CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS

Autonomous Search Strategies for Unmanned Aerial Vehicles in Subterranean Search and Rescue Missions

Tomáš Musil

Thesis supervisor: **Ing. Matěj Petrlík**

Department of Cybernetics

May 2021

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Musil Tomáš**	Personal ID number:	**483430**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Autonomous Search Strategies for Unmanned Aerial Vehicles in Subterranean Search and Rescue Missions

Bachelor’s thesis title in Czech:

Strategie pro autonomní prohledávání pomocí bezpilotních helikoptér v podzemních záchranných misích

Guidelines:

This thesis is motivated by the DARPA SubT challenge, where a team of robots is deployed into an unknown subterranean environment in search for survivors and survival-specific objects. The focus of this thesis is to develop a search strategy for unmanned aerial vehicles based on covering surfaces likely to contain objects of interest. The following tasks will be solved:
• Design multiple single-robot search strategies based on coverage of surfaces by an onboard camera in a partially known 3D environment represented by an occupancy OctoMap [1]. Develop a map representation for storing information about surface coverage.
• Implement a trajectory planner that plans collision-free trajectories to exploration nodes/viewpoints [2].
• Evaluate developed single-robot strategies in the realistic Gazebo simulator.
• Implement segmentation of visited areas into shareable compact environment representation [3]. Use this representation to share surface coverage among robots. Assume reliable communication network with sufficient bandwidth.
• Extend exploration strategies with multi-robot cooperative approaches. Literature [4-5] can be a good starting point.
• Compare cooperative approaches with non-cooperative approaches.
• Prepare the developed exploration system for deployment on hardware platforms in real-world environments.

Bibliography / sources:

[1] Wurm, Kai M. et al. “OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems.” 2010.
[2] Bircher, Andreas, et al. "Receding horizon" next-best-view" planner for 3d exploration." 2016 IEEE international conference on robotics and automation (ICRA). IEEE, 2016.
[3] Blochliger, Fabian, et al. "Topomap: Topological mapping and navigation based on visual slam maps." 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.
[4] Wurm, Kai M., Cyrill Stachniss, and Wolfram Burgard. "Coordinated multi-robot exploration using a segmentation of the environment." 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2008.
[5] Maza Ivan, Ollero Anibal. "Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms" 2007 Distributed Autonomous Robotic Systems 6. Springer, Tokyo, 2007. 221-230.

Name and workplace of bachelor’s thesis supervisor:

Ing. Matěj Petrlík, Multi-robot Systems, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **08.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Matěj Petrlík	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my adviser Ing. Matěj Petrlík for the guidance and advice he has provided me throughout the work on this thesis and for the knowledge he has given me.

I would also like to thank the members of the MRS group, who allowed me to work on this topic and also provided me with an opportunity to test the designed approaches in the real world.

Last but definitely not the least, I would like to thank my family for their unrelenting support during my studies.

Abstract

This thesis deals with the design of multiple autonomous search strategies for a team of UAVs in search and rescue (SAR) missions in subterranean environments in which the goal is to find certain objects of interest - survivors, backpacks etc. and to report their positions to a base station outside the environment in a limited time. This problem is simplified as the problem of maximizing the surface area cumulatively covered by the cameras of all robots. To store information about surface coverage, a robust and incrementally built surface mapping structure is designed and implemented. To enable quick long-distance path planning and lightweight sharing of the robots' world representation, an incrementally built topology map is designed and implemented. A framework for sharing information about topology, coverage and frontiers among robots in a lightweight manner is also presented. Finally, three different fully autonomous search strategies that utilize the designed mapping and sharing structures and focus in different parts on coverage path planning and volumetric exploration are presented and evaluated both in the realistic Gazebo simulation and on a real UAV platform.

Keywords: unmanned aerial vehicle, search and rescue, robotic exploration, cooperative exploration, coverage path planning, topology representation

Abstrakt

Tato bakalářská práce se zabývá návrhem několika strategií pro tým autonomních bezpilotních helikoptér v podzemních záchranných misích, v nichž je cílem nalézt co nejvíce předem daných objektů - přeživších, batohů, atd. v omezeném čase a předat informaci o jejich poloze na základnu u vstupu do prozkoumávaného prostředí. Tato problematika je zjednodušena převodem na problematiku maximalizace celkově prozkoumané plochy povrchů prostředí pomocí kamer umístěných na robotech. Pro ukládání informace o prozkoumanosti povrchů je v této práci navržena a implementována robustní povrchová mapovací struktura vytvářená za letu na palubě drony. Za účelem rychlého plánování cest skrze prozkoumávané prostředí, aproximace vzdáleností cest k exploračním bodům a možnosti sdílení tvaru prostředí je dále navržena a implementována topologická online mapovací struktura. Dále je také navržena architektura pro sdílení informací o topologii, povrchu a exploračních cílech mezi roboty. Následně jsou navrženy tři strategie, které používají zmíněné mapovací struktury a kombinují metody pro klasickou robotickou exploraci a merody pro prozkoumávání povrchů. Tyto strategie jsou otestovány v simulaci i na reálné bezpilotní helikoptéře.

Klíčová slova: bezpilotní helikpotéra, záchranné mise, robotická explorace, kooperativní explorace, coverage path planning, topologická reprezentace

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Motivation

In the recent years, autonomous unmanned aerial vehicles (UAVs) have started playing an increasingly significant role in areas in which they can be beneficial and even life saving for humans. Teams of autonomous UAVs are currently being used for the mapping and archiving of historical objects [1, 2], such as churches and cathedrals, where they can reach areas that would require humans to construct scaffoldings and shut down the objects for long periods of time. Another currently researched and potentially very helpful topic is using autonomous UAVs for firefighting, where they could potentially react much faster than conventional firefighting approaches by flying into windows of burning buildings and extinguishing the fires [3] or by using swarms of UAVs in woodland areas [4]. UAVs are also currently employed in the project AERIAL CORE [5, 6] for the task of assisting construction workers that operate on tall and dangerous power line towers, where fatality rates are high.

The use of autonomous UAVs most important for this thesis is that of robotic search and rescue (SAR). The goal of SAR missions is to find survivors and possible hints of survivors in areas hit by disasters such as floods, earthquakes, explosions, cave ins etc or to find people lost in the wilderness. Up until the recent years, SAR missions required human rescue teams to enter hazardous environments and risk their lives to find survivors. Recent research efforts have demonstrated [7] that autonomous robot systems could help human rescue teams by quickly providing the rescue teams with a map of the environment and even localizing survivors. Autonomous robots could therefore reduce the loss of life by finding survivors faster and also by reducing the rescuers' exposure to dangerous elements in the environment.

This thesis focuses specifically on SAR in subterranean environments, such as caves, tunnels, mines and enclosed urban structures. This branch of SAR is particularly challenging due to the fact that global navigation satellite system (GNSS) localization is not available, and as such, robots must employ precise simultaneous localization and mapping (SLAM) algorithms. Furthermore, the signal range in these areas can be very limited, which means that the robots need to utilize strategies for creating a communication network and to be able to act completely autonomously, when no robots are in range and there is no connection to an outside team base.

To support research in the field of subterranean SAR, multiple competitions have been held, one of which is the DARPA Subterranean (SubT) Challenge. This thesis is motivated by the active and very successful participation of the CTU-CRAS-NORLAB team [8, 9] in the first two rounds of the DARPA SubT challenge, and aims to design strategies for the final round of this challenge.

1.2 DARPA SubT Challenge

This thesis focuses on the final circuit of the DARPA SubT challenge[1]. The main objective of the DARPA SubT challenge is to localize objects of interest (henceforth called artifacts) – backpacks, ropes, mobile phones, figurines of survivors etc. – in hazardous subterranean environments and then to transmit the information about the objects' positions to an outside teambase. The detection of these artifacts, some of which are shown on Figure 1.2, is achieved through various object-recognition algorithms and is not the focus of this thesis.

The score for a particular run is calculated as the number of artifacts that are correctly localized (i.e. reports with a position error less than approx. 5 m) and reported back to the outside base. The aim of the designed strategies is to maximize this score for a given run.

The previous three circuits of the DARPA SubT challenge have taken place in predefined types of environments — firstly in tunnel environments, then in urban environments, and lastly in cave environments. The environments to be explored in the final circuit will have features of all the previous three circuits and as such, the developed strategies

[1]https://www.subtchallenge.com

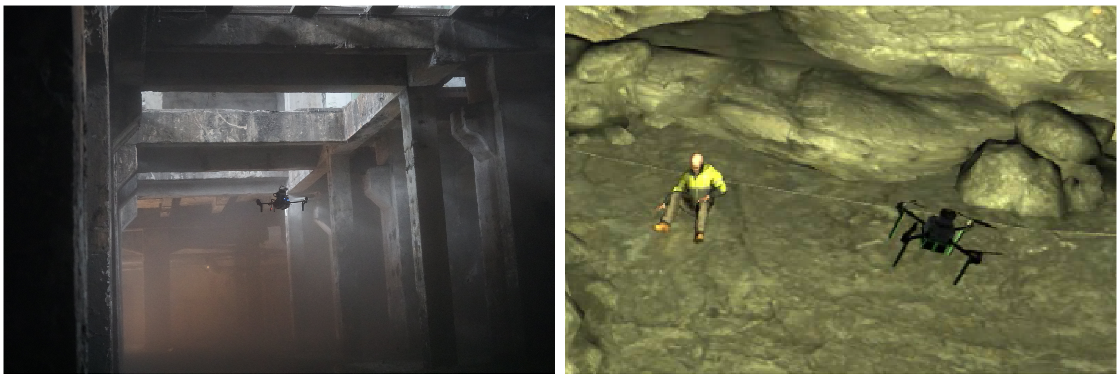
Figure 1.1: A UAV being used for exploration of an urban environment in a real world (left) and a UAV exploring a cave environment in the Gazebo simulator (right)

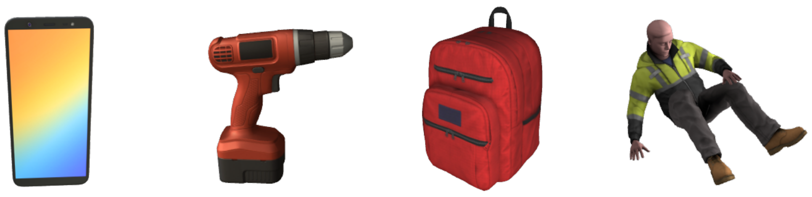
Figure 1.2: Some of the artifacts that are to be found in the DARPA SubT Challenge.

must not rely on any pre-defined environment structure. An example of an urban and a cave environment is shown in Figure 1.1.

1.3 Problem Specification

In this thesis we consider the problem of searching for known objects (henceforth called artifacts) in an unknown, possibly dangerous environment using a heterogenous multi-robot team comprised of unmanned ground vehicles (UGVs) and unmanned aerial vehicles (UAVs). The aim of this thesis is to develop strategies for the UAVs that maximize the number of artifacts that are detected and reported back to an outside base during a time-constrained mission. These assumptions are made for the missions:

1. The envinroments are completely unknown prior to the mission, meaning that the robots have no volumetric or surface representation of the environment at the start of each mission.

2. Communication is severly limited in these environments and loss of signal will usually occur after only a few tens of meters.

3. The environments may contain dangerous features (e.g. rockfalls, dust, bodies of water) and as such, robots may get stuck or completely destroyed during the mission. The designed strategies must account for this possible loss of robots.

4. Each robot is utilizing a method providing the UAV's position within the environment and continuously building an occupancy map representation of the environment. All mapping and planning algorithms designed in this thesis are based on the occupancy octree (OOT) [10] structure, and as such, we assume that each robot is either using the OOT as its primary occupancy map representation or that it is building an OOT in addition to its primary occupancy map. Furthermore, we assume that the drift of localization and mapping methods is negligible.

5. The composition of the robot team is not fixed. The team can be comprised of multiple UAVs, multiple UAVs and multiple UGVs, or a single UAV, depending on the specific mission.

6. We only design planning algorithms for the UAVs, but the mapping structures and map sharing framework are designed such that the UGVs can build the same maps as the UAVs and share them in the same manner with the UAVs.

7. If there are any UGVs in the robot team, the UGVs enter the environment first and deploy a set of breadcrumbs (communication nodes) that form a mesh network together with all UGVs and UAVs. The UAV should therefore not only be able to navigate back to home, but also to nearest communication nodes, if such nodes are available.

8. The UAVs can be launched at any time after at least 5 minutes have passed since the launch of the preceeding UAV. The homing procedure of a given UAV is initiated by a mission control node that we do not control in this thesis. We must, however, implement a method for quick pathfinding back to the base station.

9. Each UAV is equipped with sensors providing data for building an occupancy map (3D LiDAR, depth cameras) and also at least one RGB camera which is used for the detection of the objects of interest. We assume the visual detection software to be working perfectly and that if an artifact comes into the FoV of a UAV's cameras, it will be detected.

With the robot team composition and robot deployment timing pre-defined, the task of designing strategies collapses into the task of developing planning and mapping algorithms that will be running on board of the robots. For simplicity and compatibility, the mapping structures and algorithms developed in this thesis are used for each strategy. This means that each strategy is completely defined by the planning algorithms (both for planning short distance collision-free trajectories and long-distance viewpoints paths) that are employed on the UAVs and for this reason, the terms *strategy* and *planning algorithms* are used interchangeably further in this text.

The mission assumptions automatically put certain requirements on the strategies that are to be developed. Firstly, the individual UAVs must be able to act completely autonomously, since loss of signal can occur very soon after entering the environment. Therefore, the mapping and planning algorithms must be designed so that they can be run onboard of each UAV alongside visual detection and control algorithms. Since the flight time of the UAVs is limited, the algorithms should also be able to keep the UAV moving at all times to fully utilize the flight time.

With the given mission assumptions, we can simplify the problem as follows: Design mapping and planning algorithms for the robot team that guide the UAVs through trajectories that maximize the overall number of artifacts that are seen by the UAVs' cameras and then reported to an outside base station. To simplify the problem even more, let us define important terms for the surface S of the environment and the volume V is encompassed by S.

The volume of the explored environment, denoted as V, can be divided into occupied space V_{occ}, free space V_{free} (free of obstacles) and unknown space V_{unk}. We further define the known space V_{kno} as $V_{kno} = V_{occ} \cup V_{free}$. Since sensor perception considered in this work stops at most opaque surfaces, we can say that in a theoretical scenario, where we consider the sensors and mapping to be perfect, all points of V_{occ} are boundary points between V_{free} and V_{unk} and form a surface S which, for subterranean environments, can be thought of as a manifold.

At a given time, we can define $S_{kno} \subseteq S$ as the known parts of the surface (e.g. the ones discovered by the robot's sensors used for mapping) and $S_{unk} = S \setminus S_{kno}$ as the unknown parts of the surface. Furthermore, since the FoV of the sensors used for mapping can be different from the FoV of object recognition cameras, we define $S_{insp} \subset S$ as surfaces that have been inspected by the visual sensors and S_{uni} as surface that has not been inspected. We also assume the object recognition FoV to be a completely encompassed in the FoV of the mapping sensors and therefore we can write

$$S_{uni} \cup S_{insp} = S_{kno} \subseteq S. \tag{1.1}$$

Because all artifacts considered in this thesis are opaque and therefore form a part of S, and since we assume that the detection algorithms will detect an artifact every time it is visible in the robot's object recognition FoV, we can transform the problem of finding artifacts into the problem of inspecting surfaces that are likely to contain artifacts. Based on information about the artifacts' likely positions available at the start of the mission (for example in some cases we might know that no artifacts will be on the ceiling parts of the surface), we can define a reward function for parts of the surface $\mathbf{R}(s), s \in S$.

If we then assume that with the used strategies, all robots will manage to report all artifacts that they find back to the base station, we can transform the problem of searching for artifacts into the problem of maximizing a value $\mathbf{R_{total}}$ defined as

$$\mathbf{R_{total}} = \oint_{S_{total}} \mathbf{R}(s) ds, \tag{1.2}$$

$$S_{total} = \bigcup_{i=1}^{N} S^{i}_{insp}, \tag{1.3}$$

where S^i_{insp} denotes the surface inspected by robot i at the end of the mission and N is the number of robots. If $\mathbf{R}(s)$ is constant, the problem further simplifies into the problem of maximizing the total area S_{total} covered by the robots' cameras.

This problem could be solved by available coverage path planning algorithms if the surface S was known apriori. However, that is not the case in this work. Therefore, the strategies to be designed must combine methods for expanding the known surface S_{kno} and known volume V_{kno}, and also methods for inspecting the known surface.

For these reasons, this task has been divided into these sub-tasks which form the assignment of this thesis:

1. Design and implement a surface mapping structure for storing surface coverage information (S_{insp} and S_{uni}) that can be incrementally built onboard the UAV.

2. Design and implement a topological environment representation that can be used to share information about topology, surface coverage, and frontiers with other robots.

3. Develop and evaluate multiple strategies that try to solve the problem of simultaneous exploration and surface inspection by utilizing the occupancy, surface, and topology maps.

4. Implement a collision-free trajectory generator that the planning algorithms can use for short-distance collision-free navigation.

5. Extend the designed strategies with cooperative approaches.

1.4 Preliminaries

1.4.1 Mathematical Notation

Symbol	Meaning
$\mathbf{x}$	column vector in Cartesian coordinates
ξ	viewpoint consisting of a position $\mathbf{p}_\xi$ and a heading φ_ξ
$\mathbb{O}, \mathbb{S}, \mathbb{F}, \mathbb{G}, \mathbb{L}$	occupancy octree, FacetMap, FrontierMap, SegMap, L-SegMap
$V_{free}, V_{unk}, V_{occ}$	free, unknown and occupied volume
S_{insp}, S_{uni}	inspected and uninspected surface
m	voxel
$N_6(m)$	6-neighborhood of a voxel m
$\mathbf{f}_g$	frontier cluster point
F	frontier cluster
ψ	facet consisting of a position $\mathbf{p}_s$ and a normal $\mathbf{n}_s$
Ψ_{insp}, Ψ_{uni}	inspected and uninspected facets
σ	SegMap segment with a center $\mathbf{c}_\sigma$ and voxels $m \in M_\sigma$
$\mathbf{p}_{ij}$	position of a portal between σ_i and σ_j

Table 1.1: Mathematical notation and symbols that are common in this thesis

1.4.2 Occupancy OcTree

The primary environment representation of the designed system is a binary occupancy octree (OOT) implemented from the OctoMap library [10] which is being created by an octomap server node from available depth sensor data. Non-binary OOTs exist and are commonly used in robotics, but for the purpose of this task, only a binary OOT is used. A binary OOT, denoted further as $\mathbb{O}$ is a structure that divides space into cubic volumes, called voxels, that are assigned one of these three values:

- Free - the whole space of the voxel is known to be free of obstacles
- Occupied - the voxel contains an obstacle
- Unknown - no information is available for the voxel (this is the default state of all voxels)

The OOT stores voxel information in a memory-efficient manner by first creating the voxels at a set resolution and when there are 8 adjacent voxels with the same value, collapsing them into a single, larger voxel with double the side length. The voxels are then stored in a tree with a maximum depth of 16, at which the smallest voxels reside. The resolution (side length of the smallest possible voxels) of the OOT used in this work was chosen as $\rho = 0.2\,\mathrm{m}$.

Since many methods in this thesis rely on iterating through the OOT and looking at voxels adjacent to a given voxel, we define a set of voxels $N_6(m)$ for a given voxel m as the 6-neighborhood of m — meaning all voxels of the same size as m that share a face with m.

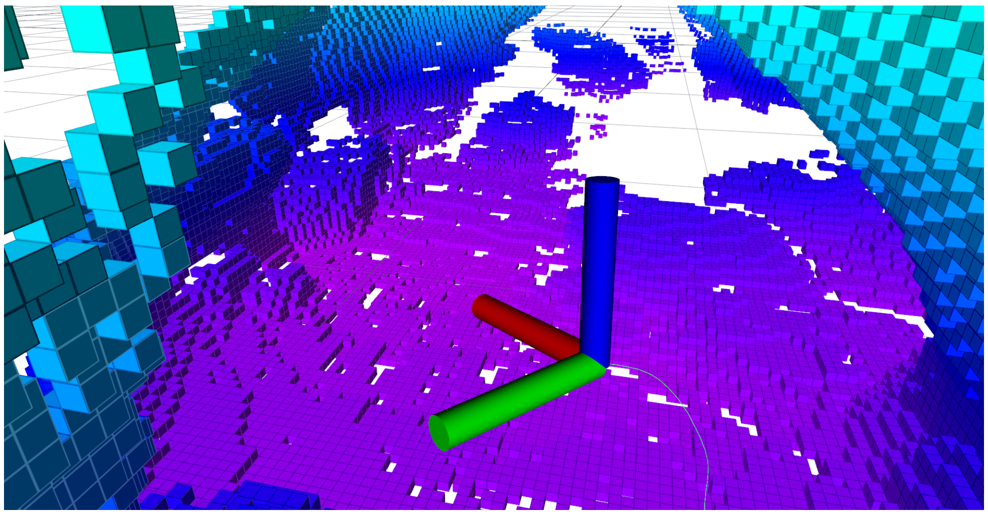

Figure 1.3: Occupancy octree representation of the environment being explored. The UAV's pose is represented by the three axes.

1.4.3 UAV Control Interface

A UAV in free space has 6 degrees of freedom (DOF) — 3 for position and 3 for orientation. In this work, we do not deal with the very rich subject of controlling the UAV's 6 DOF and assume that there is a working control manager node that takes care of this control. We only navigate the UAV by setting viewpoints ξ, which are defined by a position $\mathbf{p}_\xi$ and a rotation around the world z axis ϕ_ξ, and then assume that collision-free path planning, trajectory generation, trajectory tracking and control algorithms will safely navigate the UAV to these viewpoints across small distances.

We only set viewpoints that are safe, which in the context of this work means that the distance from the nearest occupied or unknown voxel from a given viewpoint's position is higher than some value d_{safe}. The obstacle distance at a given point $\mathbf{p}$ can be easily queried from a k-d tree [11] that is built from the OOT.

We also utilize the collision-free trajectory planner to query whether a safe path exists between two given points. The calculation of collision-free paths is computationally very expensive and is usually only done at the last step of the planning algorithms to ensure that reachable viewpoints are being set.

For higher-speed flight, we also define the concept of a *viewpoint path* as a series of viewpoints $\Xi = (\xi_1, \xi_2, \xi_3, ..., \xi_n)$ that are given as input to a collision-free path planner and trajectory generator node that creates a safe and smooth trajectory through these viewpoints. The downside of this is that to comply with the dynamics of the UAVs, it is not ensured that the position and heading of each viewpoint will be perfectly reached and also while flying through the viewpoint, the UAV can be rotated not only along the z axis. We accept this fact and optimistically set viewpoint paths assuming that the viewpoints will be reached.

1.5 Related Work

The field of manned and unmanned robotic SAR is currently being intensely researched. Many advances have already been made in using robots for search and rescue in disaster areas such as floods, earthquakes etc, collapsed tunnels etc. An extensive survey of recent state of the art approaches to robotic SAR can be found in [7].

As stated by the authors in [7], most research efforts so far have focused on the development of individual robots which usually depend on an external control center for route planning. To the author's best knowledge, not much current research has been focused on this particular task of autonomous simultaneous exploration and surface inspection in subterranean environments by a team of UAVs and as such, the related works presented below are the building blocks that could be used to solve this task.

1.5.1 Single-robot Volumetric Exploration Methods

The field most related to the task given in this thesis is autonomous robotic exploration of enclosed environments. These methods are different from the given task because they only focus on expanding the known space V_{kno} and not on the inspection of surfaces. The work that comes probably the closest to the single-UAV part of this task is [12], in which the authors present a system for both exploration and surface inspection, but only for the situation when the surface is represented by an apriori fully known mesh, which is not the case in this thesis.

The probably most popular approach to robotic exploration is to create strategies based on the concept of frontiers [13, 14, 15, 16], first described in [17]. The original approach is to detect frontiers – the boundary between V_{free} and V_{unk} in the world representation of the robot and then to navigate the robot to areas near the frontier.

Detecting the frontiers for the whole available map and planning collision-free paths in large environments can however be very computationally expensive. To tackle these issues, many authors enhance the RHNBV method by building navigation graphs or only detecting and exploring nearby frontiers.

For example in [13], the authors divide the exploration task into two modes of operation:

1. As long as there are frontiers near the robot, explore those frontiers using a local planer utilizing rapidly exploring trees (RRT) [18] which maximize information gain while minimizing traveled distance.
2. If there are no frontiers near the robot, switch to a long-distance planner using a graph that is built during the mission to navigate to the nearest frontiers.

In [14] the authors focus more on faster local exploration by constructing a structure of frontier clusters and then solving a traveling salesman's problem (TSP) for a rich set of viewpoints that are sampled near the clusters. The local viewpoint paths are then recalculated whenever the occupancy map changes, which allows for very fast exploration.

The principle of computing frontiers for the whole environment only when there are no nearby frontiers and the concept of viewpoint trajectories is adapted in the strategies presented in this thesis in chapter 4.

1.5.2 Topology Mapping

To allow for efficient high-level mission planning and fast long-distance navigation, it can be useful to build a more abstract representation of the environment than an occupancy map. The task of abstraction and compression of the robot's world representation is also important for cooperative methods for subterranean SAR, since communication is usually severely limited in subterranean environments. Many approaches have been designed for the task of building abstract topological maps but most of the research in this field so far has focused only on 2D environments [16, 19, 20, 21]. The 2D topological representations are usually based on Voronoi decomposition, which is not well applicable to 3D environments.

A very simple representation of the environment in 3D could be the trajectory traveled by a robot. This representation can allow for very naive backtracking to the base station and can also be used by other robots to extract very rough information about the environment (e.g., "the robot went through this tunnel, so I should go explore the other one").

Navigating along the traveled trajectory would however suffer in areas which the robot has passed multiple times. To solve this, the authors in [13] only build the graph along the trajectory if the given positions have not yet been traveled. This results in better navigation, but the environment's shape can only be reconstructed from it if the environment is comprised solely of tunnels of a roughly fixed size, which is generally not the case in this thesis.

An approach more suitable for environments that are more generally structured is the TopoMap, presented in [22]. The TopoMap tries to fill the explored environment with convex spherical clusters of free space voxels which are then merged into larger convex clusters. Connections, called portals by the authors, are then found at positions where the robot can transition from one cluster to another. By connecting together portals inside each segment, the authors create a navigation graph that can be used for quick path planning. Because shapes of convex clusters can also be easily approximated (by spheres, ellipsoids or bounding boxes), the TopoMap was chosen as the basis for topological mapping in this work and is further described in chapter 3.

1.5.3 Coverage and Inspection Planning

Coverage path planning is a field that deals with the problem of for example how to efficiently cover a large area on a map with a team of UAVs with downward facing cameras or how to clean a room or mow a lawn in the smallest amount of time. Most coverage planning methods deal with a 2D area which they decompose into polygons and then traveling through them in a zigzag manner [23, 24].

This could be used for the purpose of searching a subterranean environment if we would only assume artifacts to be on the ground and if the environments would consist of large areas in which the ground can be decomposed into 2D areas for the zigzag pattern. However, in the DARPA SubT challenge, artifacts can be on the walls and ceilings as well and the environments can be very vertically diverse, and for this reason, we focus more on the task of robotic surface inspection.

Surface inspection path planning methods [12, 25, 26] seek to cover a given surface with a moving camera (usually on a UAV) while minimizing some defined variables (e.g. time, distance). In most of these methods, the surface S is represented by a mesh that is known apriori, which is not the case in this work. The ideas from these works are however utilized for surface inspection of the surface representation designed in this work, described in chapter 2.

One last example of surface inspection worth mentioning is [27], in which the authors explore an environment with RGB-D cameras and combine surface and volumetric exploration. In their case, since they use the RGB-D camera for both the visual and depth sensing, the FoV for visual detection and for frontier exploration is nearly identical, which differs from the sensor model we assume, but the idea of utilizing saliency for exploration is interesting and could potentially be useful for SAR.

1.5.4 Cooperative Exploration Methods

In SAR missions, cooperative coverage path planning [23, 28] and exploration strategies [15, 29, 30, 31, 32] are vital, as without utilizing these strategies, robots could search areas already searched by other robots, which is generally not wanted. Many multi-agent planning strategies however assume centralized planning or planning where all robots can communicate at any given point. The strategies considered in this thesis must however account for loss of signal and therefore we only assume cooperative strategies in which the robots make decisions autonomously with the current information obtained from other robots.

One example of cooperative exploration that accounts for loss of signal is described in [15], where the authors define a utility value for frontier cells in a 2D occupancy map and then update the utility of the cells based on current positions and goal positions of other robots. This approach is robust in that in the worst case scenario, when no robots can communicate, each robot will explore the whole environment and any information from other robots reduces the space that is explored redundantly.

1.6 Outline

In chapter 2 the design of a surface representation that is built from the OOT, and is further used for surface inspection path planning, is described.

The design of a topological map representation for quick long-distance path planning based on the principles of the TopoMap [22] is presented in chapter 3.

The designed methods for extracting information about frontiers [17] and using this information for the generation of exploration viewpoints are discussed in chapter 4.

The designed methods for compressing information about surfaces, topology and frontiers into a shareable map and a framework for sharing and updating these maps is presented in chapter 5.

Three proposed autonomous search strategies that utilize the designed mapping structures are then described in chapter 6.

Lastly, the designed strategies and mapping structures are evaluated both in simulation and in the real world in chapter 7.

Chapter 2

Surface Mapping

Contents

The first goal of this thesis is to design and implement a robust surface representation that can store information about surface coverage and allow for surface inspection planning. The most commonly used surface representation used in surface inspection methods is a mesh, used for example in [12] or [25]. These methods consider the mesh to be static and known at the beginning of the mission. However, in the missions considered in this thesis, the environment is *completely* unknown apriori and the only available information about it is in the form of the OOT that is continually being built during a mission.

It could be possible to build a mesh dynamically from the OOT, but this approach was deemed too complicated and is not further discussed. The OOT by itself also can not be used as the required representation, as we have no reliable way of storing surface coverage information in it. Because the surface of the environment is represented in the OOT as occupied voxels, we could, theoretically, create a separate octree with the same resolution as the OOT, in which we would assign each occupied voxel a value that represents how well it has been covered by the robot's cameras. This approach was tried in the early stages of the work on this thesis, but was soon abandoned, as the OOT can be noisy and the occupied voxels in the binary OOT can shift. This approach also does not give any implicit information about the normals of the surfaces, which can be useful for better determining whether a given part of the surface has been inspected, as detecting some artifacts might require the robot to look at them at a certain angle.

For this reason, a novel surface representation, named FacetMap, is designed in this thesis. The requirements for this representation can be stated as follows:

1. It must be built from the OOT and as such, it must account for the noise of the OOT. The building of the surface representation should also be as efficient as possible, because during fast exploration, the OOT can grow very quickly.

2. It has to store a value for surfaces that symbolizes how well they have been inspected by the robot's cameras. Due to lighting, noise and other visual factors, the surface coverage value could theoretically be a real number, but for the purpose of this thesis, we only consider the surfaces to be either inspected (belonging to S_{insp}) or uninspected (belonging to S_{uni}).

3. It must provide information about surface normals so that we can only mark the surfaces that are being looked at at an acceptable as inspected. The normals will also be used for assigning to which parts of the free space the surfaces belong and for calculating surface coverage in a general area in chapter 5, where the sharing of surface coverage information is discussed.

4. It should allow for fast calculation of the surfaces that are visible by a camera at a given position $\mathbf{p}$ and rotation $\mathbf{q}$. The calculation must be fast for the reason that the UAV can move very quickly and therefore we must update the coverage value of visible surfaces at a high frequency. Fast querying of visible surfaces is also vital for any surface inspection methods that utilize viewpoint sampling and evaluation.

2.1 FacetMap

The FacetMap is a proposed surface mapping structure, built online from an OOT. The FacetMap represents the known surface as a set of small surface elements, called facets in this work. A facet ψ, in the context of this work, is defined by its center position $\mathbf{p}_\psi$, normal vector $\mathbf{n}_\psi$ and a value symbolizing the facet's coverage γ_ψ. More formally, we define a facet as a triplet:

$$\psi = (\mathbf{p}_\psi, \mathbf{n}_\psi, \gamma_\psi).$$

A single facet thus represents a small, approximately circular surface area with a radius of r_{facet} with its center at position $\mathbf{p}_\psi$ and a normal $\mathbf{n}_\psi$. In this work, we will assume that each robot is building its own FacetMap, which we denote as $\mathbb{S}$ and that each FacetMap holds a set of facets Ψ. For simplicity, we only consider two values of the coverage: $\gamma_\psi = 1$ symbolizing full coverage of a facet and $\gamma_\psi = 0$ symbolizing no coverage of the facet. Therefore, we can divide Ψ into a set of inspected facets Ψ_{insp} and uninspected facets Ψ_{uni}.

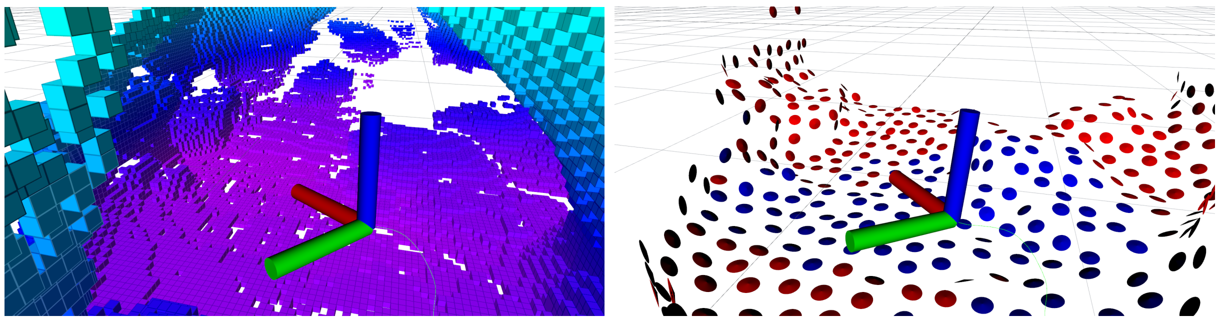
Figure 2.1: The occupancy octree (left) and the FacetMap created from it with inspected facets marked as blue and uninspected facets marked red (right)

To deal with the noise of the OOT and to also not have too granular data, the facets are generated at the positions of occupied voxels at a certain distance from any pre-existing facets. For simplicity, we set this distance to be the same as r_{facet}. Specifically in this work, since the highest voxel resolution of the OOT is $\rho = 0.2\,\mathrm{m}$, we set

$$r_{facet} = 1\,\mathrm{m}. \tag{2.1}$$

This way, even if there is small noise in the occupied voxels near a given facet, it will block new facets from being created near that position. Had we selected the value of r_{facet} too low, new uninspected facets could be created at the parts of the surface, where inspected facets are already present and that could cause the UAV to inspect some surfaces multiple times and any long term inspection planning would therefore be inefficient.

The obvious problem with this approach lies in thin surfaces that have free space on both sides. Ideally, we would want to generate facets on both sides of the surface with their normals in opposite directions. In the approach used in this work, however, the facets will usually not be generated at all, because the free space gradient of a very thin wall will usually be very small and, as described further in this chapter, we discard any possible facets for which $|\mathbf{n}_\psi|$ is below a certain threshold. Furthermore, even if the facets were generated on one side, they would block the facets on the other side from being generated due to the other facets being too close. This could be potentially solved by adding some additional conditions to the facet creation process, but for the purpose of this thesis, we assume that most walls and ceilings are sufficiently wide and do we not address this issue further.

2.2 Approximation of surface normals from the OOT

To determine the normal of a facet at position $\mathbf{p}$, the following method is used. The main idea of this method is to calculate a gradient vector $\vec{\nabla}\phi$ of free space from voxels of the OOT and if the magnitude of this vector is higher than some threshold m_{min}, normalize it and declare it the normal of the surface. Let us define a function $\phi(m)$ for a voxel m as

$$\phi(m) = \begin{cases} 1, & m \subset V_{free} \\ 0, & m \not\subset V_{free} \end{cases}$$

The surface normal is then calculated as an approximate gradient of the function ϕ in a bounding box with a side length of d_{fsg} (a user-defined parameter) centered around the point $\mathbf{x}$ at which the normal is to be calculated. In this calculation, we only assume the voxels at the highest resolution of the OOT (in this work having a side length of $\rho = 0.2\,\mathrm{m}$.

We first select all free voxels of the OOT that fall into the bounding box and denote this set of voxels as $M^{bbx}_{free}(\mathbf{x})$. We then iterate over each voxel $m \in M^{bbx}_{free}(\mathbf{x})$, look at its neighboring voxels $m' \in N_6(m)$ and if $\phi(m') = 0$, meaning that m' is either occupied or unknown, we add a normalized vector in the direction of free space to $\nabla\phi(\mathbf{x})$. Formally, we can write the gradient at position $\mathbf{x}$ as

$$\nabla\phi(\mathbf{x}) = \sum_{m \in M_{bbx}(\mathbf{x})} \sum_{m' \in N_6(m)} \frac{\mathbf{p}_m - \mathbf{p}_{m'}}{|\mathbf{p}_m - \mathbf{p}_{m'}|}(1 - \phi(m')). \tag{2.2}$$

After calculating this vector, we look at its magnitude. If $|\nabla\phi(\mathbf{x})|$ is below a certain chosen threshold, for example due to noisy data at that position, we do not generate a facet at the given position. Otherwise, we declare that the queried position $\mathbf{x}$ belongs to a surface, which at position $\mathbf{x}$ has a normal

$$\mathbf{n}_\psi = \frac{\nabla\phi(\mathbf{x})}{|\nabla\phi(\mathbf{x})|}. \tag{2.3}$$

2.3 Building the FacetMap

Because the OOT is constantly being updated around the robot, the surfaces of the FacetMap need to be periodically updated as well. The OOT only changes near the robot, and for this reason, we only recalculate facets in a bounding box D of a specified size, centered around the robot's position, and keep any facets that are outside of D. In this work, the FacetMap is updated by periodically calling the procedure **RecalculateFacets**() defined in algorithm Algorithm 1. This procedure simply finds all uninspected facets that lie in D, deletes them, and calculates new ones by sampling positions of occupied voxels at the maximum depth of the OOT that are r_{facet} far from any other inspected or uninspected facets. The inspected facets are kept in the FacetMap until the end of the mission and block any new uninspected facets from being generated in their vicinity.

2.4 Querying Visible Facets

A crucial method that is used for mapping inspected surfaces and for determining information gain of viewpoints is the method **VisibleFacets**$(\mathbf{p}, \mathbf{q})$ which tries to find the

Algorithm 1 The procedure used for recalculating uninspected facets in a bounding box D around the robot's position

Input D = Bounding box centered on the robot's position
Input r_{facet} = Facet generation blocking distance
Input m_{min} = Minimum magnitude of the free space gradient for generating a facet
Input/Output Ψ = Facets of the FacetMap

```
1: procedure RecalculateFacets($D$, $r_{facet}$, $m_{min}$, $\Psi$)
2:     $\Psi_{near} \leftarrow \{\psi \in \Psi | \mathbf{p}_\psi \in D\}$        ▷ Facets that lie in $D$
3:     $X_{occ} \leftarrow$ Center positions of all occupied voxels of the OOT in $D$
4:     for $\psi \in \Psi_{near}$ do
5:         if $\gamma_\psi = 0$ then
6:             $\Psi \leftarrow \Psi \setminus \psi$        ▷ erase uninspected facets in $D$
7:         end if
8:     end for
9:     for $\mathbf{x} \in X_{occ}$ do
10:        if $\forall\psi \in \Psi : |\mathbf{p}_\psi - \mathbf{x}| > r_{facet}$ then        ▷ if no facet is near
11:            $\vec{\nabla}\phi$ = FreeSpaceGradient($\mathbf{x}$)        ▷ calculate gradient
12:            if $|\vec{\nabla}\phi| \geq m_{min}$ then        ▷ if gradient is large enough
13:                Create new facet $\psi_{new} = (\mathbf{x}, \vec{\nabla}\phi/|\vec{\nabla}\phi|, 0)$
14:                $\Psi \leftarrow \Psi \cup \psi_{new}$        ▷ save new facet
15:            end if
16:        end if
17:    end for
18: end procedure
```

facets that are visible by the robot's cameras at position $\mathbf{p}$ and rotation $\mathbf{q}$. This is achieved by casting rays in the OOT from a given camera's position in the directions and to a distance specified by the camera's FoV rotated by $\mathbf{q}$. The OOT allows the raycasting to be performed efficiently and for each ray, it returns a position where the ray hit an occupied voxel. If the ray goes out of range or it hits an unknown voxel, it is discarded.

For each ray that hits an occupied voxel, we store the position of the hit voxel as $\mathbf{h}_i$ and the direction of the corresponding ray $\mathbf{d}_i$. For each ray, we then search for uninspected facets $\psi \in \Psi_{uni}$ near $\mathbf{h}_i$ and if the angle between $\mathbf{d}_i$ and $\mathbf{n}_\psi$ is above a certain threshold (meaning that the camera is looking at a facet ψ at an angle roughly perpendicular to the surface), we add the facet to the resulting set of visible facets Ψ_{vis}. After a chosen number of rays n has been cast, the procedure **VisibleFacets**$(\mathbf{p}, \mathbf{q})$ returns Ψ_{vis}.

The choice of n determines, how accurate we want the estimation of visible facets to be. This can be useful for evaluating viewpoints for surface inspection, as we can reduce n to increase computational speed at the cost of less precise estimation of the information value, if we need to sample many viewpoints. To speed up this procedure, which is used in the mapping and planning algorithms very often, we store facets within an octree with a resolution ρ_F lower than that of the OOT (specifically in this work, a resolution $\rho_F = 1.6\,\mathrm{m}$ is used). In the FacetMap octree, each leaf voxel holds a list of inspected and a list of uninspected facets. This allows much faster querying of facets near a position than if the facets were stored in a single list and were iterated over for each query.

2.5 Updating coverage

With the procedure **VisibleFacets**$(\mathbf{p}, \mathbf{q})$, mapping which surfaces have been inspected is simple. At a high rate, we call the procedure **VisibleFacets**$(\mathbf{p}, \mathbf{q})$ at the current robot's position and rotation and thus obtain a set of visible facets Ψ_{vis}. For each visible facet $\psi \in \Psi_{vis}$, we then set the coverage value γ_ψ to 1 and move it from Ψ_{uni} to Ψ_{ins}.

2.6 Computing viewpoint surface inspection value

With the procedure **VisibleFacets**$(\mathbf{p}, \mathbf{q})$, we also define a surface inspection information value of a given viewpoint ξ. We define this value as

$$\mathbf{I}_S(\xi) = n_{visible}, \tag{2.4}$$

where $n_{visible}$ is the amount of facets returned by the **VisibleFacets**$(\mathbf{p}, \mathbf{q})$ procedure at the position $\mathbf{p}$ and orientation $\mathbf{q}$ defined by ξ.

2.7 Generating Surface Inspection Viewpoints

The strategies for surface inspection presented in this work are sampling based and require a method for efficiently querying informative and safe viewpoints in a given area. For this reason, we define a procedure **GenerateSurfaceViewpoints**$(D, full)$, which finds interest points for facets that lie in D, samples viewpoints near these points of interest and evaluates found safe viewpoints. The method has two variants - a full calculation (applied when the argument $full = true$) for computing a rich set of viewpoints and a reduced calculation (if $full = false$) for faster computation of a sparse set of viewpoints.

The method can be described as follows:

1. From each uninspected facet $\psi_{uni} \in \Psi_{uni}$ we project a point $\mathbf{x}_p$ as
$$\mathbf{x}_p = \mathbf{p}_{\psi_{uini}} + d_{proj}\mathbf{n}_{\psi_{uni}} \tag{2.5}$$
where d_{proj} is a user-defined constant. We thus obtain a set of projected points $\mathbf{x}_p \in X_p$ which we then downsample by a distance ϵ (so that no two points are closer than ϵ from one another) and thus obtain a downsampled set of projected points X_{ds}.

2. Because we do not always want to inspect every individual facet, we filter out the projected points $\mathbf{x}_{ds}$ for which the number of the nearby originally projected points $\mathbf{x}_p \in X_p$ (the number of points $\mathbf{x}_p \in U(\mathbf{x}_{ds}, \epsilon_2)$, where $U(\mathbf{x}_{ds}, \epsilon_2)$ is the ϵ_2 neighborhood of point $\mathbf{x}_{ds}$ and ϵ_2 is a user-defined distance) is lower than a user-defined threshold ϵ_{int}. We thus obtain a set of interest points X_{int} in areas with a large amount of uninspected facets.

3. For each interest point $\mathbf{x}_{int} \in X_{int}$ we uniformly sample a defined number of points in $U(\mathbf{x}_{int}, \epsilon_2)$ and headings $\varphi \in < 0, 2\pi)$ and thus obtain a set of sampled viewpoints $\xi \in X_{sampled}$ and proceed to test the viewpoints.

4. We first test each point for whether it is in safe space, meaning that the distance from the nearest unexplored or occupied voxel of $\mathbf{x}_\xi$ is greater than d_{safe}, where d_{safe} is the minimal safe distance of the UAV from obstacles, defined at mission start.

5. Secondly, we test each viewpoint for whether it falls into a segment σ of the robot's SegMap (described in section 3.2). We discard viewpoints that do not belong to any segment of the SegMap. Furthermore, if the reduced calculation of this method is selected, we also discard any viewpoint that falls into a segment σ for which we have already found at least one acceptable viewpoint. This can speed up the calculation drastically.

6. Thirdly, we test each viewpoint for whether its surface inspection information value $\mathbf{I}_S(\xi)$ is sufficiently large. If
$$\mathbf{I}_S(\xi) < \epsilon_{infoval}, \tag{2.6}$$

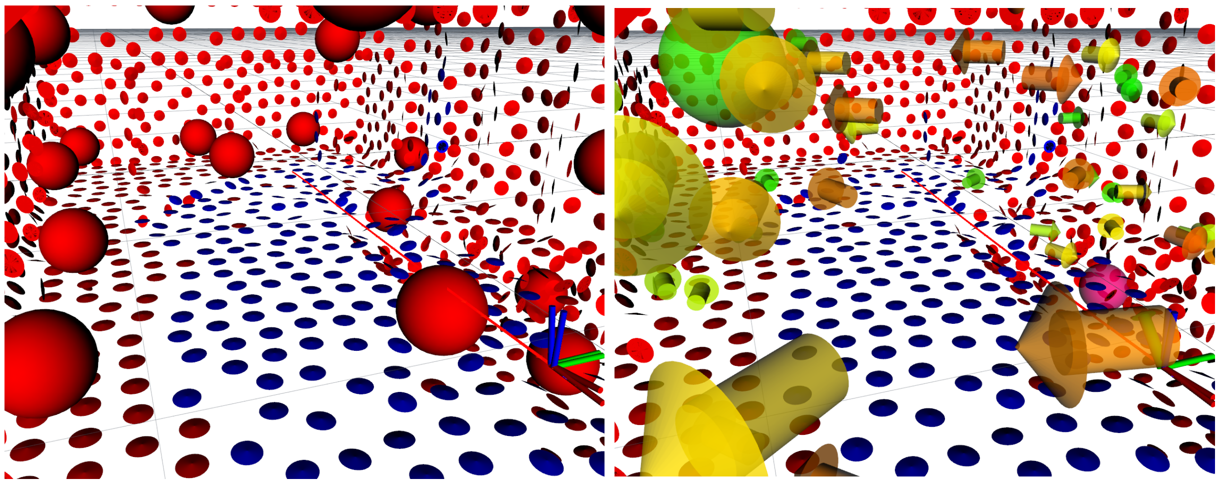

Figure 2.2: An illustration of the **GenerateSurfaceViewpoints**() procedure. On the left image, interest points (red balls) are first projected from uninspected facets and filtered. A rich set of viewpoints (colored arrows) are then generated in the vicinity of the interest points (right).

where $\epsilon_{infoval}$ is a user-defined minimal information value of the generated viewpoints, we discard it.

7. Lastly, we test each remaining viewpoint for whether it is reachable from the UAV's position. Because we required each viewpoint to belong to a segment, determining reachability is very fast, because we can use the A* navigation over the topological graph of the SegMap (described in section 3.3). We discard all viewpoints that fall into currently unreachable segments.

8. The calculation of viable viewpoints for a given interest point $\mathbf{x}_{int}$ is stopped preemptively, if enough viable viewpoints are found for the given interest point. In the full calculation variant of the algorithm, the sufficient number of viewpoints for an interest point can be defined. In the reduced calculation, we stop the sampling for $\mathbf{x}_{int}$ after finding just one safe, informative, and reachable viewpoint.

Chapter 3

Topology Mapping

Contents

This chapter focuses on the design of a topological representation of the environment for fast navigation, planning, and lightweight sharing of key features of the environment and mission-important information. The requirements for the environment representation to be used by autonomous search strategies considered in this thesis can be summarized as follows.

1. Fast and incremental building - Because we want to use the representation for planning and sharing during the mission, the representation must be able to be built onboard with the limited resources available. A single update iteration must also be fairly quick so as not to block any planning algorithms that use the representation.

2. Reconstruction of shape and connectivity - The most important aspect for autonomous multi-robot search strategies is to be able to share information about the environment (i.e., which parts of the environment are completely explored, which parts have all surfaces covered, which parts are dangerous etc.). Since the communication between robots in subterranean environments is usually severely limited, the representation must be very lightweight. This means that there should exist a way to compress the representation information into a very lightweight bundle, which the other robots can then rebuild into precise information about the environment.

3. Quick navigation - The representation should allow for quickly determining collision-free paths and approximating travel distances in both the local representation and in

the one built from the information from other robots. For the local representation, we can use this to quickly compute reachability and approximate distances to points of interest, which is useful for any kind of long-distance planning.

3.1 TopoMap

In [22], the authors present the TopoMap — a topological mapping structure that fills the explored space with convex clusters, finds connections between these clusters (on the boundaries of these clusters), and then creates a connectivity graph from these clusters and their connections which can easily be used for fast path planning. The main advantage of the TopoMap is, however, its shareability. We can easily approximate convex clusters with polygons, boxes, ellipsoids, or spheres, all of which create a different amount of uncertainty about whether a given point truly belongs to a given segment in the original TopoMap.

One disadvantage of the original TopoMap is that the authors only mention building the TopoMap offline after a robot has explored the environment. Another disadvantage is that the original pathfinding algorithm only allows path planning between points that both lie in a segment, which may not always be the case for an incrementally built version of the map. The convex merging and growing also uses the Quickhull [33] algorithm, which could be computationally expensive for on-board online map building.

For these reasons, an incrementally built, faster, and more rough variant of the TopoMap – the SegMap – was designed in this thesis and is described further.

3.2 SegMap - an online TopoMap

The SegMap is a proposed incrementally built shareable environment representation based on the principles of the TopoMap. Its main features are identical to those of the TopoMap — it is composed of segments $\sigma \in S$ which are connected by portals $\mathbf{p}_{ij} \in P$. Each segment σ represents a set of free space voxels $m \in M_\sigma, m \subset V_{free}$. And each portal represents a position $\mathbf{p}_{ij}$ at which a robot can safely pass from σ_i to σ_j. We also define $\mathbf{x}_m$ as the center point of a given voxel m and $N_6(m)$ as the 6-neighborhood of voxels adjacent to m.

The voxel resolution at which the SegMap operates is an important factor to consider. Building the SegMap at a lower resolution with very large voxels may be considerably faster, but since the cluster expansion algorithm only considers voxels that contain no occupied or unknown space, it may cause that segments will not generate in very narrow passages. Building at a higher resolution would solve this issue completely but would also bring high computation and memory requirements. All UAVs considered in this work are roughly 0.8 meter in size, and as such, the voxel resolution $r = 0.8\,\mathrm{m}$ was chosen. Identically to the TopoMap, the segments of the SegMap are also grown and merged in a convex, compact

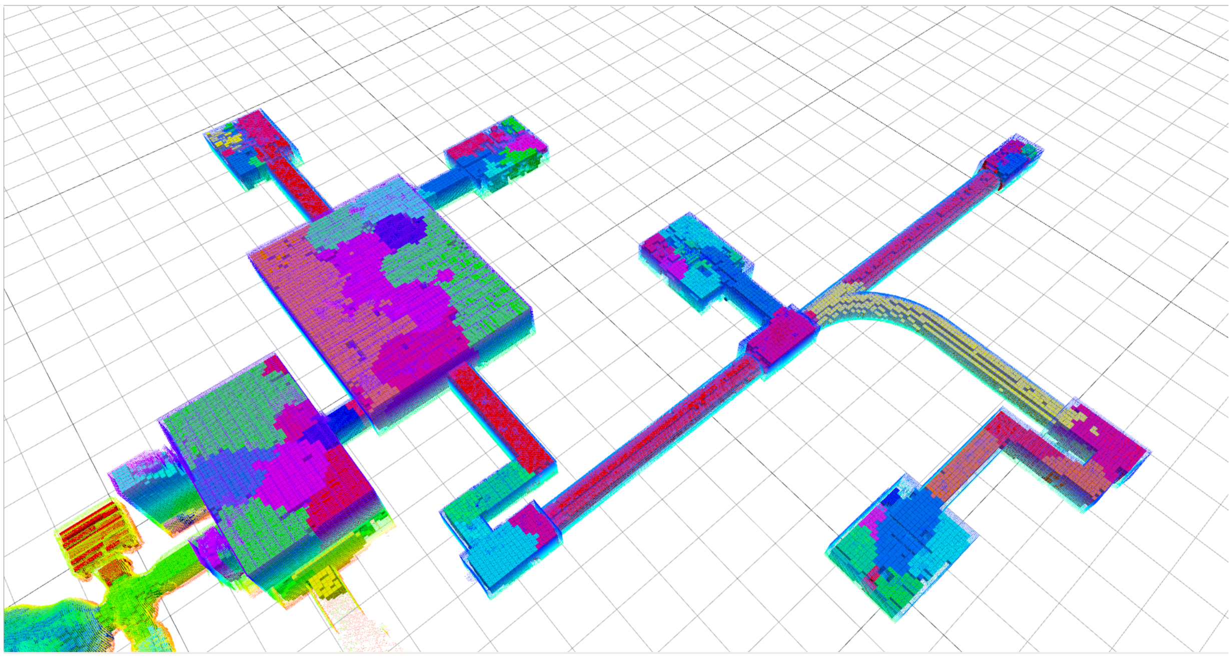

Figure 3.1: An example of a SegMap created in the finals world during an exploration run

manner. Some changes have, however, been made to the growing and merging algorithms to make them faster and more suitable for iterative onboard computation.

3.2.1 Growing Segments

The approach to growing segments used in this work and described in Algorithm 2 is very similar to the one described in [22]. Two main changes have been made to trade precision for faster computation. Secondly, the authors state that they accept candidate voxels into the segment, if rays from the voxel to all other segment voxels are obstacle-free. This can be very computationally demanding, and as such, we cast rays from the candidate voxel only to the set of downsampled points C_{ds} and discard the voxel if any of the rays passes through V_{unk} or V_{occ}, as shown on line 22 of Algorithm 2.

The expansion process is governed by three parameters that can be tuned according to specific requirements for computational speed and the overall SegMap building strategy. These parameters are:

- δ_{comp} - The compactness threshold. To ensure spherical cluster growth, Pricipal Component Analysis (PCA) is calculated for border voxels of the segment at each step. Then the maximum border voxel distance h_{max} from the segment center c_i in the direction of the least significant axis v_3 of the PCA is computed. To force the segment to expand in the direction of its least significant axis, we only add voxels m that

Algorithm 2 Function used to expand a given segment σ_i

Input δ_{comp} = Compactness factor
Input d_{ds} = Downsampling distance
Input s_{max} = Maximum segment size
Input/Output σ_i = The segment with border voxels $M^{\sigma_i}_{border}$ that is supposed to be expanded

```
1:  function ExpandSegment($\delta_{comp}$, $d_{filter}$, $s_{max}$,$\sigma_i$)
2:      $M_F \leftarrow M^{\sigma_i}_{border}$
3:      while $M_F \neq \emptyset$ do
4:          $C_F \leftarrow$ centers of voxels $M_F$
5:          $C_{ds} \leftarrow$ points from $C_F$ downsampled by distance $d_{ds}$
6:          $\mathbf{x}_c \leftarrow$ center position of $C_{ds}$
7:          $v_1, v_2, v_3 \leftarrow$ CalculatePrincipalAxes($C_{ds}$)
8:          $l_{max} \leftarrow \max_{p \in C_F} |p \cdot v_3|$
9:          $M_{test} \leftarrow \bigcup_{m \in M_F} N_6(m)$
10:         $M_{NF} \leftarrow \emptyset$
11:         for $m \in M_{test}$ do
12:             if $\exists \sigma_j : m \in M^{\sigma_j}$ then        ▷ do not add voxels from other segments
13:                 continue
14:             end if
15:             if $m \not\subset V_{free}$ then        ▷ do not add non-free voxels
16:                 continue
17:             end if
18:             if $|\mathbf{x}_m - \mathbf{x}_c| > max(l_{max} + \delta_{comp}, s_{max})$ then        ▷ maintain compactness
19:                 continue
20:             end if
21:             Cast rays $r \in R$ from $\forall p \in C_{ds}$ to $x_m$ in the OOT
22:             if $\exists r \in R : r \cap (V_{unk} \cup V_{occ}) \neq \emptyset$ then        ▷ maintain convexity
23:                 continue
24:             end if
25:             $M_{NF} \leftarrow M_{NF} \cup m$        ▷ add the voxel to frontier voxels
26:             $M^{\sigma_i} \leftarrow M^{\sigma_i} \cup m$        ▷ add the voxel to voxels of $\sigma_i$
27:         end for
28:         $M_F \leftarrow M_{NF}$
29:     end while
30: end function
```

sastisfy

$$|\mathbf{x}_m - \mathbf{x}_c| < h_{max} + \delta_{comp} \tag{3.1}$$

where $\mathbf{x}_c$ is the center of the segment at the particular step of the expansion process.

- d_{ds} - The downsampling distance. This parameter determines how far apart the downsampled points for convexity and compactness computation will be. A large value of d_{ds} will cause fewer rays to be computed, which leads to faster computation at the cost of less convex growth.

- s_{max} - The maximum size of a grown segment. A higher value will result in less merging being done, but may cause more unnatural overall segmentation than if smaller segments were grown and merged, as stated by the authors in [22].

3.2.2 Connecting Segments

Algorithm 3 Function used to find the currently safest portal positions for a given segment σ_i. This function uses the function **ObstacleDist**($\mathbf{x}$) of the OOT which returns the distance from $\mathbf{x}$ to the nearest occupied or unknown voxel of a given point.

Input σ_i = The segment with border voxels $M^{\sigma_i}_{border}$ for which the connections are supposed to be recalculated

Output P_{σ_i} = Set of portals that connect σ_i to other segments

```
1: function GetSafestPortals($\sigma_i$)
2:     for $\sigma_j \in \mathbb{G} \setminus \sigma_i$ do
3:         $X_{ij} \leftarrow \emptyset$    ▷ initialize a set of possible portal positions for every other segment
4:     end for
5:     for $m \in M^{\sigma_i}_{border}$ do
6:         for $m' \in N_6(m)$ do    ▷ look at neighbor voxels of border voxels of $\sigma_i$
7:             if $\exists \sigma_j \in S : m' \in \sigma_j$ then
8:                 $X_{ij} \leftarrow X_{ij} \cup \frac{x_m + x_{m'}}{2}$
9:             end if
10:        end for
11:    end for
12:    for $\sigma_j \in S$ do
13:        if $X_{ij} \neq \emptyset$ then
14:            $\mathbf{p}'_{ij} \leftarrow \arg\max_{\mathbf{x} \in X_{ij}}$ ObstacleDist($\mathbf{x}$)    ▷ safest portal of $\sigma_i$ and $\sigma_j$
15:            $P_{\sigma_i} \leftarrow P_{\sigma_i} \cup \mathbf{p}'_{ij}$
16:        end if
17:    end for
18: end function
```

Finding portals between segments is fairly straightforward. In [22], the authors define portals as areas where segments are adjacent and then navigate through the centers of these areas. This is a good approach for when the segments are almost perfectly convex, but this may not be the case in the rougher SegMap where the centers of these areas might lie in unsafe space. For this reason, we store a portal as the safest position (with the maximum distance from unknown and occupied space) at which there are adjacent voxels of two given segments and recalculate these positions whenever one of the connected segments changes using the function **GetSafestPortals**() shown in Algorithm 3.

3.2.3 Merging Segments

Algorithm 4 Function that tries to merge segments σ_i and σ_j. If the resulting merged segment is convex, segment σ_j is merged into σ_i

Input d_{ds} = Downsampling distance
Input c_{convex} = Convexity threshold
Input/Output σ_i, σ_j = The segments with border voxels $M^{\sigma_i}_{border}$ and $M^{\sigma_j}_{border}$ respectively, that are attempted to be merged

```
1:  function TryMergingSegments($d_{ds}$, $\sigma_i$, $\sigma_j$)
2:      $M_{CB} \leftarrow \emptyset$                                  ▷ initialize set of combined border voxels
3:      for $m \in M^{\sigma_i}_{border} \cup M^{\sigma_j}_{border}$ do
4:          for $m' \in N_6(m)$ do
5:              if $\exists m' \in N_6(m) : m' \notin M^{\sigma_i} \cup M^{\sigma_j}$ then        ▷ if voxel is border
6:                  $M_{BC} \leftarrow M_{BC} \cup \mathbf{x}_{m'}$            ▷ add it to the combined border voxel set
7:              end if
8:          end for
9:      end for
10:     $X_{BC} \leftarrow$ center positions of $X_{BC}$
11:     $X_{ds} \leftarrow$ points from $X_{BC}$ downsampled by distance $d_{ds}$
12:     Cast rays between $\forall x_a, x_b \in X_{ds}$ in the OOT
13:     $y \leftarrow$ percentage of rays that intersected $V_{occ} \cup V_{unk}$
14:     if $y > c_{convex}$ then
15:         return
16:     end if
17:     $M^{\sigma_i} \leftarrow M^{\sigma_i} \cup M^{\sigma_j}$
18:     $S \leftarrow S \setminus \sigma_j$
19: end function
```

Similarly to the expansion algorithm, Algorithm 4 has also been modified to enable for faster and iterative calculation. Its main difference from the original algorithm is that instead of using the Quickhull algorithm, a similar simplification as with the expansion algorithm is used. The combined border points X_{BC} of the segments are downsampled according to a pre-defined distance d_{ds} and then rays are cast in the OOT between each pair of $\mathbf{x}_a, \mathbf{x}_b \in X_{BC}$. The segments are then merged if the percentage of rays that intersected non-free space is below a defined threshold c_{convex}.

3.2.4 SegMap Build Loop

There are many possible ways to iteratively build the SegMap with the above defined functions for expanding, connecting, and merging segments. The approach chosen for this work is to periodically try to grow, connect, and merge some randomly selected segments in a bounding box B of a defined size around the robot.

The number of merges and expansions is specified by user-defined parameters N_{grown}, N_{expand}, and N_{merge} through which we can control the approximate time taken by one iteration. A single iteration of the update loop is defined as follows:

1. If the robot is currently in a voxel m that does not belong to any segment, initialize a new segment σ_i with $M^{\sigma_i} = \{m\}$ and expand it using **ExpandSegment**() (see Algorithm 2).

2. Cast N_{rays} from the robot's position in random directions through free space. If a ray encounters a voxel m that does not belong to a segment, then initialize a new segment at that voxel and expand it with **ExpandSegment**() . Stop the rays upon leaving V_{free} or B. The maximum number of segments grown in this this manner in a single iteration is limited to N_{grown}.

3. Randomly select N_{expand} segments inside B and then expand each selected segment using **ExpandSegment**().

4. Randomly select N_{merge} portals in B and try to merge the segments connected by these portals using **TryMergingSegments**() (see Algorithm 4). Only try merging the segments if at least one segment has changed since the last merge attempt.

After each expansion or successful merge, the safest portals for the changed segments are also recalculated using the function **UpdateConnections**() (see Algorithm 3). This process is also illustrated in Figure 3.4.

3.3 Navigation in the SegMap

One of the important reasons for the use of the SegMap is that it can be used to quickly plan paths across the segments. The path planning algorithm is analogous to the algorithm used in [22] with one difference — the authors only allow path planning from

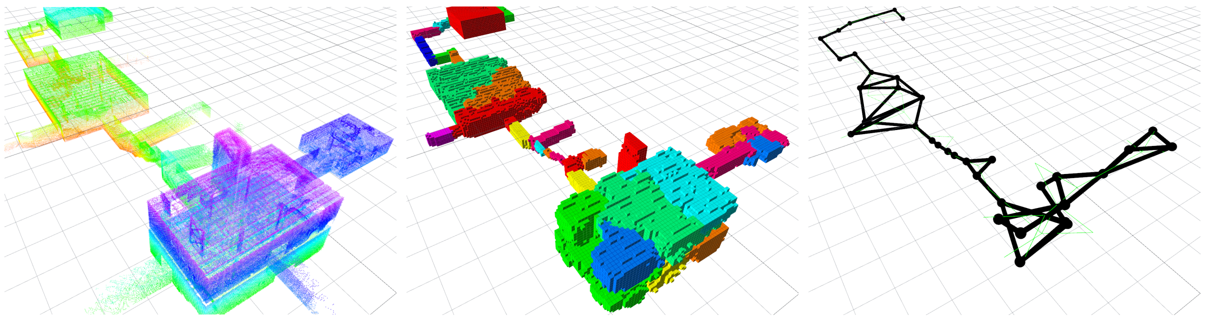

Figure 3.2: An OOT of a UAV in an urban environment (left), the SegMap created from the OOT during the flight (middle) and the navigation graph created by the SegMap (right)

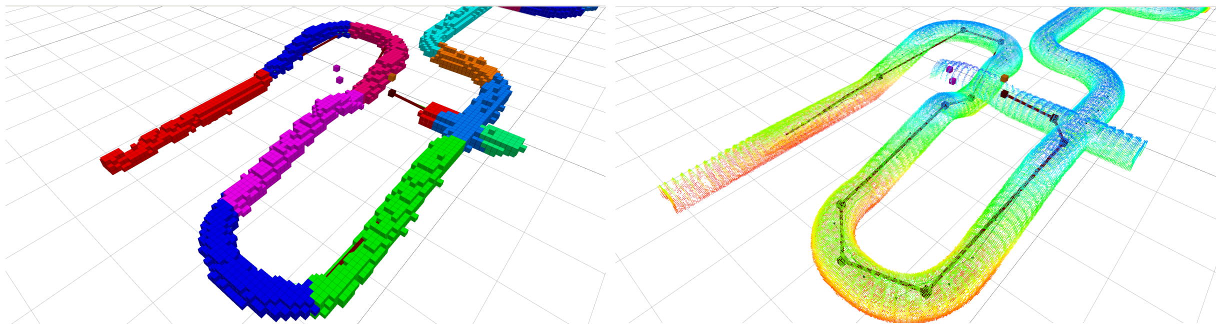

Figure 3.3: A SegMap created during the flight of a UAV that has reached a dead end (left) and the generated path across portal positions of the SegMap that leads the UAV to a frontier in another branch of the tunnel (right).

point $\mathbf{p}_a$ to $\mathbf{p}_b$ if both $\mathbf{p}_a$ and $\mathbf{p}_b$ lie in some segment. Because the update loop of the SegMap does not always fill the entire free space and navigation to a point outside a segment might be required, this condition must be relaxed.

Therefore, one step is added to the beginning of the algorithm, which tries to assign a given point $\mathbf{p}$ to some segment by casting rays in the OOT in random directions and checking whether the ray hits a segment. If at least one ray hits some segment σ_i, a collision-free path planner tries to plan a path from $\mathbf{p}$ to some point inside σ_i. If a path exists, $\mathbf{p}$ is assigned to the given segment and the algorithm proceeds to plan an A* path across the dual navigation graph of the SegMap. The navigation over segment portals to a point outside of a segment is shown in Figure 3.3.

We also define a function $\mathbf{D}(\mathbf{x}, \mathbf{y})$ which tries to find a path between points $\mathbf{x}$ and $\mathbf{y}$ in the SegMap and if such a path is found, it returns the sum of euclidian distances between the individual portals of the path.

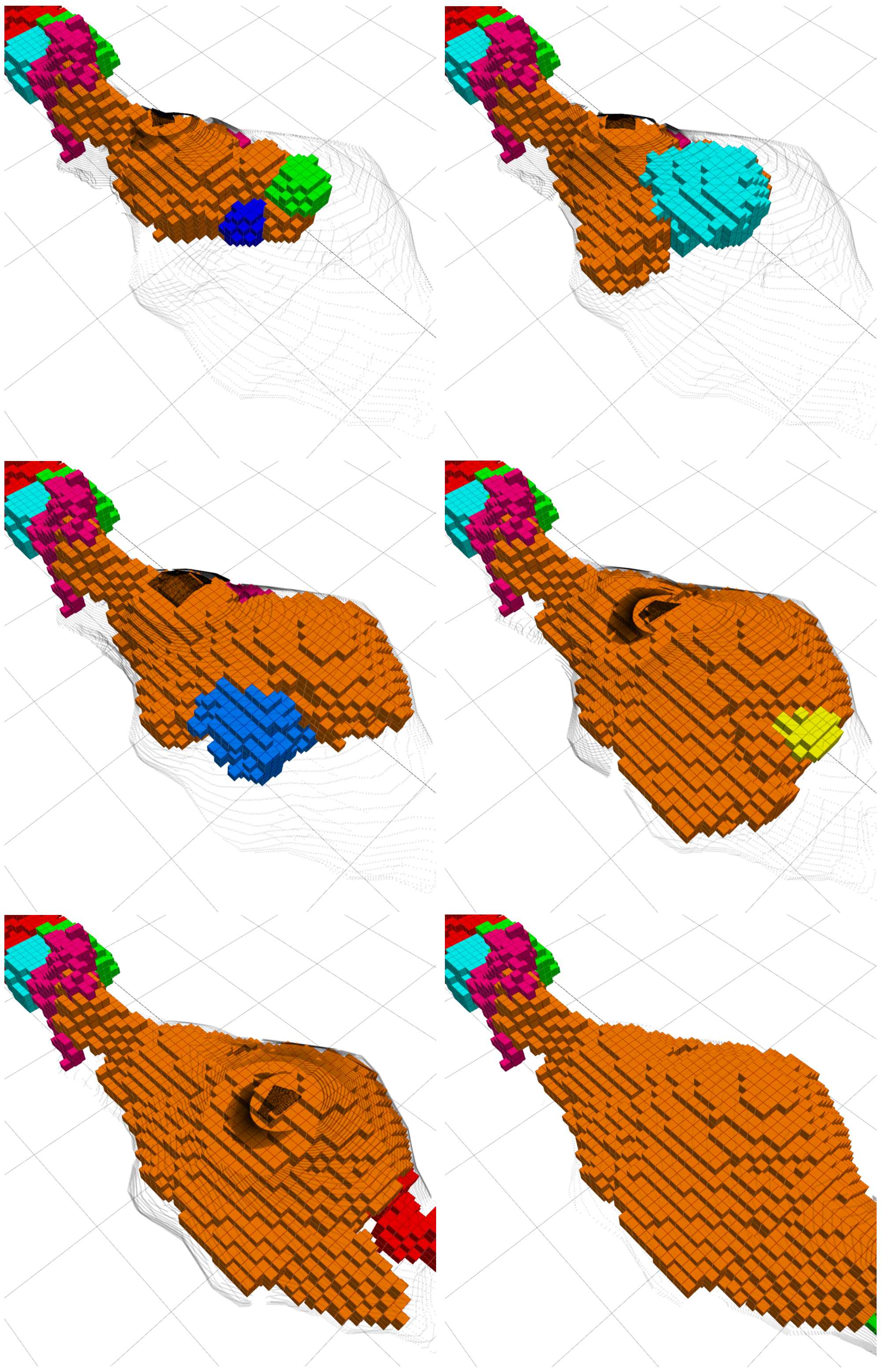
Figure 3.4: Illustration of the SegMap iterative building process. New segments are being created, expanded and then merged into the large orange cluster in a large convex cavern.

Chapter 4

Frontier Mapping

Contents

Most robotic exploration methods are based on either sampling viewpoints and computing their information value (approximate amount of unknown space V_{unk} that would be uncovered at that viewpoint with the given sensor configuration) [13], or on finding frontiers (defined in [17] as the boundary between V_{free} and V_{unk}) in the environment and then sampling viewpoints near these frontiers and navigating to them [14].

Since the environments for subterranean SAR can be large, sampling viewpoints across the whole environment could be costly, and therefore the approach using frontiers was chosen for this thesis. This chapter explains the process used in this thesis for the detection and clustering of frontiers and generating viewpoints that explore these frontiers.

4.1 FrontierMap

For a given occupancy representation, the frontier is defined as the boundary between free space V_{free} and unknown space V_{unk}. Specifically for the binary OOT, the frontier can be defined as the boundary between free voxels and occupied voxels. We define a frontier element point

$$\mathbf{f} = \frac{\mathbf{p}_{m_{free}} + \mathbf{p}_{m_{occ}}}{2} \tag{4.1}$$

as the position between a free voxel m_{free} and an occupied voxel m_{occ} and then construct the frontier representation based on these points. However, since frontier element points can be created in the OOT due to noise (e.g., free voxels being created behind a wall of occupied voxels or rays of voxels being created due to LiDAR anomalies) and since exploration of these frontiers is generally unwanted, some authors also utilize clustering methods to determine which frontiers are worth exploring [14].

In this thesis, we first cluster the frontier element points, label these clusters based on their size as either informative or as noise, and store information about these clusters in a structure $\mathbb{F}$, called a FrontierMap. The information about these clusters allows us to then sample viewpoints only in the vicinity of these clusters. For the purpose of this thesis the FrontierMap $\mathbb{F}$ is recalculated whenever the planning or map sharing algorithms need to generate frontier viewpoints, either for the whole map or for a given bounding box.

4.2 Frontier Detection and Clustering

The first step of the FrontierMap computation is to detect frontier element points in the OOT. To do this, we iterate over free-space voxels in the OOT and check whether they are adjacent to unknown space and if so, create frontier element points $\mathbf{f}$ according to the definition in (4.1). For this iteration, we utilize the important advantage of the OOT that it stores voxels in a tree, and therfore the only free-space voxels at the deepest level of the OOT are the ones near occupied voxels or near unknown space, since in large areas of free space, the voxels are merged into larger voxels.

We therefore iterate over the free-space voxels at the deepest level of the OOT and generate a set of frontier element points F_{ep}. Since the resolution of the OOT in this thesis is 0.2 m, this information is very granular, and for this reason, we downsample these points into a set of points $\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_N \in F_{ds}$ in which no points are less than 4 m apart.

After downsampling, we begin to cluster the points $\mathbf{f}_{ds} \in F_{ds}$. The clustering algorithm chosen in this approach is a modified version of the DBSCAN [34] algorithm because it does not require an apriori knowledge of the number of clusters, such as in K-means clustering. The original DBSCAN algorithm requires 2 pre-set values — N_D and ε_D and based on these values looks at the neighborhood

$$U_\varepsilon(\mathbf{f}) = \{\mathbf{f}_i \in F_{ds} | |\mathbf{f} - \mathbf{f}_i| < \varepsilon\} \tag{4.2}$$

of each downsampled point and classifies each point $\mathbf{f} \in F_{ds}$ as either a *core point* – if there is more points than N_D in $U_\varepsilon(\mathbf{f})$ or as a *non-core point*, if there is fewer points than N_D in $U_\varepsilon(\mathbf{f})$.

The most computationally demanding part of DBSCAN is querying the number of points in a given ε area around a given point which is done for every point. Ideally, we would want to label the points as *core points* or *non-core points* based on the number of frontier element points in $U_\varepsilon(\mathbf{f})$, but that would mean iterating over too many points.

For this reason, during the downsampling, we also store the number n_i of frontier element points that were downsampled into each point $\mathbf{f}_i \in F_{ds}$ and then label the points as *core points* if

$$\sum_{\mathbf{f}_i \in U_\varepsilon(\mathbf{f})} n_i \geq N_D, \tag{4.3}$$

and as *non-core points* otherwise.

Lastly, we perform the final step of the DBSCAN algorithm in the same way as in the original algorithm, which assigns each *core point* to a cluster, each *non-core point* to a cluster if there exists a cluster point in its ε neighborhood and all other points as *noise*, which is discarded. Thus, we obtain a set of clusters of points $F_1, F_2, F_3, ...F_N \in \mathbb{F}$ that make up the FrontierMap. A visualization of the resulting clustered points can be seen in Figure 4.1.

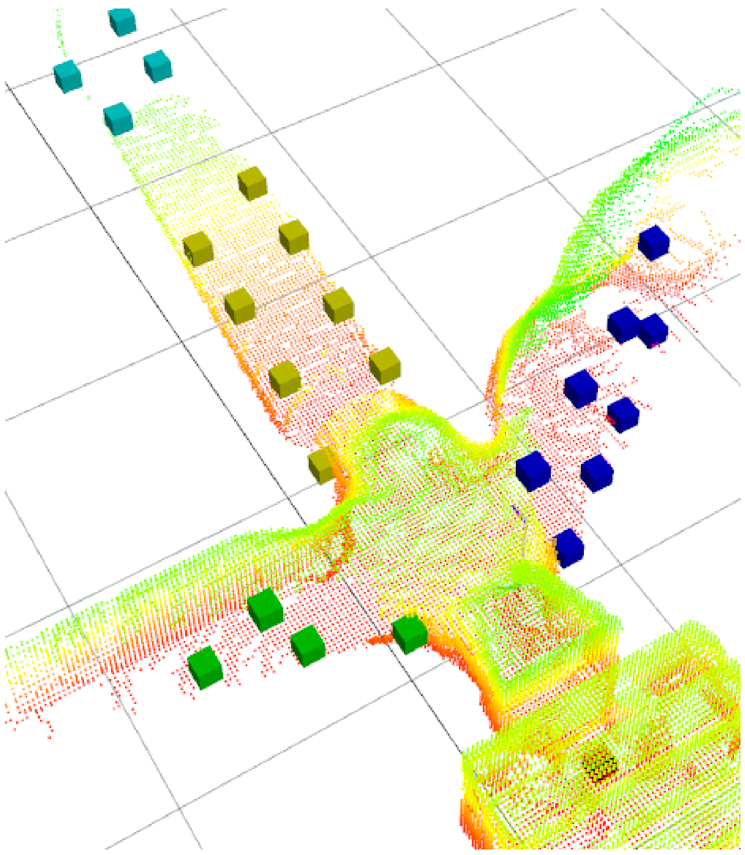

Figure 4.1: A visualization of the result of the modified DBSCAN clustering algorithm. Each colored cube represents a clustered point. Each set of cubes with the same color represents a separate cluster.

4.3 Computing Viewpoint Frontier Information Value

For the purpose of evaluating whether a given viewpoint ξ will explore frontiers, we could define the frontier exploration value of a viewpoint ξ as the number of frontier exploration points that are visible with the given sensor configuration of the UAV from the viewpoint ξ. Because we, however, do not store frontier element points, we define the frontier exploration value $\mathbf{I}_F(\xi)$ of a viewpoint ξ as

$$\mathbf{I}_F(\xi) = \frac{n_{unk}}{n}, \tag{4.4}$$

where n is a user-defined number of rays that are cast in the OOT in random directions in the FoV of the robot's mapping sensors and n_{unk} is the amount of these rays that hit an unknown voxel before hitting any occupied voxel.

4.4 Generating Frontier Exploration Viewpoints

Generally, not all parts of the frontier can be explored by a given robot robot with a given sensor configuration, due to the fact that a safe robot configuration which clears the frontier with mapping sensors might not exist. For this reason, we cannot naively send the robot to any frontier cluster, but we need to first determine if there are some safe viewpoints with high information value near these clusters. Since we assume the mapping sensor configuration of the UAV to be roughly symmetrical along the Z axis, we do not need to determine headings, and can simplify this task to finding informative, safe, reachable *positions* near each cluster and the terms *viewpoint* and *point* are used interchangeably in the rest of this section.

For this reason, we define a procedure **GenerateFrontierViewpoints**$(D, full)$. This procedure, similarly as in Section 2.7, can be run in its full computation variant, which computes a rich set of viewpoints (for local goal selection) or in its reduced variant, which only samples viewpoints for cluster points $\mathbf{f} \in F_i$ in a single cluster if they are far away from each other or no viewpoint has yet been found for the cluster (for long-distance goal selection). The procedure is illustrated on Fig. 4.2 and works thus:

1. For each point $\mathbf{f}$ that belongs to a cluster, we sample a user-defined number N of points $\mathbf{x} \in X$ uniformly in a sphere around $\mathbf{f}$ and begin to test them for safety, information value and reachability.
2. We first test each point for whether it is in safe space, meaning that the distance of $\mathbf{x}$ from the nearest unexplored or occupied voxel is greater than d_{safe}, where d_{safe} is the minimal safe distance of the UAV from obstacles, defined at mission start.
3. Secondly, we test each point for whether it falls into a segment σ of the robot's SegMap (described in Section 3.2). If the tested point does not fall into the space of

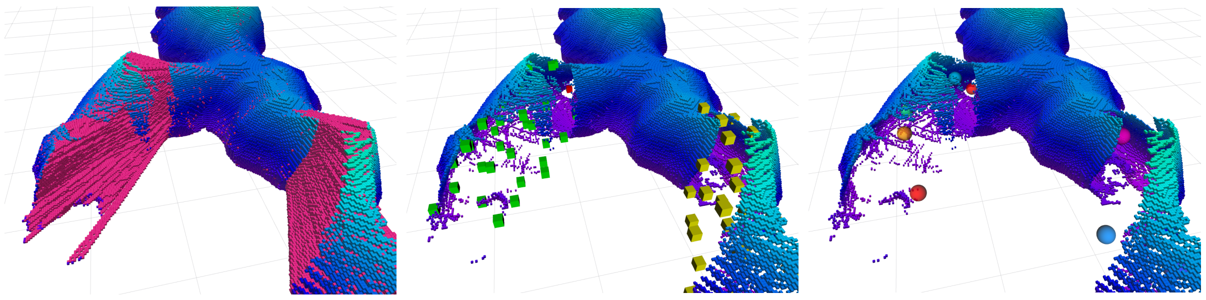
Figure 4.2: Visualization of the frontier viewpoint generation. First, the free space voxels, that are exposed to unknown space are found in the OOT (left). Then they are down-sampled and clustered using DBSCAN (middle). Finally, viewpoints that explore those frontiers are found in nearby free space (right).

any segment, we also cast rays from $\mathbf{x}$ and try to determine if any segment is at least visible from $\mathbf{x}$, and if so, assign $\mathbf{x}$ to it. We discard points that are not assigned to any segment of the SegMap.

4. Thirdly, we test the frontier information value $\mathbf{I}_F(\mathbf{x})$ of the sampled point. If

$$\mathbf{I}_F(\mathbf{x}) < \varepsilon_{infoval}, \tag{4.5}$$

where $\varepsilon_{infoval}$ is a user-defined minimal frontier information value of the generated points, we discard the point $\mathbf{x}$.

5. Lastly, we test whether the point is reachable from the UAV's position. Because we required each point to be assigned to a segment, determining reachability is fast, because we can use the A* navigation over the topological graph of the SegMap (described in Section 3.3). We discard all points that fall into currently unreachable segments.

6. The computation of viable viewpoints for a given frontier cluster point $\mathbf{f}$ stops after at least one safe, informative, reachable viewpoint has been found for $\mathbf{f}$. Furthermore, in the full computation variant of this procedure, if we find a viewpoint for a given point $\mathbf{f} \in F_i$, we also do not generate new viewpoints for other frontier points in F_i that are closer than 10 m from $\mathbf{f}$.

Chapter 5

Map Sharing

Contents

The possible methods for extracting mission-important information from a robot's local maps to share among robots are potentially endless. Deciding which information gets to be shared depends heavily on the given mission goal and communication constraints. For the task specified in this thesis, the following requirements for the sharing process were made:

- At least rough information about free, occupied and unknown space should be shared, because the robots should be able to estimate if they are currently in space that has already been discovered by other robots or if they are in completely new space. Connectivity information should also be shared, so as to enable other robots to find out for example how far a given point in the received map of another robot is from goal positions in the received map.
- Information about frontiers should be sent in some form so that the receiving robot can discern not only whether a given frontier that it creates in its own OOT belongs to the space already explored by other robots (which we call *globally explored space* in this work), but also to determine if the given frontier will lead to globally unexplored space, which should be represented by frontiers in the shared maps.
- Surface coverage should also be shared, so that strategies that focus on completely inspecting all surfaces in a given area can ignore areas that have been inspected by other robots, which we call *globally inspected areas* in this work.

The simplest solution would be to share the whole OOT and FacetMap of each robot, as that would allow the receiving robots to precisely compute frontier and surface inspection goals in the received maps and even attempt to merge these maps with their local maps. This would, however, be excessively demanding for the given bandwidth as the OOTs and FacetMaps can become quite large for bigger environments. On the other hand, sharing minimalistic information, such as executed trajectory, positions of frontiers and some clustered surfaces, could solve the issue of bandwidth, but merging information from two robots could be difficult (for example if a robot receives two maps, how could we decide whether a frontier in the first map has already been explored in the second map?).

The SegMap can prove to be very useful for the above mentioned requirements. We can utilize the convexity of the segments, that the authors of Topomap [22] used for quick path planning, to also send spatial information. Since the free-space segments are roughly convex, we can approximate their shapes with convex geometric shapes such as spheres, ellipsoids, bounding boxes or polygons, and then share these shapes. We can also easily share connectivity of these shapes simply based on which segments are connected portals, and thus create a lightweight topological graph where each node is assigned a shape. Furthermore, frontier information can be shared by sending positions of frontier viewpoints (the extraction of which is described in section 4.4) and attaching these positions to the nearest segments that we know the viewpoints are reachable from. Attaching frontiers to segments is important, as if we did not do it, a receiving robot might assign the frontier to a different segment (for example to a room that shares a wall with the room where the frontier actually is but the path to which is much longer) and then waste time going to places that bring no global exploration reward. Lastly, we can utilize the shapes to share information about areas in which the surfaces have been inspected by simply computing how many facets in the FacetMap belong to a given segment and then sending that as a single number for each segment.

For the purpose of this task, the shape used to represent segments is a bounding box with 4 degrees of freedom. This shape was chosen for the reasons that using spheres would mean we would have to keep the segments roughly spherical (since for example if we had two rectangular rooms above one another and then approximated each with a sphere, they would drastically overlap) and the usage of complex shapes such as polygons was deemed too complicated for this task. The bounding boxes can also very well approximate tunnels and rooms in urban and tunnel environments.

We define a new map type — an L-SegMap $\mathbb{L}$ as the lightweight representation of a SegMap, FacetMap and a FrontierMap using the ideas mentioned above, and we share these L-SegMaps between robots. The process of creating L-SegMaps, sharing them between robots and updating them with other L-SegMaps is described further in this chapter.

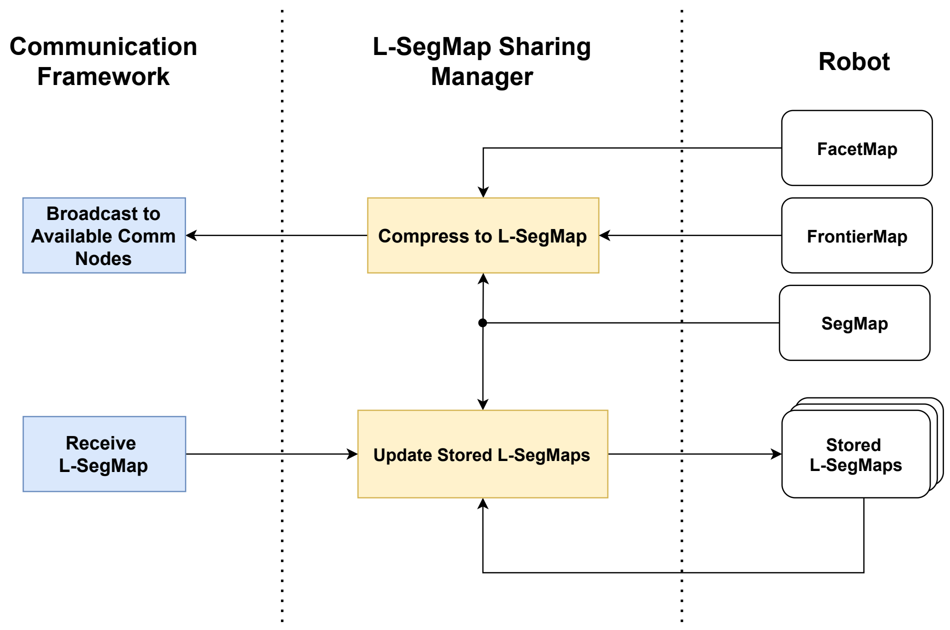

Figure 5.1: Diagram of the framework used to create, update and share L-SegMaps between robots

5.1 L-SegMap Sharing Framework

The framework for extracting and compressing important information from a robot's SegMap, FrontierMap and FacetMap into an L-SegMap, sharing of the L-SegMaps and updating of the received L-SegMaps is shown in Figure 5.1. In this section, the process of compressing information from the FacetMap, FrontierMap and SegMap of a given robot is described. The process of compressing shapes and surface coverage is also shown in Figure 5.2.

5.1.1 Sharing Topology

The geometrical shape used to compress segments' shapes in this work is a bounding box with 7 degrees of freedom (DOF):

- 3DOF - x_B, y_B, z_B - coordinates of the center of the box
- 3DOF - d_a, d_b, d_c - side lengths of the box
- 1DOF - α - rotation around the z axis

Ideally, the bounding box should maximize the number of points that belong to the segment and also belong to the bounding box of the segment and minimize the number of points that are in other segments but also fall into the bounding box. We simplify this task of

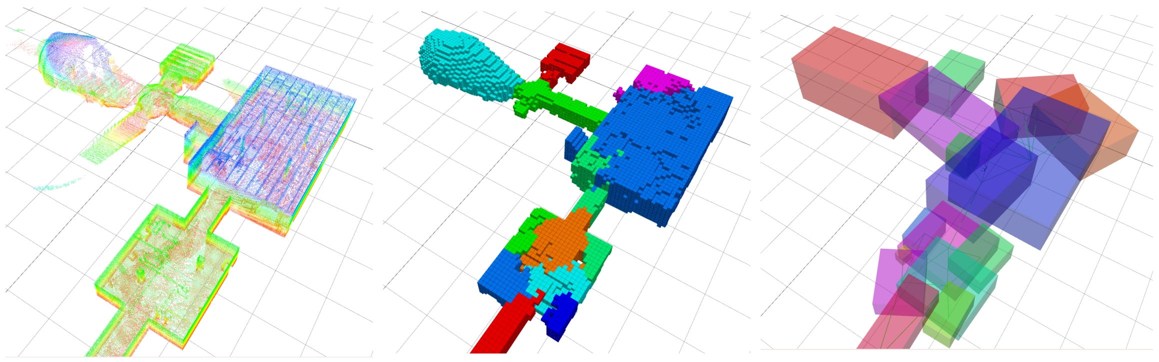
Figure 5.2: The transition from an occupancy octree (left) to a SegMap (middle) and then compression to an L-SegMap (right). The colors of the L-SegMap are distributed as a heatmap of surface coverage (red being mostly uninsptected, blue and purple mostly inspected)

finding an optimal bounding box for a given segment σ_i to the task of finding the bounding box with the minimal volume that contains all voxels $m_{\sigma_i} \in M_{\sigma_i}$.

This problem can be solved with the usage of the Quickhull algorithm [33] but for simplification and faster computation, the center point of the bounding box$(x_B, y_B, z_B)^T$ was set to be the centroid of voxels in σ_i and then a simple method of interval division for $\alpha \in < 0, \frac{\pi}{4})$ was employed to approximate the optimal α.

5.1.2 Sharing Frontiers

The approach chosen for sharing of frontier information is to send one frontier viewpoint for each frontier cluster in the FrontierMap $\mathbb{F}$ and also information about the nearest segment the viewpoint is reachable from. This way the other robots can plan paths to these frontiers in the received maps.

5.1.3 Sharing Surface Coverage

The approach chosen for sending surface coverage information is to compute the percentage of inspected surfaces that belong to a given segment. For this computation, we must first decide how to determine to which segment a given facet belongs. We do this by casting a ray from each facet in the direction of its normal and if the ray intersects a segment σ_i, we assign the facet to that segment. For each segment, we thus obtain a number of inspected facets n_{ins} and uninspected facets n_{uni} and then simply send the value

$$\gamma_{\sigma_i} = \frac{n_{ins}}{n_{ins} + n_{uni}} \tag{5.1}$$

for each segment. Doing this raycasting for each facet whenever we want to send the current SegMap would, however, require a time-consuming recomputation. For this reason, the surface coverage is cached for each segment and peridocially recomputed only for segments that are near the robot at a given time.

5.2 Localization in Received L-SegMaps

A very important feature of the L-SegMap is that it allows a robot to approximately determine whether a given point $\mathbf{x}$ belongs to the segmented free space explored by another robot which has shared its L-SegMap. Because the shared shapes are bounding boxes, we of course lose some information about the shape of the segment and as such, the points can be assigned to wrong segments.

To deal with this problem, a probabilistic approach is proposed. We define a function $\mathbf{P}(\mathbf{x}, \sigma_i^L)$ that represents an estimated probability that a point $\mathbf{x}$ belongs to the free space belonging to the original segment σ_i when only its lightweight version with just the simplified shape information σ_i^L is available.

Specifically for the used shape of bounding boxes, we define this function in the following manner. First we transform the coordinates of point $\mathbf{x}$ to the coordinate system of the bounding box by substracting the position of the box's center $\mathbf{c}_B$ and then projecting into the three axes of the bounding box, which are the original axes of the world coordinate system rotated around the world z axis by α. The transformed position can be written as

$$\mathbf{x}' = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} (\mathbf{x} - \mathbf{c}_B). \tag{5.2}$$

We then define relative distances r_a, r_b, r_c in the direction of the bounding box axes as

$$r_a = \frac{x_x}{d_a}, \quad r_b = \frac{x_y}{d_b}, \quad r_c = \frac{x_z}{d_c}. \tag{5.3}$$

where x_x, x_y, x_z are the x,y,z components of $\mathbf{x}'$ and d_a, d_b, d_c are the side lengths of the bounding box.

We select the largest relative distance as $r_{max} = \max\{r_a, r_b, r_c\}$ and then set the result of the function $\mathbf{P}(\mathbf{x}, \sigma_i^L)$ as

$$\mathbf{P}(\mathbf{x}, \sigma_i^L) = \begin{cases} 0, & r_{max} \geq 1 \\ \frac{1-r_{max}}{0.2}, & 1 > r_{max} > 0.8 \\ 1, & r_{max} \leq 0.8 \end{cases} \tag{5.4}$$

With this function, we assign 0 probability that a point belongs to the original segment σ_i when it is oustide its the bounding box. When it is inside the box, the probability increases the closer it is to the center of the box and when the point lies in a box that is the original box scaled by the factor 0.8, the probability estimate is set to be 1.

By assigning points to segments in a received L-SegMap $\mathbb{L}$, we can then search and plan along the segments and portals of the received L-SegMap. This is very useful, because this allows the planning algorithms for example to determine whether frontier viewpoints in the local maps of the robot will lead to another frontier in the received L-SegMap or if it leads into a dead end completely explored by another robot. The methods utilizing the localization feature of the L-SegMaps are described in detail in subsection 6.2.2.

5.3 Merging Information from L-SegMaps

In this thesis, the L-SegMaps are not merged into a single map, but rather the newest L-SegMap from a given robot is always kept, and when a new L-SegMap is received or when the robot's full SegMap changes, all stored L-SegMaps are updated.

The information that needs to be updated is frontier information. This is due to the fact that if an L-SegMap contains a frontier viewpoint ξ_F and then the robot receives an L-SegMap from a different robot in which ξ_F falls into explored space, we want to discard that viewpoint, as it has probably been already explored by the other robot.

For this reason, we keep a global frontier utility value $u_F(\xi)$ for each sent frontier viewpoint and then update these values based on information from other L-SegMaps, when they are recevied, and also based on the current SegMap of the robot.

Let us assume that a robot has received a set of L-SegMaps L in which only the most recent L-SegMap from each robot is saved. During a single recalculation iteration, the utility of a frontier viewpoint ξ_F in L-SegMap $\mathbb{L}$ is calculated as

$$u_F(\xi_F) = 1 - \max_{\mathbb{L}' \in L \setminus \mathbb{L}} \max_{\sigma^L \in \mathbb{L}'} \mathbf{P}(\xi, \sigma) \tag{5.5}$$

Less formally, we can say that the utility of a frontier in a received L-SegMap decreases with the probability that the frontier viewpoint falls into explored space of any other L-SegMap.

The L-SegMaps are also updated with the robot's local SegMap in the same manner, albeit with higher precision, because we can compute the intersections of bounding boxes and the voxels of a segment and also precisely determine if a frontier falls into the OOM of the robot. We could also update surface coverage values of segments based on the overlap with other segments in other L-SegMaps but that has not been explored in this thesis.

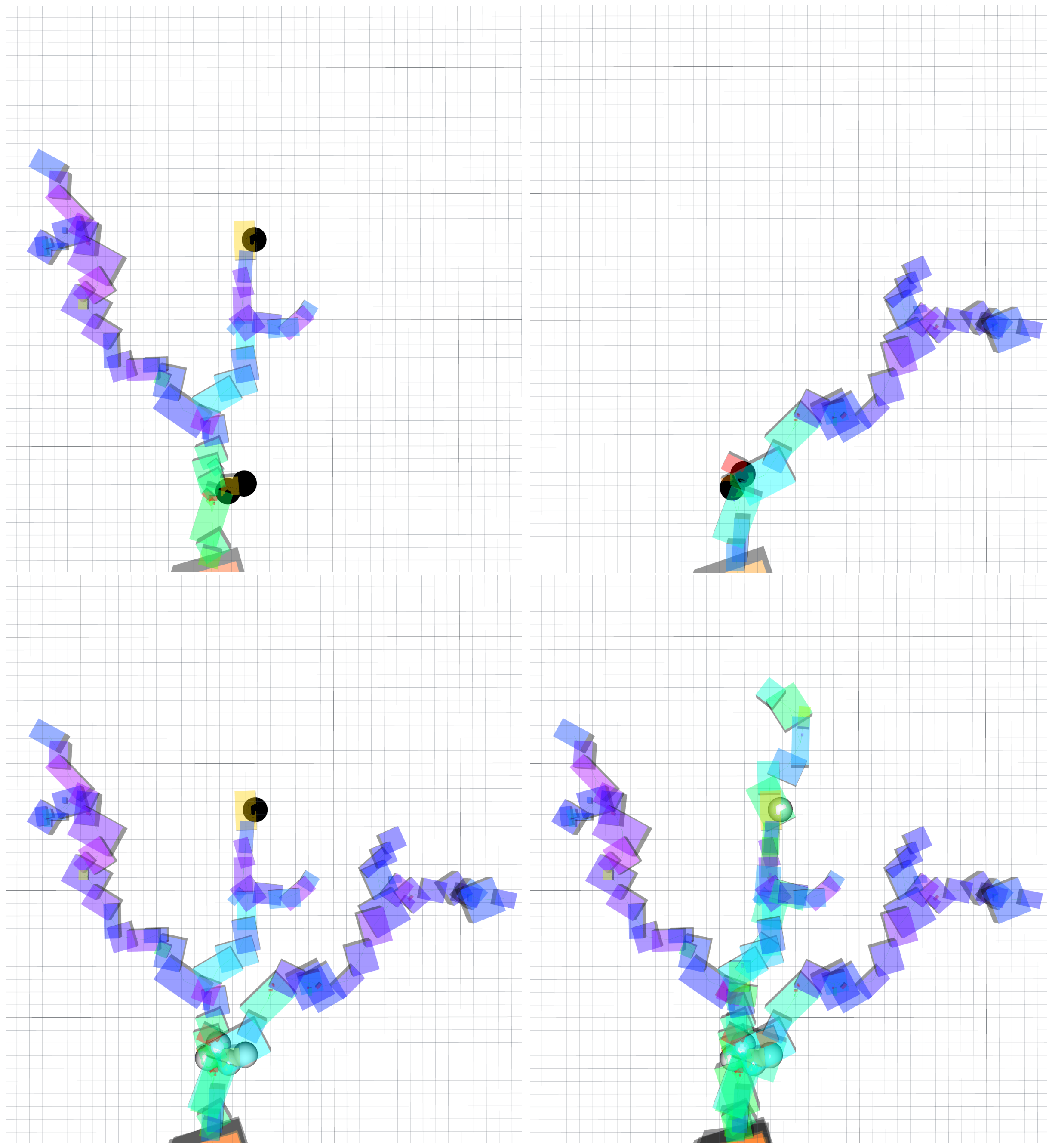

Figure 5.3: Illustration of L-SegMaps being updated with other L-SegMaps onboard robot C From left to right, top to bottom: An L-SegMap sent from robot A with frontiers with $u_F(\xi) = 1$ marked as black dots. Robot C is waiting in the staging area. (1), an L-SegMap sent from robot B (2), the two L-SegMaps overlaid and updated one with the another onboard robot C with the frontiers whose value is updated to $u_f(\xi) = 0$ marked as white (3) and the two L-SegMaps updated with the SegMap of robot C after it has launched and navigated directly to the only globally unexplored frontier (4)

Chapter 6

Search Strategies

Contents

In this chapter, three different search strategies are proposed. The main requirement for these strategies is that they need to be autonomous. This means that the robots must be able to act completely alone, as they can easily lose communication with the outside base and other robots.

For this reason, all three designed strategies are based on the principle of each robot autonomously choosing currently best goals according to a pre-defined reward function, with the available information (current state of the maps $\mathbb{O}$, $\mathbb{S}$, $\mathbb{F}$, $\mathbb{G}$ and received L-SegMaps L), and then navigating towards these goals.

For each strategy, its single-UAV and cooperative (i.e. utilizing received L-SegMaps) variants are presented. The cooperative variant will work in the same fashion as the single-UAV variant if no L-SegMaps are received, but the variants are presented separately to better explain how they were designed.

The cooperative methods rely on sharing lightweight information in the form of L-SegMaps, described in chapter 5, that are periodically broadcasted from all UAVs and UGVs. The UAVs then improve their reward functions with the currently received L-SegMaps and can easily replan when new L-SegMaps are received.

The strategies differ in the goal viewpoints they consider and the utilized reward functions. What they have in common is that they all utilize the occupancy octree, FacetMap, SegMap, and TopoMap described in the previous chapters for fast high level planning and. They also share the structure of a global and a local planner described in the section below.

6.1 Global-Local Search Planning Structure

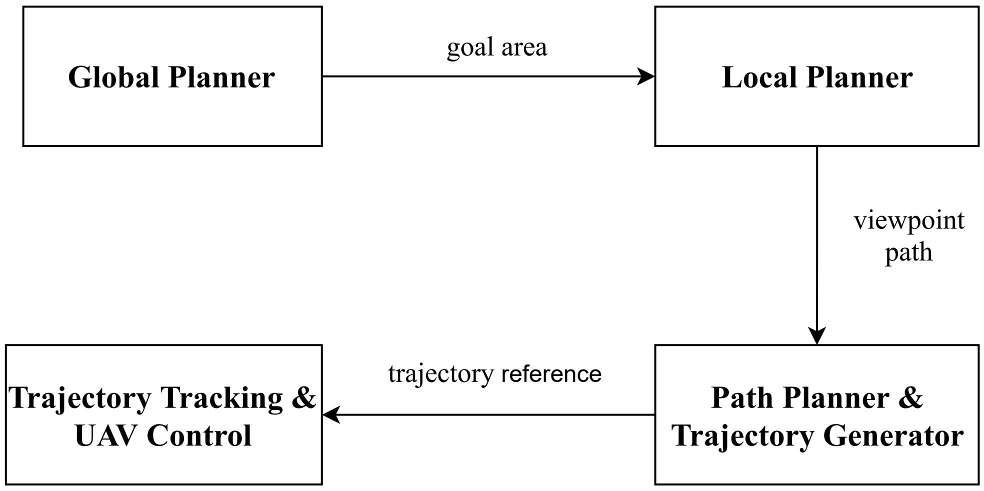

Figure 6.1: The designed planning structure that is common for all 3 strategies that are presented in this work

All three presented strategies share a structure that is common for large scale indoor exploration, which is to divide planning into a global planner and a local planner, such as in [13]. In [13], the local planner generates informative trajectories in a small area around the UAV for as long as there are any informative areas nearby. When the local planner sees no available frontiers in the nearby area (for example upon reaching the end of a tunnel), the global planner, which plans across the whole environment, is triggered and relocates the UAV to informative areas, and upon reaching them activates the local planner anew.

In this work, a similar approach to [13] was designed and is illustrated in Figure 6.1. The global planner calculates possible areas of interest (for frontier exploration or surface inspection, based on the specific strategy) in the whole environment and guides the local planner to these areas. The local planner then plans at a high frequency and tries to maximize the information gain from frontiers or surfaces in the UAV's nearby area.

Since the global planner must sample informative viewpoints (we do not want to send the robot to areas that contain frontiers or surfaces that are not explorable by any robot configuration with the given sensors, for example in narrow areas) throughout the whole environment, which is continually expanding and since no caching is utilized for these viewpoints, the density of sampled viewpoints must be drastically lower than for the local planner. For frontier viewpoints, it achieves this by utilizing the reduced variant of the method described in section 4.4 to obtain a set of widely-spaced frontier exploration viewpoints Ξ_F^G = **GenerateFrontierViewpoints**$(B_{\mathbb{O}}, false)$ in the bounding box $B_{\mathbb{O}}$ of the entire OOT. For surface inspection viewpoints in the DEI strategy, it achieves this through, analogically, using the reduced variant of the surface inspection viewpoint generation algorithm described in Ξ_S^G = **GenerateSurfaceViewpoints**$(B_{\mathbb{O}}, false)$ described in section 2.7.

The local planner only plans in a small bounding box B_{local} around the UAV's current position and can therefore use the full calculation of viewpoints to obtain a rich set of frontier viewpoints Ξ_F^L = **GenerateFrontierViewpoints**$(B_{local}, true)$, and in the DEI strategy also a rich set of surface viewpoints Ξ_S^L = **GenerateSurfaceViewpoints**$(B_{local}, true)$

Furthermore, to reduce computational strain on the UAV, the global planner replans only when either:

1. the local planner cannot find informative and reachable viewpoints in the nearby area,
2. a user-defined time has passed since the last global replanning, or
3. a new L-SegMap is received (in the cooperative variants of the strategies)

When the replanning of the global planner is triggered, it computes the sparse set of viewpoints $\Xi^G = \Xi_S^G \cup \Xi_F^G$ and selects the best viewpoint from Ξ^G according to a global reward function specified by the employed strategy. It then tries to navigate the UAV to a user-defined distance δ_{global} from the viewpoint. When the UAV gets closer than δ_{global} to the viewpoint, the planning is passed to the local planner. If the local planner is active when the global planner produces a new viewpoint, the global planner first checks, if the local planner is not already exploring near that best global viewpoint and if it is, it does not change its course.

To enable smoother motion and maximally utilize the implemented trajectory generator, the local planner does not only plan towards the best next viewpoint from the current position ξ_{curr}, but rather holds a path of viewpoints Ξ_{path}. When the current length of the viewpoint path becomes lower than a user-defined path length, a best next viewpoint is calculated *relative* to the last viewpoint in Ξ_{path}. The trajectory generator then plans smooth collision-free trajectories through these viewpoints, which can be much faster than always going to a single viewpoint, stopping and selecting a new one. The local planner then appends viewpoints to Ξ_{path} based on a reward function calculated relative to the last viewpoint in Ξ_{path}. The local planner uses a reward function that is defined as the sum of the global reward function used by the global planner and a motion factor, which takes into account the dynamics of the UAV.

All three presented strategies share this structure and planning paradigm. They differ mainly in the reward functions they use for selecting best next viewpoints and in the way they travel to viewpoints.

6.2 Receding Horizon Next Best View Strategy

The first designed strategy — an application of the receding horizon next best view (RHNBV) approach — focuses solely on the exploration of frontiers. Robotic frontier-based exploration is a widely researched topic and many methods exist for this task. In this work, the Receding Horizon Next Best View (RHNBV) method, which is used for volumetric exploration in e.g. [12, 13, 17], was chosen as the baseline method and then adapted to the global-local planner structure. The principal idea of the RHNBV method is to iteratively find viewpoints that will uncover frontiers, select the best viewpoint according to a defined reward function and then to navigate the robot to the best viewpoint.

In the single-UAV variant of this strategy, the reward function is designed to prefer the currently nearest viewpoints. Some applications of RHNV design the reward function to maximize information gain while minimizing distance. In this work, we consider all large enough frontier clusters to be equal in the terms of information value (since it would be very hard to determine how much volume of V_{unk} will be uncovered after exploring the frontier) and only minimize distance traveled to the frontiers. In the cooperative variant, the reward function also looks into received L-SegMaps to determine whether a given frontier in the robot's local FrontierMap falls into space that has already been explored by other robots (which we call *globally explored space* in the rest of this chapter). If so, the UAV evaluates the frontier based on whether exploring the frontier will move the UAV towards frontiers that have not yet been explored by any robot.

6.2.1 Single-UAV Approach

When the UAV does not have any received L-SegMaps, we assign each viewpoint that uncovers frontiers in a large enough (not considered as noise in the DBSCAN algorithm in section 4.4) frontier cluster to have the same importance and only consider distances to these viewpoints. We utilize the SegMap $\mathbb{G}$ to quickly calculate approximate distances to frontier viewpoints $\xi \in \Xi_F^G$ using the function of the segmap $\mathbf{D}(\mathbb{G}, \xi_1, \xi_2)$ described in section 3.3. We define the global reward function of the RHNBV strategy for a frontier viewpoint ξ in globally unexplored space relative to a given viewpoint ξ_{prev} as

$$\mathbf{R}_{RH}^{GU}(\xi, \xi_{prev}) \equiv -\mathbf{D}(\mathbb{G}, \xi, \xi_{prev}). \tag{6.1}$$

As defined in the global-local planner structure, the global planner simply chooses the best next viewpoint ξ^* relative to the UAV's current viewpoint ξ_{curr} (composed of the UAV's current position and heading) as

$$\xi^* \equiv \arg\max_{\xi \in \Xi_F^G} \mathbf{R}_{RH}^{GU}(\xi, \xi_{curr}). \tag{6.2}$$

and navigates the UAV to it. When the UAV gets closer than δ_{global} to ξ^*, the local planner is engaged.

The reward function for the local planner was designed as the sum of the global reward function and a factor $\mathbf{R}^M_{RH}(\xi)$ which deals with the dynamics of the UAV and depends on the specific hardware used on the UAV. In this work, we assume all UAVs to have multi-row LiDAR sensors. For these sensors, a very effective way of exploring frontiers is to dynamically determine the direction in which there is the majority visible frontier viewpoints, and then to increase the reward of frontier viewpoints based on the distance from the UAV in this direction. Formally, we write this factor as:

$$\mathbf{R}^M_{RH}(\xi, \xi_{prev}) = \alpha_{LF}\mathbf{h} \cdot (\mathbf{p}_\xi - \mathbf{p}_{\xi_{prev}}) \tag{6.3}$$

where $\mathbf{h}$ is the normalized direction towards the frontiers and α_{LF} is a user-defined parameter that specifies the importance of this factor. The direction $\mathbf{h}$ is calculated by first calculating the sum $\mathbf{h}'$ of normalized direction vectors towards all local frontier viewpoints $\Xi_{visible} \subseteq \Xi^L_F$ that have a clear line of sight towards ξ_{prev} as

$$\mathbf{h}' = \sum_{\xi' \in \Xi_{visible}} \frac{\mathbf{p}_{\xi'} - \mathbf{p}_{\xi_{prev}}}{|\mathbf{p}_{\xi'} - \mathbf{p}_{\xi_{prev}}|} \tag{6.4}$$

and then setting $\mathbf{h}$ as

$$\mathbf{h} = \begin{cases} \frac{\mathbf{h}'}{|\mathbf{h}'|}, & |\mathbf{h}'| \geq \alpha_{dirmag} \\ 0, & |\mathbf{h}'| < \alpha_{dirmag} \end{cases} \tag{6.5}$$

where α_{dirmag} is a user-defined threshold. This is done so that the local planner is more incentivized to plan according to the global rewards in complex areas, such as crossroads or large rooms, and more according to the frontier direction when in tunnels or corridors.

6.2.2 Cooperative Approach

Without any information about the exploration progress of other robots, a team of UAVs using the RHNBV strategy can and usually will explore some areas multiple times, even when there are areas that have not been explored by other robots. A very simple possible solution could be to add a preferred direction factor to the global reward function and then set different preferred directions for each UAV. This approach was briefly tested, but it does not bring an acceptable guarantee that the UAVs will not explore areas already explored by other robots.

For this reason, the following approach utilizing received L-SegMaps was designed: Let us consider a robot that is currently searching the environment and has received some L-SegMaps $\mathbb{L}_1$, $\mathbb{L}_2$, ..., $\mathbb{L}_N$ (each considered L-SegMap is the newest one received from a given robot). When the robot generates some frontier viewpoints $\xi \in \Xi_F$ (either from the global or local planner), it first assigns for each viewpoint $\xi \in \Xi_F$ and each L-SegMap $\mathbb{L}_i$ a probability estimate $\mathbf{P}_{visited}(\xi, \mathbb{L}_i)$ that ξ lies in any of the segments σ^L of $\mathbb{L}_i$ by utilizing the function $\mathbf{P}(\mathbf{x}, \sigma^L)$ defined in section 5.2 as

$$\mathbf{P}_{visited}(\xi, \mathbb{L}_i) = \max_{\sigma^L \in \mathbb{L}_i} \mathbf{P}(\mathbf{p}_\xi, \sigma^L). \tag{6.6}$$

We construct a set of L-SegMaps L in which the viewpoint ξ might lie within visited space as

$$L_{pos}(\xi) = \{\mathbb{L} \in \mathbb{L}_1, \mathbb{L}_2, ..., \mathbb{L}_N | \mathbf{P}_{visited}(\xi, \mathbb{L}) > 0\}. \tag{6.7}$$

We can then calculate a probability estimate p_{GU} that the frontier viewpoint ξ lies in globally unexplored space as

$$p_{GU}(\xi) = 1 - \max_{\mathbb{L} \in L_{pos}(\xi)} \mathbf{P}_{visited}(\xi, \mathbb{L}). \tag{6.8}$$

If $p_{GU} = 1$, the global reward function for ξ is the same as in the noncooperative variant of the RHNBV strategy. However, if $p_{GU} < 1$, exploring the frontier might not, by itself, expand the globally explored space, but exploring this frontier might move the UAV towards frontiers that were not explored by any other robots.

Since the L-SegMaps store information in a graph structure, same as the local SegMaps $\mathbb{G}$, we can search for the frontiers that are nearest to the viewpoint's position $\mathbf{p}_\xi$ in a given L-SegMap. The travel distance computation of a SegMap requires the start and end point to be assigned to a segment. The frontiers that are sent in the L-SegMaps are already each assigned to one of its segments, but we must decide to which segment to assign the queried viewpoint ξ. For each L-SegMap $\mathbb{L} \in L_{pos}$, we select the segment most likely to contain ξ as

$$\sigma^L_{ml}(\xi) = \arg\max_{\sigma^L \in \mathbb{L}} \mathbf{P}(\mathbf{x}_\xi, \sigma^L). \tag{6.9}$$

For each frontier viewpoint ξ^L_F sent in the L-SegMap $\mathbb{L}$ we can then execute topological path A* planning along the portals of $\mathbb{L}$ using the function $\mathbf{D}(\mathbb{L}, \mathbf{p}_{\xi^L_F}, \mathbf{p}_\xi)$, same as with a local SegMap, and thus obtain an estimated travel distance between ξ^L_F and ξ. We then define the global reward of exploring ξ from a current viewpoint ξ_{prev} if $p_{GU}(\xi) = 0$ as

$$\mathbf{R}^{GE}_{RH}(\xi, \xi_{prev}) = -\mathbf{D}(\xi_{prev}, \xi) + \max_{\mathbb{L} \in L_{pos}(\xi)} \max_{\xi^L_F \in \mathbb{L}} -\frac{\mathbf{D}(\xi, \xi^L_F)}{u_F(\xi^L_F)}. \tag{6.10}$$

where $u(\xi^L_F)$ is the frontier utility value that is updated for each frontier viewpoint in an L-SegMap based on the method described in section 5.3. We then combine this with the reward function for frontiers in globally unexplored space $\mathbf{R}^{GU}_{RH}$ into the total cooperative reward function for frontier viewpoints

$$\mathbf{R}^{total}_{RH}(\xi, \xi_{prev}) = p_{GU}(\xi)\mathbf{R}^{GU}_{RH}(\xi, \xi_{prev}) + (1 - p_{GU}(\xi))\mathbf{R}^{GE}_{RH}(\xi, \xi_{prev}). \tag{6.11}$$

This reward function is used for both the global planner and local planner as the global reward function in the RHNBV strategy.

6.3 Viewpoint Enhanced RHNBV Strategy

The main downside of the RHNBV strategy is that it has no notion of surface information gain. With the RHNBV strategy, the UAV can miss out on inspecting large surface areas even if it would only have to fly with a different heading.

The proposed RHNBV strategy with viewpoint enhancing (RHNBV-VPE, referred to also as VPE in this thesis) tries to take full advantage of the fact that UAVs can change their heading almost independently on the executed trajectory, meaning that with a given sequence of positions, we can roughly choose the headings throughout the trajectory (subject to maximum heading rate constraints) and the UAV will execute this trajectory in almost the same time as the original trajectory.

In this strategy, the global and local planner work in exactly the same way as in the RHNBV strategy. The only difference from the RHNBV strategy is that the generated viewpoint path is additionally enhanced before being sent to the trajectory generator node.

6.3.1 Single-UAV Approach

The viewpoints in the viewpoint paths of the RHNBV strategy have no defined heading and are defined only by their position. Therefore, in the RHNBV strategy, the UAVs will usually travel directly towards frontier positions or when traveling across large distances, between portals of segments, and naively aim the UAV in the direction of flight.

When the trajectory generator receives a viewpoint path, it generates a trajectory of tuples (position $\mathbf{p}$, heading φ) that the UAV will then attempt to follow. If the RHNBV strategy is employed, the headings of the trajectory are set by the trajectory generator and thus do not deal with surface inspection.

The method works for trajectory enhancing works as follows: Whenever the RHNBV strategy adds a new viewpoint ξ_{new} to the current viewpoint path in which the currently last viewpoint is denoted as ξ_{last}, the viewpoint enhancement is triggered and works as follows:

1. We first ask the trajectory generator, how it would normally reach ξ_{new} from ξ_{last} and thus obtain a trajectory of positions $\mathbf{p}_1$, $\mathbf{p}_2$, ..., $\mathbf{p}_N$ and headings φ_1, φ_2, ... φ_N.

2. Then, we iterate over the position and heading trajectory and cut this trajectory every $d_{cutting}$ (where $d_{cutting}$ is a user-defined parameter) meters and thus obtain N_{cut} positions and headings that we denote as $\mathbf{p}_1^{cut}$, $\mathbf{p}_2^{cut}$, ..., $\mathbf{p}_{N_{cut}}^{cut}$ and φ_1^{cut}, φ_2^{cut}, ... $\varphi_{N_{cut}}^{cut}$.

3. For each cut viewpoint ξ_i^{cut}, represented by the cut position and heading, we calculate the frontier exploration value and surface inspection value by using the methods described in section 2.6 and section 4.3 , respectively.

4. If $\mathbf{I}_F(\xi_i^{cut}) > k_F$, where k_F is a user-defined constant, we do not replace the cut viewpoint with a new one and move on to the next cut position. We do this for the reason that when the UAV moves near frontier areas, the FacetMap will usually not have enough data about surfaces to calculate $\mathbf{I}_S(\xi_i^{cut})$.

5. For each cut viewpoint, we then sample viewpoints $\xi_{sample} \in \Xi_{sample}$ in a sphere around ξ_i^{cut} and try to find a viewpoint that would cover a much larger uninspected surface area than the original cut viewpoint. Namely, we try to find any viewpoint ξ_{sample} for which
$$\mathbf{I}_S(\xi_{sample}) > k_S \mathbf{I}_S(\xi_i^{cut}) \tag{6.12}$$
where k_S is a user-defined constant that determines how much more surface inspection information the sampled viewpoints must bring than the original cut viewpoint ξ_i^{cut} to set them as additional viewpoints in the viewpoint path. We denote all viewpoints that satisfy this condition for a given cut position as Ξ_{++}.

6. For each cut position, if $\Xi_{++} \neq \emptyset$, we add the viewpoint
$$\xi^* = \arg \max_{\xi \in \Xi_{++}} \mathbf{I}_S(\xi) \tag{6.13}$$
to the viewpoint path.

7. We stop cutting the trajectory $2d_{cutting}$ before the last position of the trajectory and then try to enhance ξ_{new} (the viewpoint that was originally the only one to be added to the path) by adding a heading requirement to it. Similarly as with cut positions, we do not add a heading requirement if $\mathbf{I}_F(\xi_{new}) > k_F$, which will mostly happen when the RHNBV is currently near a frontier.

We then pass this enhanced trajectory to the trajectory generator which will adapt its trajectory according to the required viewpoints in the viewpoint path. The single-UAV variant of this strategy then works in exactly the same way as the RHNBV strategy, but uses the viewpoint enhanced trajectory generation instead of the one in RHNBV which does not consider informative headings.

6.3.2 Cooperative Approach

The cooperative extension of this strategy is simply to block the enhancement of cut viewpoints if the cut position $\mathbf{x}^{cut}$ lies in a segment of the L-SegMap that has its surface coverage value above a certain user-defined threshold γ_{min}. Formally, we block the viewpoint enhancement at position $\mathbf{x}^{cut}$ if
$$\max_{\mathbb{L} \in L_{pos}(\mathbf{x^{cut}})} \max_{\sigma^L \in \mathbb{L}} \gamma_{\sigma^L} \mathbf{P}(\mathbf{x}^{cut}, \sigma^L) > \gamma_{min}, \tag{6.14}$$
where $\gamma_{\sigma^L} \in < 0, 1 >$ is the surface coverage value of segment σ^L shared in the L-SegMaps.

6.4 Dead End Inspection Strategy

None of the strategies mentioned so far can ensure complete inspectedness of surfaces of the environment even if given enough time. One strategy that would consider not only frontier viewpoints but also surface inspection viewpoints could be to calculate frontier and surface viewpoints and then assign each frontier viewpoint a reward of $-\mathbf{D}$ each and surface inspection viewpoints $-\mathbf{D}+\alpha_{surf}$, which would greedily focus on exploring frontier and surface viewpoints, while preferring the former or the latter based on the sign of α_{surf}. This strategy was briefly tested, but soon abandoned because the assumption was made that it would be better to focus on thoroughly searching the environment from its *end* rather than from the start, because areas close to the start will be visited by most robots of the robot team and as such, the likelihood of missing an artifact can be lower than in the far reaches of the environment, where only a few robots will venture.

For this reason, a novel strategy — dead end inspection (DEI) — was designed. This strategy considers both surface inspection viewpoints Ξ_S (generated using the procedure **GenerateSurfaceViewpoints**() described in section 2.7) and frontier exploration viewpoints Ξ_F (generated using the **GenerateFrontierViewpoints**() procedure described in section 4.4). To force the UAV to fully search dead ends and to not leave areas of uninspected surfaces in distant areas of the environment, the DEI strategy also rewards viewpoints that are further from the base station.

6.4.1 Single-UAV Approach

For the DEI strategy, we define a separate global reward function for surface inspection viewpoints $\xi_S \in \Xi_S$ and frontier exploration viewpoints $\xi_F \in \Xi_F$. We define the global reward function for frontier exploration viewpoints ξ_F in globally unexplored space relative to a previous viewpoint ξ_{pref} as

$$\mathbf{R}^{GU}_{DE}(\xi, \xi_{prev}) = \mathbf{D}_{home}(\mathbb{G}, \xi) - \mathbf{D}(\mathbb{G}, \xi, \xi_{prev}). \tag{6.15}$$

and for surface inspection viewpoints as

$$\mathbf{R}^{S}_{DE}(\xi, \xi_{prev}) = \mathbf{D}_{home}(\mathbb{G}, \xi) - \mathbf{D}(\mathbb{G}, \xi, \xi_{prev}) + \alpha_{surf}, \tag{6.16}$$

where $\alpha_{surf} < 0$ is a user-defined constant that specifies how much the UAV should prefer frontier exploration over surface inspection, $\mathbf{D}_{home}(\mathbb{G}, \xi)$ is the travel distance from the base station to viewpoint ξ calculated using the SegMap and $\mathbf{D}(\mathbb{G}, \xi_1, \xi_2)$ is the travel distance between two viewpoints also calculated using the SegMap $\mathbb{G}$ (see section 3.3 for more details).

As in the RHNBV strategy, if the viewpoints are considered by the local planner, a motion-specific factor is also added to the rewards of the viewpoints. For frontier exploration viewpoints, the motion specific factor is defined in the same manner as in the

RHNBV strategy in (6.3). For surface inspection viewpoints, the factor is defined simply as

$$\mathbf{R}^{SM}_{DE}(\xi, \xi_{prev}) = -k_{heading}|\varphi_{\xi} - \varphi_{\xi_{prev}}| \tag{6.17}$$

where $k_{heading}$ is a non-negative user-defined parameter that penalizes heading change, so that if the UAV is for example examining a large room, it flies along the walls instead of chaotically changing its heading according to the nearest viewpoints.

In tree-like structured parts of the environment, this reward function will cause the UAV to execute a sort of depth first search, which can be beneficial for the multi-robot missions, because if we are sure that a branch of the environment has been completely searched, we can discard it and stop other robots from exploring it.

6.4.2 Cooperative Approach

For each frontier viewpoint ξ_F we calculate a probability estimate that it belongs to globally unexplored space $p_{GU}(\xi)$ in the same manner as in the RHNBV strategy. The reward function for exploring a frontier already explored by other robots could possibly take into account the amount of uninspected surface areas that will be accessed through this frontier, but for simplicity, this was not examined in this thesis. The reward function for frontiers with $p_{GU}(\xi) = 0$ is defined simply as the reward function $\mathbf{R}^{GE}_{RH}$ in the RHNBV strategy, enhanced with the distance from home:

$$\mathbf{R}^{GE}_{DE}(\xi, \xi_{prev}) = \mathbf{D}_{home}(\xi) - \mathbf{D}(\xi_{prev}, \xi) + \max_{\mathbb{L} \in L_{pos}(\xi)} \max_{\xi^L_F \in \mathbb{L}} -\frac{\mathbf{D}(\xi, \xi^L_F)}{u_F(\xi^L_F)}. \tag{6.18}$$

where $u(\xi^L_F)$ is the frontier utility value that is updated for each frontier viewpoint in an L-SegMap by the method described in section 5.3. We then combine this with the reward function for viewpoints in globally unexplored space $\mathbf{R}^{GU}_{RH}$ into the total cooperative reward function for frontier viewpoints

$$\mathbf{R}^{total}_{RH}(\xi, \xi_{prev}) = p_{GU}(\xi)\mathbf{R}^{GU}_{RH}(\xi, \xi_{prev}) + (1 - p_{GU}(\xi))\mathbf{R}^{GE}_{RH}(\xi, \xi_{prev}). \tag{6.19}$$

The cooperative handling of surface inspection is the same as in the VPE strategy. For simplicity, we do not assign a new reward function for surface inspection viewpoints in explored space, but rather simply discard these viewpoints if they lie in space that has a high reported surface coverage. Formally, we discard surface inspection viewpoints ξ if

$$\max_{\mathbb{L} \in L_{pos}(\mathbf{p}_{\xi})} \max_{\sigma^L \in \mathbb{L}} \gamma_{\sigma^L} \mathbf{P}(\mathbf{p}_{\xi}, \sigma^L) > \gamma_{min}, \tag{6.20}$$

where $\gamma_{\sigma^L} \in < 0, 1 >$ is the surface coverage value of segment σ^L shared in the L-SegMaps and γ_{min} is a user-defined surface coverage threshold. We then use the global reward function $\mathbf{R}^{S}_{DE}$ for the surface inspection viewpoints that were not discarded.

Chapter 7

Experimental Evaluation

Contents

This chapter describes the implementation and evaluation of the proposed theoretical strategies and mapping structures. All three proposed strategies were evaluated for single-UAV missions in the realistic Gazebo simulation and the DEI strategy was also evaluated on a real UAV platform in an urban environment. To compare the approach using L-SegMap sharing with the noncooperative approach, the RHNBV strategy was tested on a team of 3 UAVs in simulation first for the situation where UAVs share L-SegMaps and then for when no L-SegMaps are shared.

7.1 Implementation

The theoretical mapping and planning structures have been implemented as a C++ library based on ROS, that allows for both standalone map building (e.g., for creating L-SegMaps onboard UGVs and sharing them with UAVs), and also for simultaneous map building and high level planning based on the built maps. The mapping algorithms can be run with any system structure based on ROS that provides the mapping algorithms with binary OOT data, odometry and a camera sensor model. The planning algorithms were integrated into the MRS system, available as open source at https://github.com/ctu-mrs/, which handles SLAM, autonomous UAV control, trajectory tracking and artifact recognition. For the purpose of evaluating the strategies, a simple trajectory planner node (for collision-free path planning and trajectory generation) was also implemented and is described in the following section.

7.1.1 Trajectory Planner

The structure of the global-local high level planning described in Section 6.1 is compatible with any trajectory and motion planners that safely navigate the UAV through viewpoints given by the local planner. For the purpose of testing and demonstrating the designed strategies, a simple path planner that creates collision-free trajectories through the OOT and uses a primitive braking mechanism for when the predicted trajectory is not safe, was implemented along with a simple trajectory generator that creates a roughly smooth trajectory from the collision-free path. We call the whole node consisting of a path planner and trajectory generator a trajectory planner.

The implemented trajectory planner works by first calculating a safe position trajectory with no regard for heading, then sampling this trajectory based on the currently desired velocity and then assigning each sampled position a heading that aims the UAV in the direciton of flight and aligns it with the required heading φ_{ξ_1}, if heading alignment is required. The headings set at each position of the trajectory are selected to satisfy the maximum heading rate ω specified by the UAV controllers.

For the calculation of a safe position trajectory (safe in this context meaning that the distance of each position of the trajectory from the nearest unknown/occupied voxel is greater than d_{min}) from the current position of the UAV $\mathbf{x}_{curr}$ to the position of the first next viewpoint $\mathbf{x}_{\xi_1}$, the planner performs a modified A* search across free-space voxels from the MRS OOT A* library. The A* calculaton sets the free-space voxels of the OOT at the deepest level as nodes of the search graph and uses euclidian distance between the center of a given voxel and the end position as the heuristic function.

After sampling the position trajectory based on the desired velocity (as the trajectory tracking and control nodes of the MRS system set the desired velocity based on the density of the positions in the trajectory), each sampled point is assigned a heading. The heading assignment process is to set the heading of a point $\mathbf{p}_i = (x_i, y_i, z_i)^T$ as

$$\varphi_i = atan2(y_i - y_{i-1}, x_i - x_{i-1}), \tag{7.1}$$

when the UAV is at least a distance d_{align} from the end position. If heading alignment is required for ξ_i and the UAV is closer than d_{align} to the end position, the headings at the trajectory position are set as a linear interpolation from the UAV's current heading to the required heading.

This trajectory is then passed to the trajectory tracking and control nodes in the MRS system. As control errors can occur, the executed trajectory can be different from the reference one, and might therefore cause collisions. Fortunately, the MRS trajectory tracker node publishes the trajectory that is predicted to be executed. This predicted trajectory is then utilized for a simple braking mechanism which stops the UAV (sends and empty trajectory to the trajectory tracker) whenever any point in the predicted trajectory is less than d_{safe} far from any obstacles. If this causes the UAV to get stuck in a position that is less than d_{safe} far from obstacles, the value d_{safe} for path planning is temporarily lowered until the UAV is again at a safe position, and then set back to the primary value.

This trajectory planner has been utilized in the early stages of the work on this thesis and worked sufficiently for testing the designed strategies. During the work on this thesis, however, a member of the MRS team developed a safer, smoother trajectory planner. To better demonstrate the strategies (as the original trajectory planner would stop at each viewpoint, which would considerably slow down the VPE and DEI strategies simply because they publish paths with more viewpoints) and for safety reasons concerning the real world experiment, this better trajectory planner was used for the rest of the work and is also utilized in the experimental evaluation of the strategies in both simulation and in the real world experiment.

7.2 Evaluation of Strategies in single-UAV Missions

The three designed strategies RHNBV, VPE and DEI were evaluated in the realistic Gazebo Ignition simulation. To ensure identical conditions for each run, the experiments were executed in the CloudSim environment using the Amazon Web Services servers provided by DARPA for the teams preparing for the competition.

The world chosen for evaluating these strategies is the world Simple Cave 3 from the DARPA SubT tech repository [1]. This world consists of multiple branches, and was chosen mainly for the demonstration of the DEI strategy, as there are dead ends near to the starting area and also since it was expected that the results should not vary much based on which branch the UAV selects.

For each strategy, the run that has inspected the highest amount of facets is presented, and surface inspection maps and graphs of the remaining runs are shown in the full results of experiments.

7.2.1 Evaluation Metrics

For evaluating the designed strategies, the following values were computed for each run:

- $V_{kno}[\mathrm{m}^3]$ = Volume of all occupied and free-space voxels in the OOT at the end of the run

[1]https://subtchallenge.world/home

Strategy	$V_{kno}[\text{m}^3]$	$S_{insp}[-]$	$S_{kno}[-]$	$p_{insp}[-]$	$p_{spv}[-]$	$v_{exp}[\text{m/s}]$	$A[-]$
RHNBV	95742.27	12173.67	16246.67	0.75	0.127	1.64	4
VPE	90749.7	11210.5	14938.5	0.752	0.124	1.5	2
DEI	82597.31	10718.67	13842.0	0.776	0.13	1.59	4.33

Table 7.1: Values of the evaluated metrics (described in subsection 7.2.1) from the single-UAV evaluation experiments averaged over all runs of each strategy that did not end in a crash of the UAV

- $S_{insp}[-]$ = Number of inspected facets in the FacetMap at the end of the run
- $S_{kno}[-]$ = Number of facets, both inspected and uninspected, in the FacetMap at the end of the run
- $p_{insp}[-] = \frac{S_{insp}}{S_{kno}}$ = number of inspected facets divided by the number of known facets in the FacetMap at the end of the run
- $p_{spv}[-] = \frac{S_{insp}}{V_{kno}}$ = number of inspected facets divided by the total volume at the end of the run
- $v_{exp}[\text{m/s}]$ = average speed of the UAV during exploration (during the time between exiting the staging area to the beginning of the homing procedure)
- A = number of correct artifact reports at the end of the run. This value does not evaluate the strategies well, as it relies on detection algorithms and also since artifacts are not distributed evenly through the environment, but is shown nonetheless.

7.2.2 Results and Discussion

For the demonstration of the main principles of the strategies, one run for each strategy is inspected in detail in this section. The full results of the experiments containing data for each run and the resulting maps of the environment are in the full results of experiments.

As can be seen on the maps in Figure 7.2, Figure 7.6, and Figure 7.6, each strategy has demonstrated the ability to explored a considerable portion of the environment's volume and surface. As was expected, the RHNBV strategy left many patches of the surface uninspected. The RHNBV-VPE has slightly helped this issue, but not by much. The DEI strategy, however, has demonstrated its ability to almost perfectly inspect surfaces in a dead end, as shown in Figure 7.6. The main achievement of this is that when the DEI has fully inspected a branch of the environment, we can stop all other robots from exploring that branch, compared to the RHNBV strategy which might miss more artifacts in that branch.

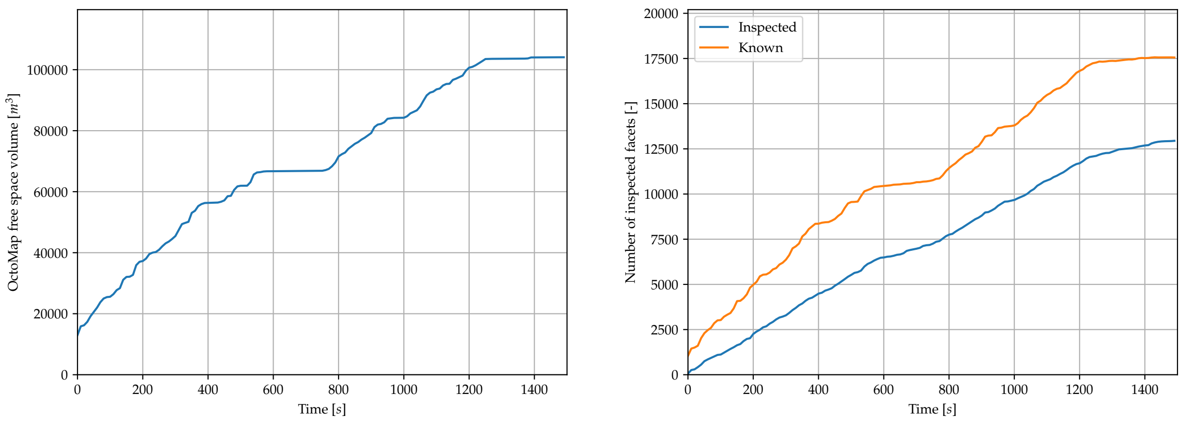

Figure 7.1: Volumetric exploration progress (left) and surface inspection progress showing the number of inspected facets and the total number of facets in the FacetMap (right) for the 2. run of the RHNBV strategy

The average explored volume and the average percentage of inspected facets shown in Table 1 also fit the expectations for all three strategies. The RHNBV strategy explored the largest volume of space but has the smallest percentage of inspected facets. The VPE strategy explored a slightly smaller volume of space, but inspected a higher percentage of facets than the RHNBV strategy, and the DEI strategy explored the least amount of volume but has the highest percentage of inspected facets.

A slight difference of the strategies can also be seen on the graphs of volumetric exploration in Figure 7.1, Figure 7.3, and Figure 7.5. The time for which the explored volume of the DEI strategy is constant (the time spent backtracking from a dead end) is considerably higher than that of the RHENBV strategy that was returning from a slightly longer branch of the environment. This means that the DEI spent a long time inspecting the surfaces in the dead end, which was expected.

The ability of the SegMap to navigate the UAV over large distances has also been demonstrated, as all the UAVs have successfully backtracked from each far end of the environment and all UAVs returned safely in each run, except for run 2 of the VPE strategy, where the UAV crashed, which was, however, not the fault of the algorithms designed in this thesis.

The FacetMap has also been shown to be useful as a surface representation of an apriori unknown environment, since the facets fit the contours of the environment very well and the inspected facets roughly fit the surfaces that have been seen by the UAV's cameras (as can be seen in the videos in the multimedia material at http://mrs.felk.cvut.cz/musil2021thesis).

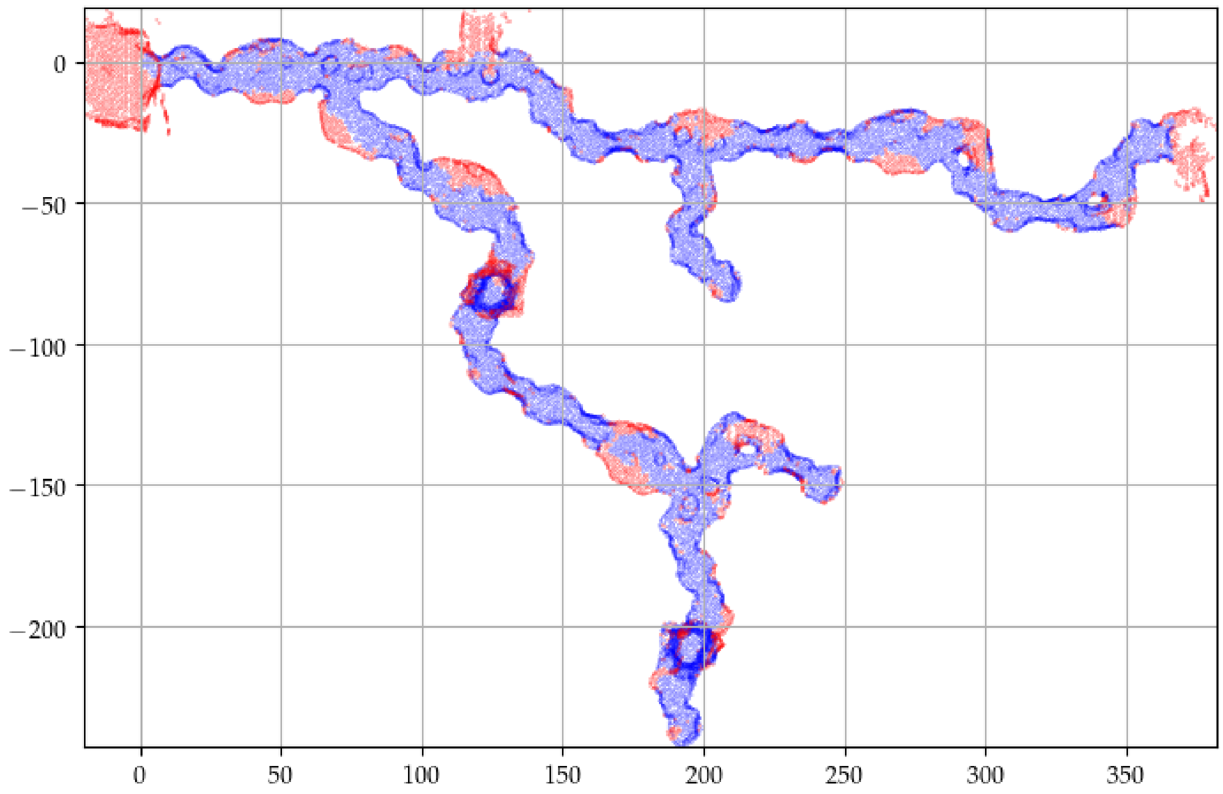

Figure 7.2: The map of inspected (blue) and uninspected (red) facets in the FacetMap projected onto the XY plane at the end of the 2. run of the RHNBV strategy. The distances are in meters.

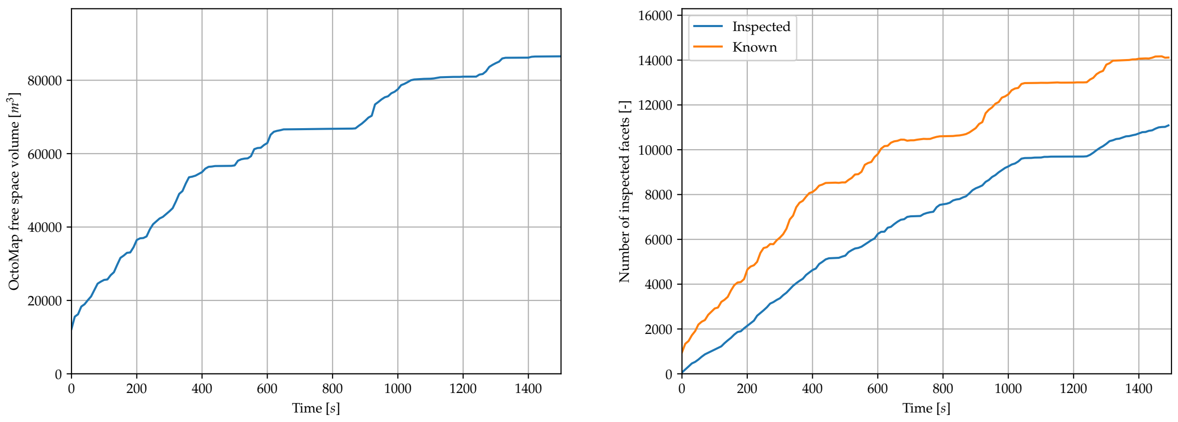

Figure 7.3: Volumetric exploration progress (left) and surface inspection progress showing the number of inspected facets and the total number of facets in the FacetMap (right) for the 2. run of the VPE strategy

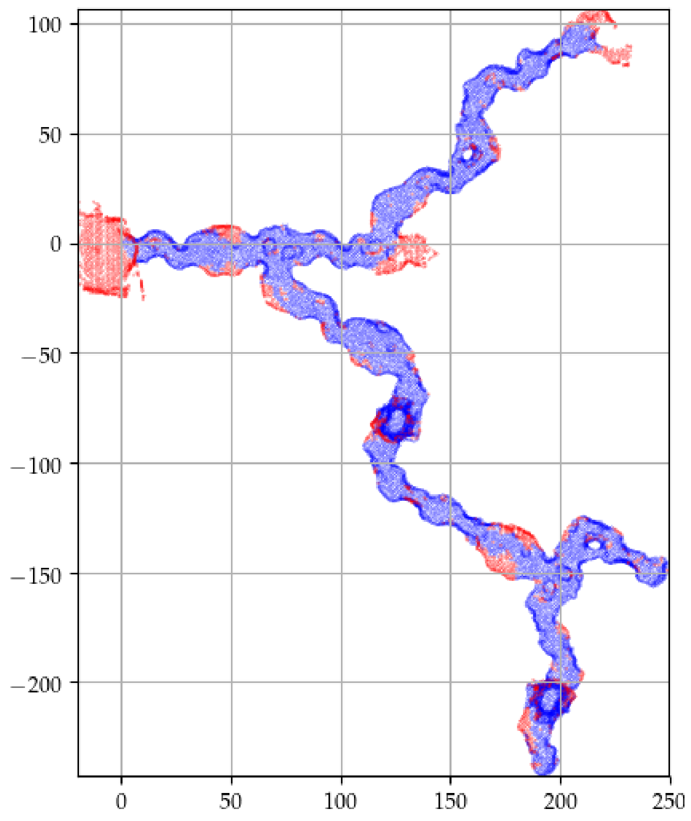

Figure 7.4: The map of inspected (blue) and uninspected (red) facets in the FacetMap projected onto the XY plane at the end of the 2. run of the VPE strategy. The distances are in meters.

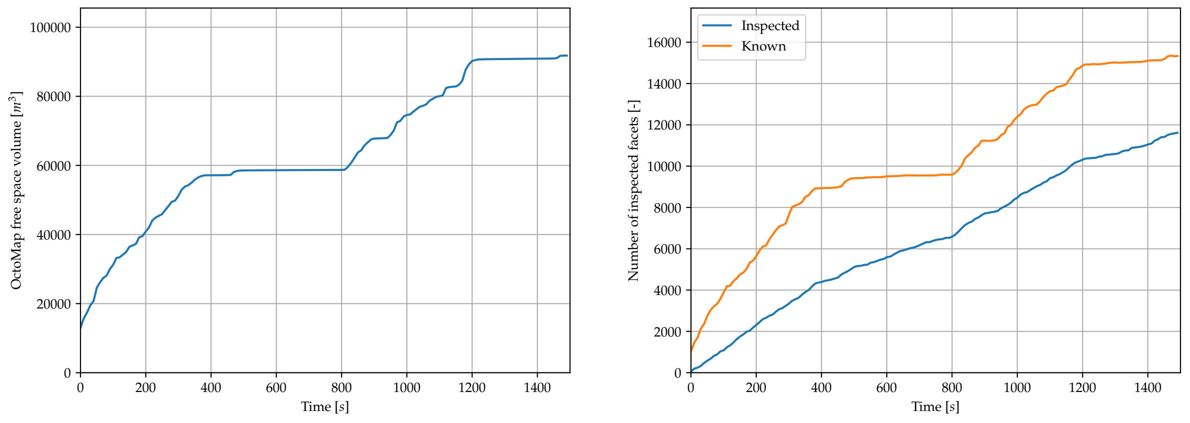

Figure 7.5: Volumetric exploration progress (left) and surface inspection progress showing the number of inspected facets and the total number of facets in the FacetMap (right) for the 3. run of the DEI strategy

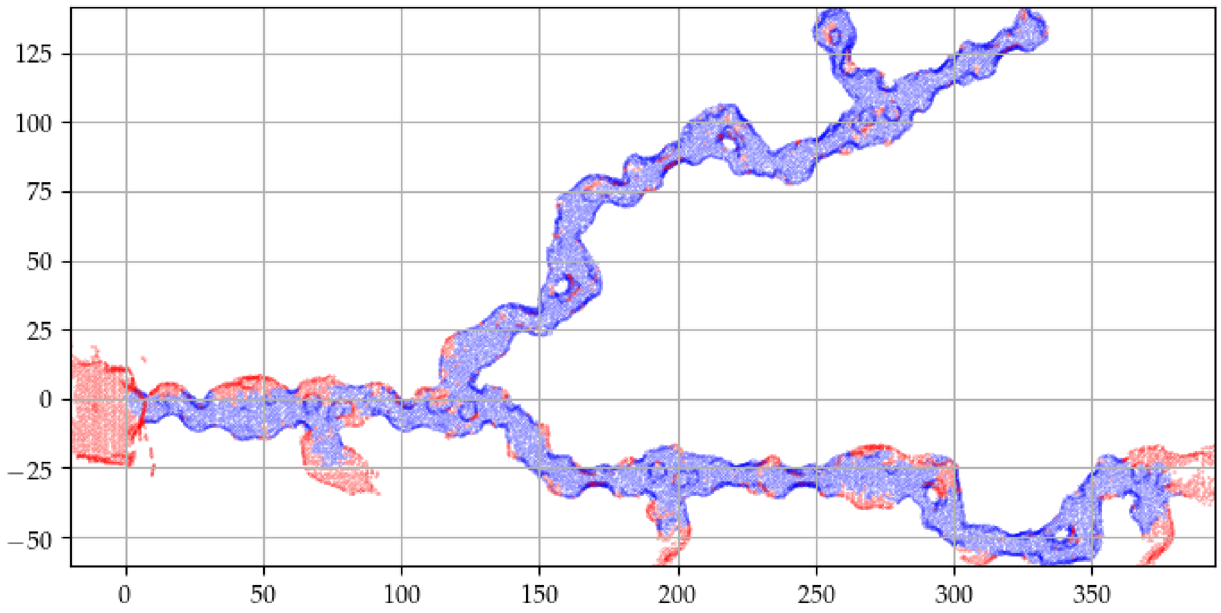

Figure 7.6: The map of inspected (blue) and uninspected (red) facets in the FacetMap projected onto the XY plane at the end of the 3. run of the DEI strategy. The distances are in meters.

7.3 Cooperation of a UGV and UAVs

During the work on this thesis, the strategies were periodically evaluated in simulation with a team comprised of UAVs and UGVs, where we only controlled the UGVs. As the mapping and sharing structures designed in this work are able to be used with any robot that provides odometry and an OOT, the FacetMap, FrontierMap and SegMap mapping structures and the sharing of L-SegMaps were integrated onto the UGVs in simulation, and one of the test runs with the UGVs sharing L-SegMaps is presented in this section.

In this experiment, which is illustrated in Figure 7.7, the UGV entered the environment first and on the first crossroad in the environment, explored the right hand side branch, but could not explore it completely, as there is a tall vertical tunnel in the branch, and went to explore further into the second branch of the environment.

The UGV deploys a trail of communication nodes, and as such, it was sharing its current L-SegMap with the UAVs even when it was far from the staging area. When the UAV1 launched, the nearest globally unexplored frontier was in the vertical tunnel of the right hand side branch of the first crossroad, and therefore it went to explore it. After exploring it, the only frontiers were near the UGV and the UAV went to explore them. However, since a slower, earlier version of the RHNBV strategy was used in this experiment, the battery of the UAV was exhausted upon reaching the next crossroad, which happened also to the other UAVs. Nonetheless, this experiment has demonstrated the applicability of the mapping and sharing methods to other robots than only UAVs.

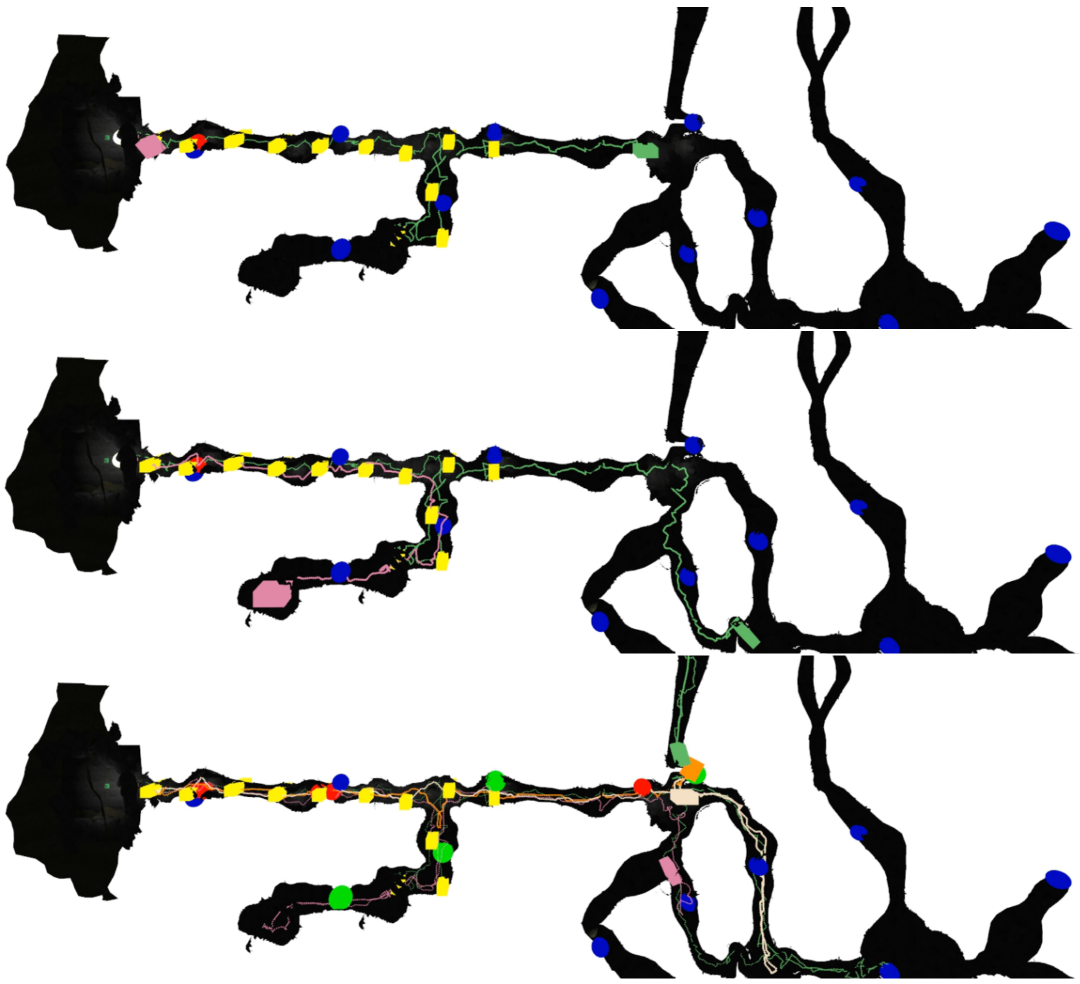

Figure 7.7: An illustration of cooperative exploration of a cave environment with UAVs (pink, orange, white) and a UGV (green). The UGV explores the ground level of the right branch and moves further into the cave (top), then UAV1(pink) explores the upper part of the right branch (middle) and then UAV1 and the remaining UAVs attempt to explore further in the cave but crash (right). Note that the UAVs do not explore the right branch again after UAV1 has explored it. A video of the run is also available in the multimedia materials at http://mrs.felk.cvut.cz/musil2021thesis

7.4 Comparison of Cooperative and Noncooperative Approaches

To compare cooperative and noncooperative variants of the designed strategies, three test runs were executed for a team of three UAVs using the cooperative variant of the RHNBV strategy and three runs using the single-UAV variant of the RHNBV strategy. As with the evaluation of single-UAV strategies, the experiments were executed on the CloudSim servers provided by DARPA.

To demonstrate the main differences of the approaches, one map is shown for the cooperative variant and one for the noncooperative variant. Maps from the rest of the runs can be found in the full results of experiments.

The main difference can be seen in that in the noncooperative run shown in Figure 7.8, the UAVs explored muliple rooms multiple times. The fact that the UAVs went separate directions at the start of the mission without cooperation is by pure luck. In the cooperative run shown in Figure 7.9 the UAVs did not explore much of the space explored by other UAVs as in the noncooperative run, which could, however, have been affected by the fact that the UAV1 and UAV3 crashed into one another. This was caused by a situation, where the UAVs met in the narrow corridor in the middle of the enviornment, exchanged information about the environment and for both of the UAVs, the nearest globally unexplored space was in the left branch of the corridor. The UAVs went to explore this frontier, each from a different side, but due to momentum, they tried to explore frontiers further in the corridor and crashed into one another. This could be solved in the future by sharing the current goals of UAVs and also by recalculating the current goal at a higher frequency when another robot is nearby.

7.5 Experiment on a real UAV

One of the goals of this thesis was to prepare the planning and mapping algorithms for deployment on a real UAV using the MRS system. The algorithms were prepared and then tested in a single-UAV mission in a harsh and dangerous environment on a UAV equipped with an Ouster OS0-128 LiDar, a Bluefox MLC200wc camera, two Realsense D435i depth cameras (one facing up, one facing down) and a NUC10i7fnk with 16GB RAM. Videos and photos from this experiment are available in the multimedia materials at http://mrs.felk.cvut.cz/musil2021thesis. The environment chosen for the real-world experiment was a dilapidated former beer brewery. This building consists of a large $\approx 8\,\mathrm{m}$ tall room on ground level with multiple large windows and pillars and also an underground level filled with small pillars that at some point were only $\approx 2\,\mathrm{m}$ apart, low ceilings and hanging chains, small pipes and water puddles. The large upper room is connected with the basement by multiple $\approx 3\,\mathrm{m}$ wide holes in the ground, shown in Figure 7.10.

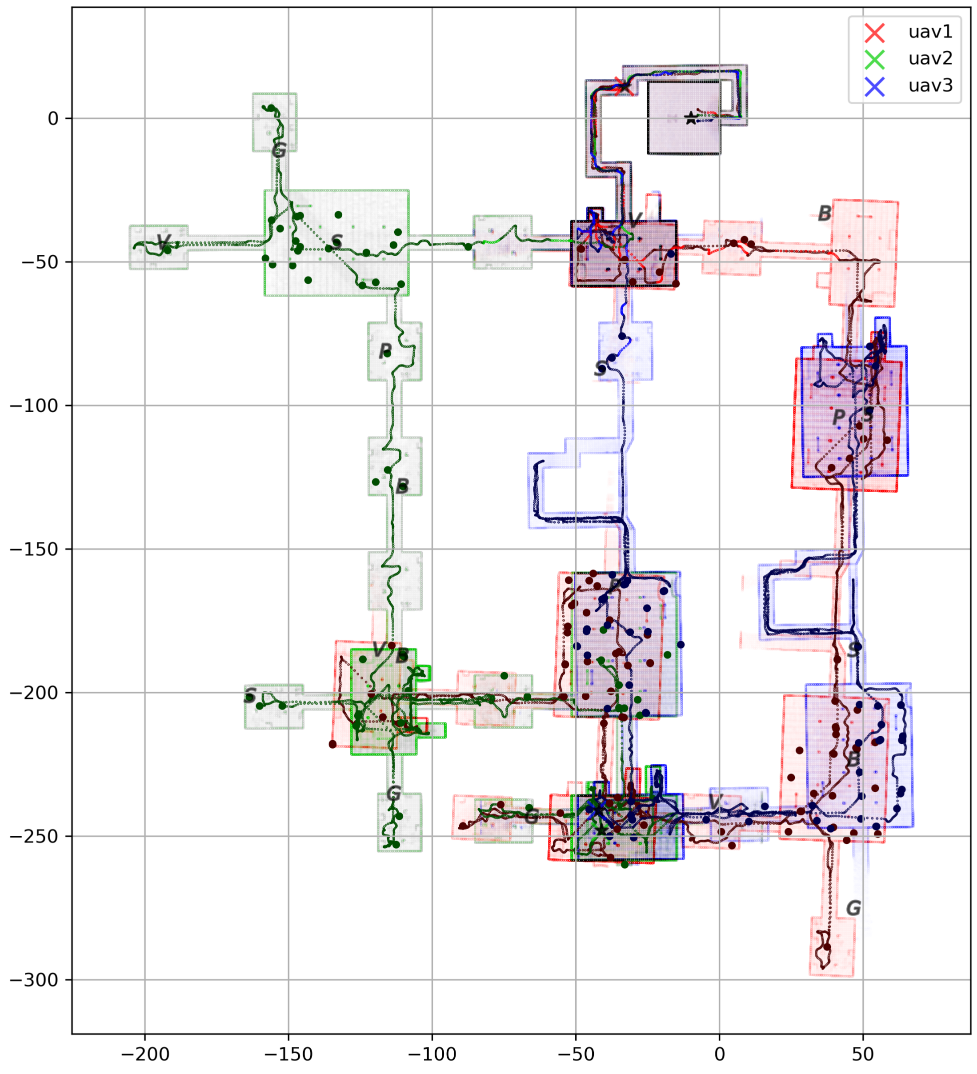

Figure 7.8: Map of the Urban Circuit Practice 3 world explored by 3 UAVs using the noncooperative variant of the RHNBV strategy. The distances in this map are in meters. The map is composed of overlapped OOTs from each robot. These OOTs are misaligned due to localization drift.

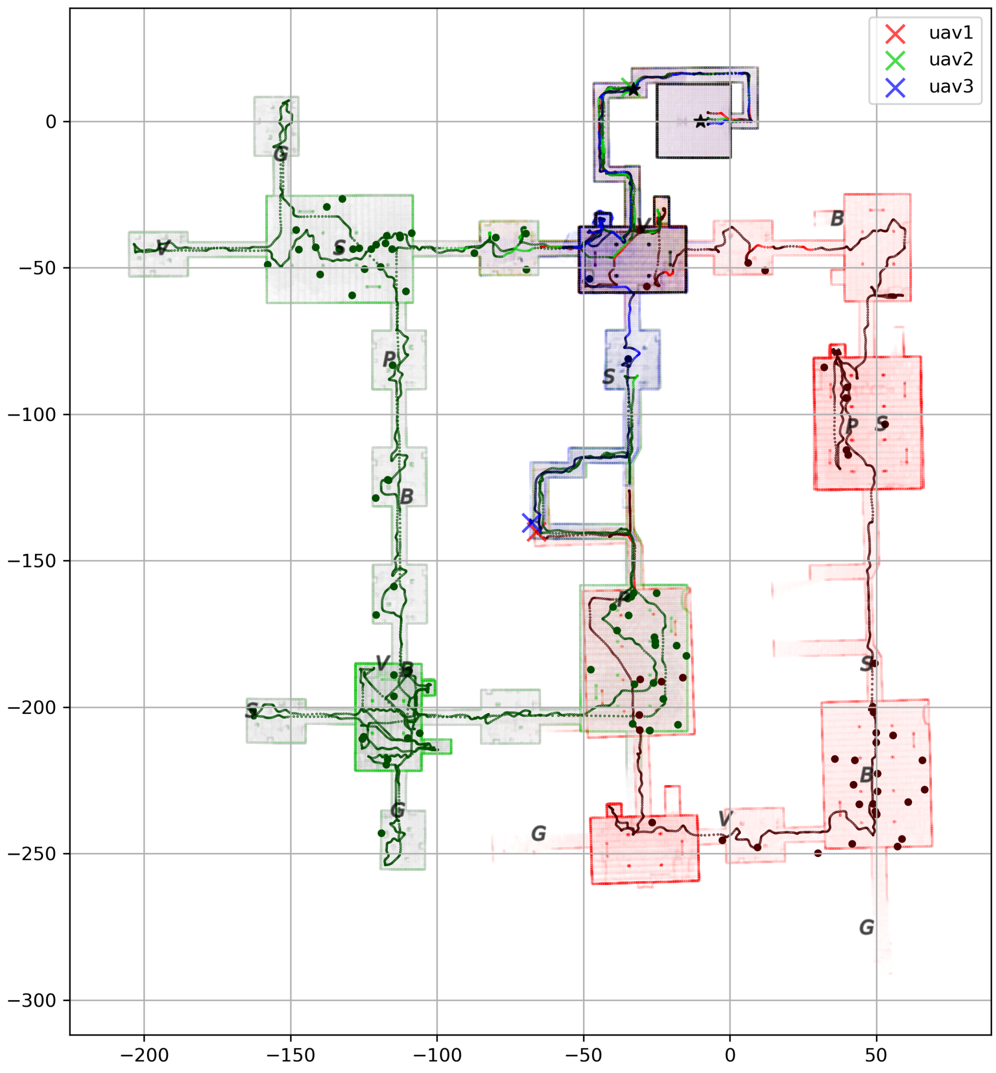

Figure 7.9: Map of the Urban Circuit Practice 3 world explored by 3 UAVs using the cooperative variant of the RHNBV strategy. The distances in this map are in meters. The map is composed of overlapped OOTs from each robot. These OOTs are misaligned due to localization drift.

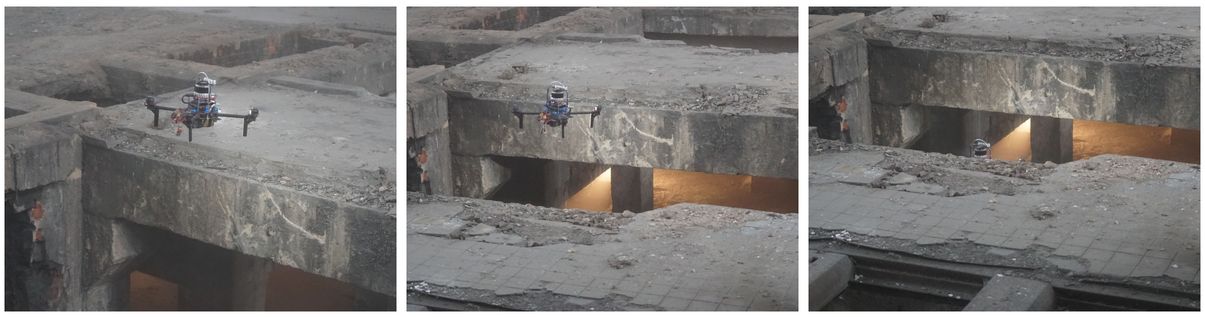

Figure 7.10: The UAV moving through a hole to explore frontiers in the lower level of the brewery.

The main danger of this environment was in the extreme amounts of dust that had accumulated there over the years. Even with intensity based LIDAR scan filtration, the OOT created onboard the UAV was very noisy. The main problem was that after the UAV had taken off in the upper room, the floor in the OOT, started to slowly rise due to large swaths of dust being lifted by the UAV. This situation is shown in Figure 7.12. This caused that new facets were created above the facets first created and explored on the ground, which was expected with such extreme noise. However, even with such extremely noisy OOT data, both the FacetMap and SegMap produced satisfying results and allowed the search algorithms to function as intended.

Two main experiments were conducted, in both of which the 3rd strategy - Dead End Inspection - was employed. The parameters α_S of the strategy was set to $\alpha_S = -30$. Since the environment was small, the feature of this strategy searching from the end of a dead end back towards home could not have been evaluated. Instead, the strategy, according to the reward functions, first explored frontiers and when there were viewpoints with high surface information value and a high bonus from small heading change, it started to prefer the viewpoints, as expected. The environment, as pereceived by the UAV and the real image of the UAV at roughly the same time are shown on Figure 7.11

In the first experiment, the operating area of the UAV was constrained to a height of $h \in (-0.5\,\mathrm{m}, 8\,\mathrm{m})$. As expected, the UAV explored frontier viewpoints for the first few minutes and then started to choose surface inspection viewpoints over frontier exploration viewpoints. Interestingly, even with the constrained minimal height, the UAV dipped into one of the holes connecting the two rooms to clear frontiers.

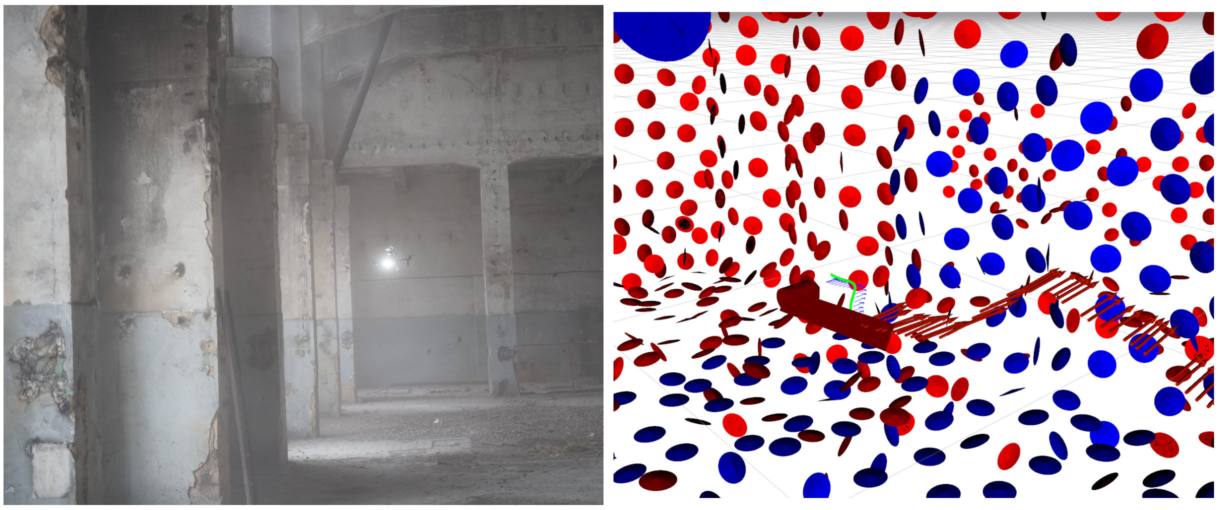
Figure 7.11: The UAV inspecting surfaces on the wall of the brewery's upper room both on camera (left) and at approximately the same time in RVIZ (right)

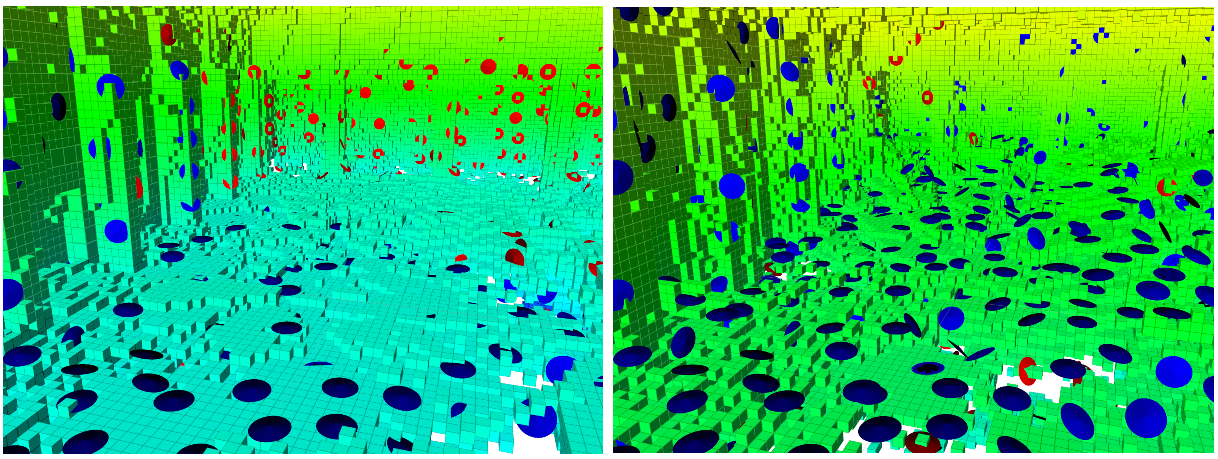
Figure 7.12: The effects of extreme noise from dust on the FacetMap - OOT and FacetMap soon after start of flight (left) and the OOT lifted by the effect of dust and hence additionally created facets (right)

Chapter 8

Conclusion

Contents

The main goal of this thesis was to design and implement multiple high level planning methods and mapping structures based on an occupancy octree (OOT) for teams of autonomous UAVs in the DARPA SubT Challenge.

Firstly, an online mapping structure for building a surface representation during the mission, storing information about coverage of surfaces, and for the generation of surface inspection viewpoints — the FacetMap — was designed and implemented in chapter 2. Secondly, a topology mapping structure — the SegMap — built upon the principles of the TopoMap [22], and enhanced with the ability to be computed online was designed and implemented in chapter 3. This topological mapping structure has demonstrated the ability for high-speed long-distance path planning and travel distance estimation. Thridly, for the task of classic robotic volumetric exploration, a structure for extracting and storing information about frontiers in the environment — the FrontierMap — was designed and implemented in chapter 4.

A framework for compressing the designed maps into a lightweight map, called an L-SegMap in this thesis, sharing L-SegMaps among robots, and updating L-SegMaps based on L-SegMaps from other robots was also designed and implemented in chapter 5.

Three high level planning strategies that combine volumetric exploration methods and surface inspection methods were designed and implemented in chapter 6 and evaluated in chapter 7. These strategies were also enhanced with cooperative approaches that utilize the shared L-SegMaps. The first strategy — RHNBV — is an application of the receding horizon next best view method [17], which, when combined with the designed mapping structures, has demonstrated high speed exploration.

The RHNBV strategy was further enhanced with surface inspection methods in the

VPE strategy, which performs receding horizon exploration in the same manner as the RHNBV strategy but also adds additional surface inspection viewpoints along the path to frontier exploration viewpoints.

Lastly, a novel strategy — DEI — was proposed in this thesis. This strategy performs frontier exploration until it reaches a dead end, and then begins to systematically inspect all surfaces from the dead end back to the best next frontier. This strategy appears to be the most promising for multi-robot exploration, as when a branch of the environment is completely searched, all other robots can be stopped from searching that area again.

The designed strategies were integrated into the MRS system and evaluated in single-UAV missions in the realistic Gazebo simulator, where they showed the ability to explore large areas of a cave environment. The cooperative extension of the RHNBV strategy was also demonstrated on multiple test runs in simulation and compared with the noncooperative version of the RHNBV strategy for a team of 3 UAVs.

Furthermore, the DEI strategy was deployed on a real UAV platform in a harsh environment, where it demonstrated the robustness of the mapping structures and the high-level planning architecture, and successfully searched the environment.

In addition to the original assignment of this thesis, the mapping structures and the map sharing framework were implemented as a C++ ROS based library that can be used on any robot that is building an OOT. The benefit of this was demonstrated in simulation in an experiment with a UGV and multiple UAVs where the UGV was building the designed mapping structures and sharing L-SegMaps with the UAVs.

8.1 Future Work

Many areas of the strategies and mapping structures proposed in this thesis leave room for improvement. Most importantly, a simple mechanism for the sharing of the current goal of each robot in the shared L-SegMaps could drastically reduce the chance of the UAVs simultaneously exploring the same area, which can happen in the current state of the cooperative strategies. The SegMap could also be enhanced with methods that would allow it to adapt to a dynamic environment (for example with gates closing or passages collapsing) and the problem of thin walls in the FacetMap could also be solved in future work. The area of future work most interesting for the author would is to design cooperative strategies that would better utilize the surface coverage information shared in the L-SegMaps for each segment (for example to explore frontiers that lead to areas with uninspected areas rather than only to explore frontiers that lead to globally unexplored frontiers) and to broach the subject of multi-agent planning.

Bibliography

[1] P. Petráček, V. Krátký, and M. Saska, "Dronument: System for reliable deployment of micro aerial vehicles in dark areas of large historical monuments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2078–2085, April 2020.

[2] V. Krátký, P. Petráček, V. Spurný, and M. Saska, "Autonomous reflectance transformation imaging by a team of unmanned aerial vehicles," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2302–2309, April 2020.

[3] V. Spurny, V. Pritzl, V. Walter, M. Petrlik, T. Baca, P. Stepan, D. Zaitlik, and M. Saska, "Autonomous Firefighting Inside Buildings by an Unmanned Aerial Vehicle," *IEEE Access*, vol. 9, pp. 15 872–15 890, 2021.

[4] A. Ahmad, V. Walter, P. Petracek, M. Petrlik, T. Baca, D. Zaitlik, and M. Saska, "Autonomous aerial swarming in gnss-denied environments with high obstacle density," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[5] G. Silano, J. Bednar, T. Nascimento, J. Capitan, M. Saska, and A. Ollero, "A Multi-Layer Software Architecture for Aerial Cognitive Multi-Robot Systems in Power Line Inspection Tasks," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2021.

[6] M. Terlizzi, G. Silano, L. Russo, M. Aatif, A. Basiri, V. Mariani, L. Iannelli, and L. Glielmo, "A Vision-Based Algorithm for a Path Following Problem ," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS).*, June 2021.

[7] J. P. Queralta, J. Taipalmaa, B. Can Pullinen, V. K. Sarker, T. Nguyen Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," *IEEE Access*, vol. 8, pp. 191 617–191 643, 2020.

[8] T. Rouček, M. Pecka, P. Čížek, T. Petříček, J. Bayer, V. Šalanský, D. Heřt, M. Petrlík, T. Báča, V. Spurný, F. Pomerleau, V. Kubelka, J. Faigl, K. Zimmermann, M. Saska, T. Svoboda, and T. Krajník, "Darpa subterranean challenge: Multi-robotic exploration of underground environments," in *Modelling and Simulation for Autonomous Systems*, 2019, pp. 274–290.

[9] V. Kratky, P. Petracek, T. Baca, and M. Saska, “An Autonomous Unmanned Aerial Vehicle System for Fast Exploration of Large Complex Indoor Environments,” *Accepted to Journal of Field Robotics*, 2021.

[10] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, “Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems,” vol. 2, 01 2010.

[11] J. A. Orenstein, “Multidimensional tries used for associative searching,” *Information Processing Letters*, vol. 14, no. 4, pp. 150–157, 1982.

[12] A. Bircher, M. S. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, “Receding horizon path planning for 3d exploration and surface inspection,” *Autonomous Robots*, vol. 42, 02 2018.

[13] T. Dang, F. Mascarich, S. Khattak, C. Papachristos, and K. Alexis, “Graph-based path planning for autonomous robotic exploration in subterranean environments,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 3105–3112.

[14] B. Zhou, Y. Zhang, X. Chen, and S. Shen, “Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 779–786, 2021.

[15] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, “Collaborative multi-robot exploration,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, 2000, pp. 476–481 vol.1.

[16] L. Fermin-Leon, J. Neira, and J. A. Castellanos, “Tigre: Topological graph based robotic exploration,” in *2017 European Conference on Mobile Robots (ECMR)*, 2017, pp. 1–6.

[17] B. Yamauchi, “A frontier-based approach for autonomous exploration,” in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA’97. ’Towards New Computational Principles for Robotics and Automation’*, 1997, pp. 146–151.

[18] S. M. Lavalle, “Rapidly-exploring random trees: A new tool for path planning,” Tech. Rep., 1998.

[19] D. Silver, D. Ferguson, A. Morris, and S. Thayer, “Topological exploration of subterranean environments,” *Journal of Field Robotics*, vol. 23, no. 6-7, pp. 395–415, 2006.

[20] H. Choset and K. Nagatani, “Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 2, pp. 125–137, 2001.

[21] R. Bormann, F. Jordan, W. Li, J. Hampp, and M. Hägele, "Room segmentation: Survey, implementation, and analysis," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1019–1026.

[22] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological mapping and navigation based on visual slam maps," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3818–3825.

[23] I. Maza and A. Ollero, *Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms*, 01 2007, vol. 6, pp. 221–230.

[24] A. Zelinsky, R. Jarvis, J. Byrne, and S. Yuta, "Planning paths of complete coverage of an unstructured environment by a mobile robot," 2007.

[25] B. Bogaerts, S. Sels, S. Vanlanduit, and R. Penne, "Near-optimal path planning for complex robotic inspection tasks," 05 2019.

[26] T. Danner and L. Kavraki, "Randomized planning for short inspection paths," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 971–976 vol.2.

[27] T. Dang, C. Papachristos, and K. Alexis, "Visual saliency-aware receding horizon autonomous exploration with application to aerial robotics," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2526–2533.

[28] J. Delmerico, E. Mueggler, J. Nitsch, and D. Scaramuzza, "Active autonomous aerial exploration for ground robot path planning," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 664–671, 2017.

[29] K. M. Wurm, C. Stachniss, and W. Burgard, "Coordinated multi-robot exploration using a segmentation of the environment," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1160–1165.

[30] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[31] K. Cesare, R. Skeele, S.-H. Yoo, Y. Zhang, and G. Hollinger, "Multi-uav exploration with limited communication and battery," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2230–2235.

[32] F. Ropero, P. Muñoz, and M. D. R-Moreno, "Terra: A path planning algorithm for cooperative ugv–uav exploration," *Engineering Applications of Artificial Intelligence*, vol. 78, pp. 260–272, 2019.

[33] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, p. 469–483, Dec. 1996.

[34] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD’96. AAAI Press, 1996, p. 226–231.

Appendices

Full Experiment Results

The full results of the single-UAV evaluation and cooperative vs noncooperative approach comparison are presented in this section.

Strategy	Run	V_{kno}[m^3]	$S_{insp}[-]$	$S_{kno}[-]$	$p_{insp}[-]$	$p_{spv}[-]$	v_{exp}[m/s]	$A[-]$
RH-NBV	1	86436.67	11365	14709	0.773	0.131	1.64	4
RH-NBV	2	104074.09	12945	17569	0.737	0.124	1.65	5
RH-NBV	3	96716.05	12211	16462	0.742	0.126	1.63	3
VPE	1	94979.83	11335	15709	0.722	0.119	1.46	2
VPE	2	86519.58	11086	14168	0.782	0.128	1.55	2
VPE	3	73204.14	7597	11591	0.655	0.104	1.59	3
DEI	1	82531.66	10633	13775	0.772	0.129	1.63	4
DEI	2	73477.19	9906	12399	0.799	0.135	1.59	4
DEI	3	91783.08	11617	15352	0.757	0.127	1.55	5

Table 1: Full results of the single-UAV strategy evaluation. The meanings of the evaluated metrics can be found in subsection 7.2.1

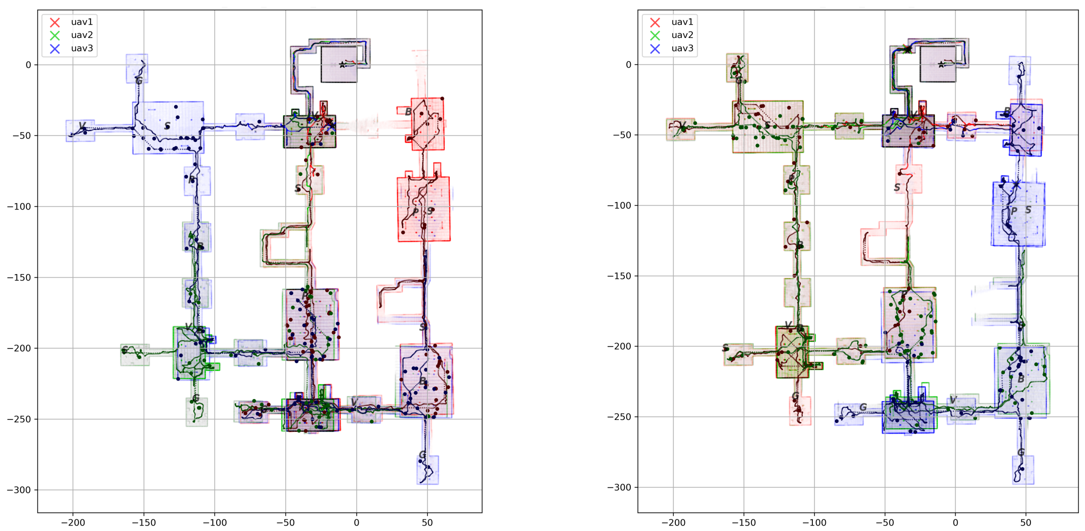

Figure 1: The two runs of the cooperative RHNBV strategy on the Urban Circuit3 world that are not shown in chapter 7.

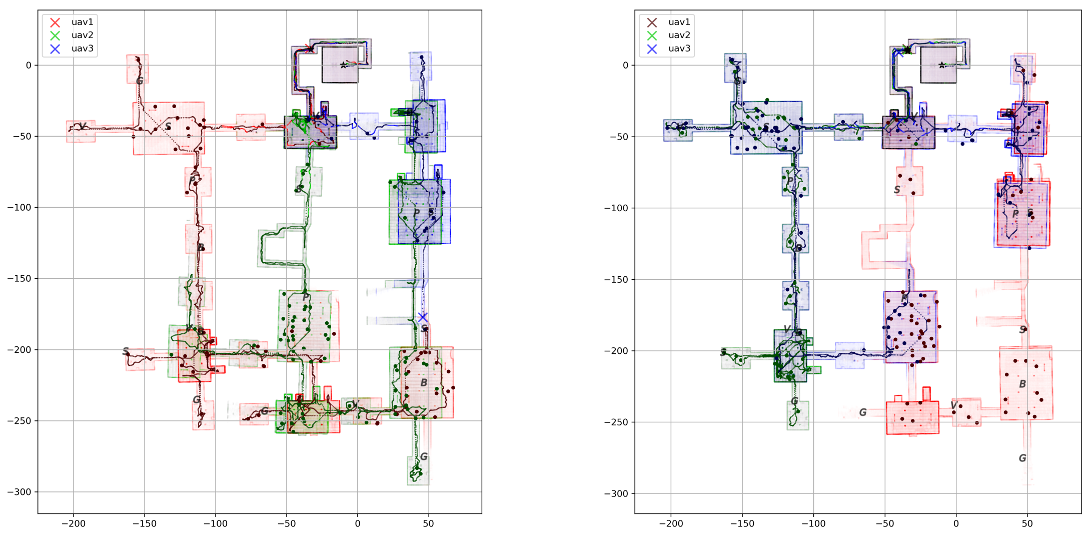

Figure 2: The two runs of the noncooperative RHNBV strategy on the Urban Circuit3 world that are not shown in chapter 7.

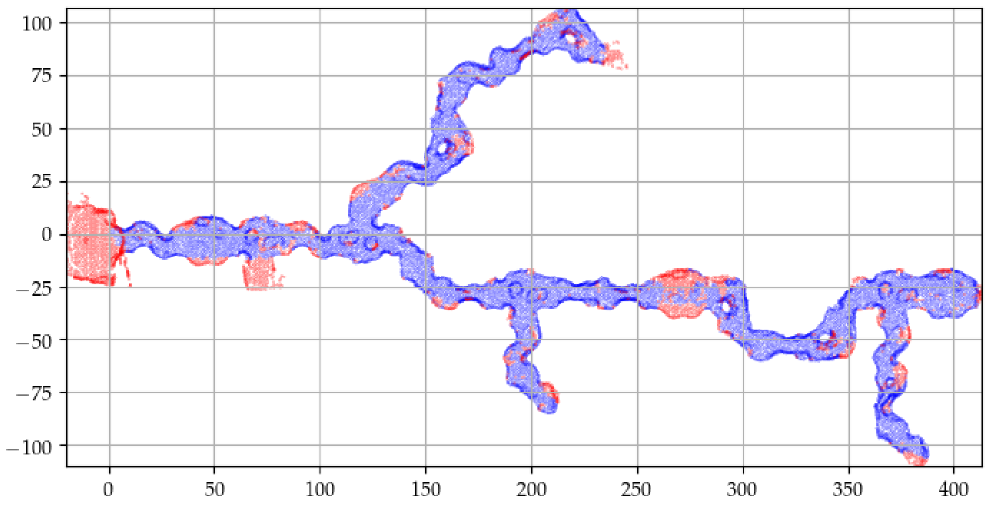

Figure 3: The map of the 1. run of the RHNBV strategy in the single-UAV strategy evaluation.

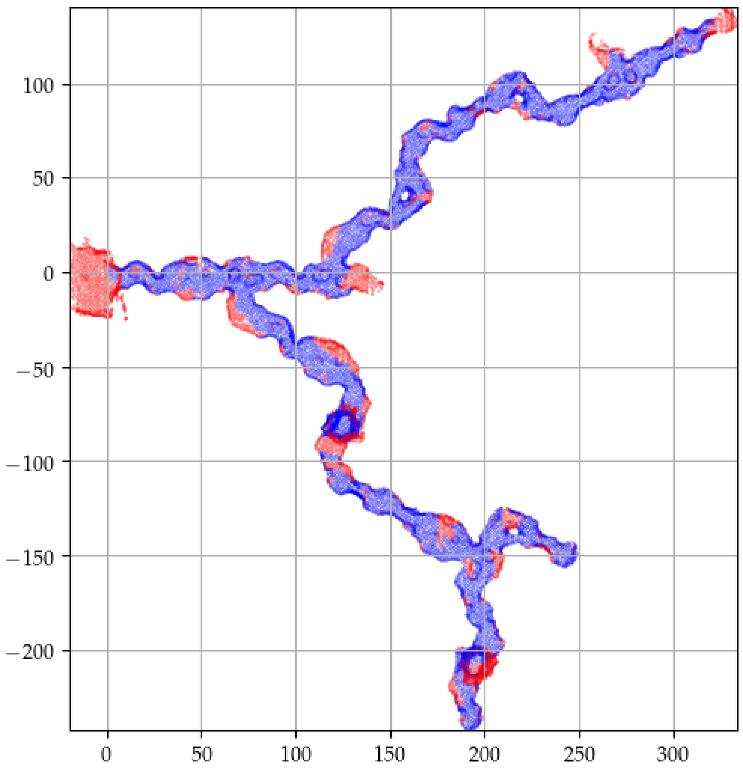

Figure 4: The map of the 3. run of the RHNBV strategy in the single-UAV strategy evaluation.

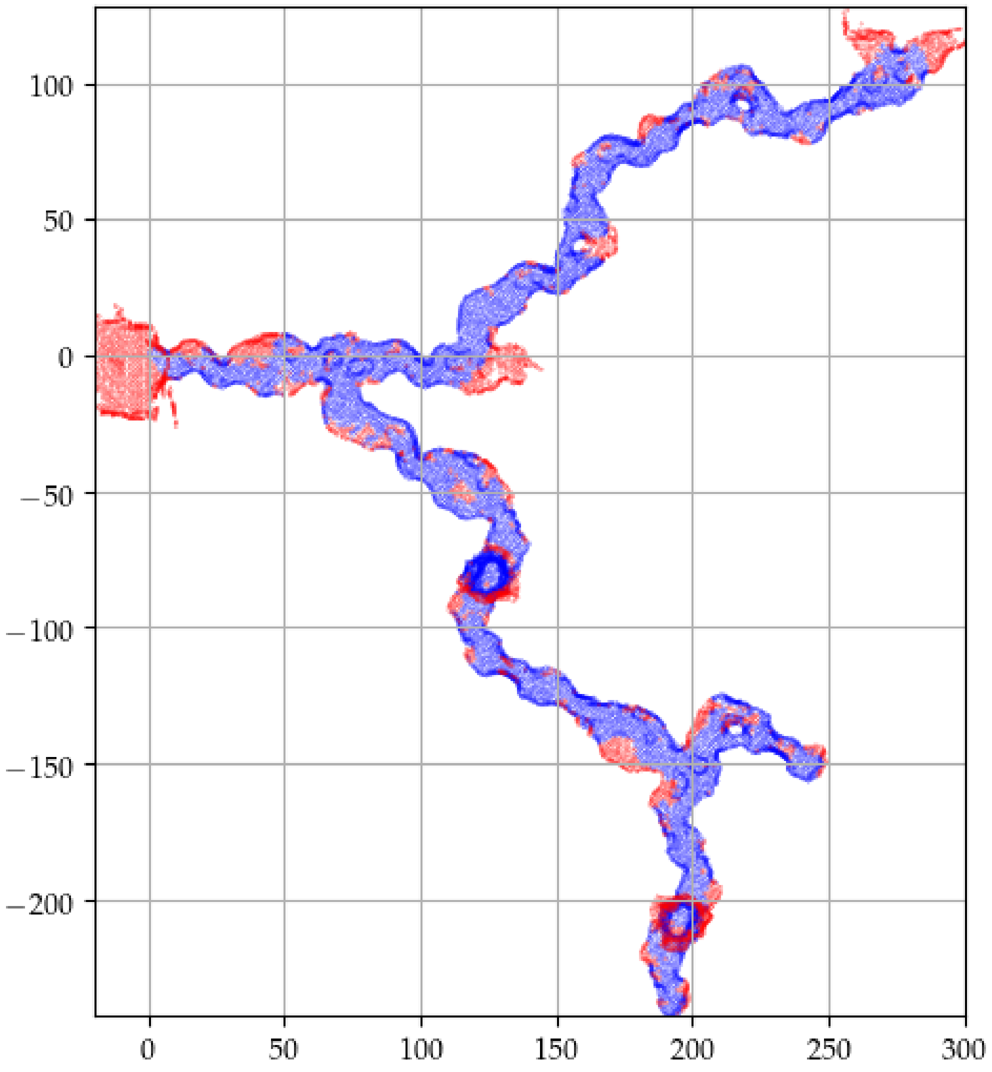

Figure 5: The map of the 1. run of the VPE strategy in the single-UAV strategy evaluation.

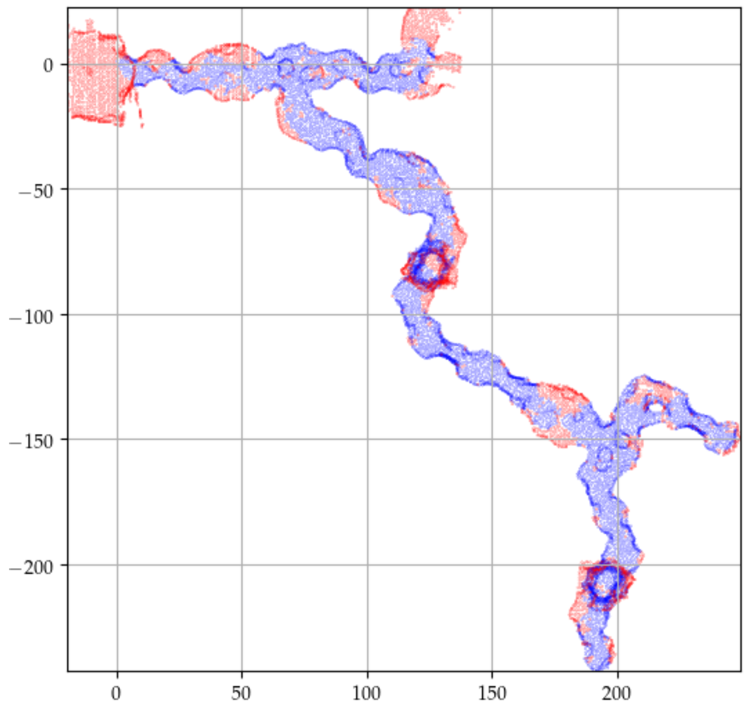

Figure 6: The map of the 3. run of the VPE strategy in the single-UAV strategy evaluation.

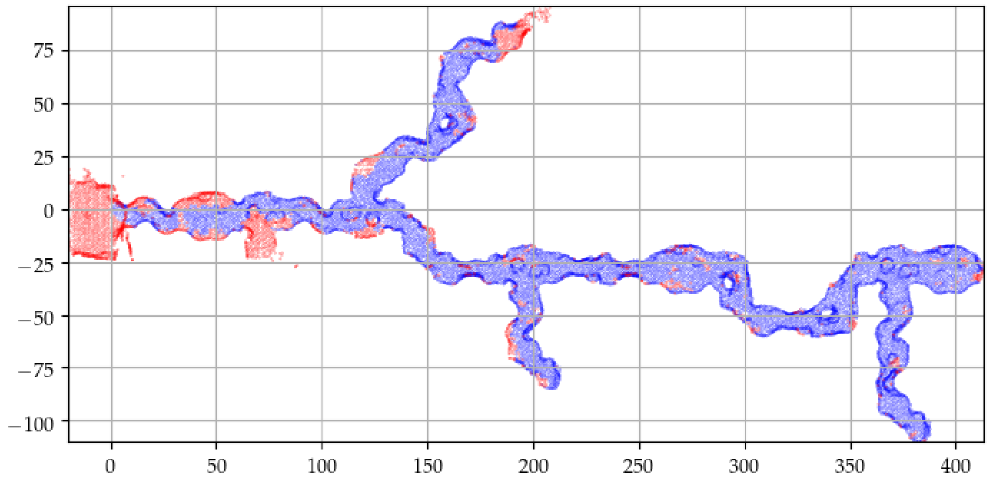

Figure 7: The map of the 1. run of the DEI strategy in the single-UAV strategy evaluation.

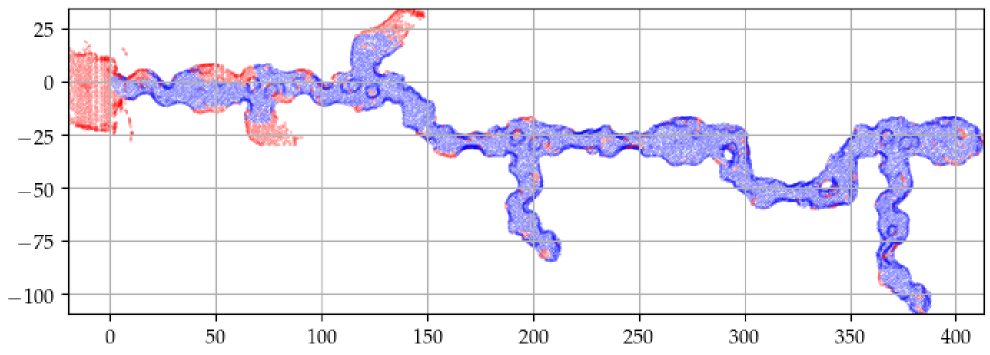

Figure 8: The map of the 2. run of the DEI strategy in the single-UAV strategy evaluation.

CD Content

In Table 2, the names of all root directories on the attached CD are listed.

Directory name	**Description**
thesis	the thesis in pdf format
src/thesis	latex source codes
src/search_planning	software source codes
media	multimedia materials

Table 2: CD Content

List of abbreviations

In Table 3 are listed abbreviations used in this thesis.

Abbreviation	**Meaning**
SAR	search and rescue
UAV	unmanned aerial vehicle
UGV	unmanned ground vehicle
GNSS	global navigation satellite system
OOT	occupancy octree
PCA	principal component analysis
SLAM	simultaneous localization and mapping
RHNBV	receding horizon next best view strategy
VPE	viewpoint enhancement strategy
DEI	dead end inspection strategy

Table 3: Lists of abbreviations