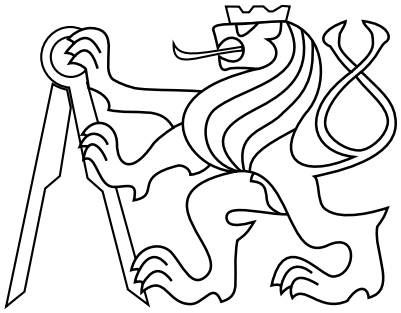
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



Comparison of Mapping Algorithms for Deployment Onboard Robots with Uncertain Pose

Bachelor's Thesis

Omar Msalha

Prague, May 2023

Study program: Open Informatics
Specialisation: Artificial Intelligence and Computer Science

Supervisor: Ing. Matouš Vrba

Acknowledgments

I would like to express my special thanks of gratitude to my thesis supervisor Ing. Matouš Vrba for his guidance and support in completing this project.

Author statement for undergraduate thesis

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, 26.05.2023 Omar Msalha

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Msalha Omar** Personal ID number: **499193**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Comparison of Mapping Algorithms for Deployment Onboard Robots with Uncertain Pose

Bachelor’s thesis title in Czech:

Srovnání mapovacích algoritmů pro jejich nasazení na robotech s nejistou pózou

Guidelines:

1. Research algorithms for 3D occupancy mapping relying on a 3D LiDAR sensor. Research publications comparing performance of such algorithms under uncertain sensor pose.
2. Integrate at least two mapping algorithms with the MRS UAV system in the Gazebo simulator.
3. Design and implement a method for robust comparison of mapping algorithms using a known environment in the simulator. Evaluate and compare the influence of sensor pose uncertainty on the map accuracy in different environments.
4. Design and implement a method for improving the map accuracy given known parameters of the pose uncertainty. Evaluate the improvement in simulated experiments

Bibliography / sources:

[1] Hornung, Armin, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees." Autonomous robots 34, no. 3 (2013): 189-206.
[2] Duberg, Daniel, and Patric Jensfelt. "UFOMap: An efficient probabilistic 3D mapping framework that embraces the unknown." IEEE Robotics and Automation Letters 5, no. 4 (2020): 6411-6418.
[3] Oleynikova, Helen, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning." International Conference on Intelligent Robots and Systems (IROS), 2017.
[4] P. Fankhauser, M. Bloesch and M. Hutter, "Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization," in IEEE Robotics and Automation Letters 3, no. 4, pp. 3019-3026, 2018.
[5] Underwood, James P., Andrew Hill, Thierry Peynot, and Steven J. Scheding, "Error modeling and calibration of exteroceptive sensors for accurate mapping applications," Journal of Field Robotics 27, no. 1 (2010): 2-20.

Name and workplace of bachelor’s thesis supervisor:

Ing. Matouš Vrba Multi-robot Systems FEE

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

Date of bachelor’s thesis assignment: **25.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

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

III. Assignment receipt

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

. _____ Date of assignment receipt

_____ Student's signature

Abstract

Pose uncertainty is one of the main problems when applying mapping methods onboard Unmanned Aerial Vehicles (UAVs). This thesis focuses on approaches regarding Light Detection and Ranging (LiDAR) based mapping on an autonomous UAV and current state of the art. These mapping methods, namely OctoMap, UFOMap, and Voxblox, are compared in accuracy using three different metrics, with respect to the pose uncertainty, and a knowledge of the ground truth poses of obstacles in the simulated environments. Furthermore, three potential improvements in mitigating the effects of the sensor noise in order to enhance overall mapping performance are discussed, first observing their effects individually on each method, and then using all of the proposed improvements in a single experiment and comparing it to the original result.

Keywords Unmanned Aerial Vehicles, Mapping algorithms, LiDAR, Mapping accuracy

Abstrakt

Nejistota pózy je jedním z hlavních problémů týkajících se mapování obsazenosti prostředí pomocí bezpilotních letounů (UAV). Tato práce se zaměřuje na metody používající LiDAR a momentální state of the art. Tyto mapovací metody, jmenovitě OctoMap, UFOMap, a Voxblox, jsou porovnány v přesnosti pomocí tří různých metrik, a to v závislosti na nejistotě pózy, a pomocí známých pozic překážek v simulovaných prostředích. Dále jsou diskutována tři potenciální vylepšení pro zmírnění efektu šumu ze senzorů, která by zvýšila celkovou schopnost těchto metod přesněji zmapovat okolní prostředí. Nejprve se pozoruje efekt těchto vylepšení na každé metodě individuálně, a poté všech vylepšení v jednom experimentu. Tyto výsledky jsou pak porovnány s původními výsledky bez diskutovaných vylepšení.

Klíčová slova Bezpilotní letouny, Mapovací algoritmy, LiDAR, Mapovací přesnost

Abbreviations

API Application Programming Interface

LiDAR Light Detection and Ranging

RTK Real-time Kinematic

SLAM Simultaneous Localization And Mapping

UAV Unmanned Aerial Vehicle

ESDF Euclidean Signed Distance Field

TSDF Truncated Signed Distance Field

Contents

Chapter 1

Introduction

As UAVs become more prevalent in various applications, the demand for accurate and efficient mapping of the environment increases. This includes using autonomous UAVs, as it allows for efficient and consistent coverage of large areas and it reduces the cost and time required for mapping, compared to human operated UAVs or manned aerial surveys. Various sensors can be used for the mapping, such as LiDAR, RGB cameras, and RGB-D cameras. Light Detection and Ranging (LiDAR) sensors are widely used onboard UAVs for mapping and obstacle avoidance due to their high precision and ability to capture detailed information about the surroundings using laser beams. There are 3 types based on the dimension of the captured measurement, 1D LiDAR (commonly known as a rangefinder), 2D LiDAR, which scans the environment in a 2D plane and is commonly used for obstacle detection and avoidance, and 3D LiDAR, which provides full information about the surroundings, making it ideal for creating 3D maps. The main downside is their relatively high price. RGB cameras capture color images of the environment and are widely available and relatively inexpensive, making them a popular choice for mapping applications, although RGB cameras can be limited in their ability to capture accurate depth information, which can make it challenging to create 3D maps. RGB-D cameras combine the benefits of both RGB cameras and LiDAR sensors, as they provide the depth of the captured image. However, accurately mapping the environment while affected by pose uncertainty, remains a significant challenge.

Figure 1.1: A UAV built on quadcopter Tarot T650 airframe [6] with a 3D LiDAR used for the experiments in this thesis.

This thesis focuses on three of the most popular LiDAR mapping techniques. These techniques are compared in their accuracy — OctoMap [17], which is older and is based on the use of octrees [28], UFOMap [7], which builds upon the OctoMap and tries to improve it in memory efficiency and accuracy, and Voxblox [13], which uses a completely different approach by creating Euclidean Signed Distance Fields (ESDFs) maps from Truncated Signed Distance Fields (TSDFs). The accuracy of mapping techniques is a critical factor in their successful deployment in various applications, such as infrastructure inspection, agriculture, and search

and rescue operations [20]. It can be affected by a variety of factors, including sensor type, processing algorithms, and environmental conditions. A problem that often arises when using mapping techniques for UAVs is the limitation of its payload. To tackle this problem, memory efficient and fast localization algorithms have to be used to navigate through the environment in real-time, often in cost of accuracy caused by pose uncertainty of the UAV.

The pose uncertainty is a measure of the accuracy of the vehicle's position and orientation estimation in a given static world frame. It arises from multiple factors, such as sensor noise or sudden environmental changes, which can significantly affect the quality of mapping and data collected.

The mentioned methods are compared in a simulation using the Gazebo simulator software[1], with the testing worlds created for the use of these methods and the ground truth coordinates of every obstacle and its volume known. These coordinates are used for the evaluation of the accuracy. The goal is to compare the methods in the same resolution of voxels and using the same pointcloud in the mapping process, while gradually increasing noise and observing its effect on the accuracy, and to propose a possible improvement of the algorithm that would take the pose uncertainty into consideration, thus mitigating the effects of the sensor noise. The experiments ran on the UAV shown in Figure 1.1 with an Ouster LiDAR model OS1-128 in Gazebo simulator integrated into MRS system [4]. Three different environments were considered – forest-, urban- and maze-like environment.

1.1 Related works

Comparison of OctoMap and UFOMap, discussed in this work, in terms of memory usage, number of nodes, cloud insertion time, and collision checking time is presented in [7]. The goal of this thesis is to compare them in overall mapping accuracy, i.e. in robustness to increasing pose uncertainty, along with Voxblox. These methods are further described in the following section. The authors of OctoMap [17] demonstrated its accuracy directly in their publication. They used an environment to which they created a 3D grid with filled cells of maximum-likelihood states (free or occupied) as a reference to measure the accuracy as the percentage of correctly mapped cells in all 3D scans. [10] demonstrates the applicability of UAV LiDAR for mapping coastal environments. A custom-built UAV-based mobile mapping system is used to simultaneously collect LiDAR and imagery data. The quality of LiDAR, as well as image-based pointclouds, are investigated and compared over different geomorphic environments in terms of their point density, relative accuracy, and area coverage. [12] presented a transparent framework for the automated aerial triangulation of UAV images and an accuracy analysis of the derived 3D image-based point cloud reconstructions through a check point analysis using Real-time Kinematic (RTK) GPS and derived a root mean square error. The achieved accuracy in the proposed aerial triangulation framework was below 5 cm and it demonstrated superior performance when dealing with acquired UAV images containing repetitive pattern and significant image distortions.

Multiple works, such as [11], [3], [5], [19], compared various Simultaneous Localization And Mappings (SLAMs) methods and technologies, being relevant to this thesis, as they compare these methods in mapping accuracy, besides algorithm efficiency, CPU load, or memory load. [11] compared three different mapping systems, namely the Mattersport with depth cameras, Slammer, and Navis. It utilized corner points of rectangular objects as features for

[1] Gazebo simulator: `https://gazebosim.org/about/`.

the comparison. The extracted corners are re-examined by an outlier detector to filter out the mismatched results and evaluate the mapping accuracy. [5] compared three modern and robust visual SLAMs: ORB-SLAM3, OpenVSLAM, and RTABMap. This paper benchmarked them against each other using several different datasets and found out that OpenVSLAM was the overall best general purpose technique for the broadest range of service robot types, environments, and sensors. [19] proposed a framework for benchmarking SLAMs. They do not compare the map itself but only the poses of the robot during the data acquisition. This allows them to objectively compare SLAM approaches that use different sensors or estimation techniques.

1.2 Preliminaries

This section discusses the methods selected for the comparison in the mapping accuracy and describes their general functionality and concepts, as they are used further in this thesis for the experiments and are important to fully understand before introducing the metrics in which they are compared, along with the possible improvements.

1.2.1 OctoMap

Octomap [17] is one of the most popular mapping frameworks today. Its goal is to minimize memory consumption while allowing efficient and probabilistic updates of occupied and free space. Endpoints of distance sensors, such as a laser rangefinder or a stereo camera, measure the occupied space, while free space is the observed area between the sensor and the endpoint. OctoMap also introduced a method for map compression that further reduces memory requirements by combining coherent map volumes. The first introduction of the OctoMap framework was in 2010. Since then, it is continuously being improved and used in a large number of robotics research subjects, such as [18], [16], or [15].

Measurements of the surroundings are stored in a 3D discretized grid containing cubic volumes of equal size (voxels). Early works using such representation, for example [21] or [22], had a major drawback, the bounding box of the grid map had to be known beforehand, so that the map could be initialized. In outdoor environments or when there was a need for higher resolution, memory consumption started to be prohibitive. One way to avoid this problem would be to store only 3D pointclouds measured by the sensors. But with this method, unknown or free areas are not modeled, and dynamic objects cannot be dealt with directly, thus making it useful only for high precision sensors in static environments. A popular approach in robotic mapping these days is to use octrees [28].

Octree is a data structure representing objects with defined resolution in octary trees. In OctoMap, these objects are voxels [25]. Each node in the tree recursively subdivides into 8 voxels with the same volume until the minimum voxel size (resolution) is reached. The leaf nodes hold log-odd values of being occupied.

Sensor readings are integrated using a probabilistic sensor model similar to [27]. The probability $P(n|z_{1:t})$ of leaf node n being occupied is then estimated with

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1}, \tag{1.1}$$

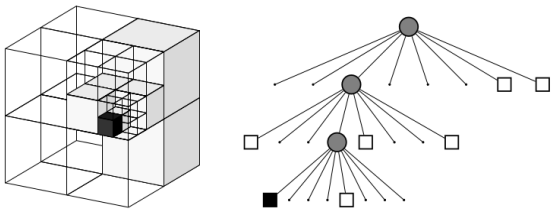


Figure 1.2: Example of an octree storing free (shaded white) and occupied (black) cells. The volumetric model is shown on the left and the corresponding tree representation on the right. [17]

where z_t is current measurement, $P(n)$ is a prior probability and $P(n|z_{1:t-1})$ is the previous estimate. With a common assumption of a uniform prior probability this formula can be rewritten into log-odds notation as

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t), \tag{1.2}$$

where

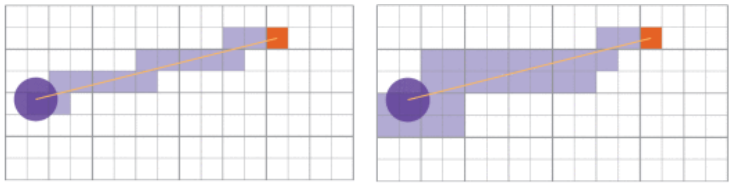
$$L(n) = \log\left[\frac{P(n)}{1 - P(n)}\right]. \tag{1.3}$$

The sensor model of a laser range finder used by OctoMap assumes that the endpoints of a measurement correspond to obstacle surfaces and that the line between the laser origin and the endpoint does not contain any obstacle. This assumption is used for the integration of new measurements into the map, i.e. updating the voxel occupancy values. To determine which voxels need to be updated, a ray-casting operation is performed by the algorithm described in [26], which approximates the laser beam. There are two ways of integrating the measurements in OctoMap. For each point in the point cloud, update the occupancy value of all voxels from the origin of a ray cast to the endpoint, or discretize the point cloud first so only one ray is cast for all the endpoints that would end up in the same voxel. The voxels along the beam are updated using the eq. (1.2) with the following inverse sensor model:

$$L(n|z_{1:t}) = \begin{cases} l_{occ} & \text{if the beam is reflected within the voxel} \\ l_{free} & \text{if the beam traversed the voxel} \end{cases}. \tag{1.4}$$

Throughout the experiments in this thesis, log-odd values of $l_{occ} = 2.95$ and $l_{free} = -0.2$ are used, corresponding to probabilities of 0.95 and 0.45 for occupied and free voxels, respectively. To avoid building up over-confidence in the values, OctoMap uses clamping thresholds. The clamping thresholds used for the experiments are set to $l_{min} = -2$ and $l_{max} = 3.5$, corresponding to probabilities of 0.12 and 0.97.

When creating a 3D map, a probability threshold can be set to distinguish between two discrete states of the node – occupied and free, so it can be used for navigation and planning. OctoMap does not model the unknown space explicitly, which may be a bottleneck for exploration, path planning, or collision checking algorithms. One workaround is to treat unknown space as free, but this approach increases the possibility of collision. Another way is to treat it as occupied space which can be prohibitive for exploration algorithms. UFOMap takes this problem into consideration.



(a) (b)

Figure 1.3: Comparison of the UFOMap integrators on a single ray. (a) shows the Simple/Discrete integrator and (b) shows the Fast discrete integrator [7]. The sensor is located to the left (the circle). The lines are the lines being traced from the sensor to a measured point. The violet and orange cells are marked free or occupied by the integrator, respectively.

1.2.2 UFOMap

First published in 2020 [7], the UFOMap allows for real-time colored octree mapping at high resolution. This method builds upon the OctoMap approach and tries to improve it by handling the unknown space efficiently. UFOMap, just like OctoMap, uses an octree-based data structure to represent the environment. The main difference between OctoMap and UFOMap is that each node in an UFOMap octree holds a probability associated with being occupied, free, or unknown, hence the name. It allows for a better use of exploration, path planning, or collision checking algorithms. The nodes also store three indicators – i_f, i_u and i_a, which are not in OctoMap. i_f and i_u indicate whether the node contains free or unknown space, respectively. The occupancy value is enough to indicate if the node contains occupied space, so it is not needed. The last indicator i_a, indicates whether all of the node's children are the same. This is useful when applying pruning to the tree, which can save some memory.

Another difference to OctoMap is the point cloud insertion time. The UFOMap insertion time of the same pointcloud is one to two times faster than the OctoMap insertion as presented in [7], which is not ideal in many applications. UFOMap addresses this shortcoming with three different methods of integration of the point cloud, each faster than the previous but with less accurate results. The most accurate integrator, called the *Simple integrator*, is the same as in OctoMap. For each point in the point cloud, decrease the occupancy value of all nodes from the origin of a ray cast to the endpoint and increase the value for the node in which the point lies. Next integrator is called the *Discrete integrator*, it discretizes the point cloud first. This way, only one ray is cast for all the endpoints that would end up in the same node. This also exists in OctoMap. The last one is the *Fast discrete integrator*, where the ray casting and discretization is done at multiple depths of the octree. In this thesis, only the fastest method called *Fast discrete integrator* is used because of its distinguishness to OctoMap, as the accuracy evaluation results when using *Simple integrator* should be almost identical to OctoMap when using the same occupancy thresholds and the sensor model.

The last significant difference from Octomap is the use of three node types: *inner node*, *inner leaf node* and *leaf node*. The *inner node* stores the log-odd occupancy value along with a pointer to an array of its children and the three indicators discussed earlier. In UFOMap, children are stored directly in an array instead of having an array of pointers pointing to each child like in OctoMap. This saves 64 bytes for the *inner nodes* when the node has 8 children, compared to OctoMap, but it means that the node in UFOMap has 8 children, or no children. The *inner leaf node* also stores the log-odd occupancy value, but the children array pointer

is null. When the *inner leaf node* gets a child, it is considered an *inner node*. This addition of the third node type, *leaf node*, which is a node with no children in the bottom depth of the tree and is more simplistic by storing only the log-odd value and no children array pointers, further reduces memory consumption up to 14% according to [7]. In OctoMap, *inner leaf node* is the only type of *leaf node* that exists.

1.2.3 Voxblox

Voxblox is another state-of-the-art open source mapping system first published in 2017 [13]. The authors proposed a method based on incrementally building Euclidean Signed Distance Fields (ESDFs) from Truncated Signed Distance Fields (TSDFs). ESDF is a data structure that stores information about the nearest object or surface from each point in 3D space. It is valuable for fast local planning, exploration, or trajectory optimization. In terms of Voxblox, these points in 3D space are represented by voxels with a predefined resolution in a hash table, which allows for faster lookups with an asymptotic complexity of $\mathcal{O}(1)$ compared to the OctoMap's and UFOMap's octree structures with a complexity of $\mathcal{O}(\log n)$. These nodes store the value of the distance to the nearest occupied voxel. TSDF data structure is similar to ESDF, but it is limited to a fixed range to an obstacle. Sets of these voxels, TSDF or ESDF, are called layers. Each layer contains independent blocks that are indexed by their position in the map, and each block contains a predefined fixed number of voxels. Additionally, Voxblox produces surface meshes that allow human operators to get a better understanding of the UAV's surroundings.

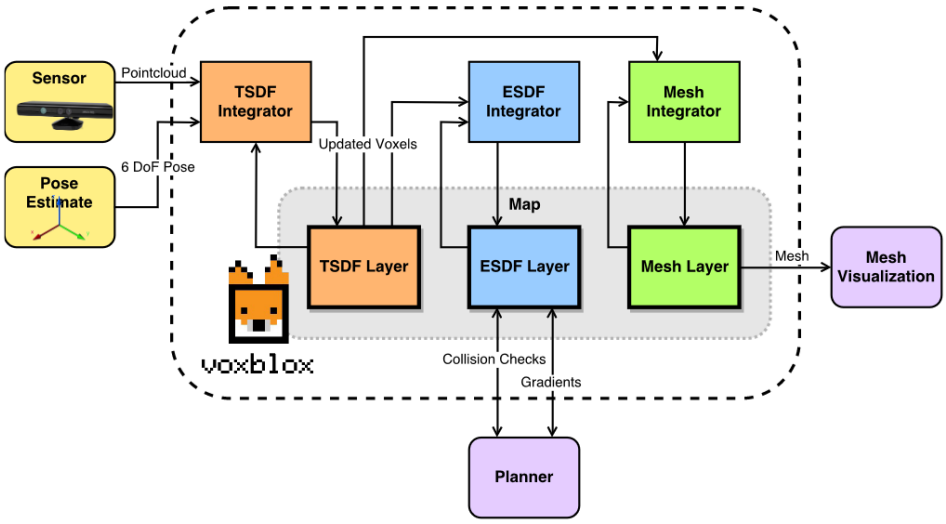


Figure 1.4: A system diagram for Voxblox [13], showing how the map layers (TSDF, ESDF, mesh) interact with each other using integrators.

The overall system of Voxblox, shown in Figure 1.4, can be divided into two parts: incorporating sensor data into a TSDF and propagating the updated voxels from TSDF to update the ESDF and mesh.

A. *TSDF construction*

The construction of TSDF layers begins with the initialization of an empty voxel grid with a predefined resolution, block size and maximum ray range. Each TSDF layer is then

populated with distance values, which are computed based on the sensor data and the current map estimate.

To populate the TSDF layer, Voxblox uses two methods: weighting, where the distance values are computed by averaging the distances between the sensor measurements and the voxel centers, and merging, where multiple points mapping to the same voxel are merged into one mean point, so only one ray cast is done. These methods help to reduce the effects of noise and uncertainty in the sensor measurements and make the integration of sensor data faster.

B. *ESDF construction*

Opposite to TSDF, an ESDF map can dynamically change its size, as new voxels can be added to the map at any time. The ESDF voxels are updated using the neighbouring voxels in the TSDF layer in a certain radius. The algorithm itself is based on the idea of *wavefronts* – waves that propagate from a start voxel to its neighbours, updating their distances and putting them into a queue to further propagate to other neighbours.

1.3 Problem formulation

A UAV equipped with a 3D LiDAR flies through an environment with obstacles. The UAV is assumed to be mapping the surroundings using one of the discussed methods – OctoMap, UFOMap or Voxblox. The accuracy of the selected method is then evaluated in several metrics using known ground truth positions of the obstacles and the maps received from the selected method. The sensors of the UAV are assumed to be affected by noise, which is gradually increased and the effect of it on the accuracy is observed. The goal is to compare the selected methods in accuracy and robustness to increasing pose uncertainty, and to propose an improvement that would increase the accuracy of the resulting map with noisy sensors.

Chapter 2

Methodology

The selected methods are compared in three simulated testing environments, where the ground truth coordinates of every obstacle and its volume are known. A UAV equipped with a 3D LiDAR flies through these environments and saves the collected pointclouds. The goal is to compare the methods in several metrics with the same resolution of voxels, mapping range, sensor model, and using the same pointclouds in the mapping process, while gradually increasing noise of the UAV's pose and observing its effect on the accuracy. The mapping range refers to the maximum distance at which the mapping algorithm is allowed to generate a local map, i.e. the points in the current measurement that are further from the sensor than this maximum distance are not used for the integration. Unless specified otherwise, in our experiments, we used the mapping range of 50m, voxel resolution of 0.3m and the sensor model corresponding to probabilities of 0.95 and 0.45 for occupied and free voxels, respectively, as described in section 1.2.1. The resulting accuracy is evaluated using three proposed metrics. To calculate these metrics, all voxels in the map, classified free or occupied by the selected method, are checked for any obstacle intersection.

2.1 Testing environment

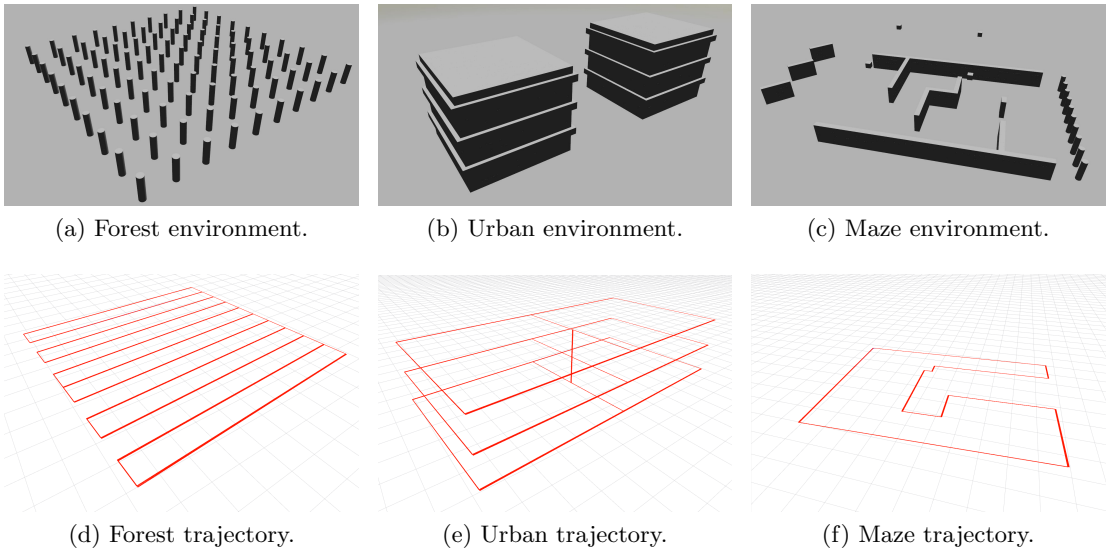
Three simulation environments were created using the Robot Operating System (ROS)[1] and Gazebo simulator[2]. The first environment consists of cylinders with 1 meter in diameter and 4 meters in height with 3 meters wide corridors between each to resemble a deep forest. The second environment consists of two three-story buildings to represent an urban area and the last one is a maze-like environment with narrow corridors and wide openings with some distant objects floating in the air. All three maps, shown in Figure 2.1, have flat ground to ensure accuracy of the mapping evaluation is as simple as possible.

With these environments, MRS group's ROS UAV packages [4] were used to run a simulation with a UAV platform based on the Tarot T650 frame [6] mounted with the Ouster OS1-128 3D LiDAR. The UAV navigated along predefined collision-free looped trajectories shown in Figure 2.1 for 3 iterations while recording ground truth UAV poses and pointcloud data, and saved them into a rosbag.

2.2 Sensor noise

To observe the effects of sensor noise on the mapping performance, the rosbag with the recorded ground truth UAV poses and pointcloud data is first transformed from the ground

[1]Robot Operating System: https://www.ros.org/
[2]Gazebo simulator: https://gazebosim.org/about



(a) Forest environment. (b) Urban environment. (c) Maze environment.

(d) Forest trajectory. (e) Urban trajectory. (f) Maze trajectory.

Figure 2.1: The simulation environments created for testing and evaluating purposes and their corresponding trajectories for the UAV to follow.

truth data to the noisy data using the noise model described in [1]. A single (noisy) point $\boldsymbol{p}_m$ measured by a ray of a LiDAR sensor with a unit direction vector $\boldsymbol{d}$ and a range l_m, can be expressed in the world frame as

$$\boldsymbol{p}_m = l_m \mathbf{R}_m \boldsymbol{d} + \boldsymbol{t}_m, \tag{2.1}$$

where $\boldsymbol{t}_m$ is the translation vector and $\mathbf{R}_m$ the rotational matrix from the world frame to the LiDAR's frame. Let us define vector a $\boldsymbol{w}$ that is drawn from a multivariate Gaussian distribution with a covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{w}}$ as

$$\boldsymbol{w} = \left[l_n, \boldsymbol{t}_{\boldsymbol{n}}^{\intercal}, \alpha_n, \beta_n, \gamma_n\right], \tag{2.2}$$

$$\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{w}}). \tag{2.3}$$

Because the ground truth point $\boldsymbol{p}_{gt}$ is known (recorded from the simulation) along with the range l_{gt} and transformation $\mathbf{R}_{gt}$, $\boldsymbol{t}_{gt}$, we can model the resulting noisy measurements as

$$l_m = l_{gt} + l_n, \tag{2.4}$$

$$\boldsymbol{t}_m = \boldsymbol{t}_{gt} + \boldsymbol{t}_n, \tag{2.5}$$

$$\mathbf{R}_m = \mathbf{R}_{gt}\mathbf{R}_n = \mathbf{R}_{gt}\mathbf{R}_x(\alpha_n)\mathbf{R}_y(\beta_n)\mathbf{R}_z(\gamma_n), \tag{2.6}$$

where l_n, $\boldsymbol{t}_n$ and α_n, β_n, γ_n represent the measurement noise drawn from the multivariate Gaussian distribution. The recorded data is then transformed with the added measurement noise picked randomly from the multivariate Gaussian.

To have the measurement noise represent different environmental conditions or sensor setups, 5 different sets of parameters for the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{w}}$ based on empirical data were used. The covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{w}}$ can be expressed as a diagonal matrix with a vector of

parameters $\boldsymbol{\sigma}^2$ on the diagonal, assuming the elements of $\boldsymbol{w}$ are independent:

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_l & \sigma_x & \sigma_y & \sigma_z & \sigma_\alpha & \sigma_\beta & \sigma_\gamma \end{bmatrix}^\intercal, \tag{2.7}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{w}} = \begin{bmatrix} \sigma_l^2 & & \\ & \ddots & \\ & & \sigma_\gamma^2 \end{bmatrix}. \tag{2.8}$$

The chosen parameters and sensor setups that they represent used for the experiments and accuracy evaluation can be seen in Table 2.1.

Parameters $\boldsymbol{\sigma}$							
Sensor setup	σ_l[m]	σ_x[m]	σ_y[m]	σ_z[m]	σ_α[rad]	σ_β[rad]	σ_γ[rad]
GPS, magnetometer	0.1	0.4	0.4	0.05	0.05	0.05	0.05
RTK, magnetometer	0.1	0.1	0.1	0.05	0.05	0.05	0.05
RTK, noisy magn.	0.1	0.1	0.1	0.05	0.15	0.15	0.15
SLAM	0.1	0.03	0.03	0.02	0.01	0.01	0.01
SLAM, noisy LiDAR	0.2	0.03	0.03	0.02	0.01	0.01	0.01

Table 2.1: Table of parameters used for the multivariate Gaussian which transform the ground truth data to the noisy data.

The first sensor setup assumes a Global Positioning System (GPS) with a magnetometer. Measurements from the GPS can be affected by many factors, such as reflections of signals from tall buildings, obstruction of the signal by heavy tree cover, signal arrival time measurmement, number of satellites in clear sight, numerical errors or atmospheric effects, resulting in wide range of possible accuracy errors [23]. Although, with the ongoing modernization of the GPS signals, objects can be localized within a few meters or even centimeters, providing highly useful and reliable information for a wide range of applications.

In magnetometers, source of the noise can be either internal, meaning it originates from within the magnetometer system caused by the electronic components or conductive materials, or external, which originates from outside of the magnetometer system, such as temperature fluctuations, vibrations or electromagnetic interference. The overall measurement error for low-cost magnetometers can be around 1° [24] or even lower when using a calibration method introduced in [9].

Another setup is an RTK with a magnetometer. RTK is a positioning technique based on global navigation satellite systems, such as GPS. It uses a fixed base station with a known location that communicates with the mobile receiver, for example a UAV, using a radio link. The base station provides the receiver with corrections to its position. These corrections are based on the known position of the receiver and they help mitigate the positioning errors caused by atmospheric distortions, signal arrival time, etc. RTK can reliably localize objects with sub-meter or even centimeter-level accuracy [8].

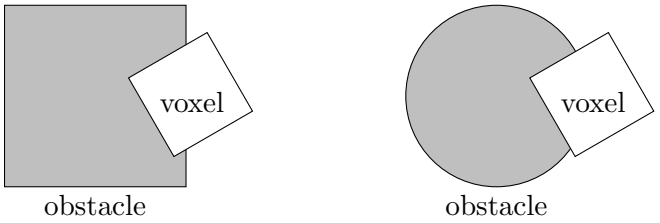
The last setup assumes a SLAM method. SLAM is a set of algorithms that use various sensors like LiDAR, RGB cameras, RGB-D cameras, etc., for self-localization of the UAV, and concurrently constructs a 2D or 3D map of features in the environment. According to experiments in [14], SLAM methods can attain high precision in position estimation, with translational deviations of only a few centimeters, as well as heading accuracy, with rotational deviations within tenths of a degree. As for the LiDAR, accuracy is specified by the

manufacturer of the selected sensor Ouster OS1-128[3] to be ± 0.3 to 2 centimeters. However, for the purposes of this thesis, an increased deviation is beneficial in order to effectively observe the impact of LiDAR inaccuracies on the resulting map.

The deviation in the z axis is assumed to be lower in all cases, given by a rangefinder mounted on the bottom of the UAV that is pointing to the ground.

2.3 Obstacle intersections

The environments for the simulation are created using 3D geometrical primitives object in Gazebo – cubes and cylinders, with varying dimensions, position and heading. This results in voxels having the bottom and upper sides parallel with the bottom and upper sides of the obstacles. The intersection of a voxel with an obstacle can then be solved separately for the 2D projection to the XY-plane and height difference as shown in Figures 2.2 and 2.3.



(a) A cubic obstacle and a voxel.

(b) A cylindric obstacle and a voxel.

Figure 2.2: Examples of voxel and obstacle intersections, where Fig. 2.2a represents possible intersection with a cubic obstacle and the Fig. 2.2b represent the intersection with a cylindric obstacle.

The 2D intersection problem is heavily studied in computer graphics. One way to approximate the intersection area is sampling – divide the voxel rectangle into grid of smaller squares, and then check if the squares are inside of the obstacle polygon. This approach does not provide an exact area and can be slow when using smaller grids for more accurate result, but is easy to implement. Another option is to use line clipping [2] – find intersection points of line segments, i.e. of rectangle sides and find vertices that are inside of the other rectangle, then calculate the area of the polygon defined by those intersection points and vertices.

For this thesis, the C++ Boost library[4] is used. It provides support for tasks and structures such as linear algebra or image processing and has every algorithm needed for this purpose already implemented. However, to calculate the area of an intersection with a circle, the circle has to be approximated using line segments. The number of line segments directly impacts the speed of the calculation, so for the purpose of this thesis, 36 line segments are used, which is sufficient for high precision, while not losing much performance, based on empirical observations.

As the intersection is calculated only in 2D, to get the total volume of the 3D intersection, the height overlap, denoted as Δz in Figure 2.3, is calculated. Let us name the center

[3]Ouster OS1: `https://levelfivesupplies.com/product/ouster-os1-128-lidar-sensor//`.
[4]Boost library: `https://www.boost.org/`.

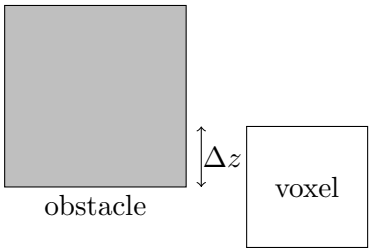


Figure 2.3: Example of a height difference between an obstacle and a voxel.

point of the obstacle $\boldsymbol{o}$ and the center of the voxel $\boldsymbol{v}$:

$$\boldsymbol{o} = \left[o_x, o_y, o_z\right]^\intercal, \tag{2.9}$$

$$\boldsymbol{v} = \left[v_x, v_y, v_z\right]^\intercal. \tag{2.10}$$

Then, Δz can be obtained as

$$\Delta z = \min\left(o_z + \frac{o_h}{2}, v_z + \frac{v_h}{2}\right) - \max\left(o_z - \frac{o_h}{2}, v_z - \frac{v_h}{2}\right), \tag{2.11}$$

where o_z is the z coordinate of the obstacle's center, v_z is the z coordinate of the voxel's center, o_h is the obstacle's height and v_h is the voxel's side length (grid resolution). The resulting intersection volume V_{obs}^{vox} is then calculated by

$$V_{obs}^{vox} = \begin{cases} \Delta z S & \text{if } \Delta z > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2.12}$$

where S is the intersection area approximated with the Boost library. However, this only evaluates the intersection volume with known obstacles, such as walls and trees, that were added to the simulation.

What remains to be evaluated is the intersection with the ground. For evaluation and comparison purposes, the ground has to be given some volume. This volume is determined by voxel height v_h and by the considered area of the simulated environment. Because the ground plane is at $z = 0$, the resulting obstacle representing ground is a rectangle across the whole considered area with height v_h and center at $\left[0, 0, -\frac{v_h}{2}\right]^\intercal$. As mentioned earlier, the ground in all the used environments is flat, therefore the bottom and top sides of the voxels are also parallel to the ground. This simplifies the problem of evaluating the intersection with the ground, which can be computed only with the knowledge of voxel size, voxel position on z axis and center of the ground obstacle. The resulting volume of intersection of a voxel with the ground V_{gr}^{vox} can be computed as

$$V_{gr}^{vox} = \begin{cases} \Delta z v_h^2 & \text{if } \Delta z > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2.13}$$

with the only distinction from eq. (2.12) being the intersection area S, which is known and is given by the voxel being a cube with side length v_h.

2.4 Occupancy mapping comparison metric

As OctoMap, UFOMap and Voxblox have Application Programming Interfaces (APIs) available in C++, a C++ ROS node was implemented using the ROS, OctoMap, UFOMap,

Voxblox and Boost packages. This node subscribes to map messages published by OctoMap, UFOMap and Voxblox nodes, which contain current information about the mapped area and related octrees with occupancy values. The node then traverses the leaf nodes (as well as inner leaf nodes for the UFOMap) of the octrees and evaluates the intersections of occupied voxels with obstacles based on the known ground truth positions of the obstacles. These mapping methods are then compared in several metrics.

A. *Unmapped Obstacle Volume Metric*

Let us consider the eq. (2.12) as a function obstacle(v) and the eq. (2.13) as a function ground(v), which take a voxel v as an argument and return their total intersection volume with all obstacles and ground. The sum of these functions over all voxels classified as occupied $\mathcal{O}$ in the map,

$$V_{int} = \sum_{v \in \mathcal{O}} \text{obstacle}(v) + \text{ground}(v), \tag{2.14}$$

is the total volume of the correctly classified occupied portion of the map, defined as V_{int}. With the known positions of obstacles and their shapes and sizes, the overall volume of obstacles can be computed as a sum of all obstacle volumes plus the volume of the ground obstacle in the considered area. Let us name the overall obstacle volume V_{obs}. This volume can then be compared with the evaluated sum of intersections V_{int} as a ratio of unmapped obstacle volume E:

$$E = \frac{V_{obs} - V_{int}}{V_{obs}}. \tag{2.15}$$

This ratio should converge to zero as the obstacles in the environment are explored and the map is filled over time. However, this metric fails to indicate the percentage of correctly classified voxels or how much of the voxel volume mapping the obstacles exceeds into free space, therefore, two additional metrics are provided.

B. *Voxel Classification Metric*

The mapping methods are also compared by volume classification accuracy. Each voxel is categorized into one of the four following types depending on their intersections with the obstacles:

- *True Positive (TP)*,
- *True Negative (TN)*,
- *False Positive (FP)*,
- *False Negative (FN)*.

In the context of this study, a voxel is considered a *TP* if it intersects with an obstacle and is correctly identified by the mapping method as *occupied*. On the contrary, a voxel is considered a *FP* if it is marked as *occupied* but does not intersect with any obstacle. A similar principle holds for the identification of free voxels, where a voxel is classified as a *TN* only if it is marked as *free* and does not intersect with any obstacle. This accuracy, named as E_{cls} can be expressed as a ratio of the sum of falsely classified voxels and the total amount of free and occupied voxels in the map shown in the following equation:

$$E_{cls} = \frac{|FN| + |FP|}{|\mathcal{O}| + |\mathcal{F}|}, \tag{2.16}$$

where $|FN|$ is the cardinality of the set of all the voxels classifed as *FN*, $|FP|$ is the cardinality of the set of all the voxels classifed as *FP* and $|\mathcal{F}|$ is the cardinality of the set of all voxels marked as *free* by the mapping method. The correct classification of voxels as *True Positive*,

True Negative, *False Positive*, or *False Negative* is crucial in evaluating the performance of the mapping methods, as it gives an overall idea of the method's voxel classification accuracy.

C. *Excessive Mapped Volume Metric*

The last comparison metric expresses the ratio between the overall volume of all voxels marked as occupied and the sum of volumes of truly occupied portions of each voxel, as defined in eq. (2.14). Let us name the volume of all voxels marked as occupied as V_{occ}. This volume is obtained as the sum of volumes of all voxels in the map marked as occupied:

$$V_{occ} = \sum_{v \in \mathcal{O}} v_h^3. \tag{2.17}$$

With this sum, the excessive volume ratio E_{exc} can be expressed as

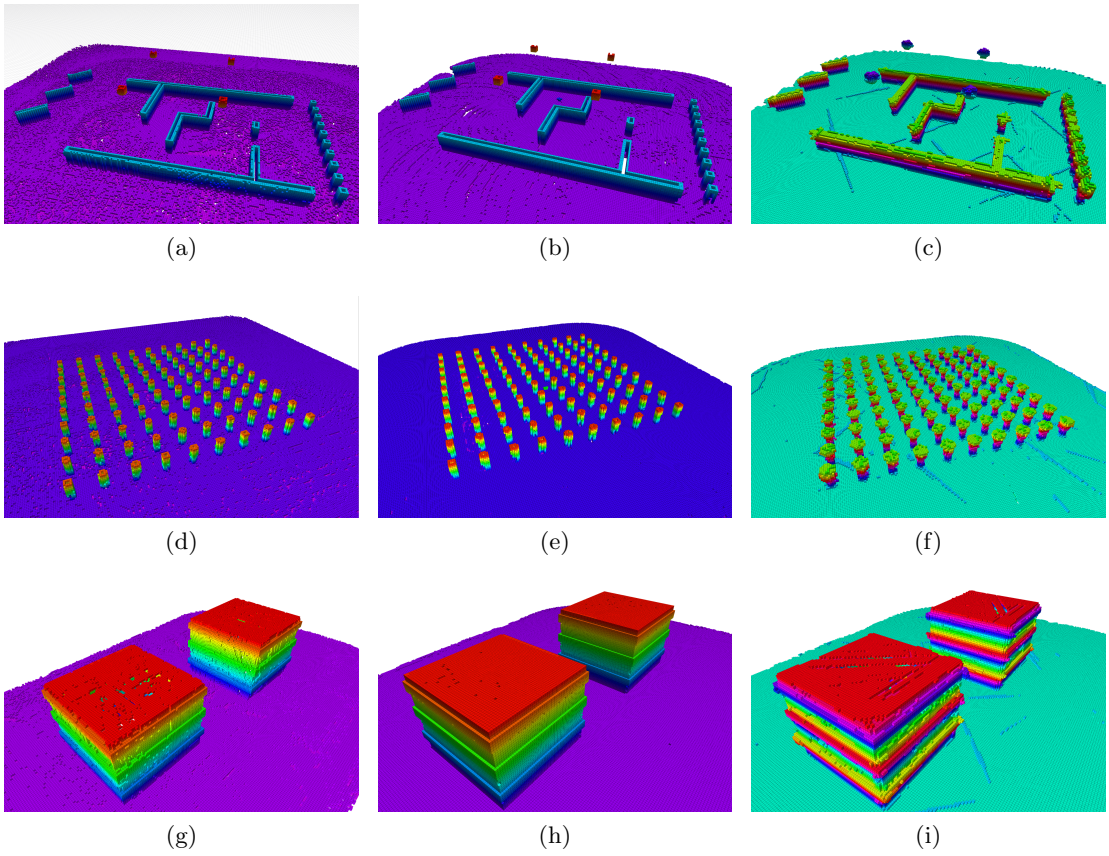
$$E_{exc} = \frac{V_{occ} - V_{int}}{V_{occ}}. \tag{2.18}$$

In other words, this ratio indicates the fraction of an overall volume of falsely occupied portion of each voxel, i.e. the sum of outer voxel parts of the intersections with any obstacle or ground, and the volume of all voxels classified as occupied by the mapping method.

Chapter 3

Results

The resulting occupancy maps of each method and in all three environments with ground truth poses can be seen in Figure 3.1. All maps look very similar, as expected. Note that the UFOMap was evaluated with the *Fast discrete integrator*, as if the UFOMap and OctoMap used the same integrators, the results should be more similar.



(a) (b) (c)

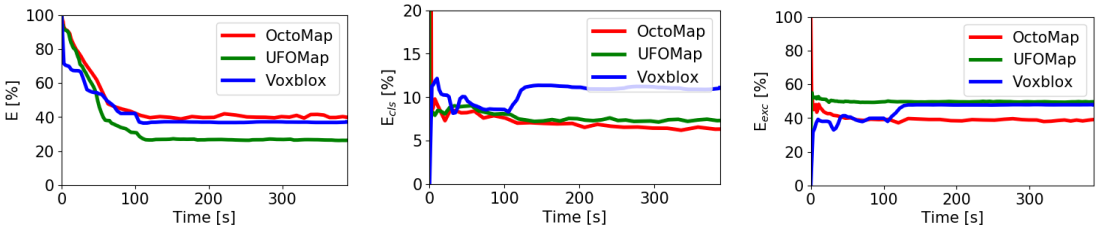
(d) (e) (f)

(g) (h) (i)

Figure 3.1: The resulting maps visualized in Rviz with ground truth (noiseless) UAV poses and pointclouds. The first row is the maze, second the forest, and the third is the urban environment. The first column is mapped by OctoMap, the second is UFOMap, and the third is Voxblox. The color scheme represents the elevation of each voxel.

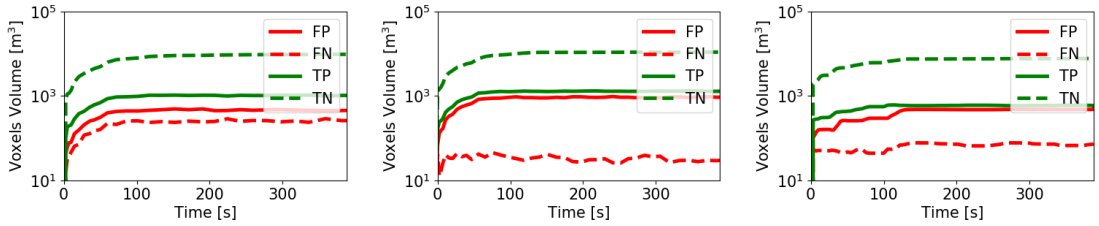
Let us first evaluate the proposed metrics with the ground truth data in the maze environment. OctoMap and Voxblox behaved very similarly in unmapped obstacle volume metric, having it around 40 percent, and UFOMap even lower. All methods converged right after the 1st iteration of the UAV's trajectory as can be seen in Figure 3.2. OctoMap and

UFOMap did slightly better than Voxblox in the voxel classification. Note that it is not possible to achieve a value of $E = 0$, as the voxels can cover only the surfaces of the obstacles, therefore an indicator of the method's robustness to the noise could also be the relative deviation of E with the noisy data from E with ground truth data. The non zero error in E_{cls} is caused by the grid alignment, mainly arising from the alignment of the voxel grid with the ground at $z = 0$, where are also the voxel boundaries. The error in E_{exc} is simply caused by the voxel resolution, false occupied voxels, and again by the grid alignment. The progress of the voxel volume classification of individual methods can be seen in Figure 3.3, where the UFOMap had a greater number of incorrectly classified occupied space, which relates to the UFOMap's result in unmapped obstacle volume and excessive volume metric, as it overall used more occupied voxels to map the environment.



(a) Unmapped obstacle vol. metric. (b) Voxel classification metric. (c) Excessive mapped vol. metric.

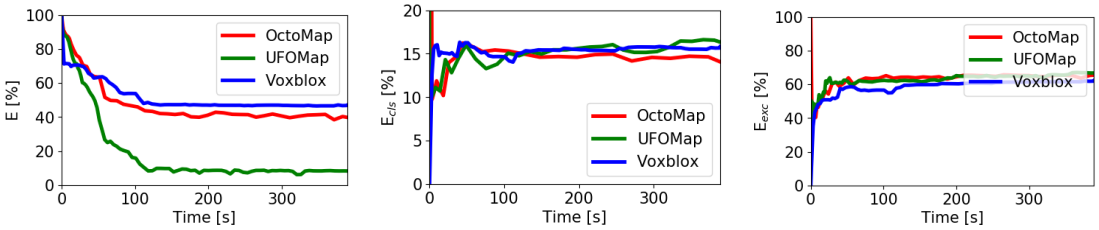
Figure 3.2: The evaluated metrics in the maze environment with ground truth data.



(a) OctoMap (b) UFOMap (c) Voxblox

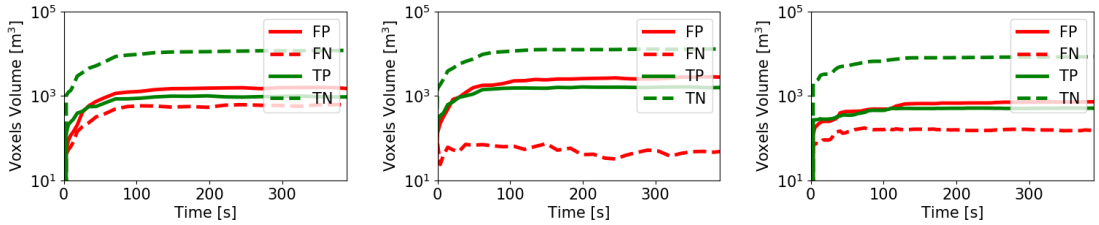
Figure 3.3: The voxel volume classification of each method over the course of mapping in the maze environment with ground truth data, in logarithmic scale.

Next, we present the results when using the parameters corresponding to the SLAM setup, as defined in Table 2.1, in the maze environment. The first noticeable thing in Figure 3.4 is that UFOMap error of the unmapped obstacle volume metric is much beneath the other two methods and even below the result with ground truth data. This could be caused by the UFOMap's more efficient algorithm of integrating new pointclouds, and as it integrates more points, more points can be incorrectly classified, or correctly classified as occupied inside the obstacles. This also correlates with the results in Figure 3.5, where the detailed progress of voxel volume classification of each method can be seen, in logarithmic scale. UFOMap has a greater amount of true and false positive voxels, which explains the results of the voxel classification metric, where the UFOMap performed slightly worse than the other two methods.



(a) Unmapped obstacle vol. metric. (b) Voxel classification metric. (c) Excessive mapped vol. metric.

Figure 3.4: The evaluated metrics in the maze environment with noisy data, modified with the parameters of SLAM.



(a) OctoMap (b) UFOMap (c) Voxblox

Figure 3.5: The voxel volume classification of each method over the course of mapping in the maze environment with noisy data, modified with the parameters of SLAM, in logarithmic scale.

All the remaining results are summarized in the following tables, showing the final value after the third iteration of the UAV's trajectory. Table 3.1 shows the results for all methods when using ground truth data, and can be seen as a reference to evaluate the method's robustness to the noise relative to the other experiments. The best result in each metric and in each environment is colored in green, the worst result is colored in red.

Ground Truth Data

	Maze			Forest			Urban		
	E	E_{cls}	E_{exc}	E	E_{cls}	E_{exc}	E	E_{cls}	E_{exc}
OctoMap	39.9	6.3	39.0	28.7	8.2	47.8	74.8	2.6	32.3
UFOMap	26.4	7.3	49.6	19.3	10.1	57.8	67.1	4.9	38.4
Voxblox	37.2	11.1	47.9	15.2	12.4	60.1	24.8	7.8	28.8

Table 3.1: The values of the evaluated metrics in all environments after the third iteration using ground truth data.

The results for the SLAM and SLAM with noisy LiDAR data shown in Tables 3.2 and 3.3 are nearly identical, as noise parameter for LiDAR σ_l increased only by 0.1m, which is effectively mitigated by the voxel resolution of 0.3m used in all experiments.

SLAM Data

	Maze			Forest			Urban		
	E	E_{cls}	E_{exc}	E	E_{cls}	E_{exc}	E	E_{cls}	E_{exc}
OctoMap	40.3	13.7	65.5	33.2	15.2	69.1	60.0	5.1	29.9
UFOMap	8.2	16.1	66.6	9.6	17.8	75.0	36.9	7.4	35.7
Voxblox	46.9	15.8	62.2	48.5	19.0	77.9	24.9	10.8	32.9

Table 3.2: The values of the evaluated metrics in all environments after the third iteration using noisy data, modified with the parameters of SLAM.

SLAM with noisy LiDAR Data

	Maze			Forest			Urban		
	E	E_{cls}	E_{exc}	E	E_{cls}	E_{exc}	E	E_{cls}	E_{exc}
OctoMap	39.7	14.1	66.4	34.5	15.1	68.8	60.4	5.1	31.6
UFOMap	6.0	15.6	68.0	10.3	17.9	76.7	36.2	6.2	39.0
Voxblox	47.7	16.3	62.9	46.3	18.8	76.9	25.2	12.9	38.4

Table 3.3: The values of the evaluated metrics in all environments after the third iteration using noisy data, modified with the parameters of SLAM and noisy LiDAR.

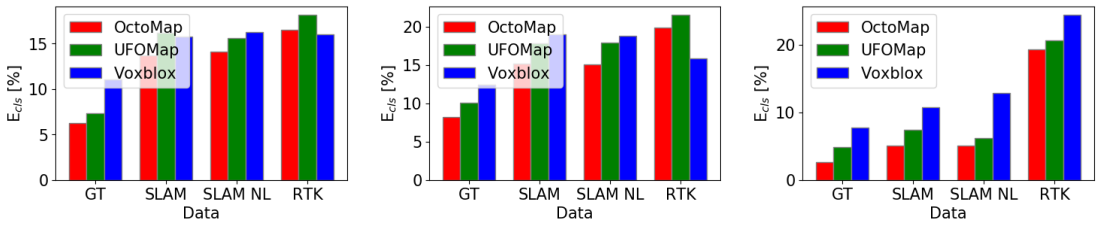
Using noise parameters of RTK GPS, as shown in Table 3.4 and Figure 3.7, further degrades the ability of each method to accurately map the environment. OctoMap has results almost comparable with Voxblox, performing slightly better or worse in each metric. UFOMap achieved the best result in unmapped obstacle volume metric in all three environments, at the cost of the higher excess volume error. Figure 3.6 compares the achieved classification error across all the environments and data. We can notice that the OctoMap performed the best almost in all cases.

RTK Data

	Maze			Forest			Urban		
	E	E_{cls}	E_{exc}	E	E_{cls}	E_{exc}	E	E_{cls}	E_{exc}
OctoMap	62.9	16.5	88.7	77.4	19.9	93.0	63.0	19.3	60.2
UFOMap	11.4	18.2	89.5	19.1	21.6	92.9	6.1	20.7	66.8
Voxblox	66.7	16.0	78.9	91.3	15.9	96.3	60.6	24.5	69.9

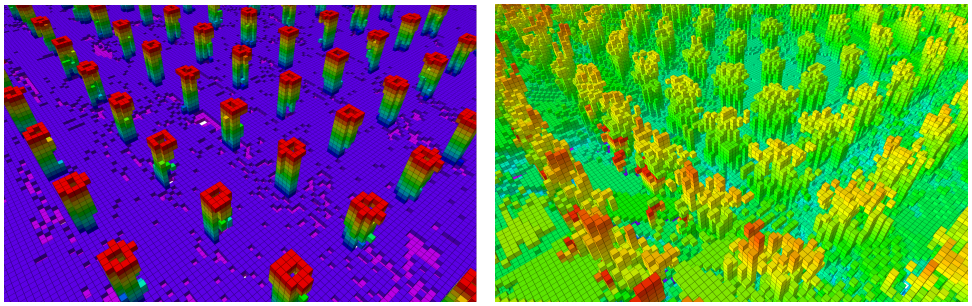
Table 3.4: The values of the evaluated metrics in all environments after the third iteration using noisy data, modified with the parameters of RTK.

The results when using RTK with a noisy magnetometer or GPS are further aggravated and do not provide any additional information to our experiments, thus they are neglected. The overall best results were achieved by OctoMap, being the best in the voxel classification accuracy, and also having the lowest excessive volume in the most instances. Even though UFOMap did not surpass OctoMap in terms of the voxel classification, it had the lowest error in classifying free voxels. UFOMap also exhibited the best values in unmapped obstacle volume metric, except in a few experiments, where Voxblox performed better. Voxblox, on the other hand, showcased the worst results in the voxel classification metric, performing slightly better only when using the RTK data, probably due to the different method of updating the voxel occupancy. It also had the highest number of results labeled as the worst.



(a) Maze (b) Forest (c) Urban

Figure 3.6: The voxel volume classification of each method after the third iteration in all environments and all the types of data, starting with ground truth (GT).



(a) Ground truth data. (b) RTK data.

Figure 3.7: The resulting occupancy maps of OctoMap visualized in Rviz, showing the map when using RTK data in comparison to the map with ground truth data.

Chapter 4

Accuracy improvement

The mapping methods evaluated in this thesis, OctoMap, UFOMap, and Voxblox, using the three established metrics, showed significantly reduced map quality with increased pose uncertainty, as expected. We propose three improvements that can mitigate the effects of the pose uncertainty on the results. The first improvement is to vary the voxel resolution. By using different voxel resolutions, we can explore how the resolution affects the accuracy of the map in the presence of pose uncertainty. The second improvement we propose is to vary the mapping range, which refers to the maximum distance at which the mapping algorithm is allowed to generate a map. Lastly, we propose changing the sensor model used for mapping, which can significantly impact the quality of the resulting map. By altering the sensor model, we can reduce or increase the confidence in the sensor measurement, which is important as the measurements become more inaccurate. Each improvement will be tested with identical data individually, while observing their effectiveness, and then all the improvements together within a single test. These experiments use a denormalized classification error $\widehat{E}_{cls}$ for comparing the results, previously defined in eq. (2.16), i.e. the equation is not divided by the total number of free and occupied voxels, and remains as the sum of falsely classified voxels as

$$\widehat{E}_{cls} = |FN| + |FP|. \tag{4.1}$$

The denormalized metric better indicates the accuracy of each method when using different distances and resolutions, as there is much more free voxels when using greater distances or finer resolutions.

4.1 Mapping distance

The idea of adjusting the mapping distance comes from the fact that the error in the orientation of the UAV results in a more significant error in position of a measured endpoint that is further from the UAV, as shown in Figure 4.1. So theoretically, for a given orientation error ϕ, the deviation d is linearly dependent on the distance of the endpoint, i.e., the endpoint twice as further from the source as the other endpoint should have the deviation twice as large, etc.

By decreasing the mapping distance, the effect that the orientation error has on the resulting map is reduced. However, the mapping of the environment can be prolonged, because the trajectory of the UAV that is mapping the environment should be adjusted to cover the whole area with the newly set distance. In our experiments, we use three different distances – 30 meters, 10 meters, and 5 meters. We observe the effect that it has on the resulting map of the forest environment when the ground truth data are transformed using the vector of parameters $\boldsymbol{\sigma}$, as previously defined in equation (2.7), with the following values:

$$\boldsymbol{\sigma} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.05 & 0.05 & 0.05 \end{bmatrix}^\intercal, \tag{4.2}$$

to demonstrate how the shortening of the mapping distance mitigates the error in the UAV orientation.

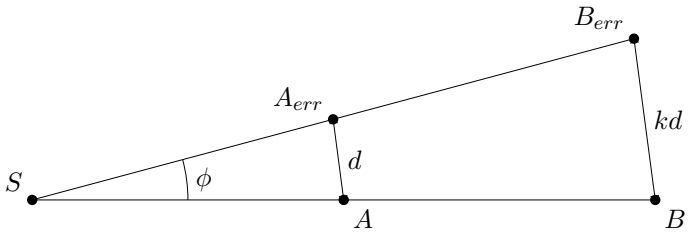


Figure 4.1: Example of deviations from the original points caused by noise in orientation. The error is denoted as ϕ, S is the source of the laser beam, point A is the first endpoint and point B is the second endpoint further from the source. A_{err} and B_{err} are the points deviated from the original points A and B by the error ϕ. Resulting deviation distance from point A is denoted as d, whereby the deviation for point B can be calculated by multiplying d by a coefficient k, which is a ratio of the distance $|SB|$ and $|SA|$.

4.2 Voxel resolution

The errors caused by a high pose uncertainty can be amplified when using fine voxel resolutions corresponding to smaller voxels. This is because the mapping method may classify a free volume in the environment as occupied and being unable to distinguish it due to the high resolution and sensor limitation, as there is a smaller chance for the endpoints of the measurements to be sampled into the same volume multiple times. As a result, the resulting map may contain artifacts and inaccuracies that do not reflect the true environment.

Using larger voxels can help mitigate the effects of pose uncertainty on the resulting map. The environment is then represented at a coarser level of detail, which makes the mapping less sensitive to deviations caused by the pose uncertainty. However, this comes at the cost of decreased map resolution and losing information about the environment. We test 3 different resolutions – 0.50m, 0.30m and 0.15m in the forest environment with the data transformed by the SLAM with noisy LiDAR parameters as defined in Table 2.1.

4.3 Sensor model and occupancy thresholds

The sensor model used in the previous experiments causes a high confidence in voxels being occupied once classified so, therefore a more suitable sensor model should be used with increased pose uncertainty. As defined in Section 1.2.1, a voxel is classified as occupied, if the log-odd value is above a certain probability threshold. The threshold for classifying an occupied voxel was set to $t_{occ} = 0.5$ in the previous experiments. In this experiment, we test three different settings of the sensor model along with the occupancy threshold. However, this last experiment involves only OctoMap and UFOMap, as Voxblox does not offer a simple way of adjusting the sensor model and does not use a probabilistic model for integrating the sensor measurements as OctoMap and UFOMap do. Three combinations of occupancy probability thresholds and sensor model log-odd values are tested in the forest environment, the first being the same as in the original experiments:

(1) $t_{occ} = 0.5$, $l_{occ} = 2.95$,
(2) $t_{occ} = 0.6$, $l_{occ} = 1.73$,
(3) $t_{occ} = 0.7$, $l_{occ} = 1.1$,

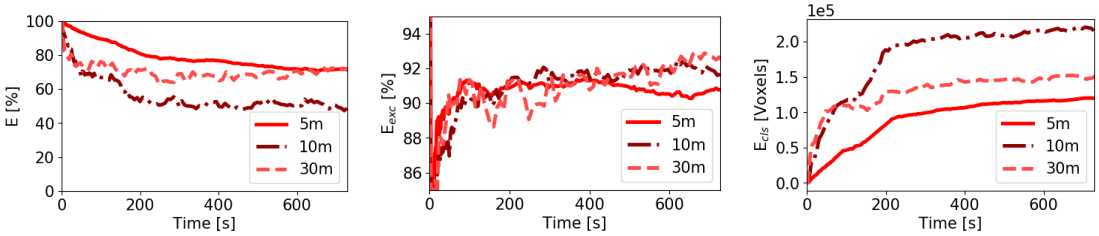
where the log-odd values of l_{occ} correspond to the probabilities of 0.95, 0.85 and 0.75, respectively. The sensor model for a beam traversing the voxel is set to $l_{free} = -0.2$ for all combinations and the data are transformed again by the SLAM with noisy LiDAR parameters.

4.4 Results

The results of the improvements are divided into four sections. The first three sections analyze the effects of the discussed improvements, voxel resolution, mapping distance and the sensor model, individually on each method (the sensor model applies only to OctoMap and UFOMap). The last section focuses on the resulting accuracy of each method when it uses all the proposed improvements simultaneously and then compares the methods between each other, although Voxblox is at a slight disadvantage because the adjusted sensor model and the occupancy thresholds of OctoMap and UFOMap cannot be applied to Voxblox.

4.4.1 Mapping distance

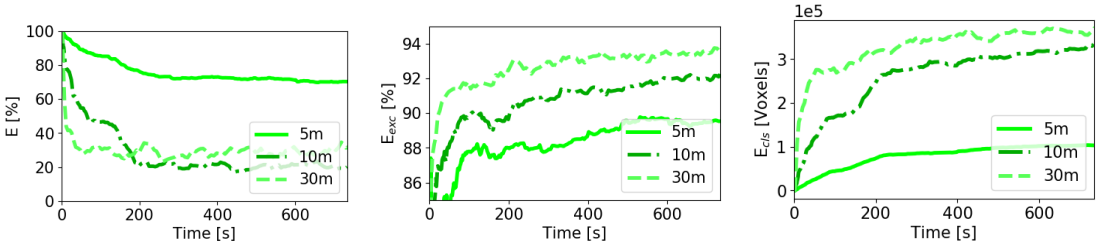
The first proposed improvement was to shorten the mapping distance, as it should reduce the accuracy error caused by the noise in the orientation of the UAV. In Figures 4.2, 4.3 and 4.4, we demonstrate the results of the evaluated metrics with the decreasing mapping distance. The unmapped obstacle volume metric for all the methods converged faster when using greater distance, as expected. However, using the shorter distance of 5 meters proved significant improvement in voxel classification both in OctoMap and UFOMap.



(a) Unmapped obstacle vol. metric. (b) Excessive mapped vol. metric. (c) Voxel classification metric.

Figure 4.2: The evaluated metrics of OctoMap in the forest environment and mapping distance variations with the noise only in UAV orientation.

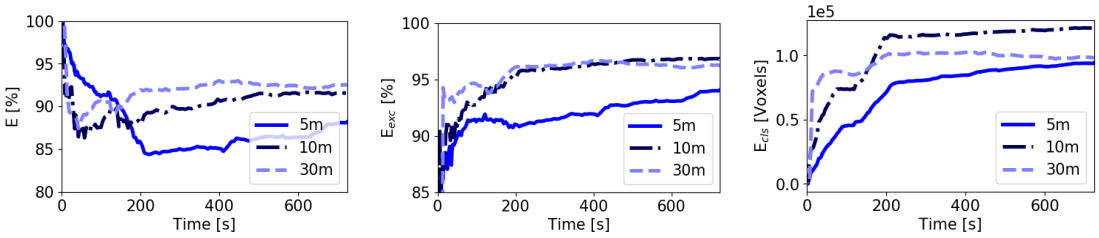
The reason why the results were the most erroneous in the voxel classification when using 10 meters for OctoMap and Voxblox might be the used trajectory when traversing the forest environment with the UAV, as the mapping distance of 10 meters covers also the adjacent tree lines. This might result in a faulty classification of the distant voxels, and having less repeated coverage of the same areas in contrast to the mapping distance of 30 meters, which covers almost the whole environment at any time throughout the test, thus has an extended duration for reclassifying any voxel. Therefore selecting an ideal trajectory also plays a crucial role when mapping the environment, as it can significantly impact the results.



(a) Unmapped obstacle vol. metric. (b) Excessive mapped vol. metric. (c) Voxel classification metric.

Figure 4.3: The evaluated metrics of UFOMap in the forest environment and mapping distance variations with the noise only in UAV orientation.

The results for Voxblox in Figures 4.4a and 4.4b were scaled to better demonstrate the effects of the mapping distance, as the shorter distance provided only a very light improvement.

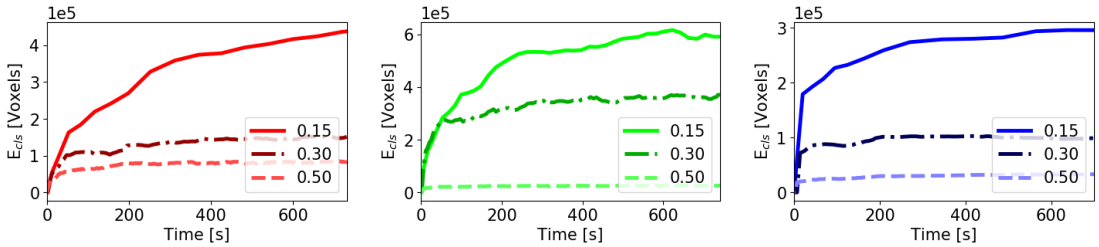


(a) Unmapped obstacle vol. metric. (b) Excessive mapped vol. metric. (c) Voxel classification metric.

Figure 4.4: The evaluated metrics of Voxblox in the forest environment and mapping distance variations with the noise only in UAV orientation.

4.4.2 Voxel resolution

The next proposed improvement was to change the voxel resolution, as it would map the environment in coarser details, however with better accuracy in the voxel classification. The results comparing the use of different resolutions and its effect on the classification accuracy in the denormalized $\widehat{E}_{cls}$ metric are shown in Figure 4.5. The results for the other metrics are excluded, as it is not fair to compare them to each other in different resolutions, since the larger voxel sizes cover more of the obstacles volume, giving us no indication of the method's performance in comparison to smaller sizes. The results show that the number of incorrectly classified voxels is decreased when using lower resolution in all of the methods. That holds even volume-wise, meaning that the incorrectly classified volume (number of incorrectly classified voxels multiplied by the voxel resolution is reduced).

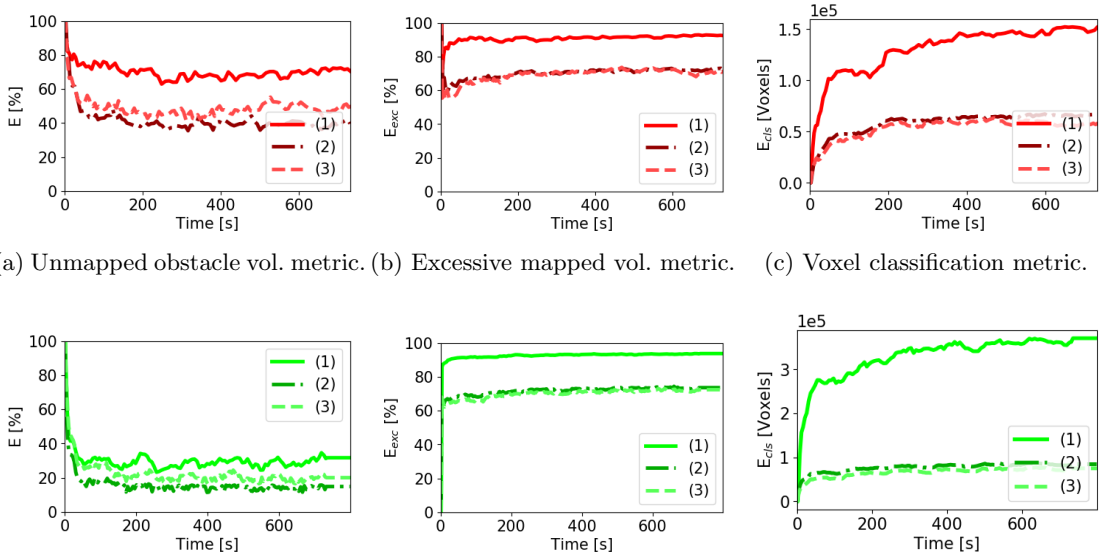


(a) OctoMap. (b) UFOMap. (c) Voxblox.

Figure 4.5: The evaluated voxel classification metric in the forest environment of each method using SLAM with noisy LiDAR data and voxel resolution variations.

4.4.3 Sensor model and occupancy thresholds

The last proposed improvement is varying the sensor model of the OctoMap and UFOMap. The results are shown in Figure 4.6, with the legend referring to the combinations listed in 4.3. Although using the sensor model with the lower log-odd values for the occupied nodes and the higher occupancy probability thresholds proved advantageous over the more confident sensor model, combination (2) showed the best results in unmapped obstacle volume metric E and almost the same results as (3) in the other two metrics, both in OctoMap and UFOMap. This is because (3) had a very strict occupancy threshold and an overly uncertain sensor model in the measurements, so it provided no further improvement to the resulting map and even started to lose information about the obstacles.



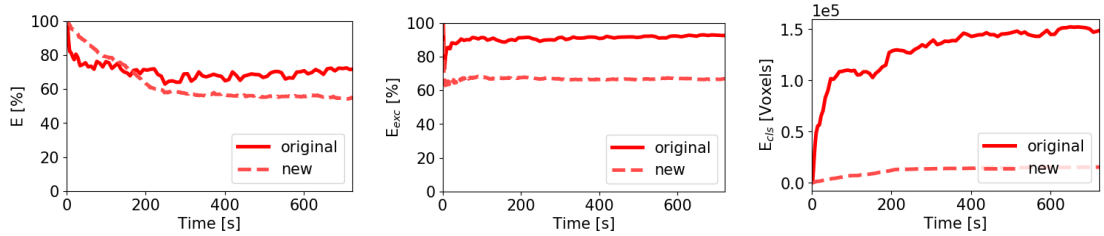
(a) Unmapped obstacle vol. metric. (b) Excessive mapped vol. metric. (c) Voxel classification metric.

(d) Unmapped obstacle vol. metric. (e) Excessive mapped vol. metric. (f) Voxel classification metric.

Figure 4.6: The evaluated metrics of OctoMap (red) and UFOMap (green) in the forest environment using SLAM with noisy LiDAR data and sensor model variations.

4.4.4 All improvements simultaneously

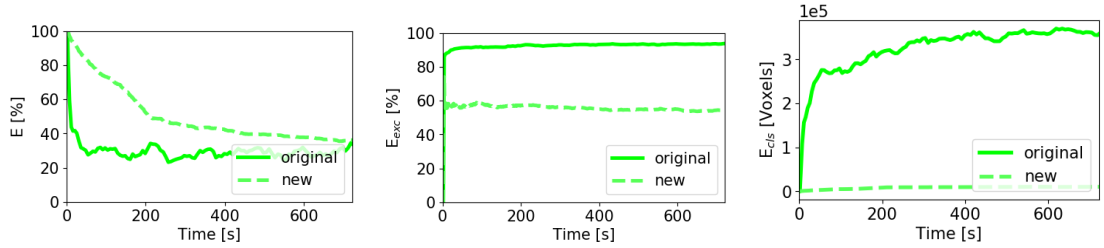
Figure 4.7 shows results where all of the proposed improvements are used: the mapping distance of 5m, the voxel resolution of 0.5m and the sensor model (3), in a single experiment for each method, except the adjusting of the sensor model for Voxblox. We compare these results to the results from the previous experiments of using the RTK data in Chapter 3 to demonstrate its effects. OctoMap and UFOMap showed great improvement in all of the metrics, however, the improvements had little to almost no effect on Voxblox, as the previous experiments indicated, and only enhanced the voxel classification accuracy, thanks to the lower resolution.



(a) Unmapped obstacle vol. metric.

(b) Excessive mapped vol. metric.

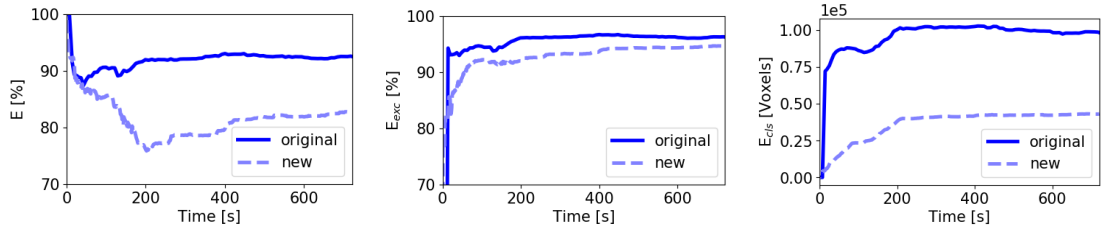
(c) Voxel classification metric.



(d) Unmapped obstacle vol. metric.

(e) Excessive mapped vol. metric.

(f) Voxel classification metric.

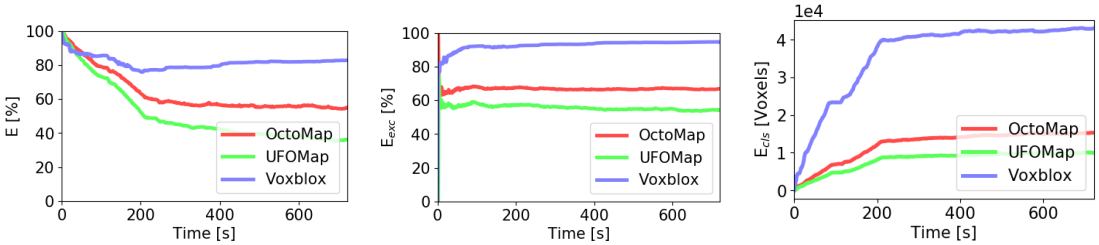


(g) Unmapped obstacle vol. metric.

(h) Excessive mapped vol. metric.

(i) Voxel classification metric.

Figure 4.7: The evaluated metrics for OctoMap (red), UFOMap (green) and Voxblox (blue and scaled to better demonstrate the improvement) using all of the proposed improvements except the sensor model for Voxblox in comparison to the original experiments.



(a) Unmapped obstacle vol. metric. (b) Excessive mapped vol. metric. (c) Voxel classification metric.

Figure 4.8: The improved results in comparison between each other.

The last Figure 4.8, compares the methods with all of the improvements between each other. Voxblox showcased poor results with uncertain pose in comparison to the other methods, as the improvements had little to no effect on the resulting map.

Chapter 5

Conclusion

This thesis aimed to compare the mapping accuracy and robustness of three widely used 3D mapping frameworks, namely OctoMap, UFOMap (with *Fast discrete integrator*), and Voxblox. The comparison was performed in three distinct simulated environments with an increasing pose uncertainty of the autonomous UAV equipped with 3D LiDAR. Three different metrics were used to assess their performance. When subjected to the increasing pose uncertainty, the methods proved sensitive to high error in the orientation of the UAV, especially at longer mapping distances, and became practically unusable when combined with a significant positional error. Three different improvements to mitigate this were proposed.

Throughout the experiments, it was observed that OctoMap and UFOMap consistently outperformed Voxblox in voxel classification accuracy, whereas UFOMap also exhibited the best results in mapping most of the obstacle volumes at the cost of the higher excessive volume error and slightly lower classification accuracy. This behaviour can be attributed to UFOMap's more efficient algorithm, enabling it to integrate a larger number of points into the map within a shorter timeframe. Consequently, it can also integrate more noisy points and incorrectly classify more voxels, especially when using longer mapping distances and a confident sensor model. As a result of this, UFOMap showcased a significant positive response to the proposed improvements, having superior results in comparison to the other methods. Voxblox, on the other hand, performed consistently well throughout the experiments, except for the use of the proposed improvements, which had little to no effect. Although it did not surpass OctoMap and UFOMap in terms of classification accuracy in most of the cases, it served as a good middle ground between OctoMap and UFOmap in several instances.

In summary, OctoMap emerged as the top-performing mapping framework in this study when none of the improvements were utilized, showcasing superior mapping accuracy compared to UFOMap and Voxblox. However, UFOMap achieved the best results in mapping of the obstacle volume, and exhibited potential for enhancement by responding the most positively to the proposed improvements. Each of these methods presented their own advantages and disadvantages. OctoMap proved to be the most reliable mapping algorithm, as it achieved the best classification results in most scenarios, while UFOMap performed the best when the improvements were applied and had the overall lowest error of incorrectly classified free space. This makes UFOMap an ideal choice in scenarios where collision-free operations are crucial. Although Voxblox did not surpass OctoMap and UFOMap in mapping accuracy, it performed almost equally well, and is a suitable choice for path planning algorithms, as it already provides ESDF layers as a resulting map.

Future works might focus on optimizing the proposed improvements for achieving the best possible results when mapping under the effects of the pose uncertainty, for example by minimizing the sum of the proposed metrics. The optimization of the improvements should also be done in real time, as the pose uncertainty of the UAV can change over time, due to unpredictable environmental effects or sensor malfunctions.

Chapter 6

References

[1] M. Vrba, V. Walter, and M. Saska, "On onboard LiDAR-based flying object detection," 2023, preprint. arXiv: 2303.05404 [cs.RO].

[2] D. Matthes and V. Drakopoulos, "Line clipping in 2D: Overview, techniques and algorithms," *Journal of Imaging*, vol. 8, 2022.

[3] Q. Zou, Q. Sun, L. Chen, B. Nie, and Q. Li, "A comparative analysis of LiDAR SLAM-based indoor navigation for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6907–6921, 2022.

[4] 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, 1 May 2021.

[5] A. Merzlyakov and S. Macenski, "A comparison of modern general-purpose visual SLAM approaches," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 9190–9197.

[6] T. Báca, 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," *CoRR*, vol. abs/2011.03356, 2020.

[7] D. Duberg and P. Jensfelt, "UFOMap: An Efficient Probabilistic 3D Mapping Framework That Embraces the Unknown," *IEEE Robotics and Automation Letters*, 2020.

[8] D. Ekaso, F. Nex, and N. Kerle, "Accuracy assessment of real-time kinematics (RTK) measurements on unmanned aerial vehicles (UAV) for direct geo-referencing," *Geo-spatial Information Science*, vol. 23, 2020.

[9] H. Wu, X. Pei, J. Li, H. Gao, and Y. Bai, "An improved magnetometer calibration and compensation method based on levenberg–marquardt algorithm for multi-rotor unmanned aerial vehicle," *Measurement and Control*, vol. 53, no. 3-4, pp. 276–286, 2020.

[10] Y.-C. Lin, Y.-T. Cheng, T. Zhou, R. Ravi, S. M. Hasheminasab, J. E. Flatt, C. Troy, and A. Habib, "Evaluation of UAV LiDAR for Mapping Coastal Environments," *Remote Sensing*, vol. 11, no. 24, 2019.

[11] Y. Chen, J. Tang, C. Jiang, L. Zhu, M. Lehtomäki, H. Kaartinen, R. Kaijaluoto, Y. Wang, J. Hyyppä, H. Hyyppä, H. Zhou, L. Pei, and R. Chen, "The Accuracy Comparison of Three Simultaneous Localization and Mapping (SLAM)-Based Indoor Mapping Technologies," *Sensors*, vol. 18, no. 10, 2018.

[12] F. He, T. Zhou, W. Xiong, S. M. Hasheminnasab, and A. Habib, "Automated aerial triangulation for uav-based mapping," *Remote Sensing*, vol. 10, no. 12, 2018.

[13] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017.

[14] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "UAV for 3D mapping applications: a review," *Applied Geomatics*, 1 Mar. 2014.

[15] G. Vallicrosa, A. Palomer, D. Ribas, and P. Ridao, "Realtime AUV terrain based navigation with OctoMap in a natural environment," in *ROBOT2013: First Iberian Robotics Conference: Advances in Robotics, Vol. 1*, M. A. Armada, A. Sanfeliu, and M. Ferre, Eds. Springer International Publishing, 2014, pp. 41–53.

[16] S. Vanneste, B. Bellekens, and M. Weyn, "3DVFH+: Real-time three-dimensional obstacle avoidance using an OctoMap," in *MORSE@STAF*, 2014.

[17] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013.

[18] R. L. Klaser, F. S. Osório, and D. F. Wolf, "Simulation of an autonomous vehicle with a vision-based navigation system in unstructured terrains using OctoMap," in *2013 III Brazilian Symposium on Computing Systems Engineering*, 2013, pp. 177–178.

[19] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardös, "A comparison of SLAM algorithms based on a graph of relations," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 2089–2095.

[20] F. Remondino and F. Nex, "On measuring the accuracy of SLAM algorithms," *Autonomous Robots*, 1 Nov. 2009.

[21] D. Cole and P. Newman, "Using laser range data for 3d slam in outdoor environments," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 1556–1563.

[22] A. Nuchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6d slam - 3d mapping outdoor environments," *Fraunhofer IAIS*, vol. 24, Nov. 2006.

[23] G. M. S. W. L. R. A. A. P., "Global Positioning System: Theory and Applications, The systems programming series," in. John Wiley and Sons., 2001, vol. 1, ch. 11.

[24] M. Caruso, "Applications of magnetic sensors for low cost compass systems," 2000.

[25] J. D. A. v. D. J. F. H. S. K. F. Foley, *Computer Graphics: Principles and Practice, The Systems Programming Series.* Addison-Wesley, 1990.

[26] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," *Proceedings of EuroGraphics*, vol. 87, Aug. 1987.

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

[28] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129–147, 1982.

Chapter A

Attachments

The folder `create_world` contains python script that generates a Gazebo simulation world with objects defined in `xml` files. The folder `eval_accuracy` is a ROS node evaluating accuracy of a running method, subscribing to topics containing whole maps of the environments. The folder `noise_node` is a ROS node transforming a rosbag according to parameters of the multivariate Gaussian, defined in `params.yaml`. The python script `plot_err.py` was used to generate graphs of the evaluated metrics. The file `README.md` contains a short description of each script and how to use them.

```
├── create_world
│   ├── end.xml
│   ├── main.py
│   ├── start.xml
│   ├── unit_box.xml
│   └── unit_cyl.xml
├── eval_accuracy
│   ├── src
│   │   └── eval_accuracy.cpp
│   ├── CMakeLists.txt
│   └── package.xml
├── noise_node
│   ├── config
│   │   └── params.yaml
│   ├── launch
│   │   └── noise.launch
│   ├── scripts
│   │   ├── noise
│   │   ├── noise_offline
│   │   └── utils.py
│   ├── CMakeLists.txt
│   └── package.xml
├── README.md
└── plot_err.py
```