. Methodology

The methodology of the following work is an iterative process to develop and optimize a novel exploration approach called Random-Sampling-Based Next-Best View Exploration (RNE). The goal of creating a versatile exploration for UGVs is based on existing research. The development follows an agile approach by evaluating every iteration and comparing it with previous iterations and other state-of-the-art exploration algorithms to deduce content for the next iteration from the particular comparison.

Before starting to work on the exploration approach, a state machine called Robot Statemachine (RSM) has been developed that supervises the robot's autonomy and focuses on inspection and exploration. It introduces an interface for exploration algorithms that alleviates the use of and enhances a widely-used robot navigation implementation [16].

The inspiration for the approach presented in this work is the Receding Horizon Next-Best-View Planner (RH-NBVP) proposed by Bircher et al. in 2016 [6]. It introduced the use of a Rapidly-exploring Random Tree (RRT) to generate sample points in free space which are evaluated as potential goals for a UAV. RH-NBVP discards the tree of samples after choosing a goal based on distance and observable unknown space. Furthermore, the UAV waits until a new goal is calculated until it moves again.

Another inspiration is the Autonomous Exploration Planner (AEP) introduced by Selin et al. in 2019 [7] which itself is based on RH-NBVP and features an improved method to more efficiently determine the node gain. It persists previous sampling points to reuse them later which helps the algorithm to escape local maxima.

The rebuilding of the RRT and waiting for the calculation of a sufficient amount of sample points to finish before continuing the exploration shows potential for improvements.

Therefore, the first step is the development of an RRT-based exploration approach with a consistent tree and the calculation of node rewards in a separate thread. Furthermore, a traversability assessment for UGVs and a navigation planner following the tree's edges are introduced as well as further optimization of the node gain calculation.

Simulation experiments and evaluations show that the consistent structure, multi-thread calculation and improved node gain calculation lead to vastly decreased exploration duration and traveled path length while observing the same amount of space. But due to the tree structure that is followed by the navigation planner, detours can be introduced when following it to another branch.

To improve the navigation planning, the second iteration focuses on replacing the RRT with a Rapidly-exploring Random Graph (RRG). It was shown to lead to an asymptotically optimal path between two configurations when approaching infinite samples by Karaman and Frazzoli [17]. Also, based on the work of Xu et al. [8], new samples for the RRG are not only generated globally but additionally in the direct vicinity of the robot.

When comparing the second iteration to the first and its inspirational approaches in simulations, it shows a significant increase in efficiency. But for larger environments, the steadily growing graph leads to an increasing overhead in computation. Also, RNE sometimes steers the robot too close to obstacles which results in collisions and failed runs.

Therefore, the third iteration of RNE introduces the inflation of sample areas which

must be without obstacles to build a topology-based graph. Incorporating further measures, e.g., heading changes and traversability, into the node reward function to adapt the exploration to different circumstances is added as an enhancement.

These changes further decrease the duration of the exploration and the required processing time compared to previous iterations. But the rising number of nodes in the graph for large explorations remains an issue.

Inspired by multiple novel approaches [7, 9, 12, 14, 18, 19, 20], the fourth and last iteration separates the exploration into a local and a global part. The former is based on the third iteration of RNE that is only run in a restricted, moving area around the robot. Unexplored nodes that fall out of this area, are stored in the global planner. They are connected using a shortest route heuristic and visited later in the exploration. This is intended to reduce memory usage and computation time as the local graph is strictly limited in size.

Simulated comparisons show that the final iteration is able to compete with state-of-the-art exploration approaches and introduces a method applicable to a wide variety of UGVs and sensor setups. An experiment with a real robot further confirms the capability of the proposed algorithm.

1.3. Contributions

This work introduces the following scientific contributions to the state of the art in autonomous exploration development and research. These contributions are separated into methodical and application-oriented contributions.

Methodical Contributions

- An open-source state machine for exploration and inspection with an enhanced interface to a robot navigation implementation that allows to supervise and control the robot's autonomy[1].
- A sampling-based exploration algorithm designed for UGVs that incorporates a traversability measure and is openly available[2].
- The introduction of a sparse, decoupled node gain calculation to determine exploration goals asynchronously.
- A graph construction that follows the environment's topology based on inflated node areas and movement to increase the distance to obstacles.
- An extension to a hybrid exploration planning approach that combines a local graph with a global exploration in a novel way. It merges and stores unexplored nodes to be explored later. The shortest route to explore all of these nodes is calculated using a heuristic.

[1] `https://github.com/MarcoStb1993/robot_statemachine`
[2] `https://github.com/MarcoStb1993/rnexploration`

Application-oriented Contributions

- Evaluation of multiple additions to the exploration approach in simulations and their impact on the performance using different autonomous UGVs.
- Validation of the exploration approach in numerous simulations and experiments.

1.4. Outline

This work is structured into the following chapters which are briefly described:

Chapter 2 introduces related work that is relevant to the proposed approach. This includes state machines for mobile robots and robotics frameworks, sampling-based path planning, autonomous exploration and inspection as well as Next-Best View calculation.

Chapter 3 elaborates mathematical foundations and concepts required for the approach proposed in this work. Sampling-based algorithms are explained together with auxiliary methods, followed by Simultaneous Localization and Mapping. Furthermore, exploration and mapping backgrounds are described as well as the calculation of shortest routes.

Chapter 4 describes the design of a state machine to supervise and control the autonomous exploration implemented in the following chapters and also to enable inspection. Its interfaces and applications are detailed.

Chapter 5 proposes the foundation for this work's autonomous exploration approach based on a Rapidly-exploring Random Tree and describes implementation details including the computation efficient gain calculation, traversability assessment and navigation planner.

Chapter 6 shows the next iteration of the exploration approach which replaces the Rapidly-exploring Random Tree with a Rapidly-exploring Random Graph and introduces additional local sampling. A comparison to the previous iteration and earlier state-of-the-art approaches is conducted as well.

Chapter 7 introduces a topology-based graph construction for the exploration and a revised reward function that incorporates different metrics. These enhancements are evaluated in comprehensive simulations.

Chapter 8 presents the last iteration of the exploration approach which separates it into local and global exploration. The local exploration is based on the previous iteration while the global part stores unexplored nodes and proposes an efficient path to visit them using a shortest route heuristic. A comparison to the previous iteration and state-of-the-art algorithms is added.

Chapter 9 concludes this work and recapitulates its contributions. Furthermore, areas for future research are outlined based on this work's findings.

2. Related Work

This chapter presents related research to the introduced work. First, existing frameworks for robotics are listed and compared briefly before one of them is selected for the proposed approach and described in detail.

Based on this framework, state machines for mobile robotic exploration and inspection are evaluated. They are put into context with the state machine proposed in Chapter 4.

Then, sampling-based exploration strategies are introduced together with their manifold adaptions and enhancements. These are the foundation and inspiration for the strategies utilized in this work which are described in Chapters 5, 6 and 7.

Furthermore, different existing robotic exploration and inspection approaches for mobile robots are examined and assessed. Also, methods to estimate the Next-Best View (NBV) for exploration or inspection are discussed. This work's contribution is inspired by and compared to some of these exploration approaches in Chapters 6 and 8.

2.1. Robotics Frameworks

This work focuses on research and implementations using the open-source Robot Operating System (ROS) [21] as it has a vibrant and steadily growing community in research and industry [22]. Additionally, it already supports most of the hardware used in UNDROMEDA which is introduced in Section 1.1.1. ROS is a package-based framework and encourages community-developed packages which implement interfaces to many robots and sensors as well as algorithms for common robotics problems. Its processes are run in nodes that can be deployed on different machines in a network and can communicate with each other using messages and services. It features powerful tools for visualization and can be easily integrated with the Gazebo simulation environment [23].

Another robot framework is Yet Another Robot Platform which is more specialized in the communication between different robot hardware [24]. Furthermore, there is the Mobile Robot Programming Toolkit which bundles several algorithms commonly deployed in mobile robotics like Simultaneous Localization and Mapping (SLAM) and navigation planning [25]. However, both lack the extensive interfaces and algorithms as well as the active community of ROS.

Macenski et al. [26] introduced ROS2 which includes Micro-ROS for embedded systems, offers real-time enforcement and has functional safety ISO 26262 required for autonomous vehicles and heavy machinery. It features non-blocking services and introduces native actions for communication compared to ROS. ROS2 uses the Data Distribution Service for communication that utilizes UDP and different settings for the reliability and durability of messages. It also provides authentication, access control and encryption.

Even though ROS2 is the successor of ROS, during this work the number of available packages and community support was sub-par to its predecessor. Therefore, ROS2 is not used for the implementation. But the yearly metric reports suggest, that the support is going to increase for ROS2 in the near future [22, 27]. A non-trivial transfer of the proposed approach to ROS2 is expected to be useful.

2.1.1. ROS

As ROS is the robotics framework used throughout this work, further details about its file system, communication and community-developed packages are listed below.

ROS is organized in distributions which are supported for a specific Ubuntu distribution. In this work, development started with distribution Kinetic but switched to Melodic as Kinetic's end of life was reached in 2021. The newest and last release for ROS in 2020 was Noetic on which the developed state machine and exploration can be run as well.

File System

ROS resources are organized in packages which contain processes called nodes, libraries, message types, configuration files and other relevant data. Packages are the smallest build item in ROS.

They can be built using the ROS build tool catkin and can include dependencies for other packages and libraries. These dependencies and further metadata are defined in a package's manifest.

A metapackage bundles multiple packages to facilitate using and building them. They also serve as semantic groups.

Nodes and libraries can be written in Python or C++ and process data. This data is retrieved from sensors or can be obtained through messages. The messages are also defined in packages.

Communication

To exchange information, nodes can use two different forms of communication in ROS which are described in the following. Both forms require a central ROS master that is aware of all available communication channels and also stores parameters for the utilized nodes.

Topics which are the first form, can be used to asynchronously communicate between multiple nodes. A node can publish a topic of an existing message type using a unique topic name which is registered with the ROS master. If another node wants to subscribe to this topic, it accesses the publisher information from the ROS master and then requests the message from the publisher. This is shown in Figure 2.1.

The second form is a synchronous service. A service must be registered at the ROS master by the providing node and it is based on a predefined message type for the request and reply. If another node wants to use this service, it becomes the client by receiving the information from the master and sending a request to the provider. The provider processes the request and sends a response. The client's process is blocked until it receives a reply.

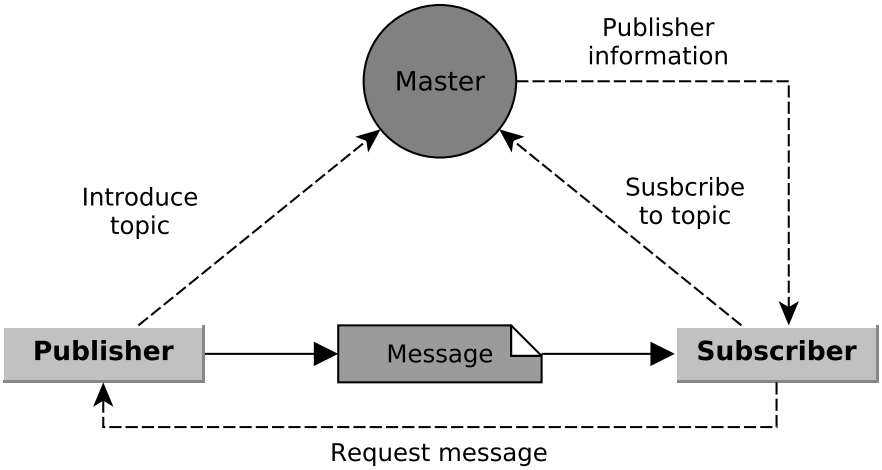


Fig. 2.1.: Communication in ROS using a topic between a publishing node and a subscribing node. The ROS master holds the information about all available topics in the system.

Nodes can be run on multiple connected machines using a central ROS master. ROS commonly uses the TCP/IP stack for communication.

Coordinate Convention

A standard coordinate convention is declared for ROS which states that all coordinate systems are right-handed. Figure 2.2 shows the ROS axis orientation in relation to a body.

ROS introduces three coordinate frames which are explained below:

base link This frame is rigidly attached to an arbitrary position and orientation on a robot.

odom The odom frame is a world-fixed frame that is continuous which means the pose of a robot does not jump in relation to it. But it is prone to drift over time as it should be derived from the robot's odometry.

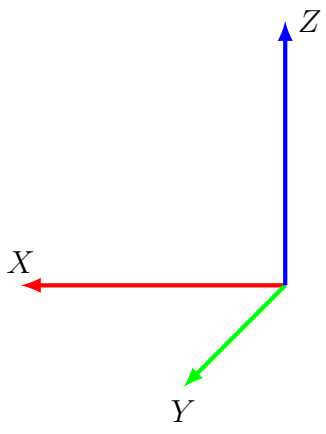


Fig. 2.2.: Right-handed ROS coordinate convention with the x-axis pointing forward from an object, y-axis pointing left and z-axis up.

map The map frame is a world-fixed frame that serves as a long-term reference for the robot's pose. Because it should be computed from a SLAM algorithm which is introduced in Section 3.2, it can show discrete jumps.

Community Packages

ROS is driven by a large community which creates and publishes packages that can be openly accessed and utilized for commercial and research purposes. These packages include e.g., sensor interfaces, localization, mapping and navigation algorithms. A selection of these packages used throughout the presented work is briefly described in the following:

tf2 The successor of the tf package is used for coordinate frame conversions and to find the relationship between different frames, e.g., a robot's main frame and a sensor's frame [28].

gmapping This package is utilized for SLAM which is explained in Section 3.2.2. It localizes the robot and builds a 2D map based on 2D lidar scans [29, 30].

navigation The ROS navigation stack includes several packages used for path planning and following for robots. Multiple global and local planners are available which use a cost map with inflated obstacles to calculate a safe path [16]. This work implements its own global planner but uses the Dynamic Window Approach (DWA) local planner [31].

octomap The OctoMap package is a mapping tool to create memory-efficient maps which utilize probabilities when integrating new measurements. It is described in detail in Section 3.3.3 and the gain calculation introduced in this work's exploration approach uses it [32].

rviz RViz is a Graphical User Interface (GUI) for ROS that allows to visualize the robot, sensor measurements, constructed maps and much more. It allows to include plugins that show additional information. A plugin for RViz to utilize the proposed state machine is developed in Section 4.1.3.

rqt This package is a GUI framework that can show arbitrary plugins based on the application framework Qt[1]. Multiple plugins can be added to a single window which enables creating a custom interface for the desired application. An rqt plugin for the introduced state machine is detailed in Section 4.1.3.

2.2. State Machines for Mobile Robots

A state machine is one of the 23 Unified Modeling Language (UML) creational patterns introduced by Gamma [33] and is based on statecharts [34]. The UML state machine is a directed graph in which nodes are states and edges symbolize state transitions. Its concept is to model an entity's state and specify under which circumstances the state changes. There are two kinds of state machines defined in [35], behavior and protocol state machines. In this work, the term state machine only refers to behavioral

[1] `https://www.qt.io/`

state machines. They are used to model a system's behavior while in a certain state or during a specific state transition.

To operate a robot using ROS, a multitude of ROS packages which offer different functionalities, must be initialized using specifically designed files or commands in most cases. Therefore, a change of robot behavior requires a skilled operator who often has to know the programming as well. There currently is a selection of state machines available in ROS which is presented in the following.

One of them is SMACH which was developed by Boren and Cousins [36] and offers a finite state machine which can be realized with Python code and supervised through a provided GUI. It is designated for executing complex high-level tasks and allows nested or concurrent states as well as state preemption. Data can be attached to states and is available to their children.

Several additions to SMACH were published which enable pre-configured, repeatable behaviors [37], decouple design from programming by providing a description language for the respective states [38] or introduce core templates and automatic code block generation [39]. SMACH and its additions allow to implement complicated behaviors whereas the state machine introduced in Chapter 4 is more focused on usability instead of complexity. Therefore, it is not as adaptable as SMACH but can be deployed with less effort for mobile robots.

FlexBE is also based on SMACH and provides flexibility by enabling changes on the fly by the operator. It introduces behaviors for pre-specified tasks which are also adjustable [40]. In addition, FlexBE furthermore integrates its own plugins for navigation, enabling more interaction with the underlying classes of the ROS navigation stack [41]. But FlexBE still requires an operator with programming experience.

The package ros_control offers a framework to control all robot joints and the robot's movement [42], therefore it is targeted at a lower level.

A more recently introduced package is SMACC which was inspired by SMACH and implemented in C++. It features readily available plugins to e.g., ROS navigation, combined with many of SMACH's advantages [43]. Unfortunately, the state machine developed in this work was finished before SMACC was published. Otherwise, it could have been used for the same tasks without the expensive implementation of a self-developed solution.

Cao et al. [44] proposed a mid-level implementation for facilitating mobile robotics research called Autonomous Exploration Development Environment (AEDE). Its integrated simulation provides interfaces to high-level planners by offering a local planner and terrain traversability analysis. This allows setting navigation goals for the robot. It is less of a state machine but a tool to develop and test exploration algorithms with. Two approaches presented in Section 2.4.2 utilize it and the final tests of the exploration algorithm proposed in this work partially make use of it which can be seen in Section 8.5.

Closely related to state machines are behavior trees which became popular due to the computer game industry utilizing them for controlling game reactions to the player.

In [45], the authors develop and utilize a behavior tree for ROS to enable tool grasping and usage with a robotic arm and gripper. They state that their behavior tree is advantageous compared to state machines as behaviors can be easily reused without the need to specifically define transitions to each particular state. They implemented sequential execution of behaviors as well as parallel execution.

Colledanchise and Ögren analyzed behavior trees and introduced a functional description to provide measures of their robustness and safety. They also describe the superior modularity of behavior trees compared to state machines and show several examples of their usage [46, 47]. Behavior trees are a better choice for dynamic environments and a wide array of behaviors to cope with different situations. But the overhead they introduce is not required for the intended application which favors utilizing a state machine.

Even though the above-mentioned state machines and behavior trees provide adaptable and extensive frameworks, the state machine proposed in this work is intended to provide a less complex solution. It is specifically tailored for repetitive inspection and autonomous exploration with mobile robots while focusing on the usability for operators. With this work's research on sampling-based exploration and UNDROMEDA's use cases in mind, a more appropriate state machine to enable the previously mentioned tasks is designed and implemented.

2.3. Sampling-Based Path Planning

The works presenting the sampling-based path planning methods PRM, RRT and RRG are described in the following.

PRM

RRT and RRG, which are described below, are inspired by Probabilistic Roadmaps (PRM) [48]. PRM consists of two phases to find a path between a start and a goal configuration, the learning phase and the query phase. The learning phase constructs a forest, which is a set of trees, by randomly sampling points in the configuration space. It tries to connect them to a node from neighboring trees in the forest in a certain distance using a fast local planner. The parameters set for the learning phase determine the density and coverage of the constructed tree which influence the path created by the query phase. Furthermore, a heuristic is employed to increase the sampling density near obstacle regions.

Kavraki et al. [49] later introduced simplified Probabilistic Roadmaps (sPRM) which remove the heuristic and build a graph instead of a tree by allowing connections to all neighboring nodes in a predefined distance metric. Both methods are detailed in Section 3.1.4.

RRT

This work's exploration uses two sampling-based path planning methods, adapts and compares them. The first sampling-based method is RRT which was proposed by LaValle [50]. RRTs are generated by randomly sampling new points in an any-dimensional state space for agents with various Degrees of Freedom (DoF).

An input function is used to determine a new node, that can be connected to the nearest node of the existing tree, based on the randomly sampled point and the agent's kinodynamic constraints. This input function simulates the agent's movement for a defined time delta towards the sampled point which results in the position of the new node.

The regions nearest to a particular node that is a center point in Euclidean space, are called Voronoi regions which are described in Section 3.1.1. Nodes closer to unexplored space have larger Voronoi regions which show that the random sampling is biased towards unexplored space. If a node is close to the goal state, the expansion finishes and the RRT's edges connecting the nodes from start to goal state serve as a path.

Kuffner and LaValle later introduced RRT-connect which uses a greedy extend function that wraps the steer function. For each random sample, the steer function is executed and new nodes are added to the tree multiple times until the random sample is reached or an obstacle blocks the expansion [51].

They also presented a bidirectional RRT planner which constructs two trees, one initialized at the start and the other at the goal state. If two of the trees' particular nodes are close to each other, they are connected and induce the path from start to goal. They furthermore demonstrate their proposed planner for agents with up to 12 DoF [52].

Further details about RRT are explained in Section 3.1.2.

RRG

The Rapidly-exploring Random Graph (RRG) was introduced by Karaman and Frazzoli [17]. RRG combines the construction of a graph from PRM's learning phase with RRT. Instead of a query phase to create a path, it is provided as soon as start and goal are connected through the graph. Additional construction time can then lead to an improved solution.

A thorough explanation of the RRG algorithm is presented in Section 3.1.3.

Improvements to RRT and RRG

These sampling-based methods are still actively researched resulting in a multitude of adaptions and improvements. The research presented in the following is a selection that is relevant to this work because it inspires several additions to it.

In [17], Karaman and Frazzoli proved that RRT's probability to obtain an optimal solution is zero and proposed RRT* which guarantees asymptotic optimality. For this, it checks for all nodes in a radius around the sample if they can be connected to the new node instead of just attempting to connect it to the nearest node. If a connection is possible, it is checked if the new connection can reduce the cost of the already present node. In this case, the tree is rewired.

Karaman et al. [53] proposed anytime RRT* in which the agent starts to move after a user-defined period of time even if no path has been found so far. While the robot moves, the algorithm continues and tries to find or optimize the path. At every node the agent reaches, the currently best path is selected. This is similar to the decoupled gain calculation proposed in Chapter 5.

For a high-dimensional configuration space, Brock and Kavraki [54] proposed to decompose the problem by representing paths as volumes in a low-dimensional space. They deploy wave-front expansion in a 3D workspace to construct a tunnel between start and goal position. It is then utilized to find a real-time solution in the configuration space. In [55], the tunnel is the foundation for a configuration space tree that is very similar to an RRT.

Shkolnik and Tedrake [56] proposed to inflate spheres around each node up to the closest obstacle in the configuration space. Therefore, the distance to the closest obstacle can be calculated or is determined lazily from failed attempts to connect a new sample to the particular node. Random samples inside one of those spheres are rejected which makes the tree sparse and increases samples in difficult regions, leading to an overall increase in efficiency.

Path planning for UAVs using RRG on a point cloud stored in a k-d tree was introduced by Gao and Shen [57]. When a new sample is generated, a line is drawn from it to its nearest neighbor and the intersection with the neighbor sphere's surface serves as the position for a new node. For each new node in the RRG, a sphere is inflated around it up to the closest obstacle which is determined using the k-d tree. If the sphere is not sufficient for the UAV to pass through, it is discarded. Otherwise, the new node is connected to all neighboring nodes with whose spheres it has an intersection area big enough for the UAV to move through. RRG exploration is stopped after a predefined amount of iterations and A* is used to find the shortest path from start to goal position in the graph. The positions and radii of all nodes on this path are then put into a quadratic optimization to find an optimized trajectory.

Gao et al. [58] reworked their initial approach and replaced the RRG with a combination of informed RRT* [59] and anytime RRT* [53]. Furthermore, when discovering new obstacles during trajectory execution, the current path is not discarded completely and a new optimization function is utilized.

The previous approaches from Brock and Kavraki [54], Rickert et al. [55], Shkolnik and Tedrake [56], and Gao and Shen [57, 58] are the inspiration for Node Area Inflation (NAI) introduced in the third iteration of RNE shown in Chapter 7.

Wang et al. [60] showed two improvements to RRT which are designated for cluttered environments. The first improvement is to dynamically adjust the sampling area that is restricted to a sphere around the node nearest to the goal. The sphere's radius is the node's distance to the goal. When encountering an obstacle, the radius is increased. The second improvement is a control value which determines the node around which the sampling occurs. If expanding a node fails too often, the focus shifts to another node to escape local traps.

In [61], dynamic region-biased RRT was proposed in which the workspace is analyzed first to produce a skeleton in obstacle-free areas. It guides the tree growth by restricting the sampling too far from the skeleton. This improves the time to find a path in obstructed environments but also requires detailed knowledge about its topology.

Informed RRT* suggests an ellipsoid with start and goal configuration as focal points and the cost of the current and theoretically best solution for diameters as a restricted sampling space after an initial path was found. This reduces overall exploration but increases the rate of finding a (near) optimal solution [59].

A similar sampling bias like in the previously introduced work is employed in the second iteration of RNE in Chapter 6 to prioritize the placement of new nodes near the robot.

A novel combination of bidirectional RRT and RRT* was employed by Krüsi et al. [62] to plan trajectories on 2.5D maps derived from a point cloud traversability analysis for a UGV. Bidirectional RRT is used to quickly find a traversable path. When it has been found, RRT* improves the path by sampling close to the initial solution. This is similar to the usage of the traversability assessment proposed in [3] which is utilized in

an experiment to evaluate this work's exploration approach in Section 8.6.

2.4. Autonomous Exploration and Inspection Planning

Recent research on autonomous exploration and inspection planning can be separated into four main categories which are frontier-based, sampling-based, information-theoretic and learning-based strategies as shown in Figure 2.3. Published work on those categories is described in the following sub-sections.

2.4.1. Frontier-Based Exploration

Yamauchi [63] introduced the Frontier Exploration (FE) in 1997 which identifies frontiers in a 2D grid map by finding unknown map tiles next to empty or occupied tiles. These frontier cells are clustered. The clusters can now be compared by size or by distance to the robot and selected as the next navigation goal for the robot. But the complexity increases the larger the map grows as all map tiles are checked in each iteration.

In [64], a wavefront FE was proposed in which previously detected frontiers serve as starting points for searching new frontiers after performing one or more scans. Only cells covered in those scans are checked for new frontiers. This reduces the number of necessary checks compared to the naive approach that evaluates all cells in the grid map at each iteration. But this approach was only demonstrated on 2D maps.

Quin et al. [65] focused on improving frontier extraction algorithms based on [63] to accelerate frontier detection and exploration. They proposed to check only occupancy map cells in the Field of View (FoV) of newly acquired sensor measurements to reduce the computational load. Furthermore, only cells intersecting the border of the observed FoV, which can be limited by obstacles, are checked to further reduce their number. They test their algorithms in 2D environments but claim that they are applicable to 3D occupancy grids as well.

Holz et al. [66] segment the map into rooms and make the robot favor frontiers in the same room. Also, the frontier that is the current navigation goal is re-checked to find out if it has already been explored which leads to another frontier becoming the

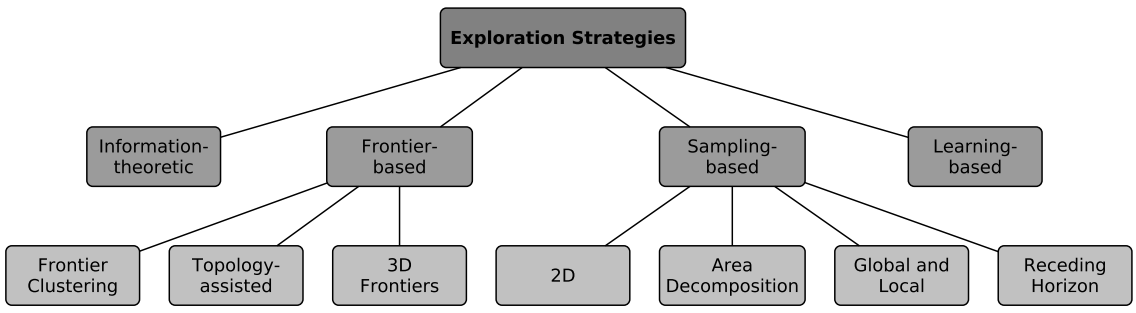


Fig. 2.3.: A visualization of the different categories of exploration strategies is shown in this figure. They are used in this section to categorize the multitude of approaches.

goal. The exploration approach proposed in this work also uses a similar metric to abort the current goal for a better one.

In [67], a thorough comparison of different FE algorithms was undertaken. The authors distinguish between integrated approaches where the map quality is regarded in the target decision, cost-utility approaches, behavior-based approaches and multi-robot approaches which can be separated into coordinated and market-based.

They concluded that integrated approaches increase map quality while algorithms using cost and utility functions increase exploration time overall but explore large parts of the environment early. Considered individually, cost-based approaches decrease exploration time while utility-based approaches explore larger volumes early in the process. As expected, multi-robot approaches improve time and map quality. Especially market-based goal assignment outperforms assigning goals in a predefined sequence.

Topology-Assisted Frontier Exploration

In [68], an occupancy map is skeletonized based on the Voronoi distance which is used to build a Voronoi graph. Based on this graph, junctions are identified that segment the map. For each segment, frontiers are determined and assigned to multiple robots. The assignment is based on each robot's distance to the particular segment and favors robots close to or already inside it.

[69] introduced the ability for an operator to provide a topographical node map of the environment to guide a global exploration. A Travelling Salesman Problem (TSP) solver, which is explained in Section 3.4, is used while FE is running locally. Each frontier detected by FE is assigned to a node to which it has the shortest path. The frontiers are explored in the order of the nodes which they are attached to and which is defined by the TSP solver's solution. The global route can be re-planned if obstacles block the connection between nodes.

Gomez et al. [70] proposed a similar approach by extending FE with a topological graph of the visited frontiers. Each frontier is categorized as a free area in the same room or a transit area to another room which is found using gap detection on walls in the map. The selection of the next goal from the available frontiers is based on the size, distance and category of the particular frontier where free area frontiers in the same room are preferred over transit area frontiers. Loop closures in the topological graph are detected and added.

The previously shown approaches utilize topological information to guide the exploration which is also the idea of the topology-based graph proposed in Section 7.2. But compared to [69], it does not need the topology information previously nor does it assess the complete grid map to retrieve it like in [68] and [70].

Frontier Clustering

In [71], k-means clustering is utilized to group frontiers into a number of clusters equal to the number of robots running the exploration task. Each robot chooses the next cluster based on a cost function that penalizes distance and existing assignment of the cluster to another robot.

By applying a TSP solver on representative nodes of frontier cells clustered by k-means, Kulich et al. [72] decrease the duration and path length of the exploration compared to the greedy strategy of choosing the nearest frontier. In a later study

[73], they compare different FE strategies for determining frontier goal candidates and assigning them to multiple robots. The results of various simulations show that putting more effort into finding better goal candidates is advantageous over utilizing complex goal-assigning strategies like a TSP solver.

In 2019, Kulich et al. [74] introduced an algorithm that outperforms their previous approach by sampling viewpoints for each frontier found by FE in the free space near the respective frontier. This is intended to maximize the number of observable frontier cells. A constrained variant of a TSP solver is then utilized to find the shortest path through the viewpoints. The first viewpoint is chosen as the next goal for the robot.

They also regarded [75], in which the authors found out that a frequency-based goal selection is superior to a decision-based goal selection. A frequency-based goal selection calculates the best goal in set time intervals while a decision-based selection chooses a new goal only after the previous goal was reached. They also state that higher frequencies further increase the benefit up to a limit at which the computational load is too great and reduces the overall performance.

Clustering multiple goals and connecting them using a TSP solver is implemented for this work's exploration in Section 8.2.3. Furthermore, the frequency-based goal selection is implemented in the proposed exploration approach in Chapter 5 as one of its main advantages compared to state-of-the-art sampling-based approaches.

3D Frontier Exploration

Joho et al. [76] detect frontiers on multi-level surface maps and calculate viewpoints by ray tracing from the frontier cell to find suitable positions for the robot. For each viewpoint, 3D ray casting determines the estimated information gain in map cells from the viewpoint. The viewpoint with the best utility considering information gain, distance and traversability score is selected as a goal.

Butzkey et al. [77] use an OctoMap described in Section 3.3.3 to find frontier cells in 3D space. For every frontier cell, possible viewpoints consisting of a position and a heading are generated. Each viewpoint is assigned a gain which is the number of frontier cells that can be observed from it and a cost which is the distance to a robot's current position and heading. An arbitrary number of robots can be used and each of them can be assigned to a viewpoint as soon as it becomes available. A high-level planner decides if a viewpoint can only be reached by a UAV and prioritizes their exploration before sending a UGV to other viewpoints.

An OctoMap is also utilized by Zhu et al. [78]. They only search for new frontiers in and remove explored frontiers from the space in the OctoMap that changed since the previous iteration of the frontier detection. Frontier cells are clustered by proximity and each cluster's geometric center is regarded as a candidate for exploration. The candidate with the best cost function is the next goal for a UAV. The cost function is derived from the proportion of unknown cells to all cells in a sphere around the candidate minus its distance to the robot.

Charrow et al. [79] proposed an approach in which global paths to frontiers are detected in a 3D voxel map while motion primitives are used to sample local paths. For both paths, ray tracing in a 3D voxel grid is utilized to find the path with the best reward. This path is optimized based on its mutual information.

Senarathne and Wang [80] use an OctoMap for exploration and an algorithm identifying surface frontiers rather than free space frontiers which favours the mapping of

obstacles instead of free space. Therefore occupied voxels are found and filtered to extract a single line of surface frontier voxels for each plane that results in vectors that show the probable expansion direction of the surface. Viewpoints are selected from samples based on these vectors and a gain is calculated to extract the best viewpoint to which the robot moves next.

Faria et al. [81] use FE in an OctoMap to find exploration goals for their UAV equipped with a 2D laser scanner mounted at a 50 degrees angle. Fly-by maneuvers are performed around the found frontier goal to map unknown space. Frontiers are derived from the OctoMap structures as described in their previous paper [82].

Vutetakis and Xiao [83] proposed to cluster frontiers that are found at the border between occupied and unknown space in an OctoMap and to sample a user-defined number of viewpoints. Each viewpoint is evaluated regarding a distance threshold to the nearest neighboring frontier and an information gain derived from ray tracing. The viewpoints with the best information gain for each frontier are connected using a TSP solver. The path to the first waypoint is calculated using RRT and the process is repeated for each new goal. The exploration is limited to the UAV's current altitude which is only changed when no frontiers remain at it.

In [84], frontiers are also extracted from an OctoMap. Viewpoints are generated in proximity to the frontier points and the robot. These viewpoints are clustered using k-means and evaluated using ray tracing to calculate the number of frontier points visible from the cluster center. The cluster with the highest amount is selected as the next goal for a UAV to explore.

A frontier exploration based on an octree structure called supereight [85] was introduced by Dai et al. [86]. They extract frontier blocks at each sensor measurement that is added to the mapping structure. A predefined number of candidates is obtained uniformly from large enough blocks and a representative for each candidate is chosen at random from the block. For every candidate, a ratio of gain, which is calculated using sparse ray casting presented in [87], and estimated travel time to the candidate is computed. The candidate with the best ratio is chosen as the next goal for a UAV using informed RRT* for path planning. According to the authors, this approach outperforms RH-NBVP which is introduced in the next section.

All previously described approaches suffer from the increasing complexity of searching for frontiers in a 3D grid map when exploring large environments.

An approach to more efficiently utilize FE with UAVs was introduced in [88]. The authors extract frontiers from an OctoMap but target frontiers in the current FoV first. A cost function tries to maintain a user-defined maximum velocity and punishes exploring frontiers that result in velocity reduction. If there are no frontiers present in the current FoV, recently skipped frontiers are visited using a classic FE approach.

In [13], frontiers are extracted in a user-defined zone around the robot. A convex polyhedron is constructed around the robot's location spanning towards all edges found by ray tracing around the robot. When the robot leaves this polyhedron, the procedure is repeated. The previous robot positions build a global graph whose neighboring nodes are connected. Frontiers are detected and clustered using the Euclidean distance outside of the polyhedron and inside the user-defined zone. The volume, distance and orientation changes to these clusters are then evaluated for the frontier clusters neighboring the robot's current position. The best frontier cluster is selected as the next goal. If no local clusters are found, previous frontier clusters are evaluated using

the cluster's volume and the robot's distance to the cluster.

The approaches presented in [13] and [88] limit the computational complexity by significantly restricting the space to deploy frontier exploration. This removes the growth of computation time with increasing map size.

2.4.2. Sampling-Based Exploration

The following related works of sampling-based exploration approaches are divided into exploration limited to 2D maps, receding horizon sampling, hybrid approaches, exploration area decomposition and further research.

2D Approaches

One of the first proposals to use randomized samples for robotic exploration was introduced by Oriolo et al. [89]. They build a structure called Sensor-based Random Tree which is inspired by PRM and RRT. At each iteration, a random sample in the free space perceived by the robot's sensors is checked. If it has a minimum distance to the robot's position and previously visited nodes of the tree, it is added to the tree and the robot starts moving towards it. If there are no valid samples at the current robot position, it backtracks to the previous node in the tree. If the robot reaches the start node while backtracking, the exploration is finished.

In [90], the authors proposed to detect frontiers in a 2D occupancy grid map by building a local and a global RRT. For each node that was sampled in an unknown cell of the map, a frontier is placed at the boundary between free and unknown space that the edge to this node crosses. Nearby frontiers are clustered and their center is evaluated as a possible goal. The number of unexplored cells in a user-defined radius around the center is the gain and the Euclidean distance between it and the robot is the cost. The frontier center with the largest difference between gain and cost is selected as the next goal. The local RRT is cleared and rebuilt after each goal the robot reaches. If no nodes remain with a gain above a user-defined threshold, the exploration terminates.

The previously introduced algorithm combines sampling- and frontier-based exploration but does not regard the potential of exploring nodes in the RRT that are not directly next to frontiers. This potentially increases the exploration efficiency, as many unexplored areas can be observed from further away.

In [91], frontiers in a 2D occupancy grid map are found by building an RRT and selecting nodes with an edge to a new node in an unknown cell, similar to [90]. But tree edges are removed after placing the nodes to save memory. Furthermore, the environment is separated into quadratic regions which are removed from the sampling area if they have more than a user-defined threshold of nodes inside.

If four surrounding regions to a center region contain more nodes than the threshold and no unknown space, the center region becomes invalid which means all nodes inside it are deleted and it is removed from the sampling area. This reduces computation and speeds up the outwards expansion of the RRT.

Fang et al. [92] proposed an approach similar to [90] where they build an RRT and frontiers are extracted when a new node is sampled in unknown terrain. The RRT is rebuilt after finding a frontier and frontiers must be placed at a certain distance from existing frontiers. Each frontier is evaluated by its information gain, which is

derived from the number of free cells in a radius around the point, and its cost, which is calculated from the Euclidean distance between the point and the robot. The confidence in its localization, that is deduced from the number of occupied cells in a radius around the robot, also influences the evaluation. The best frontier point is chosen as a navigation goal for the robot and re-evaluated after the robot has moved a specific distance.

In [93], a local and a global RRT are built to find boundaries between free and unknown space in a 2D occupancy map similar to [90]. Additionally, seeded growing near wall line segments and an adaptive step size for the RRT's edges, that decreases over time to increase granularity, are proposed.

The above approaches only regard sample positions close to frontiers and do not use samples further away which probably leads to a faster exploration. Furthermore, they create and utilize a 2D representation of the environment while the algorithm introduced in this work is intended for mapping space in 3D.

The authors of [94] create a grid map where each cell holds its probability to be a boundary between known and unknown space called boundariness map. A fuzzy-logic filter calculates the distance of an observable tile from the sensor based on its range and the robot heading. This can be accumulated over a path and is combined with the boundariness values of observable tiles to create an information gain for the path. Samples across the entire map are evaluated and the one with the best score is selected as the next goal. The score is based on observable tiles along the path which is initialized using RRT and then optimized.

This approach only uses a 2D evaluation and has the complexity of frontier-based exploration due to the creation of the boundariness map.

In [95], a semantic road map was proposed which is built similarly to an RRT. Each node is classified into being inside a corridor or a room using semantic mapping and its information gain is evaluated using ray tracing in a 2D occupancy map. The next goal is selected based on the semantic information, information gain and the distance to the node which is calculated using the A* algorithm.

The authors' follow-up paper [96] replaces the tree structure with a graph whose nodes are only sampled in the free space of current sensor measurements. The approach from Wang et al. adds nodes to the graph without collision-checking the edges. Each node's traversability is only checked if it is picked as an exploration goal.

But the authors evaluate gain in a 2D map and the semantic mapping is designed specifically for office environments.

Receding Horizon Sampling

Bircher et al. [6] introduced the Receding Horizon Next-Best-View Planner (RH-NBVP) which deploys RRT for randomly sampling the position and orientation of new nodes. They use ray casting in an OctoMap at each node to find the NBV explained in Section 3.3.2. RH-NBVP follows the RRT's branch with the most potential gain. The gain is computed by adding up the predicted visible voxels at each node in the branch multiplied with a penalizing factor based on the distance to the particular node. After reaching the first node in the branch, the tree is rebuilt. Only the previously best branch is kept. It is designed for autonomous exploration with UAVs

This work builds upon the idea introduced by Bircher et al. but adds a persistent tree and a more efficient gain calculation to speed up the exploration. Furthermore,

RH-NBVP is only designed for UAVs and cannot be used with UGVs. A comparison between an adapted version of RH-NBVP and this work is shown in Section 6.3.

In [97], belief certainty which is computed from landmarks registered by the sensors, is added to RH-NBVP and the path towards tree nodes that minimize the uncertainty, is optimized. Later, Bircher et al. [98] adapted RH-NBVP to inspect surfaces by replacing the gain function from counting unknown voxels to measuring the visible uninspected surface area. Finally, the RH-NBVP algorithm was presented in [99] in detail with all extensions for exploration and inspection path planning.

The authors of [100] utilize RH-NBVP in combination with pose-graph SLAM and local OctoMaps to better cope with drift and to allow loop closures to correct the OctoMaps. The gain evaluation of RH-NBVP polls voxels in different sub-maps based on their position in the pose-graph.

Dang et al. [101] combine RH-NBVP with an additional object recognition using a NN to identify a set of predefined objects. These objects are projected into the OctoMap. The authors deploy a special weight function for the information gain of nodes that are near one of the objects which motivates a detailed exploration and mapping of the whole identified object.

Witting et al. [102] adapt RH-NBVP and instead of sampling each node's yaw, they add all-around yaw evaluation and selection. Furthermore, the robot's position is periodically stored in a history graph. This graph is maintained by continuously checking the potential of each node using a frontier search in a mapping structure called Voxblox. It utilizes Euclidean Signed Distance Fields (ESDF) that are explained in Section 3.3.4. Nodes are refined by moving them away from obstacles using the ESDF's distance to obstacles. If multiple nodes are moved close to each other, they collapse into the same position where their information gain is combined.

The graph's nodes are used for seeding a new RRT when no nodes with sufficient information gain are found in the local RRT. The new local RRT's seed is the closest node to the robot's position in the history graph which has sufficient potential.

The topology-based graph using an ESDF map is an inspiration for NAI and moving of nodes proposed in Chapter 7 of this work.

The authors of [7] proposed the Autonomous Exploration Planner (AEP) which uses RH-NBVP as a local planner while simultaneously caching the node positions and gains from it. If there is no path in the RRT with an information gain above a user-defined threshold, the best cached node is selected as a goal. The cached nodes also contribute to a Gaussian process to calculate a mean information gain for each grid cell in a 2D map. If a new node is sampled in an area of the Gaussian process with a variance that is low enough, the mean value is used instead of calculating the information gain. If a measurement is added to the OctoMap, each cached node's gain inside twice the sensor's maximum range is recalculated. The information gain is calculated by Sparse Ray Casting (SRC) using volume elements of a spherical coordinate system all around the node. The yaw is not sampled but sliding-window summation is used to extract the yaw angle with the best information gain for each node.

This work's Sparse Ray Polling (SRP) is based on the research in [7, 87, 102] to reduce the computation for gain calculation even further while retaining a similar performance. A comparison of AEP to the proposed exploration approach can be seen in Section 6.3.

Schmid et al. [103] improve RH-NBVP by building a continuous RRT* whose root

is reset to the robot's new position after executing a trajectory. Nodes are constantly rewired to maximize their utility. The sampling area is restricted to a user-defined radius around the robot until a minimum amount of nodes is present. Voxblox instead of an OctoMap is utilized for calculating the information gain. An approximation similar to [87] is used to decrease the computation time.

The rewiring used by Schmid et al. is computationally less efficient compared to building an RRG like in the second iteration of RNE shown in Chapter 6.

A 3D exploration based on RH-NBVP and AEP is presented in [104] which adds a weighted gain factor for previously mapped areas of interest. Furthermore, the approach is intended for UGVs with a manipulator arm holding the sensor for mapping. This arm is preferred to move towards nearby goals instead of moving the whole UGV.

Moving just the manipulator arm results in more efficient exploration and inspection but is only applicable for robots with a manipulator arm.

Global and Local Exploration

Dang et al. [19] introduced an exploration algorithm which utilizes RRT* to construct paths similar to RH-NBVP. They employ a cost function for every node in the tree based on the expected observable voxels in an OctoMap in a volume around the node, the distance from the robot and the heading change. After selecting the best branch with the highest overall cost function, the robot's distance from obstacles is used to select the safest path from the best and all similar branches. The similarity is calculated using the dynamic time-warping method.

Nodes from the selected best path and nodes from other paths with a significantly high cost function are connected to nearby nodes and establish a global graph. The leaves from these paths are stored as frontiers in the global graph. When the local RRT*-based method finds no paths above a cost function threshold, the frontier with the most expected information gain is selected as the next goal. The algorithm also tracks the starting position and enables returning to it when the exploration is finished and in case the battery runs low. For calculating the shortest paths in the graph, Dijkstra's algorithm is used which is described in Section 3.1.5. The robot's battery is considered when selecting a new goal by returning to the start if the estimated travel time is higher than the remaining battery.

In another work that was published nearly simultaneously, the previously described approach is extended and named Graph-Based exploration path Planner (GBPlanner) [10]. The local planner's RRT* is replaced with RRG to sample a limited number of nodes and edges in a restricted area around the robot. The shortest paths to each node are calculated using Dijkstra's algorithm. Analog to [19], the best path and other worthy paths are propagated to the global graph. When a local dead-end is reached by the robot, the space restriction for sampling new nodes is lifted and they are connected to the global graph to find suitable areas for exploration.

The GBPlanner's distinction between global and local exploration is also introduced in this work's fourth iteration in Chapter 8. But instead of choosing the best global goal when no local goal is available, a TSP solver is used to determine the best order of visiting all global goals.

The Motion primitives-Based exploration path Planner [105] uses samples derived from motion primitives for a UAV in high-speed flight of 2 m/s in a Voxblox map. The motion primitives are based on the UAV's current velocity and various braking

and turning maneuvers. For each sampled motion primitive, collisions are checked and for the resulting velocity and pose, the possibility of a collision-free follow-up motion primitive is checked. Each branch in the tree of motion-primitive paths is assigned a gain derived from ray tracing at the final state and the time to traverse the path. In addition, deviation from a straight path is penalized. For the best branch, nearby safer paths are identified. The safest path is finally selected from them for navigation.

In [20], the authors add a global planner similar to the GBPlanner's global planner which stores valuable unvisited and visited branches for later exploration and returning to the starting position.

Lee et al. [106] employ a peacock local planner which considers kinodynamic constraints of a UAV and selects the best trajectory regarding motion efficiency and the size of detectable frontier clusters in an OctoMap. The robot's poses are stored in a global graph. If the local planner finds no feasible trajectory, a node from the global graph is selected based on the distance to it and nearby frontier cluster sizes. If a global node is detected in one of the local planner's trajectories, an active loop closing is executed where the robot moves along several previously visited nodes to improve the map accuracy.

In [20] and [106], motion primitives are employed instead of RRT or RRG which allows to maintain higher speeds with UAVs but requires more computation to place a new sample due to the number of possibilities to check.

In [14], GBPlanner is improved to GBPlanner2 which allows its use with legged robots by checking edges between nodes for traversability. Ray casting in Voxblox from the robot to the ground is undertaken to verify whether a ground exists and if the inclination is traversable. GBPlanner2 also offers the option to calculate gain only on leaf nodes and cluster them in a user-defined radius to reduce computation.

For multi-robot exploration, local sub-maps are propagated to a central computer that positions them in a Voxgraph [107] which connects multiple Voxblox maps in a pose-graph. Frontiers are searched and clustered in the particular sub-maps and points are sampled to build a graph in the free space of the sub-map. Nodes nearest to a frontier are linked to it. If a new sub-map arrives from a robot, already observed frontiers are removed. Robots periodically request a goal from the central computer and navigate towards it in the global graph. When they reach it, GBPlanner2 takes over for a user-defined period of time. Afterwards, robots request a new goal or return to the last known location where a connection to the central computer allowed requesting a new goal.

GBPlanner2 was used by the team winning the DARPA SubT Challenge and is compared to the proposed RNE in Section 8.5 regarding exploration duration, traveled path length, mapped volume and computation efficiency. The DARPA SubT challenge is detailed in Section 1.1.4.

Schmid et al. [9] proposed an exploration for large environments called GLocal which divides the sensed map into sub-maps that can be aligned to each other to enable consistent path planning and collision avoidance under severe odometry drift. A Voxgraph is used to manage several Voxblox sub-maps which can be aligned using pose restrictions. New measurements are integrated into a sliding window Voxblox map. When the pose uncertainty exceeds a threshold, older measurements are stored in a sub-map and removed from the sliding window. The local exploration uses a local area which combines all sub-maps in the proximity. An RRT* is built to explore locally using the

distance in the graph and unobserved voxels for cost and gain like in the authors' previous work [103]. If no local goals are detected, global frontiers are searched by checking the surfaces of sub-maps for frontiers while discarding frontiers that are removed by other neighboring sub-maps. The local graphs remain as sub-map skeletons which are used with A* to plan paths to global frontiers.

In [108], an exploration approach that can switch between different behaviors based on a hyper-game formulation was proposed. A UGV with a camera and a lidar can decide between exploration and visual coverage whereas the latter is further divided into sparse and dense coverage. The decision is based on the surrounding topology, the percentage of observable surface in a Voxblox map by the camera and the performance of previous iterations.

The topology is assessed using the point cloud's eigenvalues split into eight octants around the robot. Rooms have a similar distribution in all octants while tunnels do not. The previous iteration's performance is measured in the mapped volume and surface during its execution. For deciding between dense and sparse coverage, the illumination is considered which is derived from the camera's intensity measurements.

The exploration planner utilizes the authors' previous work which is the GBPlanner while the coverage planner samples viewpoints and calculates their visual gain. These viewpoints are then sub-sampled to build sets. The set with the highest gain is selected and a path is calculated using a TSP solver. Sparse and dense coverage modes use slightly different gain calculations regarding the calculated sensor range.

The approach in [108] is designed specifically for certain underground scenarios and therefore is not applicable to explore different environments.

Exploration Area Decomposition

Song and Jo [109] introduced an inspection planner for the exploration and inspection of spatially bounded, unknown areas using a depth camera. They decompose the map into sections for better online planning. A local inspection in each segment is undertaken by sampling viewpoints using RRT*. The respective gain is evaluated with ray tracing in an OctoMap for observable voxels and a Truncated Signed Distance Fields (TSDF) map for surface coverage. TSDF is explained in Section 3.3.4. A coverage analysis of the sampled viewpoints in each particular segment is executed to find a set of samples that cover most of the segment's surfaces. They are connected using a TSP solver and the constructed path is then executed by the robot. Later, Song et al. [110] add a new method to decompose the area to be mapped using seeded local growing in free space clusters. The segments are connected using a TSP solver to form a global path.

Zhu et al. [18] introduced the Dual-Stage Viewpoint Planner (DSVP) which follows a dual approach of sampling-based RRT expansion and frontier exploration where both are deployed on a local and global level. In an area around the robot, frontiers in an OctoMap are extracted which must be observable by ray tracing from the robot or a node in the local RRT. Frontiers close to each other can be clustered to reduce the total number of frontiers. RRT is sampled globally and around local frontiers based on a probability which favors the local frontiers. Every node in the local RRT is a viewpoint whose gain is calculated using ray tracing. The branch with the highest gain, reduced by travel distance and a similarity metric, which compares it to the previous best branch, is picked and executed. Viewpoints with a gain greater than zero are added to the global graph which connects them to the closest existing viewpoints. Edges that

are in a collision or too close to existing viewpoints are discarded.

If there are no local viewpoints with sufficient gain, the global graph is consulted to select a viewpoint that can observe the newest global frontier. This means it is probably closest to the robot. If multiple viewpoints can see this frontier, the one that is closest to it is selected as the goal. If no more global frontiers are available, the exploration finishes.

In [12, 111], the authors proposed a framework for exploration called Technologies for Autonomous Robot Exploration (TARE) that divides the global space into many sub-spaces. A detailed local plan is computed only in the local sub-space while the global plan connects all sub-spaces which reduces the computation time and increases the efficiency immensely. A global road map in traversable space is built which is used in the global plan that must go through all unexplored sub-space centers. The distances in the road map are calculated using A* and are used to build a distance matrix between the centers. A TSP solver is deployed to find the shortest path through them.

The intersecting points between the global path and the sub-spaces' borders are used for the local plan. A set of viewpoints for the local plan is randomly sampled from a lattice pattern in the sub-space until adding new viewpoints yields only a marginal reward in additional sub-space coverage. The sampled viewpoints are then connected to the intersection points via a TSP solver and the path is smoothed.

TARE was used for exploration in the DARPA SubT Challenge but is, like DSVP, designed for a specific sensor head and SLAM approach which was developed by the same authors. DSVP and TARE are compared to RNE in Section 8.5.

Kim et al. [15] introduced a large-scale exploration by constructing a global and a local information road map. The local roadmap consists of uniformly sampled nodes in a sliding window around the robot. At each node, the traversability information and the traversability to neighboring nodes are stored. Globally, the robot's former positions are stored as breadcrumbs. Nodes which are locally sampled near the border between free and unknown space, are denoted as frontiers. A two-level planner is deployed. First, it tries to guide the robot globally which is cascaded down to the local planner that optimizes a receding horizon path randomly sampled through the local road map.

The approach was also utilized in the DARPA SubT Challenge but is designed for legged robots and is not openly available at this time for a comparison to the exploration proposed in this work.

Further Research

An approach to detect candidate viewpoints for exploration was presented in [112] where point pairs are sampled uniformly. If one point from the pair is in the unknown space of an OctoMap while the other is in free space, the latter is selected as a possible viewpoint. For each viewpoint, the information gain is calculated using ray casting in the OctoMap. Every traversed cell is marked and does not count towards the information gain of future ray casts. Viewpoints with an information gain below a threshold are pruned. A TSP solver is used to find the best path connecting all viewpoints where the cost between them is derived from their Euclidean distance and their respective information gain. This favors high information gain viewpoints at the start of the path. All viewpoints' gains are updated and the path is re-calculated after a viewpoint has

been reached by the UAV.

Qin et al. [113] utilize and expand this approach to explore environments with a heterogeneous robot team of UGV and UAV. The UGV executes a coarser initial mapping while the UAV is deployed afterwards to increase map details and fill voids left by the UGV.

The previously described approaches calculate the next goal for a robot only after reaching the previous which increases the exploration duration while the robot stands still during computation.

Pérez-Higueras et al. [114] employ RRT* directly on a voxel-filtered point cloud cropped to an area around the robot for path planning and exploration with a UGV. A sphere is placed at every node and the points and derived surface normals in it are checked for traversability. For the exploration, all leaf nodes are evaluated by comparing their cost which is calculated using the number of points in a sphere around them. Nodes too close to walls and next to already visited nodes are filtered out from the evaluation. If no sufficient node is found, the regarded area is increased up to a user-defined threshold.

Pérez-Higueras et al. only calculate traversability and cost in a rather small area around the robot which makes it prone to getting stuck in a local maximum.

Lindqvist et al. [115] proposed to use RRT* to plan paths for exploration. Multiple goals are sampled in the FoV of the robot's sensor and RRT* is used to build trajectories to them in a user-defined amount of iterations. Goals for which path planning failed, are discarded. For feasible goals, a path cost considering the actuation model of the utilized UAV is calculated. The gain which is derived from the number of unmapped voxels observable over the interpolated positions in the previously optimized path, also influences the path cost. The best goal is then executed and the process is repeated.

In [8], a consistent PRM is built and gain estimation is based on RH-NBVP. Additionally, the authors apply increased weights to unknown voxels near surfaces and free space. Nodes are grouped into subsets by distance from the UAV and nodes with a gain below a user-defined threshold are discarded from the PRM. Different trajectories are evaluated using the node gains and the predicted execution time while dynamic obstacles can be avoided by choosing a different, collision-free trajectory. ESDF-based optimization in a Voxblox map is used to optimize the path with constraints regarding the distance to obstacles and sensor range. It is also designed solely for UAVs.

2.4.3. Information-Theoretic Exploration

A different approach to determine frontiers and the path from the robot to them was proposed by Shade and Newman [116]. Their algorithm is based on a discrete equation of gas flow from well to drains in a 3D occupancy map. The well is the robot's position while drains are unknown cells next to free cells in the occupancy map. By solving a partial differential equation, a 3D vector field indicates the gas flow. The maximum gas flow in the vector field is used to create a path from the robot to the particular frontier.

In [117], regions to be explored next are discovered based on mutual information with the assumption that the proposed functions are attracted to unknown space because it maximizes the mutual information reward. The authors prove their assumption in [118] and provide an algorithm to calculate the mutual information in a reasonable

time.

Maffei et al. [119] introduced an exploration based on potential fields that lead a robot towards frontiers and away from obstacles. The fields are guided by rails constructed from a Voronoi skeleton of the map. The authors add potential to previously visited areas which increases over time to encourage loop closures to reduce uncertainty.

In [120], map and path entropy are considered during exploration. The authors proposed to follow the gradients of potential information fields on a 2D occupancy grid to identify the next goal. Pose-based SLAM is used to calculate the entropy for the map. If the robot starts at uncertain configurations, it tries to achieve loop closure to increase certainty and overall entropy.

Wang et al. [121] introduced exploration based on information potential fields where the border to unknown space attracts the robot and obstacles repel it. When the robot stays too long in the same area, the repulsion for this area grows to eliminate local minima. The next goal is selected based on proximity to the robot and expected information gain.

In [122], the authors use an OctoMap for mapping and project its information to a 2D occupancy grid map to calculate the Shannon entropy which is used to guide the exploration of a UAV.

Pimentel et al. [123] combine an information-theoretic approach with RRT by biasing the RRT's sampling towards areas with higher expected information gain with a roulette wheel selection. The higher an area's information gain, the more likely a sample is placed towards it. But there is always a chance for random samples to escape local minima. These areas are detected by propagating wall segments at frontiers in an occupancy grid map and calculating the information gain of the propagation. The first frontier to be reached by RRT is the next goal for the robot.

In [124], paths to explore frontiers in an occupancy grid map are evaluated regarding the expected information gain and the localization and map uncertainty. If the uncertainty is high, the expected information gain is reduced which favors paths through explored regions to find loop closures.

All previous information-theoretic approaches suffer from the same disadvantage as frontier-based methods. The complete map is evaluated to find regions with high potential which increases the computation significantly while the map grows.

Shen et al. [125, 126] proposed a stochastic differential equation to simulate the expansion of particles in an area. The particles are released from the robot in certain time intervals and have ideal interactions with obstacles. This results in an equal distribution of particles over the map except for unexplored areas where no obstacles stop the particles and they can spread out further. From these increased distances between particles, exploration goals are calculated and used as navigation goals for the robot, starting with the closest one. The number of particles is kept within bounds by weighing and re-sampling them periodically.

This approach is very dependent on the user-defined number of particles and the selection of the closest unexplored area is often inferior to a more sophisticated cost function as stated in [67].

2.4.4. Learning-Based Exploration

Bai et al. [127] proposed the construction of a learning-based local planner for 2D maps that is trained with randomly created 2D dungeon maps and a mutual information optimization algorithm. The planner decides where to go when presented with a locally extracted occupancy grid map. The training and testing data sets are tested on multiple award-winning deep NNs from image evaluation contests. Their local planner is still outperformed by an exhaustive mutual information-based planner but requires far less computing power. It is also prone to getting stuck in dead ends.

In [128], a Red Green Blue - Depth (RGB-D) camera is used to explore an environment based on deep reinforcement learning that uses the camera's input and short-term as well as long-term memory to define the next action.

Chen et al. [129] proposed to utilize NNs that make decisions based on a reward information gain function. It is derived from the input of an RGB-D camera, a small detailed extract from the occupancy grid map around the robot and a larger coarse extract from the grid map. The NN is trained using a data set of human explorations of houses and refined with the reward function afterwards.

A similar approach was introduced in [130] where a NN is trained and used with the robot's position, an image of a condensed grid map and the positions of clustered frontier locations. The NN's weight function is trained to obtain as much information gain as possible early in the exploration.

The authors of [131] train a NN to predict obstacle and free areas on a 2D occupancy grid map. The prediction is limited to frontier areas and a user-defined bounding box. It is based on algorithms to complete images. Therefore, the occupancy grid area is converted to an image. The prediction from the NN is used with regular exploration approaches. Information gain for specific frontiers is based on a flood fill algorithm to calculate the size of predicted areas in the map.

Ly and Tsai [132] train a NN to find the best goal based on maximizing the information gain derived from visible boundaries between known and unknown space. The NN is trained to find global viewpoints that cover as much space as possible without regarding local path planning.

Reinhart et al. [133] proposed a NN-based exploration planner for underground tunnels and junctions which decisions are based solely on lidar data as input. It is trained with data and decisions from explorations using GBPlanner. Compared to the GBPlanner, the computation is minimal and no global map is stored while the resulting paths still resemble its model's performance.

Even though the previously presented research is computationally more efficient compared to the three other categories of exploration, it is very reliant on the training data. Therefore, it must be specifically trained for most surroundings and cannot be utilized in arbitrary environments. Furthermore, local dead-ends can stop the learning-based exploration prematurely as it is mostly based on actual sensor data.

2.5. Next-Best View Calculation

Conolly [134] introduced the term Next-Best View (NBV) for sampling viewpoints and selecting the best one to increase the observed volume of an object. Two algorithms are presented where the first one uniformly samples viewpoints on a sphere around an

object of interest and evaluates them via ray tracing in an octree. The second algorithm places a viewpoint on a vector from the origin based on the number of unseen faces for each particular axis. The second algorithm is several magnitudes faster but fails to cope with self-occlusion.

Strand and Dillmann [135] proposed to build a 3D map for indoor office environments using a 2D map for exploration and finding an NBV based on a decision diagram. The environment is categorized into rooms and floors where the former are explored in 2D. If a new room is detected in the 2D grid map, a 3D scan is undertaken to map it.

In [136], the authors utilize ray casting in an OctoMap and the kinematically restricted view space of the sensor to find an NBV for the exploration of frontiers and cavities called voids.

The authors of [137] reconstruct an object with a mobile robot utilizing the different depths of an OctoMap by applying ray tracing on its coarsest level first and increasing the ray density only for occupied voxels. This saves up to 90% computation time compared to classical ray tracing. The unknown voxels detected by ray tracing, the overlapping surface with previous observations and the distance to the current view are evaluated to select the NBV.

In a subsequent publication [138], they add orthogonality of the view and possible occlusion to the evaluation. Furthermore, neighboring viewpoints' scores are also evaluated and taken into consideration to account for positioning errors.

Vasquez-Gomez et al. [139] later adopt a RRT-like method to generate viewpoints for object reconstruction while the information gain is calculated using ray tracing for the utilized sensor's FoV. A gain is calculated for each view and the NBV is selected regarding path length and entropy.

Maurović et al. [140] introduced a 3D indoor exploration by using 2D exploration for a ground plan with edges to unexplored territory that are compared and used as navigation goals. An algorithm detects single rooms which are then explored in 3D with a tilting laser scanner. The NBV is evaluated similarly by using planes to unexplored space. After mapping a room, the 2D exploration continues. The evaluation is based on the observable unknown areas at candidate positions.

In [141], NBVs are computed with ray tracing using Markov chains to model state transitions between free, occupied and unknown cells in the occupancy grid map.

Daudelin and Campbell [142] proposed to reconstruct objects with a mobile robot by finding an NBV without a limiting bounding box or prior knowledge of the object and its size. The information gain of sampled viewpoints is calculated using ray tracing in an OctoMap. Ray tracing is performed all around the viewpoint to evaluate the information gain for all possible orientations of the robot.

The research on determining NBVs for object reconstruction and inspection is closely related to the gain calculation deployed in most of the exploration approaches presented in the previous sections.

2.6. Conclusion

This chapter introduces ROS as the robotics framework designated for the development and research presented in this work. Related work about existing state machines and behavior trees for ROS lead to the decision to design and develop a novel state machine.

It is tailored for this work's research and the UNDROMEDA project. This state machine is presented in Chapter 4.

Based on the research regarding path-planning and autonomous exploration presented in this chapter, a novel approach named RNE is developed in this work. Its goal is to enable 3D exploration on a large scale. Therefore, a sampling-based approach is utilized because frontier- and information-based explorations' computational load increases more significantly with map size.

RNE utilizes the advantages of a persistent RRT compared to a costly rebuilding of a sampling-based structure proposed by Bircher et al. [6]. A computation-efficient gain calculation is added, inspired by the works of Selin et al. [7] and Oleynikova et al. [87]. This is shown in Chapter 5.

The findings of Karaman and Frazzoli [17] inspire the switch from RRT to RRG in Chapter 6. This is intended to improve the path planning for the robot. Also, the works of Brock and Kavraki [54] and Rickert et al. [55] are the foundation for the topology-based NAI proposed in Chapter 7. The movement of node centers further away from obstacles reduces the risk for the robot and increases the observable volume.

Furthermore, the distinction between a global and a local exploration including a TSP solver for the global part is introduced in Chapter 8. It is based on [7, 9, 12, 14, 18, 19, 20] and the success of some of these hybrid exploration approaches in the DARPA SubT Challenge.

The proposed approach's goal is to be suitable for different UGVs using a variety of sensor configurations. All of the previously introduced exploration approaches are either limited to UAVs or legged robots, require a specific sensor or SLAM solution or are not openly available for replication and usage.

3. Mathematical Foundations

In this chapter, mathematical foundations and notations for the approach proposed in this work are presented. First, the foundations for the sampling-based path planning algorithms RRT, RRG and PRM are shown. These are utilized for the proposed exploration approach which is presented in Chapters 5 and 6.

Furthermore, Dijkstra's algorithm and k-d trees are explained which are utilized by RNE. Then, details about Simultaneous Localization and Mapping (SLAM) are described as it is required to allow autonomous exploration.

The exploration of unknown environments, the term Next-Best View (NBV) and three different map representations are detailed afterwards. These are based on octrees or signed distance functions. The former is utilized in this work and related work while the latter is used by several state-of-the-art approaches.

Finally, a method to find the shortest path through all nodes in a graph is presented. This is relevant for the hybrid exploration approach shown in RNE's fourth iteration in Chapter 8.

3.1. Sampling-Based Algorithms

This section introduces several sampling-based algorithms that are relevant to the proposed work or to related work that is compared to RNE. These sampling-based algorithms were introduced for robotic path planning and became popular in robotic exploration frameworks as well [6, 8, 9, 10, 18]. To understand the mode of operation of RRT and RRG, the principle of Voronoi decomposition is especially relevant. Therefore, it is described first, followed by RRT, RRG and PRM. Then, Dijkstra's algorithm is explained which is used to find the shortest paths in graphs and can be applied to RRG and PRM. Finally, k-d trees are described which can be used for effective nearest-neighbor searches.

3.1.1. Voronoi Decomposition

The Voronoi decomposition or Voronoi diagram is named after the Russian mathematician Georgy Voronoy and describes the partition of space into Voronoi regions based on the distribution of centers. For every point in space, the nearest center is determined. This means, for all points in a region, the region center is the closest center in space. In this work, the Euclidean distance D, which is shown in Equation (3.1) between the points $\mathbf{p}$ and $\mathbf{q}$ in n dimensions, is utilized as a distance metric.

Figure 3.1 shows an exemplary Voronoi decomposition in two-dimensional space using the Euclidean distance from Equation (3.2) to divide the areas into the different Voronoi regions. The diagram shows that centers near the edge of the depicted space have regions that expand up to the edge and if the space would be expanded, even further. This finding is crucial for understanding RRTs that are introduced next.

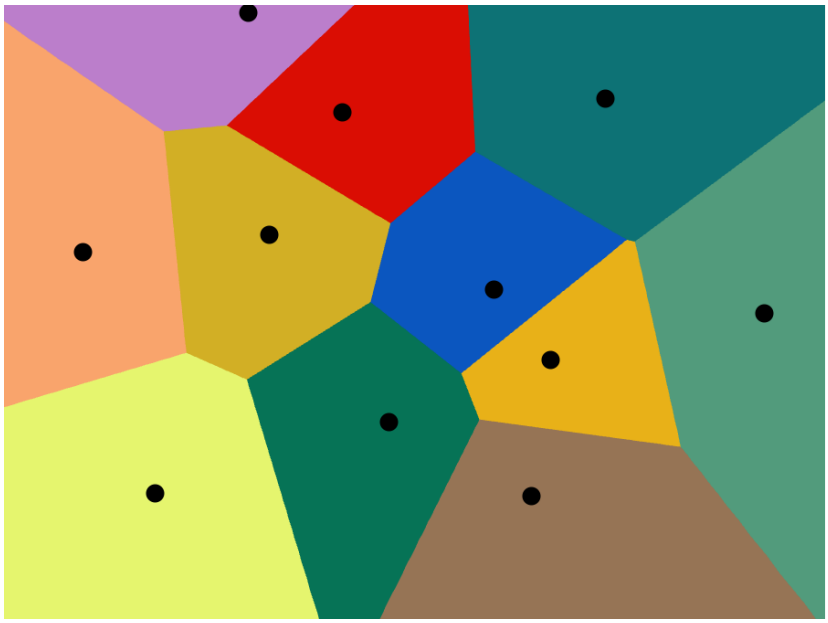


Fig. 3.1.: Exemplary Voronoi decomposition with differently colored Euclidean distance Voronoi regions where each center is represented as a black dot.

$$D(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (\mathbf{p}_i - \mathbf{q}_i)^2} \tag{3.1}$$

$$D(\mathbf{p}, \mathbf{q}) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} \tag{3.2}$$

3.1.2. Rapidly-Exploring Random Tree

The Rapidly-exploring Random Tree (RRT) was introduced by Steven LaValle [50] in 1998 and further researched together with James Kuffner [52]. It is a simple, yet effective tool to rapidly explore space which is mostly deployed for path planning. LaValle and Kuffner use it for planning paths in high-dimensional space for robots with up to 12 DoFs while considering kinematic constraints and keeping a safe distance from obstacles.

The path planning problem in the configuration space C is generally described as finding a path from the starting position $\mathbf{p}_{init}$ to the goal region $C_{goal} \subset C$ which are connected and lie entirely in the free space $C_{free} \subset C$. C is the union of free and obstacle space $C = C_{free} \cup C_{obs}$ where the latter has to be avoided.

For higher-dimensional problems, RRT is deployed in the state space where each node corresponds to a certain configuration that can be translated to the metric space. For this work, only the 2D and 3D metric spaces are relevant. Therefore, details about solving higher-dimensional problems are omitted here for brevity and can be found in [52].

Algorithm 3.1 shows the construction of an RRT with k iterations, starting at position $\mathbf{p}_{init} \in C_{free}$. First, the tree $\mathcal{T} = (N, E)$ is initialized with $\mathbf{p}_{init}$ in the set of nodes N and an empty set of edges E.

Algorithm 3.1: RRT - Construction of a Rapidly-exploring Random Tree

Input: $\mathbf{p}_{init}$

1: $N \leftarrow \{\mathbf{p}_{init}\}$
2: $E \leftarrow \emptyset$
3: **for** $k_i \leftarrow 1$ **to** k **do**
4: $\quad \mathbf{p}_{rand} \leftarrow$RANDOMSAMPLE()
5: $\quad \mathbf{p}_{near} \leftarrow$ NEARESTNEIGHBOR($\mathbf{p}_{rand}, N$)
6: $\quad \mathbf{p}_{new} \leftarrow$STEER($\mathbf{p}_{near}, \mathbf{p}_{rand}$)
7: $\quad$ **if** COLLISIONFREE($\mathbf{p}_{new}, \mathbf{p}_{near}$) **then**
8: $\quad\quad N \leftarrow N \cup \mathbf{p}_{new}$
9: $\quad\quad E \leftarrow E \cup (\mathbf{p}_{new}, \mathbf{p}_{near})$
10: $\quad$ **end if**
11: **end for**
12: **return** $\mathcal{T} := (N, E)$

During each iteration, a random sample $\mathbf{p}_{rand} \in C$ is generated for which the nearest neighbor in N is searched afterwards using a distance metric, e.g., the Euclidean distance D from Equation (3.2). Then, an input is derived from the set of possible actions for the robot defined by its movement constraints and possible obstacles blocking the path.

This input can be described as a function that takes $\mathbf{p}_{near}$ and moves towards $\mathbf{p}_{rand}$ for a specific time. A simple function is moving in a straight path but functions resulting in curved paths are also possible. With this function, a new node $\mathbf{p}_{new}$ is created at the calculated position. This can be seen in Figure 3.2 and is executed by the STEER method.

In the remainder of this work, the input function always moves the robot in a straight line. New nodes $\mathbf{p}_{new}$ resulting from the input function must have a minimum distance to $\mathbf{p}_{near}$ of d_{min} and a maximum distance to $\mathbf{p}_{near}$ of d_{max}.

Afterwards, the COLLISIONFREE method is used to check if $\mathbf{p}_{new}$ and $\mathbf{p}_{near}$ can be connected with a new edge or if an obstacle blocks the way. If there is no collision, $\mathbf{p}_{new}$ and the new edge are added to the tree.

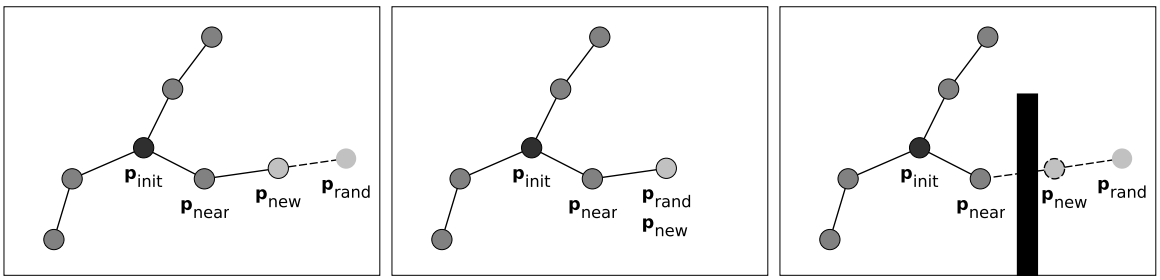


Fig. 3.2.: Three different scenarios for the input function are shown above. It is also referenced as the STEER function and describes a simple straight path with a minimum and maximum distance. On the left, $\mathbf{p}_{rand}$ is sampled and the nearest node $\mathbf{p}_{near}$ in the existing tree is used to determine the position of $\mathbf{p}_{new}$ which is added to the tree. In the middle, $\mathbf{p}_{rand}$ is sampled close enough to $\mathbf{p}_{near}$, so that the input function can reach it. This results in placing $\mathbf{p}_{new}$ at $\mathbf{p}_{rand}$. On the right, an obstacle is between $\mathbf{p}_{rand}$ and $\mathbf{p}_{near}$ which results in a failed COLLISIONFREE method and a failure to add $\mathbf{p}_{new}$ to the tree.

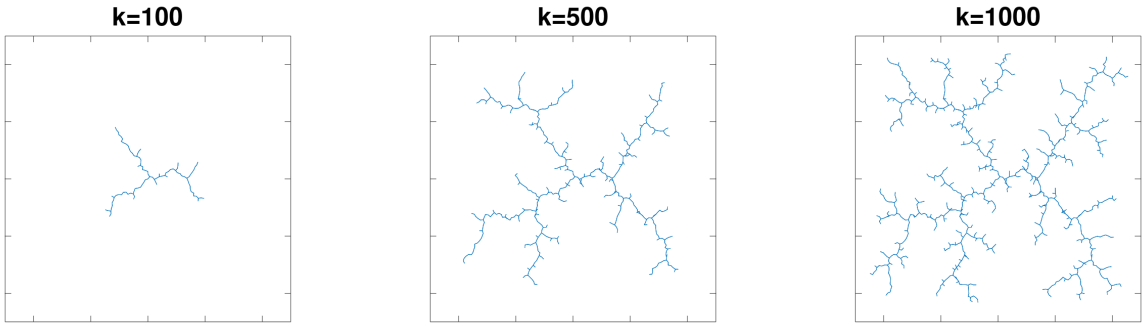


Fig. 3.3.: Three growing stages of RRT in 2D space with a simple STEER input function that always moves 1 m in a straight line. The number of iterations k is listed above each stage.

Due to connecting $\mathbf{p}_{new}$ to $\mathbf{p}_{near}$ in $\mathcal{T}$, RRT is more likely to expand outwards instead of increasing the number of nodes in central regions. This is caused by the Voronoi regions shown in Section 3.1.1. The outermost nodes in $\mathcal{T}$ have the largest Voronoi regions and are therefore more likely to be selected as $\mathbf{p}_{near}$ than other nodes. Because of this, RRT expands more rapidly than other methods which randomly select existing nodes and increment the tree from them [50]. Figure 3.3 demonstrates the growth of RRT in 2D space where the STEER input function moves in a straight line.

Karaman and Frazzoli [17] analyzed and compared multiple random-sampling algorithms for finding optimal solutions in path planning and proposed additions to existing algorithms to increase their chance of achieving the optimal solution.

They state that RRT is probabilistically complete which means that if the iterations k go to infinity and a path exists between $\mathbf{p}_{init}$ and C_{goal}, it is found. Equation (3.3) shows the probability P that a solution is found approaches 1 exponentially because one of the nodes in the tree N_k after k iterations lies inside the goal area C_{goal}. a and k_0 are constants.

$$P\left(\{N_k \cap C_{goal} \neq \emptyset\}\right) > 1 - e^{-ak},\ k > k_0 \in \mathbb{N},\ a > 0 \tag{3.3}$$

Karaman and Frazzoli also establish that RRT is not asymptotically optimal. This means that even if the number of iterations k approaches infinity, RRT does not find the optimal solution.

Therefore, they proposed asymptotically optimal RRT* in which a new node is connected to the neighboring node with the lowest cost path and not the nearest neighboring node. This can cause further rewiring of the tree to optimize each node's path.

Various further extensions and improvements to the RRT-algorithm are introduced in Section 2.3. Details about their implementation and mathematical background are referred to the respective publications.

3.1.3. Rapidly-Exploring Random Graph

Based on RRT, Karaman and Frazzoli introduced the Rapidly-exploring Random Graph (RRG) [17]. Instead of building a tree by connecting new nodes only to the nearest existing node, a new node is connected to all existing nodes within a sphere with a specified radius around its position.

Algorithm 3.2: RRG - Construction of a Rapidly-exploring Random Graph

Input: $\mathbf{p}_{init}$

1: $N \leftarrow \{\mathbf{p}_{init}\}$
2: $E \leftarrow \emptyset$
3: **for** $k_i \leftarrow 1$ **to** k **do**
4: $\quad \mathbf{p}_{rand} \leftarrow$ RANDOMSAMPLE()
5: $\quad \mathbf{p}_{near} \leftarrow$ NEARESTNEIGHBOR($\mathbf{p}_{rand}, N$)
6: $\quad \mathbf{p}_{new} \leftarrow$ STEER($\mathbf{p}_{near}, \mathbf{p}_{rand}$)
7: $\quad$ **if** COLLISIONFREE($\mathbf{p}_{new}, \mathbf{p}_{near}$) **then**
8: $\quad\quad N \leftarrow N \cup \mathbf{p}_{new}$
9: $\quad\quad E \leftarrow E \cup (\mathbf{p}_{new}, \mathbf{p}_{near})$
10: $\quad\quad P_{radius} \leftarrow$ NODESINRADIUS($\mathbf{p}_{new}, N, r$)
11: $\quad\quad$ **for each** $\mathbf{p}_{radius} \in P_{radius}$ **do**
12: $\quad\quad\quad$ **if** COLLISIONFREE($\mathbf{p}_{new}, \mathbf{p}_{radius}$) **then**
13: $\quad\quad\quad\quad E \leftarrow E \cup (\mathbf{p}_{new}, \mathbf{p}_{radius})$
14: $\quad\quad\quad$ **end if**
15: $\quad\quad$ **end for**
16: $\quad$ **end if**
17: **end for**
18: **return** $\mathcal{G} := (N, E)$

Algorithm 3.2 shows the construction of RRG. Similar to RRT, the graph $\mathcal{G} = (N, E)$ is initialized with a node at position $\mathbf{p}_{init}$ and an empty set of edges E.

In each of the k iterations, a random sample $\mathbf{p}_{rand}$ is placed and the nearest neighbor node $\mathbf{p}_{near}$ is determined. If the space between $\mathbf{p}_{near}$ and the STEER function's position $\mathbf{p}_{new}$ is collision-free, the new node and edge are added to the graph. In addition, all nodes within a radius r around $\mathbf{p}_{new}$, which are connectable using the input function, are stored in P_{radius}. For each of them, it is checked if an edge between them and $\mathbf{p}_{new}$ is without collision. If it is, the particular edge is added to E as well.

This creates an undirected graph where the cost of a path is derived from the set of edges E connecting $\mathbf{p}_{init}$ and C_{goal} with the minimum cost. Such a path can be found using Dijkstra's algorithm which is presented in Section 3.1.5. RRG is also asymptotically optimal like RRT* [17]. But it has the disadvantage of requiring additional memory space to store all additional edges compared to an RRT.

3.1.4. Probabilistic Roadmaps

Probabilistic Roadmaps (PRM) were proposed by Kavraki et al. in 1996 [48] and consist of a two-phase path planner. In the first phase, which is called the learning phase, the space is explored and a forest is built which optimally grows into a tree. The second phase, which is called the query phase, uses the tree to solve arbitrary start and goal queries in the explored space.

In a second publication, Kavraki et al. [49] introduced sPRM which removes a heuristic to increase sampling near obstacles and allows to connect a new node with all nodes in a radius around it, therefore creating a random geometric graph instead of a forest or tree.

Algorithm 3.3: sPRM - Construction of simplified Probabilistic Roadmaps

```
1: N ← ∅
2: E ← ∅
3: for k_i ← 1 to k do
4:     p_new ←randomSample()
5:     P_radius ←nodesInRadius(p_new, N, r)
6:     for each p_radius ∈ P_radius do
7:         if collisionFree(p_new, p_radius) then
8:             E ← E ∪ (p_new, p_radius)
9:         end if
10:    end for
11:    N ← N ∪ p_new
12: end for
13: return G := (N, E)
```

The PRM is, similar to RRT, designed for solving path planning in a robot's configuration space with higher dimensions but can also be applied to the metric space in 2D and 3D. In this work, only the metric space is relevant. The path planning problem must be solved in space $C = C_{free} \cup C_{obs}$ which can be divided into free space C_{free} and obstacle regions C_{obs}.

sPRM's learning phase is shown in Algorithm 3.3. The graph $\mathcal{G} = (N, E)$ is initialized with an empty set of nodes N and edges E. In each of the k iterations, a random sample $\mathbf{p}_{new}$ is placed in C_{free} and is connected to all nearby nodes P_{radius} for which a collision-free edge can be placed. All edges and the random sample are added to $\mathcal{G}$.

The query phase inserts $\mathbf{p}_{start}$ and $\mathbf{p}_{goal}$ and tries to connect them to nearby nodes in N. If the connection is successful and there exists a path from $\mathbf{p}_{start}$ to $\mathbf{p}_{goal}$ through $\mathcal{G}$, the path planning is successful. Otherwise, the learning phase requires more iterations to explore the space and connect all sub-graphs with each other. To find the shortest path in $\mathcal{G}$, Dijkstra's algorithm can be used.

While both PRM and sPRM are probabilistically complete, the former is not asymptotically optimal. This means, it does not converge to an optimal solution if the sample size approaches infinity [17].

The proposed sPRM has the disadvantage of an increasing time complexity when adding new nodes to the steadily growing graph $\mathcal{G}$. Therefore, a nearest neighbors method was proposed by Kavraki et al. [49] which only allows to connect a new node to a user-defined number of nearby nodes. This removes the growing time complexity but also the asymptotic optimality [17].

Karaman and Frazzoli [17] proposed PRM* which combines the advantages of PRM and sPRM. It is asymptotically optimal and has a constant time complexity. This is achieved by replacing the static radius r for connecting nearby nodes with a function that decreases while the sample size increases.

3.1.5. Dijkstra's Algorithm

Edsger Dijkstra published his algorithm in 1959 [143] which finds the shortest path between two nodes in a graph without negative edge weights. The algorithm can also produce a shortest-path tree which is a spanning tree in a graph that features all

Algorithm 3.4: Construction of a shortest-path tree using Dijkstra's algorithm

Input: $\mathcal{G}, n_{init}$

1: $Q \leftarrow \emptyset$
2: **for each** $n \in N$ **do**
3: $\quad c_n \leftarrow \infty$
4: $\quad p_n \leftarrow \varnothing$
5: $\quad Q \leftarrow Q \cup n$
6: **end for**
7: $c_{n_{init}} \leftarrow 0$
8: **while** $Q \neq \emptyset$ **do**
9: $\quad n_{next} \leftarrow n$ **for** $n \in Q$ **with min** c_n
10: $\quad Q \leftarrow Q \setminus n_{next}$
11: $\quad$ **for each neighbor** n_{nb} **of** n_{next} **still in** Q **do**
12: $\quad\quad c_{new} \leftarrow c_{n_{next}} + c(n_{next}, n_{nb})$
13: $\quad\quad$ **if** $c_{new} < c_{n_{nb}}$ **then**
14: $\quad\quad\quad c_{n_{nb}} \leftarrow c_{new}$
15: $\quad\quad\quad P_{n_{nb}} \leftarrow n_{next}$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end while**
19: **return** $C := \{c_0, c_1, ..., c_n\}, P := \{p_0, p_1, ..., p_n\}$

shortest paths from a root node to every other node in the graph. For the first variant, the algorithm is stopped as soon as the shortest path to the desired second node is found.

The Algorithm 3.4 requires a graph $\mathcal{G} = (N, E)$ with nodes N and edges E as well as a starting node n_{init} as input. Each edge has a designated cost $c(n_i, n_j)$ which describes the cost of moving from node n_i to node n_j. To initialize the search for shortest paths, all nodes $n \in N$ in the graph are associated with a cost $c_n \in C$ of infinity and no parent node $p_n \in P$. Afterwards, they are added to the queue Q and the cost for n_{init} is set to 0.

While Q is not empty, the node n_{next} with the lowest cost is selected and removed from it. For each of its neighbors, a new cost c_{new} is calculated that consists of the cost of n_{next} and the cost of the edge between it and the neighboring node n_{nb}. If c_{new} is lower than n_{nb}'s current cost $c_{n_{nb}} \in C$, it is replaced and n_{next} becomes n_{nb}'s parent.

Figure 3.4 shows the exploration of a small exemplary graph using Dijkstra's algorithm starting at node A. Because a queue is employed which is ordered by the nodes' cost, the shortest path for a particular node is known as soon as it is removed from Q. Therefore, the algorithm can be used to find the shortest path between n_{init} and a goal node n_{goal} by finishing when n_{goal} is removed from Q. To get the shortest-path tree, the algorithm continues until Q is empty.

3.1.6. k-d Tree

The k-d tree was introduced by Jon Louis Bentley [144] as a multidimensional binary search tree where k stands for the dimensionality. It is utilized in this work for efficient nearest-neighbor and radius searches in a graph as required by the RRT, RRG and

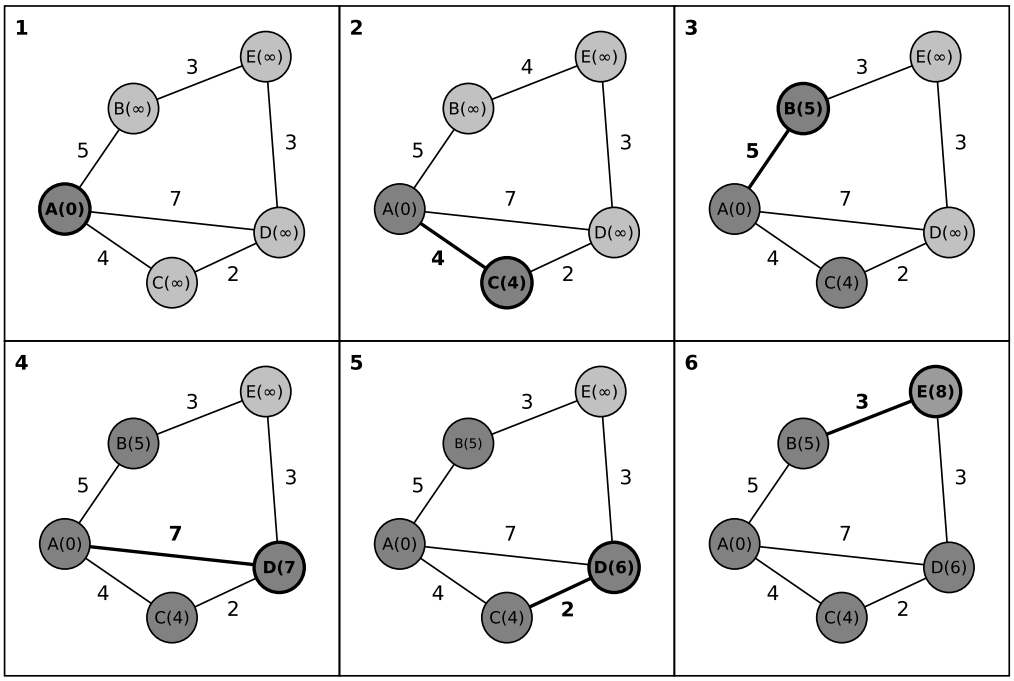


Fig. 3.4.: The application of Dijkstra's algorithm on an exemplary graph with five nodes and starting node A is shown in steps 1 to 6. The numbers inside parentheses in the nodes and the numbers on the edges show their respective cost. Nodes and edges are marked bold if they are currently updated and turn dark gray after being updated.
The first step shows the initial graph with all nodes initialized with a cost of infinity and the starting node A with a cost of 0. In the second, third and fourth steps, the neighbors of A, which are B, C and D, are updated with the respective edge costs. Step 5 shows the improvement of the cost of node D from node C. In step 6, node E is updated from node B. Updates from nodes D and E are omitted in this figure because they do not change the graph.

PRM algorithms. The k-d tree stores points in space by partitioning the space with hyperplanes. Derived from these partitions, the points are separated into the different binary branches of the tree.

To construct a k-d tree for n points in k dimensions, the median point regarding the first dimension is selected and it is split orthogonally to the point by a hyperplane in this dimension. This point is the first node and becomes the root of the tree. For all points on one side of the hyperplane, the median in the second dimension is selected and another hyperplane is constructed which is orthogonal to the second dimension at this point. The same is applied to the points on the other side of the first hyperplane. The points are then added below the root node. This process is repeated as long as there are multiple nodes on one side of the hyperplane. If only one point remains, it becomes a leaf node. While iterating through the points, the dimension orthogonally split by a hyperplane changes every iteration. If the highest dimension k was split in the previous iteration, the first dimension is split next. Figure 3.5 shows the partition of space and the corresponding constructed k-d tree for an example in two dimensions.

When inserting or removing elements from the k-d tree, re-organization or re-balancing of affected branches of the tree can be necessary to maintain the tree's performance for nearest-neighbor and radius searches.

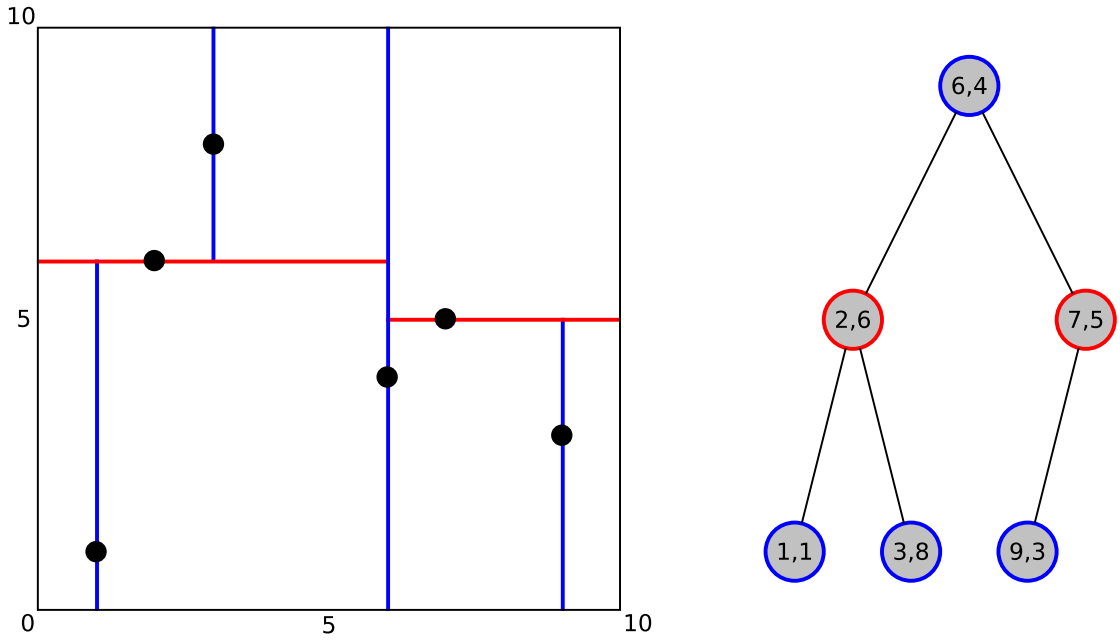


Fig. 3.5.: A 2D space partitioned by a k-d tree's points on the left and the corresponding k-d tree on the right. Splits on the x-axis are highlighted in blue and splits on the y-axis in red. The points' positions are marked with black dots on the left and correspond with the nodes' coordinates on the right.

3.2. SLAM

To explore an area and plan paths to desired positions, it is necessary to precisely localize the robot and construct a map, which is a representation of the world around it, using the robot's sensors. But to derive an exact location of the robot, a map is required with which the sensor measurements are aligned and to build a map, measurements must be integrated from a known position of the robot.

This problem was termed Simultaneous Localization and Mapping (SLAM) by Hugh Durrant-Whyte and Tim Bailey [145] but it has been researched before. In the following, three approaches to solve SLAM are explained. The descriptions, figures and equations only show the 2D case for simplicity. These are based on Extended Kalman Filters (EKFs), particle filters and graphs. The first approach is chosen to generally explain the SLAM method while the second and third are used in this work or relevant related work.

3.2.1. Extended Kalman-Filter SLAM

EKF SLAM is an online SLAM approach which means that it calculates the possibility P for the robot's pose x_t and the map m using only currently available measurements $z_{1:t}$ and odometry $u_{1:t}$ as can be seen in Equation 3.4 and Figure 3.6.

$$P(x_t, m|z_{1:t}, u_{1:t}) \tag{3.4}$$

The measurements in this context are unique landmarks in the map. Normally, for each measurement, a match between it and each particular landmark must be calculated

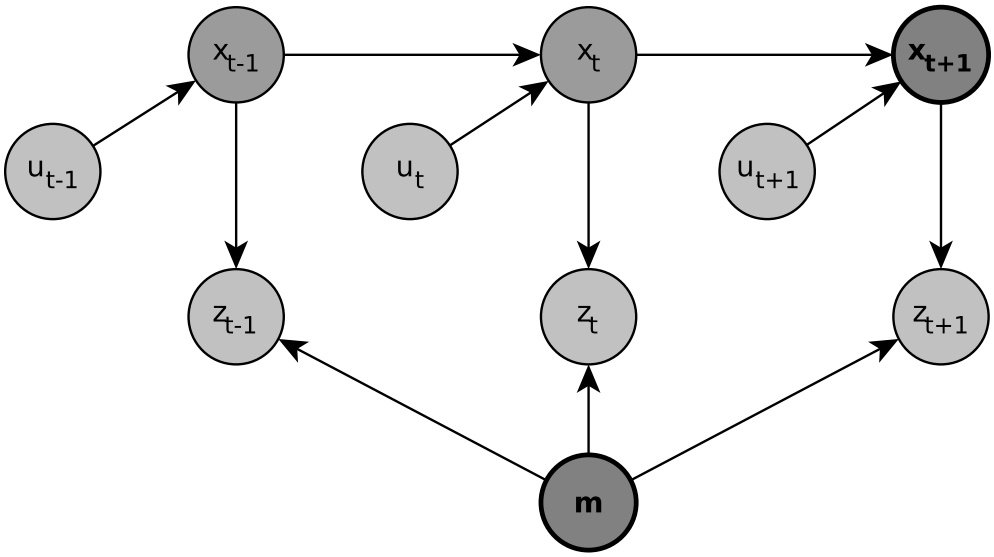


Fig. 3.6.: Calculation of the next robot pose x_{t+1} and map m from odometry u_t and sensor measurements z_t for online SLAM.

first to find the matching landmark. To simplify the process here, the correspondence of measurement and landmark is assumed known and therefore allows a distinct match.

The state vector μ consists of the mean values of the Gaussian distribution for the robot's pose $x_t = (x, y, \theta)^T$, the landmarks' positions and their signatures. The state vector μ has a size of $3 \cdot (i + 1)$ where i is the number of landmarks. The matrix $\mathbf{\Sigma}$ includes the covariance of these distributions.

Algorithm 3.5 begins by estimating the new mean value for the robot's pose $\bar{\mu}_t$ using the function G with the current odometry u_t and its previous pose x_{t-1}. The estimated covariance $\bar{\mathbf{\Sigma}}_t$ is calculated using the Jacobian matrix $\mathbf{J}$ of function G combined with the previous covariance $\mathbf{\Sigma}_{t-1}$ and the covariance of the odometry measurement $\mathbf{R}_t$. The mean values and covariance of the landmarks remain unchanged during these calculations.

In the next step, the sensor measurements z_t^i are used to build the Kalman matrix $\mathbf{K}_t^i$ for each landmark. If a measurement includes a previously unobserved landmark, its position is derived from the measurement and added to $\bar{\mu}_t$. $\hat{z}_t^i$ is the estimation for the particular landmark and $\mathbf{H}_t^i$ the corresponding Jacobian matrix. The calculations of $\mathbf{K}_t^i$ and $\hat{z}_t^i$ are omitted for brevity.

μ_t and $\mathbf{\Sigma}_t$ are calculated based on the odometry and predictions for the new robot pose and corrected using the sensor's measurements. Because the Kalman matrix is

Algorithm 3.5: Extended Kalman Filter Simultaneous Localization and Mapping

Input: $\mu_{t-1}, \mathbf{\Sigma}_{t-1}, u_t, z_t$

1: $\bar{\mu}_t \leftarrow \mu_{t-1} + G(u_t, x_{t-1})$
2: $\bar{\mathbf{\Sigma}}_t \leftarrow \mathbf{J}_t \mathbf{\Sigma}_{t-1} \mathbf{J}_t^T + \mathbf{R}_t$
3: $\mu_t \leftarrow \bar{\mu}_t + \sum_i \mathbf{K}_t^i (z_t^i - \hat{z}_t^i)$
4: $\mathbf{\Sigma}_t \leftarrow (I - \sum_i \mathbf{K}_t^i \mathbf{H}_t^i) \bar{\mathbf{\Sigma}}_t$
5: **return** $\mu_t, \mathbf{\Sigma}_t$

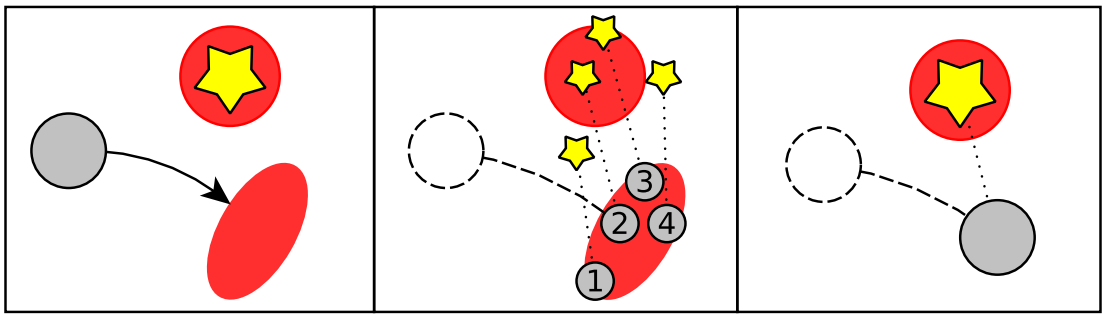


Fig. 3.7.: A simplified particle filter SLAM iteration is shown in three steps from left to right. The gray circle is the robot's position and the yellow star a landmark that can be observed by measurements. The uncertainty is shown as a red circle surrounding it. The red oval shows the probability distribution of the odometry.
After the robot moved in step two, multiple particles are sampled depicted as circles with numbers 1 to 4. The sensor measurement is applied to each particle and the derived landmark positions shown as small yellow stars are compared with the expected position. Step three shows, that particle 2 is chosen as the new robot position because of its measurement's correlation with the expected landmark position which is larger than the others.

densely populated, the observation of new landmarks leads to an improved position estimation of the robot and all other landmarks [146].

3.2.2. Particle Filter SLAM

Particle filters, which are also known as sequential Monte Carlo methods, can be employed for SLAM. They randomly sample particles and then evaluate each particle's probability. Every particle represents a possible robot trajectory and a corresponding map. Particles that seem unlikely to represent the real state of the robot are filtered out and discarded. Figure 3.7 depicts this process.

In the following, an exemplary particle filter using Rao-Blackwell's method is described. It was introduced by Giorgio Grisetti, Cyrill Stachniss and Wolfram Burgard [29, 30] as GMapping and is available as a ROS package, which is used for testing the exploration approach proposed in this work.

The probability for a possible robot trajectory $x_{1:t}$ and the corresponding map m is based on its sensor $z_{1:t}$ and odometry measurements $u_{0:t}$ as can be seen in Equation (3.5).

$$P(x_{1:t}, m|z_{1:t}, u_{0:t}) = P(m|x_{1:t}, z_{1:t})P(x_{1:t}|z_{1:t}, u_{0:t}) \tag{3.5}$$

A user-defined number of particles is sampled from the probability distribution of the odometry and they are used to build a procedural path and map for each particle. The following steps are repeated for this procedure [29]:

1. A new set of particles $X_t = \{x_t^0, x_t^1, ..., x_t^i\},\ i \in \mathbb{N}_0$ is sampled from the distribution function $\pi(x_t|z_{1:t}, u_{0:t})$ using the current set of particles X_{t-1}.

2. A weight w^i is assigned to each particle depending on the distribution function

π. Equation (3.6) shows how the weights are calculated.

$$w^{(i)} = \frac{P(x_t^i | z_{1:t}, u_{0:t})}{\pi(x_t^i | z_{1:t}, u_{0:t})} \tag{3.6}$$

3. Particles with a lower weight are discarded in favor of particles with a higher weight. This maintains the limited total number of particles. Low-weight particles can be seen as unlikely states based on the odometry and sensor measurements.

4. A map is calculated for every particle x_t^i based on the robot trajectory and the measurements incorporated in this particle $P(m_t^i | x_{1:t}^i, z_{1:t})$.

Grisetti et al. [30] formulated several improvements to the previously described algorithm. To prevent recalculating new particle weights in each iteration, a method to derive them from the last iteration is introduced. Furthermore, resampling particles is not executed every iteration but only when the criteria formulated in Equation (3.7) is met.

$$n_{\mathit{eff}} = \frac{1}{\sum_{i=1}^{n} (w^i)^2} \tag{3.7}$$

The variance of particle weights n_{eff} is used to determine the degradation of the current particles' probabilities and is derived from the normalized weight of all n particles. If n_{eff} falls below a certain threshold, the particles are resampled. The authors mention that when the robot enters known areas which is called a loop closure, n_{eff} drops significantly and causes a resampling because some particles are validated while others fail [30].

A further improvement is deriving the sampling area from a distribution function based on matching sensor measurements. Using a lidar is more precise than odometry. If the scan matching does not yield viable results, the odometry is used as a fallback option [30].

3.2.3. Graph-Based SLAM

Graph-based SLAM builds a global graph from the robot's positions, its odometry and sensor measurements. The graph's nodes show a discrete history of the robot's positions and edges represent odometry and sensor measurements connecting them. This is shown in Figure 3.8. The robot's position is stored as a node after traveling a certain distance or time. Because of imprecise measurements, edges should be represented with a probability for each connection to other nodes. To simplify the process, only the most probable connection is chosen. Constructing the nodes and edges is done by the frontend of graph-based SLAM.

This simplification causes deviations and errors in the graph resulting in a suboptimal map. A backend is introduced which tries to optimize the graph created by the frontend. It compares the robot's position with the Gaussian distribution based on odometry and sensor measurements and tries to find the best match regarding the graph. This maximization problem is solved using Gauss-Newton on the sparse matrix derived from the graph. It is sparse because nodes are mostly connected with consecutive nodes and seldom with other nodes through loop closures [147].

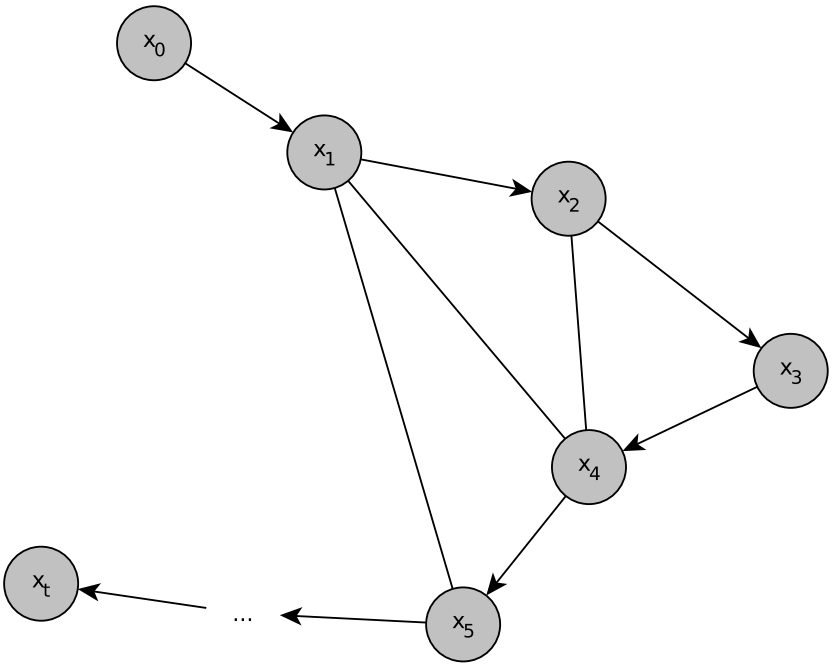


Fig. 3.8.: A graph which stores the robot positions x_0 to x_t is shown. The directed edges represent odometry measurements between two particular positions while undirected edges show sensor measurements indicating loop closures.

3.3. Exploration of Unknown Environments

To explore an unknown environment with a mobile robot, it must be categorized into distinct fractions which are introduced first. This is followed by a strategy to explore this environment and map representations utilized in this and related work which are able to store and query details of the environment.

3.3.1. Classification of the Exploration Space

Autonomous exploration in 3D is intended to map a volume V which is partitioned into unknown V_{un}, free V_{free} and occupied V_{oc} space as can be seen in Equation (3.8). A robot is utilized to classify the initially unknown environment.

$$V = V_{un} \cup V_{free} \cup V_{oc} \in \mathbb{R}^3 \tag{3.8}$$

$$V_{ex} = V \setminus V_{nx} \tag{3.9}$$

The explorable space V_{ex} is restricted by the robot's capabilities as well as occlusions caused by obstacles and the environment's topology as shown in Equation (3.9). The non-explorable space V_{nx} is defined by these occlusions, topology and the robot's inability to reach certain areas of space, e.g., UGVs versus UAVs or the robot's size.

3.3.2. Next-Best Views

The robot commonly has a limited movement and sensor range to observe the environment which necessitates an efficient strategy to classify V_{ex}. Connolly introduced

the term Next-Best View in 1985 [134] to describe the process of sampling multiple viewpoints and selecting the best one regarding the maximization of an object's observed volume. In this work, the term NBV is used to describe the next selected robot position to maximize an exploration metric.

To compare different viewpoints, their gain and cost are calculated and compared to decide which viewpoint is the NBV. Different metrics can be employed to derive gain and cost functions in 2D and 3D environments. These metrics are addressed as the reward function in the remainder of this work.

The gain can be calculated by ray tracing from the sampled viewpoints to calculate an estimated gain which is equal to the expected amount of observed, previously unknown space from the particular viewpoint as proposed by Connolly. Another metric is the identification of frontiers and their size. Frontiers are the borders between known and unknown space in occupancy grid maps and were introduced by Yamauchi [63].

Cost metrics often depend on the distance from the robot to the viewpoint [6], the time to travel to the viewpoint, the deviation from a straight path and the safety or traversability of the path to the viewpoint [105].

3.3.3. OctoMap

The OctoMap was introduced by Hornung et al. in 2013 [32] and is a tool to efficiently store and query 3D occupancy grid maps. It is based on octrees which are a tree structure where each node has exactly eight children. Starting at the root node, which is a voxel that encompasses the complete space, every child node has eight children that represent an octant of the parent's space. The leaf nodes have an edge length that equals the resolution of the OctoMap. Every leaf node is classified as occupied, free or unknown by a probability estimation based on sensor measurements. This structure can be seen in Figure 3.9

The leaf nodes store a probability value which classifies them as free or occupied depending on user-defined thresholds. The probability P for a node n is calculated using a binary Bayes filter which integrates previous measurements $z_{1:t-1}$ and the new measurement z_t which can be seen in Equation (3.10). The probability changes when

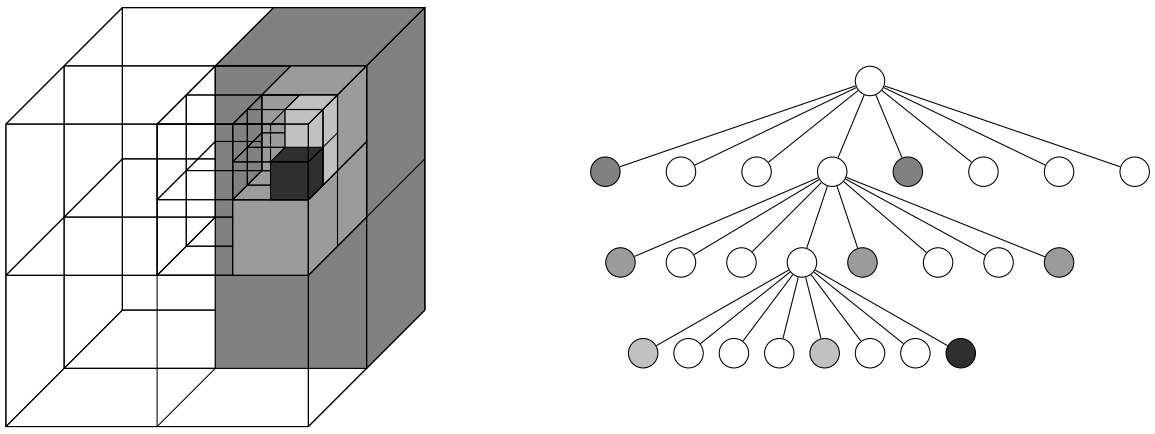


Fig. 3.9.: A volumetric octree model is shown on the left and the corresponding octree structure on the right. The black cube represents occupied space and shaded gray cubes indicate free space. These are shaded in different scales based on the particular node's level in the octree. The higher the level, the darker the shade. White leaf nodes and the corresponding space are unknown.

sensor measurements are integrated as shown in Equation (3.11) which employs an efficient log-odds notation. Using ray tracing, all nodes between the sensor's position and a measured obstacle have their probability to be occupied reduced while the node at the end of the ray has it increased. Multiple measurements decrease the uncertainty which allows pruning of leaf nodes if all eight children of one node have the same classification [32].

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{3.10}$$

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \tag{3.11}$$

The OctoMap also allows effective queries for positions and rays which make it the mapping tool of choice for multiple exploration approaches that were discussed in the previous chapter [6, 7, 10, 18] and for this work as well.

3.3.4. TSDF and ESDF Mapping

Approaches to represent the mapped space, that recently became popular, are Truncated Signed Distance Fields (TSDF) and Euclidean Signed Distance Fields (ESDF) maps. They are based on signed distance functions which determine the metric distance of an arbitrary point in space to a boundary. Curless and Levoy [148] proposed to reconstruct the surface of an object from sensor measurements using signed distance functions.

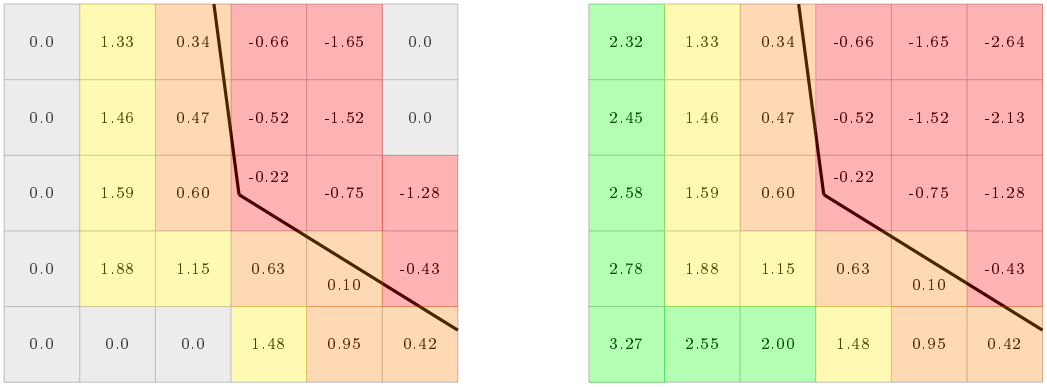
Oleynikova et al. [149] introduced Voxblox in 2017 which builds a TSDF and derives an ESDF from it. It is utilized in several state-of-the-art approaches discussed in the previous chapter [8, 9, 105] and is detailed in the following.

First, a TSDF is stored as a voxel grid with a predefined edge length v and is constructed using sensor measurements. For each point in the sensor measurement, a ray is cast from the sensor to the observed obstacle. All voxels on the ray with a distance below the truncation distance δ to this obstacle are updated. This includes voxels in front of and behind the detected obstacle.

The distance value assigned to each voxel describes the distance from the particular voxel center to the nearest detected obstacle inside the truncation range. Above this range the distance value equals zero. Distance values are positive if the voxel center is closer to the sensor than the obstacle for the respective measurement. A negative distance value shows that the voxel center is occluded by the obstacle. An exemplary 2D TSDF map can be seen in Figure 3.10a.

Each voxel stores a distance and a weight value where the distance value is a combination of all weighted distance measurements incorporated in the particular voxel. The weight is determined by the distances of previous measurements from the sensor to the obstacle. The further away the object is, the lower the weight of a particular measurement which represents the increasing inaccuracy for longer ranges. The weighting is required to average all measurements that fall into a certain voxel and increase the reliability of the representation [149].

TSDF can be utilized to create a 3D mesh with sub-voxel accuracy for visualization purposes using the voxels' distance values in a marching cubes algorithm [148]. Furthermore, an ESDF map is created from the TSDF map. The ESDF map stores



(a) TSDF **(b)** ESDF

Fig. 3.10.: In (a), a 2D TSDF grid map can be seen after several measurements with a distance sensor. The map has 1 m cell edge length and a truncation distance $\delta = 2$ m. The bold black lines show an obstacle and the numbers in each cell the distance from the respective grid cell's center to the nearest observed obstacle. Negative values indicate that the cell's center is behind the obstacle. Gray cells with a distance of 0 m are truncated cells. (b) shows the same map as an ESDF grid map which has no truncation distance.

distance values similar to the TSDF map but without a truncation distance. A distance to the nearest obstacle is known at every initialized voxel in the map. This difference is visualized in Figure 3.10b.

Voxblox adapts an algorithm introduced by Lau et al. [150] to efficiently update the ESDF map when obstacles are inserted or removed. Two different wavefront expansions are employed, one to raise and one to lower the distances in the cells. The former is called when an obstacle is removed and the distances in surrounding voxels must be invalidated and the latter when an obstacle is added and distances must be reduced. The wavefront expansions are stopped once they reach cells that are closer to another obstacle. After a raise wavefront, a lower wavefront is applied to all previously invalidated voxels and sets their distances to the new nearest obstacle [149].

3.4. Shortest Possible Route through a Graph

The partition of exploration approaches into local and global became popular in more recent work [9, 12, 14, 19, 20]. It often introduces a global graph connecting places that have not been explored but should be visited. This requires a method to find the most efficient way to visit all of the global nodes which is known as the Travelling Salesman Problem (TSP).

It describes finding the shortest route through a weighted, undirected graph that visits all of the graph's nodes exactly once and begins and starts at the same node. The term TSP was introduced by Julia Robinson in 1949 [151] but the problem has been researched even earlier. It is proven to be an NP-hard problem and therefore, multiple heuristics were proposed to deliver a good result in a feasible time compared to a brute-force approach. Heuristics are separated into tour construction and tour optimization. Held and Karp [152] formulated a lower bound which is often used to compare the heuristic's results for TSP. A selection of heuristics is described below with a focus on the complexity and Held-Karp lower bound, followed by several variations of TSP relevant to this work.

3.4.1. Tour Construction

Tour construction heuristics are employed to find an initial, viable path through all nodes n and stop afterwards without trying to further improve the path. These heuristics include a nearest neighbor search where the nearest unvisited node is selected for each node as the next node in the path. This yields a complexity of $\mathcal{O}(n^2)$ while being within 25% of the Held-Karp lower bound. Another one is the greedy heuristic which sorts all edges by length and adds them to the tour in this order with a complexity of $\mathcal{O}(n^2 \log_2(n))$. It is within 15% to 20% of the Held-Karp lower bound. There are insertion heuristics that start with a subset of nodes and insert new nodes similar to the previous two algorithms which result in the same complexities.

The Christofides-Serdyukov algorithm has a complexity of $\mathcal{O}(n^3)$ but comes within 10% of the Held-Karp lower bound by using a minimal spanning tree and minimum-weight perfect matching to build an Eulerian circuit. This is a path through a graph that traverses each edge only once and starts and ends at the same node. From this Eulerian circuit, a Hamiltonian circuit is constructed which resembles a path formulated with TSP [153].

3.4.2. Tour Optimization

Once an initial path is found, it can be optimized using one of the following heuristics. A simple approach is 2-opt as introduced by Croes [154] which iterates over all nodes in the path, selects two of them and reverses the order of nodes and edges in between. If the new order reduces the path length, the algorithm continues. If a complete iteration through all nodes does not yield any optimization, 2-opt is finished. Figure 3.11 shows an exemplary 2-opt swap removing a path crossing itself. 2-opt has a complexity of $\mathcal{O}(n^2)$ and comes within 5% of the Held-Karp lower bound [153].

3-opt is another heuristic that functions similarly to 2-opt but it takes three nodes and tries all possible combinations to reconnect them with each other. This leads to a complexity of $\mathcal{O}(n^3)$ and a result within 3% of the Held-Karp lower bound [153].

This can be extended to k-opt where k is an arbitrary number. Furthermore, the number of nodes to swap can be dynamically adjusted as proposed by Lin and Kernighan [155]. At each iteration, the Lin-Kernighan algorithm decides which number is most beneficial. This brings it within 2% of the Held-Karp lower bound with a complexity of $\mathcal{O}(n^{2.2})$ [153].

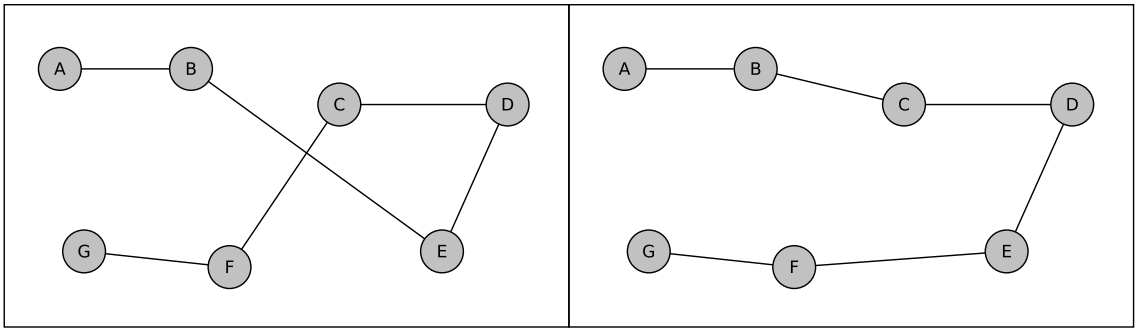


Fig. 3.11.: The images show a 2-opt swap of the paths between nodes B, C, E and F to remove a crossing. On the left, the previous path can be seen. On the right, the crossing is removed.

3.4.3. Open and Generalized TSP

While the original TSP requests a path that leads back to the starting node, this is not needed in certain applications. For them, open TSP can be applied which does not require a path that returns to the starting node. This can easily be incorporated into the heuristics by adding fake directed edges from every other node to the start node with a distance of zero.

Another variation is the Generalized TSP introduced by Noon and Bean [156] which separates the graph into disjoint sets of nodes and tries to find the shortest path visiting only one node of each subset. The distance between nodes inside a subset is disregarded and only the edges between nodes from different subsets are known.

3.5. Conclusion

This chapter explains the foundations required for understanding the proposed approach and relevant related work. The sampling-based algorithms RRT, RRG and PRM are detailed from which this work's exploration algorithm is derived. Complementary to these algorithms, Voronoi diagrams, k-d trees and Dijkstra's algorithm are shown because they are utilized in this work.

The principles of SLAM are introduced as the robot's position in the world and the map surrounding it must be known for an exploration. EKF SLAM is described as the standard algorithm while particle filter SLAM is more sophisticated and is the algorithm used to test the proposed exploration approach RNE in Sections 5.2, 6.2 and 6.3.

To explore unknown environments and store the progress, they must be classified and mapped. A distinction between free, occupied and unknown space is formulated and two mapping frameworks are introduced. These are called OctoMap and Voxblox and are able to store this information. They can be used to effectively retrieve the occupancy of certain areas using single voxel or ray queries.

These queries are utilized to formulate traversability information and information gain for exploring robots. The term NBV is explained which describes choosing the next viewpoint for the robot that maximizes a reward function incorporating information gain and costs to reach the particular viewpoint. The mapping and NBV selection are utilized in this work and shown in Chapter 5.

The TSP problem is introduced which describes finding the shortest path connecting all nodes in a graph while only visiting each of them once. In addition, several heuristics to solve it within a certain lower bound regarding the path length are shown. In Chapter 8, standard and open TSP are deployed to efficiently connect multiple viewpoints on a global scale.

4. Robot Statemachine

To alleviate the autonomous exploration proposed in the next chapter, a state machine is developed. It is used as an interface to the robot's path planning and motion control and decouples these from the exploration which enables utilizing different robots and path planning approaches.

The state machine described in this chapter is called Robot Statemachine (RSM). It offers an aided control for arbitrary mobile robots and can be used by simple GUI commands. RSM enables sending the robot to explore or follow given waypoints autonomously and execute special routines when reaching a goal. These routines can be implemented through plugins and range from informing the operator to carry out a routine to autonomously sweeping a camera and identifying certain objects to be mapped. It can be adapted for different mobile robots and tasks through the implementation of plugins for navigation, exploration and inspection tasks.

RSM includes functions intended for the EIT RawMaterials UNDROMEDA project described in Section 1.1.1. The abilities to perform inspections and patrol tasks which are detailed in this chapter are a part of these functions.

The first section of this chapter describes the design in detail regarding the underlying states and the adaptability gained by the plugins as well as the GUI. Afterwards, applications of RSM are shown.

4.1. Design

RSM is written in C++ and follows the UML state pattern [33]. It is a behavioral state machine [35] which means that actions occur during state transitions and while a state is active.

RSM is implemented as a ROS metapackage which is described in Section 2.1.1. Included in this metapackage is the package rsm_core that holds the state machine's core and its integrated states. The rsm_additions package implements several exemplary plugin states and helper functions. The RSM's GUI is realized by the rsm_rqt_plugins and rsm_rviz_plugins packages.

4.1.1. States and Connections

The class responsible for state transitions only allows volatile states which makes an additional class for handling data necessary. All implemented states need to inherit from a virtual base state whose class diagram is shown in Figure 4.1 together with the surrounding structures. Each state's base functions are called by the class handling state transitions. To obtain data necessary for processing, each state has to interact with the data handler class through ROS services and topic subscriptions.

RSM features some integrated states offering basic functionalities which are listed below:

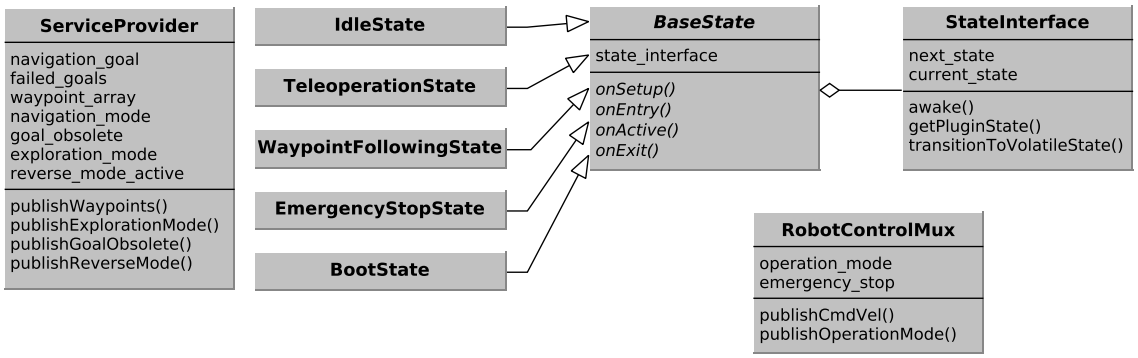


Fig. 4.1.: Partial class diagram of the RSM which shows the most important attributes and methods of each class.

- `BootState`
- `IdleState`
- `TeleoperationState`
- `EmergencyStopState`
- `WaypointFollowingState`

The `BootState` is always the first state to be called when the RSM is started and simply subscribes to a service that informs it about the boot-up process. When the process finishes, the `BootState` initiates a transition to the `IdleState`. The latter awaits input from the operator to transition to a desired state. One of these is the `TeleoperationState` which becomes active as soon as a teleoperation command is issued to the robot. The `EmergencyStopState` is activated when the operator pushes the software stop button in the GUI. A transition to the `IdleState` is only initiated when the button is released again. The `WaypointFollowingState` is called when waypoint following is started from the GUI. It manages the list of waypoints and forwards the next waypoint as a navigation goal.

To realize exploration and waypoint following, the three plugins listed below need to be implemented:

- `CalculateGoalState`
- `NavigationState`
- `MappingState`

The `CalculateGoalState` is responsible for extracting the NBV for exploration. Therefore, it has to interface a package calculating exploration goals. For example, the exploration approach presented in Chapter 5 can be utilized. The data handler class expects a list of possible exploration goals to evaluate if the current goal is still viable or has already been explored while moving which makes it obsolete. This is only used when the exploration is run in interrupting mode in which the navigation is aborted as soon as the current goal becomes obsolete. The extracted goal from

the `CalculateGoalState` is then forwarded to the `NavigationState` which needs to implement an interface to a designated navigation package. This interface has to be able to retrieve the current status of the navigation and move forward and in reverse. The `MappingState` can implement behaviors to rigorously scan the surrounding by moving the robot's sensors if necessary. The `NavigationState` is also used for reaching waypoints while waypoint following is active.

Waypoints can be linked to certain routines that are implemented as Routine States. These are called when the respective waypoint is reached and can execute arbitrary behaviors. Routine States are not mandatory to implement but there can be up to ten different routine states available.

An intended process for exploration starts with the `CalculateGoalState` forwarding a goal to the `NavigationState` which then tries to reach this goal. If it succeeds, the `MappingState` is called and transitions back to the `CalculateGoalState` to find a new

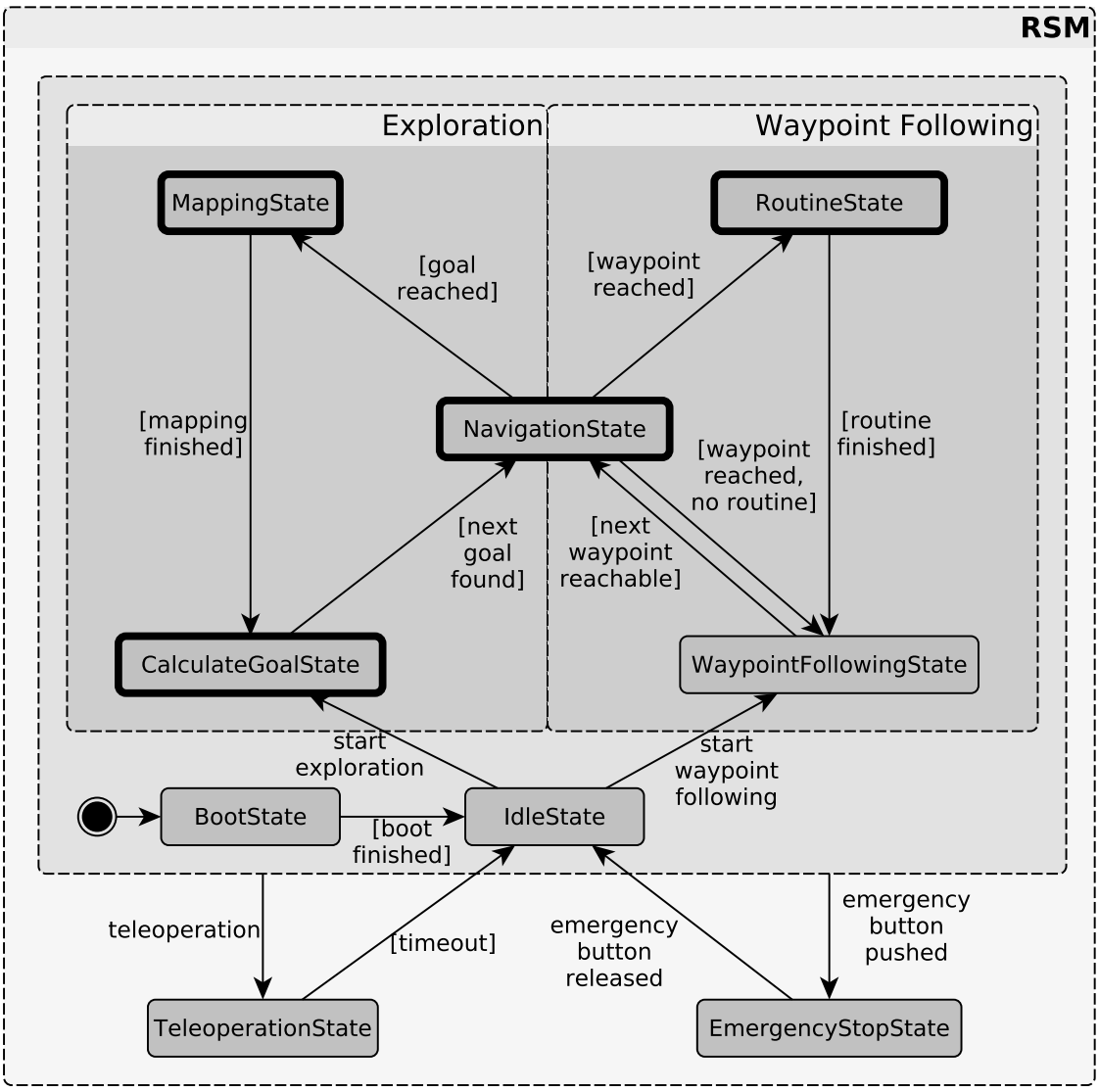


Fig. 4.2.: State diagram showing an exemplary implementation with normal states, additional plugin states with a bold border and transitions.

exploration goal after it is finished. If the navigation fails, the goal is blocklisted and the `CalculateGoalState` is called to find a new exploration goal. A state diagram showing these transitions can be seen in Figure 4.2.

Waypoint following works similarly. The `WaypointFollowingState` extracts a navigation goal from the list of waypoints and forwards it to the `NavigationState`. If the `NavigationState` succeeds in reaching the goal, the respective Routine State is called. It can execute an inspection task, e.g., reading a gauge. When the routine is finished or if there was no routine set for this waypoint, the `WaypointFollowingState` is called again to determine a new navigation goal. The respective waypoint's status is then set to visited. If the navigation fails, its status is set to unreachable and the `WaypointFollowingState` is called to provide the next navigation goal.

The RSM also features a class to control the command velocity output to the motor controller interface. It either maps the command velocity from teleoperation, from autonomous operation or sends a stop command to the motor controller interface. The command that is forwarded is set through the GUI. In the case of teleoperation, it can also be set automatically by issuing a teleoperation command. This only works when the software emergency stop is not pushed in the GUI. If it is, the stop command is sent to the motor controller interface and the robot does not move.

4.1.2. Sample Implementation of Plugin States

A sample implementation provided in the rsm_additions package features plugins for all of the previously described states. The `CalculateGoalState` interfaces the ROS package explore_lite[1] which is based on the FE algorithm introduced by Yamauchi [63] and described in Section 2.4.1. The plugin subscribes to explore_lite's visualization that shows frontiers on a 2D map and extracts the closest frontier center point to the robot as a navigation goal.

The `NavigationState` realizes an interface to the ROS navigation package. It forwards received goals to the navigation stack and gets feedback from it regarding the progress. It features a detection for when the robot is stuck for a certain amount of time. If it is stuck, it tries to move to the current goal backwards which resolves the blockade in case the robot's front is too close to an obstacle. After a brief amount of time, the robot returns to forward movement. If it is still stuck afterwards, the goal fails.

There are two implementations for the `MappingState`. The first one is a dummy that transitions back to the `CalculateGoalState`. It is intended to be used with sensors that continuously map the surroundings and do not need to execute a dedicated mapping routine, e.g., fixed cameras or rotating scanners.

The second one is designed for a simulation used for testing. This `MappingState` swivels a depth camera from one side to the other and back to create a dense 3D point cloud. A `RoutineState` called `ReversingRoutineState` is also included and toggles the reverse movement mode when the routine is executed. This means the robot is driving in reverse when it was going forward before and vice versa. An additional data handler class is provided that adds services to interface the explore_lite and navigation packages.

[1] `http://wiki.ros.org/explore_lite`

4.1.3. GUI

The RSM can be operated through a GUI that enables the use of all its core functionalities. The GUI is depicted in Figure 4.3 and can be integrated into RViz or rqt which can be used in ROS for visualization. It always shows which state is currently active and gives the user the options explained below.

The GUI offers control over the class handling the command velocities forwarded to the motor controller interface. This includes the software emergency stop as well as choosing autonomy, teleoperation or stopped. When the software emergency stop is active, the other choices are disabled and the command velocity is set to stopped until the software emergency stop is released again.

The exploration can be started and stopped by using the respective buttons in the GUI where the exploration mode can be set as well. This mode can either be *finish* or *interrupt* where the former makes the robot reach each goal before transitioning to the `MappingState` while the latter starts the transition as soon as a better goal becomes available.

Waypoint following can be started and stopped through the respective buttons. When waypoint following is stopped, there is the option to reset the current progress and restore all waypoints to their initial values. It is possible to set the waypoint following mode to one of the following three options: single, roundtrip and patrol.

The single mode lets the robot start from the first waypoint and then moves to all consecutive ones in order. Upon reaching the last one it stops. In roundtrip mode, after reaching the last waypoint, all waypoints are reset and it starts anew from the first waypoint. This is repeated until manually stopped. Patrol mode works similarly. After reaching the last waypoint, all waypoints are reset and it starts again in reverse

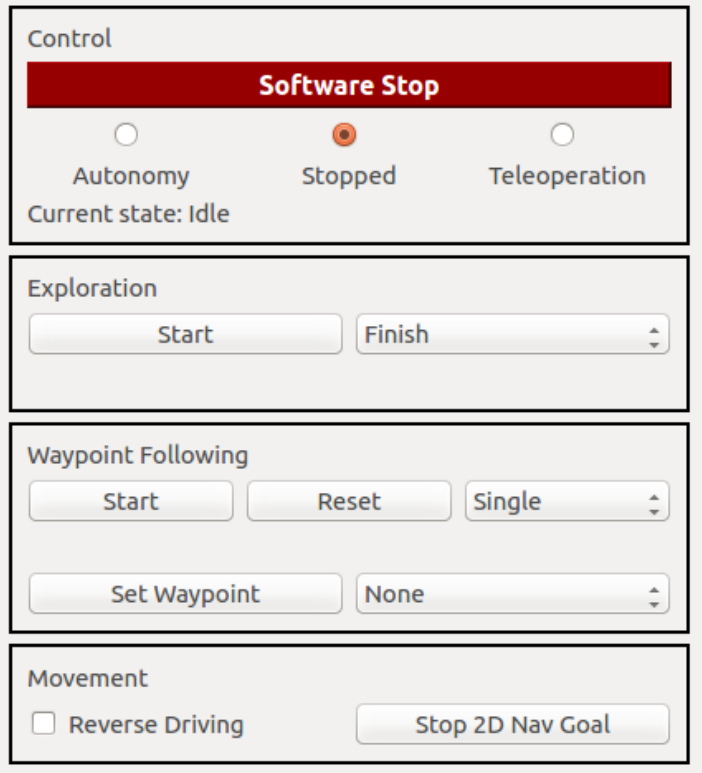


Fig. 4.3.: GUI used to control the RSM from RViz or rqt.

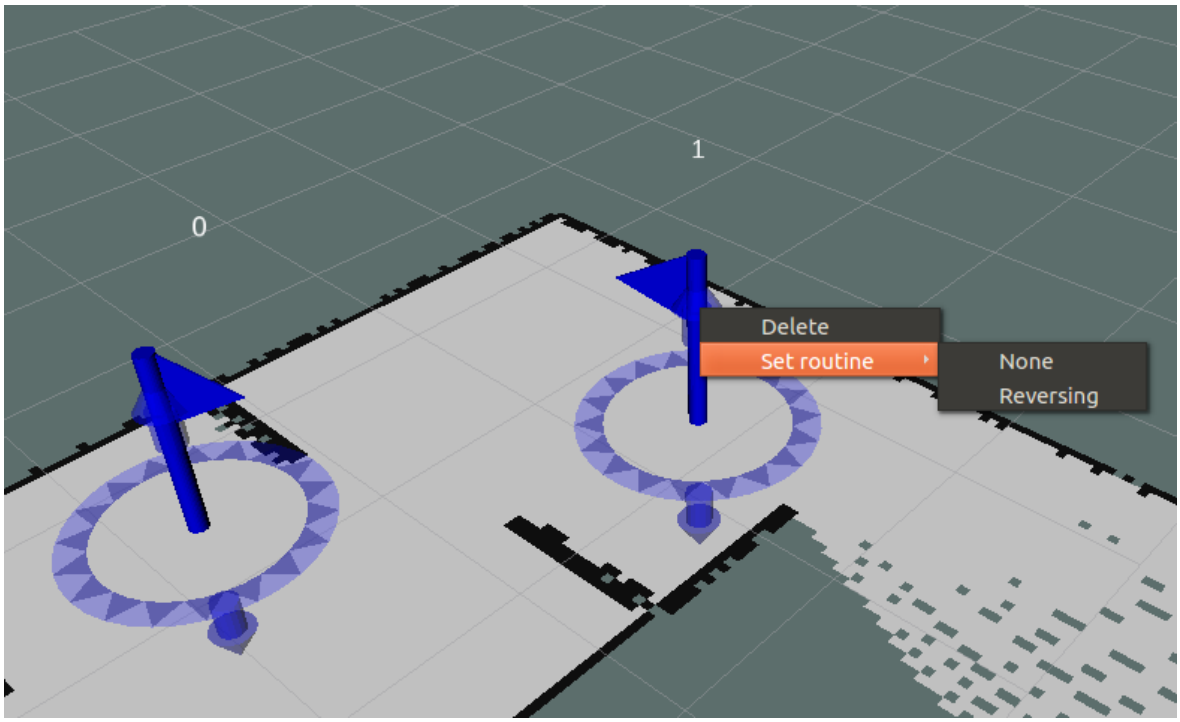


Fig. 4.4.: Waypoints as interactive markers in RViz that can be manipulated by dragging and their routines changed with the attached menu.

order. The first and last waypoints are only targeted once and their attached routines are executed only once as well. Roundtrip and patrol can only be stopped manually.

The GUI also offers the possibility to set a waypoint at the robot's current pose with a desired routine. A checkbox enables setting the reverse mode manually. When the box is checked, the robot moves in reverse.

When using RViz, waypoints can be placed in the map by utilizing the Set Waypoint Tool. The color of the marker corresponds with the waypoint's status. Blue is the default color, green means the waypoint has been visited and red that it is unreachable.

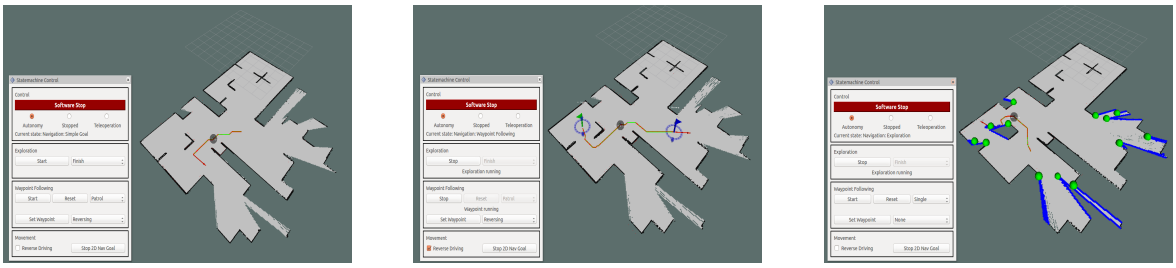
The displayed markers are interactive and can be seen in Figure 4.4. Their position and orientation can be changed. Furthermore, they offer a menu where the routine to be executed when reaching the waypoint can be set or the waypoint can be deleted.

Simple navigation goals can be sent in RViz using the 2D Nav Goal Tool. The execution of these goals can be interrupted with the respective stop button in the GUI.

4.2. Applications

RSM is designed to facilitate and supervise autonomous behavior for a wide variety of robots. Besides the usage in simulated tests and exploration experiments, it was utilized in the UNDROMEDA project and in the RoboCup RRL which are explained in Sections 1.1.1 and 1.1.4. Figure 4.5 shows RSM deployed in a simulation, demonstrating simple navigation, waypoint following and exploration with FE.

RSM was deployed on the robot Schrödi from the Nuremberg Institute of Technology's AutonOHM team for the RoboCup German Open in 2019 and the RoboCup in Sydney, Australia in 2019. Schrödi is shown in Figure 1.2. RSM was used for autonomous repetitions in some of the mobility challenges of the RRL and during the



(a) Simple goal navigation **(b)** Waypoint following **(c)** Exploration

Fig. 4.5.: RSM is used in simulations for simple goal navigation, waypoint following and exploration. Light gray areas show free space, dark gray areas unknown space and black lines obstacles. The translucent sphere marks the robot's position and green and red lines the robot's current navigation path. The blue lines in the exploration show the frontier boundaries and green spheres are the frontier markers to which the robot navigates when exploring the specific frontier.

exploration challenges. Team AutonOHM won first place in the German Open and scored fourth place in Sydney.

For UNDROMEDA, RSM was deployed on the Clearpath Robotics Husky UGV and used to supervise the robot during test runs. In the final test run, the odometry was not working correctly. Therefore, the navigation could not be used as it relies on precise odometry. Unfortunately, this ruled out the usage of RSM's autonomy functions in UNDROMEDA's final test run.

4.3. Conclusion

The proposed RSM enables a high-level control of a robot and can be utilized to supervise an autonomous exploration which is used in the upcoming chapters. It separates the robot's motion control and path planning from the exploration approach which allows to exchange the former two if desired.

RSM offers interfaces which enable aborting currently pursued goals for better ones and grant more resilience for the navigation planner. The ability to abort the current goal for a better one is required for the exploration approach detailed in the next chapter.

If necessary, mapping routines can be implemented, e.g., move a sensor head to increase the map coverage without moving the whole robot. The GUI provided by RSM aids in controlling the autonomous exploration and increases usability.

In conclusion, the RSM serves as an easy-to-use state machine, especially for the development of exploration approaches and during simulations throughout this work. The introduced plugin states enable rapid changes between different approaches, for example, FE introduced in this chapter and the sampling-based approach which is presented in the next chapter.

5. RNE 1 - RRT-Based Exploration

This chapter introduces the first iteration of the Random-Sampling-Based Next-Best View Exploration (RNE) which is the main contribution of this work. RNE enables the exploration of different environments with a UGV and is based on RRT. RRT was proposed by LaValle [50] and is described in detail in Section 3.1.2. RNE evaluates the randomly-sampled nodes in RRT to find an NBV which then is explored by the robot. NBV is explained in Section 3.3.2.

This approach is heavily inspired by RH-NBVP that was introduced by Bircher et al. in 2016 [6]. They build an RRT, evaluate the nodes and select an NBV from the best branch for a UAV to explore. After each iteration, the RRT is rebuilt except for the best branch. The proposed RNE is designed for UGVs and utilizes a permanent RRT to improve exploration efficiency compared to RH-NBVP. Furthermore, the gain calculation is decoupled from the exploration thread which means it is conducted while the robot explores instead of after reaching a goal.

AEP proposed by Selin et al. [7] utilizes RH-NBVP and adds a global exploration mechanism which stores former RRT nodes that have not been explored by the robot but yield a sufficient information gain. Furthermore, they introduce Sparse Ray Casting (SRC) for gain calculation which is the main reference for this approach's Sparse Ray Polling (SRP). SRP's goal is to reduce the computation for calculating an NBV.

The following sections go into detail about the design and ideas of RNE which include SRP, a decoupled gain calculation and the traversability analysis incorporated in the RRT's input function. This is the first iteration of the proposed exploration approach while the following chapters continue with the evolution of RNE to further increase its efficiency.

5.1. Design

This section introduces the design of RNE and highlights certain parts that differ from RH-NBVP and AEP which inspired its development.

The first subsection describes how the RRT principle is used for this exploration algorithm, followed by the general software implementation of RNE and its interface to RSM, which is presented in Chapter 4. Afterwards, the method used to classify the traversability of the grid map and particular nodes is explained.

This work's adaption of SRC is called SRP and is the designated method to calculate the potential gain of a new node. It is presented in the next subsection followed by the comparison through experiments between adaption and simple ray casting.

Finally, the differences when using coupled or decoupled gain calculation are exposed by experimental test runs. The coupling describes if the gain is calculated in the same process as the remainder of RNE's logic or in a separate thread.

5.1.1. RRT Adaption

The basic RRT algorithm, which is explained in Section 3.1.2, cannot be used for exploration directly because it is intended to connect a start and a goal configuration. Since there is no designated goal at the start of an exploration, the algorithm must be adapted.

Algorithm 5.1 shows how the RRT for exploration is constructed. The world space $V \in \mathbb{R}^3$ as well as the robot's current position in the world space $\mathbf{p}_{robot} \in V$ are provided. At first, the tree $\mathcal{T} = (N, E)$ with the set of nodes N and edges E is initialized with its root node at the robot's current position.

As long as the exploration is not finished, which is determined by the criteria explained in Section 5.1.8, the following statements are executed. The loop starts with randomly sampling a position $\mathbf{p}_{rand}$ in the known free space V_{free}. Afterwards, the node in $\mathcal{T}$ closest to $\mathbf{p}_{rand}$ is determined and saved as $\mathbf{p}_{near}$.

The steerCollisionFree function combines the steer and collisionFree functions from Section 3.1.2. It tries to add a new node between the nearest tree node's position $\mathbf{p}_{near}$ and $\mathbf{p}_{rand}$ to which the robot is able to move without collision.

As shown in Section 3.1.2, the position of the new node $\mathbf{p}_{new}$ is calculated using the robot's input function. This results in a straight line with a minimum d_{min} and maximum distance d_{max} between $\mathbf{p}_{new}$ and $\mathbf{p}_{near}$.

If a connection without obstacles between $\mathbf{p}_{near}$ and $\mathbf{p}_{new}$ can be found, $\mathbf{p}_{new}$ is created and added to the tree. This includes an edge between it and $\mathbf{p}_{near}$. The input function used in this work and the collision check are described in Section 5.1.4.

This tree growing process runs continuously during the exploration and builds $\mathcal{T}$ while its nodes are visited by the robot. A continuously growing tree is chosen over repeatedly building a new tree after reaching each goal like in RH-NBVP. Repeatedly building a tree must be limited by a time factor or the number of added nodes to decide if the exploration is finished. This means, the robot could be led to a position from which the limiting factor terminates the exploration even though there is still a substantial fraction of V_{ex} to be mapped. V_{ex} is the explorable space introduced in Section 3.3.1. The continuous tree is used to increase total map coverage by preventing the termination of a run when encountering a local dead-end.

Algorithm 5.1: Rapidly-exploring Random Tree construction for exploration

Input: $V, \mathbf{p}_{robot}$

1: $\mathcal{T} \leftarrow$initTree($\mathbf{p}_{robot}$)
2: **while not** explorationFinished() **do**
3: $\quad \mathbf{p}_{rand} \leftarrow$samplePoint($V$)
4: $\quad \mathbf{p}_{near} \leftarrow$ findNearestNeighbour($\mathcal{T}, \mathbf{p}_{rand}$)
5: $\quad$ **if** $\mathbf{p}_{new} \leftarrow$steerCollisionFree($V, \mathbf{p}_{rand}, \mathbf{p}_{near}$) **then**
6: $\quad\quad$ addNodeToTree($\mathcal{T}, \mathbf{p}_{new}, \mathbf{p}_{near}$)
7: $\quad$ **end if**
8: **end while**

5.1.2. Software Design

RNE is developed as an open-source ROS package that can be used for exploration with the RSM package. The robotics framework ROS is described in Section 2.1.1. RNE includes a state plugin for RSM, which interfaces the exploration algorithm, and a global planner that can be used as a plugin for the ROS navigation package. Details regarding the interaction between RNE and RSM can be found in Section 5.1.3.

In Figure 5.1, RNE is shown in the ROS environment controlling the robot's movement by forwarding goals to the ROS navigation package and processing the sensor input. A SLAM algorithm, which is detailed in Section 3.2, is executed on the robot. It calculates the robot's position in the surrounding space and builds a grid map in which traversable tiles and obstacles are marked. The point cloud produced by the robot's sensors is integrated into an OctoMap based on the robot's position. The OctoMap is described in Section 3.3.3.

The algorithm stores all relevant RRT details in a ROS message. It contains a list of all nodes and a reference to the node that currently is closest to the robot n_r. Each node n stores its coordinates $\mathbf{p}_n$, its gain $G(n)$, the orientation to achieve this gain φ_{max} and its state s_n. The state can be one of the seven states shown in Figure 5.2. Furthermore, it has references to its parent node including the distance to it and a list of references to all child nodes. The path P_{rn} from $\mathbf{p}_n$ to $\mathbf{p}_{n_r}$ through $\mathcal{T}$ as well as the length of this path d_{rn} is also stored for each node.

For each node added to $\mathcal{T}$, its gain is calculated using the OctoMap to assess the map coverage at the particular node's position. When the previous goal node is reached or aborted by the robot, a new goal is selected by choosing the node with the best reward function. The new goal is then forwarded to navigation.

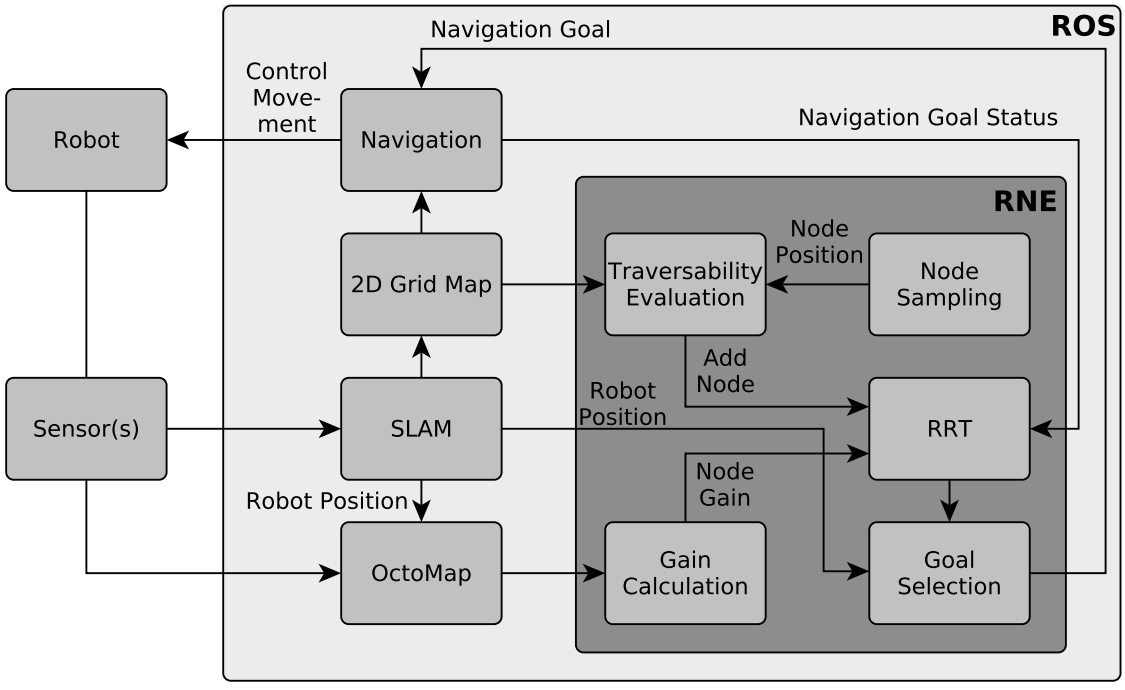


Fig. 5.1.: High-level overview of RNE's main functions interfacing other ROS packages as well as a robot and its sensor(s).

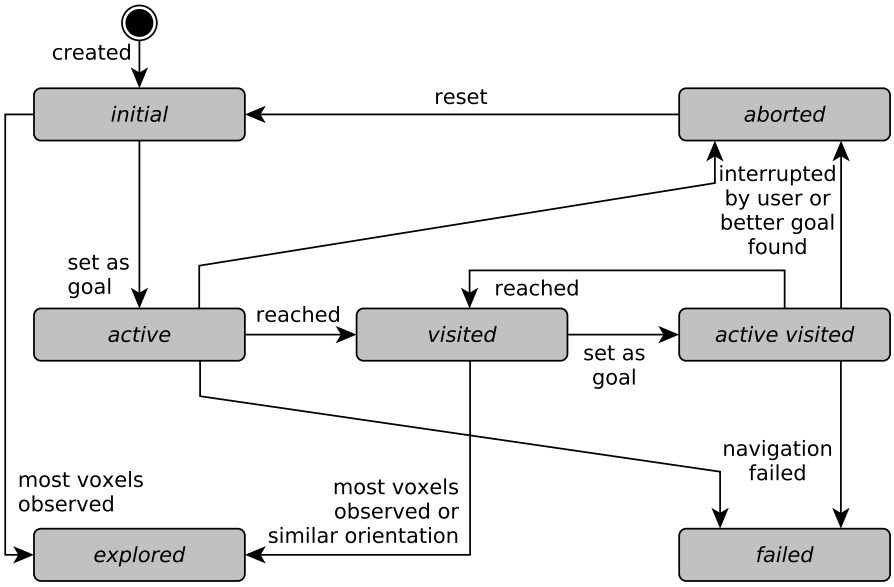


Fig. 5.2.: The state diagram of the different node states in RNE and the transitions between them.

The robot is required to have a front-facing sensor or array of sensors that outputs a combined point cloud with a known range and FoV. In addition, it must be able to perform SLAM with its sensors to enable an exploration.

When starting the exploration, $\mathcal{T}$ is initialized with a root node at $\mathbf{p}_{robot}$ with the state *visited*. All other nodes are initialized with the state *initial*. While the tree construction is running, a k-d tree to interface the nodes in $\mathcal{T}$ is maintained to enable efficient nearest neighbor and radius searches. k-d trees are described in Section 3.1.6. For every newly created node, its gain and cost are calculated. The k-d tree is implemented using nanoflann [157] which is a header-only library with a focus on run time and memory efficiency.

Furthermore, an ordered list containing references to the nodes is stored where the first entry references the node with the best reward function. This first node is chosen as the current exploration goal and its state is set to *active* or to *active visited*, if its state has been *visited* before. After reaching a node, its state is set to *visited*. If the current goal is reached, aborted or reaching it failed and the robot started to move towards it, the goal node's gain is recalculated. All nodes' gains in a radius less than two times the sensor's range around the robot are recalculated as well. The nodes in this radius are found by a radius search in the k-d tree.

The check if the robot has moved when the current goal changes, is undertaken to prevent recalculation of node gains when the robot did not move. Then, it is assumed that the map did not change which results in the expected node gains remaining the same. The radius of two times the sensor's range is chosen because it includes all nodes whose gain could have changed from the robot's current position. This is due to the robot exploring parts of the map that can also be perceived from the particular node's position.

When calculating the gain of a node with the *initial* or *visited* state and it is below a user-defined threshold, the node becomes *explored*. If it has been *visited* before, the orientation of the robot when exploring the node φ_{max} is stored and compared to the

recommended orientation from the new gain calculation. If they are similar, the node's state is set to *explored* because no further exploration of the map is expected. For a sensor with a limited horizontal field of view, multiple orientations at the same node can be worth exploring.

The following list summarizes the node states that can be seen in Figure 5.2:

initial The node has been placed and its gain is worth exploring or still unknown.

active The node is the current exploration goal.

visited The node has been visited by the robot and its gain is not yet recalculated or it still offers a sufficient gain at an orientation which differs from that of the previous visit.

active visited A previously visited node is the current exploration goal.

explored The gain, which is calculated from the number of unknown, observable voxels at this node, is below the threshold. Alternatively, a similar orientation from it is chosen as the next goal, even though it has already been visited with the same orientation before.

aborted The goal has previously been the current goal but a user command or the availability of a better goal has stopped its exploration.

failed The navigation planner failed to calculate a path towards this node.

Since RNE is built with an interface to RSM, it can also be run in two different modes which are *finish* and *interrupt*. In the former mode, an active goal node must be reached or the attempt to reach it has to fail before a new goal can be chosen. In the latter, an exploration goal can be interrupted if a better goal node is found. A better goal node is a node with a higher reward function than the current goal.

Such a node can only be found, if the reward functions change while the robot is busy navigating to the current goal. Calculating all available node's reward functions before choosing a node and then starting navigation would render the *interrupt* mode in RNE unnecessary. Therefore, RNE adds new nodes and calculates their reward function while pursuing a goal. Because most nodes' reward functions around a goal that has just been reached are recalculated, it might take some time before a valid assessment can be made about the NBV to use as the upcoming goal.

To reduce the downtime, this approach runs the computation intensive raycasting in a different thread than the RRT construction. This eliminates the need for the robot to wait after exploring a node because nearby node's gains are recomputed. It also means that not all node's gains are known when the robot starts to move towards a new goal. Therefore, the *interrupt* mode enables to switch the goal node if a better goal is found. This parallel process is called decoupling. An evaluation of decoupling can be found in Section 5.3.

5.1.3. RNE RSM Interface

RNE is designed to be operated together with RSM for which a plugin state and a data handler are added. The data handler is necessary because of RSM's volatile states which cannot store data.

The plugin state requests the current goal from RNE and sets it as the next navigation goal. Then, it triggers a transition to the navigation state.

The data handler receives the status of the ROS navigation package and forwards to RNE if a goal has been reached or reaching it failed. If the exploration mode is set to *interrupt*, it is checked if the current goal is obsolete. If it is, the data handler tells the navigation to abort it. For this, it compares if the current goal and the goal with the best reward function are the same. If they differ, the current goal is obsolete.

5.1.4. Grid Map Traversability Analysis

The steerCollisionFree function checks if there is a connection between $\mathbf{p}_{new}$ and $\mathbf{p}_{near}$ which is derived from the input function and can be traversed by the robot. Since the approach is developed for UGVs, a grid map derived from the explorable space V_{ex} is used to determine the traversability. An approach for UAVs or UGVs with manipulator arms would require a collision check in 3D instead of in a 2D grid map derived from the 3D space V. The implemented steerCollisionFree function assumes a non-holonomic robot which is able to turn on the spot.

A circular area around $\mathbf{p}_{new}$ with a radius r_{robot} is required which circumscribes the robot's footprint. A rectangular corridor between $\mathbf{p}_{new}$ and $\mathbf{p}_{near}$ with at least the robot's width w_{robot} must also be traversable. These shapes have to be translated to the grid map's discrete coordinates.

To be able to determine which grid map tiles have to be queried for assessing the traversability of an area, the structure of the grid map must be considered. The utilized grid map is an OccupancyGrid from the nav_msgs ROS package[1]. It stores meta data which includes the maps width w_m and height h_m measured in cell count and its resolution r_m in m/cell. Also, the origin of the map $\mathbf{o}_m = (x_o, y_o)^T$ in the ROS map frame consisting of x- and y-coordinates as well as the orientation are included.

The map contains a data array with every cell's occupancy probability which ranges from zero for a free cell m_{free} to 100 for an occupied cell m_{oc}. A data value of -1 indicates an unknown cell m_{un}.

Figure 5.3 shows the order of the grid map cells in the OccupancyGrid. Per default, the OccupancyGrid's width is in x-axis direction and the height in y-axis direction. The data consisting of $w_m \cdot h_m$ entries is organized in the array in a row-major order. Neighboring cells in x-axis direction have consecutive storage locations while neighboring cells in y-axis direction can be iterated over by skipping w_m entries for every step.

Furthermore, an exemplary point $\mathbf{p}$ is depicted in Figure 5.3 for which the conversion from world to grid map coordinates $\mathbf{p}^m$ is clarified in Equation (5.1). Coordinates in the grid map are denoted with a superscript m. Equation (5.2) shows the corresponding OccupancyGrid data index m_p for this point.

$$\mathbf{p} \in \mathbb{R}^3 = \begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} \Rightarrow \begin{pmatrix} \lfloor \frac{x_p - x_o}{r_m} \rceil \\ \lfloor \frac{y_p - y_o}{r_m} \rceil \end{pmatrix} = \begin{pmatrix} x_p^m \\ y_p^m \end{pmatrix} = \mathbf{p}^m \in \mathbb{N}_0^2 \tag{5.1}$$

$$m_p = y_p^m \cdot w_m + x_p^m \tag{5.2}$$

[1] http://docs.ros.org/en/melodic/api/nav_msgs/html/msg/OccupancyGrid.html

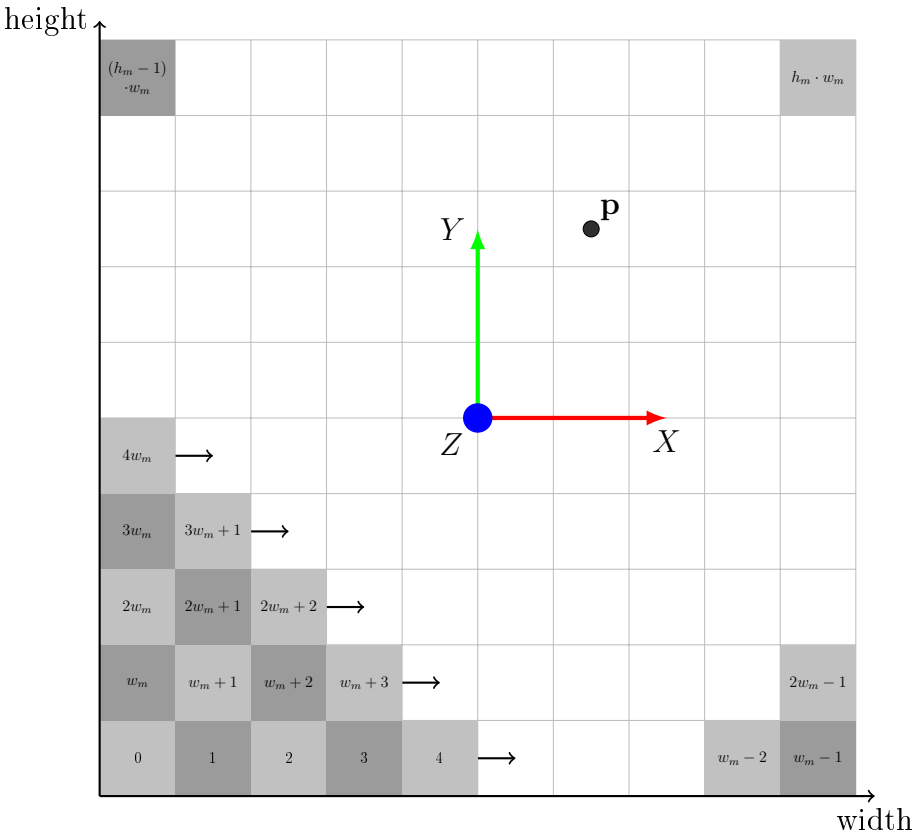


Fig. 5.3.: The row-major data order of the OccupancyGrid is shown in the differently shaded grid map cells. It starts at the grid map's origin $\mathbf{o}_m$ in the bottom left corner. The number inside a cell represents the particular data index of it which can be derived from the map's width w_m and height h_m. The relation to the axes of the world-fixed ROS map frame is depicted in the middle as well as the exemplary point $\mathbf{p}$ whose conversion from world coordinates to grid map coordinates can be seen in Equation (5.1).

To simplify the deduction of grid map cells to check for the circular area around the potential new node $\mathbf{p}_{new}$, it is aligned to the nearest grid map cell center as shown in Equation (5.3). This aligned node $\mathbf{p}_{alig}$ allows a symmetric check of grid map tiles for the circular area whose extent can be reused for every following traversability analysis as r_{robot} remains unchanged during an exploration.

$$\mathbf{p}_{new} \in \mathbb{R}^3 = \begin{pmatrix} x_{new} \\ y_{new} \\ z_{new} \end{pmatrix} \Rightarrow \begin{pmatrix} \left(\lfloor \frac{x_{new}}{r_m} \rceil + \frac{1}{2} \right) \cdot r_m \\ \left(\lfloor \frac{y_{new}}{r_m} \rceil + \frac{1}{2} \right) \cdot r_m \\ z_{new} \end{pmatrix} = \begin{pmatrix} x_{alig} \\ y_{alig} \\ z_{alig} \end{pmatrix} = \mathbf{p}_{alig} \in \mathbb{R}^3 \tag{5.3}$$

Pre-Calculation of Circle Offsets

Because of this alignment, the boundary of a circular area in the grid map can be pre-calculated and applied to every point $\mathbf{p}_{alig}$ during the exploration. The goal is to define a set of coordinates describing the circle's boundary relative to the point's center in the OccupancyGrid. These coordinates can be translated to each particular $\mathbf{p}_{alig}$

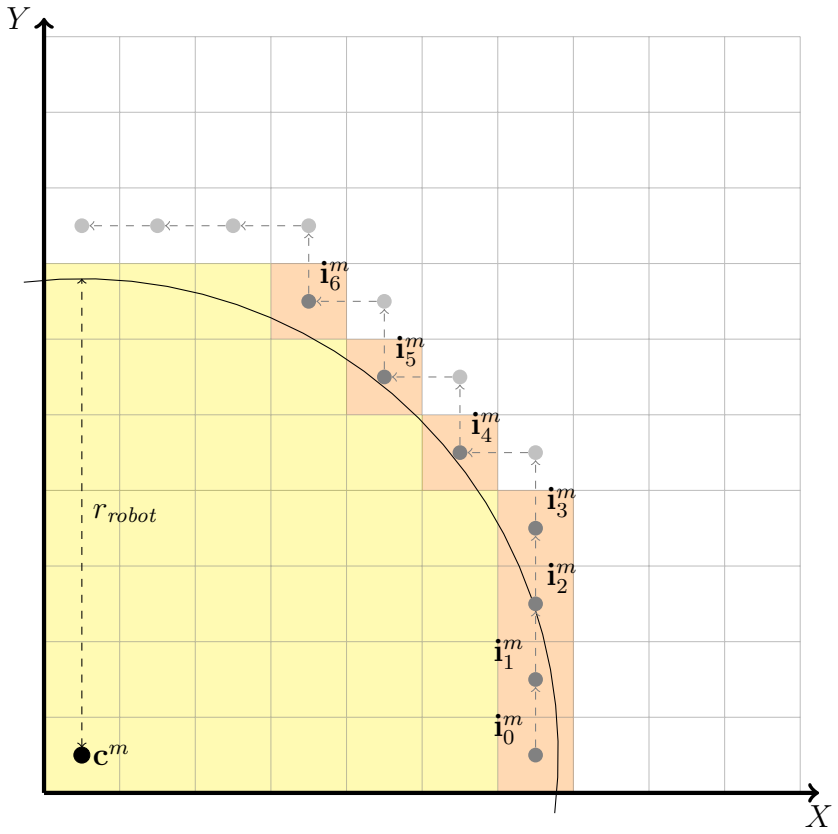


Fig. 5.4.: The first quadrant of a circle with radius r_{robot}, whose center is placed at point $\mathbf{c}^m$, is traversed along its border to derive the corresponding offsets in the grid map. These offsets are found by iterating in y-axis direction while checking if the currently observed cell intersects with the circle. Dark gray dots show the iterator positions $\mathbf{i}_k^m$, $k \in 0, 1, ..., 6$ which correspond directly to a border grid map cell shaded in orange. Light gray dots show cells the iteration skipped because they do not intersect with the circle. Yellow shaded cells represent cells inside the circular area that are not part of the offset cells.

and the area in between them has to be checked for obstacles by iterating over all cells inside it. This check is described in the next sub-section and shown in Figure 5.5b.

To make this check more efficient, each grid cell is queried regarding the row-major data indexing. Therefore, lines parallel to the x-axis are traversed for cache-friendly queries as they are stored consecutively in the data array. Algorithm 5.2 shows the construction of offsets that describe the boundary of the area to be checked. Figure 5.4 visualizes the calculation of these offsets.

To simplify the construction of the offsets, the line symmetry along the x- and y-axes is utilized. Because of it, only one quadrant of the circle must be analyzed. The simplest calculation can be executed in the positive quadrant. Therefore, the center of the circle $\mathbf{c}$ is placed in the center of the first cell next to the map frame origin. The iterator $\mathbf{i}$ is initialized with the same y-coordinate as $\mathbf{c}$ and with its x-coordinate at the center of the last cell that falls inside the circle radius. This first iterator position is then used to derive the first offset which is the number of cells between it and the circle center.

In the outer loop, which terminates when x_i falls below x_c, y_i is decremented by the grid map resolution r_m in each iteration. The inner loop decrements x_i by r_m as long as the circle does not intersect with the iterator's current cell. This is checked by

Algorithm 5.2: Pre-calculation of circle offsets for traversability assessment

1: **procedure** PRECALCULATECIRCLEOFFSETS(r_{robot})
2: $O_c \leftarrow \emptyset$
3: $\mathbf{c} := \begin{pmatrix} x_c \\ y_c \end{pmatrix} \leftarrow \begin{pmatrix} \frac{r_m}{2} \\ \frac{r_m}{2} \end{pmatrix}$
4: $\mathbf{i} := \begin{pmatrix} x_i \\ y_i \end{pmatrix} \leftarrow \begin{pmatrix} x_c + \lfloor \frac{r_{robot}}{r_m} \rceil \cdot r_m \\ y_c \end{pmatrix}$
5: $O_c \leftarrow O_c \cup \begin{pmatrix} \lfloor \frac{x_i - x_c}{r_m} \rceil \\ \lfloor \frac{y_i - y_c}{r_m} \rceil \end{pmatrix}$
6: **while** $x_i \geq x_c$ **do**
7: $y_i \leftarrow y_i + r_m$
8: **while** $x_i > x_c$ **and** $r_{robot} < \sqrt{\left(y_i - y_c - \frac{r_m}{2}\right)^2 + \left(x_i - x_c - \frac{r_m}{2}\right)^2}$ **do**
9: $x_i \leftarrow x_i - r_m$
10: **end while**
11: **if** $x_i \geq x_c$ **and** $r_{robot} \geq y_i - y_c - \frac{r_m}{2}$ **then**
12: $O_c \leftarrow O_c \cup \begin{pmatrix} \lfloor \frac{x_i - x_c}{r_m} \rceil \\ \lfloor \frac{y_i - y_c}{r_m} \rceil \end{pmatrix}$
13: **end if**
14: **end while**
15: **return** O_c
16: **end procedure**

comparing the distance between the bottom left edge of the iterator's cell and $\mathbf{c}$ with r_{robot}. If r_{robot} is greater or equal, an intersection occurs.

The if-condition in the outer loop assesses if the cell at the iterator's last position in the current iteration intersects with the circle by comparing the distance between the cell's lower y-boundary and y_c with r_{robot}. If r_{robot} is greater or equal, the iterator's position is used to derive an offset and add it to the set. The usage of this set is described next.

Circle Traversability Analysis

A circular area with radius r_{robot} at $\mathbf{p}_{alig}$ has to be without obstacles to enable the robot to turn on the spot with a safety margin. To determine if a circular area around $\mathbf{p}_{alig}$ is traversable, Algorithm 5.3 is employed. It iterates over the offsets O_c which are produced in Algorithm 5.2 and applies them to all four quadrants of the circle as can be seen in Figure 5.5b.

It starts by converting the new node's coordinates $\mathbf{p}_{alig}$ into the grid maps coordinates $\mathbf{p}^m_{alig}$ using Equation (5.1). Then, the algorithm iterates over all entries in the offsets which consist of x- and y-direction. For every entry it is checked, if the line derived from the offsets is obstacle-free. Each line is constructed by subtracting the x-offset x^m_o from x^m_{alig} for the starting cell and adding it to x^m_{alig} for the last cell in the line. The line's position on the y-axis is calculated by adding the y-offset y^m_o to y^m_{alig} for the top semicircle and subtract it for the bottom semicircle. For the first entry whose y-offset is 0, only one line is checked as it lies in the middle of both semicircles. The analysis of each line is described in Algorithm 5.4. If it returns *true*, the line is traversable.

Algorithm 5.3: Check traversability of a circular area

```
1: procedure isCircleTraversable(p_alig, O_c)
2:     (x^m_alig, y^m_alig)^T ← worldToOccupancyGrid(p_alig)
3:     for each o^m ∈ O_c, o^m := (x^m_o, y^m_o)^T do
4:         if not isLineTraversable(x^m_alig − x^m_o, x^m_alig + x^m_o, y^m_alig + y^m_o) then
5:             return false
6:         end if
7:         if y^m_o ≠ 0 then
8:             if not isLineTraversable(x^m_alig − x^m_o, x^m_alig + x^m_o, y^m_alig − y^m_o) then
9:                 return false
10:            end if
11:        end if
12:    end for
13:    return true
14: end procedure
```

Algorithm 5.4: Check traversability of a line of grid map cells

```
1: procedure isLineTraversable(x^m_start, x^m_end, y^m)
2:     if x^m_start < 0 or x^m_end ≥ w_m or y^m < 0 or y^m ≥ h_m then
3:         return false
4:     end if
5:     for l^m ← y^m · w_m + x^m_start to y^m · w_m + x^m_end do
6:         if m_l = m_oc or m_l = m_un then
7:             return false
8:         end if
9:     end for
10:    return true
11: end procedure
```

Otherwise, it is not and therefore the complete circle is not traversable.

Algorithm 5.4 receives grid map coordinates from which a single line of consecutive grid map indices is constructed. The x-coordinates range from the starting cell x^m_{start} to the last cell x^m_{end} and the y-coordinate y^m is provided as well. First, it is assured that none of these coordinates violate the OccupancyGrid map dimension boundaries. If they do, the line is not traversable.

The algorithm iterates over all data indices l^m between the start and the end cell which are derived using Equation (5.2). During this process, each cell is examined to observe if it's value m_l equals occupied m_{oc} or unknown m_{un}. If it does, the algorithm terminates and returns that the line is not traversable. If all values are in the range of free cells m_{free}, the line is traversable.

Rectangle Traversability Analysis

The traversability analysis for the rectangular corridor between the nearest node in the tree $\mathbf{p}_{near}$ and the aligned point $\mathbf{p}_{alig}$ can be divided into two cases. The first, trivial

Algorithm 5.5: Check traversability of an aligned rectangular area

1: **procedure** isAlignedRectangleTraversable($r := (x_r, y_r)^T, \gamma, d_{rem}, w_{robot}$)
2: **if** $\gamma = \frac{\pi}{2}$ **or** $\gamma = \frac{3\pi}{2}$ **then**
3: $\mathbf{r}_{bl} := \begin{pmatrix} x_{bl} \\ y_{bl} \end{pmatrix} \leftarrow \begin{pmatrix} x_r - \frac{d_{rem}}{2} \\ y_r + \frac{w_{robot}}{2} \end{pmatrix}$
4: $\mathbf{r}_{tr} := \begin{pmatrix} x_{tr} \\ y_{tr} \end{pmatrix} \leftarrow \begin{pmatrix} x_r + \frac{d_{rem}}{2} \\ y_r - \frac{w_{robot}}{2} \end{pmatrix}$
5: **else**
6: $\mathbf{r}_{bl} := \begin{pmatrix} x_{bl} \\ y_{bl} \end{pmatrix} \leftarrow \begin{pmatrix} x_r - \frac{w_{robot}}{2} \\ y_r + \frac{d_{rem}}{2} \end{pmatrix}$
7: $\mathbf{r}_{tr} := \begin{pmatrix} x_{tr} \\ y_{tr} \end{pmatrix} \leftarrow \begin{pmatrix} x_r + \frac{w_{robot}}{2} \\ y_r - \frac{d_{rem}}{2} \end{pmatrix}$
8: **end if**
9: $\mathbf{r}^m_{bl} := \begin{pmatrix} x^m_{bl} \\ y^m_{bl} \end{pmatrix} \leftarrow$worldToOccupancyGrid($\mathbf{r}_{bl}$)
10: $\mathbf{r}^m_{tr} := \begin{pmatrix} x^m_{tr} \\ y^m_{tr} \end{pmatrix} \leftarrow$worldToOccupancyGrid($\mathbf{r}_{tr}$)
11: **for** $y^m \leftarrow y^m_{tr}$ **to** $y^m_{bl},\ y^m \in \mathbb{N}_0$ **do**
12: **if not** isLineTraversable(x^m_{bl}, x^m_{tr}, y^m) **then**
13: **return** *false*
14: **end if**
15: **end for**
16: **return** *true*
17: **end procedure**

case is a corridor that is aligned with the grid map. The second case is an arbitrarily rotated rectangle.

But first, a calculation is conducted to reduce the overlapping area of rectangle and circle traversability check. The corridor is shortened to a length d_{rem} at which its corners lie on the circle's radius. Otherwise, it would cover an area from one node to the other with length $D(\mathbf{p}_{alig}, \mathbf{p}_{near})$ and width w_{robot}. d_{rem} is calculated from Equation (5.4) and reduces the area which would otherwise be checked twice by d_{diff} times w_{robot}. Figure 5.5 shows this reduction.

$$\begin{aligned} d_{diff} &= \sqrt{r^2_{robot} - \left(\frac{w_{robot}}{2}\right)^2} \\ d_{rem} &= D(\mathbf{p}_{alig}, \mathbf{p}_{near}) - 2 \cdot d_{diff} \end{aligned} \tag{5.4}$$

To decide which algorithm has to be executed for a particular rectangle, its center r and rotation γ from $\mathbf{p}_{near}$ to $\mathbf{p}_{alig}$ are calculated. It is assumed that the origin of the map's rotational component equals zero. Therefore, γ values of zero or multiples of π result in running the first algorithm for an aligned rectangle while other γ values lead to the second, more complicated case. Algorithms 5.5 and 5.6 have r, γ, d_{rem} and w_{robot} as their input.

For the first case, Algorithm 5.5 begins by calculating the coordinates of the bottom left corner $\mathbf{r}_{bl}$ and the top right corner $\mathbf{r}_{tr}$ of the aligned rectangle depending on γ. The corridor from $\mathbf{p}_{near}$ to $\mathbf{p}_{alig}$ can be parallel to the y-axis or to the x-axis. The corner

Algorithm 5.6: Check traversability of a rotated rectangular area

procedure isRectangleTraversable($r, \gamma, d_{rem}, w_{robot}$)
  $\{\mathbf{r}_l, \mathbf{r}_b, \mathbf{r}_t, \mathbf{r}_r\} \leftarrow$ calculateCornerCoordinates($r, \gamma, d_{rem}, w_{robot}$)
  $\{\mathbf{r}_l^m, \mathbf{r}_b^m, \mathbf{r}_t^m, \mathbf{r}_r^m\} \leftarrow$ worldToOccupancyGrid($\{\mathbf{r}_l, \mathbf{r}_b, \mathbf{r}_t, \mathbf{r}_r\}$)
  $a_{asc} \leftarrow \frac{x_r - x_b}{y_r - y_b}$, $a_{desc} \leftarrow \frac{-1}{a_{asc}}$
  $\overline{a} \leftarrow a_{asc}$, $\underline{a} \leftarrow a_{desc}$
  $i_x \leftarrow \left(\left\lfloor \frac{x_b}{\mathbf{r}_m} \right\rfloor + \frac{1}{2}\right) \cdot r_m$, $i_y \leftarrow \left(\left\lfloor \frac{y_b}{r_m} \right\rfloor + 1\right) \cdot r_m$
  $\overline{o} \leftarrow \frac{x_b + \overline{a} \cdot (i_y - y_b) - i_x}{r_m}$, $\underline{o} \leftarrow \frac{x_b + \underline{a} \cdot (i_y - y_b) - i_x}{r_m}$
  $y^m \leftarrow y_b^m$
  while $i_y < y_t$ **do**
    if $y_r + r_m > i_y \geq y_r$ **then**
      $\overline{a} \leftarrow a_{desc}$
      $\overline{o} \leftarrow \frac{x_r + \overline{a} \cdot (i_y - y_r - r_m) - i_x}{r_m}$
      $x_{end}^m \leftarrow x_r^m$
    else
      $x_{end}^m \leftarrow x_b^m + \lfloor \overline{o} \rceil$
    end if
    if $y_l + r_m > i_y \geq y_l$ **then**
      $\underline{a} \leftarrow a_{asc}$
      $\underline{o} \leftarrow \frac{x_l + \underline{a} \cdot (i_y - y_l - r_m) - i_x}{r_m}$
      $x_{start}^m \leftarrow x_l^m$
    else
      $x_{start}^m \leftarrow x_b^m + \lfloor \underline{o} \rceil$
    end if
    if not isLineTraversable($x_{start}^m, x_{end}^m, y^m$) **then**
      return *false*
    end if
    $\overline{o} \leftarrow \overline{o} + \overline{a}$, $\underline{o} \leftarrow \underline{o} + \underline{a}$
    $y^m \leftarrow y^m + 1$
    $i_y \leftarrow i_y + r_m$
  end while
  if not isLineTraversable($x_b^m + \lfloor \underline{o} \rceil, x_b^m + \lfloor \overline{o} \rceil, y^m$) **then**
    return *false*
  end if
  return *true*
end procedure

coordinates are converted to grid map coordinates using Equation (5.1). Then, the algorithm iterates over all grid map slices between the top right corner's y-coordinate y_{tr}^m and the bottom left corner's y-coordinate y_{bl}^m. If one of them is not traversable, the aligned rectangle is also not traversable.

The second case is not as trivial as the aligned rectangle and is described in Algorithm 5.6. At first, the coordinates of all the rectangle's corners are calculated using Algorithm 5.7 and converted to the grid map coordinates with Equation (5.1).

Then, the rectangle's gradients are used to find the lower and upper bounds of the rectangle for each y-coordinate. The algorithm iterates from the bottom corner, which

Algorithm 5.7: Calculate the positions of the corners of a rotated rectangle

procedure CALCULATECORNERCOORDINATES($\mathbf{r} := (x_r, y_r)^T, \gamma, d_{rem}, w_{robot}$)
  if $\gamma < -\frac{\pi}{2}$ **then**
    $\gamma \leftarrow \gamma + \pi$
  else if $\gamma > \frac{\pi}{2}$ **then**
    $\gamma \leftarrow \gamma - \pi$
  end if
  if $0 < \gamma < \frac{\pi}{2}$ **then**
    $\mathbf{r}_l \leftarrow \begin{pmatrix} x_r + \frac{w_{robot}}{2}\cos(\gamma) - \frac{d_{rem}}{2}\sin(\gamma) \\ y_r + \frac{d_{rem}}{2}\cos(\gamma) + \frac{w_{robot}}{2}\sin(\gamma \end{pmatrix}$
    $\mathbf{r}_b \leftarrow \begin{pmatrix} x_r - \frac{w_{robot}}{2}\cos(\gamma) - \frac{d_{rem}}{2}\sin(\gamma) \\ y_r + \frac{d_{rem}}{2}\cos(\gamma) - \frac{w_{robot}}{2}\sin(\gamma \end{pmatrix}$
    $\mathbf{r}_t \leftarrow \begin{pmatrix} x_r + \frac{w_{robot}}{2}\cos(\gamma) + \frac{d_{rem}}{2}\sin(\gamma) \\ y_r - \frac{d_{rem}}{2}\cos(\gamma) + \frac{w_{robot}}{2}\sin(\gamma \end{pmatrix}$
    $\mathbf{r}_r \leftarrow \begin{pmatrix} x_r - \frac{w_{robot}}{2}\cos(\gamma) + \frac{d_{rem}}{2}\sin(\gamma) \\ y_r - \frac{d_{rem}}{2}\cos(\gamma) - \frac{w_{robot}}{2}\sin(\gamma \end{pmatrix}$
  else
    $\mathbf{r}_l \leftarrow \begin{pmatrix} x_r - \frac{w_{robot}}{2}\cos(\gamma) - \frac{d_{rem}}{2}\sin(\gamma) \\ y_r + \frac{d_{rem}}{2}\cos(\gamma) - \frac{w_{robot}}{2}\sin(\gamma \end{pmatrix}$
    $\mathbf{r}_b \leftarrow \begin{pmatrix} x_r - \frac{w_{robot}}{2}\cos(\gamma) + \frac{d_{rem}}{2}\sin(\gamma) \\ y_r - \frac{d_{rem}}{2}\cos(\gamma) - \frac{w_{robot}}{2}\sin(\gamma \end{pmatrix}$
    $\mathbf{r}_t \leftarrow \begin{pmatrix} x_r + \frac{w_{robot}}{2}\cos(\gamma) - \frac{d_{rem}}{2}\sin(\gamma) \\ y_r + \frac{d_{rem}}{2}\cos(\gamma) + \frac{w_{robot}}{2}\sin(\gamma \end{pmatrix}$
    $\mathbf{r}_r \leftarrow \begin{pmatrix} x_r + \frac{w_{robot}}{2}\cos(\gamma) + \frac{d_{rem}}{2}\sin(\gamma) \\ y_r - \frac{d_{rem}}{2}\cos(\gamma) + \frac{w_{robot}}{2}\sin(\gamma \end{pmatrix}$
  end if
  return $\{\mathbf{r}_l, \mathbf{r}_b, \mathbf{r}_t, \mathbf{r}_r\}$
end procedure

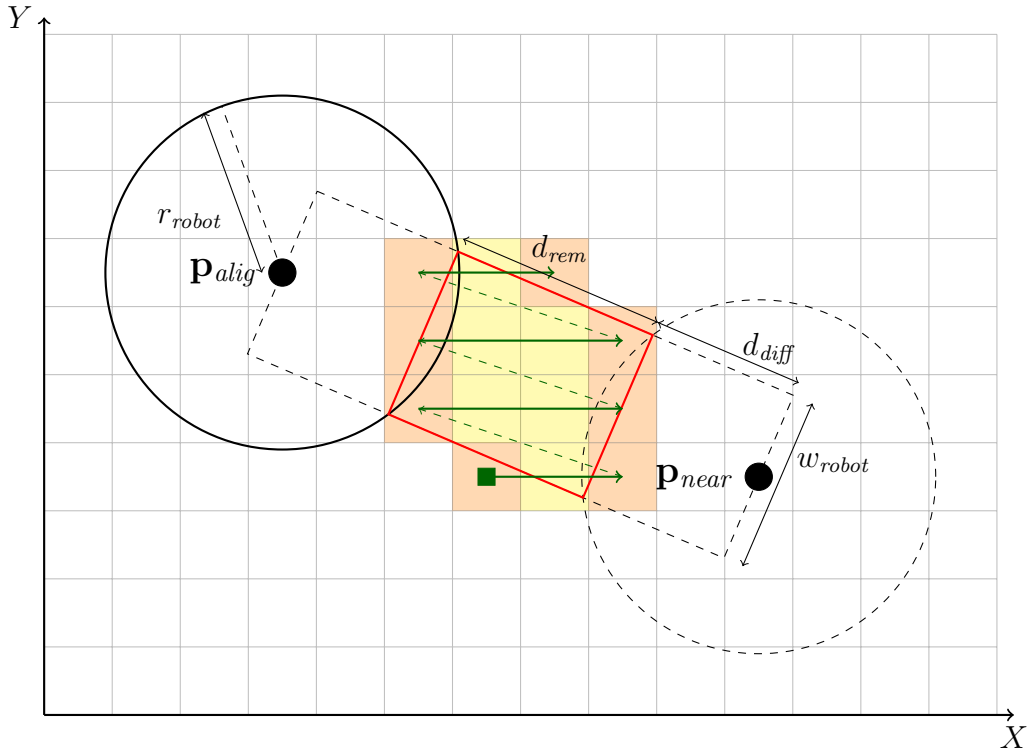
has the lowest y-coordinate of the rectangle, to the top corner. To derive the bounds, the intersection of the boundary of the currently observed cell to the next cell in y-direction with the particular rectangle edge is calculated. The x-coordinates of these cells provide the lower and upper bounds for each slice. The slices in which the left or the right corner or both are present, mark the change of gradient from ascending to descending and vice versa.

Algorithm 5.7 shows that for angles above $\pi/2$ and below $-\pi/2$, the direction is reversed so that all regarded corridors have an angle γ between $-\pi/2$ and $\pi/2$. This removes mirrored cases and the following calculation of the corners is reduced to two cases instead of four.

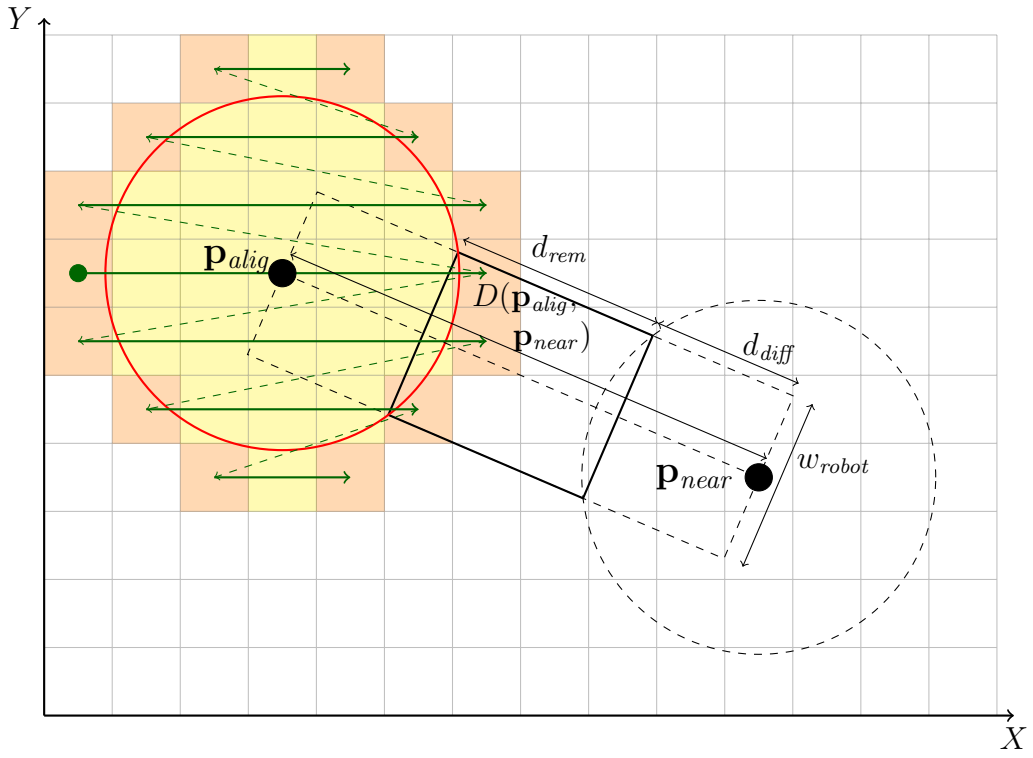
Each corner is calculated using d_{rem} and w_{robot}, as well as sine and cosine of γ. γ also determines which coordinates are assigned to which of the four corners of the rotated rectangle: left $\mathbf{r}_l$, bottom $\mathbf{r}_b$, top $\mathbf{r}_t$ and right $\mathbf{r}_r$.

These coordinates are used to derive the gradient of the edges between the bottom and the left and right corners in Algorithm 5.6. Because of the symmetry in rectangles, the gradient from the bottom to the right corner equals the gradient a_{asc} from the left to the top corner. The gradient from the bottom to the left corner equals the gradient

a_{desc} from the right to the top corner which is also a_{asc}'s inverse.



(a) Traversability check of a rotated rectangular area



(b) Traversability check of a circular area

Fig. 5.5.: Occupancy checks for the grid tiles that intersect with the corridor are shown in (a) and the circle in (b) when $\mathbf{p}_{alig}$ should be connected to $\mathbf{p}_{near}$. Orange tiles mark the outline found by the proposed methods while yellow tiles are also checked. The green arrows show the direction of iterating over the tiles. The green rectangle and circle mark the start.

i_y starts from the top boundary of the cell containing $\mathbf{r}_b$ and iterates over all boundaries along the y-axis until it passes the cell containing $\mathbf{r}_t$. The intersection on the x-axis of the upper and lower edges with the cell boundary at i_y can be calculated by starting at the bottom corner's x-coordinate x_b. Then, the respective gradient times the difference between i_y and y_b is added.

The offsets $\overline{o}$ and $\underline{o}$ from the upper and the lower iterator to the baseline i_x are calculated to derive the start and end coordinates for the line traversability checks described in Algorithm 5.4. i_x is defined as the center of the cell at which $\mathbf{r}_b$ is on the x-axis. Furthermore, y^m represents the current iteration's y-coordinate in the grid map coordinate system and is initialized with y_b^m. The current gradients of the upper edge and the lower edge are stored in $\overline{a}$ and $\underline{a}$ respectively.

While the algorithm loops over all cells between $\mathbf{r}_b$ and $\mathbf{r}_t$ in y-direction, it checks if the left or right corner is passed by the iterator i_y. If it is, the particular upper or lower gradient changes from a_{asc} to a_{desc} for $\overline{a}$ and from a_{desc} to a_{asc} for $\underline{a}$. The upper or lower iterator is reinitialized using the particular x-coordinate of the left or right corner and the difference between the corner's y-coordinate and i_y times the respective gradient. The particular offset is recalculated as well and set to the difference between the matching iterator and i_x in grid cells.

If the left or right corner is passed, the particular grid cell bounds x_{start}^m and x_{end}^m are set to the respective corner's x-coordinate x_l^m or x_r^m. Otherwise, $\overline{o}$ or $\underline{o}$ are added to x_b^m to retrieve x_{start}^m or x_{end}^m respectively. Then, Algorithm 5.4 is called with x_{start}^m, x_{end}^m and y^m. If the line is traversable, the algorithm continues, otherwise it stops and returns that the rectangle is not traversable.

Afterwards, $\overline{o}$ and $\underline{o}$ are increased by the respective gradient and y^m and i_y move to the next grid cell in y-direction. When $\overline{o}$ and $\underline{o}$ are reinitialized at reaching the left or right corner, the respective gradient for one grid cell is subtracted first because they have been previously incremented in each iteration. In these iterations, x_{start}^m and x_{end}^m are derived directly from the particular corner's x-coordinate in the grid map.

This concludes one iteration and the next one begins. After the last iteration, that ends when i_y passed the top corner, a last line traversability check is executed.

Application of the Traversability Analysis

The previously described algorithms are applied as shown in Figure 5.5. The corridor is checked first, followed by the circle. If both checks pass, the new node is added to $\mathcal{T}$.

Figure 5.6 depicts a screenshot from RViz which highlights the actual area of multiple nodes and corridors between them that have been checked in the OccupancyGrid map.

5.1.5. Sparse Ray Polling

After the nodes are placed, it has to be decided if it is rewarding to explore a specific node or if the space surrounding it is already mapped. To obtain this information, the possible map coverage to be added when exploring this node needs to be calculated. It is approximated using the amount of previously unknown space the robot is expected to perceive at the node's position which is called the node's gain.

To calculate this amount, rays can be traced from the position the sensor would be at at this node into every direction that could be in the sensor's FoV. Therefore, the

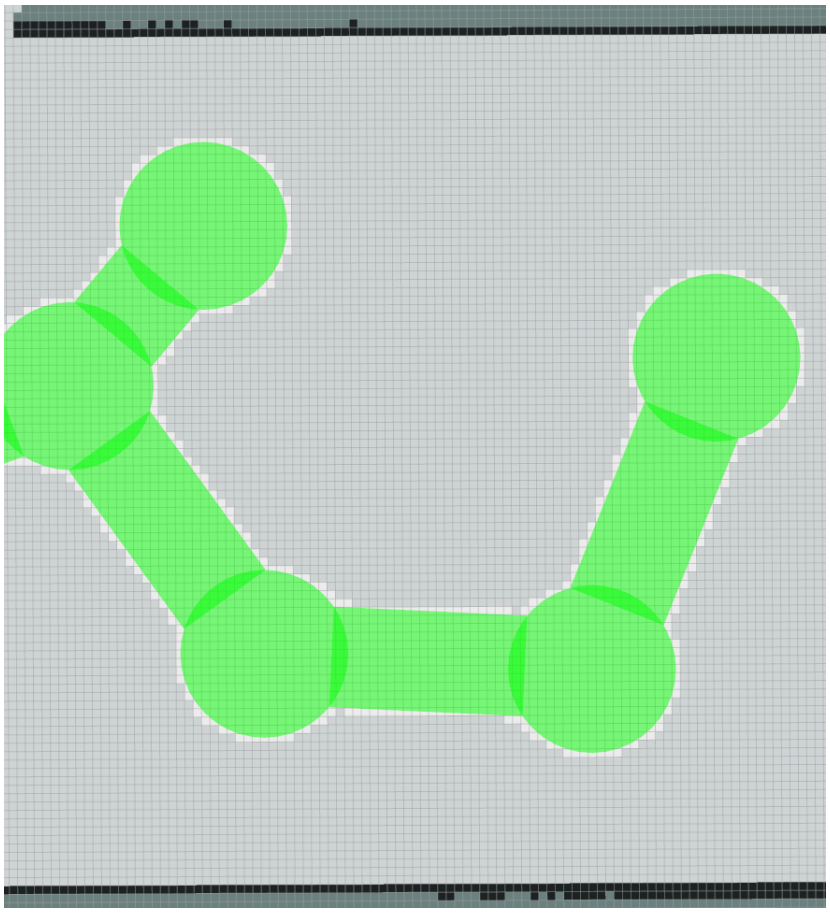


Fig. 5.6.: An RViz screenshot including the circular and rectangular areas checked for the traversability analysis is shown. The areas that are highlighted in green represent the actual areas while the grid cells in the lightest shade of gray have been checked using Algorithms 5.3, 5.5 and 5.6. The grid map cells in a medium shade of gray represent free space, a dark shade of gray is unknown space and black cells show occupied space.

node is placed at a height above the ground that is equal to the sensor's height h_{sensor} measured from the bottom of the robot. Vertical ray tracing is used to identify the ground height at the sampled node's x- and y-coordinates.

To identify the orientation that yields the best gain for each node, ray tracing is executed all around it. Ray tracing operations in an OctoMap are a computation heavy task. Therefore, SRC which is a less computation intensive option proposed by Selin et al. [7], is used as a foundation for the following approach that is called Sparse Ray Polling (SRP).

SRP samples a set P of predefined positions in the space V surrounding a node. This set of sampling points P is built based on the parameters supplied to the algorithm and is defined in Equation (5.5). P consists of all positions that are defined by the iterators r_i, ϑ_j and φ_k which range from their respective lower to upper bounds. r_{min} and r_{max} for r_i are derived from the sensor's minimum and maximum range, ϑ_{min} and ϑ_{max} for ϑ_j range from the minimum of the sensor's vertical FoV to its maximum and $\varphi_{min} = 0$ and $\varphi_{max} = 2\pi$ for φ_k describe the circle's circumference.

$$
\begin{aligned}
P = &\bigcup_{r_i}\bigcup_{\vartheta_j}\bigcup_{\varphi_k} P(r_i, \vartheta_j, \varphi_k) \\
&\{r_i = \Delta_r \cdot i \mid i \in \mathbb{N},\ \frac{r_{min}}{\Delta_r} \leq i \leq \frac{r_{max}}{\Delta_r}\} \\
&\{\vartheta_j = \Delta_\vartheta \cdot j \mid j \in \mathbb{N}_0,\ \frac{\vartheta_{min}}{\Delta_\vartheta} \leq j \leq \frac{\vartheta_{max}}{\Delta_\vartheta}\} \\
&\{\varphi_k = \Delta_\varphi \cdot k \mid k \in \mathbb{N}_0,\ 0 \leq k < \frac{2\pi}{\Delta_\varphi}\}
\end{aligned}
\tag{5.5}
$$

The iterators are bound to user-defined step sizes $\Delta = (\Delta_r, \Delta_\vartheta, \Delta_\varphi)$. They should be set according to the voxel grid size's edge length, the sensor's resolution and the processing power of the computer running the gain calculation. If the steps are too small, the computation time increases and voxels are likely to be sampled multiple times. Too large steps decrease the computation time but can lead to omitted voxels and less informative results. Figure 5.7 shows exemplary sample points in a voxel grid.

When the exploration is running, the respective node's position is added to each pre-calculated sample point's position to obtain the coordinates to check in the OctoMap for calculating the node's gain. If one of the sample points on a ray is occupied, the remaining points on this ray, that are further away from its start, are not sampled because the sensor's vision in that direction is blocked. This pre-calculation is expected to reduce the time to calculate each node's gain because the OctoMap's ray cast method does not have to be used to count the unknown voxels for each new node first.

To obtain the particular node's gain $G(n)$, P is translated to an exemplary node's position $\mathbf{p}_n$. The translation on the z-axis is assessed by initially setting the height z_n of $\mathbf{p}_n$ to the height of its parent node in $\mathcal{T}$.

Afterwards, vertical ray tracing in the OctoMap is performed to find the ground's

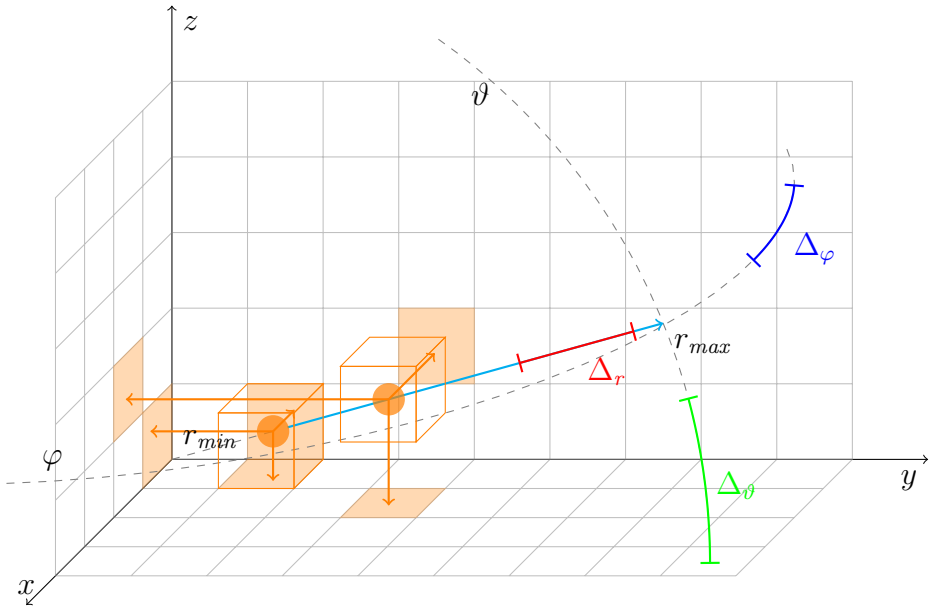


Fig. 5.7.: SRP is shown in a voxel grid with two exemplary sample points colored in orange that are on the cyan arrow, which is the ray cast direction. The orange cubes represent the voxels, that are sampled, while the orange arrows and grid tiles aid in identifying the voxels' positions in the grid.

height h_{ground} at $\mathbf{p}_n$. The node's height is then set to $z_n = h_{ground} + h_{sensor}$. If no ground within a maximum height difference h_{max} from z_n is found, the gain function finishes and $G(n)$ is set to -1.

The full horizontal revolution is polled to obtain the best orientation φ_{max} for the robot at $\mathbf{p}_n$. Therefore, a horizontal sector with the size of the sensor's horizontal FoV with the most gain $G(\varphi_{max})$ is determined. Equation (5.6) describes finding $G(\varphi_{max})$ over all discrete orientations φ_i using the sum of gains for each slice of poll points $G_\varphi(i)$ inside the horizontal FoV φ_{hor}.

$$G(\varphi_{max}) = \max_{\varphi_i} \sum_{i=\varphi_i - \frac{\varphi_{hor}}{2}}^{\varphi_i + \frac{\varphi_{hor}}{2}} G_\varphi(i),\ i \in \varphi_k \tag{5.6}$$

The node's next state s_{n+1} is derived using Equation (5.7) which depends on the maximum number of observable voxels in the sensor's FoV g_{max} and a user-defined threshold g_{min}. A repeated exploration of a node is not reasonable if the node's current status s_n equals *visited* and the recalculated $\varphi_{max+1} \approx \varphi_{max}$. This could lead to the robot getting stuck re-exploring the same node repeatedly. If s_n is *visited* and $\varphi_{max+1} \neq \varphi_{max}$, the node's new state depends solely on its gain.

$$s_{n+1} = \begin{cases} \textit{explored}, & G(\varphi_{max})/g_{max} < g_{min} \text{ or} \\ & s_n = \textit{visited} \text{ and } \varphi_{max+1} \approx \varphi_{max} \\ \textit{initial}, & G(\varphi_{max})/g_{max} \geq g_{min} \end{cases} \tag{5.7}$$

The node's gain $G(n)$ is set to $G(\varphi_{max})$ and the best orientation φ_{max} is stored if the node's state remains *initial* or *visited*. If it is set to *explored*, $G(n)$ and φ_{max} are set to zero.

5.1.6. Reward Function

RNE's goal is to enable a robot to create a 3D map of a previously unknown environment. For this, RRT is deployed to create a tree of nodes which serve as possible exploration goals. The order in which the nodes are selected as the next goal and the decision if a particular node is worth exploring, is based on a reward function.

This reward function $R(n)$ is used to determine the NBV which serves as the next exploration goal. It is calculated for all nodes in $\mathcal{T}$ whose state is not *explored* or *failed*. The node with the highest reward is selected as the NBV. $R(n)$ is shown in Equation (5.8) and consists of the gain $G(n)$ which was introduced in the previous section and the cost $C(n)$. The reward function is adapted from RH-NBVP [6].

$$R(n) = G(n) \cdot C(n) = G(\varphi_{max}) \cdot e^{-d_{rn}} \tag{5.8}$$

$C(n)$ for node n is determined by the distance along the tree's edges d_{rn} between the node closest to the robot $\mathbf{p}_{n_r}$ and the particular node's position $\mathbf{p}_n$. This distance metric guarantees that a path with the calculated length exists while the Euclidean distance disregards possible obstacles between both nodes and resulting detours.

The formula for $C(n)$ gives a strong bias towards nearby goals. This should reduce the total exploration duration and the traveled path length because it avoids back-and-forth motion. On the other hand, the explored volume is expected to increase steadily.

A greedier approach that only focuses on maximizing the gain explores vaster amounts of space early in the exploration.

5.1.7. Global Navigation Planner

The exploration approach introduced in this chapter creates a tree $\mathcal{T}$ in the world space to map an unknown environment. It identifies the nodes in the tree that are worth exploring. A part of this approach is the navigation planning from the robot's current position to a particular goal node.

For this, a global planner for the ROS navigation stack is proposed which follows the tree's edges when moving from one node in $\mathcal{T}$ to the next. The path also resembles the distance from the cost function $C(n)$ and therefore gives a more precise estimation of the real cost for traversing to a particular node.

The introduced planner is named `RneGlobalPlanner` and is refined in the following chapters. The word global in its name refers to the naming of the ROS navigation package which divides planners into local and global. A global planner proposes a more direct path that does not necessarily regard the robot's kinematics and relies on a map built by the robot. A local planner considers the robot's kinematics and actual sensor input to be able to react to dynamic obstacles or changes in the map in a restricted area around the robot. This differentiation is independent of the distinction between local and global exploration shown in Chapter 8

The distance d_{rn} and the path P_{rn} in $\mathcal{T}$ between $\mathbf{p}_{n_r}$ and $\mathbf{p}_n$ are stored for each node n during the exploration. P_{rn} holds an ordered list of references to all nodes in the path from the particular node to $\mathbf{p}_{n_r}$ including itself. Therefore, the robot's position is actively monitored and it is always checked which node in $\mathcal{T}$ is closest to the robot. If this node changes due to the robot's movement, d_{rn} and P_{rn} are updated for all nodes.

This update is based on the following logic. When a new node n is added to $\mathcal{T}$, d_{rn} and P_{rn} are calculated based on its parent node n_p. d_{rn} equals the parent node's distance to the robot d_{rn_p} plus the distance between node and parent node. P_{rn} copies P_{rn_p} and the new node is added to the list.

The update of d_{rn} and P_{rn} for each node when the robot moves and $\mathbf{p}_{n_r}$ changes, depends on the direction of the movement in the tree regarding the particular node:

Towards node: The robot moves towards the regarded node along a tree edge. d_{rn} is reduced by the distance between new and old nearest node. The former nearest node is removed from P_{rn}.

Away from node: The robot moves away from the regarded node along a tree edge which results in d_{rn} being incremented by the distance between new and old nearest node. The new nearest node is added to P_{rn}.

Jump to node: The new and old nearest node have no connecting edge in the RRT. This is caused by a localization error or a local planner deviation from the global path. All d_{rn} and P_{rn} are recalculated starting at the currently nearest node with a breadth-first expansion through the complete tree. The same logic as for initializing a new node is used. The exception is that d_{rn} and P_{rn} are not always initialized from the parent node but also from child nodes, depending on the expansion direction.

When the next exploration goal is selected, the particular node's path is retrieved, intermediate poses are added between all edges and the robot's position is connected to the first or second node in the path. The robot can be closer to the first node but already on the edge between the first and second node. If this is the case, the first and currently nearest node is removed from the path and the robot's position is directly connected to the second node. Otherwise, it is connected to the first node.

5.1.8. Termination Condition

There are two exit conditions for RNE which are provided to it as parameters. The first is the timer duration t_{exit} which is started when the list of nodes to be explored is empty and stopped when a new node is successfully added. The timer value should be set depending on the area to be explored and the available computation power. Restricted areas with narrow passages tend to increase the time RRT needs to place new nodes. Once the timer finishes after no new nodes have been placed for the set amount of seconds, the exploration finishes.

The minimum gain threshold g_{min} is the indirect second exit condition because it influences when nodes are set to the status *explored*. The exploration terminates as soon as all nodes are explored and no new nodes can be added. If the value of g_{min} is too low, all nodes with at least a minimal gain are visited which increases the overall duration. If it is too high, most nodes are set to *explored* before they are visited which reduces the total map coverage.

5.2. Sparse Ray Casting and Sparse Ray Polling

This work's approach SRP is significantly inspired by SRC which was introduced by Selin et al. [7]. They proposed a sparse sampling in an OctoMap at user-defined intervals in r, φ and ϑ directions.

Spherical volume elements are attributed along the ray in radial distance with an increasing volume further away from the node. The volume contributes to the specific gain, so that an unknown voxel closer to the node yields less gain than one further away. They sample all around a node and extract the orientation with the maximum gain for the particular node.

Selin et al. showed that their approach outperforms RH-NBVP while requiring far less computation time. In RH-NBVP, the sensor orientation is randomly sampled like the node's position. For this orientation, a view frustum is constructed from the FoV and range of the sensor. Then, all voxels inside the frustum in the OctoMap are iterated over to derive the gain. Selin et al. compared their SRC with RH-NBVP's approach which takes approximately 100 times longer to calculate the gain.

Oleynikova et al. [87] employed sub-sampling of the frustum derived in RH-NBVP by checking only one voxel out of a user-defined number. They showed, that sampling only one out of 20 voxels reduces the computation time by factor three while the percentage of unknown voxels only deviates by one percent compared to the full sampling method.

Based on the findings in [7, 87], SRP is developed to reduce the time required for calculating the potential gain of a node while sustaining a similar level of map coverage. It is detailed in Section 5.1.5. The following comparison through a set of experiments highlights the differences between SRC and SRP.

Experimental Setup

The experiments are conducted in the simulation environment Gazebo and run on a computer using Ubuntu 18.04 with 16GB RAM, an AMD Ryzen 5 1600 six core processor and an NVIDIA GeForce GTX 1050Ti GPU. The world used for these experiments can be seen in Figure 5.8a.

Since the calculation depends on the sensor's FoV and range, three different sensor setups are used for variation which are listed below with their respective abbreviations:

RS: The first setup has an Intel RealSense depth camera with a FoV of 87x58 degrees mounted on a swiveling joint which enables a total horizontal FoV of 223 degrees. The camera's sensing range lies between 1 m and 8 m and is placed on a Clearpath Robotics Husky UGV[2]. It can be seen in Figure 5.8d.

VL: The second sensor is a Velodyne VLP-16 lidar[3] tilted at a 90 degrees angle on a rotating joint which results in an all around 360x135 degrees FoV. The Velodyne's range starts at 0.5 m and extends up to 100 m. It is placed on a Clearpath Robotics Husky UGV as well and depicted in Figure 5.8c

T3: The last setup also has the above-mentioned depth camera without a swiveling joint mounted on a TurtleBot3 Burger[4] and is shown in Figure 5.8b.

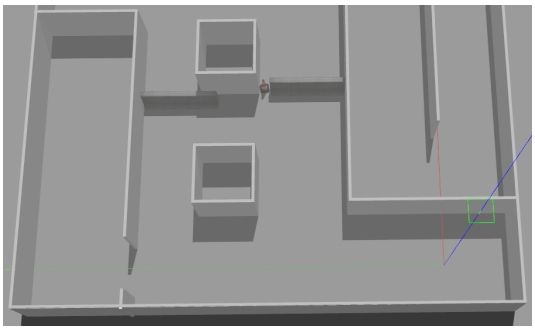
The ROS package GMapping is used as the SLAM approach which is described in Section 3.2.2. GMapping needs odometry from the robot to estimate the localization which is provided by the simulated robot. The Husky UGV uses an EKF to fuse the Inertial Measurement Unit (IMU) and wheel encoder readings to derive an odometry.

A SLAM approach is required to localize the robot and create a map to use for the traversability analysis. Since the environments for the evaluation are flat, a 2D SLAM is sufficient with no need for an additional algorithm to evaluate traversability based on the robot's kinematic constraints. The ROS navigation stack's DWA planner is employed for local planning which follows the proposed `RneGlobalPlanner`'s waypoints while trying to maintain a secure distance to obstacles.

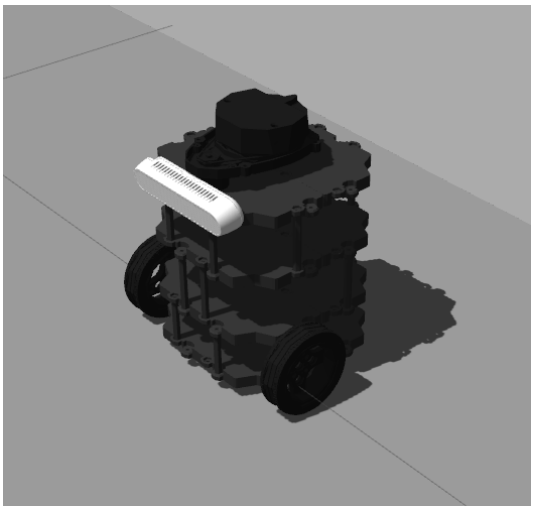
SRP and SRC are used directly after each other so that the OctoMap is assumed to be identical for every node created by RNE whose gain must be calculated. For each pair of calculations, the duration t, the number of voxels that are checked v and the resulting best yaw φ_{max} are recorded. The maximum number of voxels v_{max} is the highest recorded number of observed voxels that is seen from a single node during an experiment. The number of voxels and max voxels is aggregated over all φ steps and henceforth the view score refers to this total number of voxels and not to the number of voxels inside the FoV of the best orientation φ_{max}.

Depending on the step sizes $\Delta = (\Delta_r, \Delta_\vartheta, \Delta_\varphi)$, the duration and the number of voxels varies greatly. To verify similar behavior throughout different settings, a variety of step sizes is chosen for each sensor. The different step sizes are 5, 10 and 15 degrees for Δ_ϑ and Δ_φ for these experiments. They determine the number of rays to trace. The Δ_r step size varies between 2, 5, 10 and 15 cm for the depth camera sensor setups and 3, 5, 10 and 15 cm for the lidar sensor setup.

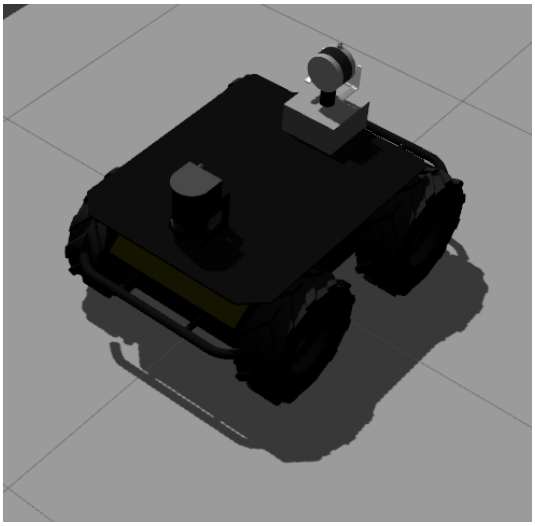
[2] `https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/`
[3] `https://velodynelidar.com/products/puck/`
[4] `https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/`



(a) Simulation environment with an omitted roof



(b) Turtlebot3 with an Intel RealSense depth camera as T3



(c) Husky UGV with a rotating Velodyne PUCK VLP-16 lidar as VL

(d) Husky UGV with an Intel RealSense depth camera as RS

Fig. 5.8.: The Gazebo simulation environment world can be seen in (a) and different robots and sensor setups with their abbreviations in (b) to (d).

The computer used for the simulation is unable to cope with the lidar setup's longer range and therefore increased number of voxels to check for a step size of 2 cm. Therefore, it is increased to 3 cm. The voxel edge length of the OctoMap for these experiments is always the same as Δ_r.

The above-mentioned step sizes are combined which results in 12 configurations for each sensor setup. For each configuration, one exploration is conducted and all gain calculations are saved in pairs. For every pair, the difference in best yaw φ_{dif} and the difference in view score vs_{dif} is calculated with Equations (5.9) and (5.10) respectively. v and v_{max} for SRP and SRC respectively are used to deduct a view score vs for each gain calculation. φ_{dif} must be between 0 and 180 degrees since 0 degrees equals 360 degrees orientation.

$$\varphi_{dif} = \begin{cases} |\varphi_{SRC} - \varphi_{SRP}|, & 0 \leq |\varphi_{SRC} - \varphi_{SRP}| \leq 180 \\ 360 - |\varphi_{SRC} - \varphi_{SRP}|, & |\varphi_{SRC} - \varphi_{SRP}| > 180 \end{cases} \tag{5.9}$$

$$vs_{dif} = \frac{v_{SRC}}{v_{SRC_{max}}} - \frac{v_{SRP}}{v_{SRP_{max}}} \tag{5.10}$$

For the duration, the number of voxels and the maximum number of voxels factors are determined by dividing the SRC values by the SRP values which results in the duration factor $\lambda_t = \frac{t_{SRC}}{t_{SRP}}$, the voxels factor $\lambda_v = \frac{v_{SRC}}{v_{SRP}}$ and the maximum number of voxels factor $\lambda_{v_{max}} = \frac{v_{SRC_{max}}}{v_{SRP_{max}}}$.

Experiment Results

To compare SRP and SRC, the mean of every recorded attribute is computed which includes the factors λ_t, λ_v and $\lambda_{v_{max}}$. For the yaw φ_{dif} and view score differences vs_{dif}, their respective Standard Deviation (SD) σ is computed in addition to their mean value μ. The averages over the different configurations can be seen in Table 5.1 and the values for each configuration in the Appendix in Tables A.1 (p. 160) and A.2 (p. 162).

The overall results show that SRP is around 12 times faster than SRC. For the particular configurations it can be deduced that this value is corresponding to the different step sizes and henceforth the maximum amount of checked voxels. The values range from a factor of around 2.5 for the lidar with step sizes of 10 degrees for Δ_ϑ and Δ_φ and 3 cm for Δ_r, to approximately 54 for the depth camera with step sizes of 5 degrees for Δ_ϑ and Δ_φ and 15 cm for Δ_r.

It is noticeable that the factor scales differently when changing Δ_ϑ and Δ_φ or Δ_r. Increases and decreases in all step sizes proportionally lead to an increase or decrease in the maximum number of checked voxels. But only Δ_ϑ and Δ_φ correlate with the average duration and henceforth the duration factor while Δ_r seems to be inversely proportional to the duration factor. This is caused by the SRC duration remaining approximately the same when only changing Δ_r while the SRP duration scales proportionally with it. Therefore, the SRC duration is mostly based on the number of rays cast and the length of the ray while the voxel edge length only plays an inferior part. This indicates that there are configurations with a sufficiently fine-grained resolution where SRC is faster than SRP. But the computer used for these experiments is not able to operate with these configurations as they lead to a crash of the simulation. Therefore, they are probably not suitable to be used on a mobile robot with more limited processing power.

The average factor for the number of observed voxels for the swiveling depth camera is at roughly 1.41 which means that SRC examines around 40 percent more voxels than SRP. For the lidar, this factor is at approximately 1 which means they are equal in average but the particular configurations vary from a factor of 0.79 to 1.13. The rigidly mounted depth camera setup's average factor is 1.34. For the swiveling and rigidly mounted depth camera, the maximum voxels factor is around 50 percent higher for SRC while for the lidar the factor is at 1.28.

The higher maximum voxels factors are caused by SRC checking every voxel a ray is cast through while SRP only samples along the ray. For horizontal or vertical rays,

Tab. 5.1.: The averages of the duration λ_t, voxels λ_v and maximum number of voxels factor $\lambda_{v_{max}}$ as well as the averages μ and SDs σ of the yaw φ_{dif} and view score differences vs_{dif} are shown for the comparison between SRC and SRP. These values are depicted for the Husky robot with a depth camera as RS, the Husky robot with a lidar as VL and the Turtlebot3 robot with a depth camera as T3. They are also averaged over all configurations of the comparison.

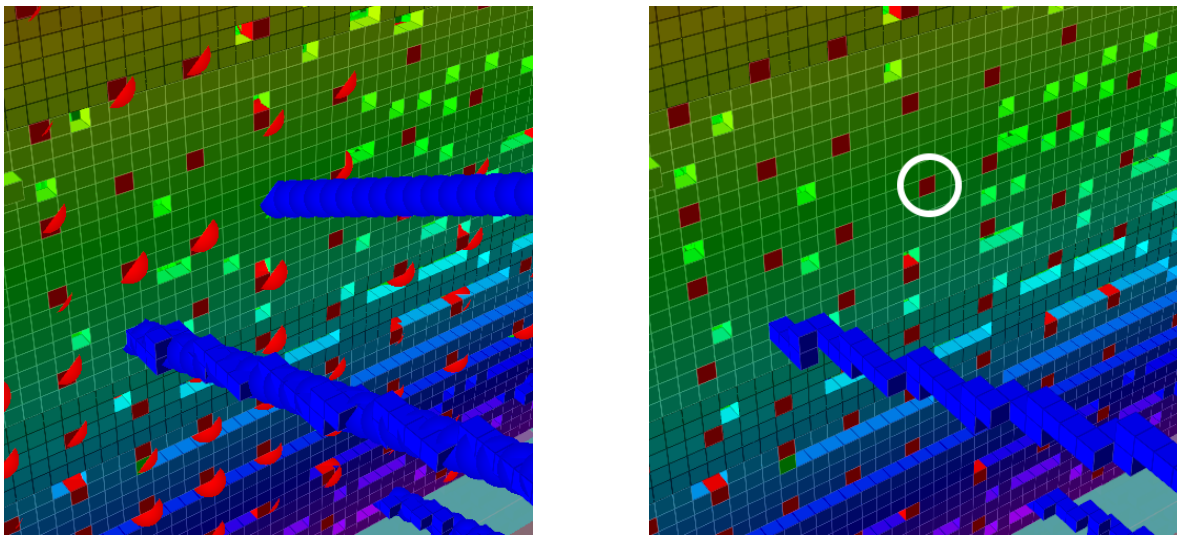
Configuration	λ_t	λ_v	$\lambda_{v_{max}}$	φ_{dif}		vs_{dif}	
				μ	σ	μ	σ
RS	8.52340	1.41394	1.47323	4.87078	17.58356	-0.00917	0.01329
VL	7.55720	1.01844	1.28347	21.98247	37.73998	-0.04124	0.04423
T3	18.88893	1.34252	1.51288	9.42592	26.14318	0.00493	0.03981
All	11.65651	1.25830	1.42319	12.09305	27.15557	-0.01516	0.03244

the number of voxels is the same for both variants. Because both depth camera setups have the same range, voxels and maximum voxels factors are expected to be similar.

Furthermore, the behavior described below is observed during the experiments. Rays traced by SRP are not always aborted upon hitting a solid wall. Figure 5.9 shows a wall in the OctoMap and rays traced by SRP and SRC.

The image on the left features both SRP and SRC while SRP is hidden on the right. A ray going through a gap in the wall in the bottom left corner can be seen where both approaches continue the ray tracing. A second ray can be seen in Figure 5.9a which is only traced by SRP while Figure 5.9b shows that SRC stops tracing because of the wall as indicated by the white circle. It can be deduced that SRP occasionally penetrates a wall in the OctoMap. This is caused by an unfavorable angle and distance where the sample points are placed directly before and behind the wall.

Because of the significantly longer range of the lidar sensor compared to the depth camera, the penetration effect for SRP described before is likely to cause similar voxels and max voxels factors between SRP and SRC for the lidar setup. If such a penetration



(a) SRC and SRP **(b)** SRC

Fig. 5.9.: Penetration of walls for SRP depicted as blue and red spheres and SRC as blue and red cubes shown with OctoMap voxels as purple to green cubes. (a) shows SRP and SRC while (b) only shows SRC with the white circle marking the position where the ray in (a) penetrates the wall.

effect occurs for the lidar setup, far more voxels are added than if it occurs for the depth camera.

In addition to this, the longer range in the enclosed simulation environment increases the impact of horizontal rays for the long-range lidar since there often is more distance to a far wall than to the ceiling or ground. These horizontal or near horizontal rays have the same number of voxels for SRP and SRC. The shorter range of the depth camera prevents this effect.

The overall combined average yaw difference is approximately 12.1 degrees with a SD of 27.2 degrees. For the swiveling depth camera, the average yaw difference is only 4.9 degrees with a minimum of 1.6 degrees for $\Delta_\vartheta = \Delta_\varphi = 15$ degrees and $\Delta_r = 10$cm. The rigidly mounted depth camera has an average yaw difference of 9.4 degrees and the lidar 22.0 degrees with a maximum of 35.2 degrees for $\Delta_\vartheta = \Delta_\varphi = 15$ degrees and $\Delta_r = 5$cm.

The significantly higher average yaw difference and SD of the lidar setup is probably caused by the sensor's 360 degrees horizontal FoV. Minor differences in the sum of perceived unknown voxels are the reason that a completely different yaw is chosen. A smaller horizontal FoV creates a stronger focus towards a particular direction, that according to the lower average yaw differences for the depth cameras, is less prone to these minor differences between SRC and SRP.

The view score differs heavily between the different setups and configurations which is reflected in the SD being larger than the mean values. When regarding the view score and the view score differences for single configurations in Appendix A (p. 160), it can be seen that the differences are small compared to the overall view score mean values.

This indicates that using SRC or SRP has only a minor impact on the view score of each particular node which results in a similar gain $G(n)$ as well as selecting the same nodes as the next exploration goals.

In conclusion, SRP is around 12 times faster than SRC while obtaining similar results regarding the best orientation and view score which is derived from the identified amount of observed unknown voxels. It coincides with the findings in [87] which shows that sub-sampling decreases the computation time while having only a minor impact on the calculated relative gain.

5.3. Coupled and Decoupled Gain Calculation

To demonstrate the advantage of decoupled gain calculation over a coupled gain calculation, simulations are conducted. Decoupled gain calculation describes the computation of the node gain in a different process than the remaining RNE as explained in Section 5.1.2. Coupled gain calculation blocks the exploration process while it determines the node gains after reaching a goal.

The simulations are run in Gazebo and reuse the previously introduced environment and configurations from Section 5.2 which can be seen in Figure 5.8. Step sizes are set to 10 degrees for Δ_φ and Δ_ϑ and 10 cm for Δ_r.

For each of the configurations, 10 runs with coupled gain calculation and 10 runs with decoupled gain calculation are executed. For the coupled gain calculation, the exploration mode is set to *finish* and for decoupled gain calculation, it is set to *interrupt*. These modes can be set in the RSM and are explained in detail in Section 4.1.3. The

Tab. 5.2.: The mean value μ and SD σ of exploration duration, path length and mapped occupied voxels for different configurations are shown for multiple simulation runs using coupled and decoupled gain calculation as CF and DI respectively. The configurations are the Husky robot with a depth camera as RS, the Husky robot with a lidar as VL and the Turtlebot3 robot with a depth camera as T3. The best mean values for each robot configuration are printed in bold letters.

Configuration	Duration [s]		Path [m]		Voxels	
	μ	σ	μ	σ	μ	σ
RS-CF	336.35	90.08	74.71	30.56	148038.8	42746.3
RS-DI	**315.68**	35.60	**54.34**	10.73	**153983.4**	10013.1
VL-CF	467.95	261.64	74.72	39.33	127038.5	22119.7
VL-DI	**434.54**	68.19	**65.28**	20.25	**127473.4**	10327.3
T3-CF	616.65	214.15	96.23	39.68	100886.4	12412.2
T3-DI	**445.69**	95.92	**58.94**	9.66	**102779.4**	7950.0

interrupt mode allows abandoning a currently pursued goal if a better goal becomes available while the *finish* mode enforces the completion of the current goal before transitioning to the next.

The coupled gain calculation combined with the *finish* exploration mode is referenced as CF and the decoupled gain calculation with the *interrupt* exploration mode as DI. Table 5.2 shows the results of the simulations regarding exploration duration, path length and mapped occupied voxels. For all values, the mean μ and the SD σ are presented.

For all configurations, the decoupled gain calculation leads to shorter duration and paths as well as a slightly increased amount of perceived occupied voxels. The large SD is caused by the randomness of the node sampling and is two to three times as large for CF because sub-optimal goals are pursued even when better alternatives become available.

The improvement in duration varies between approximately 7% for RS and VL and peaks at around 38% for T3. The path length is reduced by approximately 13% for VL to 39% for T3. The improvement in mapped occupied voxels is below 5% for all configurations.

The experiment demonstrates that the ability to continue moving towards a goal while not all nodes' gains are calculated and to abort inferior goals when better ones become available, significantly reduces the exploration duration and path length.

5.4. Conclusion

This chapter describes RNE's first iteration that is based on an RRT to produce a tree structure whose nodes are evaluated regarding their particular information gain and distance to the robot to select an NBV. The information gain of each node is calculated using SRP and is decoupled from the remaining exploration process. This enables aborting a current goal as soon as a superior goal becomes available. A distance metric based on the path from the node nearest to the robot to the particular node is proposed. It is also used to derive a global navigation path for the robot to follow.

But there are the following drawbacks with this implementation:

1. The larger the already explored space grows, the fewer samples are placed close

to the robot. This increases the likelihood of back-and-forth motions because no new nodes are placed near it and it has to move to unexplored nodes further away.

2. The global navigation path offers a previously traversability-checked, safe corridor for the robot but can lead to large detours due to the tree structure. In the case of a nearby goal that is not in the same branch of the tree, the global path goes back along the tree's edges. Only when the first mutual node towards the goal node is reached, it moves upwards in this branch.

3. These occasionally inefficient global paths are also used to calculate the distance to each node which favors the exploration of the current tree branch but hinders exploring nearby promising nodes in other branches.

Because of these drawbacks, early comparisons with implementations of RH-NBVP and AEP adapted to UGVs outperform this first iteration of RNE. This can be seen in an experiment shown in the following chapter's Table 6.1 and outline that RNE requires improvements.

When using the default global planner from the ROS navigation stack instead of the introduced global path planner, RNE is able to increase the map coverage and reduce the exploration time. The default global planner follows a direct path towards its goal if there are no obstacles in the way.

But the proposed global planner's included traversability check of all nodes and edges is an important aspect of RNE. Instead, other improvements to the exploration approach are introduced in the next chapter which are expected to compensate the drawbacks mentioned above.

6. RNE 2 - RRG-Based Exploration

The second iteration of RNE is presented in this chapter. It builds on top of Chapter 5 in which RRT is utilized for building a tree that guides the exploration. A global planner based on this tree, that was introduced in the first iteration, causes large detours and decreases the exploration's efficiency.

In this iteration, RRT is replaced with RRG, that is explained in Section 3.1.3. Replacing the tree with a graph allows for advanced connectivity between the nodes to be explored. Karaman and Frazzoli [17] showed that the RRG is asymptotically optimal unlike RRT. This means that the resulting path for an infinite amount of nodes is optimal while this is not guaranteed for RRT. This change is intended to remove the drawbacks from the previous iteration regarding the sub-optimal global path planner and the distance metric.

Local sampling is introduced which adds additional nodes in proximity to the robot. This should enhance local exploration and reduce the back-and-forth motion from the RRT-based approach. It is inspired by the works of Wang et al. [60], Denny et al. [61] and Gammell et al. [59] which adjust the sampling space to improve sample placement. Their approaches are introduced in Section 2.3.

In the upcoming sections, the RRG-based approach is detailed first, followed by the adapted path and distance calculation in the graph and the addition of local sampling. Furthermore, the proposed approach is compared to the first iteration and to versions of the algorithms RH-NBVP and AEP that have been adapted for UGVs.

6.1. Graph-Based Design

The change from RRT to RRG is implemented by adapting the algorithm, that constructs a tree, to build a graph instead. Therefore, it connects all nearby nodes with each other. Furthermore, the distance and path calculations require changes to find the optimal paths through the graph which is not as trivial as in a tree. Finally, local sampling enhances local exploration and is intended to reduce the total traveled distance.

Similar to RRT, all information regarding the graph is stored in a ROS message which is detailed in Section 5.1.2. Compared to the previous message, nodes no longer contain information about parent and children nodes but a list of references to all edges connecting them to other nodes. Furthermore, a list of edges is added to the message where each edge e holds references to the nodes it is connected to and its length.

6.1.1. Adapted Algorithm

The algorithm to construct the RRG $\mathcal{G} = (N, E)$ for exploration is shown in Algorithm 6.1. It requires the robot's position $\mathbf{p}_{robot}$, the maximum distance d_{max} the robot can

Algorithm 6.1: Rapidly-exploring Random Graph construction for exploration

Input: $V, \mathbf{p}_{robot}, d_{max}$

```
1: G ←initGraph(p_robot)
2: while not explorationFinished() do
3:     p_rand ←randomlySamplePoint(V)
4:     p_near ← findNearestNeighbour(G, p_rand)
5:     if p_new ←steerCollisionFree(V, p_rand, p_near) then
6:         N_d ← findNodesInRadius(G, p_new, d_max)
7:         N_c ← ∅
8:         for p_d ∈ N_d do
9:             if collisionFree(V, p_new, p_d) then
10:                N_c ← N_c ∪ p_d
11:            end if
12:        end for
13:        addNodeToGraph(G, p_new, N_c)
14:    end if
15: end while
```

move according to the input function explained in Section 3.1.2 and the world space V.

The sets of nodes N and edges E are initialized with the method initGraph which creates a root node at $\mathbf{p}_{robot}$ and sets $E \leftarrow \emptyset$. The remaining algorithm is executed while the explorationFinished method, which is explained in section 5.1.8, is not satisfied. First, $\mathbf{p}_{rand} \in V$ is generated and the node closest to it $\mathbf{p}_{near} \in N$ is determined.

Afterwards, the steerCollisionFree method, that is introduced in Section 5.1.1, is used to determine the position of a new node $\mathbf{p}_{new}$ based on the robot's input function. If it can be placed and connected to $\mathbf{p}_{near}$ with a minimum edge length of d_{min} and a maximum edge length of d_{max} without a collision, the iteration continues. Otherwise, the sampled position is discarded.

Then, all nodes $N_d \in N$ within the radius d_{max} around $\mathbf{p}_{rand}$ are identified. Every node that can be connected to $\mathbf{p}_{new}$ without a collision is added to the set N_c. The collisionFree method's collision check is described in Section 5.1.4. Finally, $\mathbf{p}_{new}$ is added to the graph with edges to all nodes in N_c.

Apart from switching from RRT to RRG as the structure for the nodes and connections, most of the implementation remains the same. The traversability analysis and the termination condition are the same. Only the traversability check for the corridor is repeated for every additional edge added to the RRG.

6.1.2. Distance and Path Calculation

The distance d_{rn} and the path P_{rn} between the node nearest to the robot n_r and n are still maintained and stored in the particular node. But due to the change from RRT to RRG, several adaptions are introduced to the calculation and the `RneGlobalPlanner`.

If n_r changes, all d_{rn} and P_{rn} are recalculated using Dijkstra's algorithm starting at n_r as described in Section 3.1.5. This approach's implementation uses a self-balancing binary search tree for the queue with the remaining nodes in Dijkstra's algorithm which results in a computational complexity of $\mathcal{O}(|E| \log(|N|))$.

(a) (b) (c)

Fig. 6.1.: The effect of local sampling on the sample density for the exploration of an exemplary environment is shown above with light gray areas indicating free space, dark gray unknown space and black occupied space. The robot can be seen as a black rectangle. The nodes and edges of the graph appear in blue and green. Image (a) shows the initially explored map. Images (b) and (c) depict the graph after 30 sec of exploration. In (c), local sampling is activated.

When a new node n_{new} is added to the graph, $d_{rn_{new}}$ and $P_{rn_{new}}$ are derived from n_{new}'s neighbor with the shortest d_{rn} similar to the first iteration. The edge and edge length to this neighbor are added to P_{rn} and d_{rn} respectively and assigned to n_{new}. Then, Dijkstra's algorithm is started from n_{new} but without resetting all other nodes' d_{rn} and P_{rn} first. Only nodes, whose d_{rn} is larger than that of the newly established connection, are therefore improved.

6.1.3. Local Sampling

The random sampling in V can be extended by local sampling around the robot to increase the sample density in its proximity. This enhances the local expansion of the graph, especially in large environments where new samples are unlikely to be placed close to the robot's location.

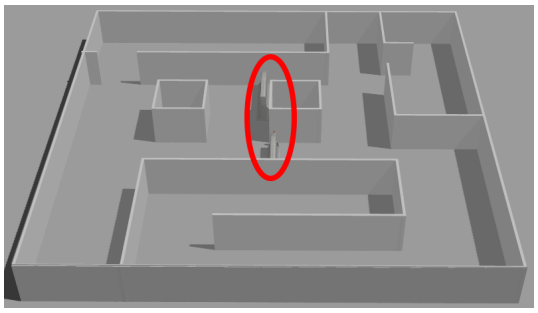
Additional samples are placed in a circular area with radius r_{ls} around the robot. The local sampling can be executed in addition to the sampling in all of V so that two nodes are added in each iteration of the RRG construction.

Figure 6.1 shows the effect of local sampling in an exemplary environment after 30 sec of exploration compared to no local sampling. The additional density and connectivity in the graph are visible while the outward expansion remains similar.

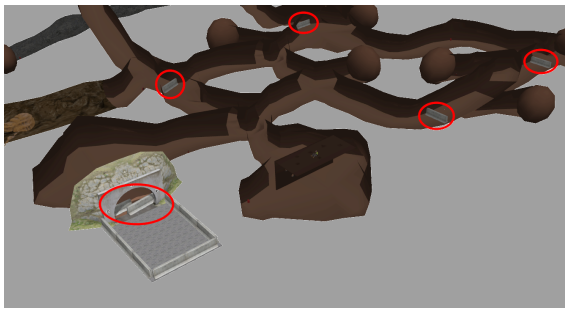
6.2. Comparison to Tree-Based Exploration

To show the increased efficiency of RRG with local sampling, RRG and RRT with and without local sampling are compared to each other. Local sampling is referenced as LS in the following figures and tables.

The simulation environment and computer for the comparison are the same as in Section 5.2 and the two utilized robot configurations in this experiment are similar as well. The first robot configuration is called VL which can be seen in Figure 5.8c and features a rotating Velodyne PUCK VLP-16 lidar. The second configuration is referenced as C. It looks the same as the RS configuration shown in Figure 5.8d but its Intel RealSense depth camera is rigidly attached and cannot be swiveled.



(a) Indoor simulation environment with dimensions of 25x25x2.5 m with an omitted roof and added barriers in the red circles.



(b) Cave simulation environment with dimensions of 60x90x30 m with added barriers in the red circles.

Fig. 6.2.: Gazebo simulation environments used for the comparison.

Figure 6.2a depicts the small and medium environments referenced as SE and ME respectively. SE is a small part on the left of ME that is separated by barriers which are removed in the ME scenario. Figure 6.2b shows a large underground cave environment referenced as CE that is modified from [158] to make untraversable areas inaccessible to the robot. All runs in SE and ME are limited to 30 min and runs in CE to 1 h.

All simulations use the following parameters: $t_{exit} = 10\text{s}$, $d_{min} = 1\text{m}$, $d_{max} = 2\text{m}$, $r_{ls} = 5\text{m}$, $\Delta_\varphi = \Delta_\vartheta = 10$ degrees, $\Delta_r = 0.1\text{m}$, $g_{min} = 0.05$ for VL in SE and ME and $g_{min} = 0.1$ for every other combination. The OctoMap resolution is $e_V = 0.1\text{m}$.

Four different combinations are executed in 5 variants with 10 runs each. The combinations are referenced as RNE which is RRG with LS, RRG, RRT with LS and RRT.

The variants are C-SE, C-ME, VL-SE, VL-ME and VL-CE. Runs in which the robot gets stuck during navigation are discarded. This is caused by insufficient localization due to erroneous odometry which leads to navigation planning too close to obstacles. A minimum of 7 valid runs is required for each combination and variant, otherwise failed runs are repeated.

The results of the comparison can be seen in Table 6.1 which shows that RRG is superior to RRT in every scenario regarding the duration and traveled path length. The mean duration of RNE compared to RRT+LS is decreased by up to 48.1% and the mean path length by up to 43.5% for VL-ME. The mapped volume is approximately equal throughout the runs but with a decrease for runs that ended prematurely because of the time limit. This causes the SD of 0 for VL-CE RRT+LS since all runs stop because of it.

RRT also has a higher SD for the duration and path length that is caused by the random tree structure in which the robot has to backtrack to reach different branches while the RRG's interconnected graph leads to more reliable and reproducible explorations.

LS improves the duration and path length for RNE and has the most significant impact on VL-CE. The advantage of RNE is more prominent in larger environments compared to small environments like C-SE where mean duration and distance are only decreased by up to 3% and 0.1% respectively.

Tab. 6.1.: Comparison between RRG with local sampling as RNE, RRG, RRT+LS and RRT showing the mean μ and SD σ of duration, traveled path length and mapped volume. The depth camera and lidar configurations as C and VL respectively are used in the small, medium and cave environments as SE, ME and CE respectively. The best mean values for each variant are printed in bold letters.

Configuration	Duration [s]		Path length [m]		Mapped volume [m^3]	
	μ	σ	μ	σ	μ	σ
C-SE RNE	**532.50**	87.73	**80.14**	11.79	**803.6**	14.1
C-SE RRG	549.00	58.40	80.22	8.51	799.8	15.6
C-SE RRT+LS	798.33	157.28	113.40	30.96	768.4	87.8
C-SE RRT	836.67	323.44	129.17	32.71	798.5	23.1
C-ME RNE	**1056.67**	66.29	**192.09**	9.71	1685.9	7.0
C-ME RRG	1116.67	42.50	202.31	17.83	**1692.6**	9.8
C-ME RRT+LS	1651.88	180.26	248.18	71.44	1551.8	256.0
C-ME RRT	1713.00	164.59	276.18	32.69	1480.8	270.6
VL-SE RNE	**270.00**	32.40	**45.09**	7.84	**928.2**	23.4
VL-SE RRG	319.50	57.51	57.11	8.37	913.9	24.8
VL-SE RRT+LS	370.50	76.58	72.55	10.16	910.8	26.5
VL-SE RRT	372.00	60.75	70.70	9.49	917.6	32.3
VL-ME RNE	**768.00**	93.49	**157.86**	28.09	1684.7	35.1
VL-ME RRG	850.50	102.18	211.15	38.54	**1702.5**	20.0
VL-ME RRT+LS	1480.50	293.15	279.56	36.59	1701.7	57.3
VL-ME RRT	1150.50	377.51	228.57	48.44	1690.9	58.1
VL-CE RNE	**2155.00**	199.42	**481.41**	55.97	10007.4	200.1
VL-CE RRG	2655.12	217.87	682.87	72.58	10083.9	255.5
VL-CE RRT+LS	3600.00	0.00	650.81	96.36	9946.6	505.1
VL-CE RRT	3551.67	145.00	897.58	82.35	**10296.8**	44.8

6.3. Comparison to State-of-the-Art Approaches

To compare the proposed approach to the current state-of-the-art, sampling-based approaches RH-NBVP and AEP, they are adapted to the previously presented robot configurations. RH-NBVP and AEP are detailed in Section 2.4. Since they are originally implemented for UAVs, the adaptions use this approach's STEER function and 2D sampling method without local sampling.

Furthermore, their existing gain functions are replaced with SRP and integrate this work's exit conditions g_{min} and t_{exit}. For RH-NBVP and AEP, t_{exit} replaces the maximum tries to find new samples. If the timer runs out, the current best node or frontier for AEP is designated as a goal. If there is no node with a minimum gain, the exploration terminates.

Because of these adaptions, the two approaches are referenced as RH-NBVP* and AEP* in the following. The same five variants as in the previous simulations are executed 10 times each. The RRT's maximum edge length for both is $l = 1m$, RH-NBVP*'s degression coefficient is set to $\lambda = 0.5$ and AEP*'s to $\lambda = 0.75$, its GCR threshold to $g_{zero} = 1$. The maximum and minimum amount of nodes are $N_{max} = 400, N = 30$ for ME and CE and $N_{max} = 200, N = 15$ for SE with RH-NBVP*. AEP*'s N is the same and N_{max} is only half of RH-NBVP*'s. These values are based

Tab. 6.2.: RNE, RH-NBVP* and AEP* simulation results with mean μ and SD σ of duration, traveled path length and mapped volume. The depth camera and lidar configurations as C and VL respectively are used in the small, medium and cave environments as SE, ME and CE respectively. The best mean values for each variant are printed in bold letters.

Configuration	Duration [s]		Path length [m]		Mapped volume [m^3]	
	μ	σ	μ	σ	μ	σ
C-SE RNE	**532.50**	87.73	**80.14**	11.79	**803.6**	14.1
C-SE RH-NBVP*	623.33	96.66	89.33	16.18	746.0	7.0
C-SE AEP*	658.50	100.26	100.61	14.28	756.0	12.7
C-ME RNE	**1056.67**	66.29	**192.09**	9.71	**1685.9**	7.0
C-ME RH-NBVP*	1782.86	45.36	250.79	63.73	1494.3	245.8
C-ME AEP*	1762.50	59.37	256.19	17.78	1575.0	87.9
VL-SE RNE	**270.00**	32.40	**45.09**	7.84	**928.2**	23.4
VL-SE RH-NBVP*	820.71	170.08	111.61	23.67	922.3	21.4
VL-SE AEP*	589.29	64.64	86.11	14.77	911.9	25.1
VL-ME RNE	**768.00**	93.49	**157.86**	28.09	**1684.7**	35.1
VL-ME RH-NBVP*	1786.88	24.63	234.72	5.63	1574.3	106.9
VL-ME AEP*	1386.43	346.07	180.43	74.59	1479.5	204.9
VL-CE RNE	**2155.00**	199.42	481.41	55.97	**10007.4**	200.1
VL-CE RH-NBVP*	3519.38	228.04	398.83	25.78	8471.9	837.9
VL-CE AEP*	3429.00	530.29	**387.44**	81.84	8934.4	1137.7

on [7].

Table 6.2 shows the results of RH-NBVP* and AEP*. The results for RNE from Table 6.1 are listed again for better comparability. Furthermore, Figure 6.3 displays the mean volume and the path length over time for RNE, AEP* and RH-NBVP* in the VL-CE variant as well as the OctoMap after 30 minutes of exploration for AEP* and RNE. It can be seen, that the continuously-built RRG with LS and the decoupled gain calculation lead to a vaster explored area of the map in less time while traveling shorter distances.

The proposed RNE achieves an increase in the mapped volume of 18.1% compared to RH-NBVP* and 12% to AEP* in the VL-CE variant while finishing the exploration in 38.8% and 37.1% less time respectively. The path lengths of RH-NBVP* and AEP* are shorter because of their slower pace compared to RNE and the time limit at which they are still not finished with the exploration. Even in the smaller C-SE variant, RNE decreases the duration by 14.6% compared to RH-NBVP* and 19.1% to AEP* as well as the distance by 10.3% and 20.3% respectively.

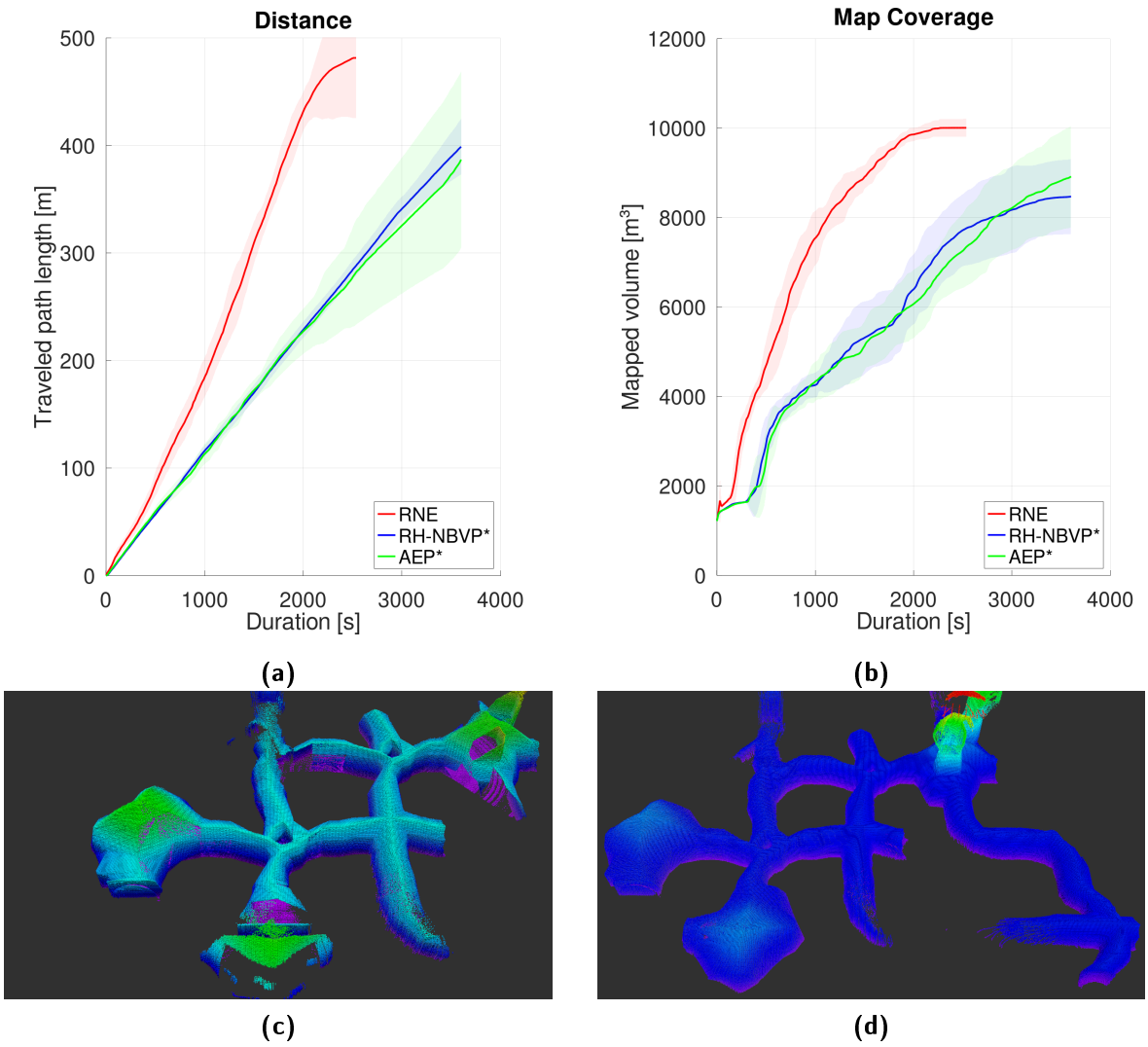


Fig. 6.3.: The mean mapped volume in (a) and path length in (b) are shown over time for the lidar configuration in the cave environment. The tinted areas depict the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. The recorded OctoMap after 30 minutes of exploring with AEP* can be seen in (c) and with RNE in (d). The OctoMap's voxels are colored according to their relative height.

6.4. Conclusion

In this chapter, the advantages of an RRG-based exploration including local sampling compared to an RRT-based approach are detailed. The comparison to the first iteration can be seen in Table 6.1 and shows that the disadvantages of the RRT-based implementation are eliminated. This second iteration's contributions are listed in the following:

1. The transition from RRT to RRG reduces the exploration duration and the path length significantly when using the proposed `RneGlobalPlanner` as a global planner in the ROS navigation stack.

2. Local sampling additionally decreases the duration and traversed path length of the exploration because it promotes local exploration and reduces back-and-forth

motion.

3. The second iteration of RNE is compared to RH-NBVP and AEP adapted to UGVs and demonstrates superior efficiency regarding duration, path length and mapped volume.

But for larger explorations, the number of nodes and edges can significantly grow. This also increases the required computation time to calculate all node gains and detect the optimal path and distance to each of them using Dijkstra's algorithm. To reduce this computational load, an approach to reduce the number of nodes and edges in the graph is proposed in the next chapter. This third iteration's goal is to keep the efficiency of the exploration on the same level.

The current iteration also produces zigzagging navigation paths due to the randomness of placed samples. A strategy to put samples and therefore paths closer to the center between obstacles is introduced in the next chapter as well. It is expected to straighten the paths, reduce the path length and speed up the exploration.

To enable adapting the exploration to different situations, the current reward function is improved in the next iteration. It allows customization and incorporates heading changes, traversability and more into the selection of the NBV.

7. RNE 3 - Topology-Based Exploration

The previous chapter presented the second iteration of RNE which introduced an RRG-based exploration to optimize paths from one node to another along the graph's edges. It added a local sampling in an area around the robot to increase the sample density near it. These changes allow RNE to outperform the state-of-the-art approaches RH-NBVP and AEP which were adapted for usage with UGVs.

In this chapter, the previously introduced, RRG-based exploration is further improved by decreasing the number of nodes and edges in the graph. The goal is to reduce the computation time while retaining a similar level of exploration efficiency.

The traversable area around each node is incrementally inflated and checked for obstacles up to a user-defined maximum radius which should be equal to or less than the robot's sensor range. These inflated nodes have a higher safety margin to obstacles and increase the amount of explorable unknown space from them compared to nodes close to obstacles which block the robot's vision.

This Node Area Inflation (NAI) is inspired by related works from Brock and Kavraki [54], Rickert et al. [55], Shkolnik and Tedrake [56] and Gao and Shen [57, 58] which are described in Section 2.3.

Since the inflation process is most useful when nodes are placed in the center of hallways, rooms or drifts, a topology-based NAI is proposed. It pushes nodes away from obstacles, unknown space and other nodes encountered during the inflation process to increase their respective radius.

Furthermore, a revised cost function is presented which can be customized by user-defined factors and takes gain, distance, traversability, heading and node radius into consideration. Another method to update node rewards during the robot's movement towards a goal node is introduced as well. This enables replacing the current goal node with a better node if its state changes to *explored* during the approach of the robot. An increased exploration efficiency at the cost of more computation time is expected from this addition.

NAI is described first, followed by the method to move nodes away from obstacles during the inflation process. Then, the new reward function and a heuristic to update node gains before reaching a goal are presented with an extensive evaluation. Finally, the inflation process and new reward function are compared to the previous iteration of RNE.

7.1. Node Area Inflation

NAI is introduced to increase the safety margin between the robot and obstacles. It also enhances the amount of observable unknown space for a particular node due to less occlusion by nearby obstacles which is shown in Figure 7.1.

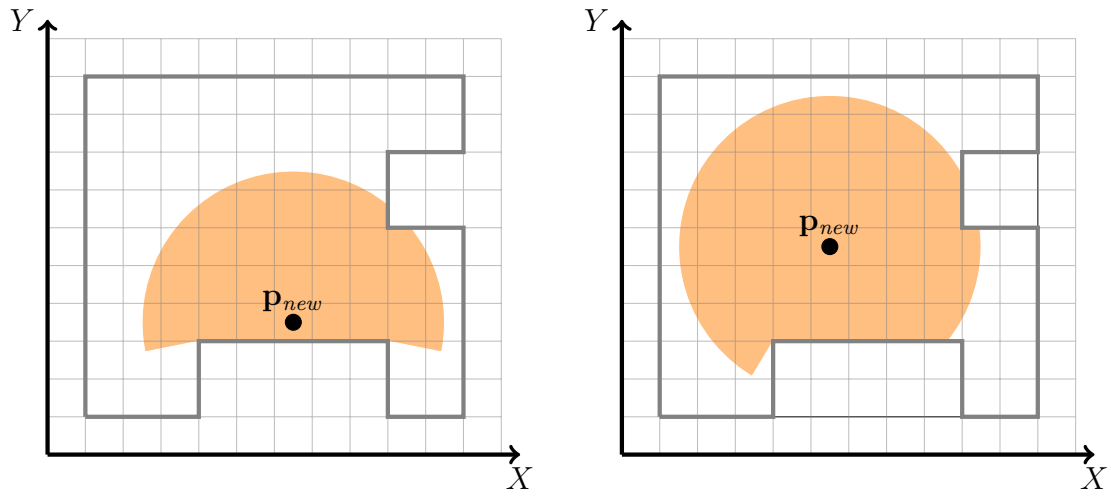


Fig. 7.1.: The images show the placement of a new node $\mathbf{p}_{new}$. The thick dark gray lines depict solid walls while the orange area is the FoV from the particular node. It is blocked by the walls and limited to a maximum range. It can be seen that the placement further away from obstacles in the right image enables a much greater observable space than in the left image.

The new node placement and the NAI's traversability check, which are described in the following sub-sections, replace the input function introduced in Section 3.1.2 and the traversability check detailed in Section 5.1.4.

7.1.1. Node Placement

The previous input function determines the position of a new node $\mathbf{p}_{new}$ based on the sampled point $\mathbf{p}_{rand}$ and the nearest neighbor in the existing graph $\mathbf{p}_{near}$. $\mathbf{p}_{new}$ is placed on a ray from $\mathbf{p}_{near}$ through $\mathbf{p}_{rand}$ on which it has to be positioned below a maximum distance and above a minimum distance from $\mathbf{p}_{near}$.

For NAI, this variable distance is removed. Instead, new nodes are placed directly next to the nearest node in the graph $\mathcal{G}$ with an intersecting area that has a width equal to the robot's width w_{robot}. This means no rectangular traversability checks have to be conducted anymore as can be seen in Figure 7.2.

The distance to the nearest node d_{infl} to place a new node is based on Equation (5.4) and can be seen in Equation (7.1) where r_{near} is the radius of $\mathbf{p}_{near}$. Figure 7.2 shows this distance between a new node $\mathbf{p}_{new}$ and its nearest node $\mathbf{p}_{near}$ in $\mathcal{G}$.

$$d_{infl} = \sqrt{\frac{r_{robot}^2 + r_{near}^2}{2} - \left(\frac{w_{robot}}{2}\right)^2} \tag{7.1}$$

The desired distance d_{infl} to place $\mathbf{p}_{new}$ is chosen to ensure that the distance between both intersection points of the node areas of $\mathbf{p}_{near}$ and $\mathbf{p}_{new}$ equals w_{robot}. This assures that there is a traversable corridor between both nodes that is wide enough for the robot to fit through.

Before this new input function is executed, the nearest neighbor must be determined. This is not as simple as using the radius search from a k-d tree as described in Section 3.1.6 because the relevant distance is not the distance between $\mathbf{p}_{near}$ and $\mathbf{p}_{rand}$ but between the area boundaries of the two nodes. While $\mathbf{p}_{rand}$ is initialized with a radius of r_{robot}, the radius of $\mathbf{p}_{near}$ can be significantly larger.

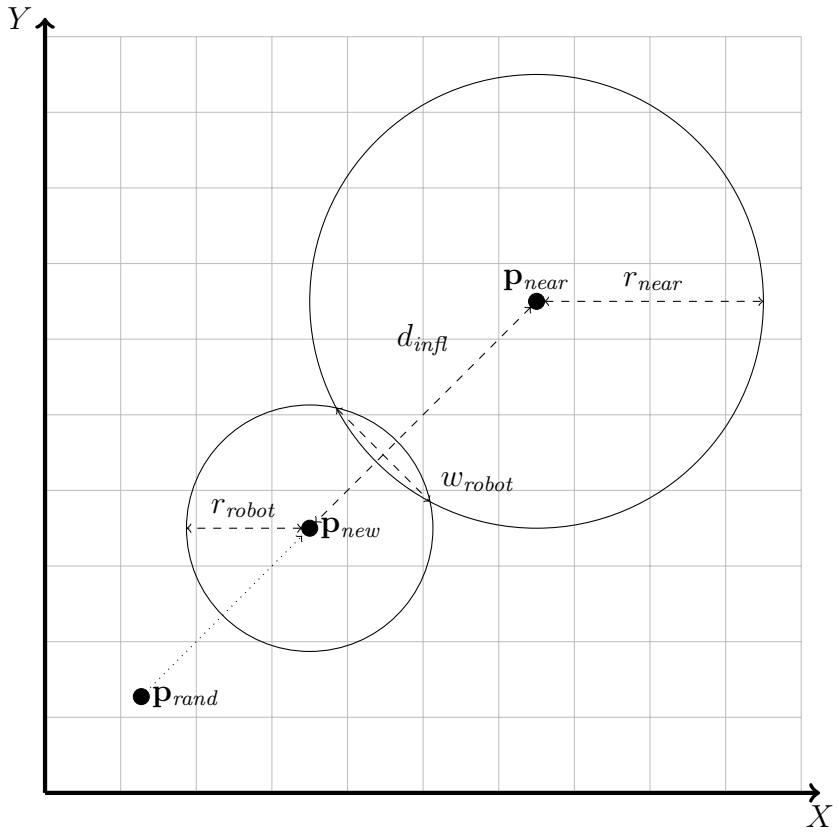


Fig. 7.2.: The placement of a new sample $\mathbf{p}_{rand}$ which is moved to $\mathbf{p}_{new}$ next to its nearest neighbor $\mathbf{p}_{near}$. The calculated distance d_{infl} between $\mathbf{p}_{new}$ and $\mathbf{p}_{near}$ assures that the robot can safely traverse from one node to the other because of the width of the intersection area that must be equal to w_{robot}.

To find the nearest neighbor, an exhaustive search over all nodes in $\mathcal{G}$ has to be conducted where r_{near} is subtracted from the distance between $\mathbf{p}_{near}$ and $\mathbf{p}_{rand}$. The node with the smallest resulting distance is the nearest neighbor in $\mathcal{G}$. If this distance is below zero, $\mathbf{p}_{rand}$ is discarded as it lies inside an existing node's area.

Because the node areas are expected to be small compared to the total exploration area, the number of nodes to be checked for the nearest neighbor search can be reduced by the following heuristic. The largest node radius that is currently present in $\mathcal{G}$ is stored and its sum with the initial node radius r_{robot} is used for a radius search in the k-d tree. Only the nodes returned by this radius search must be checked as all other nodes are too far away from $\mathbf{p}_{rand}$. If the radius search returns no nodes, the nearest node given by the k-d tree is used as $\mathbf{p}_{near}$, even though it might not be the closest node. This is intended to reduce the computation time for large graphs.

7.1.2. Traversability Check

After the node has been placed and passed its traversability check for the circle with radius r_{robot}, incrementally growing rings around the node are analyzed and added to the node radius, if they are traversable. The radius is incremented by the grid map's resolution in each iteration. New nodes are aligned to the grid map as described in Section 5.1.4. This alignment is not detailed here for simplicity.

To determine which grid map cells to check and to prevent checking cells repeatedly,

Algorithm 7.1: Calculation of increasing circle offsets for traversability assessment

procedure CALCULATENEXTINFLATEDCIRCLEOFFSETS(O_c, O, R)
  $j \leftarrow |O|$
  if $j = 0$ **or** $r_j + r_m \leq r_{max}$ **then**
    $O_p \leftarrow \emptyset$
    if $O = \emptyset$ **then**
      $r_{j+1} \leftarrow \left(\lceil \frac{r_{robot}}{r_m} \rceil + 1\right) \cdot r_m$
      $O_p \leftarrow O_c$
    else
      $r_{j+1} \leftarrow r_j + r_m$
      $O_p \leftarrow O_j$
    end if
    $O_{j+1} \leftarrow$PRECALCULATECIRCLEOFFSETS(r_{j+1})
    for $o \leftarrow 1$ **to** $|O_p|$ **do**
      $x^m_{start} \leftarrow x^m_{p_o} + 1,\ x^m_{p_o} \in O_{p_o}$
      $O_{j+1_o} \leftarrow O_{j+1_o} \cup x^m_{start}$
    end for
  end if
  return O, R
end procedure

the offsets for each ring are calculated using Algorithm 7.1. It requires the offsets O_c for the circle with the robot's radius. O_c is calculated using Algorithm 5.2. Also, the already existing ring's offsets O and the set of corresponding radii R are required.

Algorithm 7.1 calculates the next inflation ring's offsets and radius until the maximum node radius is reached. For this radius, the sensor range r_{max} is used. Therefore, the radius of the largest checked inflation ring r_j plus the grid cell size r_m is compared to r_{max} to determine if another inflation ring's offsets must be calculated.

Then, the new ring's radius r_{j+1} and the previous ring's offsets O_p are determined. If no inflation ring offsets have been calculated before, its radius is derived from the robot radius in grid cells plus one and O_p is set to O_c. Otherwise, r_{j+1} is based on the previous ring's radius r_j plus r_m and the previous ring's offsets are assigned to O_p.

The next ring's offsets O_{j+1} are calculated using Algorithm 5.2 with radius r_{j+1} instead of r_{robot}. Then, for every y-offset, a start value x^m_{start} for the x-axis is determined from O_p and added to the set. An exemplary application of this algorithm can be seen in Figure 7.3.

Algorithm 7.2 applies the previously calculated ring offsets O and analyzes if a ring at index j at the position p_{new} is traversable. For all offsets in O_j, the lines derived from it are checked in the grid map using Algorithm 5.4.

For every offset apart from the set referencing the circle's center, four lines are constructed and checked using the y-offset y^m_j, the x-offset x^m_j and the x-axis start offset $x^m_{start_j}$.

Algorithm 7.2: Check traversability of a ring

```
1:  procedure isRingTraversable(p_new, O, j)
2:      p^m_new := (x^m_new, y^m_new) ←worldToOccupancyGrid(p_new)
3:      for each J ∈ O_j, J := {x^m_j, y^m_j, x^m_start_j} do
4:          if not isLineTraversable(x^m_new + x^m_start_j, x^m_new + x^m_j, y^m_new + y^m_j) or
5:           not isLineTraversable(x^m_new − x^m_start_j, x^m_new − x^m_j, y^m_new + y^m_j) then
6:              return false
7:          end if
8:          if y^m_j > 0 then
9:              if not isLineTraversable(x^m_new + x^m_start_j, x^m_new + x^m_j, y^m_new − y^m_j) or
10:              not isLineTraversable(x^m_new − x^m_start_j, x^m_new − x^m_j, y^m_new − y^m_j) then
11:                 return false
12:             end if
13:         end if
14:     end for
15:     return true
16: end procedure
```

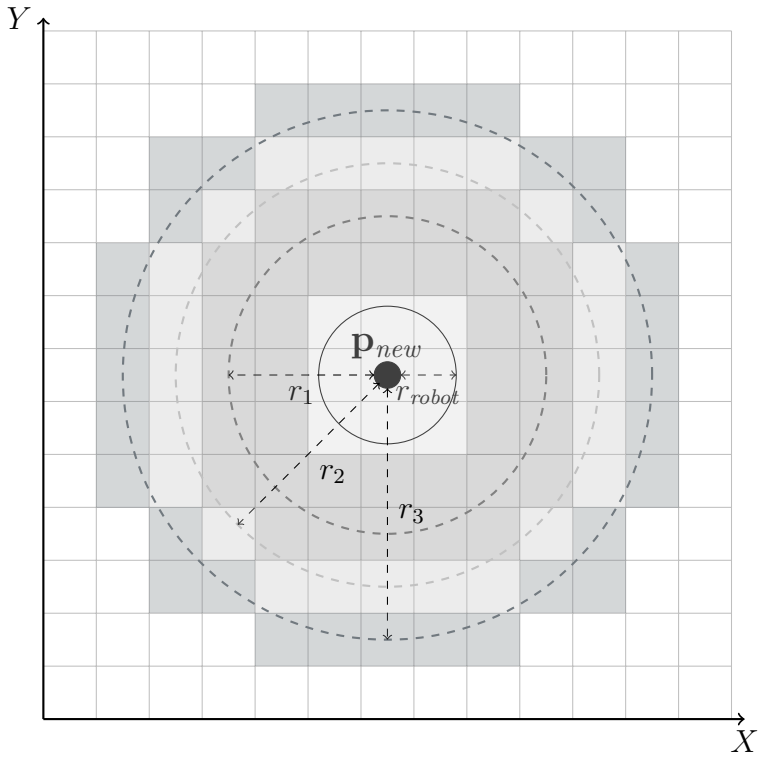


Fig. 7.3.: This image shows $\mathbf{p}_{new}$ and its initial node radius r_{robot} with the circle's area that is checked in the grid map for traversability tinted in light gray. The dashed circles around it and the grid map cells colored in matching colors depict the different NAI steps 1 to 3. The circles have the radii $r_1, r_2, r_3 \in R$ and the marked grid map cells are defined by the corresponding offsets O.

7.1.3. Application of Node Area Inflation

After the a new random sample is placed with a large enough intersection to the nearest node in the graph and a circle with radius r_{robot} at the potential new node's position passed its traversability check, NAI starts.

The offsets for the smallest ring radius are selected from O and used with Algorithm 7.2. If no collision is detected, the inflation process continues. Otherwise, it is stopped and the node's radius remains equal to r_{robot}.

In each following iteration, the next ring's offsets and respective radius are used as an input for Algorithm 7.2. A failed traversability analysis results in canceling the inflation process and the node's radius is set to the previous iteration's radius.

If the offsets for a new inflation step do not exist yet, Algorithm 7.1 is called to calculate them and the corresponding radius r.

If NAI fails due to unknown grid map cells, this is stored in the node. When the particular node is updated because of the exploration of a nearby node, it is checked if a previous inflation process failed due to unknown cells. In this case, the inflation process is continued and it is checked, if new connections to neighbor nodes become available because of it.

7.1.4. Connection to Neighbor Nodes

After NAI, the edges to all connectable nodes must be found and added to the graph. All nodes in a radius around the newly placed node are retrieved by using a radius search in the k-d tree. This radius is defined as the node's radius plus the largest node radius in the graph.

For every retrieved node $\mathbf{p}_n$, it is checked if the distance between $\mathbf{p}_n$ and the new node $\mathbf{p}_{new}$ is equal or less than d_{infl}. If it is, an edge between $\mathbf{p}_n$ and $\mathbf{p}_{new}$ is added to $\mathcal{G}$. d_{infl} is defined in Equation (7.1) and describes the minimum required distance between two nodes with radius r_{robot} for $\mathbf{p}_{new}$ and r_{near} for $\mathbf{p}_n$. This assures that the intersection between both node areas has a width of w_{robot} and guarantees the possibility of a safe passage between both nodes for the robot.

Finally, Dijkstra's algorithm is run as described in Section 6.1.2. It is started at the new node and the other node's distances are not reinitialized. Therefore, only better connections that became available because of the new node, are updated.

7.2. Topology-Based Node Area Inflation

To enhance the placement of nodes, a topology-based NAI process is proposed. It allows larger NAI and therefore bigger safety margins to obstacles and more suitable view distances to objects for the sensors.

Therefore, collisions occurring during NAI no longer immediately stop the inflation process but are used to deflect the node's position in the opposite direction. After moving the node, the inflation process is continued until the next collision occurs which repeats the previous steps. This causes nodes to move away from obstacles and unknown cells during the inflation process and migrate towards the center of free space. This can be seen in Figure 7.4

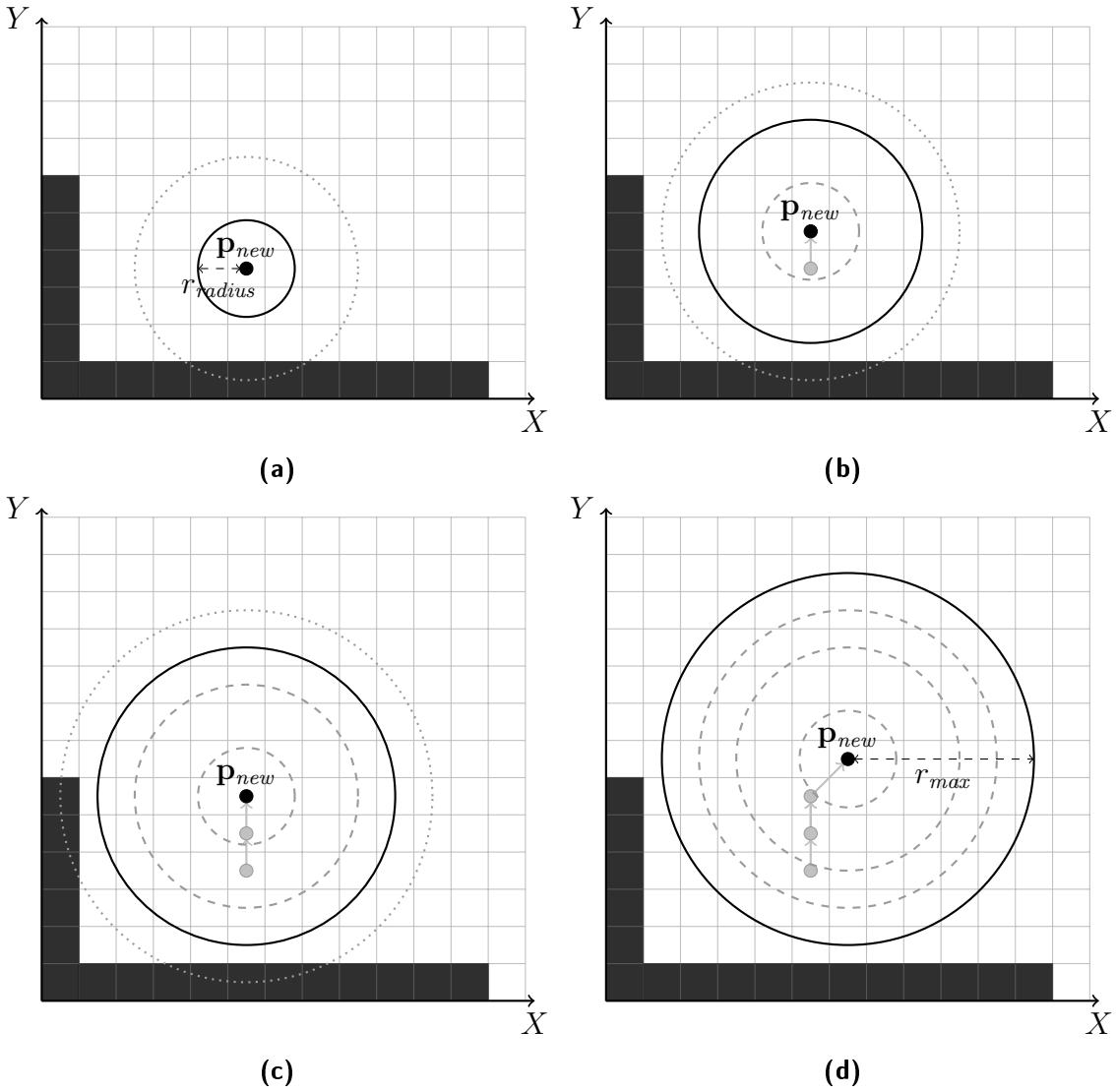


Fig. 7.4.: NAI with a new node $\mathbf{p}_{new}$ which is moved away from obstacles, that are depicted as dark gray cells, is shown in the images from (a) to (d). If $\mathbf{p}_{new}$'s inflated area displayed as a dotted gray circle collides with an obstacle, it is moved in the opposite direction in the following image. The current inflation radius at its current position is shown as a black circle while previous inflation steps are displayed as dashed gray circles. The previous node positions can be seen as gray dots. The process is stopped at image (d) because the inflation area reached the maximum range of the robot's sensor r_{max}.

If there are collisions in multiple contradicting directions, the node cannot be moved and the inflation process is terminated. Similar directions can be combined to adjust the movement of the node.

Furthermore, the node should move away from its nearest neighbor node to avoid being encircled by it or encircling it. For each inflation step, it is checked if the node has to move in the opposite direction from its neighbor node to maintain the distance d_{infl} between them.

7.2.1. Node Area Inflation with Moving Nodes

Algorithm 7.3 describes the process of inflating a node area including the movement away from obstacles and the nearest neighbor. It requires the aligned, newly sampled node's position $\mathbf{p}_{new}$ for which the circular offsets O_c with radius r_{robot} have already been checked using Algorithm 5.3. Furthermore, $\mathbf{p}_{new}$'s nearest node $\mathbf{p}_{near}$ and the existing offsets O and radii R for the inflation process are provided.

The inflation radius r_{new} is initialized with r_{robot} and the iterator j over the inflation offsets is set to one. The set of previous positions P_{prev} consists only of $\mathbf{p}_{new}$ at the start.

The movement direction $\mathcal{D}$ is a tuple containing the direction dir_d and the information if the direction has already been merged dir_m. dir_d can have one of the following states:

center: The node remains stationary and is not moved. This indicates that no collision with obstacles or unknown space occurred and that the neighboring node's distance to $\mathbf{p}_{new}$ is in bounds.

none: Multiple contradictory collisions are detected and there is no direction to move to. The inflation process must be stopped.

Algorithm 7.3: Inflate node area and move it away from obstacles

```
1:  procedure INFLATENODEAREA(p_new, p_near, O, R)
2:      r_new ← r_robot
3:      j ← 1
4:      P_prev ← {p_new}
5:      D := {dir_d, dir_m} ← {none, false}
6:      while r_new ≤ r_max do
7:          if j > |R| then
8:              O, R ←CALCULATENEXTINFLATEDCIRCLEOFFSETS(O_c, O, R)
9:          end if
10:         D ←ISRINGTRAVERSABLE(p_new, O, j)
11:         if dir_d = none then
12:             return r_new
13:         else
14:             if dir_d = center then
15:                 r_new ← r_j
16:                 j ← j + 1
17:             else if not CHECKMOVEMENTWITHNEARESTNODE(p_new, p_near, D)
18:              or not MOVENODE(p_new, dir_d, P_prev) then
19:                 return r_new
20:             end if
21:         end if
22:     end while
23:     return r_new
24: end procedure
```

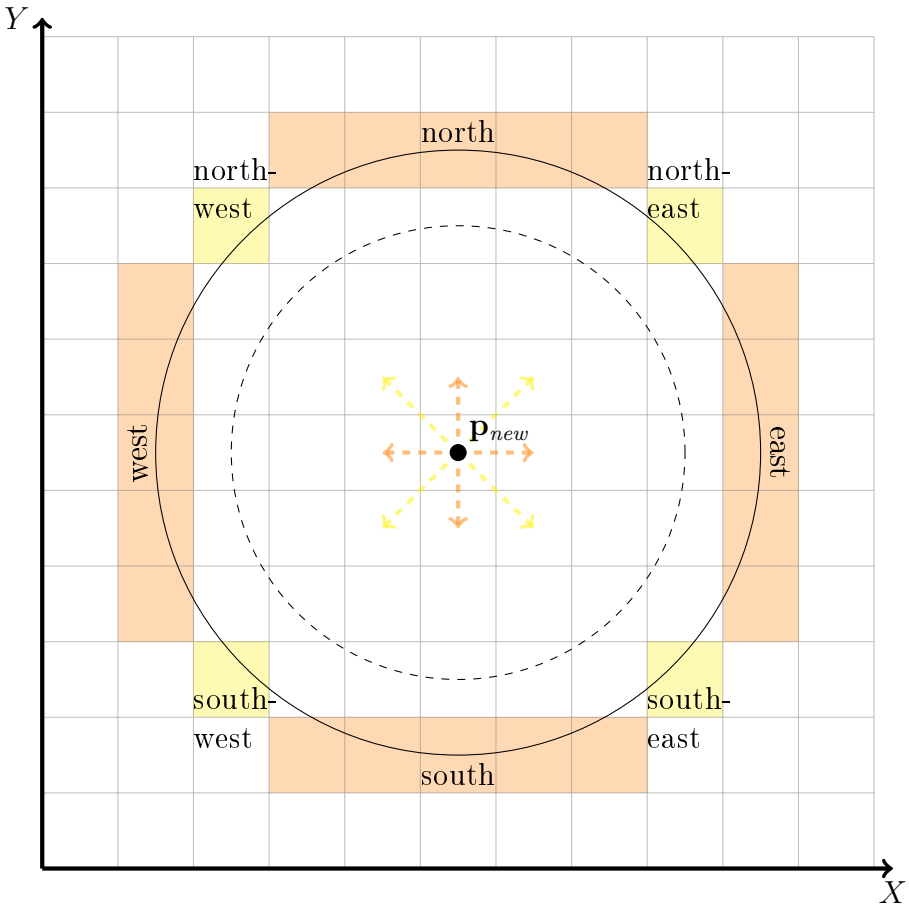


Fig. 7.5.: Node n_{new}'s radius is inflated from the dashed circle to the solid circle at its position $\mathbf{p}_{new}$. The shaded grid map tiles mark the areas that are checked for collisions. If a collision with an obstacle or unknown space is detected, the opposite direction is returned as the movement direction. For example, an obstacle in the southeast cell leads to an evasion in northwest direction.

e.g. *north*: One of the eight directions shown in Figure 7.5 that point towards the particular grid cell which is adjacent to the grid cell $\mathbf{p}_{new}$ is located at.

The algorithm loops over all inflation rings until the maximum radius r_{max} is reached. Before Algorithm 7.2 is called, it is checked if there are further entries in R for the iterator j to proceed. If there are not, the next inflation ring is calculated using Algorithm 7.1 and added to O and R.

For movable nodes, Algorithm 7.2 is adjusted to return the direction in which the node has to be moved in case of a collision with occupied or unknown grid cells. As can be seen in Figure 7.5, the node is moved in the opposite direction of the detected collision. If multiple collisions are detected, their directions are merged using Algorithm 7.4 which is explained in Section 7.2.2. The returned direction is stored in $\mathcal{D}$.

If there is no direction to evade the collisions, the algorithm terminates and returns the current radius r_{new} up to which the node area has been inflated. If the node does not need to be moved, r_{new} is increased to the next inflation radius $r_j \in R$ and the iterator j is incremented.

Otherwise, Algorithms 7.5 and 7.6 are called to assess if the proposed movement direction can be applied to the node. If one of these two algorithms returns *false*, the inflation process is terminated as well.

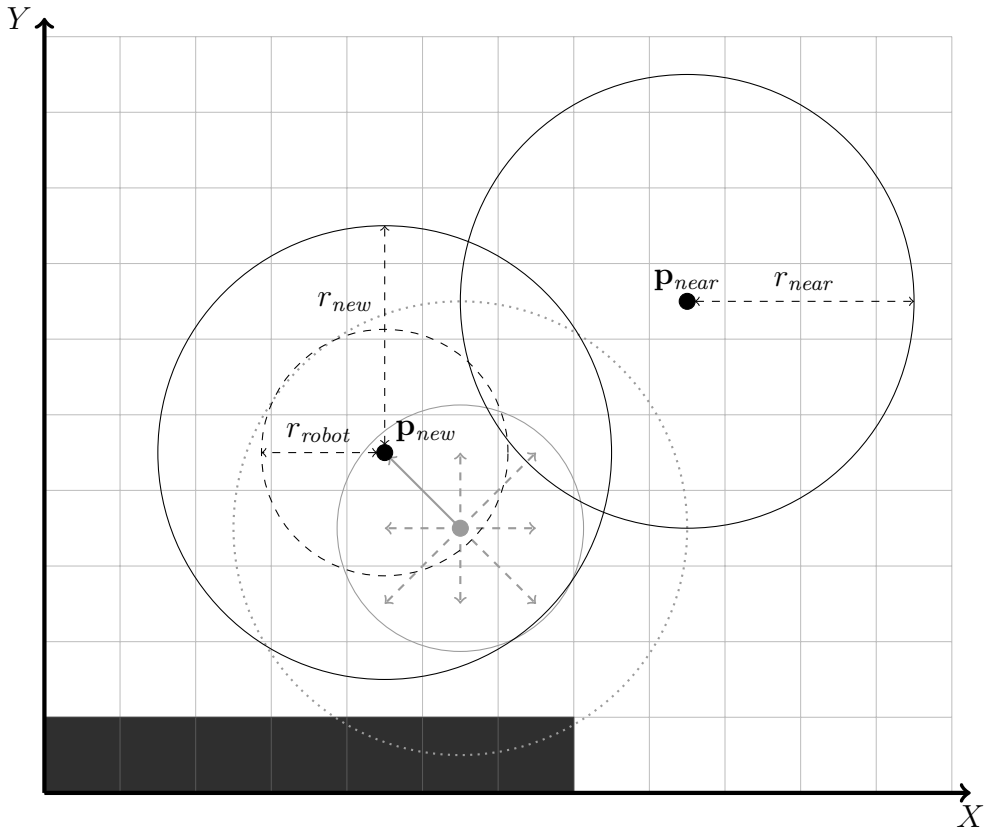


Fig. 7.6.: The new node $\mathbf{p}_{new}$ is in collision with the dark gray obstacle after being inflated, which is shown as a dotted circle, and therefore has to be moved north. Due to an increasing proximity with its nearest neighbor node $\mathbf{p}_{near}$, the moving direction is adjusted to northeast to avoid obstacle and $\mathbf{p}_{near}$. The dashed arrows around $\mathbf{p}_{new}$'s former position indicate the available directions for its movement. The chosen direction is drawn as a solid arrow. The dashed black circle shows $\mathbf{p}_{new}$'s initial radius and the solid black circle indicates the inflated area.

Figure 7.6 illustrates the concept of the repulsion from occupied cells and the neighbor node applied to a newly placed node $\mathbf{p}_{new}$.

7.2.2. Handling Multiple Collisions

When multiple collisions with occupied or unknown grid cells are detected or the new node $\mathbf{p}_{new}$ moves too far away from or too close to its neighbor node $\mathbf{p}_{near}$, multiple directions to move the node to are found. Depending on these directions, they either can be merged or lead to the termination of the inflation process.

In Algorithm 7.4, the tuple $\mathcal{D}$ which is outlined in Section 7.2.1 and a new direction d_{new} are merged. The algorithm returns *true* if the merge is successful and *false* if it fails. Furthermore, $\mathcal{D}$ is changed to the merged direction when it is successful.

First, it is checked if dir_d equals *center*. If it does, it is overwritten with dir_{new} and the merge completes successfully.

Otherwise, the difference dir_{diff} between dir_d and dir_{new} is calculated which is the absolute difference between both directions in degrees. Because all directions are multiples of 45 degrees and the maximum difference in a circle is 180 degrees, dir_{diff} is limited to the numbers listed in the algorithm.

If dir_{diff} equals zero, the merge is successful as both directions are the same. In

Algorithm 7.4: Merge two directions of node movement

```
1: procedure MERGEDIRECTION(D := {dir_d, dir_m}, dir_new)
2:     if dir_d = center then
3:         dir_d ← dir_new
4:     else
5:         dir_diff ← |dir_d − dir_new|, dir_diff ∈ 0, 45, 90, 135, 180
6:         if dir_diff = 0 then
7:             return true
8:         else if dir_m and dir_diff ≥ 90 or dir_diff ≥ 135 then
9:             return false
10:        else
11:            dir_d ←COMBINEDIRECTIONS(D, dir_new, dir_diff)
12:            dir_m ← true
13:        end if
14:    end if
15:    return true
16: end procedure
```

case of a previous merge dir_m and a difference of greater equals 90 degrees or just a difference of greater equals 135 degrees, merging fails.

Otherwise, the directions are combined based on Equation (7.2). The first two cases describe merging two directions with a difference of 45 degrees. In this case, the diagonal direction in the grid map is kept. All other cases represent 90 degrees differences where the intermediate direction is stored in dir_d. It does not matter which of the variables has which direction for these cases.

$$dir_d = \begin{cases} dir_d & dir_{diff} = 45 \text{ and } dir_d \bmod 90 \neq 0 \\ dir_{new} & dir_{diff} = 45 \text{ and } dir_d \bmod 90 = 0 \\ north & \{dir_d, dir_{new}\} = \{northeast, northwest\} \\ east & \{dir_d, dir_{new}\} = \{northeast, southeast\} \\ south & \{dir_d, dir_{new}\} = \{southeast, southwest\} \\ west & \{dir_d, dir_{new}\} = \{southwest, northwest\} \\ northeast & \{dir_d, dir_{new}\} = \{north, east\} \\ northwest & \{dir_d, dir_{new}\} = \{north, west\} \\ southeast & \{dir_d, dir_{new}\} = \{south, east\} \\ southwest & \{dir_d, dir_{new}\} = \{south, west\} \end{cases} \tag{7.2}$$

Finally, dir_m is set to *true*. If the merge did not terminate before, it returns its success.

7.2.3. Moving Nodes

Before a new node $\mathbf{p}_{new}$ is moved during the inflation process, it is assessed if a movement in the proposed direction is acceptable. This regards the distance to its nearest neighbor $\mathbf{p}_{near}$ in the graph and if it has been moved to a new and traversable position.

Algorithm 7.5: Assess node movement regarding the nearest neighbor

```
1: procedure checkMovementWithNearestNode(p_new, p_near, D)
2:     p_moved ←movePoint(p_new, dir_d)
3:     if D(p_moved, p_near) − r_near < 0 or D(p_moved, p_near) > d_infl then
4:         dir_n ←getDirection(p_moved, p_near)
5:         if not mergeDirection(dir_n, D) then
6:             return false
7:         end if
8:         p_merged ←movePoint(p_moved, dir_d)
9:         if D(p_merged, p_near) − r_near < 0 or D(p_merged, p_near) > d_infl then
10:            return false
11:        end if
12:    end if
13:    return true
14: end procedure
```

First, Algorithm 7.5 simulates the movement of $\mathbf{p}_{new}$ in the given direction dir_d to a new position. The method movePoint returns the coordinates of $\mathbf{p}_{new}$'s adjacent grid cell center $\mathbf{p}_{moved}$ according to dir_d. The Euclidean distance between $\mathbf{p}_{moved}$ and $\mathbf{p}_{near}$ is calculated and it is checked if $\mathbf{p}_{moved}$ is inside the nearest node's radius r_{near} or outside the distance d_{infl} from Equation (7.1). d_{infl} is the distance at which both nodes can be connected with a safe width of the intersection area. If one is *true*, the node must not be moved. Otherwise, this algorithm returns *true* because the movement violates no constraints.

The direction dir_n, in which the node moves, is acquired from the method getDirection. It calculates the arctan between $\mathbf{p}_{moved}$ and $\mathbf{p}_{near}$ and rounds it to the nearest of the eight cardinal directions shown in Figure 7.5. The opposite direction is chosen if $\mathbf{p}_{moved}$ must move closer to $\mathbf{p}_{near}$ instead of away from it.

Afterwards, Algorithm 7.4 is used to merge new and given directions. If it fails, the given movement is not allowed regarding the distance to the nearest node $\mathbf{p}_{near}$ and the inflation process must be terminated. Otherwise, the merged direction is stored in dir_d.

A new position $\mathbf{p}_{merged}$ is then simulated for the merged direction and it is checked if the distance between $\mathbf{p}_{merged}$ and $\mathbf{p}_{near}$ violates the distance boundaries. If $\mathbf{p}_{merged}$ is not inside r_{near} and the distance is equal to or less than d_{infl}, the node movement can be conducted. Otherwise, it fails and the inflation process stops.

This assessment's goal is to add nodes to the graph which are connectable to at least the nearest node due to the distance constraint d_{infl}. Furthermore, it prevents newly sampled nodes from collapsing into the position of already existing nodes because of the required minimum distance larger than the nearest node's radius r_{near}.

Algorithm 7.6 defines the process of moving the node $\mathbf{p}_{new}$ in direction dir_d while regarding the previous node positions P_{prev}. After $\mathbf{p}_{new}$ has been moved to a new position $\mathbf{p}_{moved}$, which is the center of an adjacent grid map cell in the direction of dir_d, it is checked if $\mathbf{p}_{moved}$ already has been moved to this position before.

In this case, the algorithm fails and the inflation process is stopped as $\mathbf{p}_{new}$ is moved back and forth. Otherwise, the new position is stored in $\mathbf{p}_{new}$ and it is added to P_{prev}.

Algorithm 7.6: Move node away from obstacles

```
1: procedure MOVENODE(p_new, dir_d, P_prev)
2:     p_moved ←MOVEPOINT(p_new, dir_d)
3:     if p_moved ∉ P_prev then
4:         p_new ← p_moved
5:         P_prev ← P_prev ∪ p_new
6:         return true
7:     end if
8:     return false
9: end procedure
```

7.2.4. Adding Nodes to the Graph

When the inflation process finishes, node $\mathbf{p}_{new}$ is added to the graph $\mathcal{G}$ and connected to nearby nodes using the method described in Section 7.1.4. Nearby nodes are found using a radius search and edges to them are added depending on the distance d_{infl} introduced in Equation (7.1).

Afterwards, Dijkstra's algorithm is executed to update the node's distances and paths to the robot as described in Section 6.1.2.

7.3. Topology-Based Node Area Inflation Conclusion

In conclusion, Figure 7.7 shows the difference between the different strategies of basic RRG, additional NAI and topology-based NAI. The exemplary comparison highlights the improved overall distances of node centers to obstacles when using the inflation process. The movement of nodes away from obstacles also leads to a topology-based graph and a significant reduction of nodes compared to the basic RRG approach while maintaining a similar area coverage.

Furthermore, the topology-based graph has far less zigzagging edges between the nodes which should speed up the exploration due to straighter and shorter paths.

Another option to reduce zigzagging deployed by state-of-the-art approaches like TARE, which is introduced in Section 2.4.2, is path smoothing. But this is often only used on the already selected path to an exploration goal which would deviate from the traversability-checked path in this work. The topology-based NAI furthermore incorporates the straightened path in the cost function to enable a more realistic exploration goal selection.

A thorough evaluation is undertaken in Section 7.7. This comparison emphasizes the different layouts of the graphs built due to the varying strategies. To more adequately take the different topologies into consideration, the cost function is revised in the next section to also regard other metrics besides the distance.

7.4. Revised Reward Function

In this section, an enhanced reward function is introduced which replaces the previous, simpler function shown in Section 5.1.6. It is based on the RH-NBVP's cost function

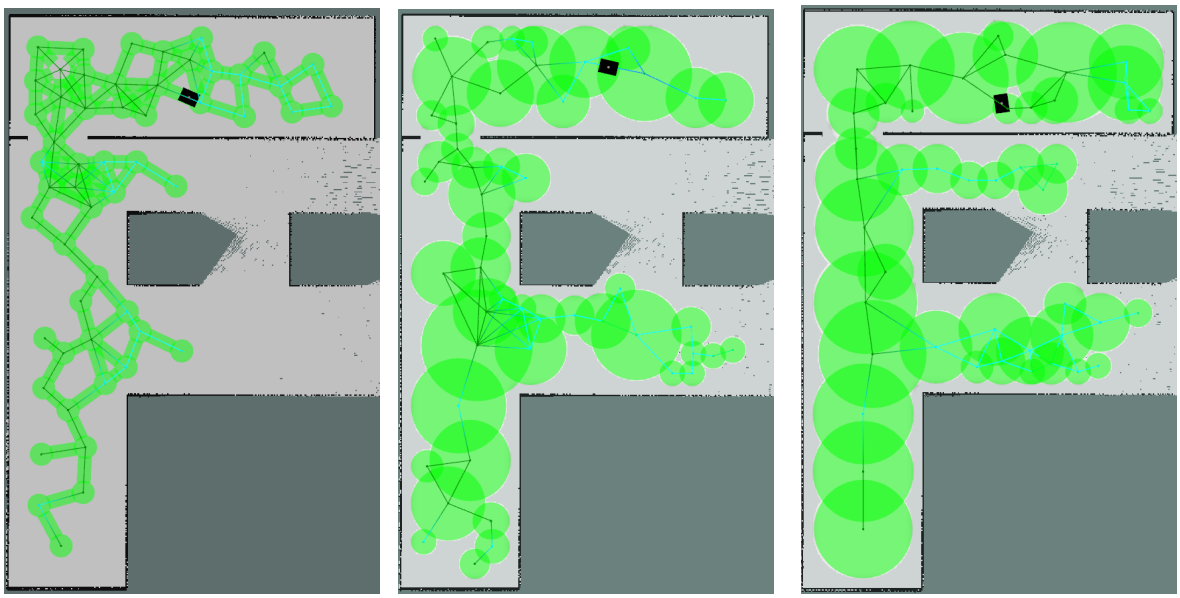


Fig. 7.7.: Comparison of an exemplary node placement between RRG (a), NAI (b) and topology-based NAI (c) after 30 sec of exploration in the medium environment shown in Figure 6.2a. Green circular and rectangular areas show traversable space identified by the proposed approach. Dark green and blue lines and dots show the edges and nodes of the graph respectively. The black rectangle marks the robot's position. Rectangular traversability checks are no longer required for NAI shown in images (b) and (c).

and solely relies on the number of observable unknown voxels and the distance to the goal node.

For the new reward function $R(n) = G(n) \cdot C(n)$, the gain $G(n)$ and cost functions $C(n)$ are updated. While the cost function is changed to include more factors, both $G(n)$ and $C(n)$ are adapted to have more comparable values.

7.4.1. Gain Function

The gain function is adjusted from the number of observable unknown voxels at the best robot orientation $G(\varphi_{max})$ to the ratio $G(\varphi_{max})/g_{max}$ that is introduced as the view score in Section 5.1.5. It uses the maximum number of observable unknown voxels for the sensor's FoV g_{max}.

This new gain function is in a range between zero and one instead of between zero and an arbitrary voxel number that differs widely from sensor to sensor. This makes the different factors of the reward function easily readable and more comparable.

7.4.2. Cost Function

The revised cost function $C(n)$ includes the distance, change in heading, traversability and radius of a node n and is shown in Equation (7.3). For comparability, a revised cost function $C_b(n)$ for the basic RRG is introduced in Equation (7.4) which omits the radius part.

$$C(n) = e^{-\frac{d \cdot D(n) + h \cdot H(n) + t \cdot T(n)}{1 + r \cdot I(n)}} \tag{7.3}$$

$$C_b(n) = e^{-(d \cdot D(n) + h \cdot H(n) + t \cdot T(n))} \tag{7.4}$$

$C(n)$ is still based on the exponential function with a negative power which lets it decay rapidly for increasing costs. For all parts, user-definable factors are introduced, which allow to modify the exploration focus depending on the particular requirements. The different parts of the cost function are detailed in the following.

Distance

The distance $D(n)$ remains the distance d_{rn} between the node nearest to the robot n_r and the node n. It is measured as the sum over all edge lengths in the path between the nodes in meters as described in Section 6.1.2.

The distance factor $d \in [0, 1]$ can be used to reduce the influence of the distance on the overall cost function. This means that nodes further away from the robot do not inflict a significantly higher cost than closer nodes. If d is set to zero, the distance has no impact on $C(n)$. This is expected to increase the traveled distance but probably leads to more information gain early in the exploration as nodes with a high gain are prioritized.

Heading

To calculate the heading $H(n)$ to a node n along the edges of the graph from n_r, the heading change at each node must be aggregated. This includes the heading change from the robot's current heading to n_r and the heading change to the best orientation at the specific node.

Therefore, additional attributes are stored in the graph's ROS message that is described in Section 6.1. Each edge e in the list of edges stores its length, which is the distance between the node positions it connects, as well as the indices of those two nodes in the list of nodes. They are stored as the first and second node where the first node is always the node with the lower index. Furthermore, the edge's orientation φ_e is stored in degrees which is calculated from the first node to the second using arctan. The orientation is measured in the ROS map frame that is described in Section 2.1. Nodes also hold the following three additional attributes:

Heading into node: The orientation φ_{in} is the angle in degrees at which the edge e enters the particular node n measured from the ROS map frame. Because the edge's orientation φ_e differs depending on the direction in which it is traversed, the indices of both nodes connected by e have to be regarded. If the index $i_{n_{prev}}$ of the previous node n_{prev} in the path to the robot P_{rn} is smaller than the index i_n of node n, φ_{in} equals φ_e. Otherwise, φ_e must be inverted. If $n = n_r$, the orientation of the robot φ_r is used because no n_{prev} exists. This differentiation is shown in Equation (7.5).

$$\varphi_{in}(n) = \begin{cases} \varphi_e(n_{prev}, n), & i_{n_{prev}} < i_n \\ \overline{\varphi_e}(n_{prev}, n), & i_n < i_{n_{prev}} \\ \varphi_r, & n = n_r \end{cases} \tag{7.5}$$

Heading change to node: The accumulated change in heading $\Phi(n)$ in degrees along the graph's edges in path P_{rn} is calculated iteratively. Therefore, the heading change $\varphi_{diff}(n_{prev}, n)$ from node n_{prev} in the path to node n is added to the previous node's accumulated heading change $\Phi(n_{prev})$ as shown in Equation (7.7).

$$\varphi_{diff}(n_{prev}, n) = |\varphi_{in}(n) - \varphi_{in}(n_{prev})| \tag{7.6}$$

$$\Phi(n) = \begin{cases} \Phi(n_{prev}) + \varphi_{diff}(n_{prev}, n), & n \neq n_r \\ 0, & n = n_r \end{cases} \tag{7.7}$$

$\varphi_{diff}(n_{prev}, n)$ is the absolute heading difference in degrees between the expected heading into n and n_{prev}. If $n = n_r$, the heading change to this node is zero.

The first node in P_{rn} can be omitted if the robot is already on the edge heading towards the second node, based on the metric shown in the last paragraph of Section 5.1.7.

Heading change to node's best orientation: The accumulated heading change to a node n in addition to the heading change to this node's best orientation φ_{max} results in the heading cost factor $H(n)$ that is shown in Equation (7.8). φ_{max} is introduced in Section 5.1.5 and references the orientation at a particular node which yields the most information gain.

Additionally, $H(n)$ is divided by the factor 90 which results in a quarter-turn of the robot inflicting the same cost as 1 m of distance.

$$H(n) = \frac{\Phi(n) + |\varphi_{max} - \varphi_{in}(n)|}{90} \tag{7.8}$$

$H(n)$ equals zero, if the node is *explored* or *failed*, or if φ_{max} has not been calculated. If it is *explored* or *failed*, it is not considered for the exploration goal selection anymore. φ_{max} is a result of the information gain calculation for a node which is required before a node can be considered as an exploration goal.

The heading factor $h \in [0, 1]$ influences the heading's impact on the cost function. When h equals one, a node in 2 m distance in front of the robot is more preferable for exploration than a node 1 m directly behind the robot. Assuming that both nodes have an equal gain. If h equals 0.5, a turn of 180 degrees equals 1 m of distance regarding the cost function.

Traversability

The traversability $T(n)$ is calculated based on the traversability analysis undertaken to place and connect new nodes. Similar to the heading, the traversability is the sum over the particular traversability of all nodes and edges along the path P_{rn}.

Therefore, the methods introduced and referenced in Sections 5.1.4 and 7.1.2 are modified to accumulate the values of all individual cells for an area in addition to checking their traversability. The sum of these values which range between m_{free} and m_{oc}, are stored for all particular nodes and edges. Furthermore, the accumulated value

c_{trav}, the total number of cells c_{cells} and the number of cells c_{oc}, that are not equal to m_{free}, inside each node and edge area are saved.

To retrieve the traversability cost τ for a node n, the formula from Equation (7.9) is used which requires c_{trav}, c_{cells} and c_{oc}. The average traversability value of all cells whose values are unequal to m_{free} is divided by m_{oc}. This normalizes the value to a range between zero and one. In addition, the ratio of cells with a value above m_{free} to c_{cells} is calculated. To take the larger radius of inflated nodes and the corresponding higher number of cells into account, τ is multiplied by the ratio between the robot's radius and the particular node's radius r_n. This effectively decreases the cost for larger radii.

$$\tau(n) = \left(\frac{c_{trav}}{c_{oc} \cdot m_{oc}} + \frac{c_{oc}}{c_{cells}} \right) \cdot \frac{r_{robot}}{r_n} \tag{7.9}$$

$$T(n) = \begin{cases} T(n_{prev}) + \tau(n), & n \neq n_r \\ \tau(n), & n = n_r \end{cases} \tag{7.10}$$

Equation (7.10) describes the calculation of $T(n)$ which is based on the traversability of the previous node in the path n_{prev}. n_r is initialized with just its own traversability cost $\tau(n)$. A node with, e.g., a quarter of its cells having a value of $m_{oc}/4$, has a traversability cost of 0.5.

For comparison, Equations (7.11), (7.12) and (7.13) show the calculation of $T_b(n)$ for basic RRG without NAI. The node's traversability cost $\tau_b(n)$ is calculated without a radius ratio but includes the maximum traversability value of a cell in the node's radius c_{max}. This maximum value is omitted for inflated nodes as it is typically $m_{oc} - 1$ because the node's area is inflated until a collision occurs.

The traversability cost τ_b for each edge e is derived in the same way. The traversability $T(n)$ consists of the cost of the previous node n_{prev} plus the particular node's cost and the cost of the edge connecting these two. The cost for n_r is set to $\tau_b(n)$. If there are no tiles with a value unequal to m_{free}, the traversability cost for a particular node or edge is zero.

$$\tau_b(n) = \frac{\frac{c_{trav}}{c_{oc}} + c_{max}}{m_{oc}} + \frac{c_{oc}}{c_{cells}} \tag{7.11}$$

$$\tau_b(e) = \frac{\frac{c_{trav}}{c_{oc}} + c_{max}}{m_{oc}} + \frac{c_{oc}}{c_{cells}} \tag{7.12}$$

$$T_b(n) = \begin{cases} T_b(n_{prev}) + \tau_b(n) + \tau_b(e(n_{prev}, b)), & n \neq n_r \\ \tau_b(n), & n = n_r \end{cases} \tag{7.13}$$

The user-defined traversability factor $t \in [0, 1]$ adjusts the impact of the traversability on the cost function. Lower values reduce the importance of choosing paths that are without traversable obstacles.

Radius

The node radius takes a special role in the cost function because its goal is to guide the robot towards more central paths with a safer distance from obstacles and a better observation range.

The node radius r_n is determined by the inflation process introduced in Section 7.1 and starts at the robot's radius r_{robot}. It is limited by the maximum sensor range r_{max}.

$$\rho(n) = \begin{cases} \rho(n_{prev}) + r_n, & n \neq n_r \\ r_n, & n = n_r \end{cases} \tag{7.14}$$

$$I(n) = \frac{\rho(n)}{|P_{rn}| \cdot r_{robot}} - 1 \tag{7.15}$$

Equation (7.14) models the accumulation of all radii ρ in P_{rn} which is used to derive $I(n)$ shown in Equation (7.15). $I(n)$ is divided by the robot's radius r_{robot} and the number of nodes in the path P_{rn} to retrieve the ratio of each node's radius in the path compared to r_{robot}. $\rho(n)$ for n_r is set to r_n. To set $I(n) \in [0, r_{max}/r_{robot} - 1]$, one is subtracted. This enables a radius factor r in the cost function.

$I(n)$ is used to scale the complete cost function $C(n)$ to emphasize paths through nodes with larger radii. This is intended to increase the safety margin to obstacles and the observable volume due to a greater viewing distance. The distance, heading and traversability in $C(n)$ are divided by one plus the radius function times the radius factor r.

The radius factor $r \in [0, 1]$ determines how much the cost is influenced by the radius. If larger radii are not desired to have an impact on a node's cost, the factor should be set to zero. A path through nodes with a respective radius of $2 \cdot r_{robot}$ has half the cost compared to a path through nodes with a respective radius of r_{robot}.

7.5. Cost-Based Path Planning

The new cost function introduced in the previous section is also intended for the global navigation planner `RneGlobalPlanner`. It is proposed in Section 5.1.7 and modified in Section 6.1.2.

The new path planning is not solely based on the particular node's distance to the robot d_{rn} but on the complete cost function $C(n)$. Dijkstra's algorithm is applied to the graph with $C(n)$ replacing d_{rn}.

The best path to each node is recalculated as soon as the robot traverses to a new nearest node n_r in the graph. Starting at n_r, Dijkstra's algorithm is run and the path with the least cost for each node is found. Furthermore, due to the influence of the robot's heading on the cost function, paths must be recalculated as well when the heading changes.

For all nodes with an edge to n_r and for n_r itself, the heading from the robot to the particular node is identified. If the robot is between two nodes, the heading and remaining cost is calculated based on the robot's current orientation. This means the heading cost is based on the heading difference between the robot and the particular node directly without assuming a possible detour through n_r.

7.6. Re-Updating Nodes

To further speed up the exploration, re-updating the gain of unexplored nodes is introduced. While the robot moves to the current goal node, only newly added or updated

nodes with a better reward can cause the selection of a new goal which aborts the current one. If no better node becomes available, the node gains are only updated after reaching the current goal. During the travel to the current goal, the gain of nodes on the path is changing and needs to be updated accordingly. This is expected to enhance the exploration by abandoning goal nodes before the robot reaches them. The switch to better goals is anticipated to reduce the exploration duration and traveled path length.

Nodes with status *initial* or *visited* can be re-updated while no new or initial node gains must be calculated. A list of all nodes to re-update is maintained and all nodes that are not *explored* or *failed* are added to it and sorted by their Euclidean distance to the robot in ascending order. The closest node's gain function is re-updated first. After a node's gain has been calculated, it is removed from the list.

Furthermore, the list is re-initialized every time a new node becomes the nearest node to the robot. For this, all nodes currently in the list of re-updatable nodes are discarded and each node with status *initial* or *visited* is added again afterwards. A list of the three previous nearest nodes is maintained to prevent constant re-updating when the robot localization is imprecise and the nearest node switches back and forth rapidly. The re-initialization is only conducted if the new nearest node is not already in this list.

When the nearest node is the current goal node, a re-update of it is triggered when the robot comes closer than one-tenth of its radius. Especially if the robot has to align its heading to the best orientation for the specific goal node, this re-update can cause the selection of a new goal and skip the slow final approach to the current goal.

Also, a heuristic to reduce the number of re-updatable nodes is used. If a node should be re-updated whose d_{rn} is larger than r_{max}, re-updating is stopped and the list of re-updatable nodes is cleared. It is assumed that nodes at this distance are only marginally impacted by the robot's current observations.

7.7. Comparison to RRG-Based Exploration

To evaluate the enhancements introduced to RNE in this chapter, a series of simulations is conducted that explores the impact of each change compared to the basic RRG-based algorithm.

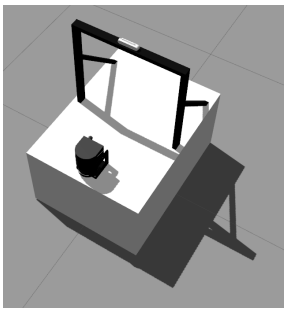
First, the setup for the simulations is explained, followed by the results for the different enhancements which goes into detail about the impact of each factor.

7.7.1. Evaluation Setup

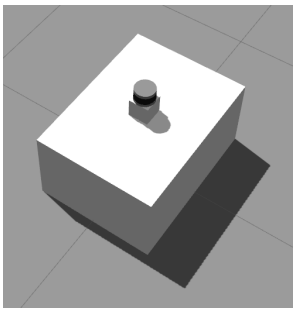
While previous simulations were conducted using a desktop computer specified in Section 5.2, the following experiments are run on an NVIDIA DGX-2. The DGX-2 is equipped with a Dual Intel Xeon Platinum 8168 processor with 2.7 GHz and 24 cores, 16 NVIDIA Tesla V100 GPUs and 1.5 TB RAM.

But each simulation is limited to using one GPU, 12 CPU threads and 16 GB of RAM. Furthermore, the simulation is run in a headless Docker container with RViz and Gazebo GUIs deactivated. The container also runs an X server using Xvfb[1] because the Gazebo depth camera simulation requires it to render the camera data.

[1] `https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml`



(a) Simulated box robot with depth camera and 2D lidar as C.



(b) Simulated box robot with 3D lidar as L.

Fig. 7.8.: Simulated artificial box robot configurations for the factor comparison experiments.

The environments shown in Section 6.2 are used for the simulations. The environment from Figure 6.2a without the added obstacles is referenced as ME and from Figure 6.2b as CE. They use the same time limits, 30 min for the former and 1 h for the latter.

The C configuration's Clearpath Robotics Husky UGV shown in Figure 5.8d is replaced with a simple box that can move and rotate horizontally but has the same sensor setup and placement. The second configuration features a rigidly attached Velodyne PUCK VLP-16 lidar and is referenced as L. Both configurations can be seen in Figure 7.8.

The Husky is replaced with the artificial box robot to reduce the impact of the simulated wheels on the result. The Husky is a skid steer drive which can cause jerky motions while rotating on the spot. This is avoided using the hovering box robot that has no direct ground contact and utilizes Gazebo's planar move plugin[2].

Furthermore, the robot localization is directly extracted from Gazebo and therefore without uncertainty. This removes the impact of partly sub-optimal localization on the following experiment's results.

The parameters are the same as in Section 6.2 but g_{min} for the L configuration is set to 0.03. Three variants are run which are C-ME, L-ME and L-CE. For each variant, ten runs are executed and the mean μ and SD σ for the duration, path length, mapped volume and algorithm run time are reported.

The algorithm run time is the percentage of how much time the calculations of RNE require. For this, the required processing time of the main algorithm and the decoupled gain calculation are aggregated. These values can be higher than one because two processing threads are used.

Furthermore, the number of failed and total runs is shown as well. Failed runs are automatically detected by a difference between explored and placed nodes of more than 10 and the overall mapped volume. If it is below 1500 m^3 for ME and 8000 m^3 for CE, the run counts as failed. These values are based on the experience from previous experiments where failed runs were defined manually.

The simulations to compare the impact of each factor are carried out in three stages. The first stage uses the basic RRG-based RNE proposed in Chapter 6. The second stage adds NAI from Section 7.1 and the third stage utilizes topology-based NAI shown in Section 7.2.

For each stage and factor, different combinations to explore the particular factor's

[2] `https://classic.gazebosim.org/tutorials?tut=ros_gzplugins#PlanarMovePlugin`

impact are run. To reduce the total number of simulation runs, only the following four combinations are chosen for each factor F:

F0.5 The first combination sets the respective factor F to a value of 0.5. The remaining factors are set to 1.

F0 F is set to value 0, the other factors to 1.

NF0.5 All factors besides F are set to a value of 0.5. F is set to 1.

NF0 F is set to 1, the remaining factors to a value of 0.

For each combination, the descriptor F is replaced with the particular factor's abbreviation which are D for the distance factor d, H for the heading factor h, T for the traversability factor t and R for the radius factor r. At each stage, one set of runs is executed with all factors set to 1 for comparability which is referred to as 1.

The results of all 1350 runs from the 135 combinations are discussed in the following subsections and are listed in detail in Appendix A (p. 163).

7.7.2. Basic RNE Cost Factors

The first stage is about the comparison of the newly introduced cost function's impact on the exploration performance of the RRG-based RNE algorithm. Without NAI, the cost function $C_b(n)$ from Equation (7.4) is used and only the distance d, heading h and traversability factors t are evaluated.

Table A.3 (p. 167) shows that reducing the distance factor d significantly reduces the exploration duration in all environments as nodes with the most gain are preferred regardless of their distance to the robot. This comes at the price of a vastly increased traveled path length as the robot has to backtrack later to explore smaller, left-out areas.

The overall algorithm run time is mostly unaffected by the varying distance factor but the amount of failed runs increases for the ND combinations. This is probably caused by selecting the closest node which might be close to obstacles and has a high traversability cost that is not regarded in these combinations. Ultimately, the robot gets stuck on these nodes as it maneuvers too close to obstacles too often.

In Figure A.1 (p. 164), it can be seen that a reduced distance factor increases the amount of observed map volume early on as well as the traveled path length. The curves of the ND combinations are very similar to the default combination 1. This shows the major impact the distance has on the cost function.

The heading factor causes hardly any change in the results when being reduced which can be observed in Table A.4 (p. 168) and Figure A.2 (p. 165). When it is the only remaining factor, it significantly speeds up the exploration but also increases the traveled path length. This is caused by the robot preferably following mostly straight paths. It also increases the explored volume early in the run compared to other combinations.

Table A.5 (p. 168) and Figure A.3 (p. 167) show that the traversability factor, similar to the heading factor, causes no significant impact on the duration or path length when being reduced. When the other factors are reduced to 0 and the traversability factor t remains 1, the traversed path length drastically increases. This is caused by following

paths that stay clear of obstacles first and returning to nodes closer to obstacles later in the exploration.

Furthermore, when reducing the traversability factor, more runs fail in the C-LE variant because badly traversable nodes are not evaded anymore.

7.7.3. Node Area Inflation RNE Cost Factors

The second stage adds NAI to the RNE algorithm and utilizes the cost function $C(n)$ from Equation (7.3). The distance d, heading h, traversability t and radius factors r are varied and discussed in the following.

In Figure A.4 (p. 170), a clearer distinction between the different combinations for the distance factor can be seen compared to basic RNE. A decrease of d causes a faster exploration with more explored volume early on but also a longer traveled path length.

The reduction of all factors but the distance factor has the opposite effect. Less early map coverage and a decreased path length are also reported in Table A.6 (p. 175).

An increased rate of failed runs for D0 and the ND combinations for the L-CE variant shows that selecting nearby or far away nodes can get the robot stuck in larger environments.

The heading factor for the inflation process behaves similarly to the basic RNE as shown in Table A.7 (p. 175) and Figure A.5 (p. 171). Reducing it has hardly any effect on the exploration but the NH combinations significantly decrease the duration and increase the path length. For the L-CE variant, the path lengths are similar for the different combinations. But the non-NH combinations run into the time limit while NH finishes the exploration.

Table A.8 (p. 176) shows that the traversability factor's effect is also similar to the basic RNE. It has a minimal impact when reduced and reducing all other factors increases the traveled path length vastly. This can also be seen in Figure A.6 (p. 173). It is notable, that the number of failed runs is lower than for the basic algorithm. This is probably due to the increased safety margin to obstacles because of NAI.

The radius factor is not added to the cost function like the remaining factors but divides it and therefore decreases the cost when it rises. Table A.9 (p. 176) and Figure A.7 (p. 174) show that reducing the radius factor increases the duration of the exploration. For the C-ME and L-CE variants, the reduction of r also decreases the traveled path length which is probably caused by selecting nearby nodes over further away nodes with a larger radius.

As the radius factor can only reduce the cost of nodes, the NR0 combination leads to failed and significantly worse explorations. The NR0.5 combination reduces the exploration duration for all variants and increases the path length for C-ME and L-CE which is likely caused by the reduction of the distance factor.

For the R0 combination, the number of failed runs in the L-CE variant is increased which is caused by not favoring paths with larger radii that increase the safety margin to obstacles.

7.7.4. Topology-Based Node Area Inflation RNE Cost Factors

The topology-based NAI exploration is the third stage of the cost factor comparison. It uses the same cost function and factors as NAI RNE.

Decreasing the distance factor again reduces the duration and increases the path length as shown in Table A.10 (p. 183). Also, the map coverage is increased early and all combinations are distinguishable in Figure A.8 (p. 178), similar to the NAI results. The ND combinations reduce the traveled path length but increase the duration as well.

Table A.11 (p. 183) and Figure A.9 (p. 179) show only a minor impact for reducing the heading factor as well as for combination NH0.5. Combination NH0 causes a reduction of the duration and for variant C-ME an increase in traveled path length. Following the topology-based, inflated nodes seems to speed up the exploration without drawbacks if a sensor with a 360 degrees FoV is deployed. Otherwise, the robot has to backtrack to map unobserved space.

The traversability factor's impact can be seen in Table A.12 (p. 184) which shows that reducing it causes minor decreases in duration and path length. Furthermore, the NT combinations lead to an increased duration for C-ME, a decreased duration for L-ME and a significantly decreased duration for L-CE. The NT0 combination shows an elongated path length which can also be seen in Figure A.10 (p. 181).

Reducing t does not lead to a significant increase in failed runs compared to the other two stages which is caused by the topology-based paths having a greater safety distance to obstacles.

Table A.13 (p. 184) shows that decreasing the radius factor increases the exploration duration which emphasizes its positive effect on the exploration.

The NR combinations behave similarly to the second stage as can be seen in Figure A.11 (p. 182). NR0.5 reduces the duration and increases the path length for C-ME and L-CE variants probably caused by the reduced distance factor. NR0 leads to failed runs and considerably worse results which is expected.

7.7.5. Re-Updating Nodes RNE Cost Factors

In addition to the previously described cost factor evaluations, another 10 runs for each variant are conducted for the topology-based NAI together with re-updating nodes. The configurations and parameters remain the same. All cost factors are set to one for this comparison.

Table 7.1 shows the results of comparing the basic RRG-based RNE implementation referenced as RRG, RNE with NAI referenced as I, topology-based NAI referenced as T and re-updating nodes referenced as R.

The results averaged over time can be seen in Figure 7.9 which shows that each enhancement increases the map coverage earlier but also leads to longer traveled path lengths. The run time is decreased for variants L-ME and C-ME when using NAI or topology-based NAI. Re-updating nodes increases the run time as expected but it is still below the basic RRG-based approach for L-CE.

For variant C-ME, NAI and re-updating nodes add an overhead in processing compared to the gain calculation. It is not as expensive for C because of the much smaller range of the depth camera.

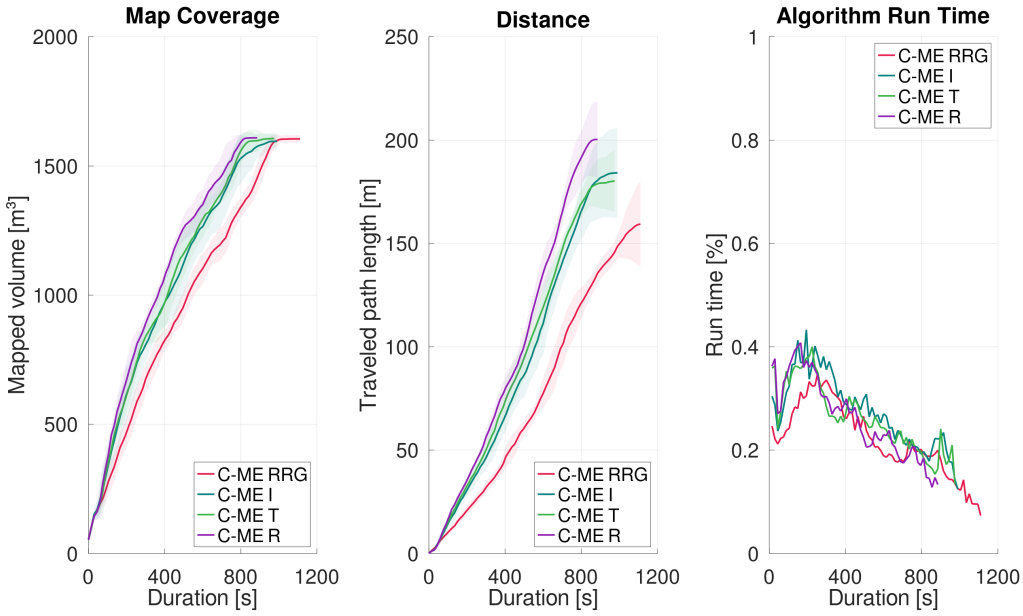
The topology-based NAI with re-updating achieves a reduction in exploration duration of around 20% for C-ME and L-CE, and 41% for L-ME compared to RRG. The increased path length of 31% is caused by the RRG runs being stopped at the time limit of 60 min for L-CE.

Tab. 7.1.: Comparison of basic RRG-based RNE as RRG, RNE with NAI as I, RNE with topology-based NAI as T and RNE with re-updating nodes as R. It shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively. The best mean values for each variant are printed in bold letters.

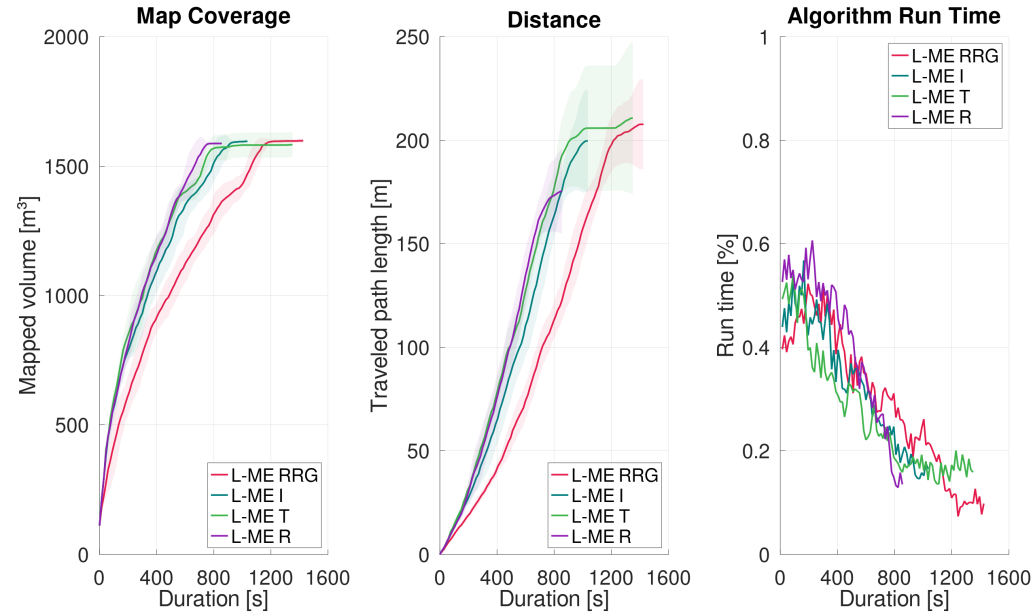
Configu-	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
ration	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME RRG	1020.00	57.01	**159.36**	20.24	1604.2	15.4	**0.232**	0.014	0	10
C-ME I	883.50	58.52	184.10	21.64	1594.6	31.1	0.288	0.023	0	10
C-ME T	856.50	52.55	180.18	14.88	1605.5	16.3	0.276	0.009	0	10
C-ME R	**820.50**	44.63	200.27	18.18	**1608.3**	3.2	0.274	0.015	0	10
L-ME RRG	1236.00	79.44	207.70	21.71	1597.9	8.4	0.327	0.032	0	10
L-ME I	945.00	72.11	199.69	24.65	1595.6	7.4	0.349	0.029	0	10
L-ME T	963.33	165.64	216.64	33.34	**1598.0**	6.9	**0.318**	0.023	1	10
L-ME R	**729.00**	64.50	**175.47**	20.49	1587.3	4.5	0.452	0.023	0	10
L-CE RRG	3594.00	7.75	**499.02**	39.87	8327.6	157.8	0.534	0.032	0	10
L-CE I	3399.00	143.04	654.21	28.63	**8424.2**	74.8	0.488	0.017	0	10
L-CE T	3337.50	126.54	649.15	38.90	8394.5	54.3	**0.473**	0.020	0	10
L-CE R	**2929.50**	239.55	655.71	53.88	8417.9	75.3	0.533	0.018	0	10

When comparing topology-based NAI to NAI, only a small decrease of maximally 3% in duration can be measured. But the run time is decreased by around 4%, 9% and 3% for the variants C-ME, L-ME and C-LE respectively.

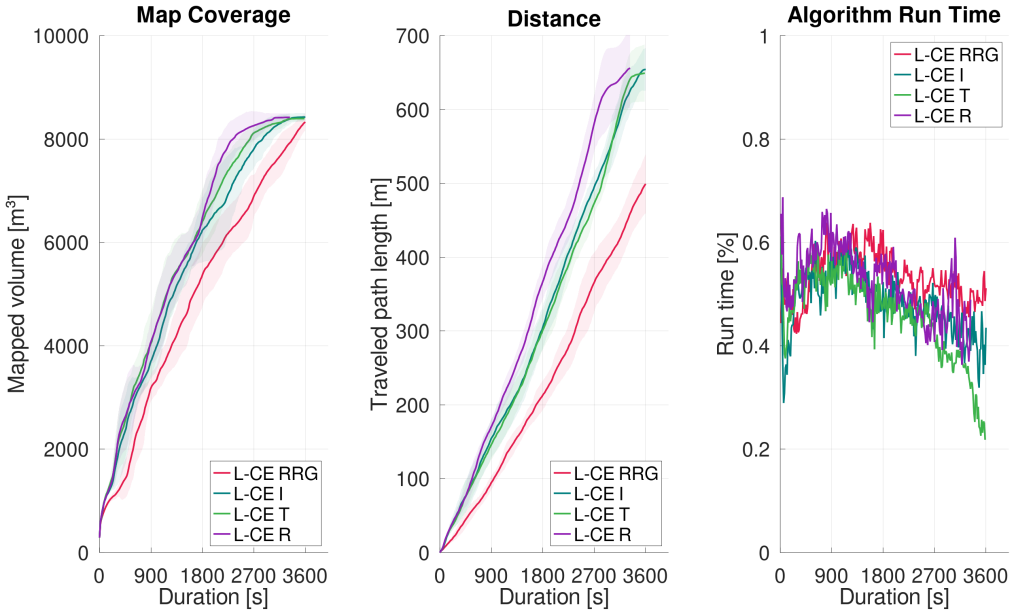
Re-updating nodes compared to topology-based NAI decreases the duration by up to 24% for L-ME and 12% for L-CE. On the other hand, the run time is increased by 42% for L-ME but only 12% for L-CE.



(a)



(b)



(c)

Fig. 7.9.: Mean mapped volume, path length and algorithm run time over the duration for basic RRG-based RNE as RRG, RNE with NAI as I, RNE with topology-based NAI as T and RNE with re-updating nodes as R. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.

7.7.6. Evaluation Conclusion

The introduction of the revised reward function highlights the varying impact of the different factors of the cost function. The distance factor takes the most prominent part as it significantly influences the traveled path length, duration and early map coverage for each of the enhancements. The heading and traversability factors have minor roles

but are shown to be able to alter the exploration results when the distance factor is reduced. For the simulations with NAI, the radius factor decreases the exploration duration.

NAI and topology-based NAI each decrease the exploration duration and the computation time. They also reduce the risk of failed runs for the different variations significantly. The re-updating of nodes speeds up the exploration even more but comes at a vast increase in computation time.

7.8. Conclusion

This chapter introduces several enhancements for the RRG-based RNE implementation and evaluates them in a set of experiments. The following enhancements are proposed with the described benefits:

Reward Function A new reward function is introduced which standardizes all factors. It consists of the node gain and the cost function which takes distance, heading change, traversability and node radius into consideration. The path planning is also changed to use this cost function instead of just the distance like in previous iterations. This enables more refined paths depending on the cost function's factors.

Node Area Inflation NAI increases the distance between the robot and obstacles. This reduces the risk of collisions and makes the robot observe larger areas which improves the exploration by decreasing its duration.

Topology-based Node Area Inflation Moving new nodes away from obstacles, unknown space and nearby existing nodes during the inflation process creates topology-based graphs. These perform better in an exploration than graphs with just NAI.

Re-Updating Re-updating node gains while the robot moves to the current goal allows to replace the goal early on with a better node. This reduces the duration significantly but also increases the algorithm run time.

The described enhancements increase the efficiency of the proposed third iteration of RNE compared to the previous iterations using RRT and RRG. The new cost function can be adapted to specific robots and exploration requirements. For example, a reduced distance factor leads to more observed space early in the exploration and a decreased duration at the cost of a longer robot path length.

The goal to reduce the computation time using the topology-based NAI is successful when not utilizing re-updating. But there is still a large overhead produced by the steadily growing graph size during large explorations.

Hybrid approaches that separate the exploration algorithm into a global and a local part were highly successful in the DARPA SubT Challenge which is introduced in Section 1.1.4. The separation of RNE into a global and a local part in its fourth iteration is inspired by these approaches and is expected to reduce the computation time growing with graph size.

8. RNE 4 - Local and Global Exploration

This chapter describes the fourth and final iteration of RNE which introduces the distinction between local and global exploration to reduce the computation overhead of maintaining a constantly growing graph.

The algorithm described in Chapter 7 becomes the local part of this hybrid RNE implementation. Its graph is limited to a local area around the robot while a global exploration graph keeps track of unexplored nodes from the local exploration.

These nodes must be visited to complete the global exploration. A shortest route has to be found which goes through every node exactly once to efficiently visit all of them. This is implemented with a TSP solver which is introduced in Section 3.4.

The implementation of a hybrid exploration including a TSP solver is inspired by [7, 9, 12, 14, 18, 19, 20]. Some of these approaches detailed in Section 2.4 were used in the DARPA SubT Challenge described in Section 1.1.4.

To remove and add nodes in the local graph as well as to integrate unexplored, removed nodes into the global exploration, further methods are introduced to RNE. A clustering of global nodes is also implemented to reduce the number of nodes that have to be connected by the TSP solver.

The first section describes the changes to the local exploration followed by the global exploration and implementation details of the finished RNE package. To highlight the fourth iteration's improvements compared to the previous implementations, simulations are conducted. Another comparison to state-of-the-art approaches, from which some were successfully deployed in the DARPA SubT Challenge, is executed. Finally, the proposed approach is demonstrated in an experiment with a real robot in an underground environment.

8.1. Local Exploration

The local exploration uses the algorithms introduced in the previous chapters which conclude to RNE with a sampling- and topology-based NAI including re-updating nodes. Several adaptions are introduced in the following that allow RNE to be used in a local exploration area which dynamically follows the robot's position.

8.1.1. Exploration Area

The local exploration uses a sliding area that moves along with the robot. New nodes are only sampled inside this area and nodes that fall outside when the robot moves, are removed. To reduce computation time, nodes are only pruned from the graph when a new node becomes the nearest node to the robot instead of a frequency-based pruning.

The local exploration area is defined as a circular area around the robot with a user-defined radius r_G. For each node in the graph $\mathcal{G}$, the distance to the robot is calculated. To determine more computation efficiently if a node is inside this area, the squared radius is used.

If the distance is above the pruning radius $r_{pr} = r_G + r_{robot}$, the particular node is removed from the graph. The area in which new nodes are sampled and added to $\mathcal{G}$ is restricted to r_G. r_{robot} is added to r_G for r_{pr} to prevent nodes from being removed and added at a similar spot when the robot moves back and forth or the localization oscillates slightly.

Nodes that are outside r_{pr} are removed and all edges connected to them as well. But it must be checked if there are nodes connected to the nodes to be removed that have no other connection to the local graph. These nodes are classified as disconnected and are removed as well.

For all removed edges, it is checked if the path to the robot P_{rn} of the node at the particular edge has a node that is pruned as its next node in P_{rn}. If it has, the node is added to a list of disconnected nodes. For each of the disconnected nodes, that have been added to the list, all neighboring nodes, where the disconnected node is the next node in P_{rn}, are added to the list as well. If one of these neighboring nodes has a P_{rn} without a disconnected node, it is added to a list of connected nodes. If a connected node later is disconnected, it is removed from the list of connected nodes again.

After all nodes with an edge to removed or disconnected nodes have been categorized, Dijkstra's algorithm is executed from each connected node without resetting all others cost function as described in Section 6.1.2. All nodes that are reconnected to the graph by Dijkstra's algorithm, are removed from the list of disconnected nodes. Finally, all remaining disconnected nodes are removed from the graph. Figure 8.1 shows an example for the previously described process.

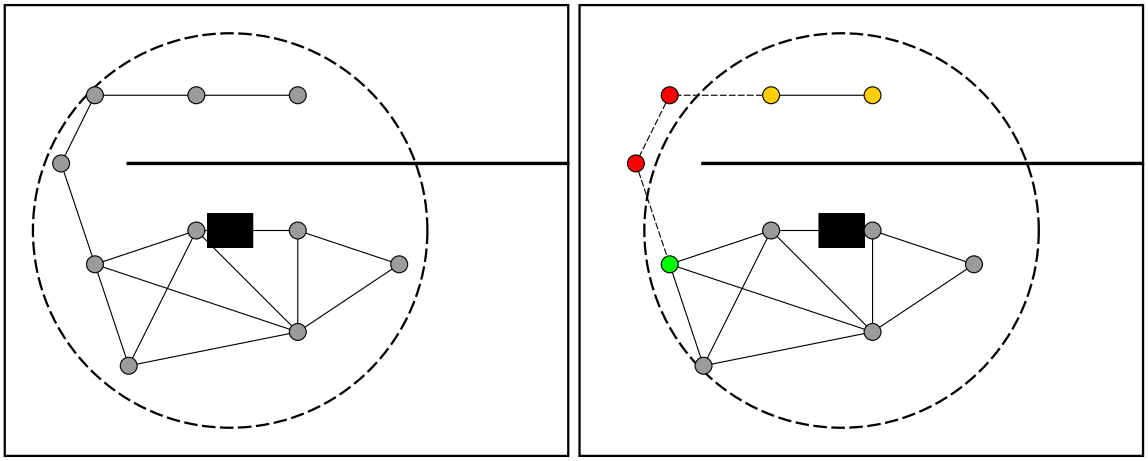


Fig. 8.1.: The left figure shows a local graph before the robot moves and the right figure afterwards. Gray circles are nodes in the graph and the lines connecting them are the edges. The black rectangle represents the robot, the dashed circle the local pruning radius r_{pr} and bold black lines show obstacles. In the right figure, red circles show removed nodes that are outside of the local area and dashed lines represent removed edges. The yellow circles are disconnected nodes that are finally removed as their path to the robot leads through removed nodes. The green circle symbolizes a connected node that remains in the graph.

8.1.2. Removing and Adding Nodes

To allow a sliding exploration area, nodes have to be removed from the graph. The lists of nodes and edges are stored in a list as a part of the ROS message introduced in Section 5.1.2 and refined in the previous iterations of RNE. Both lists are ordered by the respective node and edge indices which are also used for referencing connections.

To reduce the computation, nodes and edges are not removed from the lists because this requires re-indexing all nodes or edges with a higher index than the removed item as well as their references. Instead, a new *inactive* status is introduced to which a removed node is set. Edges receive a field to indicate if they are inactive. Only if a removed node or edge is the last element in the list, it is removed to free memory.

Another two lists holding the respective indices of inactive entries for nodes and edges are maintained. If a new node or edge is added, the inactive entry with the smallest index is replaced with a new active entry.

If the last entry of the list of nodes or edges is removed and the prior entries are inactive, they are removed up to the first active entry. Furthermore, the removed entries' indices are deleted from the list of inactive entries.

When nodes are removed, they are added to the list of global targets if they are not already *explored* or *failed*. The mechanism of adding them is described in Section 8.2.

8.1.3. Pruning of Encircled Nodes

Due to the added possibility of removing nodes and edges from the graph, a further reduction of the number of nodes is introduced.

If an inflated node completely encircles another node, which means that the complete area of the encircled node lies inside the other node's area, it is pruned from the graph. This encirclement is detected when searching for neighboring nodes of the inflated node and therefore adds only marginal computational load.

As the larger node is already connected to all neighbors of the encircled node due to the overlapping areas, the encircled node can be removed without the need to add new neighbors to the encircling node. The removal is handled as described in Section 8.1.2.

If the encircled node is connected to one or more global paths and is the node nearest to the robot, it cannot be removed and remains in the graph. Because it lacks a next node in the path to the robot, the global paths to the local graph cannot be reconnected which prevents the node's removal. The global paths are detailed in Section 8.2.1.

8.1.4. Transition between Local and Global Exploration

Because the local exploration is based on the previous iteration of RNE, it also inherits the exit conditions presented in Section 5.1.8. If no unexplored nodes are present in the local graph, a timer is started. Adding new, explorable nodes stops this timer.

When the timer ends, the local exploration is terminated and the global exploration is initialized. The closest node to the robot is added as the start to the global planner and connected to all global targets. This process is explained in detail in the next section.

8.2. Global Exploration

The global exploration is used to keep track of unexplored, pruned nodes from the local exploration and calculates a path through all of them that is ordered to minimize the path length. For this, a TSP solver is deployed. To reduce the required computation time, removed unexplored nodes are clustered into global targets. During the local exploration, paths to all of these global targets are maintained.

When a global target is reached by the robot during the global exploration, the local exploration is initialized with a root node at the global target's position.

The global targets and connections are stored in a ROS message. The targets and connections are organized in lists and can be set to inactive or can be removed, similar to the nodes and edges of the local exploration shown in Section 8.1.2.

The Global graph $\mathcal{H} = (T, C)$ contains global targets T with a respective position called viewpoint $\mathbf{p}_t$ and a merged distance d_t, which is explained in Section 8.2.2. $\mathcal{H}$ includes global connections $C = (t_i, t_j \text{ or } G),\ t_i, t_j \in T,\ i > j$ with waypoints $\mathbf{w}_{c_1}, \mathbf{w}_{c_2}, ..., \mathbf{w}_{c_n} \in W_c$ and length l_c. A global connection which is also referenced as a global path, connects a global target t_i with another global target t_j or the local graph $\mathcal{G}$. The first target always refers to the target with the larger index in the list of targets.

8.2.1. Global Connections

Global paths connect global targets with the local graph and with each other. They are actively maintained while the local exploration area moves with the robot. Global paths consist of removed nodes which serve as waypoints for a connection between two global targets or a global target and the local graph.

When an unexplored node is removed from the local graph, a new global target is added or merged with an existing global target as explained in the next section. To calculate a path through all global targets starting from the robot's current position, which is in the center of the local exploration area, each global target is connected to the local graph via a global path.

For all new global targets, a path to the local graph is created which is connected to the first still active node in the previously removed node's path to the robot P_{rn}. To keep track of the global path's connection, the particular node stores a reference to the path that is connected to it. When this node is removed from the local graph, the global path is continued to the next node in the removed node's P_{rn}.

Pruning Global Goals and Connections

When a new node's area is inflated and added to the local graph, global paths and targets can be pruned or removed if they are too close to it. This is shown in Figure 8.2.

A k-d tree, which is described in Section 3.1.6, is used to find all global targets around the new node that can be connected to it according to the metric defined in Section 7.1.4. If the distance between both nodes is smaller than the robot radius, the global target is removed immediately. Otherwise, a node is sampled exactly at the global target's position in the next iteration with the goal to connect it to the local graph and remove the global target.

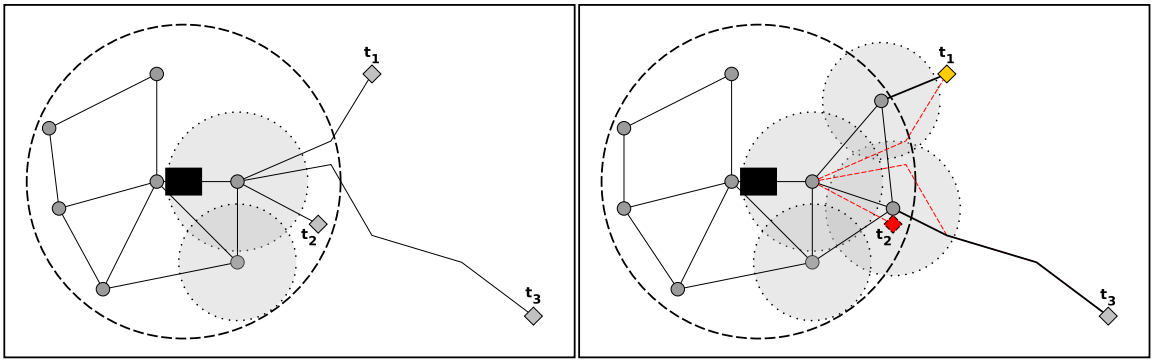


Fig. 8.2.: Both images show a local graph with nodes indicated by gray circles and the edges between them as lines. The global targets t_1, t_2, t_3 are shown as rhombuses with their respective global connections to the local graph. The robot is depicted as a black rectangle, the local exploration radius r_G as a dashed circle and relevant node radii as dotted, semi-transparent circles.
In the right image, two nodes have been added to the local graph, one of them near t_1 and the other directly next to t_2. t_1 is shown in yellow because it can be directly connected to one of the new nodes. Therefore, a new node is sampled on top of it in the next iteration of the graph construction. t_2 is pruned from the global graph due to the proximity to a new node. Furthermore, the global path to t_3 is shortened and connected to the same new node that removes t_2. Red dashed lines show the removed global connections, bold lines the new ones.

When a global target is removed because a node from the local graph is placed at its position, the path to the local graph of global targets connected to the removed global target can be optimized. If their respective path to the removed global target is shorter than their path to the local graph, the former can replace the latter.

Furthermore, for each global target's connection c to the local graph, a k-d tree is used to find all waypoints W_c that intersect with the new node's radius. The waypoint closest to the global target is connected to the new node if it and other waypoints are inside the new node's area. All waypoints between the closest waypoint and the local graph are pruned from the global path and it is connected to the new node.

A further check is conducted to find any global targets connected to the new node whose connecting path's length can be reduced when being redirected through it. Also, new global target to target connections are established if both are connected to the new node and have not been connected with each other previously.

Global Target to Target Paths

Paths between all global targets must be established to deploy a TSP solver. They can be created when the local exploration is finished and are constructed through the node nearest to the robot. But to reduce the overall global path length, connections between global targets are attempted based on mutually connected nodes. When an unexplored node is removed and added as a global target, it is checked if there already is a global path connected to the removed node or the next node in the removed node's path to the robot P_{rn}. If there is, a global path between both targets is created by copying the existing global target's paths to the local graph and adding the removed node or connected node to the list of waypoints.

When a node, that serves as a connecting node for one or more global paths, is removed from the local graph these paths have to be reconnected to another node in the local graph. The next remaining node in the removed node's P_{rn} is selected as the

new connecting node. All removed nodes between the former and new connecting node are added as waypoints to the respective global path. If the new connecting node has connections to other global paths, it is checked if the global targets of the particular paths can be merged, which is described in Section 8.2.2. Otherwise, a new global path is established between the global targets by joining both paths' waypoints to the connecting node.

Before the TSP solver is deployed, all missing global paths must be built. Therefore, all missing target to target connections are detected and calculated. These missing connections are calculated using Dijkstra's algorithm. It is started from the connecting node of each global target with missing connections. The goal is to find the shortest paths through the local graph from one global target's connecting node to all other

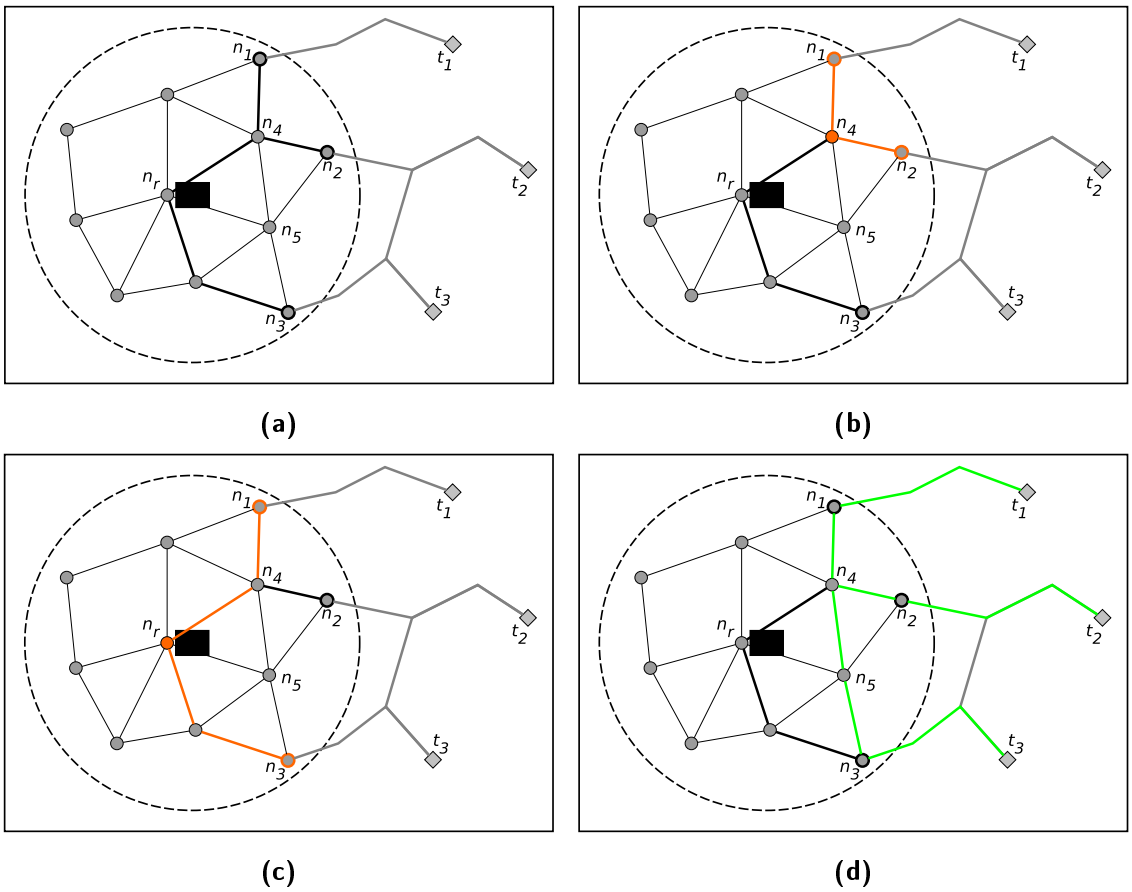


Fig. 8.3.: The creation of the missing global paths between the global targets t_1, t_2 and t_3 can be seen in the images above. Gray points depict nodes in the local graph whose area is indicated with a dashed circle. The black rectangle is the robot and rhombuses are the global targets. Thick gray lines represent the global paths, nodes with a bold border the connecting nodes and thick, black lines the edges between the connecting nodes n_1, n_2 and n_3 and the node nearest to the robot n_r.
Image (a) depicts the local and global graphs before the missing connections are established. The first mutual node in the connecting paths of t_1 and t_2 is searched in (b). Node n_4 is this first mutual node and colored in orange. Dijkstra's algorithm now has an upper bound of the orange path's length from n_1 to n_2 over n_4. In (c), the first mutual node for the connecting paths of t_1 and t_3 is determined which is n_r. The upper bound is increased to the much longer orange path from n_1 to n_3.
Image (d) shows the resulting global paths marked in green which are found by Dijkstra's algorithm and added to the global graph. Because t_2 and t_3 are already connected, no new paths have to be created for them.

global target's connecting nodes which are not already connected.

To reduce the computation required by Dijkstra's algorithm, an upper bound for the distance to each regarded node is calculated. This upper bound is found by searching for mutual nodes in P_{rn} of the connecting nodes. This is checked for each combination of global targets with a missing connection. The nearest mutual node to the connecting nodes is used to calculate the path length from one connecting node through the mutual node to the other connecting node via the local graph's edges. The worst mutual node regarding the path length is the node nearest to the robot n_r at which all paths converge. The longest mutual path found for all missing connections of a global target is used as the upper bound for path lengths in Dijkstra's algorithm. This creation of missing global paths can be seen in Figure 8.3.

The paths found by Dijkstra's algorithm are then appended to both global target's paths to the local graph to establish a new global path between the formerly unconnected targets. If multiple global target's paths to the local graph share a connecting node, the calculated paths can be reused to reduce computation time.

Finally, all global paths to the local graph are extended to n_r from their respective connecting node by using its P_{rn}. Then, all global targets are connected to each other and to n_r.

8.2.2. Goal Clustering

Every time an unexplored node falls outside the local exploration area and is removed from the graph, it is added to the list of global targets. The amount of global targets should be as small as possible to reduce the computation time required by the TSP solver.

Therefore, a heuristic is introduced with the goal to cluster nearby global targets which are inside the local exploration radius r_G of each other, if a local exploration would be started at one of them. This heuristic is applied each time when two or more global target's connections are set to the same connecting node in the local graph. Figure 8.4 visualizes the heuristic.

For the heuristic, a global target t_f is focused which is being created or whose local connecting node is replaced with a new one. The heuristic iterates over all global targets t_i with a respective connection c_i to the new connecting node of t_f. For each pair of targets t_f and t_i, it is checked if they can be merged.

t_f and t_i with their respective global connections to the local graph c_f and c_i are needed for this decision. The position of the specific global target $\mathbf{p}_t$, the distance of other targets already merged into it d_t and the length of its global connection l_c are also required for t_f and t_i.

During the iterations, it can be detected that one of the global targets cannot be merged into other targets because its distance d_t is too large. Then, it is stored as t_{nm} including its connection to the local graph c_{nm} and the length of this connection $l_{c_{nm}}$.

If a global target already has other targets merged into it, the Euclidean distance to the furthest of these targets is stored in the remaining global target as the merged distance d_t. When this global target is a candidate for clustering, its d_t plus the Euclidean distance to the other global target must not be larger than r_G. If it is, the target cannot be merged into any other target. Only the other way around is allowed, if this constraint is not also violated. This guarantees that all merged global targets

are covered by a local graph area placed at the remaining global target. Otherwise, valuable targets might be lost.

The following list describes the decision if a pair of targets can be merged:

- If the combined length of both connections is $l_{c_f} + l_{c_i} \leq 2 \cdot r_G$ and the Euclidean distance between both targets' viewpoints is $D(\mathbf{p}_{t_f}, \mathbf{p}_{t_i}) \leq r_G$, the targets can potentially be merged. The combined length has to be below or equal to $2 \cdot r_G$ instead of just r_G because of possible obstacles between two targets or detours in the global path which prevent a direct connection.
- The merged distance of t_f or t_i plus the Euclidean distance between both viewpoints has to be $d_{t_f} + D(\mathbf{p}_{t_f}, \mathbf{p}_{t_i}) \leq r_G$ or respectively $d_{t_i} + D(\mathbf{p}_{t_f}, \mathbf{p}_{t_i}) \leq r_G$.
  - If only t_i cannot be merged, it is checked if t_{nm} has been set in previous iterations. If it has not been set, t_i is set as t_{nm}.
    - If t_{nm} already exists, the combined connection path length has to be $l_{c_f} + l_{c_i} < l_{c_{nm}}$. If it is, t_i replaces the current t_{nm}.
  - Else, t_i is added to the list of global targets that can be merged.
- Else, both targets cannot be merged.

Afterwards, it is checked if t_{nm} exists. In this case, all targets, that are in the list of targets that can be merged, are merged into t_{nm}. Otherwise, the global target with the shortest connection to the local graph is chosen from the list of targets that can be merged. All other targets from this list are then merged into it.

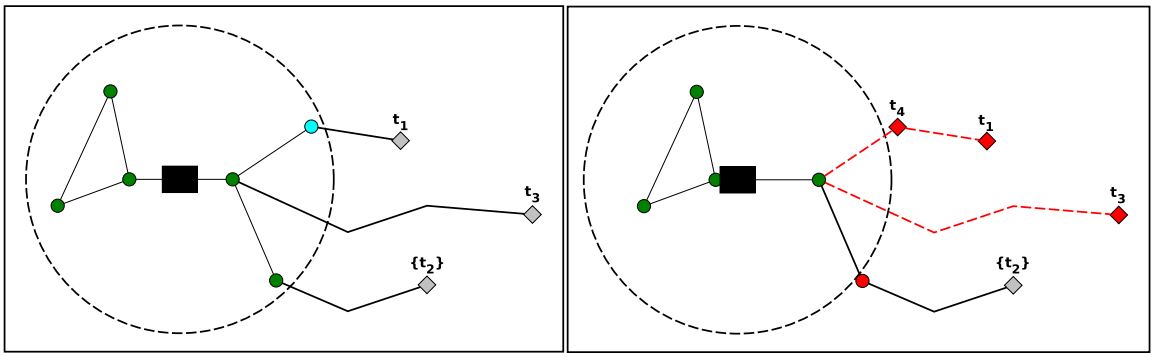


Fig. 8.4.: The clustering of global targets is shown which occurs when unexplored nodes shown as blue points are removed from the local exploration area depicted as a dashed circle. The clustering also occurs when global paths are continued because their connecting node is removed. The robot is shown as a black rectangle and global targets t_1, t_2 and t_3 as rhombuses. t_2 has curly braces around it to symbolize that it has other targets merged into it. The global paths are bold lines while edges in the local graph are normal lines. Explored nodes in the local graph are shown as green points.
On the left image, all global targets are connected to separate nodes in the local graph. Due to the robots movement, the local graph area is moved in the right image and causes the pruning of two local nodes. The first one is still unexplored and becomes global target t_4 into which the existing target t_1 is merged initially. The second one is the connecting node for t_2 whose global path is extended to the remaining node. This results in all global targets being newly connected to a single node in the local graph.
Even though t_4 is the closest target to the local graph, t_2 cannot be merged into other targets due to other, further away targets already merged into it. Therefore, t_2 remains and all other targets are merged into it and removed including their global connections which are colored in red.

The merged distance of the remaining global target is increased to the largest Euclidean distance to one of the merged target's viewpoints, if it is larger than the current merged distance. All targets that have been merged into another target are removed from the global graph including all of their connections.

Removing Unexplored Nodes

When an unexplored node is removed from the local graph, it is transformed into a global target that is connected to the local graph through a global path to the next node in P_{rn} of the removed node. But before, it is checked if the new global target can be clustered with other nearby global targets.

If multiple nodes worth exploring get removed, they are stored in a list which is sorted ascendingly by their local path length to the robot. Nodes in this list, that are each others neighbors, are grouped. For each group, only the node with the shortest path to the robot is considered for becoming a global target. All other nodes are discarded.

Before any of the remaining nodes becomes a global target, all global paths connected to a particular removed node and its global path's connecting node are iterated over. For each combination with the potential new global target, the above heuristic is executed.

If one of the other global targets cannot be clustered because its merged distance is too large, it becomes t_{nm} and the new global target is merged into it. In case multiple global targets can be clustered together, the global target with the shortest path to the local graph is selected and it is attempted to merge all other targets into it. If the merged distance constraint is violated, the responsible global target is removed from the merge process and the new potential target is the remaining global target. The maximum distance between the merged targets is stored in the remaining target.

Global paths to every other global target, that has not been clustered during the previously described process, are added afterwards.

Continuing Global Paths

When continuing a global path because its respective connecting node is removed from the local graph, it is checked if there are already other global paths connected to the new connecting node.

If there are, it is attempted to cluster these global targets. The above heuristic is utilized to assess the possibility of clustering these targets. This means, all potential new connections are checked regarding the Euclidean distance between them, the combined distances of the global paths connected to the particular node in the local graph and their merged distances. Based on possible targets that cannot be merged and the distance to the local graph of each global target, it is decided which targets are merged.

All remaining global targets are connected with a global path which is constructed from both target's paths to the local graph.

8.2.3. Traveling Salesman Problem Solver

Once the local exploration finishes when no explorable nodes remain, a route connecting all global targets must be found. As previously stated, all active global targets

are connected with each other through global paths and to the robot's position via the global connections to the local graph. If there are no active global targets, the exploration is completed and RNE terminates.

The initial tour connecting all global targets and the robot is constructed chaining the targets ascendingly by their index, starting with the robot's position. Since the global targets are added chronologically, this initial tour often yields a good enough estimate. Also, only the first target is going to be the goal for the robot and the process is repeated afterwards.

The 2-opt swap TSP solver, which is described in Section 3.4.2, is applied to the initial tour to optimize it. The first target of the tour is omitted from the 2-opt swap as it is the robot's current position and cannot be changed. Furthermore, open TSP is utilized because the robot is not required to return to its current position. Open TSP is described in Section 3.4.3.

When no further improvement is detected in an iteration of 2-opt swap, the first target after the robot's position in the tour is selected as the goal for the robot. The global connection from the robot to it is sent to the navigation as a global plan.

2-opt swap is chosen over other heuristics because of the reduced computational complexity of $\mathcal{O}(n^2)$ and the resulting path still being within 5% of the Held-Karp lower bound [153]. Even though other heuristics, like 3-opt or Lin-Kernighan are more likely to deliver a better tour, their complexity is larger [153]. Because only the first goal in the tour is executed, the 2-opt swap is considered sufficient for this use case.

8.2.4. Global Navigation

The first global target is selected from the tour calculated by the 2-opt swap TSP solver. The corresponding global connection from the target to the robot's position is used to construct a path for the global navigation planner. It is introduced in Section 5.1.7 and improved in Sections 6.1.2 and 7.5.

The waypoints of the global connection are reversed because they are ordered from global target to robot. Also, intermediate points are added in between.

When the robot follows the waypoints towards a global target, a k-d tree is used to keep track of the waypoint closest to the robot. As soon as the robot reaches the previous to last waypoint, the local exploration is started and the local graph is initialized at the last waypoint, which is also the global target's viewpoint.

Furthermore, when the navigation to a global target fails, the next global target from the previously calculated tour is chosen as a goal. If there are no other targets left, the exploration is finished as well.

To navigate to the next global target, the robot follows the global connection of the failed target back to the local graph. Upon reaching the former target's first waypoint, the global connection from the local graph to the new global target is pursued. If another target fails, the procedure is repeated. When five consecutive targets fail, the exploration is terminated as the robot is assumed stuck.

8.2.5. Homing

RNE has the option to return the robot to the starting position after the exploration is finished, similar to the approaches utilized in the DARPA SubT Challenge, e.g., GBPlanner2 and TARE.

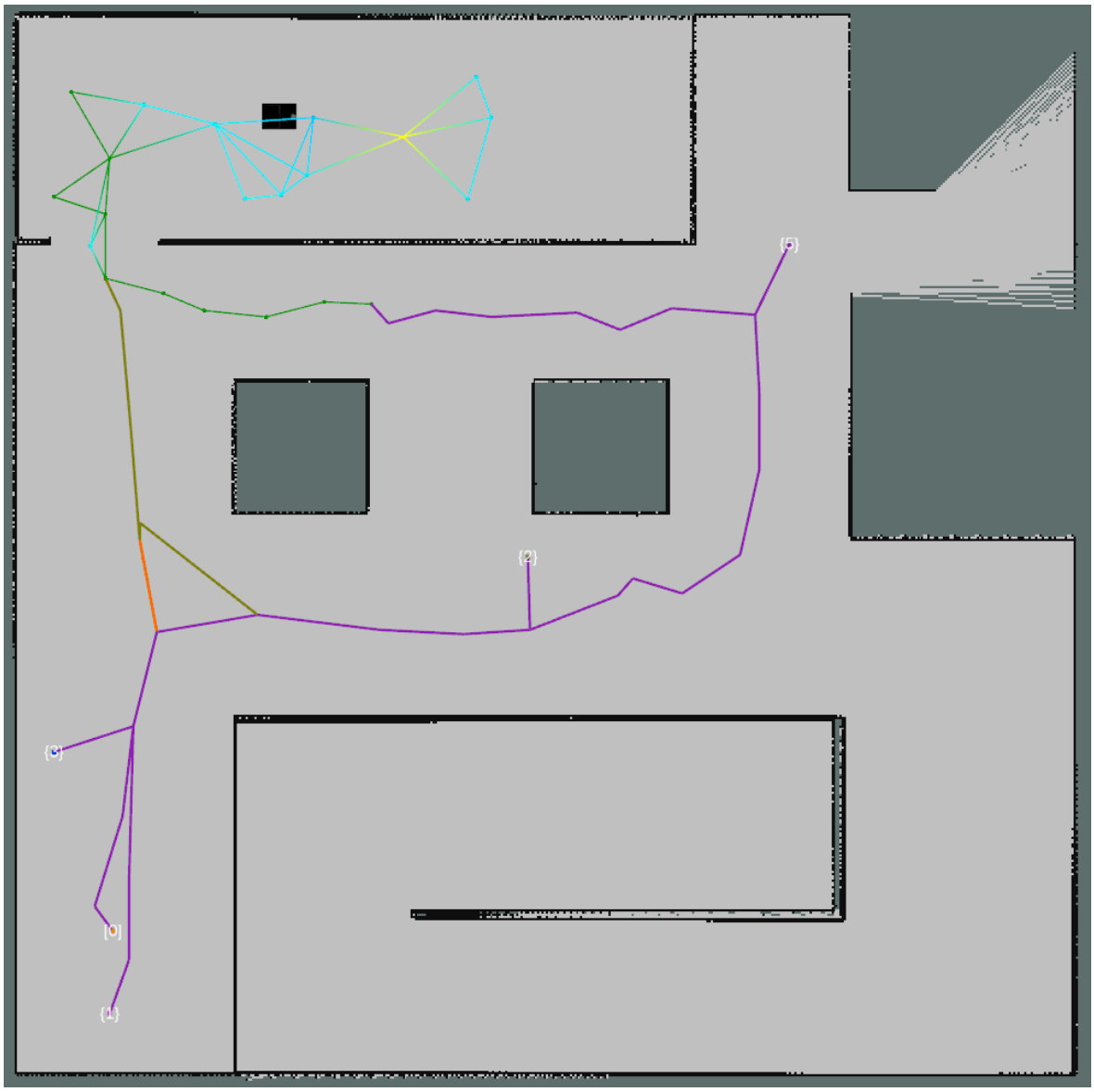


Fig. 8.5.: A combination of local and global exploration is shown in this screenshot taken in RViz of the environment introduced in Figure 6.2a. The local graph, which is set to a small radius for demonstration purposes, is placed around the robot in the top of the image. Green dots and lines symbolize explored nodes and edges while blue dots are unexplored nodes. The node colored yellow is the current goal of the robot.
The global graph's targets have numbers attached to them with curly braces for clustered targets. All targets except for the origin, which is the second target from the bottom of the image, have other targets merged into them. The global connections are shown in purple, orange and dark yellow and link local graph and global targets with each other.

Therefore, the origin of the exploration is added as a global target at the start. A global path with the local graph's root node as the connecting node is appended as well. The origin is treated as a special global target because it must not be merged with any other target. Therefore, it is simply ignored when applying the heuristic for clustering global targets described in Section 8.2.2.

Also, when using the 2-opt swap TSP solver, the origin must always be the last target in the route. To accomplish this, it is always inserted at the end of the route and it cannot be swapped.

When all other global targets have been explored, the origin is the last goal for the robot. It references the maintained global connection from the local graph to it for navigation.

Figure 8.5 shows an exemplary exploration with a local and global graph as well as the origin.

8.3. Implementation

This section goes into detail about the final implementation of the RNE algorithm described in this work. It elaborates where the different parts of the algorithm are located and how they interact while utilizing the ROS framework. Furthermore, the interfaces to other package like the navigation stack, OctoMap and RSM which is introduced in Chapter 4, are presented. In Figure 8.6, an overview of the packages, nodes, plugins and classes can be seen.

The complete RNE is available as an open-source metapackage. A metapackage bundles several other packages as described in Section 2.1. In the following, the description of the implementation is structured by the different packages.

8.3.1. rrg nbv exploration

This is the main package that contains all of the algorithms and heuristics introduced before. By supplying parameters to the respective nodes, the user sets various settings of the exploration. In addition to the user-defined settings mentioned in the previous description of RNE, a user can turn the local sampling, NAI, node movement and

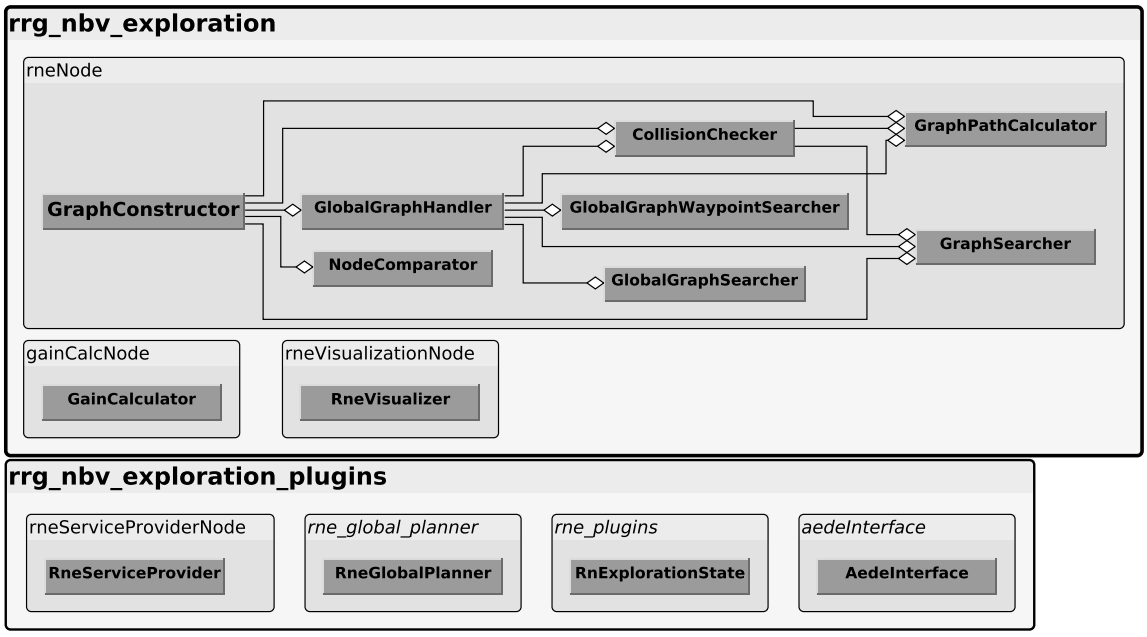


Fig. 8.6.: Class diagram showing all classes utilized in RNE grouped by the ROS nodes or plugins using them. Furthermore, the associations between the classes in the `rneNode` are depicted as well as the ROS packages containing the nodes, plugins and classes. Methods and attributes are omitted for brevity.

hybrid exploration on or off.

The package includes three nodes which are the `rneNode`, `gainCalcNode` and `rneVisualizationNode` that are described below.

rneNode

The `rneNode` accesses the `GraphConstructor` class which itself utilizes the majority of classes in the package. The `GraphConstructor`'s main method is called from the `rneNode` using a timer with a user-defined frequency. These classes are listed in the following. Shared pointers enable access to each class from every other class if necessary while only one class object is instantiated.

`GraphConstructor` The class `GraphConstructor` is the main class of the `rneNode` and runs the algorithm which builds the local graph. It manages and publishes the local graph's current state, selects and supervises the current goal for the robot and publishes the nodes to be updated in a ROS message topic. It subscribes to the updated nodes' topic as well.

Furthermore, it controls the switch between local and global exploration while also pruning the local graph based on its sliding area. It publishes the current goal and if it should be aborted. The `GraphConstructor` also includes services to start and stop the exploration.

`CollisionChecker` The `CollisionChecker` is responsible for all traversability checks which determine node placement and connections. It includes the algorithms introduced in Chapter 5.1.4 which check if a circular or rectangular area is traversable. Also, NAI and node movement algorithms proposed in Chapter 7.1 are placed in this class. Therefore, it subscribes to a user-defined OccupancyGrid map topic which holds the required information.

`GraphPathCalculator` When the robot moves from one node to the next, the `GraphPathCalculator` updates the path from each particular node to the robot. This path is maintained in the node and can be retrieved for the `RneGlobalPlanner`. Furthermore, the heading changes and the traversability along this path are calculated and added up for each node's cost function as shown in Equation (7.4). The path is based on Dijkstra's algorithm using the cost function for each node explained in Section 7.5.

`NodeComparator` The `NodeComparator` stores a list of all nodes which can still be explored. This list is used to determine the node with the best reward, which is then proposed as the next goal to explore.

`GraphSearcher` The `GraphSearcher` is an interface to the nanoflann header-only library for building k-d trees. All of the RRG's nodes are stored in a k-d tree to enable a fast nearest neighbor and radius search for constructing the graph and updating nodes in a radius.

`GlobalGraphHandler` The class `GlobalGraphHandler` is responsible for adding and merging global targets as well as their respective global connections as described in Section 8.2. It includes the 2-opt TSP solver to determine the next global goal and the global path towards it.

`GlobalGraphSearcher` This class also employs the nanoflann header-only library to store and access the global targets in a k-d tree for nearest neighbor and radius searches.

`GlobalGraphWaypointSearcher` The `GlobalGraphWaypointSearcher` uses the nanoflann header-only library on global connection waypoints for pruning the connections and determining the closest waypoint to the robot.

gainCalcNode

The `gainCalcNode` uses the `GainCalculator` class to calculate each node's gain using SRP which is introduced in Section 5.1.5. It also sets each node's height by obtaining the ground's height from ray tracing in the OctoMap. Nodes to be updated are subscribed to in the respective ROS topic and updated nodes, whose gains have been calculated, are published in another topic.

rneVisualizationNode

The `RneVisualizer` is the only class in the `rneVisualizationNode` and subscribes to the published local and global graph to visualize them in RViz. It shows local nodes as spheres colored according to the particular node's state listed below. The edges are shown as lines in between them which interpolate the nodes' colors they are connecting.

White The node's gain has not been calculated yet.

Blue The node's gain is calculated and the intensity of the color indicates its reward function. Dark blue for a higher reward and light blue for a lower reward.

Yellow This node is the robot's current goal node.

Light green This node has already been visited by the robot but is still not fully explored.

Orange The node has already been visited and is the current goal node for the robot.

Dark green This node is fully explored.

Red Navigation to this node failed previously.

It also publishes text visualization that shows the nodes' number and reward function. If activated by using parameters or rqt's reconfigure GUI[1], all parts of the reward and cost function can be shown individually as well.

Furthermore, the global targets are depicted as cubes with a unique, random color. Their index is visualized above them with curly brackets around if the target has other targets merged into it. Global connections are shown as lines and share the color of the target with the highest index that they are connected to. For example, if the global target with index 3 has connections to targets 1 and 2, the connections have the same color as target 3.

[1] `http://wiki.ros.org/rqt_reconfigure`

8.3.2. rrg nbv exploration msgs

This package contains all ROS messages and services defined for the RNE algorithm and makes them known to the ROS master.

This includes the messages that contain the local and the global graph. The former uses a list of local nodes and edges that each have their own message. The local node message stores its index, position, status and all required information for the cost and reward function as well as the indices of its connecting edges and the indices of the nodes in its path to the robot. It also includes references to all global targets which have this node as their connecting node to the local graph. The edges contain their own index as well as the indices of the connected nodes, orientation, length and the traversability cost.

The global graph message is separated into a list of targets and connections similar to the local graph message. In the global target message, index, position, indices of its global connections and merged distance are included. The global connection message holds its index, indices of the targets it connects, length in m and list of waypoints. The indices of the targets can have special values for the origin and the local graph. If a global path is connected to the local graph, the index of the connecting node in the local graph is stored as well.

The package also contains the messages for updating nodes which are exchanged between the `rneNode` and the `gainCalcNode`. Furthermore, it exposes services which can request the position of the goal proposed by RNE and the path towards it as well as update the current goal's status.

8.3.3. rrg nbv exploration plugins

This package is an auxiliary package that interfaces RSM and the ROS navigation package. It implements a state for exploration called `RnExplorationState` which retrieves the currently best goal from RNE and passes it on to RSM, that itself forwards it to the navigation. A node called `rneServiceProviderNode` is used to communicate between RSM and RNE by forwarding the status of the current goal navigation to RNE and if it is obsolete to RSM.

The `RneGlobalPlanner` is the global navigation planner introduced in Section 5.1.7 and refined in Sections 6.1.2 and 7.5. It retrieves the path to the current goal from RNE and uses it in the planning of the ROS navigation package.

The package also includes Unified Robot Description Format (URDF) and ROS launch files that simplify starting a simulation to test RNE. The URDF files describe different robot configurations to validate different scenarios, e.g., with a depth camera or a lidar scanner.

Alternatively, an interface to the AEDE's terrain analysis and local planner is provided. AEDE is introduced in Section 2.2. It forwards the path constructed for the `RneGlobalPlanner` directly to AEDE's local planner which follows it waypoint by waypoint.

Compared to the ROS navigation stack, AEDE is considerably faster in reaching new goals and switching between goals but sacrifices the ability to target a desired orientation at a goal. Therefore, it is only suited for sensor configurations with an all-around FoV.

Furthermore, it is less precise in reaching the goals exactly which leads to sub-optimal coverage compared to the calculated, expected gain.

8.4. Comparison to Local-Only Exploration

To highlight the improvements to the previous iteration, hybrid RNE is compared to plain RNE introduced in Chapter 7. The same evaluation setup as shown in Section 7.7.1 is used but the vehicle configuration with the camera is discarded because it cannot be utilized with the AEDE-based local planner. The number of runs for each configuration is increased from 10 to 15.

For the local-only exploration, which is referenced as R, and RNE with the global exploration, all cost factors are set to 1. Topology-based NAI and re-updating nodes are activated for all runs. For RNE, global exploration is active and the local graph radius is set to $r_G = 20$ m.

Furthermore, the AEDE-based planner mentioned in Section 8.3.3 is used in the simulation as an alternative to the comparably slow ROS navigation stack. The corresponding interface for RNE enables utilizing the terrain analysis and local planner which allows increased velocities and faster goal position changes. Its goal is to increase the overall exploration speed.

The AEDE-based local planner does not allow to select the goal orientation of the robot nor is it as fine-grained as the navigation stack when using the default parameters defined by its developers in the open-source examples. Therefore, the proposed interface has a higher goal tolerance which means it must not be as close to the goals as the Dynamic Window Approach (DWA) local planner.

Runs with this approach are referenced as RNEA and have a goal tolerance of 0.35 m from the robot's center compared to 0.2 m used by the navigation stack's DWA planner.

Figure 8.7 and Table 8.1 show the results from the simulations. While the overall duration and path length of RNE are slightly inferior to R, the goal to reduce the computation is achieved. The mean algorithm run time is decreased by approximately 21% for L-ME and nearly 18% for L-CE.

Tab. 8.1.: Comparison between local-only RNE as R, hybrid RNE as RNE, RNE using the AEDE-based local planner as RNEA and RNE with optimized parameters as RNE+. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the lidar configuration as L on the medium and cave environments as ME and CE respectively. The best mean values for each variant are printed in bold letters.

Configu-ration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
L-ME R	765.00	79.37	**191.82**	34.47	1590.2	6.1	0.442	0.032	0	15
L-ME RNE	791.54	102.68	208.68	36.30	1586.4	5.7	0.349	0.038	2	15
L-ME RNEA	422.00	42.22	254.26	25.22	1616.8	11.4	0.698	0.026	1	15
L-ME RNE+	**211.00**	34.70	223.84	38.00	**1893.6**	57.7	**0.285**	0.026	0	15
L-CE R	2838.00	236.71	637.90	52.49	8401.9	65.8	0.531	0.016	0	15
L-CE RNE	3257.14	230.72	754.19	73.69	8442.4	78.1	0.436	0.045	1	15
L-CE RNEA	2562.50	234.76	1222.00	146.51	8013.6	6.3	0.751	0.014	13	15
L-CE RNE+	**433.27**	58.81	**489.41**	66.34	**9699.9**	386.4	**0.317**	0.024	0	15

In Figure 8.7, it can also be seen that the global exploration's map coverage and distance are similar to R during most of the run. Only at the end, the global exploration adds some overhead, probably due to remaining global targets that only add marginal map coverage.

Just switching to the AEDE-based local planner while all other parameters stay the same, leads to a large amount of failed exploration attempts and a vastly increased algorithm run time. This is caused by the faster planning and movement which does not leave enough time to sample goals and calculate their gain before the exploration finished timer is triggered. This causes a switch to the global exploration and leads to stopping the exploration prematurely.

Therefore, the parameters for RNE and some of the other ROS packages are changed to make the exploration as effective as possible. This should also increase its performance compared to the state-of-the-art approaches presented in the next section. The parameters listed in the following are changed to optimize RNE:

- Timer duration $t_{exit} = 5$s
- Horizontal steps $\Delta_\varphi = 20°$,
- Radius steps $\Delta_r = 0.35$m
- Sensor range for gain calculation $r_{max} = 20$m
- Minimum view score $G_{min} = 0.03$ for ME and $G_{min} = 0.15$ for CE
- OctoMap resolution $e_V = 0.35$m
- Distance factor $d = 0.15$
- Local sampling radius $r_{ls} = 10$m
- Grid map resolution $r_m = 0.1$m

With these parameters, RNE with the AEDE-based local planner, which is referenced as RNE+, outperforms basic RNE and RNEA in nearly every category. Compared to the basic RNE, the duration is decreased by 73% for L-ME and 87% for L-CE. The algorithm run time is reduced by 18% and 27% respectively. The path length for L-ME is increased by 7% because the exploration favors more rewarding nodes over nearby nodes which leads to some back-and-forth motion. For L-CE, the path length is decreased by 35%.

The mapped volume is more difficult to compare because the OctoMap voxel edge length is increased from 0.1 m for R, RNE and RNEA to 0.35 m for RNE+. Voxels on the outer boundaries of the closed environments can extend the mapped volume further for the larger edge length. For example, the box-shaped ME's dimensions are 25x25x2.5 m and its volume is 1562.5 m^3. For a resolution of 0.1 m, the dimensions can extend up to 25.2x25.2x2.7 m which results in a volume of 1714.61 m^3. An edge length of 0.35 m can result in dimensions of 25.7x25.7x3.2 m which equals a volume of 2113.57 m^3. This is an increase in map volume of around 20% which roughly equals the increase of the mapped volume from RNEA to RNE+ for L-ME and L-CE. It indicates that the mapped volume remains approximately the same when considering the different resolutions.

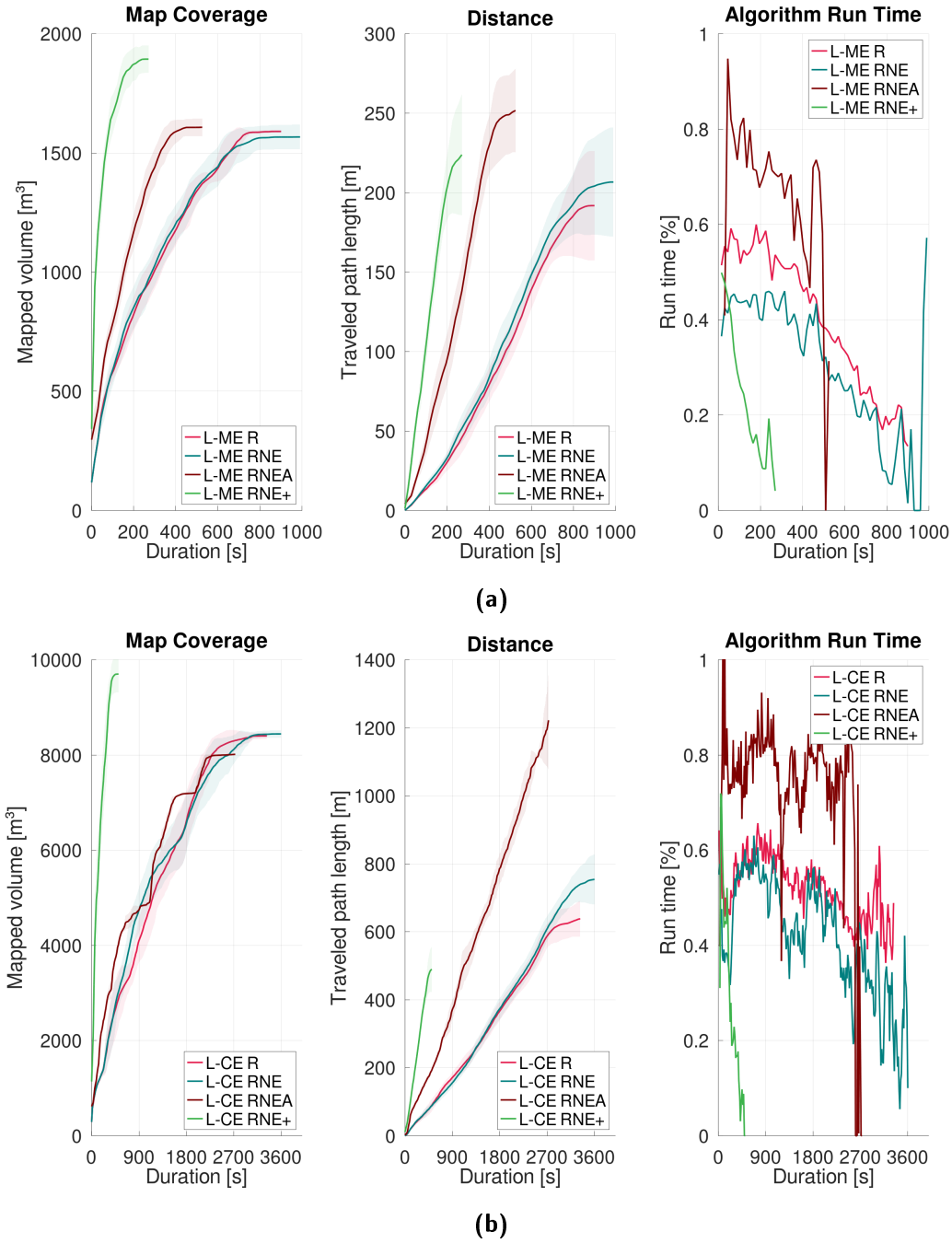


Fig. 8.7.: Mean mapped volume, path length and algorithm run time over the duration for local-only RNE as R, hybrid RNE as RNE, RNE using the AEDE-based local planner as RNEA and RNE with optimized parameters as RNE+. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the lidar configuration in the medium environment and (b) the lidar configuration in the cave environment.

Figure 8.7 underlines the performance increase of RNE+ compared to RNEA and basic RNE. It shows that RNE+ is able to explore larger amounts of the map significantly faster than the previous attempts. Therefore, RNE+ is compared to the state-of-the-art approaches introduced next.

8.5. Comparison to State-of-the-Art Approaches

To demonstrate the effectiveness of the exploration approach introduced in this work, it is compared to several state-of-the-art exploration algorithms for UGVs. The optimized approach RNE+ shown in the previous section and the other algorithms are executed in both environments shown before and referred to as ME and CE.

The other approaches are DSVP, TARE and GBPlanner2, which is abbreviated as GBP2. RNE uses the artificial box robot shown in Figure 7.8b. While DSVP and TARE simulations are based on the simulated robot they use in AEDE, which was introduced by the same authors. GBP2 is executed using their own simulated robot as well. Because their open-source approaches are closely intertwined with the particular robot and planning approach, these are not changed for the comparison. The robot dimensions are comparable to the robot depicted in Figure 7.8b. DSVP, TARE and GBPlanner2 are detailed in Section 2.4.

Furthermore, the default parameters from the open-source repositories of the state-of-the-art approaches are used during this comparison. For GBP2, the parameters declaring a bounded exploration area are removed as the ME and CE environments are enclosed. While DSVP and TARE publish their exploration run time, a timer to calculate it for GBP2 is added to its source code.

The path length and mapped volume are evaluated, like in previous simulations, by calculating the length between the connected positions of the robot in the map and counting observed OctoMap voxels respectively. Therefore, an OctoMap node is added to TARE and GBP2 runs with a resolution of 0.35 m for comparability. DSVP already uses OctoMap with the same resolution for gain calculation.

Table 8.2 shows the results of the simulations. It can be seen, that DSVP and TARE outperform RNE while it is still more efficient than GBP2. DSVP and TARE respectively take around half or one-third of the time to explore ME but are only twice as fast for the larger CE. The path traversed by the robot during the exploration is increased by 50% compared to DSVP for ME and 108% compared to TARE. For CE, it is increased by 17% and 20% respectively. The explored map volume is approximately the same for ME and compared to TARE in CE. But RNE explores 12% more map volume in CE compared to DSVP.

Tab. 8.2.: Comparison between RNE, DSVP, TARE and GBPlanner2 as GBP2 which shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs. The lidar configuration as L is used in the medium and cave environments as ME and CE respectively. The best mean values for each variant are printed in bold letters.

Configu-	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
ration	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
L-ME RNE	211.00	34.70	223.84	38.00	1893.6	57.7	0.285	0.026	0	15
L-ME DSVP	102.67	21.79	149.55	28.11	1908.5	29.6	0.077	0.018	0	15
L-ME TARE	**73.00**	11.15	**107.49**	15.16	**1912.4**	6.8	0.229	0.019	0	15
L-ME GBP2	415.00	59.64	152.27	20.27	1799.4	94.6	**0.028**	0.003	0	15
L-CE RNE	433.27	58.81	489.41	66.34	9699.9	386.4	0.317	0.024	0	15
L-CE DSVP	288.00	26.71	417.94	37.54	8679.4	239.6	0.079	0.012	0	15
L-CE TARE	**278.57**	15.25	**408.23**	21.32	9568.6	89.5	0.303	0.011	1	15
L-CE GBP2	1136.79	50.33	465.84	21.49	**10027.3**	79.8	**0.026**	0.002	1	15

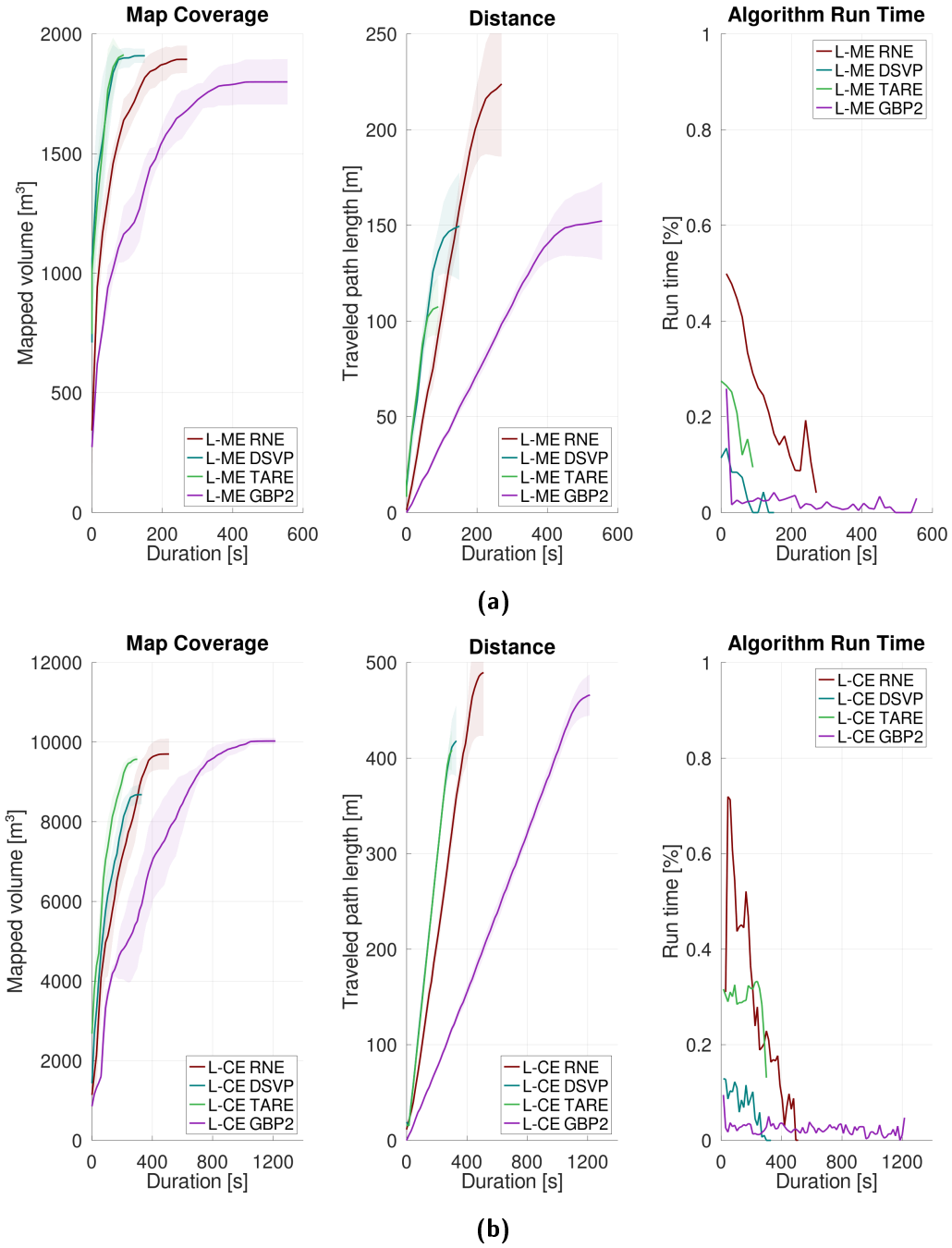


Fig. 8.8.: Mean mapped volume, path length and algorithm run time over the duration for RNE, DSVP, TARE and GBPlanner2 as GBP2. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the lidar configuration in the medium environment and (b) the lidar configuration in the cave environment.

While RNE is 50% faster than GBP2 in ME and 62% in CE, its path length is increased by 47% for the former and 5% for the latter. The explored map volume differs only by 5% for ME and 3% for CE.

The algorithm run time of RNE is magnitudes larger than DSVP and very similar to TARE while GBP2 has the lowest computation time. This contradicts the findings in [12] and [18]. They measure a significantly decreased run time of their approaches DSVP and TARE compared to GBP2. On their website, the authors of DSVP and

TARE also show that TARE's algorithm run time is lower than DSVP's in all of the displayed tests [159]. Therefore, it seems difficult to directly compare the algorithm run times to each other as they heavily depend on the utilized processor and system.

In Figure 8.8, it can be seen that DSVP and TARE explore more volume earlier while TARE slightly outperforms DSVP. RNE shows a similar early exploration rate but flattens out a bit earlier and therefore takes more time to finish the exploration. The map coverage and distance curve of GBP2 is significantly flatter which indicates that it does not cover as much room as the other approaches. The algorithm run time of RNE is large at the start but significantly decreases over time as more areas become explored and the gain calculation's larger load drops.

This comparison shows that RNE is able to compete with the current state-of-the-art approaches while it is independent of the utilized robot, sensors and SLAM. This makes it more flexible and easier to use in various applications.

8.6. Experiment

To verify that RNE also works under real circumstances and not solely in simulations, an experiment is conducted in the rock-cut cellars in Lauf an der Pegnitz, Germany. The robot Georg is used which was the first robot from the Nuremberg Institute of Technology's AutonOHM team participating in the RoboCup RRL Competition, that is introduced in Section 1.1.2. It was built in 2012 and is a skid-steer drive with roughly the same dimensions as the Clearpath Robotics Husky and the simulated box robot used in the simulations.

Georg is equipped with two Velodyne VLP-16 lidars as can be seen in Figure 8.9 but has no IMU nor wheel odometry. Only the more elevated Velodyne in the back is used during the experiment. The ROS package laser scan matcher[2] is deployed to calculate the odometry for GMapping which is the utilized SLAM approach. Therefore, the mean ring from the Velodyne is used in the 2D scan matching as it is aligned horizontally.

Furthermore, the Velodyne points are required for a traversability analysis introduced by Koch et al. [3] which is utilized to assess the uneven ground in the rock-cut cellars

[2] `http://wiki.ros.org/laser_scan_matcher`

Fig. 8.9.: Robot Georg in the rock-cut cellars in Lauf an der Pegnitz, Germany. The left image shows a transition from rock-cut ground to cobblestones and the right image depicts the robot in front of debris blocking a corner of the cellar.

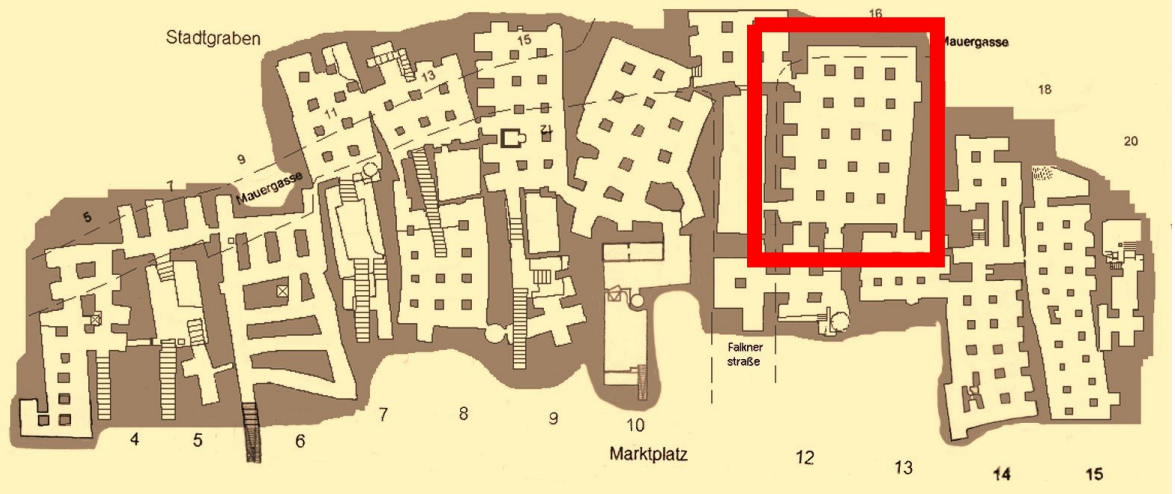


Fig. 8.10.: Plan of the rock-cut cellars in Lauf an der Pegnitz, Germany [160] with a red square to mark the area in which the experiment takes place.

featuring steps and holes. It publishes a grid map whose values correspond to the traversability of the particular patch based on the points in this and neighboring grid cells and the robot's kinematics. Further details about the traversability analysis can be obtained from the publication.

Figure 8.10 shows the floor plan of the rock-cut cellars with a red rectangle marking the area that was explored in this experiment. Most of the cellars are connected via stairs or steps which cannot be traversed by the robot. The marked cellar was selected as it is one of the largest and also has electric lights.

In Figure 8.9, robot Georg can be seen in the rock-cut cellars during the experiment. The experiment was conducted before the AEDE-based local planner interface has been implemented. Therefore, the ROS navigation stack was used including the DWA local planner. The robot was placed in the middle of the cellar at the start of the experiment.

The goal of the experiment was to explore the complete cellar using RNE's final iteration. Due to untraversable connections to neighboring cellars, the area was limited to the cellar marked in Figure 8.10. Therefore, only a local exploration was conducted as the area is not large enough for a transition between local and global exploration to take place.

Figure 8.11 shows that RNE successfully explored the complete cellar, in which the robot was deployed, in 7 minutes and 32 seconds. Due to the missing IMU, turning on the spot sometimes caused a wrong localization which led to duplicated walls in the grid map, e.g. in the top left corner. The approach proposed in this work is able to thoroughly explore difficult environments under real-life conditions.

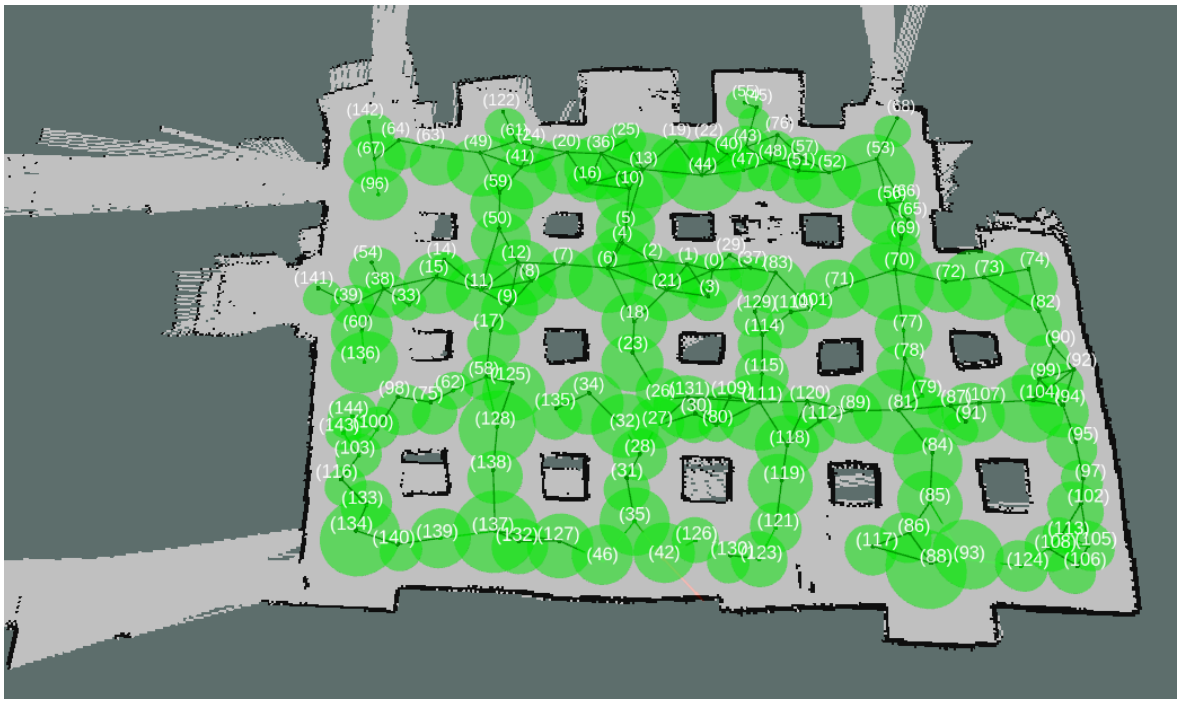


Fig. 8.11.: Explored map of the rock-cut cellars in Lauf an der Pegnitz, Germany. The grid map shows obstacles in black, free cells in light gray and unknown cells in dark gray. Green dots and lines show nodes and edges of the graph respectively and lighter green circles the areas of the particular nodes whose indices are shown in brackets.

8.7. Conclusion

The fourth and final iteration of RNE enables efficient exploration of larger areas than previous iterations. While the total computation time can still grow because of an increasing number of global targets, the largest part of RNE's computation time relates to the maximum local graph size. It limits the number of nodes that can be placed simultaneously and whose gains have to be calculated.

The efficient clustering of global targets and the employed 2-opt TSP solver help in reducing the total computation time as can be seen in the results of the simulations in Table 8.1.

Due to the transitions between local and global exploration, the exploration duration is increased as well as the overall path length. The hybrid RNE iteration is not as efficient regarding exploration duration and traveled path length but fulfills the goal of reducing the growing computation time.

When comparing RNE with the ROS navigation planner to the state-of-the-art approaches, it performs significantly worse. However, the other approaches employ their own planners which are tailored to the specific robot and therefore more efficient. These approaches are DSVP, TARE and GBPlanner2.

By utilizing the AEDE-based local planning approach and optimizing the parameters, RNE's final iteration is able to perform on the same level as the state-of-the-art approaches. These were utilized in the DARPA SubT Challenge and are tailored to specific robots while RNE can be used with a variety of UGVs.

Finally, RNE's usage is demonstrated in a real-world experiment which verified that it can be utilized in difficult, GPS-denied environments and successfully explore them.

9. Conclusion

In this work's final chapter, its research is recapitulated. The contributions are summarized and possible directions for future research are presented, followed by a closing remark.

9.1. Contributions

The research in this work focuses on autonomous exploration using UGVs for underground, GPS-deprived environments. It was motivated by the EIT RawMaterial's UNDROMEDA project and the exploration category of the RoboCup RRL. As both intended applications of the research had different robots and sensor configurations, the exploration approach needed to be versatile and adaptable which ruled out all of the existing open-source explorations available at the time.

To alleviate the development and application of an exploration approach, a state machine for mobile robots was created. Furthermore, it should offer the possibility to inspect and repeatedly patrol certain areas which was a requirement in the UNDROMEDA project and the RoboCup RRL as well.

This state machine was developed for ROS and open-sourced as Robot Statemachine (RSM). A paper about it was published in 2020 [1] and it was successfully utilized by the AutonOHM team from the Nuremberg Institute of Technology in the RoboCup GermanOpen RRL in 2019 and the RoboCup RRL in Sydney in 2019.

As sampling-based approaches for UGVs with an open-source ROS package were not available after the RSM was finished, a new approach had to be developed. Inspired by RH-NBVP, which was introduced by Bircher et al. [6] for UAVs, an RRT-based exploration algorithm was developed to gradually improve it compared to its inspirational predecessor. Furthermore, the goal was to adapt it to UGVs.

The first iteration of this development features a traversability evaluation based on a grid map and a sparse, decoupled gain calculation for each node in the RRT called Sparse Ray Polling (SRP). SRP which is based on SRC introduced by Selin et al. [7], uses an OctoMap and further reduces the number of required samples to decrease the computation time. The decoupled calculation means that it is executed in a separate thread from the remaining exploration algorithm. This allows to abandon the current goal node if a better one becomes available and eliminates the need to wait until all gains are calculated after reaching a goal. Furthermore, a persistent tree is utilized as well as a global navigation planner that follows the tree's edges to the designated goal. This goal is selected using a function that considers each particular node's gain and the distance from the robot to the node along the tree's edges.

The second iteration replaces RRT with RRG which enables finding optimal paths through the graph between the robot and the goal. Combined with Dijkstra's algorithm to find the shortest path in the graph, the goal selection and global navigation planner are significantly improved.

This research was published in 2021 [2]. The exploration approach was named Random-Sampling-Based Next-Best View Exploration (RNE) and the algorithm was made available as an open-source ROS package. A comparison to re-implementations of RH-NBVP and AEP for ground-based robots, which demonstrates RNE's superiority, was also conducted.

The third iteration introduces a topology-based graph construction method which means that nodes are placed as central as possible according to the environment's topology. To achieve this, the node areas are inflated until they collide with an obstacle or until a maximum radius is reached which is as large as the sensor's maximum range. In case of a collision, the node is moved away from the obstacle and can continue to grow in size to create a graph with larger node areas. This is advantageous because the node centers, which can serve as goals for the robot or parts of the paths to the goals, are further away from obstacles. This reduces the risk of collision and increases the volume observable by the robot's sensors.

In addition to the topology-based graph, a more sophisticated reward function was introduced that considers the traversability and heading change along the path to each particular node in addition to the distance and node gain. Furthermore, the node radius is included in the cost function which leads to favoring nodes with large areas. This new cost function includes adjustable parameters that enable optimizing the exploration focus for specific scenarios.

To analyze the impact of these parameters, an exhaustive experiment was conducted in which different combinations of them were run in simulations. A total of 1350 runs for 135 different combinations were executed that revealed the influence of the particular parameters on the exploration performance.

In the third iteration, re-updating node gains was also added to the algorithm. This causes the re-calculation of node gains around the robot while it is moving from node to node. This allows abandoning goals whose gains are significantly reduced while the robot navigates towards them. It increases the efficiency of the exploration but also increases the algorithm run time.

The final iteration of RNE was inspired by several approaches featuring a distinction between local and global exploration [7, 9, 12, 14, 18, 19, 20]. Some of them were utilized in the DARPA SubT Challenge. Based on them, the previously described topology-based algorithm became the local part of the exploration while a global approach was added.

This requires the ability to remove nodes from the local graph as it is restricted to a user-definable radius around the robot that moves with it. When a node falls outside the radius due to the robot's movement, it is removed. New nodes are only sampled inside the current area around the robot. Nodes, that are removed but still offer enough gain, are added to the global exploration as global targets.

These global targets are connected to the local graph using global paths that are built from removed nodes and edges. These paths are always connected to one of the local graph's nodes. If multiple paths are connected to the same node, connecting paths between the two or more particular global targets are established.

As soon as the local graph does not offer any more explorable nodes, the global exploration is started. All global paths are continued to the robot's current position. Global targets without a global path connecting them to each other are connected through the shortest path in the local graph between their respective connecting nodes.

A 2-opt TSP solver is employed to find the most efficient order to visit all existing global targets regarding the overall distance. The first global target in this order is selected as the next goal for the robot. When it is reached, a local exploration is initialized at the target's position.

Global targets are efficiently clustered using their Euclidean distance and distance to each other via their connecting global paths. This reduces the required computation time to find the optimal order using a TSP solver and to maintain the attached global paths.

Furthermore, a homing option was proposed that adds the origin from which the exploration is started as a global target. After all other global targets have been explored, the robot follows the global path to the origin of the exploration.

Also, local and global exploration include sophisticated ways to handle navigation failures to a particular goal to keep the exploration going and finally return the robot to the origin, even after one or more goals have been blocked.

The proposed RNE was compared to three state-of-the-art approaches which are DSVP, TARE and GBPlanner2. These were deployed in the DARPA SubT Challenge and team CERBERUS, that developed and utilized GBPlanner2, won the finals. The comparison showed that RNE is able to compete with the previously listed approaches in the executed simulations.

In addition, RNE was deployed in an experiment to autonomously explore a rock-cut cellar in Lauf an der Pegnitz, Germany. A traversability analysis was used with RNE which was introduced by Koch et al. [3]. It was successfully demonstrated that the proposed approach can be utilized in simulations and real-world scenarios to explore and map an unknown underground environment.

9.2. Future Research

There are several directions in which this work could be extended in future research. Recent publications in the field of exploration algorithms for mobile robotics revolved heavily around the DARPA SubT Challenge and revealed that multi-robot and UAV-based exploration approaches as well as approaches for legged robots, are the current state of the art. Because RNE can already be applied to a variety of ground robots, it can be utilized with legged robots. But extensions to enable UAVs to use RNE and which allow a coordinated multi-robot approach, could be a possible direction for future research.

As UAVs can move freely on the z-axis, the node sampling has to be extended in this axis. The current traversability checks would require an additional metric to translate volumetric collision checks into a 2D grid map. An ESDF-based mapping approach like Voxblox introduced by Oleynikova et al. [149] could alleviate collision checking as every voxel stores a distance to the nearest obstacle. This can also be utilized to implement an efficient topology-based graph construction as new nodes can move towards positions with a larger distance to obstacles.

A transformation of a 3D ESDF map to a 2D grid could be used to implement this efficient topology-based graph construction for the ground-based exploration case. The ESDF could additionally be used for sparse gain calculation and replace the OctoMap.

A multi-robot exploration could be realized with a global graph shared between the robots while each of them runs its own local exploration. To efficiently assign global

targets to each robot, a multi-robot TSP solver could be employed.

Different approaches to include the node gain in the reward function would also be an option for research. The current solution requires the user to set a minimum view score which needs experience to select an efficient value. Finding clusters of unexplored voxels in the gain calculation could be a solution to avoid classifying nodes with widely distributed unexplored voxels as worth exploring. But this would also require a user-defined value to define which cluster sizes are worth exploring.

A dynamic calculation of a threshold to rate nodes as worth exploring could be another valuable research. This would also consider changing environments as the number of unexplored voxels around a node is higher in large, open areas compared to narrow tunnels. A Neural Network could be trained and applied to execute this dynamic calculation.

9.3. Closing Remarks

The DARPA SubT Challenge significantly amplified the research effort on exploration algorithms and birthed a multitude of sophisticated exploration approaches specialized in underground environments. The massive funding led to large teams from universities all over the world putting great effort into research and engineering developments. Their findings were a great inspiration to improve the approach proposed in this work.

The increased availability of legged robots recently also shifted a lot of research towards their use as they have significant advantages over wheel- and track-driven UGVs in difficult underground environments. Future applications regarding autonomous exploration are probably going to revolve heavily around the use of legged robots in combination with UAVs.

Unfortunately, RNE could only be tested on a real robot very late in the research due to the Covid pandemic-related travel restrictions. When the restrictions were finally lifted, the UNDROMEDA project's funding had ended and no experiment in an underground mine with the robot could be conducted. Therefore, a low-budget experiment in the rock-cut cellars had to suffice as well as a heavy reliance on simulations. Nonetheless, the efficiency of the proposed exploration algorithm was successfully demonstrated.

Bibliography

[1] Marco Steinbrink, Philipp Koch, Stefan May, Bernhard Jung, and Michael Schmidpeter. "State Machine for Arbitrary Robots for Exploration and Inspection Tasks". In: *Proceedings of the 2020 4th International Conference on Vision, Image and Signal Processing*. New York, NY, USA: ACM, Dec. 2020, pp. 1–6. DOI: 10.1145/3448823.3448857.

[2] Marco Steinbrink, Philipp Koch, Bernhard Jung, and Stefan May. "Rapidly-Exploring Random Graph Next-Best View Exploration for Ground Vehicles". In: *2021 European Conference on Mobile Robots (ECMR)*. IEEE, Aug. 2021, pp. 1–7. DOI: 10.1109/ECMR50962.2021.9568785.

[3] Philipp Koch, Marco Steinbrink, Stefan May, and Andreas Nuechter. "Traversability Analysis for Wheeled Robots using Point-Region-Quad-Tree based Elevation Maps". In: *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, Apr. 2022, pp. 192–197. DOI: 10.1109/icarsc55462.2022.9784803.

[4] *Technology readiness levels (TRL); Extract from Part 19 - Commission Decision C(2014)4995*. 2014. URL: https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf (visited on 07/04/2022).

[5] *League Overview – RoboCupRescue Robot League*. URL: https://rrl.robocup.org/league-overview/ (visited on 06/30/2022).

[6] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. "Receding horizon next-best-view planner for 3D exploration". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2016-June.January 2018 (2016), pp. 1462–1468. DOI: 10.1109/ICRA.2016.7487281.

[7] Magnus Selin, Mattias Tiger, Daniel Duberg, Fredrik Heintz, and Patric Jensfelt. "Efficient autonomous exploration planning of large-scale 3-d environments". In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019), pp. 1699–1706. DOI: 10.1109/LRA.2019.2897343.

[8] Zhefan Xu, Di Deng, and Kenji Shimada. "Autonomous UAV Exploration of Dynamic Environments Via Incremental Sampling and Probabilistic Roadmap". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2729–2736. DOI: 10.1109/LRA.2021.3062008.

[9] Lukas Schmid, Victor Reijgwart, Lionel Ott, Juan Nieto, Roland Siegwart, and Cesar Cadena. "A Unified Approach for Autonomous Volumetric Exploration of Large Scale Environments under Severe Odometry Drift". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4504–4511. DOI: 10.1109/LRA.2021.3068954.

[10] Tung Dang, Frank Mascarich, Shehryar Khattak, Christos Papachristos, and Kostas Alexis. "Graph-based Path Planning for Autonomous Robotic Exploration in Subterranean Environments". In: *IEEE International Conference on Intelligent Robots and Systems* November (2019), pp. 3105–3112. DOI: 10.1109/IROS40897.2019.8968151.

[11] *DARPA subterranean (SubT) challenge*. URL: https://www.darpa.mil/program/darpa-subterranean-challenge (visited on 06/29/2022).

[12] Chao Cao, Hongbiao Zhu, Howie Choset, and Ji Zhang. "TARE: A Hierarchical Framework for Efficiently Exploring Complex 3D Environments". In: *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, July 2021. DOI: 10.15607/rss.2021.xvii.018.

[13] Fan Yang, Dung Han Lee, John Keller, and Sebastian Scherer. "Graph-based Topological Exploration Planning in Large-scale 3D Environments". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2021-May (2021), pp. 6768–6774. DOI: 10.1109/ICRA48506.2021.9561830.

[14] Mihir Kulkarni et al. *Autonomous Teamed Exploration of Subterranean Environments using Legged and Aerial Robots*. Tech. rep. 2021. DOI: 10.48550/ARXIV.2111.06482.

[15] Sung Kyun Kim, Amanda Bouman, Gautam Salhotra, David D. Fan, Kyohei Otsu, Joel Burdick, and Ali Akbar Agha-Mohammadi. "PLGRIM: Hierarchical Value Learning for Large-scale Autonomous Exploration in Unknown Environments". In: (2021), pp. 652–662. DOI: 10.48550/ARXIV.2102.05633.

[16] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. "The office marathon: Robust navigation in an indoor office environment". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2010), pp. 300–307. DOI: 10.1109/ROBOT.2010.5509725.

[17] Sertac Karaman and Emilio Frazzoli. "Incremental sampling-based algorithms for optimal motion planning". In: *Robotics: Science and Systems*. Vol. 6. 2011, pp. 267–274. DOI: 10.15607/rss.2010.vi.034.

[18] Hongbiao Zhu, Chao Cao, Yukun Xia, Sebastian Scherer, Ji Zhang, and Weidong Wang. "DSVP: Dual-Stage Viewpoint Planner for Rapid Exploration by Dynamic Expansion". In: *IEEE International Conference on Intelligent Robots and Systems*. 2021, pp. 7623–7630. DOI: 10.1109/IROS51168.2021.9636473.

[19] Tung Dang, Shehryar Khattak, Frank Mascarich, and Kostas Alexis. "Explore Locally, Plan Globally: A Path Planning Framework for Autonomous Robotic Exploration in Subterranean Environments". In: *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, Dec. 2019, pp. 9–16. DOI: 10.1109/ICAR46387.2019.8981594.

[20] Mihir Dharmadhikari, Tung Dang, and Kostas Alexis. "Appendix for the Motion Primitives-based Path Planning for Fast and Agile Exploration Method". In: (2020). DOI: 10.48550/ARXIV.2012.03228.

[21] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. "ROS: an open-source Robot Operating System." In: *ICRA Workshop on Open Source Software*. 2009, pp. 679–686.

[22] Tully Foote and Katherine Scott. *Community Metrics Report Contents*. Tech. rep. Open Source Robotics Foundation, 2020. URL: http://download.ros.org/downloads/metrics/metrics-report-2020-07.pdf.

[23] Nathan Koenig and Andrew Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. IEEE, 2004, pp. 2149–2154. DOI: 10.1109/IROS.2004.1389727.

[24] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. "YARP: Yet another robot platform". In: 3.1 (2006), pp. 043–048. DOI: 10.5772/5761.

[25] José-Luis Blanco. "Contributions to Localization, Mapping and Navigation in Mobile Robotics". PhD thesis. PhD. in Electrical Engineering, University of Malaga, nov 2009. URL: https://riuma.uma.es/xmlui/handle/10630/9841.

[26] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (May 2022). DOI: 10.1126/scirobotics.abm6074.

[27] Tully Foote and Katherine Scott. *Community Metrics Report Contents*. Tech. rep. 2021. URL: http://download.ros.org/downloads/metrics/metrics-report-2020-07.pdf.

[28] Tully Foote. "Tf: The transform library". In: *IEEE Conference on Technologies for Practical Robot Applications, TePRA* (2013). DOI: 10.1109/TePRA.2013.6556373.

[29] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2005. 2005, pp. 2432–2437. DOI: 10.1109/ROBOT.2005.1570477.

[30] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improved techniques for grid mapping with Rao-Blackwellized particle filters". In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46. DOI: 10.1109/TRO.2006.889486.

[31] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics and Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977.

[32] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous Robots* 34.3 (2013), pp. 189–206. DOI: 10.1007/s10514-012-9321-0.

[33] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. "Creational Patterns". In: *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995, pp. 305–313.

[34] David Harel. "Statecharts: a visual formalism for complex systems". In: *Science of Computer Programming* 8.3 (1987), pp. 231–274. DOI: 10.1016/0167-6423(87)90035-9.

[35] Steve Cook, Conrad Bock, Pete Rivett, Tom Rutt, Ed Seidewitz, Bran Selic, and Doug Tolbert. *Unified Modeling Language (UML) Version 2.5.1*. Standard. Object Management Group (OMG), Dec. 2017. URL: https://www.omg.org/spec/UML/2.5.1.

[36] Jonathan Boren and Steve Cousins. "The SMACH high-level executive". In: *IEEE Robotics and Automation Magazine* 17.4 (Dec. 2010), pp. 18–20. DOI: 10.1109/MRA.2010.938836.

[37] Hai Nguyen, Matei Ciocarlie, Kaijen Hsiao, and Charles C Kemp. "ROS commander (ROSCo): Behavior creation for home robots". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2013, pp. 467–474. DOI: 10.1109/ICRA.2013.6630616.

[38] Michalis Foukarakis, Asterios Leonidis, Margherita Antona, and Constantine Stephanidis. "Combining finite state machine and decision-making tools for adaptable robot behavior". In: *Universal Access in Human-Computer Interaction. Aging and Assistive Environments*. Springer International Publishing, 2014, pp. 625–635. DOI: 10.1007/978-3-319-07446-7_60.

[39] Barry Ridge, Timotej Gaspar, and Ales Ude. "Rapid state machine assembly for modular robot control using meta-scripting, templating and code generation". In: *IEEE-RAS International Conference on Humanoid Robots* (2017), pp. 661–668. DOI: 10.1109/HUMANOIDS.2017.8246943.

[40] Philipp Schillinger, Stefan Kohlbrecher, and Oskar Von Stryk. "Human-robot collaborative high-level control with application to rescue robotics". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2016-June. IEEE, May 2016, pp. 2796–2802. DOI: 10.1109/ICRA.2016.7487442.

[41] David C. Conner and Justin Willis. "Flexible Navigation: Finite state machine-based integrated navigation and control for ROS enabled robots". In: *SoutheastCon*. IEEE, Mar. 2017, pp. 1–8. DOI: 10.1109/SECON.2017.7925266.

[42] Sachin Chitta et al. "ros_control: A generic and simple control framework for ROS". In: *The Journal of Open Source Software* 2.20 (2017), p. 456. DOI: 10.21105/joss.00456.

[43] Brett W. Aldrich and Pablo I. Blasco. *SMACC – State Machine Asynchronous C++*. 2020. URL: https://smacc.dev/ (visited on 05/26/2021).

[44] Chao Cao, Hongbiao Zhu, Fan Yang, Yukun Xia, Howie Choset, Jean Oh, and Ji Zhang. "Autonomous Exploration Development Environment and the Planning Algorithms". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, May 2022, pp. 8921–8928. DOI: 10.1109/ICRA46639.2022.9812330.

[45] J Andrew Bagnell et al. "An integrated system for autonomous robotics manipulation". In: *IEEE International Conference on Intelligent Robots and Systems*. 2012, pp. 2955–2962. DOI: 10.1109/IROS.2012.6385888.

[46] Michele Colledanchise and Petter Ögren. "How Behavior Trees modularize robustness and safety in hybrid systems". In: *IEEE International Conference on Intelligent Robots and Systems*. 2014, pp. 1482–1488. DOI: 10.1109/IROS.2014.6942752.

[47] Michele Colledanchise and Petter Ogren. "How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees". In: *IEEE Transactions on Robotics* 33 (2017), pp. 372–389. DOI: 10.1109/TRO.2016.2633567.

[48] Lydia E. Kavraki, Petr Švestka, Jean Claude Latombe, and Mark H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation*. Vol. 12. 4. 1996, pp. 566–580. DOI: 10.1109/70.508439.

[49] Lydia E. Kavraki, Mihail N. Kolountzakis, and Jean Claude Latombe. "Analysis of probabilistic roadmaps for path planning". In: *IEEE Transactions on Robotics and Automation* 14.1 (Feb. 1998), pp. 166–171. DOI: 10.1109/70.660866.

[50] Steven M. LaValle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. 1998.

[51] James J Kuffner and Steven M. La Valle. "RRT-connect: an efficient approach to single-query path planning". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2. 2000, pp. 995–1001. DOI: 10.1109/robot.2000.844730.

[52] S. M. LaValle and J. J. Kuffner. "Randomized kinodynamic planning". In: *International Journal of Robotics Research* 20.5 (2001), pp. 378–400. DOI: 10.1177/02783640122067453.

[53] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. "Anytime motion planning using the RRT". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2011), pp. 1478–1483. DOI: 10.1109/ICRA.2011.5980479.

[54] Oliver Brock and Lydia E. Kavraki. "Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces". In: 2 (2001), pp. 1469–1474. DOI: 10.1109/ROBOT.2001.932817.

[55] Markus Rickert, Oliver Brock, and Alois Knoll. "Balancing exploration and exploitation in motion planning". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2008), pp. 2812–2817. DOI: 10.1109/ROBOT.2008.4543636.

[56] Alexander Shkolnik and Russ Tedrake. "Sample-Based Planning with Volumes in Configuration Space". In: 1109.3145 (2011). DOI: 10.48550/ARXIV.1109.3145.

[57] Fei Gao and Shaojie Shen. "Online quadrotor trajectory generation and autonomous navigation on point clouds". In: *SSRR 2016 - International Symposium on Safety, Security and Rescue Robotics*. 2016, pp. 139–146. DOI: 10.1109/SSRR.2016.7784290.

[58] Fei Gao, William Wu, Wenliang Gao, and Shaojie Shen. "Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments". In: *Journal of Field Robotics* 36.4 (June 2019), pp. 710–733. DOI: 10.1002/rob.21842.

[59] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. *Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic*. Iros. 2014, pp. 2997–3004. DOI: 10.1109/IROS.2014.6942976.

[60] Xinda Wang, Xiao Luo, Baoling Han, Yuhan Chen, Guanhao Liang, and Kailin Zheng. "Collision-free path planning method for robots based on an improved rapidly-exploring random tree algorithm". In: *Applied Sciences (Switzerland)* 10.4 (2020). DOI: 10.3390/app10041381.

[61] Jory Denny, Read Sandström, Andrew Bregger, and Nancy M. Amato. "Dynamic Region-biased Rapidly-exploring Random Trees". In: *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*. Springer International Publishing, Cham, 2020, pp. 640–655. DOI: 10.1007/978-3-030-43089-4_41.

[62] Philipp Krüsi, Paul Furgale, Michael Bosse, and Roland Siegwart. "Driving on Point Clouds: Motion Planning, Trajectory Optimization, and Terrain Assessment in Generic Nonplanar Environments". In: *Journal of Field Robotics* 34.5 (2017), pp. 940–984. DOI: 10.1002/rob.21700.

[63] B Yamauchi. "A Frontier Based Approach for Autonomous Exploration". In: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA* (1997), pp. 146–151. DOI: 10.1109/CIRA.1997.613851.

[64] Phillip Quin, Alen Alempijevic, Gavin Paul, and Dikai Liu. "Expanding wavefront frontier detection: An approach for efficiently detecting frontier cells". In: *Australasian Conference on Robotics and Automation, ACRA*. Vol. 02-04-Dece. 2014.

[65] Phillip Quin, Dac Dang Khoa Nguyen, Thanh Long Vu, Alen Alempijevic, and Gavin Paul. "Approaches for Efficiently Detecting Frontier Cells in Robotics Exploration". In: *Frontiers in Robotics and AI* 8 (Feb. 2021), p. 1. DOI: 10.3389/frobt.2021.616470.

[66] Dirk Holz, Nicola Basilico, Francesco Amigoni, and Sven Behnke. "Evaluating the efficiency of frontier-based exploration strategies". In: *Joint 41st International Symposium on Robotics and 6th German Conference on Robotics 2010, ISR/ROBOTIK 2010* 1.June (2010), pp. 36–43.

[67] Miguel Juliá, Arturo Gil, and Oscar Reinoso. "A comparison of path planning strategies for autonomous exploration and mapping of unknown environments". In: *Autonomous Robots* 33.4 (2012), pp. 427–444. DOI: 10.1007/s10514-012-9298-8.

[68] Kai M Wurm, Cyrill Stachniss, and Wolfram Burgard. "Coordinated multi-robot exploration using a segmentation of the environment". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2008, pp. 1160–1165. DOI: 10.1109/IROS.2008.4650734.

[69] Stefan Obwald, Maren Bennewitz, Wolfram Burgard, and Cyrill Stachniss. "Speeding-Up Robot Exploration by Exploiting Background Information". In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 716–723. DOI: 10.1109/LRA.2016.2520560.

[70] Clara Gomez, Alejandra C Hernandez, and Ramon Barber. "Topological frontier-based exploration and map-building using semantic information". In: *Sensors (Switzerland)* 19.20 (2019). DOI: 10.3390/s19204595.

[71] Agusti Solanas and Miguel Angel Garcia. "Coordinated multi-robot exploration through unsupervised clustering of unknown space". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 1. 2004, pp. 717–721. DOI: 10.1109/iros.2004.1389437.

[72] Miroslav Kulich, Jan Faigl, and Libor Preucil. "On distance utility in the exploration task". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2011, pp. 4455–4460. DOI: 10.1109/ICRA.2011.5980221.

[73] Jan Faigl and Miroslav Kulich. "On determination of goal candidates in frontier-based multi-robot exploration". In: *2013 European Conference on Mobile Robots, ECMR 2013 - Conference Proceedings*. 2013, pp. 210–215. DOI: 10.1109/ECMR.2013.6698844.

[74] Miroslav Kulich, Jiří Kubalík, and Libor Přeučil. "An integrated approach to goal selection in mobile robot exploration". In: *Sensors (Switzerland)* 19.6 (2019). DOI: 10.3390/s19061400.

[75] Francesco Amigoni, Alberto Quattrini Li, and Dirk Holz. "Evaluating the impact of perception and decision timing on autonomous robotic exploration". In: *2013 European Conference on Mobile Robots, ECMR 2013 - Conference Proceedings*. IEEE, Sept. 2013, pp. 68–73. DOI: 10.1109/ECMR.2013.6698822.

[76] Dominik Joho, Cyrill Stachniss, Patrick Pfaff, and Wolfram Burgard. "Autonomous exploration for 3D map learning". In: *Informatik aktuell* (2007), pp. 22–28. DOI: 10.1007/978-3-540-74764-2_4.

[77] Jonathan Butzkey, Andrew Dornbushy, and Maxim Likhachevy. "3-D exploration with an air-ground robotic system". In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 2015-Decem. IEEE, Sept. 2015, pp. 3241–3248. DOI: 10.1109/IROS.2015.7353827.

[78] Cheng Zhu, Rong Ding, Mengxiang Lin, and Yuanyuan Wu. "A 3D frontier-based exploration tool for MAVs". In: *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*. Vol. 2016-Janua. IEEE, Nov. 2016, pp. 348–352. DOI: 10.1109/ICTAI.2015.60.

[79] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. "Information-Theoretic Planning with Trajectory Optimization for Dense 3D Mapping". In: *Robotics: Science and Systems XI* (2015). DOI: 10.15607/RSS.2015.XI.003.

[80] P. G.C.N. Senarathne and Danwei Wang. "Towards autonomous 3D exploration using surface frontiers". In: *SSRR 2016 - International Symposium on Safety, Security and Rescue Robotics*. IEEE, Oct. 2016, pp. 34–41. DOI: 10.1109/SSRR.2016.7784274.

[81] Margarida Faria, António Sérgio Ferreira, Héctor Pérez-Leon, Ivan Maza, and Antidio Viguria. "Autonomous 3D exploration of large structures using an UAV equipped with a 2D LIDAR". In: *Sensors (Switzerland)* 19.22 (2019). DOI: 10.3390/s19224849.

[82] Margarida Faria, Ivan Maza, and Antidio Viguria. "Applying Frontier Cells Based Exploration and Lazy Theta* Path Planning over Single Grid-Based World Representation for Autonomous Inspection of Large 3D Structures with an UAS". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 93.1-2 (2019), pp. 113–133. DOI: 10.1007/s10846-018-0798-4.

[83] David G. Vutetakis and Jing Xiao. "An Autonomous Loop-Closure Approach for Simultaneous Exploration and Coverage of Unknown Infrastructure Using MAVs". In: *2019 International Conference on Robotics and Automation (ICRA)*. Vol. 2019-May. IEEE, May 2019, pp. 2988–2994. DOI: 10.1109/ICRA.2019.8794110.

[84] Liang Lu, Carlos Redondo, and Pascual Campoy. "Optimal frontier-based autonomous exploration in unconstructed environment using rgb-d sensor". In: *Sensors (Switzerland)* 20.22 (2020), pp. 1–16. DOI: 10.3390/s20226507.

[85] Emanuele Vespa, Nikolay Nikolov, Marius Grimm, Luigi Nardi, Paul H.J. Kelly, and Stefan Leutenegger. "Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1144–1151. DOI: 10.1109/LRA.2018.2792537.

[86] Anna Dai, Sotiris Papatheodorou, Nils Funk, Dimos Tzoumanikas, and Stefan Leutenegger. "Fast Frontier-based Information-driven Autonomous Exploration with an MAV". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2020, pp. 9570–9576. DOI: 10.1109/ICRA40945.2020.9196707.

[87] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. "Safe Local Exploration for Replanning in Cluttered Unknown Environments for Microaerial Vehicles". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1474–1481. DOI: 10.1109/LRA.2018.2800109.

[88] Titus Cieslewski, Elia Kaufmann, and Davide Scaramuzza. "Rapid exploration with multi-rotors: A frontier selection method for high speed flight". In: *IEEE International Conference on Intelligent Robots and Systems*. 2017, pp. 2135–2142. DOI: 10.1109/IROS.2017.8206030.

[89] Giuseppe Oriolo, Marilena Vendittelli, Luigi Freda, and Giulio Troso. "The SRT method: Randomized strategies for exploration". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2004.5 (2004), pp. 4688–4694. DOI: 10.1109/robot.2004.1302457.

[90] Hassan Umari and Shayok Mukhopadhyay. "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees". In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 2017-Septe. IEEE, Sept. 2017, pp. 1396–1402. DOI: 10.1109/IROS.2017.8202319.

[91] Wenchuan Qiao, Zheng Fang, and Bailu Si. "Sample-Based Frontier Detection for Autonomous Robot Exploration". In: *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, Dec. 2018, pp. 1165–1170. DOI: 10.1109/ROBIO.2018.8665066.

[92] Baofu Fang, Jianfeng Ding, and Zaijun Wang. "Autonomous Robotic Exploration Based on Frontier Point Optimization and Multistep Path Planning". In: *IEEE Access* 7 (2019), pp. 46104–46113. DOI: 10.1109/ACCESS.2019.2909307.

[93] Zeyu Tian, Chen Guo, Yi Liu, and Tianxiao Cui. "Autonomous exploration of RRT robot based on seeded region growing". In: *Chinese Control Conference, CCC*. Vol. 2020-July. IEEE Computer Society, July 2020, pp. 3936–3941. DOI: 10.23919/CCC50068.2020.9188916.

[94] Di Deng, Runlin Duan, Jiahong Liu, Kuangjie Sheng, and Kenji Shimada. "Robotic exploration of unknown 2d environment using a frontier-based automatic-differentiable information gain measure". In: *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*. Vol. 2020-July. 2020, pp. 1497–1503. DOI: 10.1109/AIM43001.2020.9158881.

[95] Chaoqun Wang, Delong Zhu, Teng Li, Max Q.H. Meng, and Clarence W. De Silva. "Efficient Autonomous Robotic Exploration with Semantic Road Map in Indoor Environments". In: *IEEE Robotics and Automation Letters* 4.3 (July 2019), pp. 2989–2996. DOI: 10.1109/LRA.2019.2923368.

[96] Chaoqun Wang, Wenzheng Chi, Yuxiang Sun, and Max Q.H. Meng. "Autonomous Robotic Exploration by Incremental Road Map Construction". In: *IEEE Transactions on Automation Science and Engineering* 16.4 (Oct. 2019), pp. 1720–1731. DOI: 10.1109/TASE.2019.2894748.

[97] Christos Papachristos, Shehryar Khattak, and Kostas Alexis. "Uncertainty-aware receding horizon exploration and mapping using aerial robots". In: *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, May 2017, pp. 4568–4575. DOI: 10.1109/ICRA.2017.7989531.

[98] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. "Receding horizon path planning for 3D exploration and surface inspection". In: *Autonomous Robots* 42.2 (2018), pp. 291–306. DOI: 10.1007/s10514-016-9610-0.

[99] Christos Papachristos, Mina Kamel, Marija Popović, Shehryar Khattak, Andreas Bircher, Helen Oleynikova, Tung Dang, Frank Mascarich, Kostas Alexis, and Roland Siegwart. "Autonomous Exploration and Inspection Path Planning for Aerial Robots Using the Robot Operating System". In: *Studies in Computational Intelligence*. Vol. 778. Springer Verlag, 2019, pp. 67–111. DOI: 10.1007/978-3-319-91590-6_3.

[100] Bing Jui Ho, Paloma Sodhi, Pedro Teixeira, Ming Hsiao, Tushar Kusnur, and Michael Kaess. "Virtual Occupancy Grid Map for Submap-based Pose Graph SLAM and Planning in 3D Environments". In: *IEEE International Conference on Intelligent Robots and Systems*. 2018, pp. 2175–2182. DOI: 10.1109/IROS.2018.8594234.

[101] Tung Dang, Christos Papachristos, and Kostas Alexis. "Autonomous exploration and simultaneous object search using aerial robots". In: *IEEE Aerospace Conference Proceedings*. Vol. 2018-March. IEEE, Mar. 2018, pp. 1–7. DOI: 10.1109/AERO.2018.8396632.

[102] Christian Witting, Marius Fehr, Rik Bähnemann, Helen Oleynikova, and Roland Siegwart. "History-Aware Autonomous Exploration in Confined Environments Using MAVs". In: *IEEE International Conference on Intelligent Robots and Systems*. 2018, pp. 5208–5215. DOI: 10.1109/IROS.2018.8594502.

[103] Lukas Schmid, Michael Pantic, Raghav Khanna, Lionel Ott, Roland Siegwart, and Juan Nieto. "An Efficient Sampling-Based Method for Online Informative Path Planning in Unknown Environments". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1500–1507. DOI: 10.1109/LRA.2020.2969191.

[104] Menaka Naazare, Francisco Garcia Rosas, and Dirk Schulz. "Online Next-Best-View Planner for 3D-Exploration and Inspection with a Mobile Manipulator Robot". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 3779–3786. DOI: 10.1109/LRA.2022.3146558.

[105] Mihir Dharmadhikari, Tung Dang, Lukas Solanka, Johannes Loje, Huan Nguyen, Nikhil Khedekar, and Kostas Alexis. "Motion Primitives-based Path Planning for Fast and Agile Exploration using Aerial Robots". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2020, pp. 179–185. DOI: 10.1109/ICRA40945.2020.9196964.

[106] Eungchang Mason Lee, Junho Choi, Hyungtae Lim, and Hyun Myung. "REAL: Rapid Exploration with Active Loop-Closing toward Large-Scale 3D Mapping using UAVs". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sept. 2021, pp. 4194–4198. DOI: 10.1109/IROS51168.2021.9636611.

[107] Victor Reijgwart, Alexander Millane, Helen Oleynikova, Roland Siegwart, Cesar Cadena, and Juan Nieto. "Voxgraph: Globally Consistent, Volumetric Mapping Using Signed Distance Function Submaps". In: *IEEE Robotics and Automation Letters* 5.1 (2020), pp. 227–234. DOI: 10.1109/LRA.2019.2953859.

[108] Mihir Dharmadhikari, Harshal Deshpande, Tung Dang, and Kostas Alexis. "Hypergame-based Adaptive Behavior Path Planning for Combined Exploration and Visual Search". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2021-May. IEEE, May 2021, pp. 269–275. DOI: 10.1109/ICRA48506.2021.9561451.

[109] Soohwan Song and Sungho Jo. "Surface-Based Exploration for Autonomous 3D Modeling". In: *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, May 2018, pp. 4319–4326. DOI: 10.1109/ICRA.2018.8460862.

[110] Soohwan Song, Daekyum Kim, and Sungho Jo. "Online coverage and inspection planning for 3D modeling". In: *Autonomous Robots* 44.8 (2020), pp. 1431–1450. DOI: 10.1007/s10514-020-09936-7.

[111] Chao Cao, Hongbiao Zhu, Howie Choset, and Ji Zhang. "Exploring Large and Complex Environments Fast and Efficiently". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2021-May. 2021, pp. 7781–7787. DOI: 10.1109/ICRA48506.2021.9561916.

[112] Zehui Meng, Hailong Qin, Ziyue Chen, Xudong Chen, Hao Sun, Feng Lin, and Marcelo H. Ang. "A two-stage optimized next-view planning framework for 3-D unknown environment exploration, and structural reconstruction". In: *IEEE Robotics and Automation Letters* 2.3 (July 2017), pp. 1680–1687. DOI: 10.1109/LRA.2017.2655144.

[113] Hailong Qin, Zehui Meng, Wei Meng, Xudong Chen, Hao Sun, Feng Lin, and Marcelo H. Ang. "Autonomous Exploration and Mapping System Using Heterogeneous UAVs and UGVs in GPS-Denied Environments". In: *IEEE Transactions on Vehicular Technology* 68.2 (Feb. 2019), pp. 1339–1350. DOI: 10.1109/TVT.2018.2890416.

[114] Noé Pérez-Higueras, Alberto Jardón, Ángel Rodríguez, and Carlos Balaguer. "3D exploration and navigation with optimal-RRT planners for ground robots in indoor incidents". In: *Sensors (Switzerland)* 20.1 (2020). DOI: 10.3390/s20010220.

[115] Bjorn Lindqvist, Ali Akbar Agha-Mohammadi, and George Nikolakopoulos. "Exploration-RRT: A multi-objective Path Planning and Exploration Framework for Unknown and Unstructured Environments". In: *IEEE International Conference on Intelligent Robots and Systems* (2021), pp. 3429–3435. DOI: 10.1109/IROS51168.2021.9636243.

[116] Robbie Shade and Paul Newman. "Choosing where to go: Complete 3D exploration with Stereo". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2011), pp. 2806–2811. DOI: 10.1109/ICRA.2011.5980121.

[117] Brian J Julian, Sertac Karaman, and Daniela Rus. "On mutual information-based control of range sensing robots for mapping applications". In: *intelligent Robots and Systems* (2013), pp. 5156–5163. DOI: 10.1109/IROS.2013.6697102.

[118] Brian J. Julian, Sertac Karaman, and Daniela Rus. "On mutual information-based control of range sensing robots for mapping applications". In: *International Journal of Robotics Research* 33.10 (Sept. 2014), pp. 1375–1392. DOI: 10.1177/0278364914526288.

[119] Renan Maffei, Vitor A.M. Jorge, Edson Prestes, and Mariana Kolberg. "Integrated exploration using time-based potential rails". In: *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, May 2014, pp. 3694–3699. DOI: 10.1109/ICRA.2014.6907394.

[120] Joan Vallvé and Juan Andrade-Cetto. "Potential information fields for mobile robot exploration". In: *Robotics and Autonomous Systems* 69.1 (July 2015), pp. 68–79. DOI: 10.1016/j.robot.2014.08.009.

[121] Chaoqun Wang, Lili Meng, Teng Li, Clarence W. De Silva, and Max Q.H. Meng. "Towards autonomous exploration with information potential field in 3D environments". In: *2017 18th International Conference on Advanced Robotics, ICAR 2017*. IEEE, July 2017, pp. 340–345. DOI: 10.1109/ICAR.2017.8023630.

[122] Evan Kaufman, Kuya Takami, Zhuming Ai, and Taeyoung Lee. "Autonomous quadrotor 3D mapping and exploration using exact occupancy probabilities". In: *Proceedings - 2nd IEEE International Conference on Robotic Computing, IRC 2018*. Vol. 2018-Janua. Institute of Electrical and Electronics Engineers Inc., Apr. 2018, pp. 49–55. DOI: 10.1109/IRC.2018.00016.

[123] Jhielson M. Pimentel, Mário S. Alvim, Mario F.M. Campos, and Douglas G. Macharet. "Information-Driven Rapidly-Exploring Random Tree for Efficient Environment Exploration". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 91.2 (2018), pp. 313–331. DOI: 10.1007/s10846-017-0709-0.

[124] Henry Carrillo, Philip Dames, Vijay Kumar, and José A. Castellanos. "Autonomous robotic exploration using a utility function based on Rényi's general theory of entropy". In: *Autonomous Robots* 42.2 (Feb. 2018), pp. 235–256. DOI: 10.1007/s10514-017-9662-9.

[125] Shaojie Shen, Nathan Michael, and Vijay Kumar. "Stochastic differential equation-based exploration algorithm for autonomous indoor 3D exploration with a micro-aerial vehicle". In: *International Journal of Robotics Research* 31.12 (Oct. 2012), pp. 1431–1444. DOI: 10.1177/0278364912461676.

[126] Shaojie Shen, Nathan Michael, and Vijay Kumar. "Autonomous indoor 3D exploration with a micro-aerial vehicle". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2012), pp. 9–15. DOI: 10.1109/ICRA.2012.6225146.

[127] Shi Bai, Fanfei Chen, and Brendan Englot. "Toward autonomous mapping and exploration for mobile robots through deep supervised learning". In: *IEEE International Conference on Intelligent Robots and Systems* 2017-Septe (2017), pp. 2379–2384. DOI: 10.1109/IROS.2017.8206050.

[128] Amir Ramezani Dooraki and Deok Jin Lee. "An end-to-end deep reinforcement learning-based intelligent agent capable of autonomous exploration in unknown environments". In: *Sensors (Switzerland)* 18.10 (2018). DOI: 10.3390/s18103575.

[129] Tao Chen, Saurabh Gupta, and Abhinav Gupta. "Learning exploration policies for navigation". In: *7th International Conference on Learning Representations, ICLR 2019*. 2019. eprint: 1903.01959. URL: https://sites.google.com/view/exploration-for-nav.

[130] Farzad Niroui, Kaicheng Zhang, Zendai Kashino, and Goldie Nejat. "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 610–617. DOI: 10.1109/LRA.2019.2891991.

[131] Rakesh Shrestha, Fei Peng Tian, Wei Feng, Ping Tan, and Richard Vaughan. "Learned map prediction for enhanced mobile robot exploration". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 1197–1204. DOI: 10.1109/ICRA.2019.8793769.

[132] Louis Ly and Yen Hsi Richard Tsai. "Autonomous exploration, reconstruction, and surveillance of 3d environments aided by deep learning". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2019-May.September 2018 (2019), pp. 5467–5473. DOI: 10.1109/ICRA.2019.8794426.

[133] Russell Reinhart, Tung Dang, Emily Hand, Christos Papachristos, and Kostas Alexis. "Learning-based Path Planning for Autonomous Exploration of Subterranean Environments". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2020, pp. 1215–1221. DOI: 10.1109/ICRA40945.2020.9196662.

[134] C. Connolly. "The determination of next best views". In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. Institute of Electrical and Electronics Engineers, 1985, pp. 432–435. DOI: 10.1109/ROBOT.1985.1087372.

[135] Marcus Strand and Rüdiger Dillmann. "Using an attributed 2D-grid for next-best-view planning on 3D environment data for an autonomous robot". In: *Proceedings of the 2008 IEEE International Conference on Information and Automation, ICIA 2008* July (2008), pp. 314–319. DOI: 10.1109/ICINFA.2008.4608017.

[136] Christian Dornhege and Alexander Kleiner. "A frontier-void-based approach for autonomous exploration in 3d". In: *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011* 27.6 (Apr. 2011), pp. 351–356. DOI: 10.1109/SSRR.2011.6106778.

[137] J. Irving Vasquez-Gomez, L. Enrique Sucar, and Rafael Murrieta-Cid. "Hierarchical ray tracing for fast volumetric next-best-view planning". In: *Proceedings - 2013 International Conference on Computer and Robot Vision, CRV 2013* (2013), pp. 181–187. DOI: 10.1109/CRV.2013.42.

[138] J. Irving Vasquez-Gomez, L. Enrique Sucar, Rafael Murrieta-Cid, and Efrain Lopez-Damian. "Volumetric next-best-view planning for 3D object reconstruction with positioning error". In: *International Journal of Advanced Robotic Systems* 11 (2014). DOI: 10.5772/58759.

[139] J. Irving Vasquez-Gomez, L. Enrique Sucar, Rafael Murrieta-Cid, and Juan Carlos Herrera-Lozada. "Tree-based search of the next best view/state for three-dimensional object reconstruction". In: *International Journal of Advanced Robotic Systems* 15.1 (2018), pp. 1–11. DOI: 10.1177/1729881418754575.

[140] Ivan Maurović, Marija Dakulović, and Ivan Petrović. *Autonomous exploration of large unknown indoor environments for dense 3D model building*. Vol. 19. 3. IFAC, 2014, pp. 10188–10193. DOI: 10.3182/20140824-6-ZA-1003.01275.

[141] Christian Potthast and Gaurav S. Sukhatme. "A probabilistic framework for next best view estimation in a cluttered environment". In: *Journal of Visual Communication and Image Representation* 25.1 (2014), pp. 148–164. DOI: 10.1016/j.jvcir.2013.07.006.

[142] Jonathan Daudelin and Mark Campbell. "An Adaptable, Probabilistic, Next-Best View Algorithm for Reconstruction of Unknown 3-D Objects". In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1540–1547. DOI: 10.1109/LRA.2017.2660769.

[143] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (1959), pp. 269–271. DOI: 10.1007/BF01386390.

[144] Jon Louis Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Communications of the ACM* 18.9 (1975), pp. 509–517. DOI: 10.1145/361002.361007.

[145] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: Part I". In: *IEEE Robotics and Automation Magazine* 13.2 (2006), pp. 99–108. DOI: 10.1109/MRA.2006.1638022.

[146] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 9780262201629.

[147] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. "A tutorial on graph-based SLAM". In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43. DOI: 10.1109/MITS.2010.939925.

[148] 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 - SIGGRAPH '96* (1996), pp. 303–312. DOI: 10.1145/237170.237269.

[149] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning". In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 2017-Septe. IEEE, Sept. 2017, pp. 1366–1373. DOI: 10.1109/IROS.2017.8202315.

[150] Boris Lau, Christoph Sprunk, and Wolfram Burgard. "Improved updating of Euclidean distance maps and Voronoi diagrams". In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*. 2010, pp. 281–286. DOI: 10.1109/IROS.2010.5650794.

[151] Robinson Julia. "On the Hamiltonian Game (A Traveling Salesman Problem)". In: *Project Rand* (1949), pp. 1–10. URL: https://apps.dtic.mil/sti/citations/AD0204961.

[152] Michael Held and Richard M. Karp. "The Traveling-Salesman Problem and Minimum Spanning Trees". In: *Operations Research* 18.6 (Dec. 1970), pp. 1138–1162. DOI: 10.1287/opre.18.6.1138.

[153] Mary E. Kurz. "Heuristics for the Traveling Salesman Problem". In: *Wiley Encyclopedia of Operations Research and Management Science*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Feb. 2011. DOI: 10.1002/9780470400531.eorms0929.

[154] G. A. Croes. "A Method for Solving Traveling-Salesman Problems". In: *Operations Research* 6.6 (Dec. 1958), pp. 791–812. DOI: 10.1287/opre.6.6.791.

[155] S. Lin and B. W. Kernighan. "Effective Heuristic Algorithm for the Traveling-Salesman Problem." In: *Operations Research* 21.2 (Apr. 1973), pp. 498–516. DOI: 10.1287/opre.21.2.498.

[156] Charles E. Noon and James C. Bean. "An Efficient Transformation Of The Generalized Traveling Salesman Problem". In: *INFOR: Information Systems and Operational Research* 31.1 (Feb. 1993), pp. 39–44. DOI: 10.1080/03155986.1993.11732212.

[157] Jose Luis Blanco and Pranjal Kumar Rai. *nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees*. 2014. URL: https://github.com/jlblancoc/nanoflann.

[158] Anton Koval, Christoforos Kanellakis, Emil Vidmark, Jakub Haluska, and George Nikolakopoulos. "A subterranean virtual cave world for gazebo based on the DARPA SubT challenge". In: (2020), pp. 5–6. DOI: 10.48550/ARXIV.2004.08452.

[159] Chao Cao, Hongbiao Zhu, Choset, Howie, and Ji Zhang. *TARE Planner*. URL: https://www.cmu-exploration.com/tare-planner (visited on 10/08/2022).

[160] *Altstadtfreunde Lauf an der Pegnitz*. URL: https://altstadtfreunde-lauf.de/de/2-sehenswuerdigkeiten/2c-felsenkeller (visited on 10/03/2022).

A. Appendix

Sparse Raycasting and Sparse Ray Polling Comparison

Tab. A.1.: This table shows the mean duration t, the perceived voxels v, the maximum perceived voxels v_{max} and the view score vs for each configuration of the comparison between SRP and SRC and the different step sizes $\Delta = (\Delta_r, \Delta_\vartheta, \Delta_\varphi)$. These values are depicted for the Husky robot with a depth camera as RS, the Husky robot with a lidar as VL and the Turtlebot3 robot with a depth camera as T3.

Sensor setup	Δ_φ [deg]	Δ_ϑ [deg]	Δ_r [deg]	Variant	t [ns]	v	v_{max}	vs
RS	5	5	2	SRP	142448275.9	68373.9	193860.0	0.35270
RS	5	5	2	SRC	962175697.9	92704.1	282185.0	0.32852
RS	5	5	5	SRP	64944852.9	16066.6	77220.0	0.20806
RS	5	5	5	SRC	861540441.2	22827.3	114632.0	0.19914
RS	5	5	10	SRP	44528037.4	8262.2	38340.0	0.21550
RS	5	5	10	SRC	837630841.1	11666.6	55885.0	0.20876
RS	5	5	15	SRP	38441295.5	5089.3	25380.0	0.20052
RS	5	5	15	SRC	824170040.5	7323.0	37260.0	0.19654
RS	10	10	2	SRP	56283704.6	19850.5	49542.0	0.40068
RS	10	10	2	SRC	260569753.8	27352.8	72322.0	0.37821
RS	10	10	5	SRP	35642857.1	6674.2	19734.0	0.33821
RS	10	10	5	SRC	233755760.4	9522.9	29151.0	0.32668
RS	10	10	10	SRP	31061624.6	3022.2	9798.0	0.30845
RS	10	10	10	SRC	228700280.1	4353.4	14613.0	0.29791
RS	10	10	15	SRP	28831325.3	2211.3	6486.0	0.34093
RS	10	10	15	SRC	225597590.4	3165.4	9535.0	0.33198
RS	15	15	2	SRP	38267233.2	8205.9	21540.0	0.38096
RS	15	15	2	SRC	128436260.6	11085.2	31767.0	0.34895
RS	15	15	5	SRP	29484330.5	2820.4	8580.0	0.32872
RS	15	15	5	SRC	117532763.5	4023.7	12679.0	0.31735
RS	15	15	10	SRP	28602240.9	1487.4	4260.0	0.34915
RS	15	15	10	SRC	116512605.0	2145.4	6300.0	0.34054
RS	15	15	15	SRP	26767883.2	941.6	2820.0	0.33389
RS	15	15	15	SRC	112905109.5	1358.7	4182.0	0.32490
VL	5	5	3	SRP	523696774.2	962459.6	4381574.0	0.21966
VL	5	5	3	SRC	2086571428.6	912591.7	5790461.0	0.15760
VL	5	5	5	SRP	214385756.7	344764.0	2372112.0	0.14534
VL	5	5	5	SRC	1701626112.8	271964.0	3152436.0	0.08627
VL	5	5	10	SRP	70000000.0	68031.5	795853.0	0.08548
VL	5	5	10	SRC	1510267379.7	72309.0	993902.0	0.07275
VL	5	5	15	SRP	113895765.5	148065.5	800714.0	0.18492
VL	5	5	15	SRC	1548107491.9	159408.9	1080118.0	0.14758
VL	10	10	3	SRP	264289682.5	478400.2	1196929.0	0.39969
VL	10	10	3	SRC	671507936.5	541583.3	1578183.0	0.34317

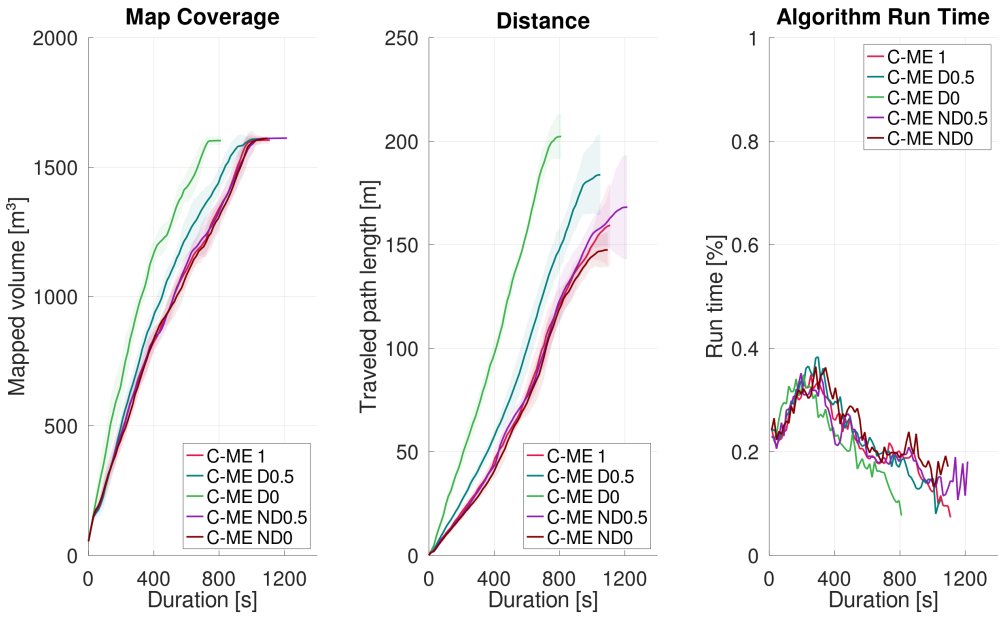
Scanner	Δ_φ [deg]	Δ_ϑ [deg]	Δ_r [m]	Variant	t [ns]	v	v_{max}	vs
VL	10	10	5	SRP	110100182.1	179401.8	648703.0	0.27655
VL	10	10	5	SRC	471730418.9	181010.3	826899.0	0.21890
VL	10	10	10	SRP	45932270.9	43580.0	325005.0	0.13409
VL	10	10	10	SRC	405095617.5	44569.5	416672.0	0.10697
VL	10	10	15	SRP	45383275.3	43538.7	192701.0	0.22594
VL	10	10	15	SRC	395167247.4	45869.6	250549.0	0.18308
VL	15	15	3	SRP	106254417.0	178915.0	572730.0	0.31239
VL	15	15	3	SRC	294551236.7	178834.3	733893.0	0.24368
VL	15	15	5	SRP	51147991.5	65804.6	344680.0	0.19091
VL	15	15	5	SRC	228756871.0	64047.8	446109.0	0.14357
VL	15	15	10	SRP	34750853.2	27448.3	136919.0	0.20047
VL	15	15	10	SRC	200703071.7	30375.7	163543.0	0.18574
VL	15	15	15	SRP	32213864.3	22358.9	92766.0	0.24102
VL	15	15	15	SRC	200535398.2	23431.5	112036.0	0.20914
T3	5	5	2	SRP	96195945.9	18874.1	64620.0	0.29208
T3	5	5	2	SRC	938545045.0	23142.3	97494.0	0.23737
T3	5	5	5	SRP	31452950.6	4057.5	25740.0	0.15763
T3	5	5	5	SRC	847347687.4	5501.1	38902.0	0.14141
T3	5	5	10	SRP	18875939.8	1973.6	12780.0	0.15443
T3	5	5	10	SRC	833399436.1	2687.2	19333.0	0.13900
T3	5	5	15	SRP	15200413.2	1549.9	8460.0	0.18321
T3	5	5	15	SRC	827203512.4	2102.9	12893.0	0.16310
T3	10	10	2	SRP	37296774.2	6740.7	17232.0	0.39117
T3	10	10	2	SRC	250104218.4	8518.4	25758.0	0.33071
T3	10	10	5	SRP	16467377.7	1630.4	6864.0	0.23754
T3	10	10	5	SRC	223143663.7	2239.2	10314.0	0.21711
T3	10	10	10	SRP	11903790.1	627.6	3408.0	0.18415
T3	10	10	10	SRC	216446064.1	873.0	5122.0	0.17043
T3	10	10	15	SRP	11090057.3	559.0	2256.0	0.24781
T3	10	10	15	SRC	214048932.8	760.6	3408.0	0.22317
T3	15	15	2	SRP	18677212.5	2560.6	7180.0	0.35663
T3	15	15	2	SRC	112202650.7	3413.9	10859.0	0.31438
T3	15	15	5	SRP	12243988.7	725.0	2860.0	0.25349
T3	15	15	5	SRC	102924328.1	958.5	4345.0	0.22060
T3	15	15	10	SRP	10625102.5	321.8	1420.0	0.22665
T3	15	15	10	SRC	101314192.0	447.0	2166.0	0.20639
T3	15	15	15	SRP	10308108.1	262.9	940.0	0.27967
T3	15	15	15	SRC	100090090.1	362.0	1438.0	0.25176

Tab. A.2.: The duration λ_t, voxels λ_v and maximum number of voxels factor $\lambda_{v_{max}}$ comparing SRP with SRC as well as the mean μ and SD σ for the yaw φ_{dif} and view score difference vs_{dif} for the respective configuration can be seen in this table. These values are depicted for the Husky robot with a depth camera as RS, the Husky robot with a lidar as VL and the Turtlebot3 robot with a depth camera as T3.

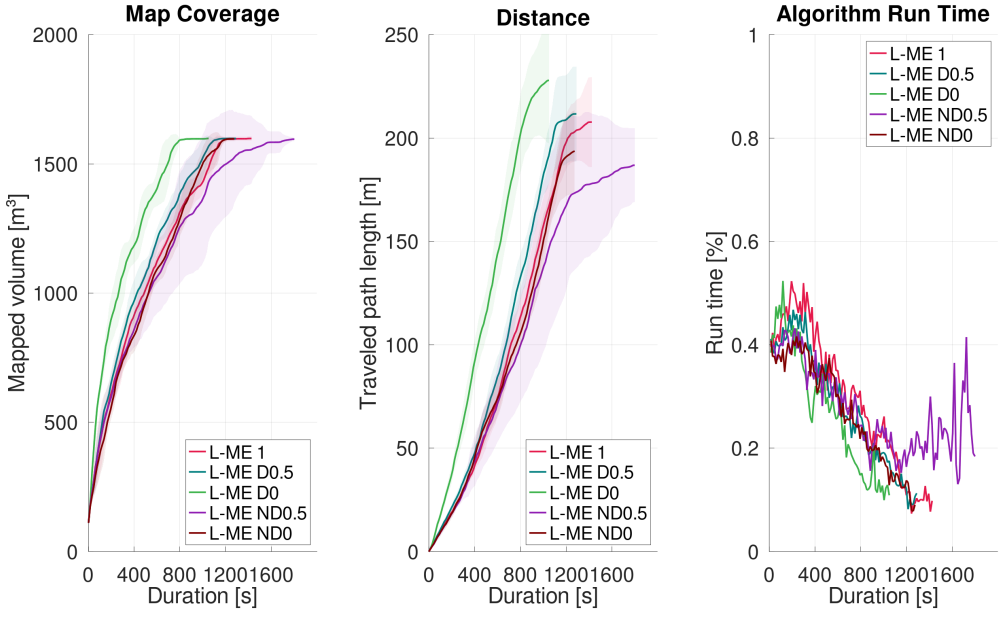
Sensor setup	Δ_φ [deg]	Δ_ϑ [deg]	Δ_r [deg]	λ_t	λ_v	$\lambda_{v_{max}}$	φ_{dif} [deg]		vs_{dif}	
							μ	σ	μ	σ
RS	5	5	2	6.75456	1.35584	1.45561	8.40722	20.04333	-0.01624	0.02649
RS	5	5	5	13.26572	1.42079	1.48449	5.45956	22.34149	-0.00893	0.00883
RS	5	5	10	18.81131	1.41204	1.45762	4.85981	20.86687	-0.00674	0.00652
RS	5	5	15	21.43971	1.43892	1.46809	4.95951	20.74816	-0.00398	0.00809
RS	10	10	2	4.62958	1.37794	1.45981	7.70223	21.76745	-0.00804	0.03088
RS	10	10	5	6.55828	1.42681	1.47720	2.25806	12.92965	-0.01154	0.01063
RS	10	10	10	7.36279	1.44048	1.49143	2.46499	11.22182	-0.01054	0.00912
RS	10	10	15	7.82474	1.43149	1.47009	2.50602	11.29252	-0.01154	0.01063
RS	15	15	2	3.35630	1.35088	1.47479	12.49292	30.48085	-0.01054	0.00912
RS	15	15	5	3.98628	1.42665	1.47774	2.80627	14.33386	-0.00895	0.01310
RS	15	15	10	4.07355	1.44241	1.47887	1.55462	10.19641	-0.00861	0.01099
RS	15	15	15	4.21793	1.44303	1.48298	2.97810	14.78030	-0.00444	0.01504
VL	5	5	3	3.98431	0.94819	1.32155	13.63871	22.75416	-0.06307	0.06840
VL	5	5	5	7.93722	0.78884	1.32896	19.94065	32.73208	-0.05907	0.03747
VL	5	5	10	21.57525	1.06288	1.24885	13.86096	31.39570	-0.01273	0.02127
VL	5	5	15	13.59232	1.07661	1.34894	15.40065	34.20334	-0.03733	0.03808
VL	10	10	3	2.54080	1.13207	1.31853	31.34127	40.99396	-0.05652	0.03915
VL	10	10	5	4.28456	1.00897	1.27470	29.00911	43.69630	-0.05058	0.05681
VL	10	10	10	8.81941	1.02271	1.28205	20.15936	37.80167	-0.02712	0.03119
VL	10	10	15	8.70733	1.05354	1.30020	18.18815	36.99186	-0.04286	0.04900
VL	15	15	3	2.77213	0.99955	1.28139	30.63251	43.93837	-0.06871	0.03850
VL	15	15	5	4.47245	0.97330	1.29427	35.18816	50.68176	-0.04735	0.04377
VL	15	15	10	5.77549	1.10665	1.19445	18.60410	39.57064	-0.01474	0.04589
VL	15	15	15	6.22513	1.04797	1.20773	17.82596	38.11987	-0.01478	0.06124
T3	5	5	2	9.75660	1.22614	1.50873	20.54842	36.98974	-0.01004	0.05982
T3	5	5	5	26.94017	1.35578	1.51134	10.04785	29.24515	-0.00981	0.02483
T3	5	5	10	44.15141	1.36156	1.51275	5.04699	20.62640	0.00131	0.02919
T3	5	5	15	54.41980	1.35677	1.52400	6.36880	23.68990	-0.00258	0.03618
T3	10	10	2	6.70579	1.26372	1.49478	16.01787	34.54139	0.03618	0.06126
T3	10	10	5	13.55065	1.37339	1.50262	4.42158	19.01974	0.00585	0.02961
T3	10	10	10	18.18295	1.39099	1.50293	3.93586	19.00152	0.00233	0.02177
T3	10	10	15	19.30098	1.36046	1.51064	9.04321	28.00236	0.01420	0.03900
T3	15	15	2	6.00746	1.33323	1.51240	16.37366	35.58846	0.02948	0.04977
T3	15	15	5	8.40611	1.32211	1.51923	8.86209	27.10630	0.01185	0.05113
T3	15	15	10	9.53536	1.38901	1.52535	5.37982	19.46499	0.00348	0.03424
T3	15	15	15	9.70984	1.37712	1.52979	7.06486	20.44221	-0.02306	0.04095

Revised Cost Function Factor Comparison

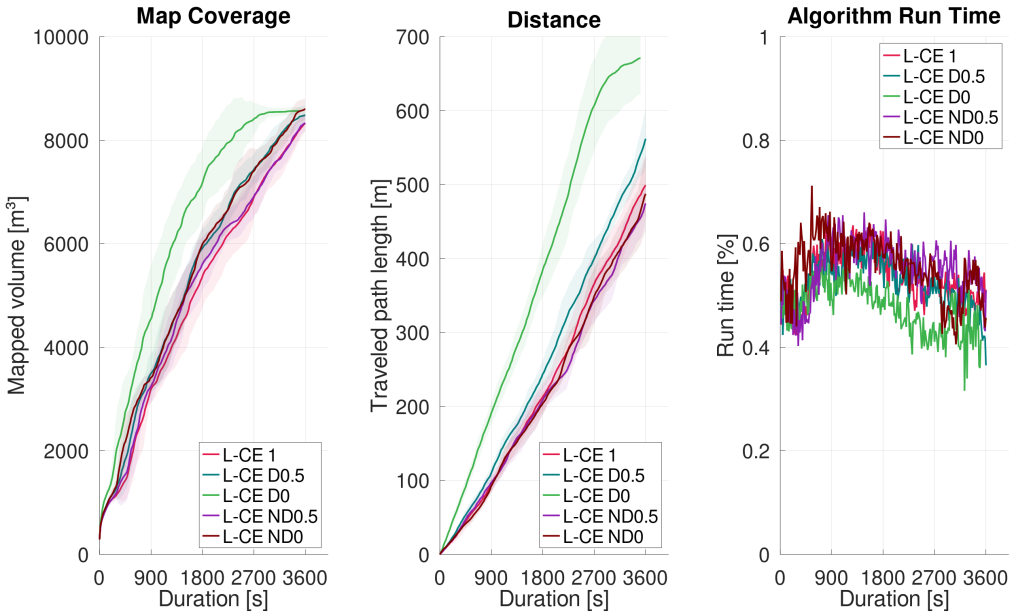
Basic RNE



(a)

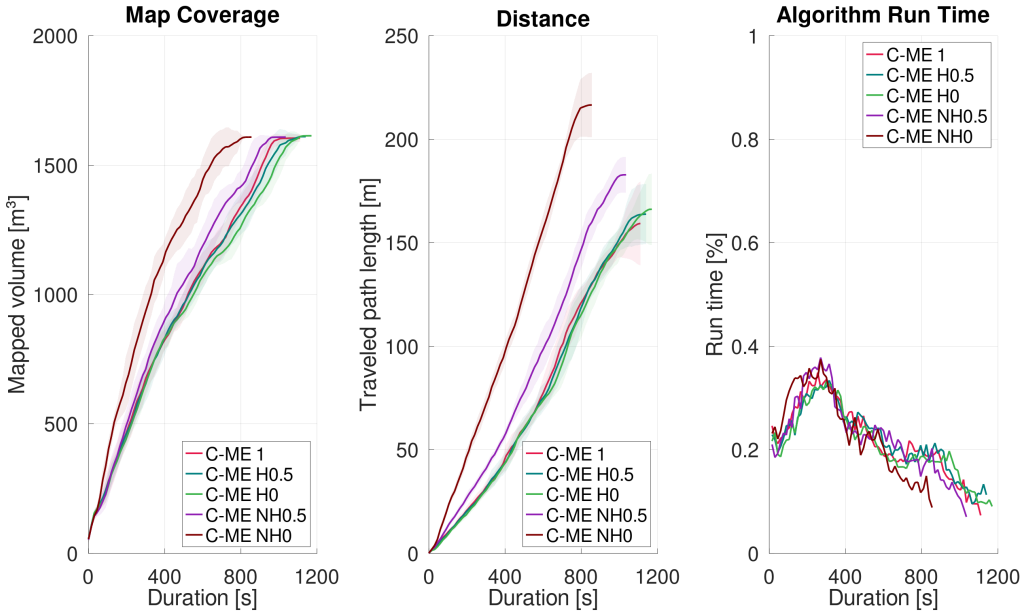


(b)

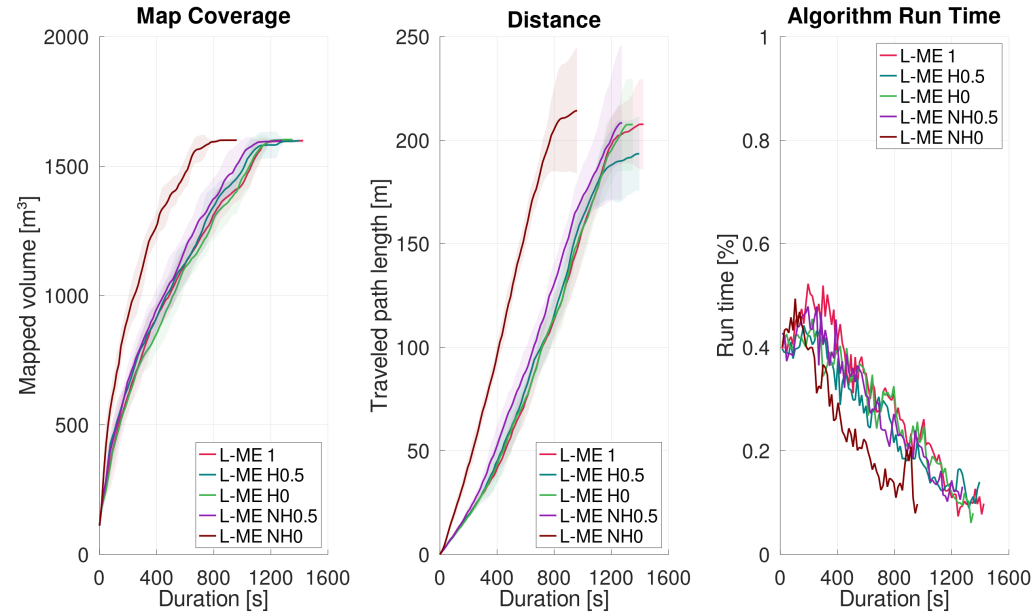


(c)

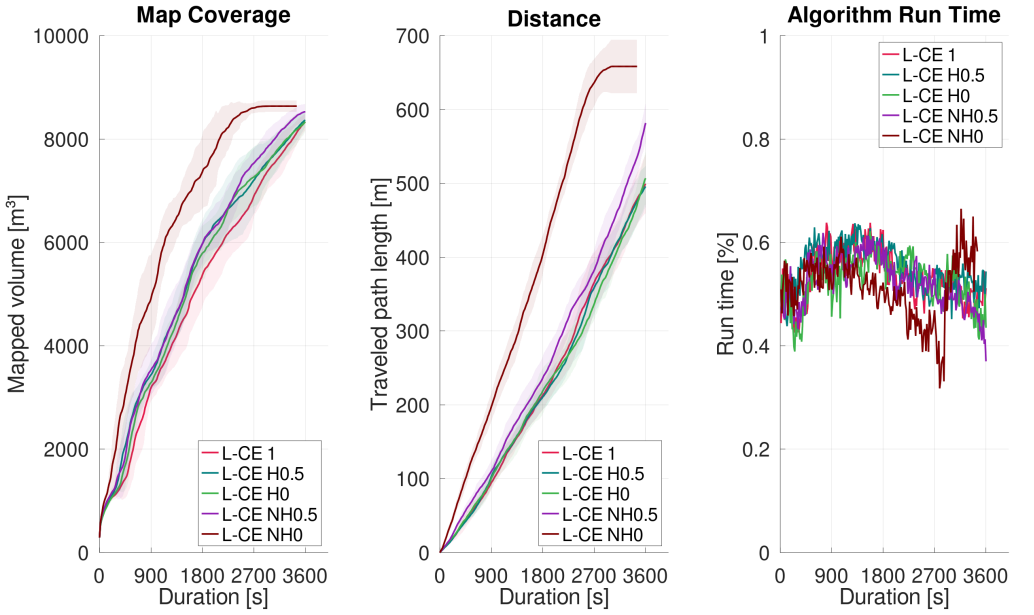
Fig. A.1.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying distance factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.



(a)

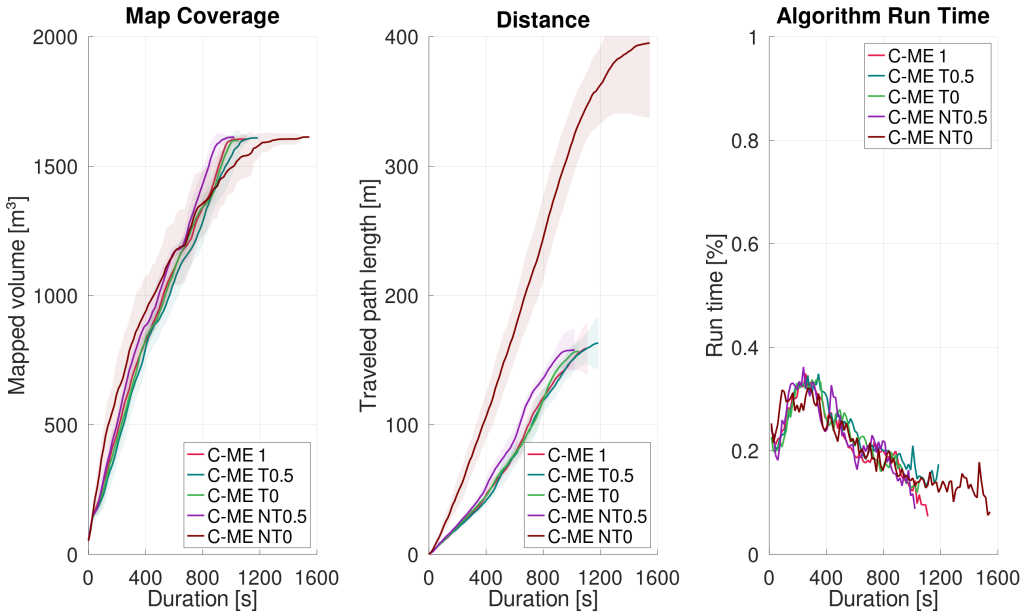


(b)

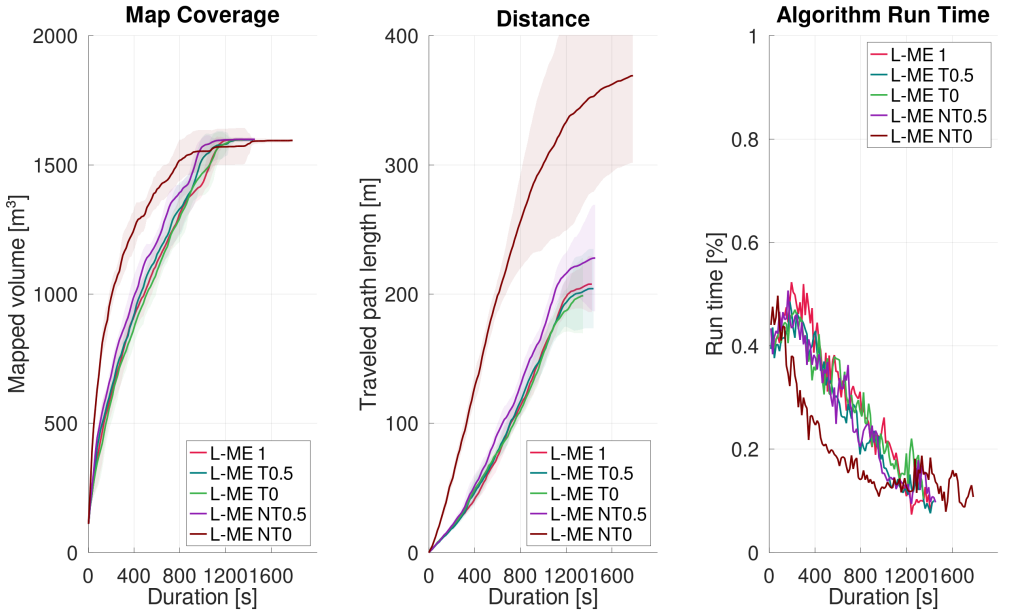


(c)

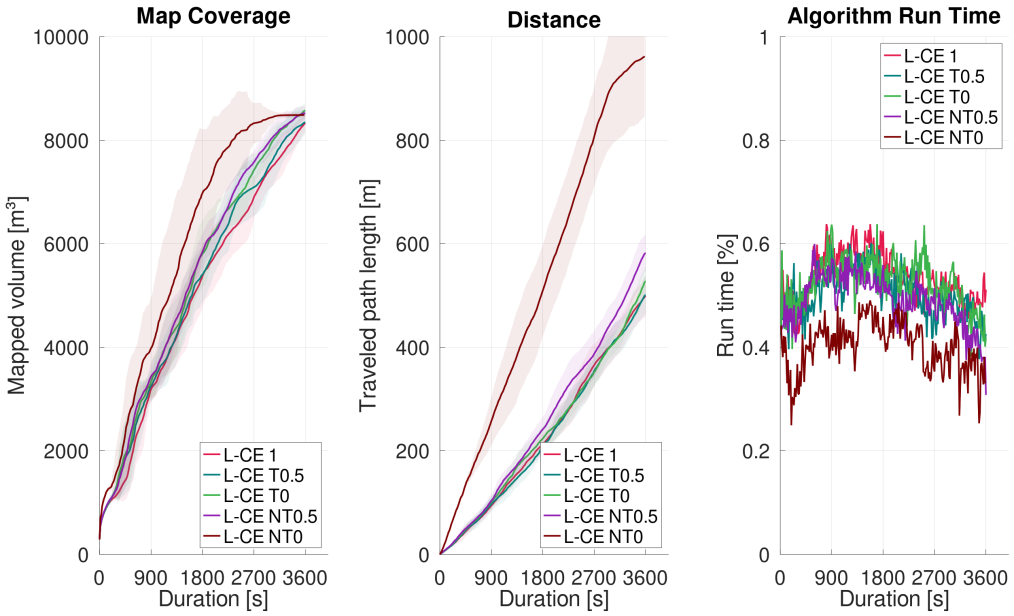
Fig. A.2.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying heading factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.


Map Coverage
Mapped volume [m³]
Duration [s]
C-ME 1
C-ME T0.5
C-ME T0
C-ME NT0.5
C-ME NT0
Distance
Traveled path length [m]
Algorithm Run Time
Run time [%]

(a)


Map Coverage
Mapped volume [m³]
Duration [s]
L-ME 1
L-ME T0.5
L-ME T0
L-ME NT0.5
L-ME NT0
Distance
Traveled path length [m]
Algorithm Run Time
Run time [%]

(b)



(c)

Fig. A.3.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying traversability factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.

Tab. A.3.: The impact of a varying distance factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m³]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	1020.00	57.01	159.36	20.24	1604.2	15.4	0.232	0.014	0	10
C-ME D0.5	967.50	54.87	183.80	19.36	1609.0	3.1	0.246	0.030	0	10
C-ME D0	754.50	39.17	202.28	10.94	1602.4	16.0	0.237	0.018	0	10
C-ME ND0.5	1054.50	77.57	168.13	25.06	1611.8	2.1	0.228	0.024	0	10
C-ME ND0	1014.00	36.88	147.52	8.34	1610.9	2.0	0.250	0.034	0	10
L-ME 1	1236.00	79.44	207.70	21.71	1597.9	8.4	0.327	0.032	0	10
L-ME D0.5	1128.00	70.99	211.65	22.89	1597.8	6.2	0.306	0.022	0	10
L-ME D0	895.50	84.72	227.91	30.40	1598.3	5.8	0.302	0.038	0	10
L-ME ND0.5	1314.00	233.15	186.85	17.80	1595.4	7.1	0.290	0.045	0	10
L-ME ND0	1182.00	59.92	193.65	19.79	1596.7	7.0	0.286	0.015	0	10
L-CE 1	3594.00	7.75	499.02	39.87	8327.6	157.8	0.534	0.032	0	10
L-CE D0.5	3593.33	7.91	561.88	39.68	8482.0	189.1	0.526	0.025	1	10
L-CE D0	2868.00	308.85	670.89	48.90	8564.5	73.7	0.484	0.039	0	10
L-CE ND0.5	3595.00	7.75	474.47	21.36	8328.5	156.4	0.553	0.035	4	10
L-CE ND0	3594.00	8.22	487.48	51.55	8601.1	195.3	0.558	0.029	5	10

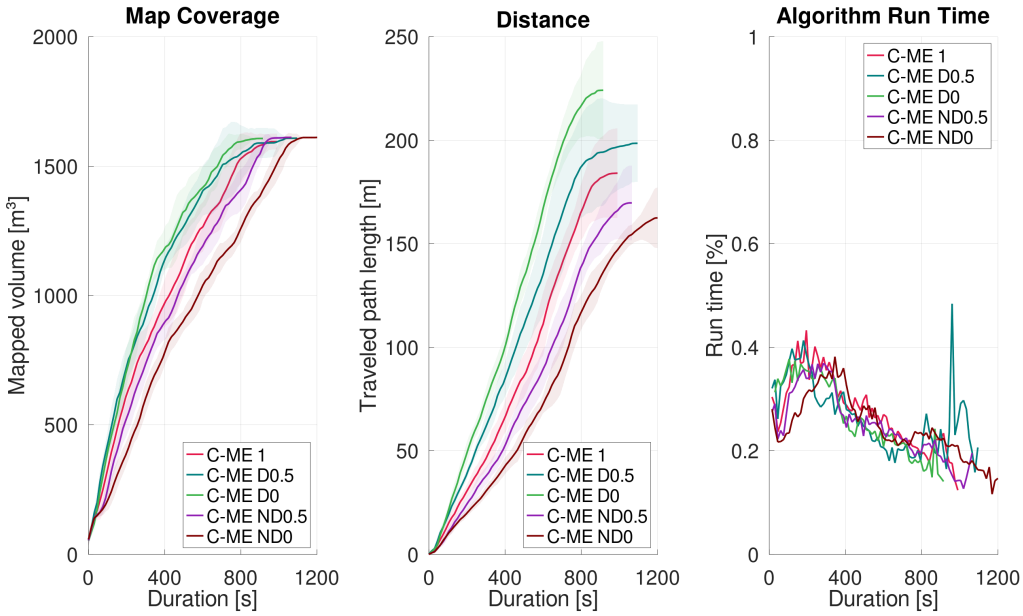
Tab. A.4.: The impact of a varying heading factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	1020.00	57.01	159.36	20.24	1604.2	15.4	0.232	0.014	0	10
C-ME H0.5	1057.50	59.74	163.84	14.49	1610.2	3.8	0.230	0.030	0	10
C-ME H0	1096.50	51.17	166.18	17.20	1613.3	2.9	0.217	0.013	0	10
C-ME NH0.5	973.50	44.29	182.83	8.65	1608.1	3.2	0.235	0.015	0	10
C-ME NH0	795.00	38.73	216.54	15.50	1608.3	2.9	0.247	0.019	0	10
L-ME 1	1236.00	79.44	207.70	21.71	1597.9	8.4	0.327	0.032	0	10
L-ME H0.5	1173.00	91.87	193.47	17.80	1597.5	3.3	0.290	0.017	0	10
L-ME H0	1224.10	75.82	207.56	22.11	1602.4	5.6	0.307	0.036	0	10
L-ME NH0.5	1135.50	125.58	208.37	37.26	1598.0	8.3	0.313	0.043	0	10
L-ME NH0	790.50	93.73	214.28	30.17	1600.0	4.7	0.289	0.030	0	10
L-CE 1	3594.00	7.75	499.02	39.87	8327.6	157.8	0.534	0.032	0	10
L-CE H0.5	3576.00	48.06	495.57	27.88	8361.6	146.2	0.552	0.029	0	10
L-CE H0	3596.25	6.95	506.76	35.38	8331.2	179.2	0.526	0.040	2	10
L-CE NH0.5	3595.00	7.50	581.56	25.73	8526.9	146.6	0.525	0.042	1	10
L-CE NH0	2753.33	316.97	658.23	36.30	8635.3	109.8	0.508	0.024	1	10

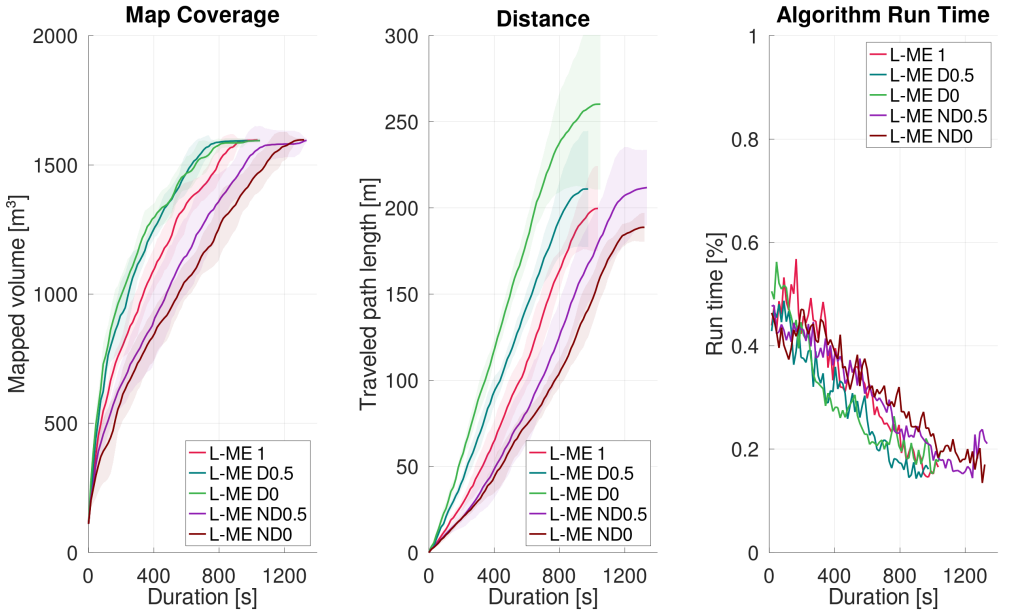
Tab. A.5.: The impact of a varying traversability factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	1020.00	57.01	159.36	20.24	1604.2	15.4	0.232	0.014	0	10
C-ME T0.5	1066.50	70.98	163.36	20.63	1608.9	2.8	0.240	0.022	0	10
C-ME T0	1012.50	25.16	156.88	6.79	1598.6	19.6	0.236	0.010	0	10
C-ME NT0.5	933.00	39.87	157.91	16.11	1611.5	3.8	0.243	0.020	0	10
C-ME NT0	1225.50	189.02	394.99	57.67	1612.3	2.7	0.216	0.019	0	10
L-ME 1	1236.00	79.44	207.70	21.71	1597.9	8.4	0.327	0.032	0	10
L-ME T0.5	1180.50	125.54	204.21	30.69	1596.8	6.8	0.303	0.029	0	10
L-ME T0	1201.50	102.31	198.76	29.15	1596.9	9.2	0.318	0.033	0	10
L-ME NT0.5	1174.50	115.27	227.86	41.25	1598.6	7.2	0.309	0.022	0	10
L-ME NT0	1195.50	296.39	368.98	67.48	1594.4	6.0	0.229	0.033	0	10
L-CE 1	3594.00	7.75	499.02	39.87	8327.6	157.8	0.534	0.032	0	10
L-CE T0.5	3589.29	7.33	500.96	36.30	8345.2	236.1	0.505	0.030	3	10
L-CE T0	3592.50	8.76	527.48	28.40	8572.5	147.1	0.523	0.045	4	10
L-CE NT0.5	3564.00	48.58	582.06	35.19	8522.7	137.7	0.492	0.042	0	10
L-CE NT0	3133.50	295.44	961.35	115.54	8488.3	45.6	0.409	0.036	0	10

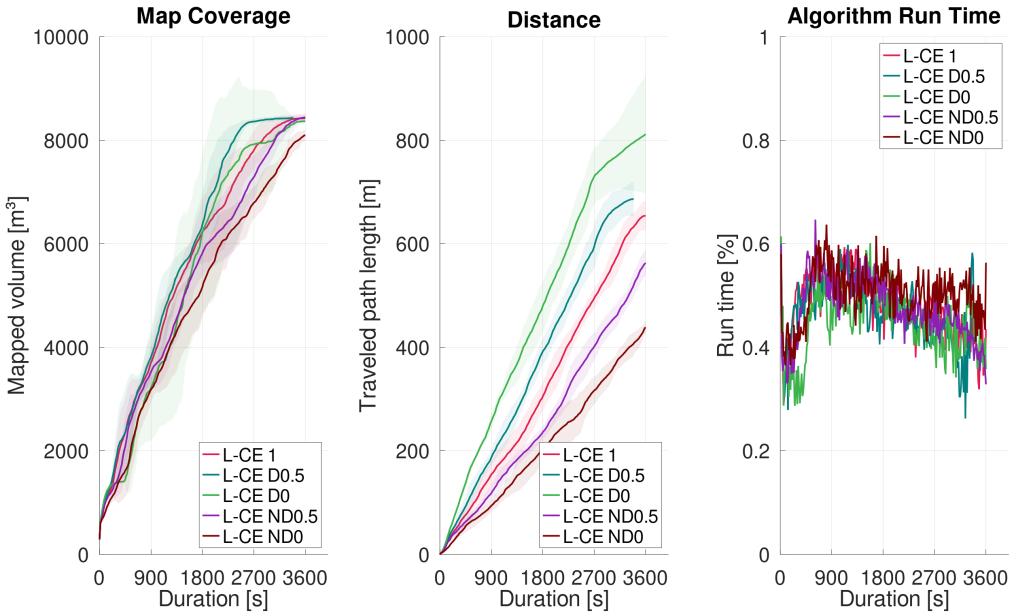
Node Area Inflation RNE



(a)

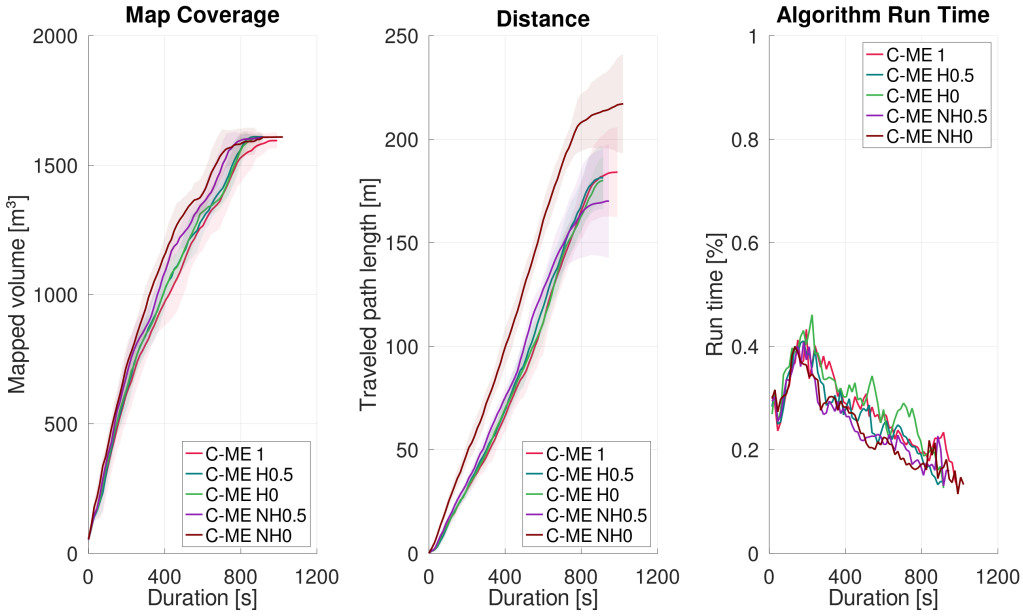


(b)

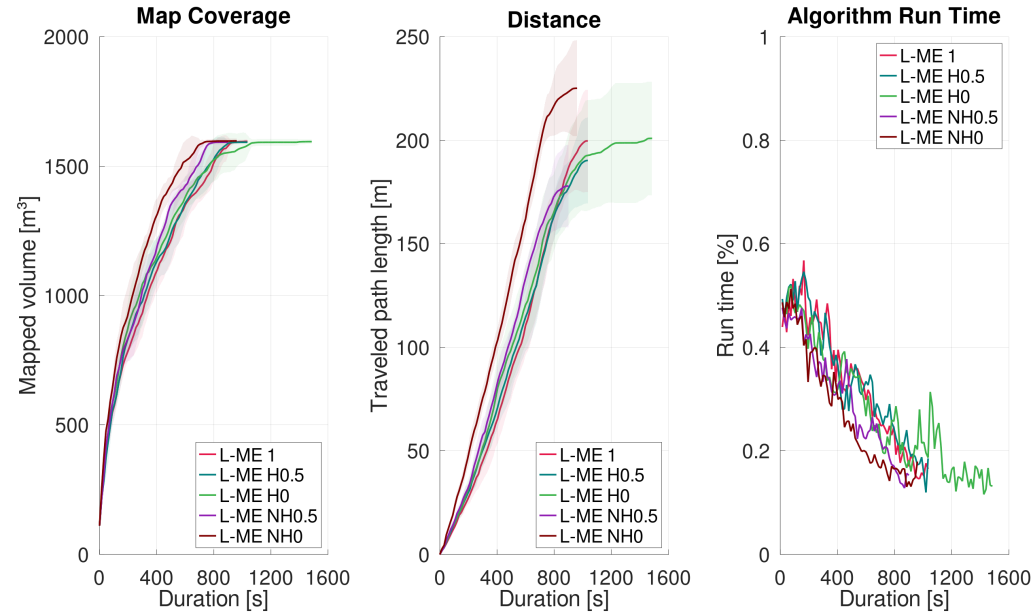


(c)

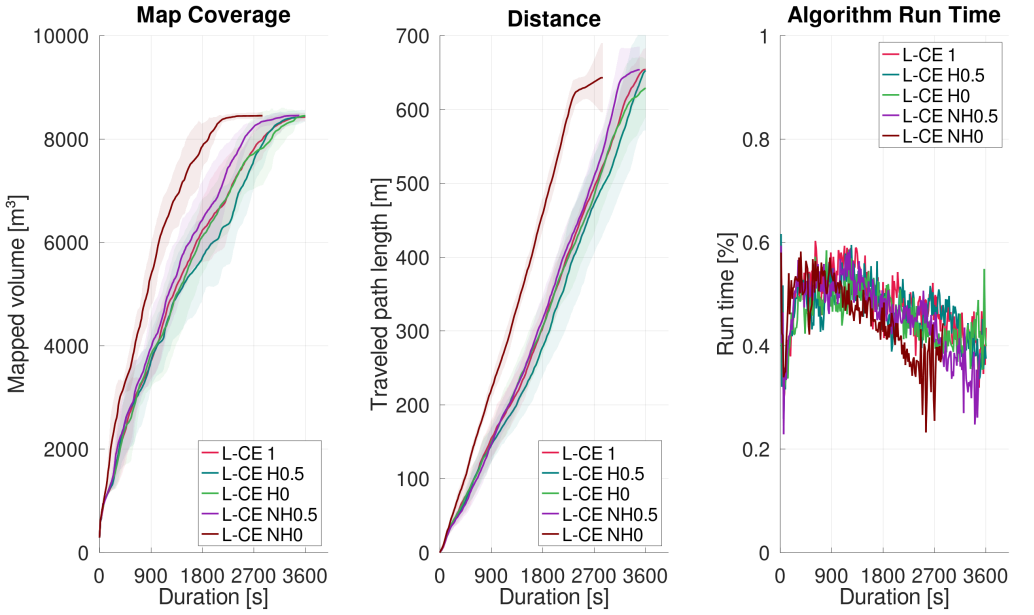
Fig. A.4.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying distance factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.



(a)

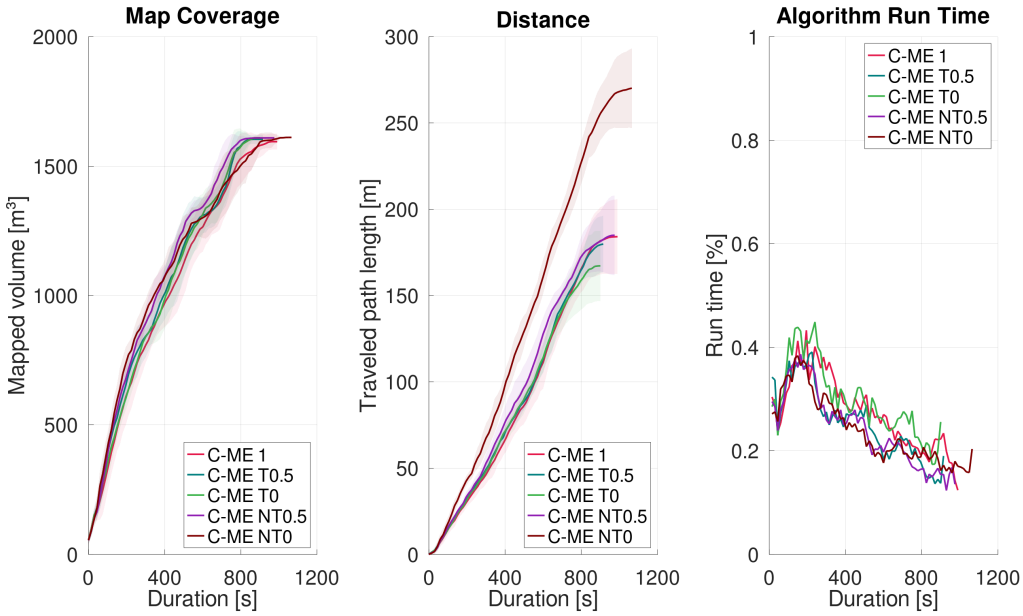


(b)

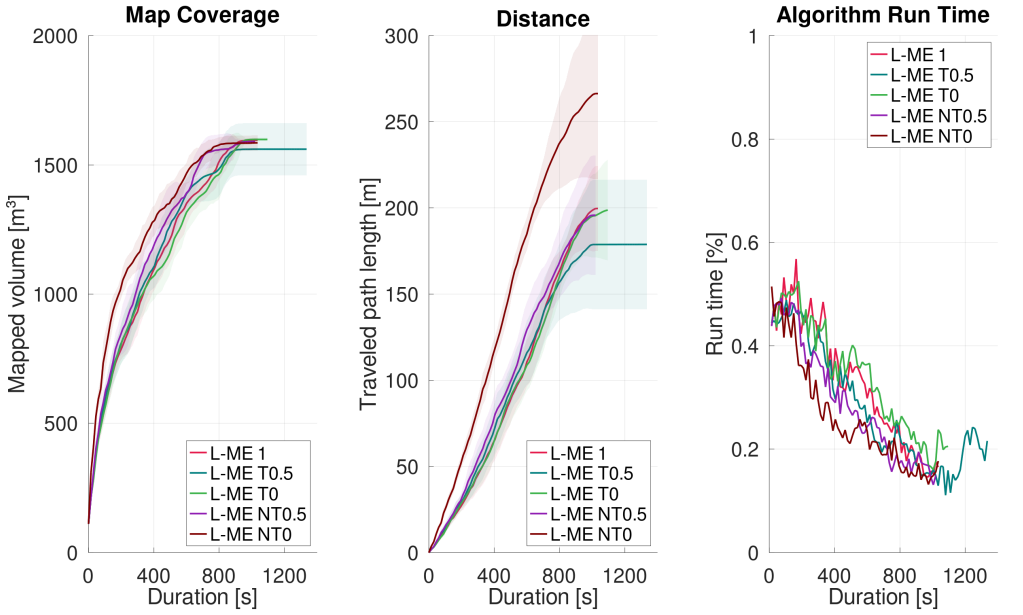


(c)

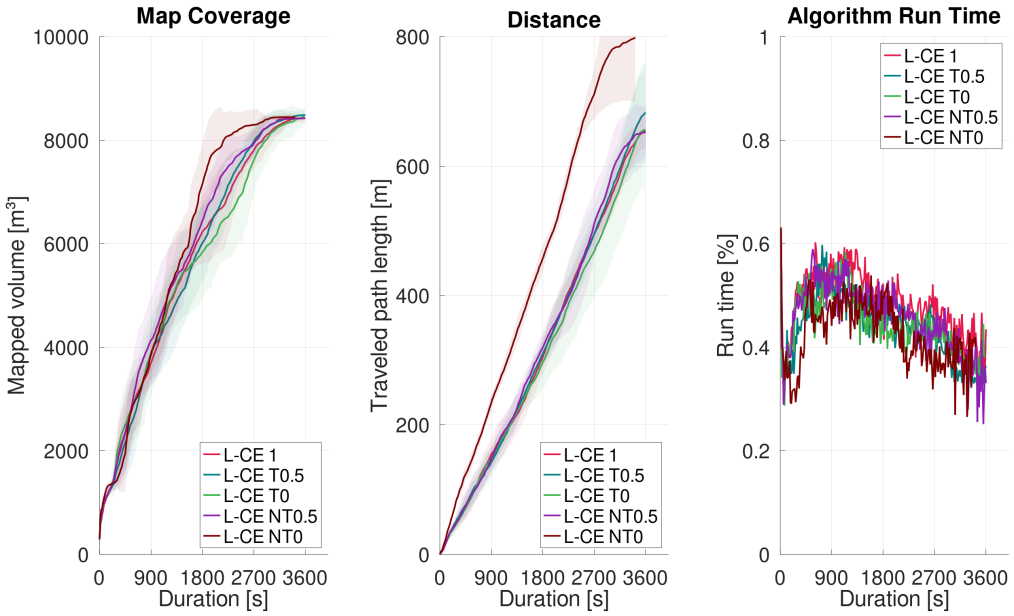
Fig. A.5.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying heading factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.


Map Coverage
Mapped volume [m³]
Duration [s]
Distance
Traveled path length [m]
Duration [s]
Algorithm Run Time
Run time [%]
Duration [s]
C-ME 1
C-ME T0.5
C-ME T0
C-ME NT0.5
C-ME NT0

(a)

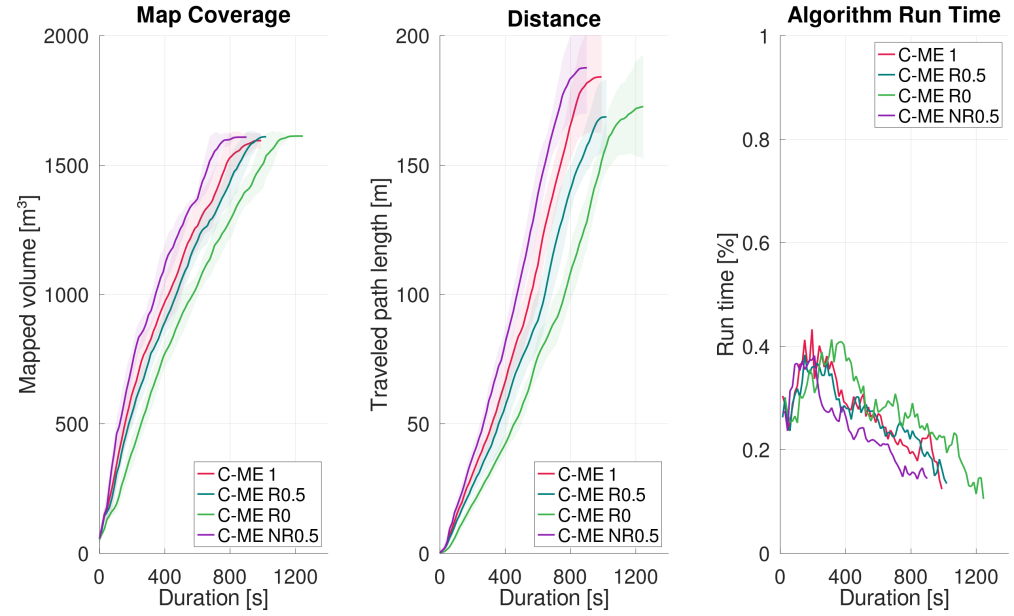

Map Coverage
Mapped volume [m³]
Duration [s]
Distance
Traveled path length [m]
Duration [s]
Algorithm Run Time
Run time [%]
Duration [s]
L-ME 1
L-ME T0.5
L-ME T0
L-ME NT0.5
L-ME NT0

(b)

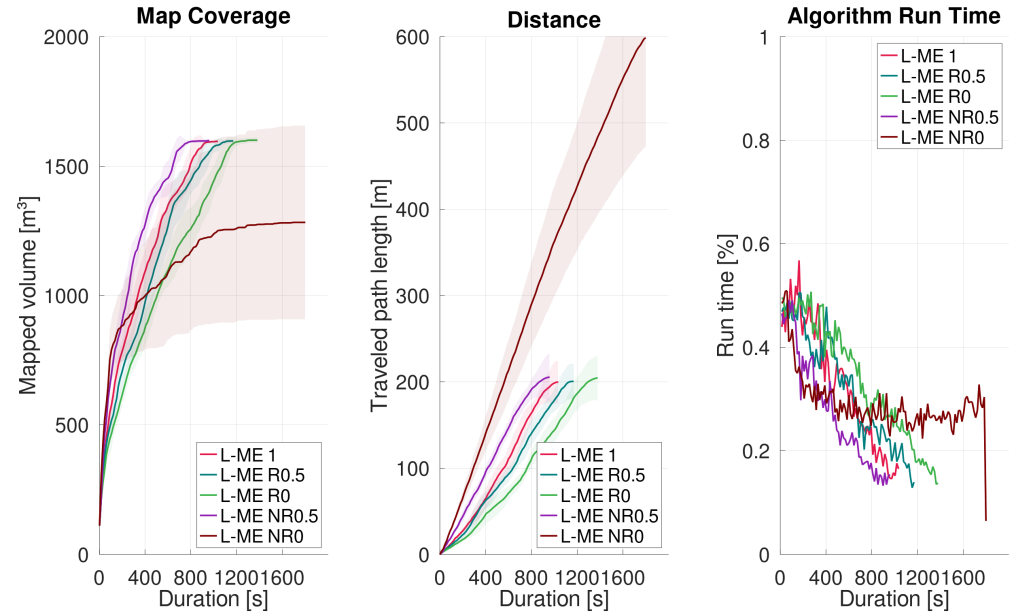


(c)

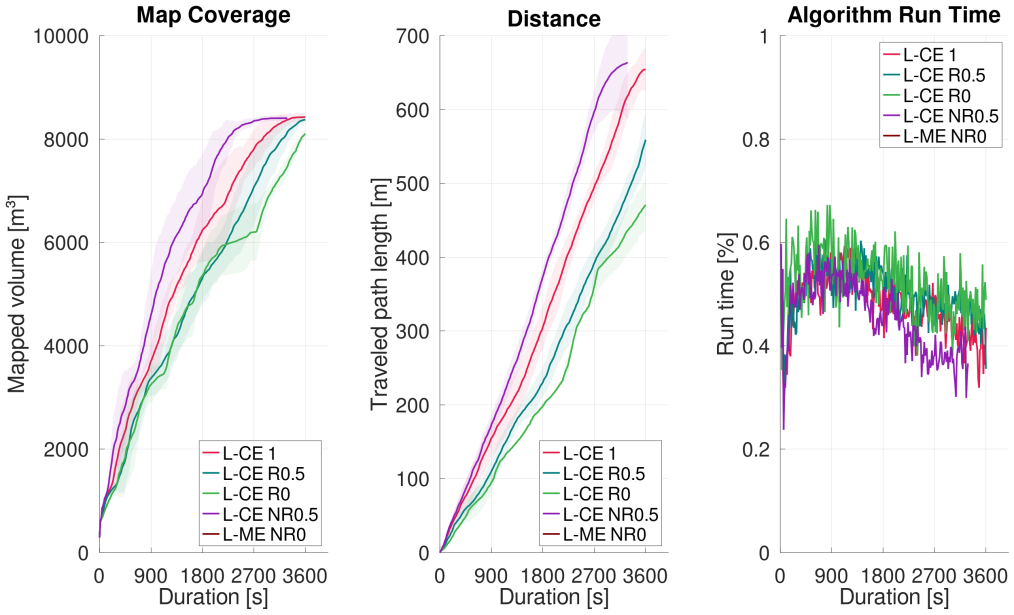
Fig. A.6.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying traversability factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.



(a)



(b)



(c)

Fig. A.7.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying radius factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.

Tab. A.6.: The impact of a varying distance factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	883.50	58.52	184.10	21.64	1594.6	31.1	0.288	0.023	0	10
C-ME D0.5	837.00	108.58	198.52	18.68	1607.6	2.4	0.275	0.025	0	10
C-ME D0	789.00	71.83	224.14	23.73	1606.7	3.3	0.278	0.018	0	10
C-ME ND0.5	957.00	74.09	169.74	17.98	1610.8	4.2	0.263	0.013	0	10
C-ME ND0	1087.50	58.42	162.53	14.66	1610.6	3.1	0.256	0.005	0	10
L-ME 1	945.00	72.11	199.69	24.65	1595.6	7.4	0.349	0.029	0	10
L-ME D0.5	835.50	100.67	211.00	33.63	1592.8	7.5	0.314	0.027	0	10
L-ME D0	834.00	142.16	260.20	49.57	1593.5	7.7	0.321	0.037	0	10
L-ME ND0.5	1140.00	108.86	211.81	21.81	1594.9	6.9	0.319	0.016	0	10
L-ME ND0	1204.50	69.71	188.75	8.41	1596.5	6.0	0.326	0.012	0	10
L-CE 1	3399.00	143.04	654.21	28.63	8424.2	74.8	0.488	0.017	0	10
L-CE D0.5	3038.33	214.00	686.26	33.13	8428.0	43.9	0.473	0.032	1	10
L-CE D0	2997.50	463.37	811.75	115.15	8366.3	44.1	0.457	0.043	4	10
L-CE ND0.5	3573.75	48.61	562.85	24.31	8442.2	70.9	0.479	0.009	2	10
L-CE ND0	3588.00	6.71	438.54	12.90	8095.0	106.4	0.504	0.010	5	10

Tab. A.7.: The impact of a varying heading factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	883.50	58.52	184.10	21.64	1594.6	31.1	0.288	0.023	0	10
C-ME H0.5	862.50	35.25	181.41	15.18	1609.6	2.8	0.279	0.042	0	10
C-ME H0	873.00	37.28	180.07	11.16	1604.4	17.1	0.308	0.023	0	10
C-ME NH0.5	790.50	81.06	170.11	27.18	1607.8	1.7	0.269	0.012	0	10
C-ME NH0	795.00	101.49	217.08	23.91	1608.5	2.7	0.267	0.008	0	10
L-ME 1	945.00	72.11	199.69	24.65	1595.6	7.4	0.349	0.029	0	10
L-ME H0.5	912.00	86.54	190.23	20.74	1592.2	7.8	0.361	0.035	0	10
L-ME H0	979.50	234.19	200.89	27.41	1594.6	8.9	0.328	0.041	0	10
L-ME NH0.5	790.50	64.48	177.85	19.83	1592.4	8.2	0.328	0.012	0	10
L-ME NH0	793.50	72.00	225.09	23.20	1597.4	9.0	0.305	0.013	0	10
L-CE 1	3399.00	143.04	654.21	28.63	8424.2	74.8	0.488	0.017	0	10
L-CE H0.5	3478.33	146.48	652.60	81.18	8447.9	114.5	0.472	0.026	1	10
L-CE H0	3333.00	207.58	628.93	39.52	8449.9	57.0	0.460	0.023	0	10
L-CE NH0.5	3225.00	127.48	653.80	31.43	8454.1	41.0	0.470	0.017	0	10
L-CE NH0	2424.00	161.43	642.98	46.89	8450.2	67.4	0.469	0.010	0	10

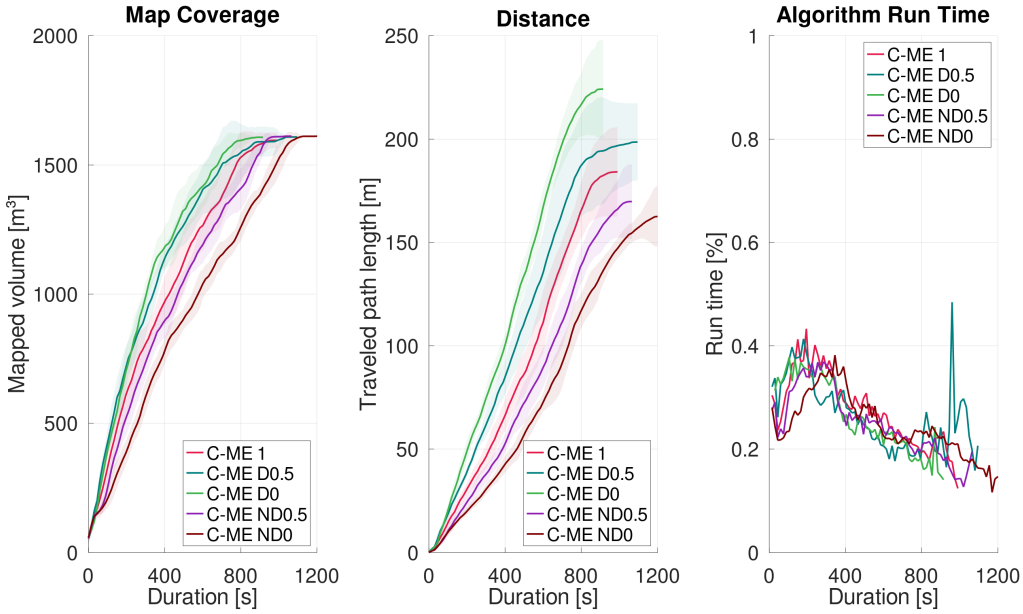
Tab. A.8.: The impact of a varying traversability factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	883.50	58.52	184.10	21.64	1594.6	31.1	0.288	0.023	0	10
C-ME T0.5	858.33	36.57	179.83	16.35	1603.7	17.5	0.263	0.007	1	10
C-ME T0	831.67	59.58	167.21	20.35	1608.9	2.3	0.308	0.032	1	10
C-ME NT0.5	831.00	88.37	184.94	23.03	1608.9	4.0	0.260	0.011	0	10
C-ME NT0	945.00	70.00	270.15	22.97	1610.8	3.1	0.250	0.007	0	10
L-ME 1	945.00	72.11	199.69	24.65	1595.6	7.4	0.349	0.029	0	10
L-ME T0.5	906.67	199.15	187.63	26.82	1591.8	19.2	0.330	0.025	1	10
L-ME T0	952.50	75.97	198.74	29.15	1598.2	6.3	0.364	0.037	0	10
L-ME NT0.5	876.00	121.08	195.77	34.61	1589.8	6.6	0.314	0.022	0	10
L-ME NT0	856.50	140.18	266.36	49.68	1585.0	29.5	0.285	0.018	0	10
L-CE 1	3399.00	143.04	654.21	28.63	8424.2	74.8	0.488	0.017	0	10
L-CE T0.5	3427.50	233.15	682.53	77.10	8487.5	96.9	0.446	0.016	0	10
L-CE T0	3457.50	185.48	657.12	89.56	8461.8	135.1	0.448	0.012	2	10
L-CE NT0.5	3258.33	186.56	652.52	39.15	8423.9	30.8	0.471	0.012	1	10
L-CE NT0	2896.67	326.70	798.29	99.12	8447.2	65.2	0.433	0.011	1	10

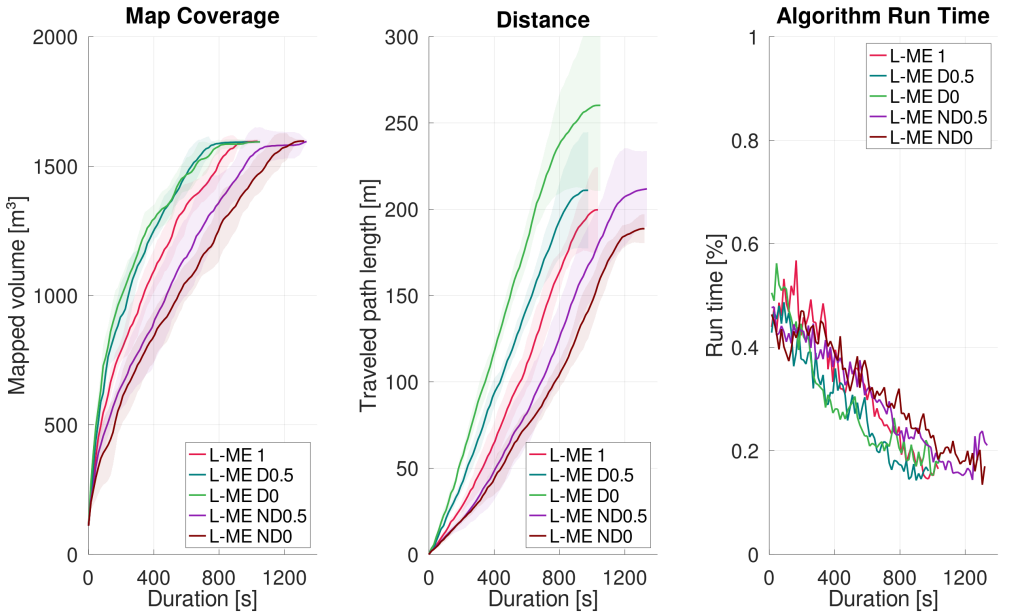
Tab. A.9.: The impact of a varying radius factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	883.50	58.52	184.10	21.64	1594.6	31.1	0.288	0.023	0	10
C-ME R0.5	952.50	56.18	168.62	14.25	1609.5	3.9	0.275	0.022	0	10
C-ME R0	1101.00	77.20	172.62	19.90	1612.1	3.4	0.291	0.026	0	10
C-ME NR0.5	798.00	65.88	187.60	17.79	1607.9	3.1	0.260	0.014	0	10
C-ME NR0	0.00	0.00	0.00	0.00	0.0	0.0	0.000	0.000	10	10
L-ME 1	945.00	72.11	199.69	24.65	1595.6	7.4	0.349	0.029	0	10
L-ME R0.5	1072.50	99.53	200.53	20.57	1596.9	8.3	0.337	0.020	0	10
L-ME R0	1240.50	81.12	204.59	25.82	1600.0	9.9	0.358	0.015	0	10
L-ME NR0.5	810.00	89.72	205.46	27.27	1598.2	7.8	0.301	0.011	0	10
L-ME NR0	1581.00	324.43	628.28	128.65	1588.9	5.3	0.241	0.023	5	10
L-CE 1	3399.00	143.04	654.21	28.63	8424.2	74.8	0.488	0.017	0	10
L-CE R0.5	3598.12	5.30	558.90	35.71	8373.2	66.0	0.504	0.023	2	10
L-CE R0	3595.00	8.66	470.60	35.47	8101.2	72.0	0.525	0.023	7	10
L-CE NR0.5	2922.00	201.84	663.56	76.07	8402.2	53.9	0.467	0.011	0	10
L-CE NR0	0.00	0.00	0.00	0.00	0.0	0.0	0.000	0.000	10	10

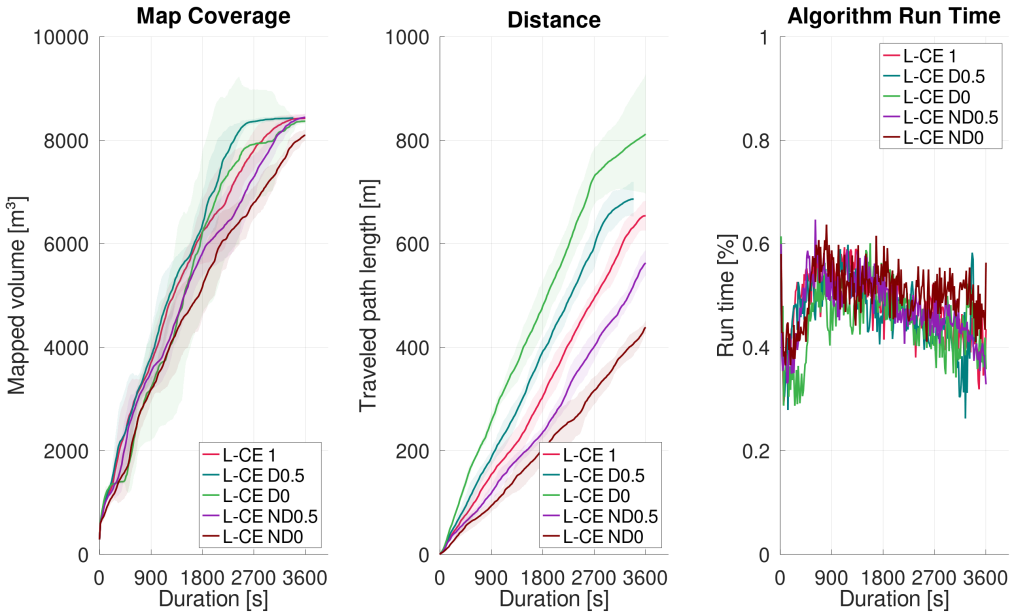
Topology-Based Node Area Inflation RNE



(a)

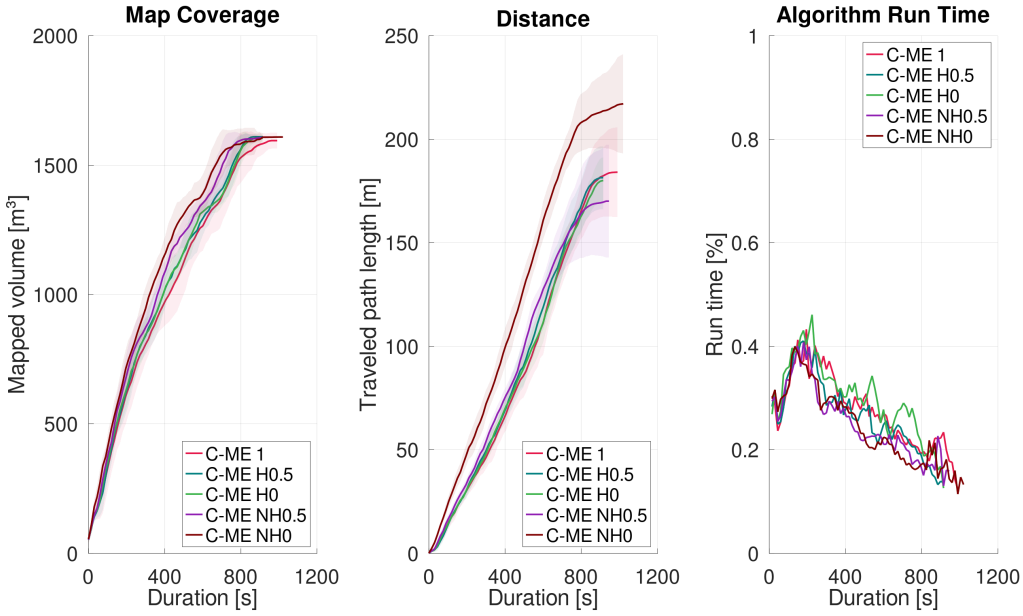


(b)

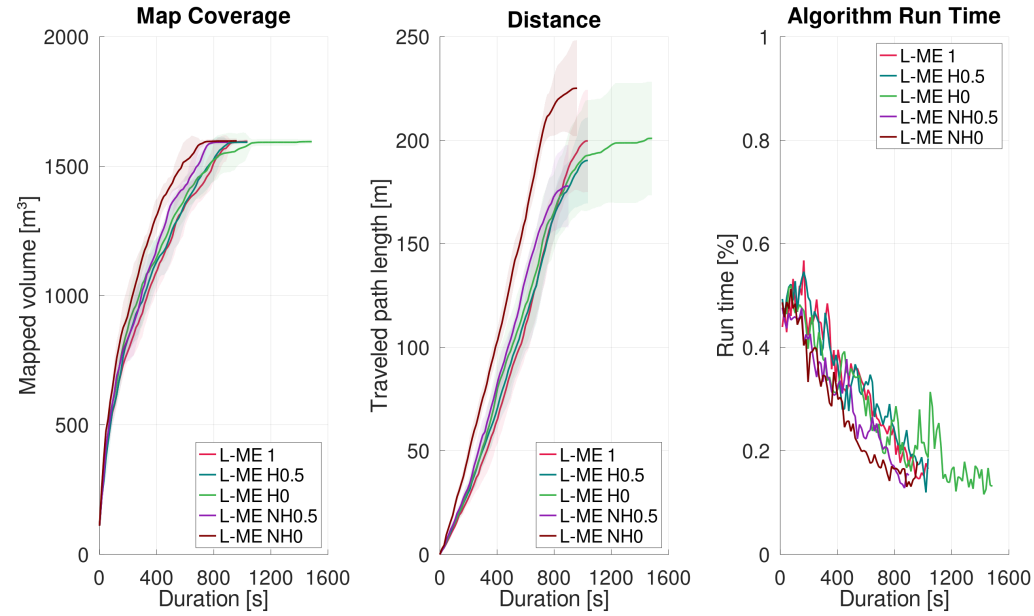


(c)

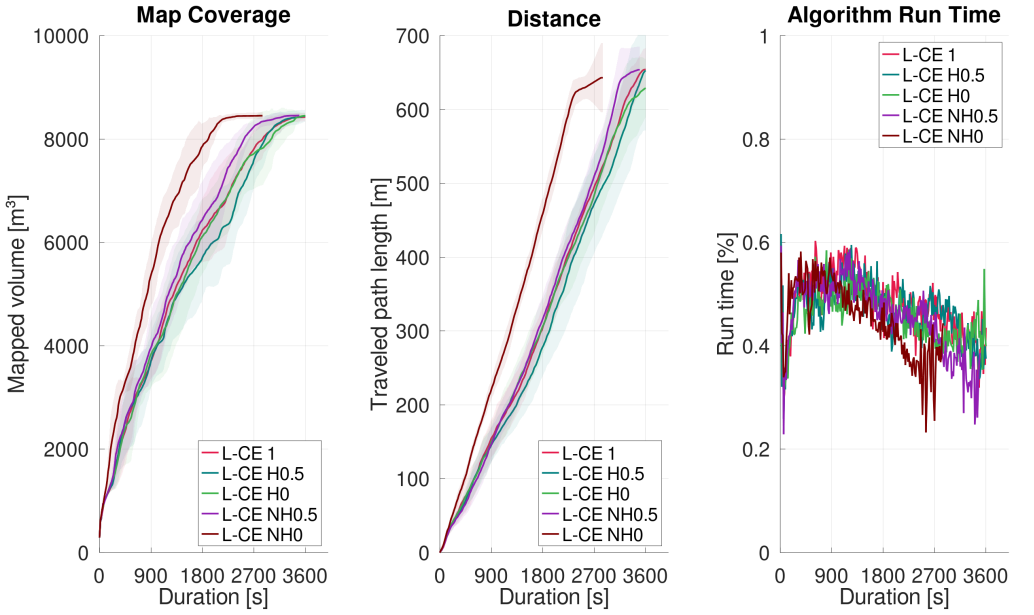
Fig. A.8.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying distance factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.



(a)

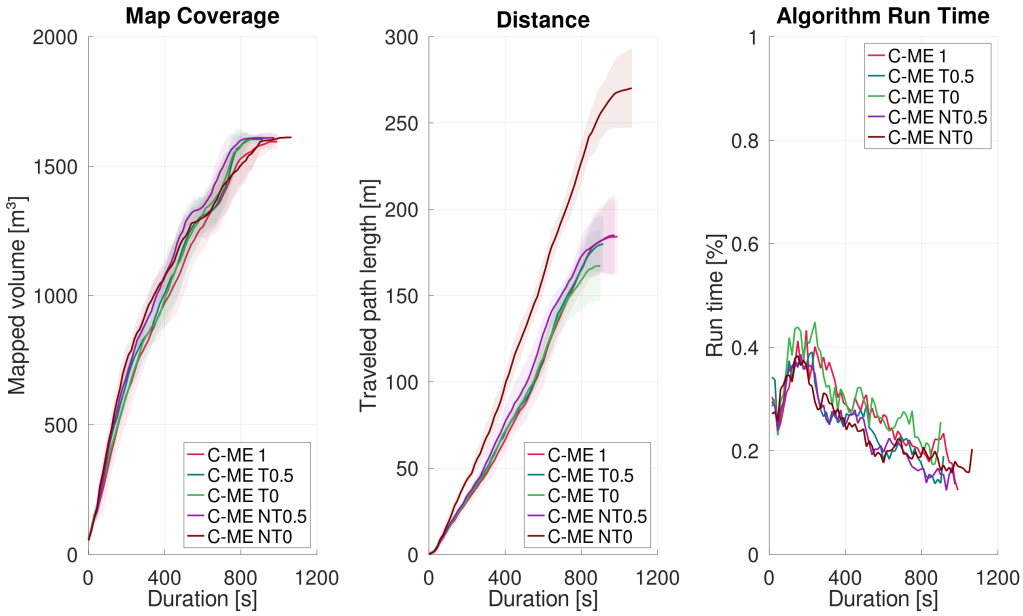


(b)

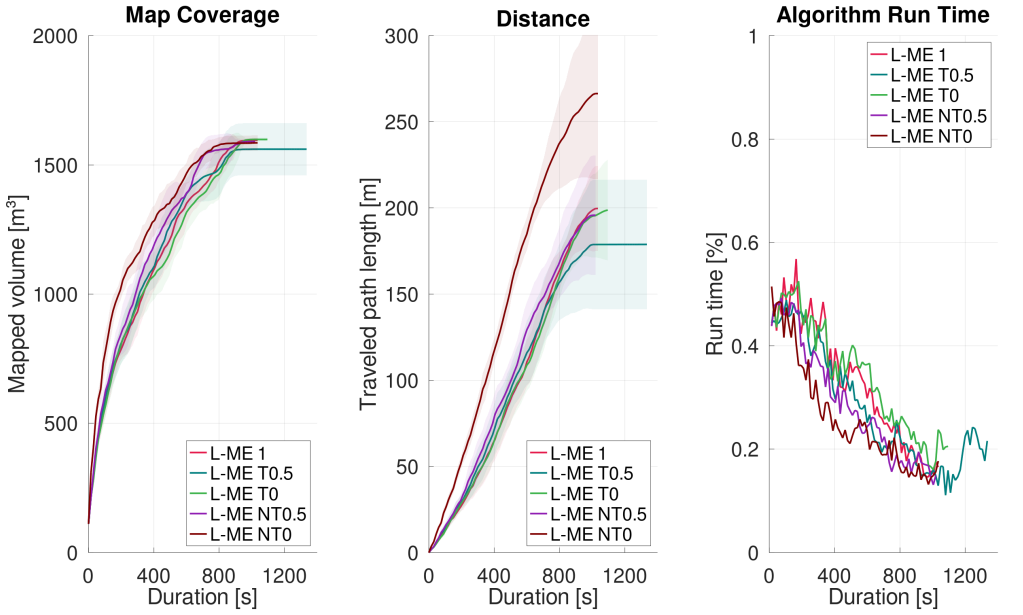


(c)

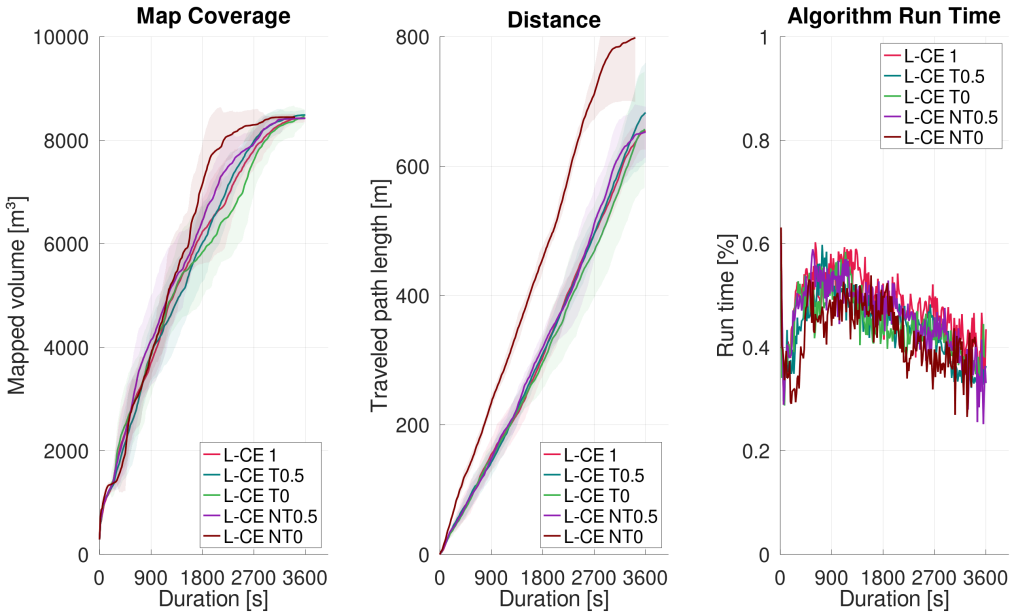
Fig. A.9.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying heading factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.


Map Coverage
Mapped volume [m³]
Duration [s]
C-ME 1
C-ME T0.5
C-ME T0
C-ME NT0.5
C-ME NT0
Distance
Traveled path length [m]
Algorithm Run Time
Run time [%]

(a)

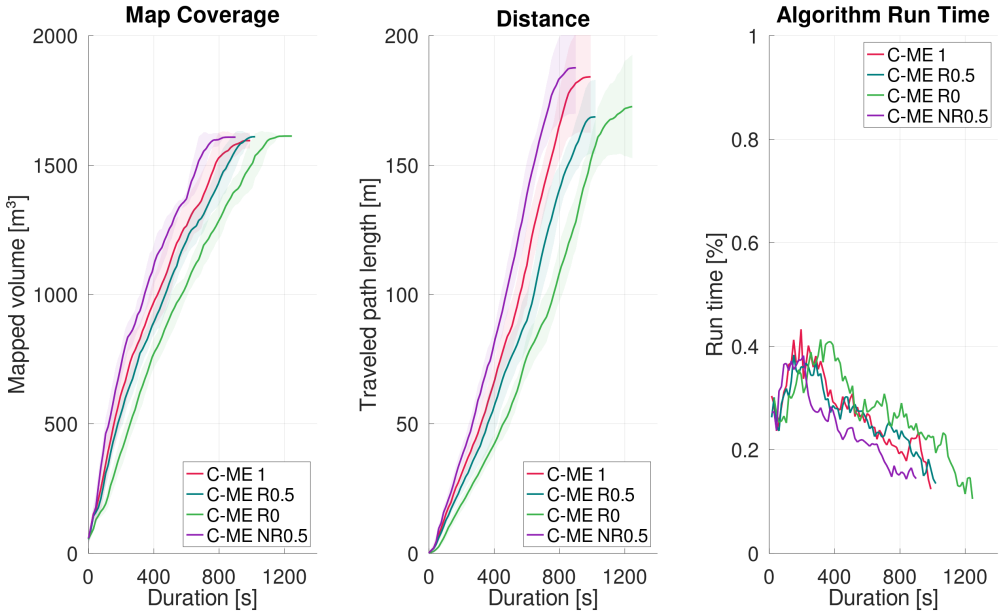

Map Coverage
Mapped volume [m³]
Duration [s]
L-ME 1
L-ME T0.5
L-ME T0
L-ME NT0.5
L-ME NT0
Distance
Traveled path length [m]
Algorithm Run Time
Run time [%]

(b)

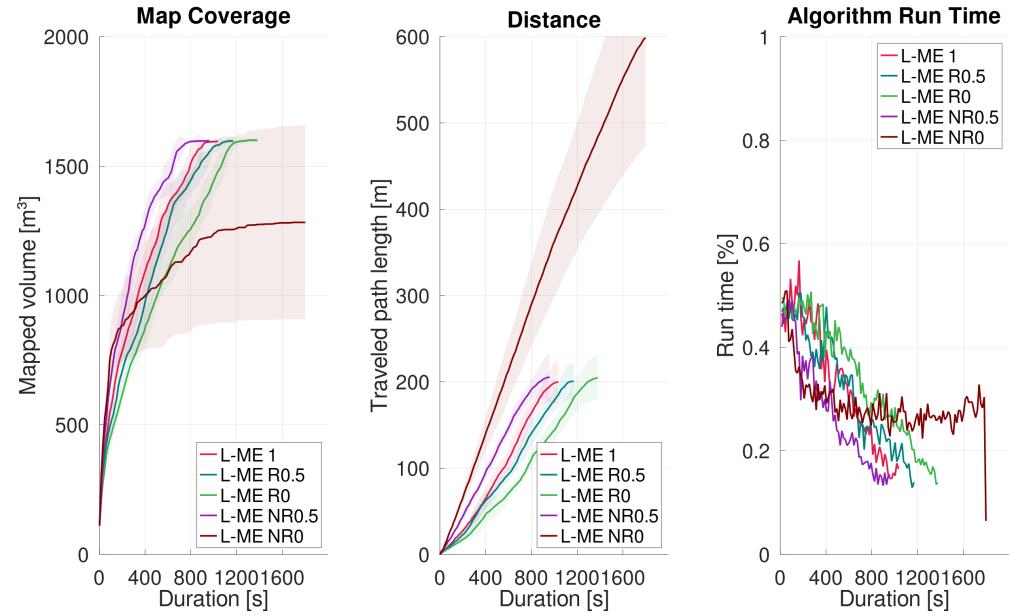


(c)

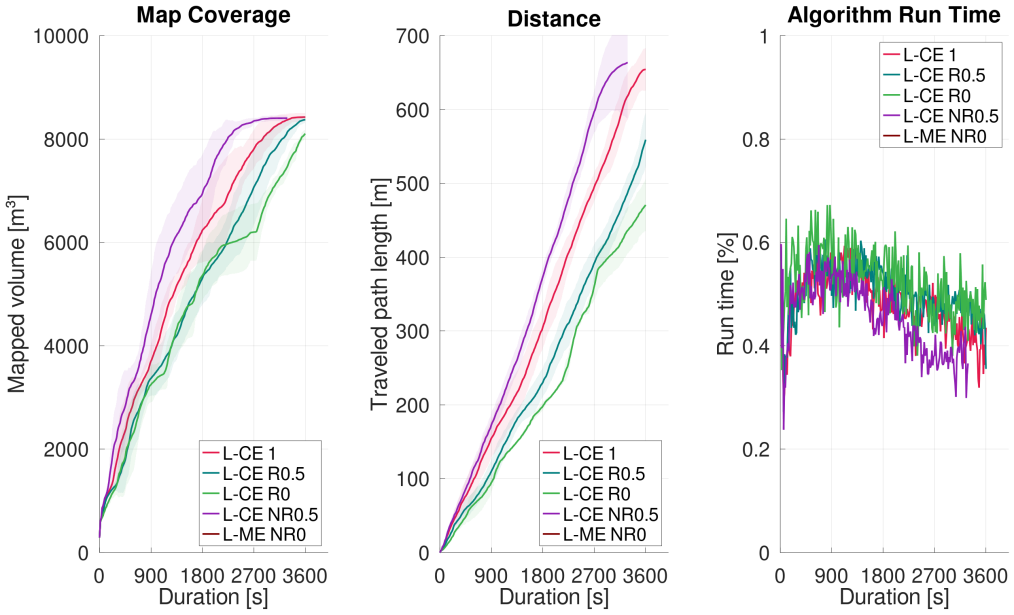
Fig. A.10.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying traversability factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.



(a)



(b)



(c)

Fig. A.11.: Mean mapped volume, path length and algorithm run time over the duration for different variants with a varying radius factor. The tinted areas show the SD of the particular values. A line ends at the final duration of the longest run of the particular variant. Sub-figure (a) shows the camera configuration in the medium environment, (b) the lidar configuration in the medium environment and (c) the lidar configuration in the cave environment.

Tab. A.10.: The impact of a varying distance factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	856.50	52.55	180.18	14.88	1605.5	16.3	0.276	0.009	0	10
C-ME D0.5	810.00	63.64	198.71	20.51	1608.0	3.5	0.267	0.012	0	10
C-ME D0	816.00	79.12	240.37	17.77	1607.8	4.2	0.268	0.011	0	10
C-ME ND0.5	954.00	53.94	180.88	15.36	1609.6	1.8	0.276	0.016	0	10
C-ME ND0	1098.00	54.68	166.35	12.27	1610.4	3.6	0.271	0.011	0	10
L-ME 1	963.33	165.64	216.64	33.34	1598.0	6.9	0.318	0.023	1	10
L-ME D0.5	863.33	133.35	210.62	38.81	1590.6	9.7	0.316	0.024	1	10
L-ME D0	841.50	344.52	235.82	39.60	1587.2	13.1	0.304	0.034	0	10
L-ME ND0.5	1084.50	134.04	203.41	34.12	1595.3	6.6	0.324	0.019	0	10
L-ME ND0	1192.50	74.28	187.75	20.92	1598.8	4.9	0.354	0.018	0	10
L-CE 1	3337.50	126.54	649.15	38.90	8394.5	54.3	0.473	0.020	0	10
L-CE D0.5	2782.50	334.24	633.35	75.62	8373.8	58.3	0.454	0.019	0	10
L-CE D0	2630.62	314.94	759.08	112.93	8403.0	56.7	0.442	0.012	2	10
L-CE ND0.5	3573.00	69.93	598.13	52.03	8360.6	149.3	0.478	0.017	0	10
L-CE ND0	3576.43	49.64	427.14	36.53	8234.8	168.8	0.526	0.024	3	10

Tab. A.11.: The impact of a varying heading factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	856.50	52.55	180.18	14.88	1605.5	16.3	0.276	0.009	0	10
C-ME H0.5	862.50	55.78	176.14	19.22	1610.4	2.8	0.273	0.015	0	10
C-ME H0	864.00	45.39	180.35	13.31	1608.3	3.2	0.275	0.007	0	10
C-ME NH0.5	862.50	47.97	189.48	16.43	1609.4	3.4	0.306	0.036	0	10
C-ME NH0	835.00	79.73	234.50	23.64	1602.1	15.3	0.302	0.022	1	10
L-ME 1	963.33	165.64	216.64	33.34	1598.0	6.9	0.318	0.023	1	10
L-ME H0.5	1035.00	273.50	208.73	21.82	1595.8	5.7	0.310	0.029	0	10
L-ME H0	928.33	126.71	203.11	45.79	1597.7	8.1	0.327	0.017	1	10
L-ME NH0.5	922.50	164.30	201.70	33.47	1595.6	10.2	0.356	0.040	0	10
L-ME NH0	741.00	55.32	208.85	19.74	1593.3	5.3	0.352	0.021	0	10
L-CE 1	3337.50	126.54	649.15	38.90	8394.5	54.3	0.473	0.020	0	10
L-CE H0.5	3330.00	267.02	600.97	63.74	8372.7	105.0	0.459	0.024	0	10
L-CE H0	3262.50	233.67	660.08	35.63	8431.7	40.3	0.454	0.008	0	10
L-CE NH0.5	3045.00	161.38	607.97	45.24	8380.2	55.9	0.498	0.030	1	10
L-CE NH0	2287.50	211.26	612.91	64.13	8376.6	62.2	0.495	0.015	0	10

Tab. A.12.: The impact of a varying traversability factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	856.50	52.55	180.18	14.88	1605.5	16.3	0.276	0.009	0	10
C-ME T0.5	810.00	56.12	167.72	15.91	1607.1	3.0	0.277	0.007	0	10
C-ME T0	831.00	52.54	170.91	13.76	1609.9	3.6	0.280	0.008	0	10
C-ME NT0.5	825.00	86.60	179.85	26.62	1607.4	2.8	0.313	0.016	0	10
C-ME NT0	957.00	61.56	270.76	29.66	1605.1	14.6	0.278	0.024	0	10
L-ME 1	963.33	165.64	216.64	33.34	1598.0	6.9	0.318	0.023	1	10
L-ME T0.5	952.50	130.06	219.18	49.81	1600.1	6.9	0.324	0.023	0	10
L-ME T0	900.00	129.03	192.78	25.97	1594.2	8.2	0.323	0.021	0	10
L-ME NT0.5	817.50	53.94	185.86	19.22	1593.2	4.5	0.358	0.030	0	10
L-ME NT0	921.67	89.41	281.77	38.01	1594.1	8.0	0.329	0.028	1	10
L-CE 1	3337.50	126.54	649.15	38.90	8394.5	54.3	0.473	0.020	0	10
L-CE T0.5	3320.62	179.63	625.75	58.45	8422.4	77.2	0.453	0.028	2	10
L-CE T0	3126.00	280.55	616.02	47.69	8464.5	49.6	0.457	0.019	0	10
L-CE NT0.5	3069.00	191.21	639.52	54.38	8439.4	58.1	0.482	0.029	0	10
L-CE NT0	2743.50	335.95	728.44	54.09	8370.2	103.9	0.452	0.024	0	10

Tab. A.13.: The impact of a varying radius factor on the exploration performance can be seen. The table shows the mean μ and SD σ of duration, traveled path length, observed volume and algorithm run time as well as the amount of total and failed runs for the camera and lidar configurations as C and L respectively in the medium and cave environments as ME and CE respectively.

Configuration	Duration [s]		Path [m]		Volume [m^3]		Run time [%]		Runs	
	μ	σ	μ	σ	μ	σ	μ	σ	Failed	Total
C-ME 1	856.50	52.55	180.18	14.88	1605.5	16.3	0.276	0.009	0	10
C-ME R0.5	898.50	53.11	162.95	16.51	1609.6	2.6	0.273	0.014	0	10
C-ME R0	1044.00	49.09	171.24	22.06	1607.0	16.0	0.272	0.009	0	10
C-ME NR0.5	801.00	32.56	190.51	15.19	1607.9	2.2	0.294	0.019	0	10
C-ME NR0	0.00	0.00	0.00	0.00	0.0	0.0	0.000	0.000	10	10
L-ME 1	963.33	165.64	216.64	33.34	1598.0	6.9	0.318	0.023	1	10
L-ME R0.5	1104.00	173.23	199.96	36.05	1592.8	9.4	0.323	0.023	0	10
L-ME R0	1197.00	87.12	208.00	24.74	1600.9	7.4	0.323	0.015	0	10
L-ME NR0.5	735.00	73.48	185.48	20.50	1594.5	8.7	0.345	0.025	1	10
L-ME NR0	1620.00	338.25	646.76	133.60	1586.1	15.6	0.259	0.023	5	10
L-CE 1	3337.50	126.54	649.15	38.90	8394.5	54.3	0.473	0.020	0	10
L-CE R0.5	3518.33	173.33	572.71	41.29	8313.9	148.7	0.478	0.016	1	10
L-CE R0	3597.00	6.71	482.16	26.35	8312.3	71.0	0.489	0.033	5	10
L-CE NR0.5	2874.00	447.95	669.14	130.08	8394.6	67.8	0.450	0.032	0	10
L-CE NR0	0.00	0.00	0.00	0.00	0.0	0.0	0.000	0.000	10	10