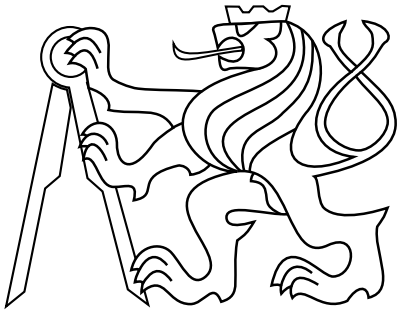
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



Volumetric Mapping Onboard Unmanned Helicopter

Bachelor's Thesis

David Čapek

Prague, May 2023

Study programme: Cybernetics and Robotics
Supervisor: Ing. Tomáš Báča, Ph.D.

Author Statement

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 26 May 2023 .....

David Čapek

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Čapek David** Personal ID number: **499116**

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:

Volumetric Mapping Onboard Unmanned Helicopter

Bachelor’s thesis title in Czech:

Volumetrické mapování a plánování pro bezpilotní helikoptéru

Guidelines:

Autonomous mapping is one of the key building blocks of autonomous mobile robotic systems. A map of an environment is a fundamental structure used for autonomous path planning, which is essential for the autonomous navigation of mobile robots in an unstructured environment. This thesis will focus on revisiting the basic principles of robotic mapping and planning for drones and using existing state-of-the-art 3D mapping tools. Furthermore, the student will attempt to implement a custom 3D planner to plan a collision-free path to be used with a 3D mapper of his choice. The student will validate his solution using the MRS UAV System with the Robot Operating System (ROS).
The thesis will be divided into the following parts:
The student will familiarize himself with ROS and the MRS UAV System [1].
The student will research state-of-the-art methods for 3D volumetric mapping [2, 3, 4, 5].
The student will implement or adapt selected mapping frameworks using the MRS UAV System.
The student will design and implement a 3D path planning algorithm using a select mapper.
The student will evaluate the mappers and compare their properties using a provided dataset.
The developed system will be tested experimentally in the Gazebo/ROS robotic simulator. Moreover, if the experimental hardware allows it, the student will conduct experimental validation in a real environment

Bibliography / sources:

[1] Baca, Tomas, et al. "The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles." Journal of Intelligent & Robotic Systems 102.1 (2021): 1-28.
[2] Hornung, Armin, et al. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees." Autonomous robots 34.3 (2013): 189-206.
[3] Duberg, Daniel, and Patric Jensfelt. "UFOMap: An efficient probabilistic 3D mapping framework that embraces the unknown." IEEE Robotics and Automation Letters 5.4 (2020): 6411-6418.
[4] Oleynikova, Helen, et al. "Voxblox: Incremental 3d euclidean signed distance fields for on-board MAV planning." 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017.
[5] Reijgwart, Victor, et al. "Voxgraph: Globally consistent, volumetric mapping using signed distance function submaps." IEEE Robotics and Automation Letters 5.1 (2019): 227-234.

Name and workplace of bachelor's thesis supervisor:

Ing. Tomáš Báča, Ph.D. Multi-robot Systems FEE

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

Date of bachelor's thesis assignment: **18.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Tomáš Báča, Ph.D. Supervisor's signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. 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

Acknowledgments

I would like to thank my supervisor, Ing. Tomáš Báča, Ph.D., for the great guidance, insightful advice, and the wealth of knowledge he has given me throughout the work on this thesis.

My thanks also go the members of the MRS group for answering all my questions, and for the help they provided during the testing of this thesis in real-world.

Finally, my gratitude goes to my family, my girlfriend and my friends for their unwavering encouragement and support during my studies.

Abstract

This thesis focuses on the development of a mapping and planning pipeline for an Unmanned Aerial Vehicle. The thesis presents the core concepts required for creating volumetric maps. The proposed pipeline focuses on creating a consistent global volumetric map, that allows for global path planning. The strategy being utilized to create a globally consistent map, involved combining two state-of-the-art mapping frameworks. A partially exploratory planning manager is presented, that utilizes the available volumetric maps to generate a path towards a set goal, while continuously conducting checks for possible future collisions. The proposed pipeline is implemented and extensively tested in simulation, on a dataset and in real-world experiments, resulting in positive outcomes.

Keywords Unmanned Aerial Vehicles, Volumetric mapping, Path planning, ROS

Abstrakt

Tato práce se zaměřuje na vývoj mapovací a plánovací pipeliny pro bezpilotní helikoptéru. Práce představuje základní koncepty potřebné pro tvorbu volumetrických map. Navrhovaná pipeline se zaměřuje na vytvoření konzistentní globální volumetrické mapy, která umožňuje globální plánování cesty. Strategie použitá k vytvoření globálně konzistentní mapy zahrnuje kombinaci dvou nejmodernějších mapovacích frameworků. Představen je částečně průzkumný plánovací manažer, který využívá dostupné volumetrické mapy k vytvoření cesty k zadanému cíli, přičemž průběžně provádí kontrolu možných budoucích kolizí. Navržený postup je implementován a rozsáhle testován v simulaci, na datasetu a při reálných experimentech, které vedly k pozitivním výsledkům.

Klíčová slova Bezpilotní Prostředky, Volumetrické mapování, Plánování cest, ROS

Abbreviations

ESDF Euclidean Signed Distance Field

GPS Global Positioning System

ICP Iterative Closest Point

IMU Inertial Measurement Unit

LiDAR Light Detection and Ranging

MRS Multi-robot Systems Group

OcTree Octal Tree

PC Point Cloud

ROS Robot Operating System

RTK Real-time Kinematics

SDF Signed Distance Field

SLAM Simultaneous Localization And Mapping

TSDF Truncated Signed Distance Field

UAV Unmanned Aerial Vehicle

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Over the past decade, there has been a remarkable growth in various fields linked to autonomous robotic systems. The advances make robotic systems easier to deploy for different tasks. Robotic systems are being deployed to allow computers to perform tasks in physical environments. Among robotic systems small Unmanned Aerial Vehicles (UAVs) have gathered significant attention and currently are at the forefront of research due to their versatility. Unlike ground robots multirotor helicopter UAVs are able to traverse 3D environments efficiently while also being able to hover in place. These two factors make UAVs ideal for developing robotic systems upon. The UAV used in this thesis is shown in Figure 1.1.

Figure 1.1: Photo of the UAV used in the thesis[1].

The UAV systems are being used in a number of unique scenarios such as mapping cultural heritage sites [2], localization of radiation sources [12] or subterranean search and rescue operations [3]. The mentioned systems perform highly specialized jobs, however they all rely on a volumetric representation of the environment they operate in for planning of their motion. The task of creating an accurate representation of the operating environment is the backbone of every autonomous system. Without an accurate volumetric map, it is more complex to efficiently plan autonomous missions in large environments.

To efficiently deploy autonomous UAVs, a complete pipeline that connects all the key components used for mapping and planning must be created. In this context, this thesis focuses on the creation of the pipeline by combining current state-of-the-art approaches to mapping and planning.

[1] https://fel.cvut.cz/aktualne/novinky/2023/04/mrs/15563/image-thumb__15563__FullImage/10_mrs_print20x30_fotoneugebauerpetr_2023.jpg

1.1 Related works

The field of developing autonomous systems is currently being heavily researched. In the context of this thesis, the two main research fields are mapping the operational environment and being able to safely navigate to a desired position.

1.1.1 Volumetric mapping

In the field of volumetric mapping, numerous research efforts have been undertaken to develop robust and consistent techniques of representing the environment. With the increased affordability of UAVs, that require three-dimensional representations compared to ground robots, significant progress has been made in the field of volumetric maps. The main goal of volumetric mapping is creating accurate and memory efficient representations of the environment from a vast amount of noisy measurements. Different approaches have been proposed to tackle the problem. Most approaches generating a dense reconstruction discretize the space into occupancy grids, consisting of 3D cells.

One way of representing the occupancy grid is by using an Octal Tree (OcTree) presented in OctoMap [32]. OctoMap is a probabilistic occupancy map framework designed for low memory footprint. The space that has to be mapped is divided into octants which then recursively subdivide into more octants. The created representation is hierarchical and allows for multi-resolution mapping. Since the integration of new data was inefficient, in [26] the authors proposed an improvement over OctoMap. Main speed improvements were achieved by early terminating ray tracing and a new volume inquire method. The authors of [15] proposed a mapping framework based on two-tier OcTrees with adaptive resolution mapping. The two-tier OcTree enables the dynamic selection of resolution, so that the underlying continuous function is accurately represented up to a certain point. VDB mapping [14] swaps the OcTree representation for an alternative hierarchical data structure, OpenVDB [33]. OpenVDB was first developed for computer graphics and rendering, and thus contains highly optimized routines for ray casting, this significantly improves the speed of data insertion.

The second most popular way of representing the occupancy grid is by using Signed Distance Field (SDF). Compared to the methods above using OcTrees SDF do not represent the probability that a cell is occupied but models a continuous distance function. The pioneer publication was KinectFusion [37], which presented a mapping framework for small environments, based on Truncated Signed Distance Fields (TSDFs). TSDFs can achieve sub-cell resolution in surface reconstruction and allow for smoothing of sensor noise. The main idea introduced in KinectFusion was the incremental updating of the TSDF. Numerous works extend KinectFusion, like Chisel [30], Voxblox [28] and Voxfield [10]. All of the methods are specifically designed for deployment onboard resource constrained devices like UAVs. Compared to the Chisel, Voxblox and Voxfield that build a TSDF map and then recompute the whole Euclidean Signed Distance Field (ESDF), FIESTA [21] directly builds the ESDF map. FIESTA proposes novel algorithm for updating the ESDF map directly and introdues an efficient data structure for incremental map updates.

Simultaneous Localization And Mapping (SLAM) techniques are also useful for creating a volumetric map. In SLAM frameworks the process of mapping and localization is combined and done concurrently, and as a consequence trying to tackle the chicken and egg problem as a map is often needed for localization and position for mapping. Many different SLAM algorithms have been proposed. In LOAM [31] the authors proposed a method to segment

the problem into two algorithms. One algorithm performing a high frequency but low fidelity odometry estimation. The second algorithm performs fine matching and registration of the pointclouds, which runs significantly slower compared to the other algorithm. By combining these two algorithms, real-time performance of localization and map creation was achieved. LIO-SAM [20] formulates the SLAM problem atop a factor graph allowing multiple different sources of odometry to be incorporated into the system as a factor. Scan matching of the input PC is done at a local scale and then the local maps are saved as keyframes of set size, together they create the global map.

New methods using machine learning are being proposed, these methods are powered by the sudden growth in neural networks. The proposed methods can leverage the constant computation time neural networks provide, while also creating the opportunity to combine multiple modalities. In iSDF [9] the authors present a continual learning system for real-time reconstruction of SDFs. The underlying neural network is continuously learning the representation, and does not require any pretraining. Paper [4] extends the idea of iSDF and creates a submap based framework for mapping and planning larger areas. The framework is able to map multiple indoor rooms based on the submap approach. Authors of AVL map [1] use machine learning to create a multi-modal representation of the environment, using audio, visual and language cues to understand the environment and navigate it. All the machine learning methods mentioned here are in their infancy and are not viable for the usage in large scale outdoor mapping.

1.1.2 Path planning

In the field of path planning, significant research efforts have been dedicated to developing robust and efficient techniques for generating optimal paths in various environments. This section provides an overview of key works that have contributed to the advancements in path planning techniques, with a particular focus on UAV applications. Path planning for UAVs is particularly interesting as their planning space is much larger than for ground robots. The main goal for path finding algorithms is finding a collision-free path in an environment, this requires representation of the obstacles in the environment and the traversable space, some algorithms require even more information. Numerous different planning frameworks exist, most of them are heavily dependent on the volumetric map used to represent the environment, as different strategies are available depending on the representation.

Authors of [24] proposed a method for creating a sparse graph with topological information that can be used for three-dimensional planning. An ESDF environment representation is used to create a 3D Generalized Voronoi Diagram, further more a thin skeleton diagram is obtained. The thin skeleton diagram is turned into a sparse graph, which is used for planning paths. This method was created to be used with Voxblox.

UFOExplorer [6] is an exploration planner based on the UFOMap [17] mapping framework. Every time the UFOMap is updated, a new dense graph based planning structure is updated. The planing structure is then used together with a simple exploration heuristic.

SphereMap [8] proposes a multi-layer graph structure that is incrementally built, and enables fast retrieval of topological-volumetric map. SphereMap generates clearance information that allows for rapid safety-aware path planning between any two points in an dynamic environment. SphereMap can receive occupancy information either from UFOMap or OctoMap mapping frameworks. During the DARPA SubT Challenge, the SphereMap was successfully

tested in an underground tunnel system. In [3] multiple different strategies for path planning were used, one of the methods was SphereMap.

1.2 Problem statement

The existing approach in the Multi-robot Systems Group (MRS) for generating volumetric maps relies on the OctoMap mapping framework. OctoMap is effective for mapping compact environments, whether in the laboratory or outside. However, challenges arise when OctoMap is used to map larger environments, particularly in terms of the speed of integrating new data. Consequently, this affects the accessibility to the created map, for subsequent tasks like planning, as it is built using a monolithic data structure. This is a limiting factor for further research, that relies on autonomous flights in large scale environments.

1.3 Contributions

As described in Section 1.1 various approaches to volumetric mapping exist. However, most of the current solutions struggle with large scale volumetric mapping. This thesis presents an approach to the challenge of large-scale volumetric mapping and planning aboard UAVs. The proposed solution works by combining two state-of-the-art mapping frameworks, each with its strengths and weaknesses. A mapping and planning pipeline was developed based on the approach and integrated to work with MRS UAVs, and tested in simulation, on captured data and in real-world experiments.

Chapter 2

Preliminaries

This chapter summarizes the tools on which this thesis is based on. The chapter will provide an introduction to the basic principles of Bayesian probability, Robot Operating System (ROS), MRS UAV system, the simulation environment Gazebo, and a brief overview of the main sensor used in volumetric mapping the Light Detection and Ranging (LiDAR) sensor.

2.1 Bayesian probability theory

Bayesian probability is a key theoretical and practical framework for making decisions and reasoning under uncertainty by updating beliefs in light of new evidence. Reasoning about a given system in Bayesian probability is not given in binary conclusions, but with conclusions accompanied by the measure of uncertainty. In Bayesian probability, a system is fully described by a number of variables which can be continuous or discrete. Probability density functions are functions whose value at a certain point can be directly represented as the likelihood that a random variable would have the value of the point.

At the core of Bayesian probability is the Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions relating to the studied event. Mathematically it is defined as:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}, \tag{2.1}$$

where $P(A)$ and $P(B)$ represent the probabilities of events A and B, and $P(A \mid B)$ stands for the conditional probability of event A given event B, analogically for $P(B \mid A)$. In Bayesian probability, the probabilities are interpreted as degree of belief.

Bayesian probability is used to model all types of behaviors from simple to complex scenarios. Bayes filter, Bayesian networks, Markov chains, and Monte Carlo methods all rely on the Bayesian probability theory.

2.1.1 Bayes filter

The Bayes filter algorithm is the most basic algorithm for calculating beliefs [35]. The algorithm calculates the belief distribution from the incoming measurements. Bayes filtering is a way to calculate the marginal posterior distribution of a state $\mathbf{x}_k$ at each time step k, while historical measurements ($\mathbf{y}_{1:k}$) are taken into account

$$p(\mathbf{x}_k \mid \mathbf{y}_{1:k}). \tag{2.2}$$

The algorithm works in three steps: initialization, prediction, and update. The initialization step sets the initial assumption of the prior distribution $p(\mathbf{x}_0)$ for the recursion. The prediction step uses the Chapman-Kolmogorov equation to calculate the predictive distribution model of the state $\mathbf{x}_k$ at time step k. The Chapman-Kolmogorov equation is:

$$p(\mathbf{x}_k \mid \mathbf{y}_{1:k-1}) = \int p(\mathbf{x}_k \mid \mathbf{x}_{k-1}) \cdot p(\mathbf{x}_{k-1} \mid \mathbf{y}_{1:k-1}) d\mathbf{x}_{k-1}. \tag{2.3}$$

The update step updates the posterior distribution of state $\mathbf{x}_k$ based on the measurement $\mathbf{y}_k$ at the time step k. The update is computed using the Bayes rule:

$$p(\mathbf{x}_k \mid \mathbf{y}_{1:k}) = \frac{1}{Z_k} p(\mathbf{y}_k \mid \mathbf{x}_k) \cdot p(\mathbf{x}_k \mid \mathbf{y}_{k-1}), \tag{2.4}$$

where the normalization constant Z_k is given as:

$$Z_k = \int p(\mathbf{y}_k \mid \mathbf{x}_k) \cdot p(\mathbf{x}_k \mid \mathbf{y}_{1:k-1}) d\mathbf{x}_k. \tag{2.5}$$

Bayes filter is used in robotics to construct probabilistic representations of an environment, or to calculate the believed position of a robot.

2.2 Robot Operating System

Robot Operating System (ROS) is an open-source framework widely used to develop complex robotic systems. ROS provides a set of tools, libraries, and certain conventions that enable the development of complex robotic systems. The name ROS is not an operating system such as Linux or Windows. ROS could be better described as a messaging framework that allows one to exchange data between running programs. In ROS multiple programs are organized into packages. A package is the core unit of ROS and contains the source code and other resources that are used to solve a specific problem. Packages are grouped into workspaces, which are directories that contain all the packages that are used in a project.

ROS uses a peer-to-peer network of ROS processes called the computational graph. The building blocks of the Computational Graph are Nodes, a Master mode, Parameter Server, messages, services, and topics. Each of these components exchanges data with the graph in a specific way. The nodes are the core components for performing computations. For example, one node can be responsible for receiving data from a camera, and another node can be responsible for processing the picture. The sharing of data between two nodes ROS uses messages. Messages are a data structure that contains data fields that should be a specific data type. Custom data types can be primitive types, such as integers, floats, and strings, or they can be more complex types, such as nested structures or a previously defined message.

To efficiently route messages between nodes ROS uses a publish/subscribe model. In this model, a node publishes a message on a topic, and other nodes can subscribe to that topic to receive the message. It is also possible to have multiple nodes subscribe to the same topic and even multiple nodes publishing to the same topic. The Master is a node that provides the Computational Graph with information about the nodes, topics, and services. However, this topic model is not ideal for request-reply type of communication. For this, ROS uses services. They are similar to topics, but are defined by a pair of message structures. The first message

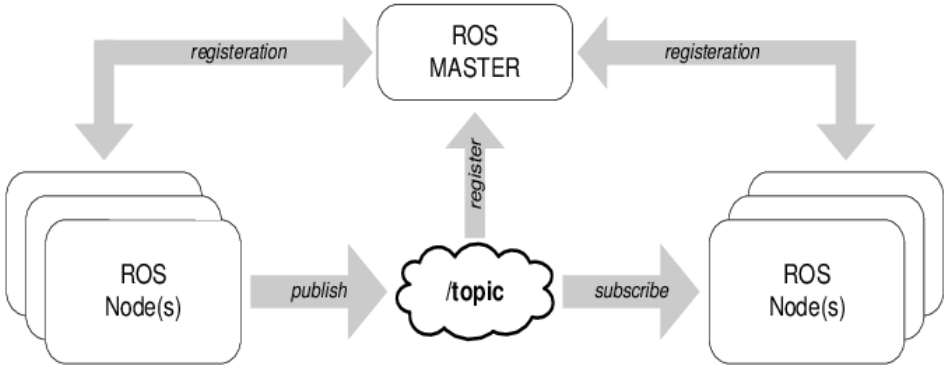


Figure 2.1: The figure show how ROS nodes communicate together from [25].

is the request message, and the second is the response message. Figure 2.1 shows the method by which communication between ROS nodes works.

To log messages for offline use ROS uses *bags* that are a file format to store messages. The bag format is very useful for research purposes, as it allows one to record the data that are being published on a topic and then replay it later. When testing new systems, tests that are run with data from bags can be more realistic like compared to running them in a simulation environment.

2.3 MRS UAV System

This thesis uses the MRS UAV system [13]. The MRS UAV system is a full-stack UAV platform, that provides fully autonomous control for all MRS group UAVs. The system is designed to provide a wide range of functionality from state estimation to safely controlling the UAV. MRS UAV system is suited for both realistic simulations and real-world experiments. The four main components of the system are state estimation and sensor fusion, control, tracking and trajectory generation. The complete architecture of the system is shown in Figure 2.2.

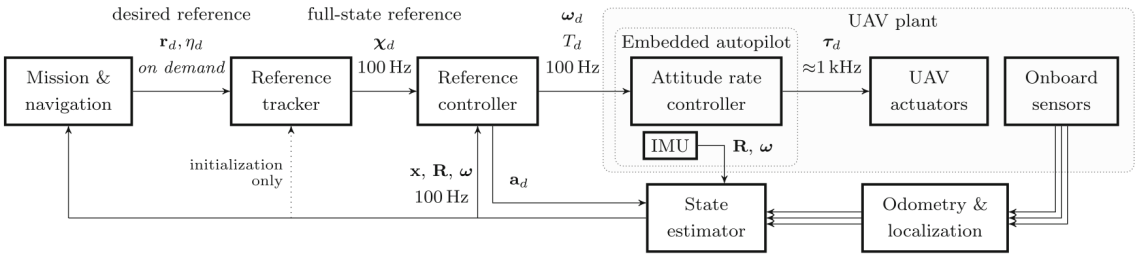


Figure 2.2: A diagram of the MRS UAV system architecture [13].

For state estimation, the MRS UAV system implements a bank of Kalman filters to enable the UAV to estimate its state independently from various onboard sensors. The system allows for smooth transitions between different filters and estimations, for example, from GPS used outside to SLAM for indoor use.

As the MRS UAV system is responsible for controlling the UAV it utilizes a linear model predictive control to use as a reference to the feedback controller.

For users, the MRS UAV system implements trajectory generation that takes a set of waypoints and in real time generates a reference trajectory for the UAV. The system solves a nonlinear optimization problem that solves both the geometry of the path and time sampling in one problem. The generated trajectory is then used as a reference for the controller.

More information is available at the MRS UAV system GitHub repository[1] and in the publication [13].

2.4 Gazebo

Gazebo is an open-source 3D simulation environment that is used to model the behavior of robots in indoor and outdoor environments. Gazebo is well integrated into ROS. Gazebo utilizes a numerical physics engine to produce realistic simulations. The convenience of using Gazebo is that it can run the same ROS instance that is then used for the real robot in the simulation environment. Even though Gazebo simulator has many limitations, such as only simulation of basic mechanics of solid objects, single threaded execution, or unrealistic visualization, it is still used as the main simulation environment when using ROS. Figure 2.3 depicts one UAV in a forest in the Gazebo simulation.

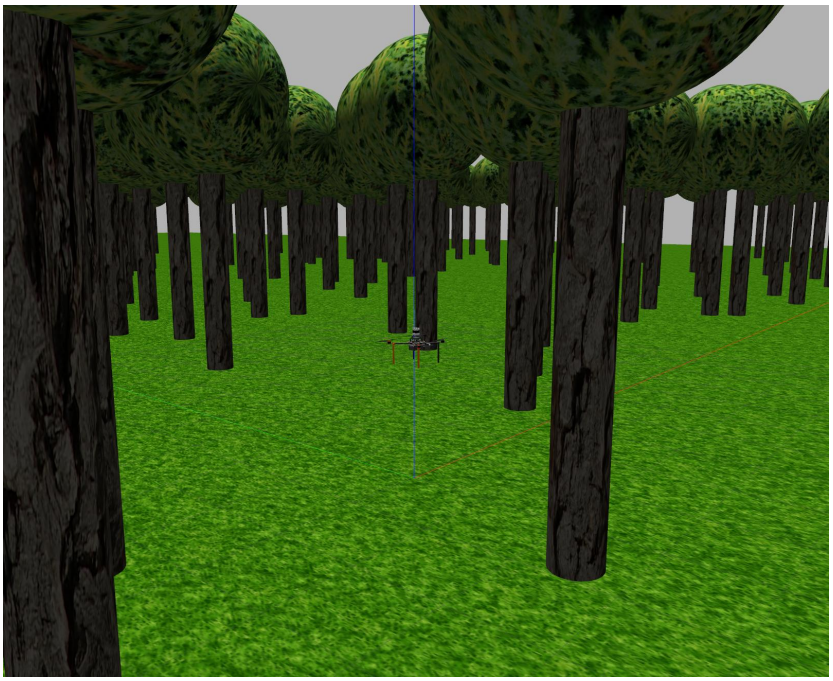


Figure 2.3: Screenshot from the Gazebo simulator showing a single UAV in a forest.

2.5 Light Detection and Ranging

LiDAR is an active remote sensing device that emits a beam of light to measure the distance to an obstacle. The usage of LiDARs has became essential to various fields like robotics, autonomous vehicles, environmental monitoring, and mapping.

[1] https://github.com/ctu-mrs/mrs_uav_system

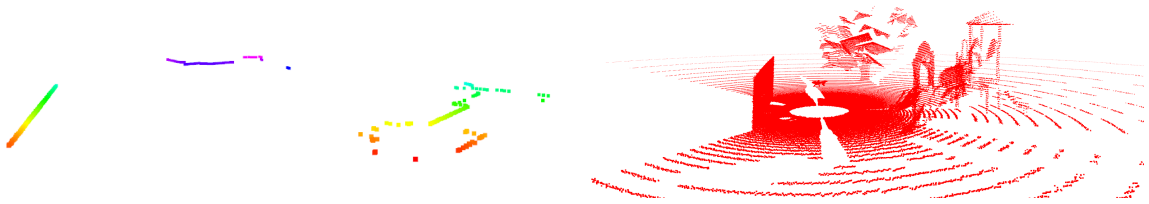
LiDARs work on a similar principle as a classical radar. Instead of relaying on radio waves that bounce off an obstacle LiDAR uses a beam of light or a laser beam. The LiDAR emits a short beam of light, which propagates through the environment. When the beam meets a solid object, a part of the beam is reflected or diffused back into the sensor of the LiDAR.

After the sensor detects the echoed beam, different techniques can be used to calculate the traveled distance. The simplest way to calculate the distance is by measuring the time it took the beam to travel from the beam emitter to the obstacle and back to the sensor, this is formulated in this equation

$$d = \frac{c \cdot \Delta t}{2 \cdot n}, \tag{2.6}$$

where c is speed light, Δt is the time between emitting and receiving the beam and n is the refractive index of the environment, which is approximately 1 for air. There have been several methods developed to determine the exact time each beam has been sent. The most widely used is modulating the emitted beam in a deterministic way based on the current time.

LiDARs can be categorized into two main categories, 2D LiDARs and 3D LiDARs. A 2D LiDAR provides measurements of distance to objects on a 2D plane by rotating the emitter and sensor. For each measurement, the distance to an object and the current angle of the rotating sensor are stored to represent the point in space. While 2D LiDARs provide valuable measurements, it cannot capture information above or below the scanning plane, which is limiting for certain applications. On the other hand 3D LiDARs have the capability to capture measurements in a wider field of view. 3D LiDARs do not only capture the distance and angle information, but also the elevation of the measured beam. This allows the 3D LiDAR to represent each measured point in a three-dimensional space. The complete list of measured points in one sweep by a LiDAR is commonly called a laser scan. Figure 2.4 compares the output laser scans of a 2D and 3D LiDAR.



(a) An example of a 2D laser scan. (b) An exaple of a 3D laser scan.

Figure 2.4: The comparison of the outputs of the 2D LiDAR (a) and 3D LiDAR (b). Both outputs show the same scene and were captured from the same position.

In this thesis, all further mentions of LiDARs refer to the 3D LiDAR as it is suitable for volumetric mapping.

Several types of LiDAR exist each utilizing different principals and being developed for different applications. Two main types are mechanical scanning LiDARs and solid-state LiDARs. Solid-state LiDARs have no moving parts, while mechanical LiDARs have a rotating sensor and emitter. In this thesis a mechanical LiDAR is used. More information about the different types of LiDAR can be obtained in [11].

The processed data from a LiDAR is represented as a set of points in a three-dimensional coordinate space. The created set is commonly called a Point Cloud (PC) and mathematically is described as

$$PC = \{p_i \mid p_i = (x_i, y_i, z_i) \in \mathbb{R}^3;\ i = 1, \ldots, N\}, \tag{2.7}$$

where p_i is a 3D point typically in the coordinate frame of the LiDAR, and N the number of scanned points.

The evaluation of a LiDAR system encompasses various factors, each of which is significant depending on the intended application. Resolution, determined by the number of columns and rows, is a prominent characteristic. Additionally, the maximum range of the emitted light beam is crucial, although it is often traded off with the field of view when striving for an extended range. Adverse weather conditions, such as the presence of small water droplets, smoke, or intense sunlight, can significantly affect the performance of LiDARs. Furthermore, the choice of wavelength influences the susceptibility of LiDARs to interference from sunlight, necessitating the implementation of point filtration techniques, as "ghost" points often appear in the output.

Chapter 3

Volumetric mapping

The creation of volumetric maps is considered to be one of the fundamental building blocks to the deployment of autonomous systems that operate in unknown environments. This chapter focuses on introducing the important concepts used in the creation of volumetric maps.

3.1 Problem definition

The problem of volumetric mapping on board mobile robots entails the task of constructing a three-dimensional representation of the surrounding environment using sensor data collected by the robot. The objective is to create a comprehensive and accurate representation of the physical space, including both the geometric structure and, sometimes, even semantic information, such as obstacles, objects, and traversability. This mapping problem involves addressing challenges related to noisy volumetric measurements, drifting odometry, registration, and efficient storage and processing of volumetric data. By effectively solving the problem of volumetric mapping, mobile robots can acquire a detailed understanding of their environment, enabling them to perform tasks such as navigation, localization, and object recognition in complex and dynamic real-world scenarios.

3.2 Occupancy grid

When creating a 2D map representation of an environment, the occupancy grid has been proven to be a viable representation. A 2D occupancy grid discretizes a 2D slice of the environment into a regular grid of cells. This is shown in Figure 3.1, where white cells are free and black cells are occupied.

This approach can be extended to the three-dimensional space. The 3D occupancy grid creates a volumetric representation of the environment by discretizing the space into a grid of cubic volumes. The cubic volumes are typically called voxels. The 3D variant of the occupancy grid is represented as a three-dimensional array of either ones or zeros to indicate whether a voxel is occupied or free. An extension to occupancy grids are probabilistic occupancy grids [42] that do not model discrete states, but the probability of a voxel being occupied.

Compared to the usage of occupancy grids in 2D, 3D naive occupancy grids were not viable, due to the huge memory requirements. Before beginning the mapping process, an occupancy grid, a three-dimensional array, has to be initialized to the size of the area being mapped. These requirements have been proven to be impossible to satisfy in real-world large-scale mapping and are only used in classroom environments.

Different approaches exist to representing the occupancy grid in a memory-efficient way while solving the problem of initializing the whole grid beforehand. Two of the main approaches to the problems are OcTrees and voxel hashing methods.

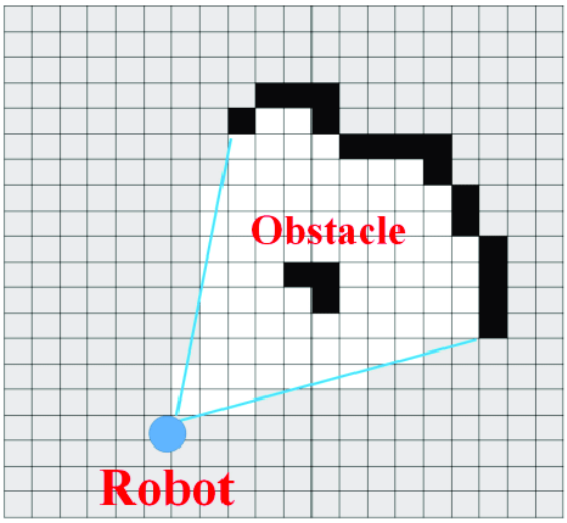


Figure 3.1: A visualization of a 2D occupancy grid, the white cells are free and black occupied [27].

3.3 Octal tree grid representation

Octal Tree (OcTree) is a data structure for dividing data into partitions, allowing for faster traversal of data in contrast to a traditional list based approach. OcTree is a type of graph, in which each node has eight children. Furthermore, the graph is unidirectional and only one path exists between two vertices, making it a tree. OcTrees are commonly used in robotics as a way to efficiently represent a 3D occupancy grid. The main benefits being the ability to dynamically resize the environment being mapped and faster data manipulation.

The usage of OcTrees for efficient volumetric representation was introduced in [43] [41]. OcTrees are used for representing occupancy grids by subdividing the environment into voxels. In an OcTree, a voxel is represented as a node within the graph structure. Voxels are recursively subdivided into eight voxels until the set minimum voxel size is achieved. In Figure 3.2 a cubic volume is seen divided into individual voxels, with a corresponding OcTree.

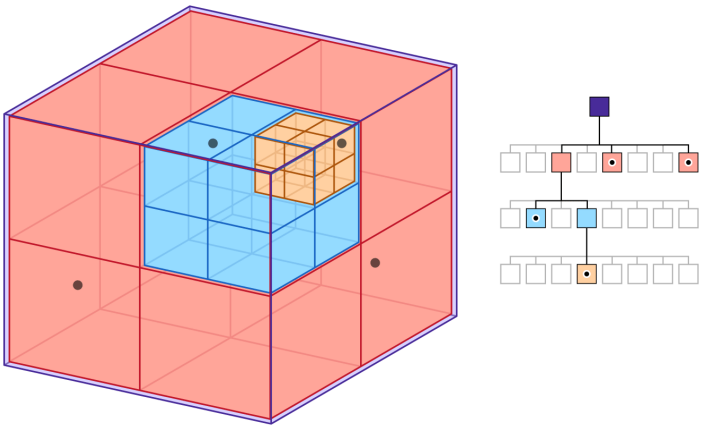


Figure 3.2: A graphical representation of how OcTrees subdivide volumes[1].

Maintaining the inner nodes of the tree accordingly enables the OcTree structure, as a hierarchical data structure, to be sampled at any level to yield a coarser subdivision. This is highly advantageous for applications where maximum resolution is not necessary. This can be seen in Figure 3.3, where an OcTree is sampled at three different depths.

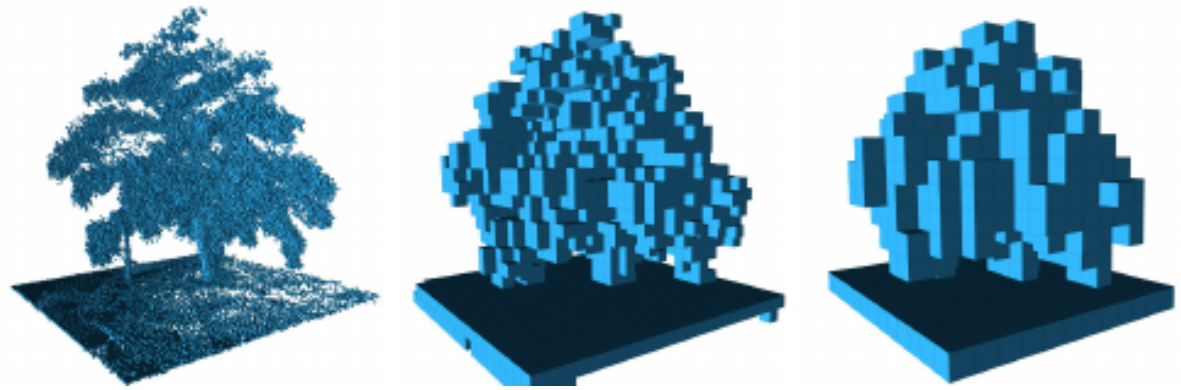


Figure 3.3: A visualization of sampling the OcTree at different depths to get different resolutions [32].

In the context of volumetric mapping, the basic OcTree data structure is initially limited to representing a Boolean occupancy state. However, this simplistic approach poses challenges in accurately representing voxel states due to uncertainty and noise in real-world robotic systems. To address this, a probabilistic representation is employed, where the belief of occupancy is stored instead of discrete values. By leveraging the Bayes filter algorithm (Section 2.1.1), the prior probabilities are continuously updated based on the likelihood of sensor measurements being true positives, enabling more robust and adaptable representations. Bayes filter algorithm is used in most implementations of OcTree mapping frameworks like OctoMap [32].

Numerous abstractions and simplifications have been made to the theory, to enable real-time performance. Each node of an OcTree holds the log-probability of the voxel being occupied, compared to storing a classical probability. The update of a node's log-probability is simpler compared to the classical probability as it only uses additive operations, and thus the update is computationally efficient. To determine the discrete state of a voxel, bounds, that determine the state, are created to suit a specific application. Root nodes can be updated in numerous ways depending on the application, the value can be an average, the minimum, or the maximum of its children.

Leveraging the fact that the parent nodes represent a coarser representation, we are able to prune the OcTree when all the child nodes have the same occupancy value. Different methods can be used to prune the probabilistic OcTree, however, pruning will always be lossy. Due to the nature of noisy measurements, neighbor nodes will have different probabilities so pruning can not be triggered only when all sibling nodes have the same log-probability. In the implementation of [32] pruning is done whenever the log-probabilities reach the bounds of an occupancy state.

[1] `https://developer.apple.com/documentation/gameplaykit/gkoctree`

3.4 Signed distance fields

Signed Distance Fields (SDFs) are a powerful and versatile tool in computer graphics and robotics. They are commonly used to represent the geometry of objects and scenes in a 3D space. SDFs define a continuous scalar field that assigns to each point in space the distance to the nearest surface of an object. Moreover, this distance is signed to indicate whether the point is inside or outside the object. This property enables SDFs to provide not only accurate distance information but also the ability to determine the location and orientation of the surfaces that make up the object.

SDFs can be used for a variety of applications, including collision detection, ray tracing, surface reconstruction, shape analysis, and motion planning. One of the advantages of SDFs is their ability to represent complex shapes with high accuracy using a relatively small amount of memory. Additionally, SDFs can be computed from a variety of sources, including point clouds, mesh data and volumetric data, making them a flexible tool for a wide range of applications.

In recent years, SDFs have gained popularity in the field of robotics due to their ability to efficiently represent the geometry of the environment. They are particularly useful in applications such as simultaneous localization and mapping (SLAM), where they can be used to model the environment and estimate the robot's position and orientation. SDFs have also been used in collision avoidance and motion planning, where they can be used to efficiently compute the distance between the robot and obstacles in its environment.

SDFs can be mathematically defined as an implicit function $f : \mathbb{R}^3 \to \mathbb{R}$ that assigns to each point in space the distance to the closest point on the surface of an object, with a sign indicating whether the point is inside or outside the object. The distance is negative if the point is inside the object, positive if it is outside, and zero if it is on the surface. To formally define the SDFs we first have to define the unsigned distance function which we use to get the distance from point $\mathbf{p}$ to the closet point in a given set Ω:

$$dist(\mathbf{p}, \Omega) = \inf_{q \in \Omega} \|\mathbf{q} - \mathbf{p}\|. \tag{3.1}$$

The definition of the signed variant can then be defined as:

$$d_S(\mathbf{p}, \Omega) = \begin{cases} dist(\mathbf{p}, \Omega) & \text{if } \mathbf{p} \in \Omega \\ 0 & \text{if } \mathbf{p} \in \partial\Omega \\ -dist(\mathbf{p}, \Omega) & \text{if } \mathbf{p} \in \Omega^c \end{cases}, \tag{3.2}$$

where Ω^c is the complementary set to Ω and $\partial\Omega$ is the boundary of the set.

3.4.1 Euclidean signed distance fields

Euclidean Signed Distance Fields (ESDFs) are a type of SDFs where the distance function is calculated in Euclidean space. The $dist(\mathbf{p}, \Omega)$ is then defined as:

$$dist(\mathbf{p}, \Omega) = \inf_{q \in \Omega} \sqrt{(\mathbf{q} - \mathbf{p})}, \tag{3.3}$$

where $\mathbf{q}$ is a point from the Ω set and $\mathbf{p}$ is the point we are calculating the minimum distance for.

3.4.2 Truncated signed distance fields

Truncated Signed Distance Fields (TSDFs) are an extension of ESDFs that are commonly used in practical applications. In contrast to the standard ESDF which computes and stores values for all the points in the operating space, TSDFs restrict the distance values to a predefined range around the surface of the object. This allows the field to capture the geometric features of the object more efficiently and compactly while maintaining a high level of accuracy.

To generate a TSDF, the signed distance function is first calculated as usual. The distance values are then truncated by a maximum value τ, which determines the extent of the region around the surface that will be considered. Any distance values greater than τ are set to τ, effectively saturating the function in these regions. Mathematically, we can define the function as:

$$TSDF(\mathbf{p},) = \begin{cases} \min(d_S(\mathbf{p}, \Omega), \tau) & \text{if } d_S(\mathbf{p}, \Omega) \geq 0 \\ \max(d_S(\mathbf{p}, \Omega), -\tau) & \text{if } d_S(\mathbf{p}, \Omega) < 0 \end{cases}. \tag{3.4}$$

3.5 Factor graph

Factor graphs are graphical models, that were developed to model complex estimation problems [38]. Factor graphs can be classified as a special case of a Bayesian network, which is a way of representing uncertain knowledge. They are constructed as a bipartite graph $G = (V, F, E)$ with edges (E) and two types of nodes, variables (V) and factors (F). Variables represent unknown quantities, while factors represent functions on a subset of variables. The edges of a factor graph are always constructed between factors and variables and indicate how the individual factor depends on the specific variable. Factor graphs are a flexible structure that can be used to model numerous problems in robotics. They are often used as a structure that allows for efficient ways to compute a solution to a problem.

Factor graphs are exploiting the way that global functions factor, and use the distributive law to simplify the summations. Suppose that we have a function $g(x_1, \ldots, x_n)$, which is factored into a product of several local functions. Each local function has a subset of variables $x_1, \ldots, x_n$ as input arguments. A function $g(x_1, \ldots, x_n)$ can then be reconstructed as

$$g(x_1, \ldots, x_n) = \prod_{i \in I} f_i(X_i), \tag{3.5}$$

where I are the possible indexes, X_i is a subset of $x_1, \ldots, x_n$, and $f_i(X_i)$ is a local function.

The main usage for factor graphs in robotics is as the backbone of SLAM systems. In these applications, multiple sources of creating factors in the graph are used. Factors often called constraints of the problem can originate from the Inertial Measurement Unit (IMU), loop closure, GPS, and depending on the available data many more. If stationary objects in the environment can be localized, they can be added to the factor graph as landmarks, which when seen from multiple views help to refine the estimation. Figure 3.4 shows an example factor graph with landmarks and four types of factors.

The main power of factor graphs comes from the formulation of the problem as an optimization problem. The process of inference from the factor graph gives rise to a nonlinear least-squares optimization problem. A minimization function is defined by minimizing

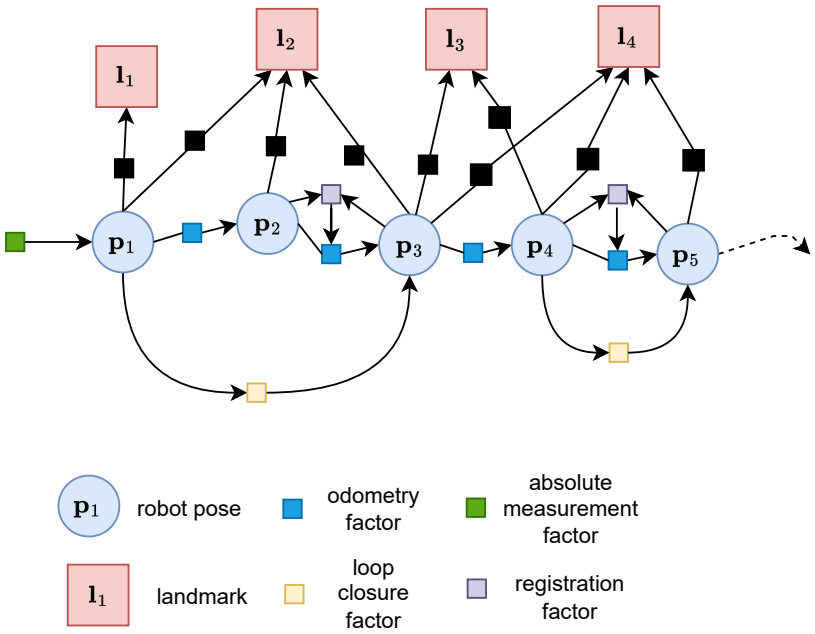


Figure 3.4: Example visualization of a factor graph.

the total error of all factor graph constraints. Open-source libraries such as G2o [36], Jaxfg [16] or GTSAM [5] were built to solve the factor graph optimization problem. The key factor to solving the optimization problem is computing the Jacobean or Hessian for each factor and then using methods such as Levenberg-Marquardt optimization.

3.6 Iterative closest point algorithm

The Iterative Closest Point (ICP) algorithm is widely used for geometric alignment of three-dimensional models and was introduced in [40]. The algorithm is capable of aligning two three-dimensional models purely on the geometry of the models. In robotics, ICP is widely used to minimize the distance between two measurements from LiDAR sensors. Matching two three-dimensional models has many applications, for example, to localize a robot in a known environment or to estimate odometry from two consecutive LiDAR measurements. The main work of the ICP algorithm is divided into three parts. First, the algorithm ingests the source and destination models, and an initial guess transformation of the two models is performed. In the second step within each iteration, the source model is moved so that it is closer to the destination model. The second step is iterated until convergence or the maximum number of iterations is reached. As the last step, the translation vector and the rotational matrix are returned. The detailed working of the algorithm is shown in Algorithm 1, where ICP operates on two input PCs. The process of the algorithm is also shown in Figure 3.5.

Many improved variants of the naive ICP algorithm exist. Some improvements to the naive implementation and their effects are detailed in [39]. The first change can be made in the selection of point pairs, as using all available points can be computationally expensive. Numerous sampling methods have been proposed, and each is well suited for a specific application. Improvements can also be made in finding the closest point because if the initial guess is incorrect, the naive algorithm struggles to converge. Different functions can be used

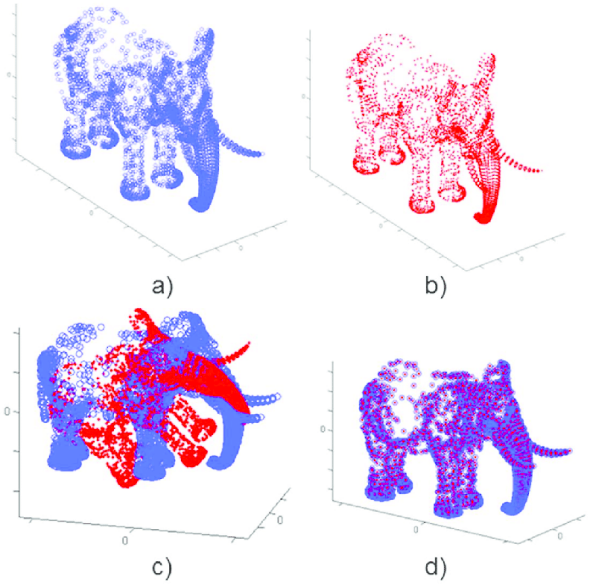


Figure 3.5: Two input PCs used as input are depicted in a) and b). The initial guess is depicted in c) and the final alignment of the ICP algorithm in d), from [29].

to calculate the error metric and find the minimum of the error metric. New steps can even be added to make ICP more robust like weighting pairs or rejecting pairs to minimize the effects of outliers in noisy measurements.

Algorithm 1 The naive ICP algorithm

applications example

Input source point cloud S and destination point cloud D, convergence threshold ϵ, the maximum number of iterations $maxIterations$

Output translation t, rotation R

$n \leftarrow 0$
$E \leftarrow \infty$
$\mathbf{M_d} \leftarrow \frac{1}{N_d} \sum_{i=1}^{N_d} \mathbf{p_i}$ ▷ Center of mass of the destination point cloud
$\mathbf{M_s} \leftarrow \frac{1}{N_s} \sum_{i=1}^{N_s} \mathbf{p_i}$ ▷ Center of mass of the source point cloud
$S' \leftarrow Transform(S)$ **s.t.** $M_s = M_d$ ▷ Transform the source point cloud to destination center of mass
while $E > \epsilon$ **or** $n \leq maxIterations$ **do**
    for $\mathbf{s_i}$ **in** *source* **do**
        $\mathcal{C}(\mathbf{s_i}) \leftarrow (\mathbf{d_j}, \mathbf{s_j}) = \min \|(\mathbf{d_j}, \mathbf{s_j})\|_2^2$
    end for
    $S \leftarrow S'$
    $\mathbf{R}_{[n]}, \mathbf{t}_{[n]} \leftarrow \sum_{i,j \in N_d, N_s} \|\mathbf{d_i} - \mathbf{R} \cdot \mathbf{s_j} - \mathbf{t}\|_2^2$
    $S' \leftarrow Transform(S, \mathbf{R}_{[n]}, \mathbf{t}_{[n]})$ ▷ Transform the source point cloud by $\mathbf{R}$ and $\mathbf{t}$
    $E \leftarrow \frac{1}{|I|} \sum_{i \in I} (\mathbf{R}_{[n]} \mathbf{s_i} + \mathbf{t}_{[n]} - \mathbf{d_j})^2$
    $n \leftarrow n + 1$
end while
return $\mathbf{R}, \mathbf{t}$

Chapter 4

Path planning

Path planning for mobile robots is a fundamental problem in the field of robotics that involves determining an optimal or feasible trajectory for a robot to navigate from a given starting location to a desired goal location while avoiding obstacles in the environment. This problem arises in various real-world applications, such as autonomous vehicles, UAVs, and mobile manipulators, where the robot needs to plan a safe and efficient path to perform its assigned tasks.

The goal of path planning is to identify a collision-free path that fulfills specific constraints and objectives, taking into account the environmental limitations and optimization criteria. To achieve this, an effective path planning system necessitates both an appropriate representation of the operating environment and the usage of path planning algorithms to generate a sequence of robot positions that can be followed to successfully traverse the environment.

The path planning problem can be formulated as a search problem in a high-dimensional configuration space, where the robot's state is represented by its pose or configuration. The key challenge lies in efficiently exploring this high-dimensional space to find a feasible and optimal path, considering the computational complexity.

The effectiveness of a path planning algorithm is typically evaluated based on criteria such as path length, computation time, smoothness of the trajectory, ability to handle dynamic obstacles, and robustness to uncertainties. Additionally, real-world considerations, such as energy efficiency, execution time constraints, and task-specific requirements, further contribute to the complexity of the problem and influence the choice of path planning algorithms and techniques.

4.1 A* algorithm

The A* algorithm, also known as the A-star, is a popular and widely used optimal path finding algorithm. It is specifically designed to find the shortest path between two nodes in a graph, taking into account the cost of moving from one node to another. The algorithm combines the advantages of both uniform cost search and greedy best-first search, making it efficient and effective for solving path finding problems.

At its core, the A* algorithm utilizes a heuristic function that estimates the cost of reaching the goal from each node in the graph [44]. This heuristic, often denoted $h(n)$, provides an optimistic estimate of the remaining cost from a given node to the goal. The A* algorithm maintains two lists: an open list and a closed list. The open list maintains the nodes that have been discovered but not yet expanded, on the other hand the closed list maintains the nodes that have already been expanded.

The A* algorithm begins by initializing the open list with the starting node. Then, A* iteratively selects the node with the lowest $f(n)$ f-value. The f-value is computed:

$$f(n) = g(n) + h(n), \tag{4.1}$$

where $g(n)$ represents the cost of reaching node n from the start node. The algorithm expands the selected node by examining its neighbors, calculating their f-value, and adding them to the open list if they are not already present. The process continues until the goal node is reached or the open list becomes empty. The detailed view of the algorithm is visible in Algorithm 2, where early stopping can occur, resulting in a partial path to the goal.

The key strength of the A* algorithm lies in its ability to efficiently explore the most promising paths towards the goal, guided by the heuristic function. By considering both the actual cost of reaching a node $g(n)$ and the estimated cost to the goal $h(n)$, A* strikes a balance between completeness and optimality.

Algorithm 2 The A* planning algorithm

```
1: procedure findShortestPath(start, goal, maxNumExpanded, timeout)
2:     openQueue ← PriorityQueue                    ▷ Highest priority has the lowest cost
3:     closed ← List
4:     path ← List
5:     numExpanded ← 0
6:     starTime ← currenttime
7:     openQueue.Add(start)
8:     while openQueue is not empty or numExpanded ≤ maxNumExpanded do
9:         current ← openQueue.Pop
10:        closed.Add(current)
11:        if current.Equals(goal) then
12:            break
13:        end if
14:        elapsedTime ← current time - startTime
15:        if elapsedTime ≥ timeout then
16:            break
17:        end if
18:        neighbors ← current.Expand                ▷ Return all traversable neighbors
19:        for neighbor in neighbors do
20:            neighbor.Cost ← computeCost(neighbor)
21:            if neighbor in closedSet then
22:                if neighbor.Cost < closed.Get(neighbor).Cost then
23:                    closed.Remove(neighbor)
24:                else
25:                    continue
26:                end if
27:            end if
28:            if neighbor in openQueue then
29:                if neighbor.Cost < openQueue.Get(neighbor).Cost then
30:                    openQueue.Remove(neighbor)
31:                else
32:                    continue
33:                end if
34:            end if
35:        end for
36:        numExpanded ← numExpanded + 1
37:    end while
38:    while not current.Equals(start) do
39:        path.Add(current)
40:        current ← current.Parent
41:    end while
42:    return path
43: end procedure
```

Chapter 5

Methodology

In Section 1.3 the main objective of this thesis was to create a robust volumetric mapping and planning pipeline. This chapter provides a comprehensive overview of the proposed pipeline's components and how they function together. The pipeline can be subdivided into four distinct segments according to the function and is shown in Figure 5.1.

The first segment is the handling of raw PC (Section 5.1), and in the pipeline diagram in Figure 5.1 it is in red. The second segment is the GPS independent source of odometry (Section 5.2), and in the diagram it is in yellow. The purpose of the third segment is to ingest PCs and create the volumetric map (Section 5.3), and the color of this segment in the diagram is green. And the last segment is meant for planning a trajectory based on the volumetric map (Section 5.4), the diagram shows it in blue.

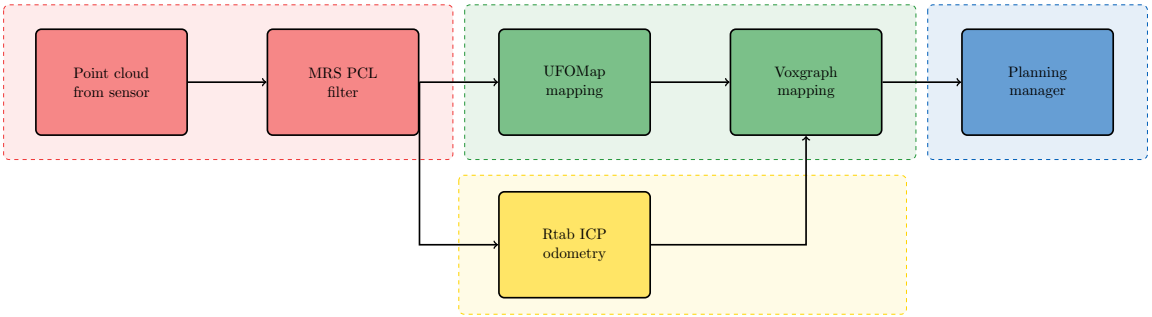


Figure 5.1: A diagram of the created pipeline used for mapping and planning.

5.1 Point cloud processing

The primary input for the mapping and planning pipeline is obtained from the output of the LiDAR. Unlike in simulations, under real-world conditions, LiDARs can struggle and their output suffers from noise. To mitigate the effects, a filtering process is applied to the output PCs. To filter the PCs MRS PCL tools[1] was employed. MRS PCL tools implement numerous ROS nodes that work with PCs.

Since the 3D LiDAR is mounted on top of the UAV, the UAV itself is present in the LiDAR's output. To address this issue, points in the PC that are in near the sensor, corresponding to the UAV, are removed. Furthermore, the performance of a LiDAR can be affected by harsh sunlight in real-world conditions. That is caused by both the laser beam and sunlight beam entering the sensor and potentially causing confusion between the two, resulting in false

[1] https://github.com/ctu-mrs/mrs_pcl_tools

positive detection of an object. To eliminate the fictitious "ghost" points resulting from this effect, filtration based on the point intensity[2] is applied to the PC.

For commonly used sensors like the Ouster OS0-128[3], the output PC contains 262144 points. The OS0-128 has 128 scan lines and 2048 points per line, which for most applications is too dense. To reduce the number of points in the PC, a selection of lines and rows is done, the selected are subsequently removed from the PC.

After the mentioned preparation steps, the PC is ready to be used for the following tasks volumetric mapping and estimation of odometry.

5.2 Odometry estimation

Number of different methods were tried in simulation and on captured real-world data, however most methods struggled with keeping an accurate estimate in captured real-world data. The selected method of estimating odometry was using the PCs from LiDARs, compared to visual methods for the estimation. Odometry is calculated by integrating the estimated position change between two time steps. The relative global position generated by these methods is important to enable the system to function in GPS denied environments.

In this thesis the ICP algorithm (Section 3.6) is used to estimate the transformation between two consecutive PCs. An implementation of the ICP algorithm from RTAB-Map [22] was used as numerous improvements have been made compared to the naive algorithm. The main improvement is adding measurements from an IMU to improve the accuracy of the initial transformation guess. More information about the implementation can be found at the GitHub page[4] and in the publication [22].

5.3 Creating a volumetric map

The main part of this thesis was to design and then create the pipeline mapping segment. When starting the work on this part of the framework a number of different available mapping frameworks were tested, however none of them were able to perform adequately on real data of a long flight. A plan was devised to combine two mapping frameworks together. The first framework is fast in integrating sensor data in smaller areas, this framework is called UFOMap [17]. The second framework is ideal for large-scale global mapping, but struggles with high-frequency noisy sensor measurements, it is the Voxgraph mapping framework [19]. Combining these two completely different mapping frameworks was done by using UFOMap as a small local map around the robot and then using this local map as input for the global map created by Voxgraph. In Figure 5.1 it is shown that the mapping segment is composed from the two frameworks that work together. No similar approach was found that would leverage two completely different map representations in unison.

In this chapter the two frameworks will be elaborately introduced and the changes that had to be made to each framework are described.

[2] LiDAR intensity is the power of the returned ray.

[3] `https://ouster.com/products/scanning-lidar/os0-sensor/`

[4] `https://github.com/introlab/rtabmap`

5.3.1 UFOMap mapping

UFOMap is a probabilistic OcTree based mapping framework that was introduced in [17]. UFOMap improves the popular OcTree based mapping framework OctoMap [32]. Compared to OctoMap, which only explicitly represents Free and Occupied nodes in the OcTree, UFOMap represents all three states explicitly adding the Unknown. The main advances compared to OctoMap are significantly faster methods for incorporating data into the OcTrees, improved overall efficiency mainly by allowing inserting and deleting data while iterating all at the same time and making all functions capable of working on different resolutions of the OcTree. In UFOMap, a node's occupancy value has the same occupancy value as the maximum occupancy value of all of its children. This update method ensures that while traversing the OcTree at any resolution no obstacle can be missed.

Moreover, UFOMap introduces three indicators that are used in UFOMap's nodes. The three indicators are i_f, i_u, and i_a. The i_f, i_u indicate if a node contains free space or unknown space, respectively. The indicator i_a indicates that all the children nodes are the same, this is used when automatic pruning is disabled and is used to mimic the same behavior. In this thesis automatic pruning has been disabled to enable multithreading in the UFOMap server. If UFOMap had been used to create the global map, this would be a concern, due to increasing memory usage and slower operations. However, because in this pipeline UFOMap acts as a local sliding window map, this will affect the performance in a negligible way.

As UFOMap explicitly defines the Unknown state, two thresholds need to be defined to determine the discrete state from the log-probability. The t_o threshold determines whether a node is Occupied, while t_f is used for Free nodes. The occupancy state can then be determined by

$$state(n) = \begin{cases} Unknown & \text{if } t_f \leq n.p_{log} \leq t_o \\ Occupied & \text{if } t_o < n.p_{log} \\ Free & \text{if } t_f > n.p_{log} \end{cases}, \tag{5.1}$$

where n is a node from the OcTree and $n.p_{log}$ is the log-probability. This is very powerful, as by changing the thresholds a completely different characteristic of the resulting map can be achieved.

To accomplish fast traversal of the OcTree, Morton codes are used. Morton codes are used to encode an n-dimensional space onto a linear list of numbers. Morton codes define a space filling Z-shaped, coordinates that are close to each other in the original space are close to each other on the Z-curve. In UFOMap Morton codes enable fast traversal from root node to a child by simply looking at the three next bits of the Morton code.

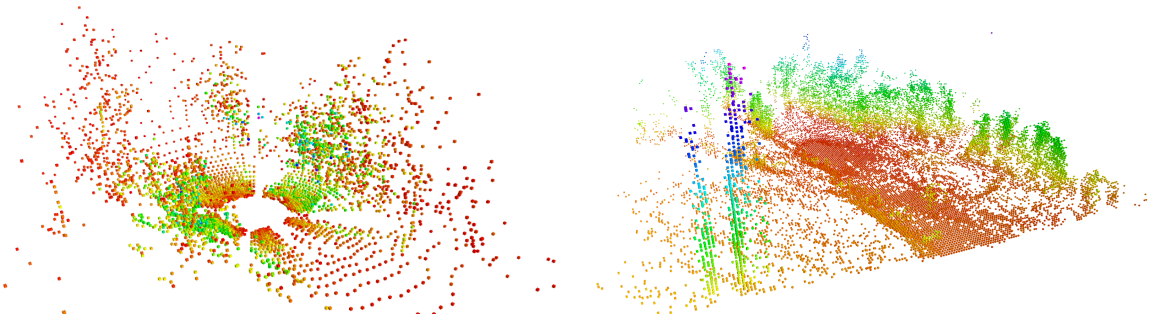
The improved performance in incorporating new data into the OcTree can be attributed to the fast discrete integrator. The fact that many points in the PC will update the same node is used to discretize the PC and only one ray cast is performed for each segment of the PC. Furthermore, the step above is done at different lower resolutions. First, a crude ray tracing algorithm is used at the coarser resolution until the number of nodes to the end point is n and continues with a more robust ray casting methods at higher resolutions. Thus, ensuring that only free space is updated at the coarser resolution and occupied space is updated at highest resolutions. To perform the update, two constants are defined, one to update the nodes of free space ($miss_{prob}$) and the second for occupied space (hit_{prob}). The $miss_{prob}$ is subtracted when ray tracing determines that a node is free, and a hit_{prob} is added to the node when it is at the end of the ray. In this thesis, these values were used $miss_{prob} = 0.4$ and $hit_{prob} = 0.7$.

5.3.2 Changes proposed to the UFOMap framework

In this thesis it was very important to only create a local UFOMap. The local map is created by using the sliding window technique. We create a region around the current position of the UAV and only the part of the map in the region is left. As the UAV moves the map is being cropped in the set frequency, in this thesis 0.1 Hz has been used. The cropping is done by alternating between two instances of UFOMap. During each crop these steps are made:

1. An oriented bounding box of the region is created for iterating thru the current OcTree.
2. Probabilities of each OcTree node at the maximum resolution is taken and then is passed to the fast discrete integrator.
3. The integrator recomputes the key for the node and inserts the nodes and their probabilities to an empty OcTree.
4. The original OcTree is cleared so it would be ready for the next crop round.

The local UFOMap has two use cases in this thesis. The first and main use case is creating a large noise-free PC. To create a PC from the local map an iterator over the OcTree Occupied nodes is used. The position of each node is retrieved and then added into the output PC. The resulting PC is then used as the input PC for Voxgraph. Figure 5.2 shows the difference between the PC from the LiDAR and UFOMap. The second use case is collision checking in the planned trajectory of the UAV.



(a) A point cloud from the LiDAR. (b) A point cloud created from UFOMap.

Figure 5.2: The comparison of the output PC from the LiDAR (a) and the PC from the UFOMap (b). Both outputs were captured at the same position.

To represent the environment, and utilize maximum information from the UFOMap in the pipeline, it is important to properly configure the Occupancy and Free thresholds in UFOMap. When creating a PC from UFOMap the goal is to label cells as occupied only when there is a high probability of occupancy, aiming to effectively filter out any noise from individual measurements. To achieve this the threshold value is set 0.8. As mentioned earlier, UFOMap is utilized also for collision checking of the planned trajectory, this necessitates a strict threshold for the Free state, as only Free space can be in the planned trajectory. To ensure accurate checking, the Free threshold is set to 0.3, making it hard to satisfy the constraint. Due to these stringent thresholds, larger portion of nodes will be classified as Unknown.

5.3.3 Voxgraph mapping

Voxgraph [19] is a robust framework for building globally consistent volumetric maps. The core of the approach is to represent the environment as a set of overlapping SDF submaps,

with an underlying factor graph. In Figure 5.3 the overall framework is shown, and in this section all the parts will be detailed, note that Voxgraph is divided into two parts the front-end and the back-end.

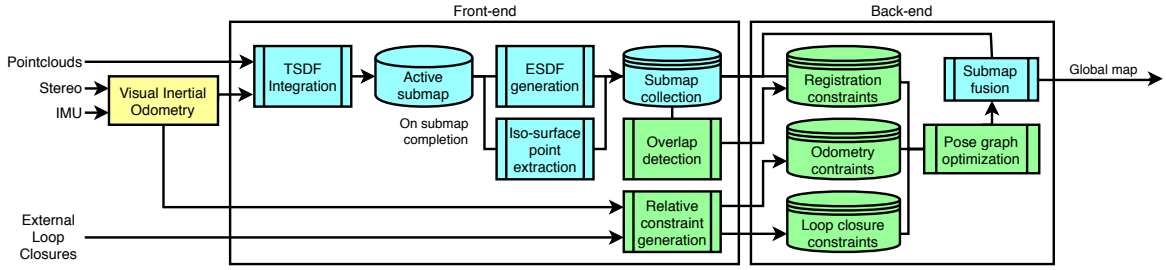


Figure 5.3: An overview of the Voxgraph framework as presented in [19].

Front-end

The front-end converts the incoming sensor readings into submaps and adds the generated constraints. When creating a global volumetric map with noisy or drifting localization of the sensor, the resulting map can accumulate significant error. The created errors in the resulting maps are often non recoverable.

Voxgraph builds on a map structure introduced in Cblox [23], which introduced a map structure that tries to remain globally consistent by representing the scene as a collection of locally consistent sub-volumes. Each sub-volume is represented as a TSDF submap into which the input PCs are being combined into. The representation of each TSDF submap involves the usage of spacial hashing and storing it in a hash table. Details regarding the spacial hashing technique can be found in [34]. The integration of PC is made efficient by using two techniques, weighting and merging. Weighting is used for indicating the belief of the represented distance, and is based on a model of the sensor characteristics. Merging uses a similar principal to the fast discrete integrator in UFOMap. When a new PC is integrated a ray-cast is performed from the sensor and every voxel in its path has its distance and weight updated. Projection mapping is used to calculate the distance to an obstacle, this is a major source of error, as the calculated distance will always match or overestimate the distance to the nearest obstacle.

Submaps are created at a fixed rate, and each submap is created with an attached frame and a parameterized transformation to the world frame. The rate at which submaps are created is important, as the assumption is that the error in odometry is introduced slowly during mapping, thus by having short intervals between creation ensures local consistency of odometry in the submaps. The trajectory of the sensor through the submap is stored as submap-relative poses in the submap. After a submap is completed the ESDF representation is computed from the created TSDF submap, the algorithm is detailed in [28]. The algorithm propagates the Euclidean distances outside the truncation area, this is achieved by a wavefront-wave based algorithm. As stated above, the TSDF submap already includes some error by using the projective distance. Furthermore, the ESDF building process introduces more error as the created submap does not use Euclidean distances, but a quasi-Euclidean distance, which is the distance to the closest adjacent voxel. The created ESDF submap is used for planning a safe trajectory, but due to the accumulated errors, planners should add about 10 % to their safety distance from obstacles.

The last step done in the front-end is generating the constraints for the underlying factor graph optimization problem. It should be noted that in the paper [19] the factor graph is incorrectly named pose graph. Three constraints are defined: Odometry, registration, and loop closure. The odometry constraint penalizes the deviation of the odometry estimate between two submaps, which is important for having a consistent sensor trajectory estimate between the two submaps. The loop closure constrain relate the sensor position at arbitrary time instances, this is not utilized in this thesis as it requires an external system to detect the loop closure. This can be resource intensive, and it is shown in this thesis that the pipeline in this thesis does not require it. The most important constraint is the registration constraint. The registration constraint ensures that the overlap of submaps is maintained, and overlap with all previous submaps is always checked.

Back-end

In the back-end part, the generated constrains are maintained and the estimation of the submap collection alignment is created. The alignment is done by minimizing the total error of all the constraints of the factor graph. The optimization problem is formulated as a non-linear least squares minimization:

$$\arg\min_{\mathcal{X}} \sum_{(i,j)\in\mathcal{R}} \|\mathbf{e}^{i,j}_{reg}(T_{WS^i}, T_{WS^j})\|^2_{\sigma_r} + \sum_{(i,j)\in\mathcal{O}} \|\mathbf{e}^{i,j}_{odom}(T_{WS^i}, T_{WS^j})\|^2_{\Sigma_O} + \sum_{(i,j)\in\mathcal{L}} \|\mathbf{e}^{i,j}_{loop}(T_{WS^i}, T_{WS^j})\|^2_{\Sigma_L}, \tag{5.2}$$

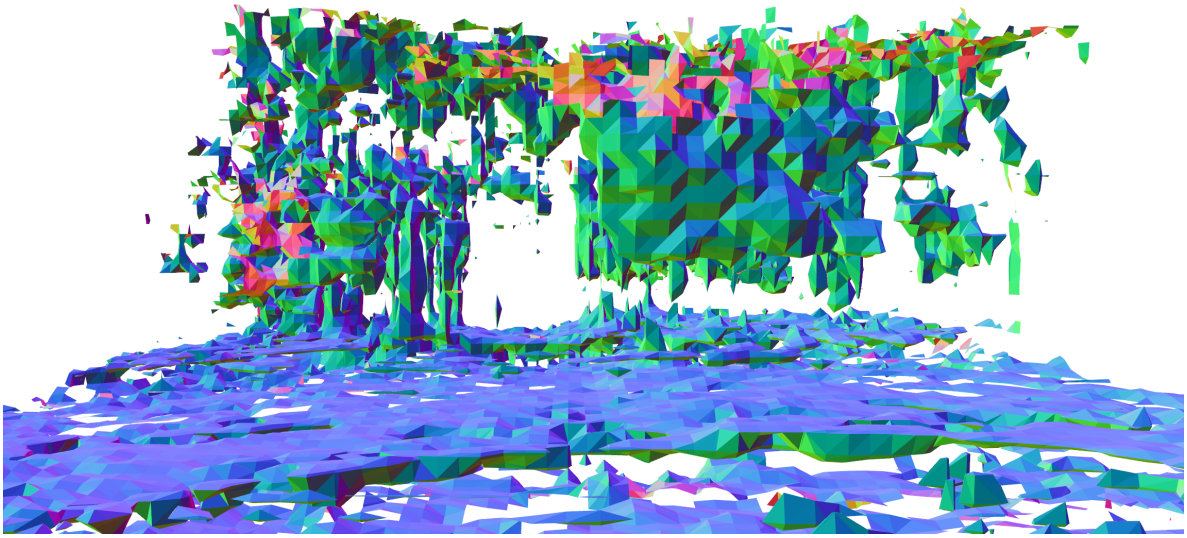
where $\mathcal{X} = \{T_{WS^1}, T_{WS^2}, \ldots, T_{WS^N}\}$ are the submap poses, $\mathcal{R}$, $\mathcal{L}$ and $\mathcal{O}$ are the sets containing the registration, loop closure, and odometry constraints, respectively. The $\|e\|^2_{\sigma_r}$ is the total weighted distance with the scalar weight σ_r, which corresponds to the registration constraints. The elements corresponding to the loop-closure and odometry constraints are $\|e\|^2_{\Sigma_L}$, $\|e\|^2_{\Sigma_O}$, which corresponds to the squared Mahalanobis distance. The Σ_L and Σ_O represent the covariance matrices. More information on the optimization process is found in paper [19].

Changes proposed to the Voxgraph pipeline

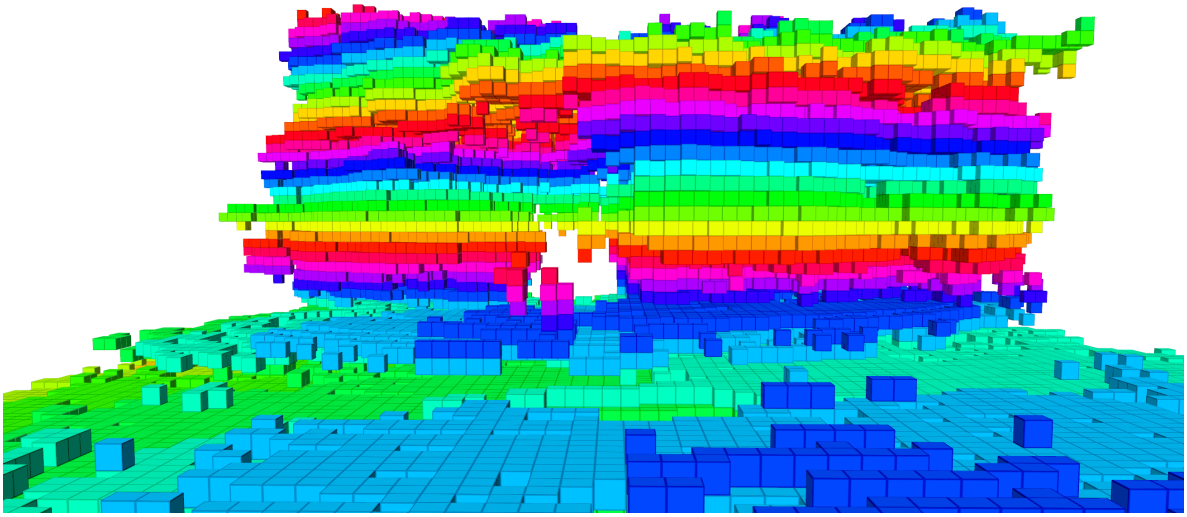
Several changes were made to Voxgraph, for it to properly work within the pipeline. The following section outlines the key changes that were made, elaborating on their significance.

To visually inspect the quality of the generated map, visualizations are often used. Natively Voxgraph implements a mesh visualization of the generated map, that is approximated from the underlying data. However, the created mesh does not accurately reflect the true representation of the underlying TSDF map. This is shown in Figure 5.4a, where trees float in the air separated from the ground, the tree tops are also disconnected and the ground has patches of holes. Using the mesh visualization the mapping framework could look broken as it does not correspond to the physical world, thus a more accurate representation of the map was created for visualization purposes. By combining all the TSDF submaps into a single map and then iterating through all the voxels and visualizing voxels, that are on the surface or inside an object and the weight of the voxel is larger than a set threshold. The major advantage is by directly sampling the TSDF no further approximations are needed. The resulting visualization is shown in Figure 5.4b, the view is from the same location and with the same map as in Figure 5.4a. The created visualization depicts the environment more realistically, as the trees

do not float in mid air and no holes are in the ground. The occupancy marker visualization is far superior to the mesh visualization when debugging the mapping and planning pipeline.



(a) The mesh visualizing the map, the mesh does not represent the environment as the trees are floating in the air, the trees have holes in them and the ground is not completely filled in.



(b) The created occupancy marker visualization of the map, note that the trees are connected to the ground and the ground is without holes.

Figure 5.4: Figure (a) shows the inaccurate mesh visualization and Figure (b) shows the created occupancy marker visualization. Both figures show the same map from the same location.

In order to utilize Voxgraph as the global volumetric representation, it must be adapted to support global path planning functionalities. By default, Voxgraph does not provide a map

representation, that would be suitable for global path planning. During the process of path planning, a safety margin is established to ensure a minimum distance is maintained between the path and any surrounding obstacles. Compared to the visualization of occupied space, path planning requires distance information further from the surface of an obstacle, thus the ESDF map representation should be used to generate an input for the planner. First the ESDF map has to be generated for the active submap, because the ESDF map is normally computed after the completion of the submap. Afterwards the submaps are iterated through, and an expanded occupancy representation is created, from now on this representation is called the obstacles. Voxels that closer than the safety distance to a surface are treated as occupied and are added into the obstacles representation for the planner. A visualization of the obstacles representation was created for debugging purposes.

The last significant modification made to the original implementation is the inclusion of loop closure constraints. While loop-closure constraints were deemed necessary for consistent global mapping in the Voxgraph paper [19], however they are not considered necessary in the proposed pipeline. This simplification can be attributed to the utilization of UFOMap as the local map, whereby only filtered data from UFOMap is integrated into a submap, allowing for improved registration constraint detection of overlap. Furthermore the framework does not need to discard any data due to slow integration time. It is worth noting that both the registration constraint and the loop-closure constraint serve the same purpose in the factor graph, of connecting the same visited location in two different submaps. The only difference being that the loop-closure detection is conducted externally from the Voxgraph map.

5.4 Planning manager

The planning manager component of the pipeline is responsable for the movement of the UAV within the operating environment, with a primary focus on ensuring the safety of the UAV. The planning manager operates as a partially exploratory system designed for basic missions in an unknown environment. At its core the planning manager can be described as a state machine, consisting of three states Idle, Planning and Moving. The underlying concept behind the planning manager revolves around periodically discarding the previously planned path and generating a new path towards the set goal using the updated volumetric representation of the environment.

The planning manager receives multiple inputs to facilitate the planning process. The primary input for planning is the obstacles message generated from the global Voxgraph map, which provides the positions of voxels, that are considered as obstacles and should be avoided during path planning. Additionally, the local UFOMap serves as a secondary source of volumetric information, enabling the collision checking process along the planned path. The last essential input for the planning manager is the desired goal position, that is obtained from the user.

The internal state machine of the planning manager initiates in the Idle state and remains Idle until a goal position is provided. Upon receiving a goal position, the state transitions to Planning. In the Planning state, the volumetric representation of obstacles generated from Voxgraph is utilized to plan a safe path. The planning manager employs the A* planning algorithm, described in Section 4.1, to plan the optimal path towards the set goal. It is important to note, that the planning process is not done from the current position, but rather from the predicted position estimated by the model predictive control at the end of the planning phase. This approach is necessary to mitigate the jerking motion that would occur if

the stating point was the current position, due to the frequent replanning. The output of the A* algorithm consists of a set of waypoints, discrete sequence of positions, which can either be the complete or partial path leading to the goal position.

Due to the exploratory nature of the planning manager, it is essential to limit the duration of a planned mission, because of the potential for significant changes in the environment. The estimation of the flight time involves calculating the distance between consecutive waypoints, and considering the current flight constrains determining the flight time. If the projected mission duration exceeds a predefined threshold the waypoints are removed to satisfy the flight time limit, the maximum flight time for a single plan was set to 15 s. Subsequently, the trajectory generator within the MRS UAV system is used to approximate a continuous curve, known as the trajectory, based on the discrete waypoints. The generated trajectory is used by the autopilot to control the drone. A visual representation of an example trajectory can be observed in Figure 5.5 as a green line.

Once the trajectory is generated and published, the state machine transitions to the Moving state to initiate the movement of the UAV. In the Moving state, two checks are performed. Firstly, it is checked whether the UAV has reached the goal position, and if so the state is switched back to Idle. Secondly, the elapsed time is examined since last path planning. If the time surpasses the predefined replanning interval, the state is transitioned to Planning, thus triggering the generation of a new path.

A continuous verification process is carried out when the state machine is in either Planning of Moving state to ensure the safety of the planned trajectory. This verification involves utilizing the local UFOMap, to perform safety check within a defined distance around the planned trajectory. A bounding box is created between consecutive waypoints, and the space within the bounding box is examined by iterating through the nodes of the OcTree. If only Free space is detected within the bounding box, the path is considered safe. However, if Occupied or Unknown space is encountered, the path is deemed as unsafe, and the planned path is shortened accordingly. This approach ensures that any potential new or dynamical obstacles, not adequately represented in the Voxgraph map, are taken into account. The flowchart illustrating the implemented state machine for the planning manager is illustrated in Figure 5.6.

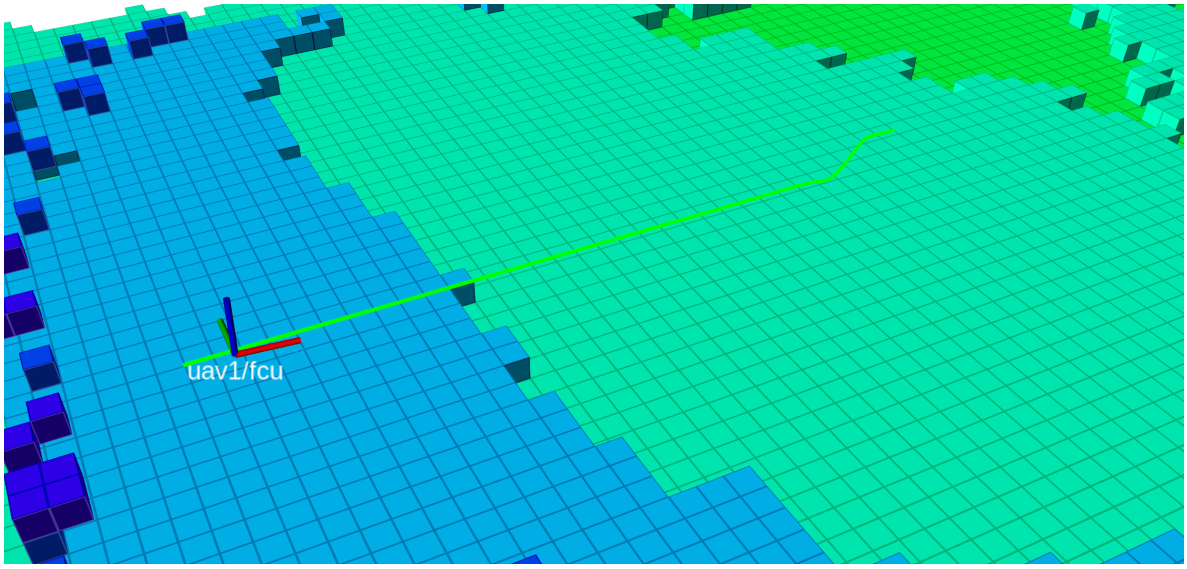


Figure 5.5: A planned trajectory is shown in green.

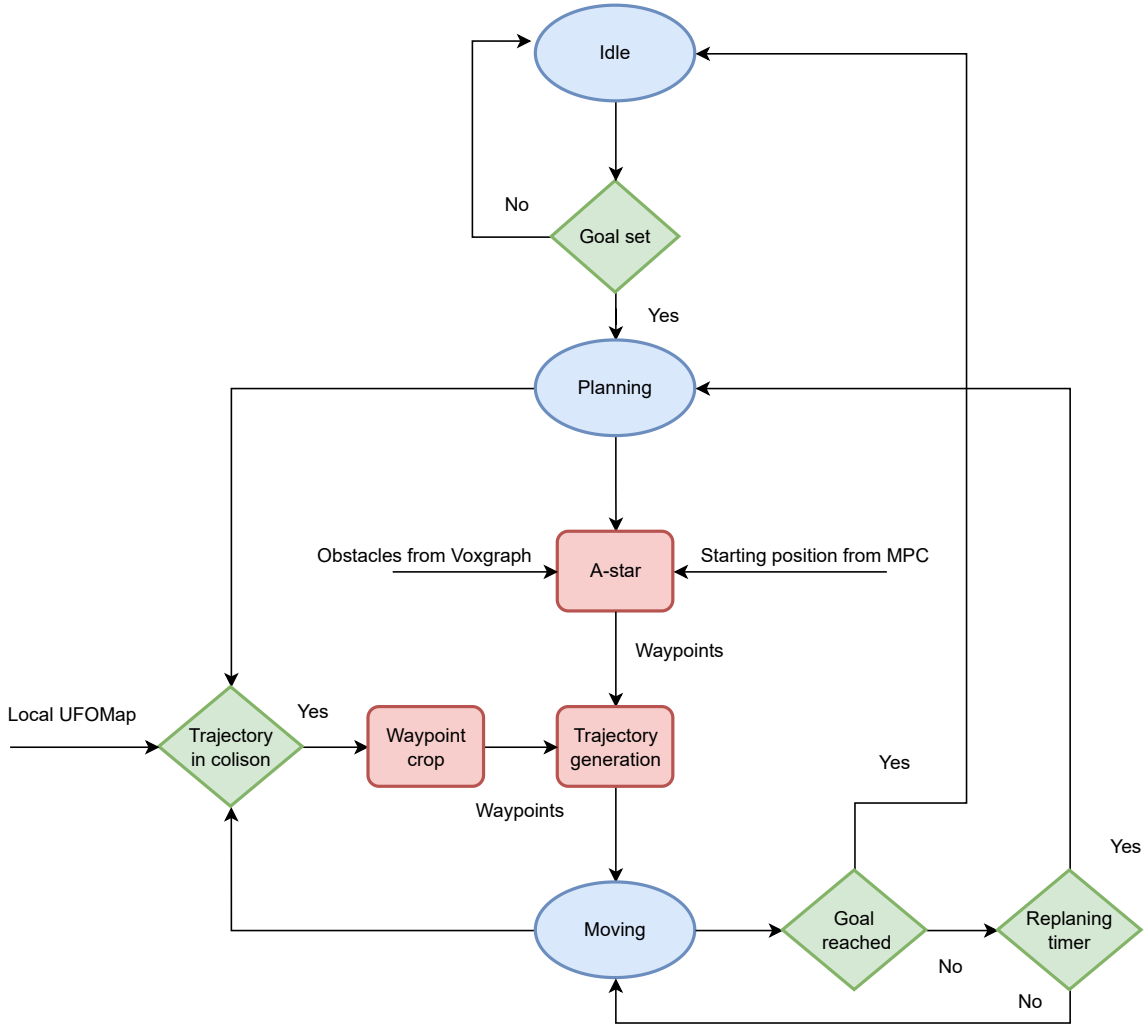


Figure 5.6: A flow chart of the planning manager.

Chapter 6

Verification

This chapter will demonstrate the functionality of the proposed pipeline. The verification of the pipeline was carried out in two phases. In the early stages the testing was done in a realistic Gazebo robotic simulation. Later, verification was done on a dataset from a long flight in a forest [3]. Tests done using the Dataset should be taken with more weight as it provides realistic data to test with. The dataset was furthermore used to fine-tune all the parameters of the pipeline for real-world tests.

6.1 Simulation

The simulation experiments were all run in Gazebo using the MRS Gazebo simulation package[1]. In all the performed simulations the t650 UAV platform was used. The LiDAR model used in all the simulation experiments is Ouster OS0-128. To test the ability of the pipeline, most of the tests were carried out in a dense forest world, as shown in Figure 6.1. Simulations were run on a laptop with AMD Ryzen 9 5000HS CPU and NVIDIA GeForce RTX 3060 Laptop GPU with 32 GB of available memory.

Figure 6.1: The forest world in the Gazebo simulator.

The verification of the proposed pipeline is separated into two parts. First the mapping section of the pipeline was verified to ensure that the proposed pipeline functioned as intended

[1] https://github.com/ctu-mrs/simulation

(Section 6.1.1). Secondly the planning manager was rigorously verified by running different configurations of the environment and the goals (Section 6.1.2).

6.1.1 Mapping

During the verification process the mapping section of the pipeline, a significant number of simulated flights were conducted. In all but one tested scenario the proposed pipeline performed as intended, and the resulting map represented the environment accurately.

The problematic testing scenario was a simple dynamic environment, which exposed a flaw in Voxgraph. The scenario consisted of placing a firetruck in front of the UAV, during the period a submap containing the firetruck was created. The UAV was left stationary and the firetruck was moved to the left of the UAV as can be seen in Figure 6.2a, and a second submap was created. To get the resulting global map the two submaps are merged and visualized as can be seen in Figure 6.2b, the resulting maps now contains two firetrucks. This problem is created by performing a simple merge of the maps. This is a limiting factor in the proposed pipeline, and thus it should be used only in static environments.

6.1.2 Planning

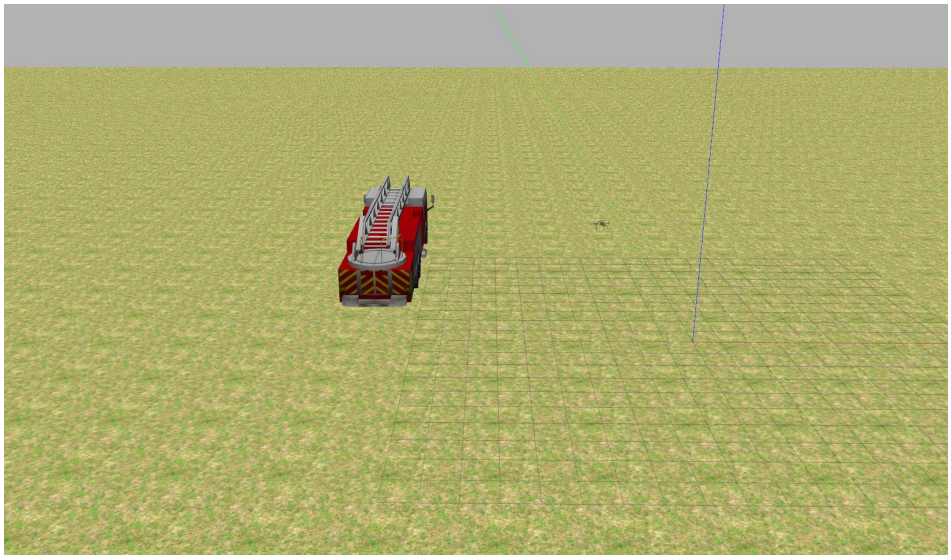
After heavily testing the mapping part of the pipeline the planning manager was verified. During these scenarios, goals were set outside the seen part of the environment, so exploration would have been needed. This was done to mimic the conditions under which the planning manager would be tested in real-world experiments. The verification consisted of a substantial number of autonomous missions, aimed on testing the planning manager. All the performed tests were successful and the drone was able to navigate safely in all the tested environments. The safety distance during the tests was set to 1.5 meters, thus the obstacles were inflated by 1.5 meters. The planned trajectory is shown in the volumetric map (Figure 6.3a) and with the obstacles used for planning (Figure 6.3b), during a test in the forest environment.

6.2 Dataset

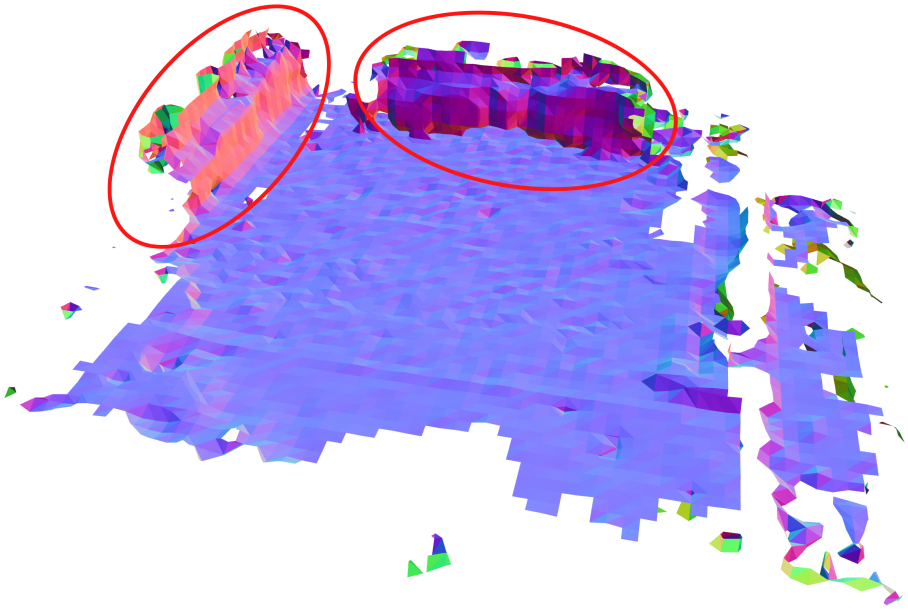
The forest dataset introduced in [3] contains data from a long 19 minute flight that was approximately 1.25 km. The flight captures a sparse forest, which is a challenging environment to obtain accurate odometry readings. The captured data are stored in a rosbag file that was described in Section 2.2.

During the flight sensor data from the OS0-128 LiDAR, Realsence d435 and Garmin Lidar Lite were recorded. From the sensors only data from the OS0-128 LiDAR were used to create the volumetric map.

Number of tests were conducted on real this data to test the performance of the pipeline, however planning tests cannot be done due to the nature of the data. The performance in these tests is more important than the verification in simulation as it reflects more accurately real-world performance. During all the performed tests the resolution of Voxgraph mapping was set to 0.4 meters.



(a) The view of the simulation world with one fire truck on the left.

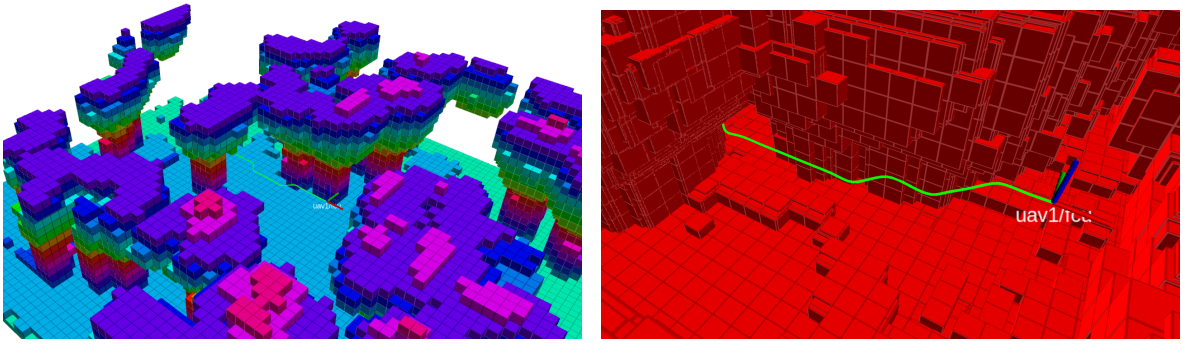


(b) The created map with two fire trucks, marked with red ovals, one on the top and the second on the left, even tho the map should show the fire truck on the left.

Figure 6.2: Figure (a) shows the last state of the environment in simulation and Figure (b) shows the failed resulting map.

6.2.1 Mapping

To verify the performance of the proposed pipeline on real data, a verification test was conducted. The main objective of this test was to evaluate the performance of the ICP odometry on real data from the rosbag. The results of this test showed that some problems might occur when applying the pipeline to real data, as demonstrated in Figure 6.4. The test revealed several shortcomings, mainly that the ICP algorithm was frequently losing track and



(a) The volumetric map with the planned trajectory.

(b) The visualisation of the obstacles used for planning and the planned trajectory.

Figure 6.3: Figure (a) shows the volumetric map and Figure (b) shows the obstacles used for planning both have the planned trajectory visualized in green.

required frequent resets. The failure of the ICP algorithm during the long flight was expected, as the environment is very challenging for the algorithm. However, the resulting map shows, that two major elevation changes happened during the mapping process, even tho that did not happen during the flight. The estimated trajectory had the largest error in the z axis, which resulted in the deformation of the created map. The created global map could not be used to navigate the environment due to the warps. The problem of major z axis error could be attributed to inaccurate data from the IMU of the Ouster LiDAR, to tackle these challenges an external precise IMU could be added on top of the LiDAR. These issues are expected to occur during real-world experiments.

6.2.2 Mapping with GPS

A verification test was conducted to assess the performance of the pipeline when using real drifting GPS localization. The objective was to evaluate the output of the pipeline with GPS in comparison to the ICP output. The results of this test are presented in Figure 6.5, which clearly demonstrates that the pipeline performs better with noisy GPS, compared to the estimated odometry using ICP. To achieve the result seen in Figure 6.5, numerous tests were performed to fine-tune all available parameters.

The parameters that were fine-tuned during these tests are: UFOMap resolution, local UFOMap size, frequency of publishing the PC from UFOMap, Voxgraph submap creation interval, Voxgraph truncation distance, and maximum ray length in Voxgraph. The first tests were done with either default or random values. From those initial tests, it was evident that the effects of each parameter had to be checked to optimally choose their values. The main issue from the initial tests was the doubling of the part of the map that was visited multiple times, in [19] global loop-closure was used to resolve this issue. This can be seen in Figure 6.6, where the two copies of the same place are next to each other.

The first parameter experimented with was the frequency of the PC from UFOMap publishing. It was observed that as more PCs were integrated into each Voxgraph submap, the worse the resulting global Voxgraph map looked. With this knowledge, the PC publishing rate and the Voxgraph submap creation interval were set to the same value. The best results were achieved with the values set at 10 seconds. The next important parameter was the resolution of the local UFOMap, which was set to the same resolution as Voxgraph. This was chosen

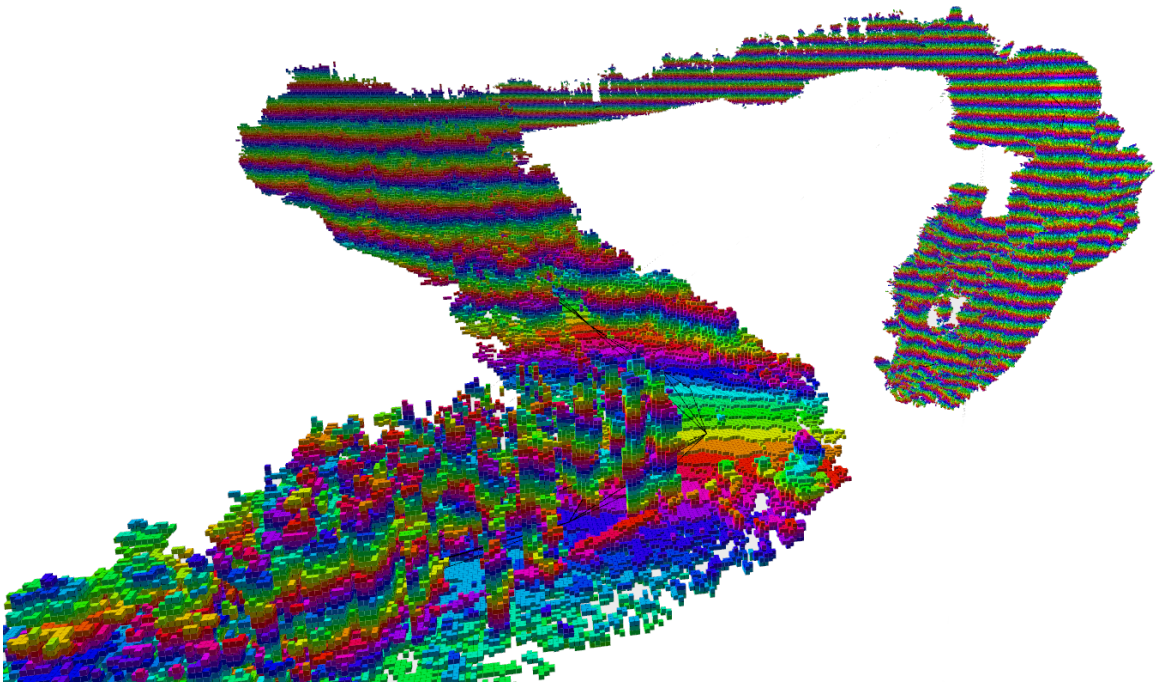


Figure 6.4: This figure shows the created map with the proposed pipeline. It shows how the ICP algorithm struggled to provide reliable odometry, resulting in a warped map, with inaccurate elevation.

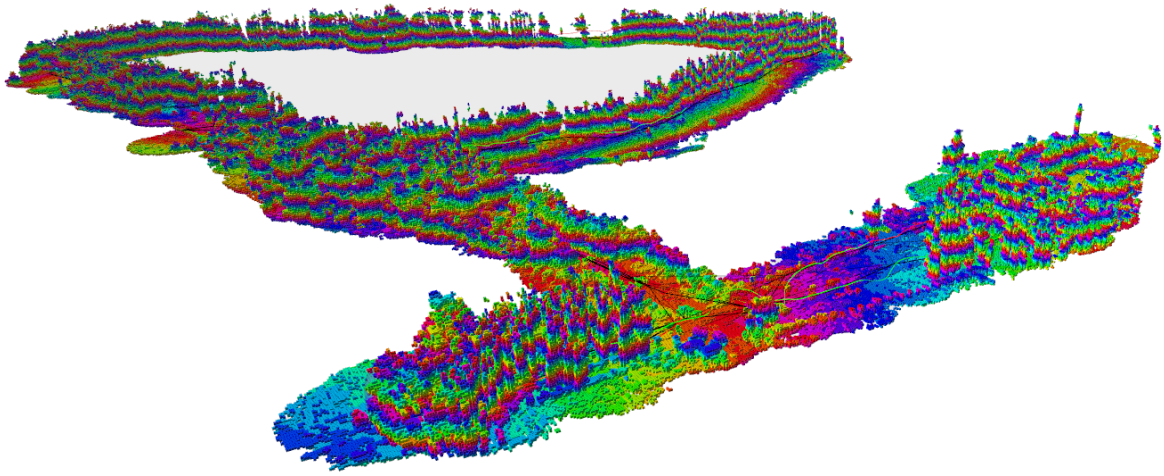


Figure 6.5: The figure shows the created map when localization was provided by GPS.

because of the way the Voxgraph integrator merges multiple points that fall into the same voxel into the same point. The size of the local map depends on the frequency of publishing the PC from local map. When the time between PC publishing is long, the local map needs to be larger. For the 10 second interval, an 80 x 80 meter sized local UFOMap was ideal. The maximum ray distance was set to 26 meters, however shorter rays result in smoother operations as less computations need to be performed during ray tracing. The truncation distance is very important and should be set to $truncation_{dist} = 2^n \cdot resolution$. To achieve

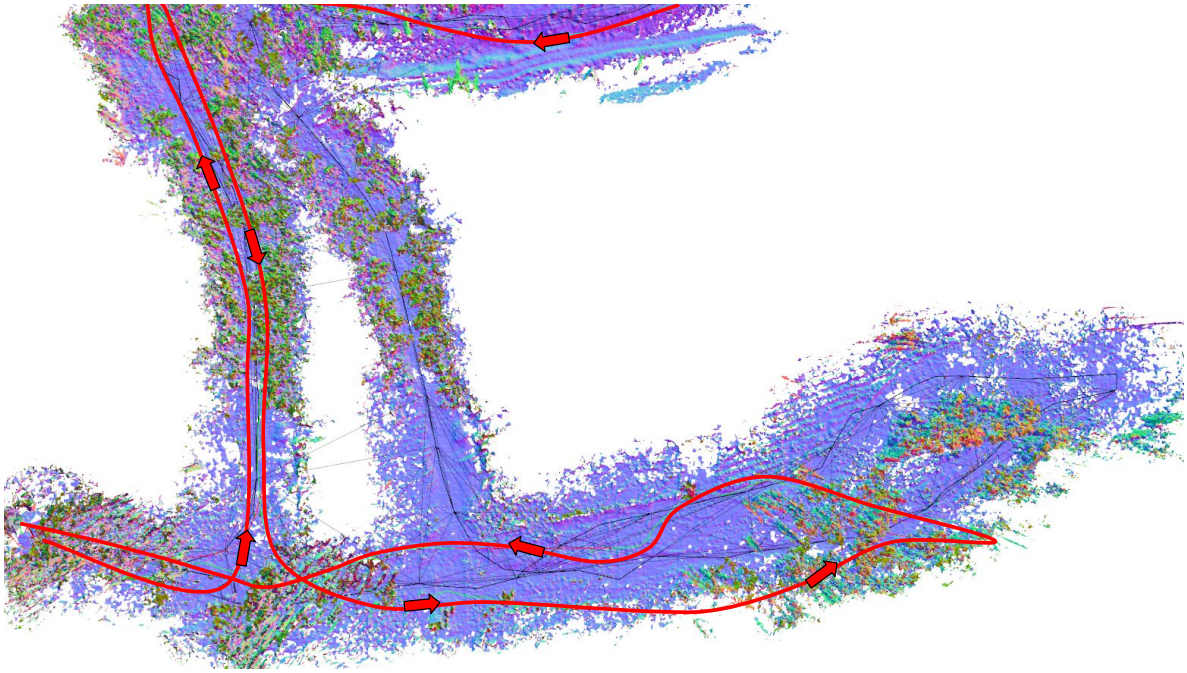


Figure 6.6: The figure shows a part of a map where doubling occurred. The red line is the ground-truth trajectory, while the black line is the estimated trajectory.

the best result, n was set to 1 as the error generated from the projective distance used in TSDF map is minimal. With these parameters, the flight shown in Figure 6.5 was carried out. The trajectory comparison for this flight is shown in Figure 6.7, where the black trajectory is estimated from the factor graph optimization and the red and green path is the "ground truth" from Real-time Kinematics (RTK) GPS. The important feature to look at is the start and end point (right-lower corner), which should be identical, and for neither one is, this proves that the problem of accurate localization has not been solved.

To put in context the performance of the created pipeline key metrics were recorded. These key metrics were: memory requirements, PC integration time, time taken to create the different outputs and processor usage. The memory required for the global map created by Voxgraph is about 5.5 GB and average PC insertion took about 0.6 s. The large memory needed was expected as the flight took about 19 Min. and the map representation is not very memory efficient. The average integration time was very revealing and explained the better performance with less frequent PC insertions. UFOMap on the other hand is very efficient with average memory usage of memory 4.6 MB, its average integration speed was also as expected very low at 0.011 s. An average crop of the local UFOMap took about 0.05 s with maximum integration time at 1.5 s, the source of the deviation from the average time was not determined. It took on average 0.01 s to generate a PC from the local UFOMap. The output of Voxgraph are the obstacles and on average it took 1.6 s to generate the output, this could be largely improved by publishing obstacles only from a subset of submaps. The data from the tests are shown in Table 6.1, together with data measured with the currently used mapping framework in MRS OctoMap for comparison.

To sum up the findings, the most important discovery was finding the correlation between the period of publishing the PC from local UFOMap and the submap creation period. This was the key to achieving results comparable to using an external SLAM system doing

Figure 6.7: A comparison of the estimated pose from GPS by Voxgraph factor graph (black) compared to RTK GPS (green and red).

Metric	Voxgraph	Local UFOMap	OctoMap
Avg. scan integration time [s]	0.64	0.01	0.07
Max. scan integration time [s]	2.37	0.05	0.35
Avg. total memory [MB]	5500	4.6	3.8
Avg. local map crop time [s]	-	0.05	-
Avg. PC generation from UFOMap [s]	-	0.01	-
Avg. obstacles generation time [s]	1.62	-	-

Table 6.1: Table presenting the measured values of diverse metrics associated with the developed pipeline, and includes a comparison with the OctoMap mapping framework presently used in the MRS.

global loop closure and adding constrains to the factor graph, which was done in [19]. The created system is far more efficient compared to the system proposed in [19].

Chapter 7

Real–world experiments

After conducting a successful verification of the pipeline in the Gazebo simulator and in the forest dataset, real-world experiments were carried out to test the real performance. During the experimental campaign, three types of experiments were conducted. The experiments were carried out during the MRS experimental campaign in Temešvár.

The first type of experiment (Section 7.2) was to test the proposed pipeline without autonomous flight. During this experiment, the drone was remotely controlled and took place in a forest. Except for the planning manager, the whole pipeline was being tested, especially this experiment was focusing on real-world performance of the used ICP algorithm. The experiment was carried out to establish the safety of using the odometry generated from PCs during autonomous flight.

The second type of experiment (Section 7.3) tested the proposed pipeline but using noisy measurements from GPS. This experiment was also conducted in manual flight and took place in the same forest to compare the outputs with the previous experiment. During this experiment, the odometry generation and planning segments were inactive.

The third type of experiment (Section 7.4) was conducted to test the pipeline in autonomous flight. The experiment was carried out in two locations on a meadow and in a forest. The goal was to test the integration of the planning manager with the mapping segment.

7.1 Hardware platform

During the real-world experiments a MRS UAV was used [7]. The UAV is build upon the x500 platform. The UAV is equipped with a Intel Nuc computer with 16 GB of memory and an Intel Core i7. The used UAV is shown and described in Figure 7.1. During the experiments two types of Ouster LiDARs were used the OS0-128 and the OS1-16, however the performance was better with the OS0-128 and the conducted experiment used it.

7.2 Manual flight

This real-world experiment aimed to assess the proposed mapping pipeline by manually flying a UAV through a forest, while the odometry estimate was provided by the ICP algorithm. Unfortunately, the test was unsuccessful, as the ICP algorithm was unable to compute the transform of the consecutive point clouds after approximately 10 meters, resulting in periodic odometry resets. Due to the frequent resets of the odometry, the resulting map was not even created as can be seen in Figure 7.2.

The cause of the failure was attributed to the low number of distinct features, which made it challenging for the ICP algorithm to function effectively. To address this issue, it

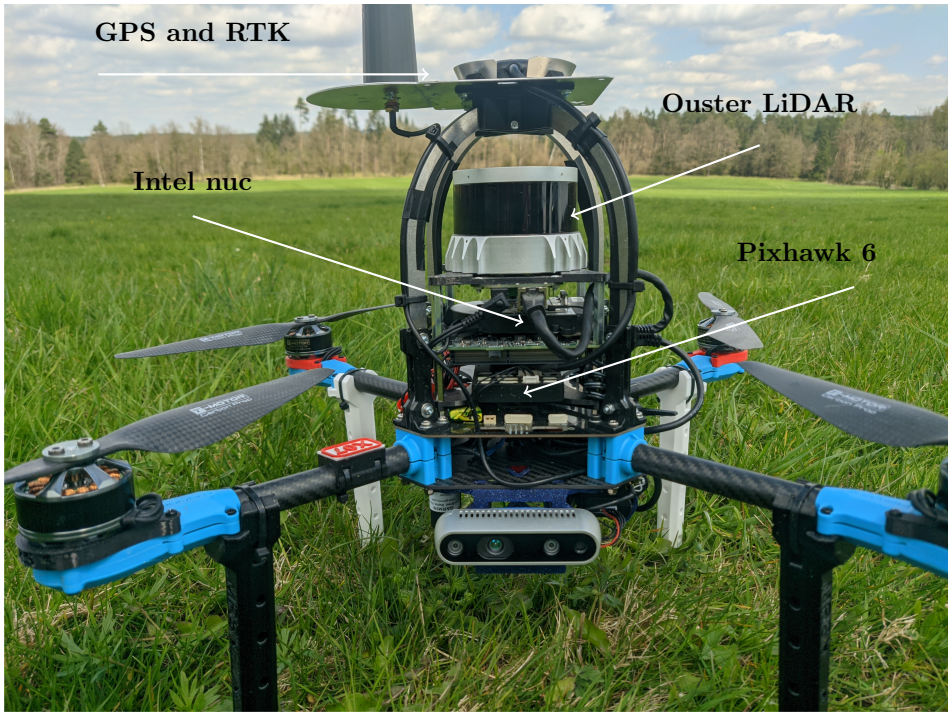


Figure 7.1: Closeup view of the drone used in the experiments.

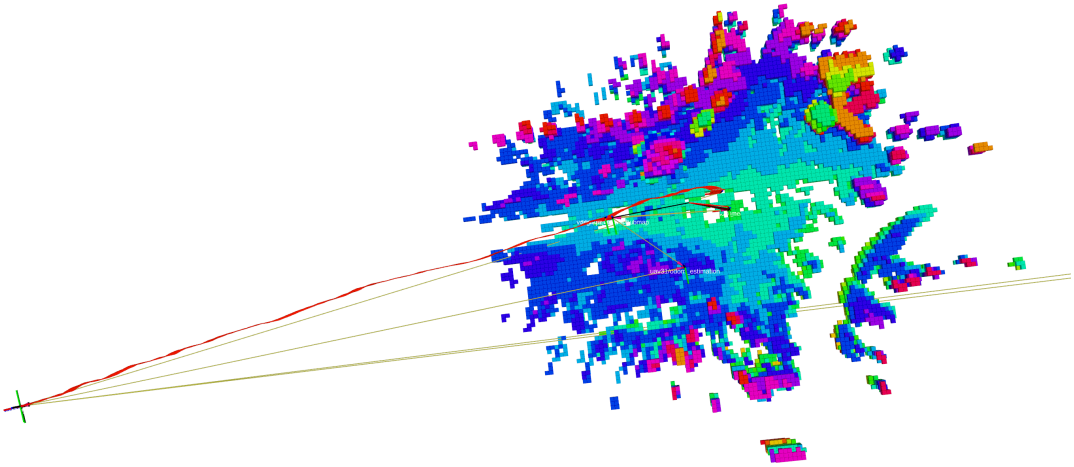


Figure 7.2: View of the unsuccessful creation of a volumetric map using ICP odometry.

was proposed that odometry be generated from signed distance fields by using points with a high curvature of the distance function, which could be processed by the ICP algorithm. This change to the ICP algorithm could be made while still being effective, as described in Section 3.6 this change would act as the selection of points used for the matching. A similar solution is proposed in [18]. Further exploration of this solution could help to improve the effectiveness

of the mapping pipeline in challenging GPS denied environments.

7.3 Manual flight with GPS

To assess the effectiveness of the proposed mapping pipeline, we conducted an experiment in a forested area using a UAV equipped with a noisy GPS sensor for odometry. The goal was to evaluate whether the proposed pipeline can generate an accurate and drift-free map in the presence of noisy odometry.

During the experiment, the UAV was flown in the forested area and the odometry estimate was obtained from the noisy GPS. Despite the inherent noise in the GPS measurements, the proposed pipeline was able to generate a map that was free from common artifacts created due to drifting odometry measurements.

Figure 7.3 shows the complete map with occupancy markers, the same flight is shown in Figure 7.4, however, it is visualized using a mesh. To point out how the submaps overlapped during the flight, Figure 7.5 shows each submap mesh in a different color.

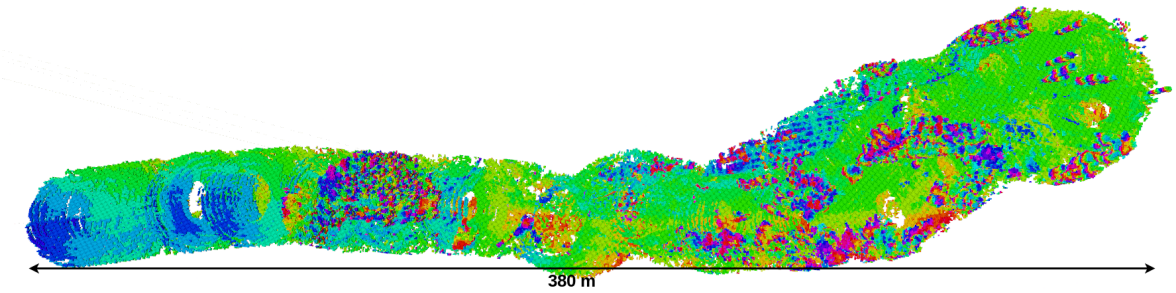


Figure 7.3: Visualization of the complete map with occupancy markers.

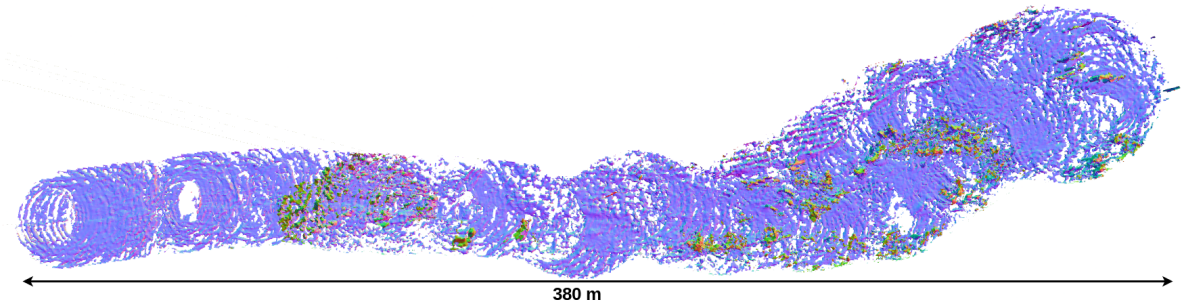


Figure 7.4: Visualization of the complete map as a combined mesh.

An important part of evaluating the created map is to inspect the created factor graph, which is visualized in Figure 7.6. As would be expected most constrains, which are visualized in black, were created in the right part of the map as the number of overlapping submaps is quite high. This points out that the performance of creating a globally consistent map will be better in flights with trajectories that are dense together, allowing for a higher number of registration and thus more constrains. The area that has a lot of factor graph constraints is shown in detail in Figure 7.7 and Figure 7.8. The difference between the two visualizations is important the mesh visualization does not accurately show the underlying map representations

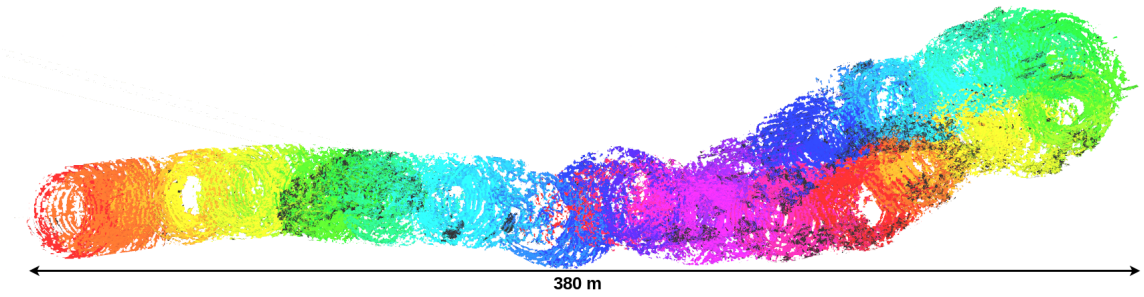


Figure 7.5: Visualization of the complete map where each mesh color visualizes a different submap.

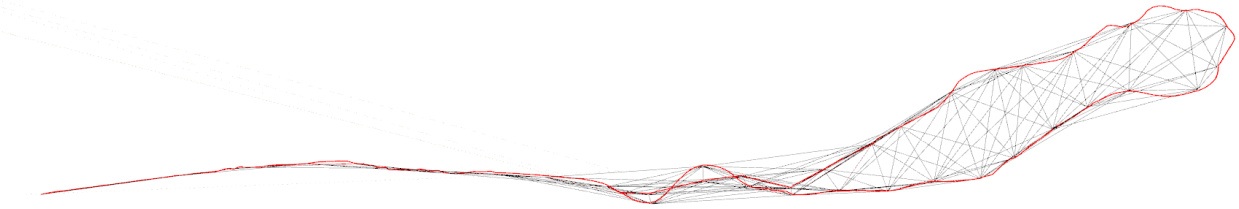


Figure 7.6: Visualization of the underlying factor graph (in black) and the flown trajectory (in red).

as can be seen on the ground or the trees in the foreground. In contrast the occupancy markers are created directly from the TSDF map representation and show the map more accurately.

The resulting map was of high quality and accurately represented the forested area. The map showed all the trees and obstacles in the area. This experiment demonstrated the effectiveness of the proposed pipeline for mapping in environments where odometry estimates are noisy or prone to drift.

This experiment provides evidence that the proposed pipeline can be effective in real-world applications where GPS odometry is commonly used.

7.4 Mapping and planning with GPS

The last experiment was conducted to test the performance of the pipeline, specifically the planning manager component. The experiment was carried out twice, once in an open environment, a meadow, and then in a sparse forest. During the experiments, the flights that were conducted were of an exploration nature, which is more challenging than planning a path in a known environment.

The first flight in the meadow was conducted to test the pipeline in a safe environment without obstacles. The planning manager had a goal position and had generated a safe trajectory to follow while ensuring that the UAV remained within a specified altitude range. This test was successful and the planning manger worked as expected, creating a safe trajectory with periodic replanning based on updated sensor measurements.

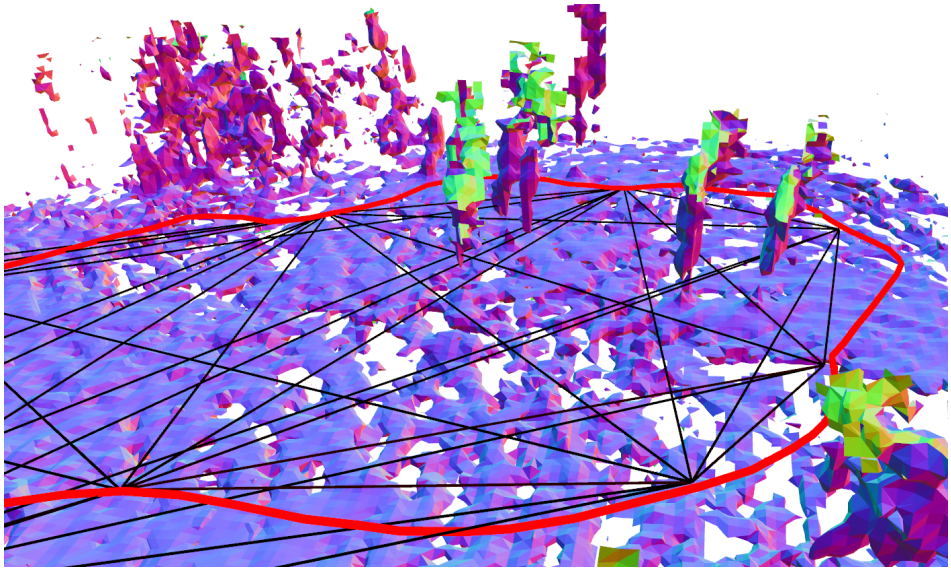


Figure 7.7: A visualization using mesh of the region of the map with high number of factor graph constrains(in black) and the flown trajectory (in red).

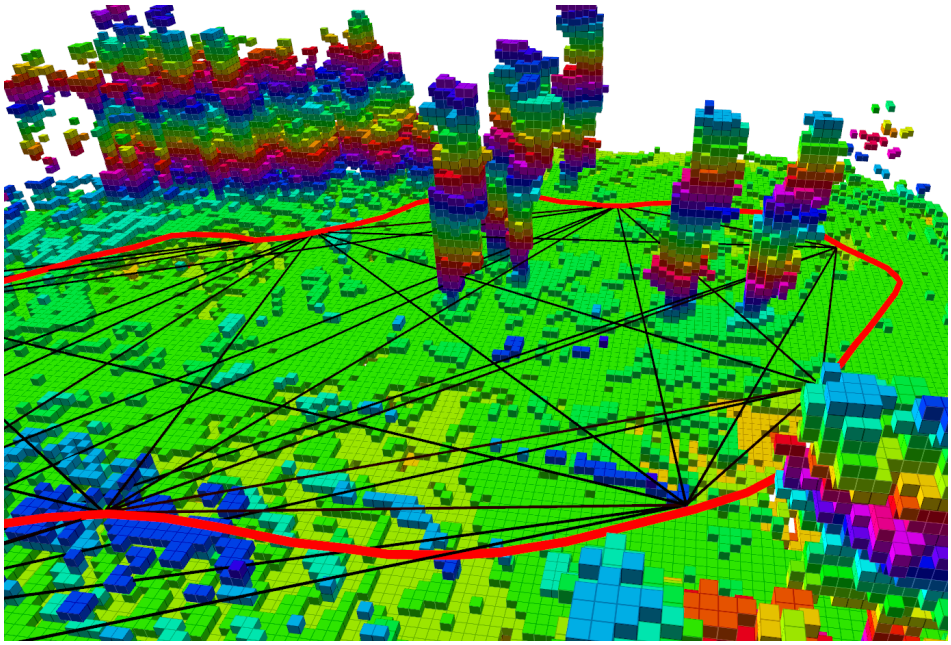


Figure 7.8: A visualization using occupancy markers of the region of the map with high number of factor graph constrains(in black) and the flown trajectory (in red).

The second flight in the forest has proven to be more challenging. Due to a bug in the code has introduced a memory leak when the desired goal was located in an obstacle, unfortunately this was not tested in the verification phase. This issue caused the UAV's system to run out of memory and the experiment had to be ended. The bug was subsequently fixed and the code was rigorously tested in a Gazebo simulation to prevent similar issues from arising in the future.

Despite the challenges faced during the experiment, the results were overall positive.

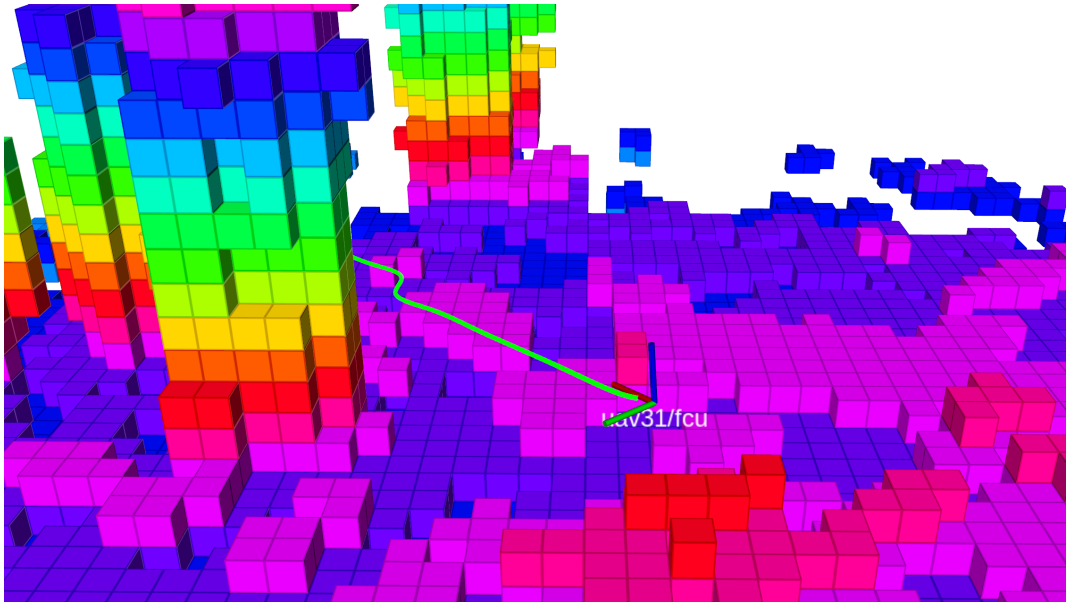


Figure 7.9: The figure shows in green the planned trajectory in the forest.

The proposed pipeline demonstrated that such an approach to mapping and planning can be taken. The bugs that were identified during the experiments were fixed and the pipeline was further improved based on real-world performance.

Chapter 8

Conclusion

This thesis presented a solution to the challenge of creating accurate volumetric maps of large unknown environments. A mapping and planning pipeline was proposed and implemented to be used with MRS UAV system. The created pipeline was extensively tested in simulation and on real-world data from a dataset. Furthermore, real-world experiments were conducted with various degrees of success. Chapter 2 summarized the bedrock tools used in this thesis like ROS and LiDARs. The theoretical overview of key principles for creating volumetric maps was given in Chapter 3. A brief introduction to path planning was given in 4. A detailed overview of creating the proposed pipeline was given in Chapter 5. The principle of combining two existing mapping frameworks, UFOMap and Voxgraph, and integrating it for the usage on MRS group's UAVs was detailed. A partially exploratory planning manager was created, which relies on both mapping frameworks to guarantee safe flights. The proposed pipeline was successfully tested in simulation and on a provided dataset, the results were discussed in Chapter 6. The verification was partially successful and the variables in the pipeline were tuned. The proposed pipeline was then tested in real-world experiments. Multiple different flights were done, however more rigorous experiments would have to be done for the pipeline to be usable real-world mapping and planning. In conclusion, all the goals set in the thesis assignment were all accomplished and even the testing of an alternative source of odometry for use in GPS denied environments was performed.

8.1 Future work

The work presented in this thesis provides an introduction to volumetric mapping and planning. Lessons should be taken from the created system and a fully integrated mapping and planning framework should be created. The concept of completely different methods for creating a local and global map should be kept. The limitations of the global mapper, like adapting it for dynamic environments and errors arising from the TSDF representation, should be addressed. An odometry generation system should be created and be based directly on one of the map's representations.

Chapter 9

References

[1] C. Huang, O. Mees, A. Zeng, and W. Burgard, "Audio Visual Language Maps for Robot Navigation," *arXiv preprint arXiv:2303.07522*, 2023.

[2] P. Petracek, V. Kratky, T. Baca, M. Petrlik, and M. Saska, "New Era in Cultural Heritage Preservation: Cooperative Aerial Autonomy," *IEEE Robotics and Automation Magazine*, pp. 2–19, Feb. 2023.

[3] M. Petrlik, P. Petracek, V. Kratky, T. Musil, Y. Stasinchuk, M. Vrba, T. Baca, D. Hert, M. Pecka, T. Svoboda, and M. Saska, "UAVs Beneath the Surface: Cooperative Autonomy for Subterranean Search and Rescue in DARPA SubT," *Field Robotics*, vol. 3, pp. 1–68, Jan. 2023.

[4] G. S. Camps, R. Dyro, M. Pavone, and M. Schwager, *Learning Deep SDF Maps Online for Robot Navigation and Exploration*, 2022. arXiv: 2207.10782 [cs.RO].

[5] F. Dellaert and G. Contributors, *borglab/gtsam*, version 4.2a8, May 2022. [Online]. Available: https://github.com/borglab/gtsam).

[6] D. Duberg and P. Jensfelt, "UFOExplorer: Fast and Scalable Sampling-Based Exploration With a Graph-Based Planning Structure," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2487–2494, 2022.

[7] D. Hert, T. Baca, P. Petracek, V. Kratky, V. Spurny, M. Petrlik, M. Vrba, D. Zaitlik, P. Stoudek, V. Walter, P. Stepan, J. Horyna, V. Pritzl, G. Silano, D. Bonilla Licea, P. Stibinger, R. Penicka, T. Nascimento, and M. Saska, "MRS Modular UAV Hardware Platforms for Supporting Research in Real-World Outdoor and Indoor Environments," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022, pp. 1264–1273.

[8] T. Musil, M. Petrlík, and M. Saska, "SphereMap: Dynamic Multi-Layer Graph Structure for Rapid Safety-Aware UAV Planning," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 007–11 014, 2022.

[9] J. Ortiz, A. Clegg, J. Dong, E. Sucar, D. Novotny, M. Zollhoefer, and M. Mukadam, *iSDF: Real-Time Neural Signed Distance Fields for Robot Perception*, 2022. arXiv: 2204.02296 [cs.RO].

[10] Y. Pan, Y. Kompis, L. Bartolomei, R. Mascaro, C. Stachniss, and M. Chli, "Voxfield: Non-Projective Signed Distance Fields for Online Planning and 3D Reconstruction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[11] T. Yang, Y. Li, C. Zhao, D. Yao, G. Chen, L. Sun, T. Krajnik, and Z. Yan, *3D ToF LiDAR in Mobile Robotics: A Review*, 2022. arXiv: 2202.11025 [cs.RO].

[12] T. Baca, P. Stibinger, D. Doubravova, D. Turecek, J. Solc, J. Rusnak, M. Saska, and J. Jakubek, "Gamma Radiation Source Localization for Micro Aerial Vehicles with a Miniature Single-Detector Compton Event Camera," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, Jun. 2021, pp. 338–346.

[13] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 26, pp. 1–28, 1 May 2021.

[14] M. G. Besselmann, L. Puck, L. Steffen, A. Roennau, and R. Dillmann, "VDB-Mapping: A High Resolution and Real-Time Capable 3D Mapping Framework for Versatile Mobile Robots," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 448–454.

[15] N. Funk, J. Tarrio, S. Papatheodorou, M. Popovic, P. F. Alcantarilla, and S. Leutenegger, *Multi-Resolution 3D Mapping with Explicit Free Space Representation for Fast and Accurate Mobile Robot Motion Planning*, 2021. arXiv: 2010.07929 [cs.RO].

[16] B. Yi, M. A. Lee, A. Kloss, R. Martín-Martín, and J. Bohg, *Differentiable Factor Graph Optimization for Learning Smoothers*, 2021. arXiv: 2105.08257 [cs.RO].

[17] D. Duberg and P. Jensfelt, "UFOMap: An efficient probabilistic 3D mapping framework that embraces the unknown," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6411–6418, 2020.

[18] A. Millane, H. Oleynikova, C. Lanegger, J. A. Delmerico, J. I. Nieto, R. Siegwart, M. Pollefeys, and C. Cadena, "Freetures: Localization in Signed Distance Function Maps," *Computing Research Repository*, 2020.

[19] V. Reijgwart, A. Millane, H. Oleynikova, R. Siegwart, C. Cadena, and J. Nieto, "Voxgraph: Globally Consistent, Volumetric Mapping Using Signed Distance Function Submaps," *IEEE Robotics and Automation Letters*, 2020.

[20] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and R. Daniela, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5135–5142.

[21] L. Han, F. Gao, B. Zhou, and S. Shen, *FIESTA: Fast Incremental Euclidean Distance Fields for Online Motion Planning of Aerial Robots*, 2019. arXiv: 1903.02144 [cs.RO].

[22] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[23] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena, "C-blox: A Scalable and Consistent TSDF-based Dense Mapping Approach," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 995–1002.

[24] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, *Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning*, 2018. arXiv: 1803.04345 [cs.RO].

[25] M. Achmad, G. Priyandoko, R. Roali, and M. Daud, "Tele-Operated Mobile Robot for 3D Visual Inspection Utilizing Distributed Operating System Platform," *International Journal of Vehicle Structures and Systems*, vol. 9, Sep. 2017.

[26] J. Chen and S. Shen, "Improving octree-based occupancy maps using environment sparsity with application to aerial robot navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3656–3663.

[27] T. Nam, J. Shim, and Y. Cho, "A 2.5D Map-Based Mobile Robot Localization via Cooperation of Aerial and Ground Robots," *Sensors*, vol. 17, p. 2730, Nov. 2017.

[28] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1366–1373.

[29] H. Mora, J. Mora-Pascual, A. Garcia-Garcia, and P. Martinez-Gonzalez, Oct. 2016.

[30] M. Klingensmith, I. Dryanovski, S. S. Srinivasa, and J. Xiao, "Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields," in *Robotics: science and systems*, Citeseer, vol. 4, 2015.

[31] J. Zhang and S. Singh, "LOAM : Lidar Odometry and Mapping in real-time," *Robotics: Science and Systems Conference (RSS)*, pp. 109–111, Jan. 2014.

[32] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous robots*, vol. 34, pp. 189–206, 2013.

[33] K. Museth, "VDB: High-Resolution Sparse Volumes with Dynamic Topology," *ACM Transactions on Graphics*, vol. 32, no. 3, 2013.

[34] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 1–11, 2013.

[35] S. Särkkä, *Bayesian Filtering and Smoothing*, ser. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013.

[36] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[37] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[38] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[39] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Third International Conference on 3-D Digital Imaging and Modeling*, 2001, pp. 145–152.

[40] P. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[41] J. Wilhelms and A. Van Gelder, "Octrees for Faster Isosurface Generation," *ACM Transactions on Graphics*, vol. 11, no. 3, pp. 201–227, Jul. 1992.

[42] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 116–121.

[43] D. Meagher, *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*, Oct. 1980.

[44] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

Chapter A

Appendix A

Table A.1 presents the folders in the root directory included in the provided `appendix.zip` file.

Filename	Comment
`ros_packages`	The selected ROS packages, Voxgraph and UFOMap mapping servers and path manager.
`tmux`	Tmux sessions used for testing the pipeline.

Table A.1: A list of attached files in the root folder.