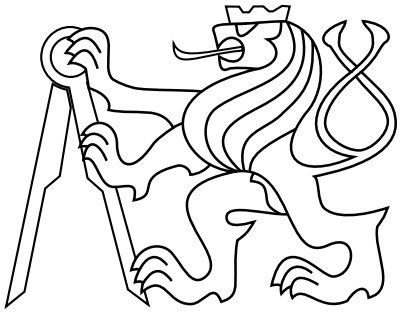
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



Development of a Safe Flocking Algorithm for UAVs Using 3D Lidar and Collaborative Multi-Robot Coordination

Bachelor's Thesis

Michal Kamler

Prague, May 2025

Study programme: Electrical Engineering and Information Technology
Branch of study: Cybernetics and Robotics

Supervisor: MSc. Manuel Boldrer, Ph.D.

Acknowledgments

Firstly, I would like to express my gratitude to my supervisor.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kamler Michal** Personal ID number: **516070**

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:

Development of a Safe Flocking Algorithm for UAVs Using 3D Lidar and Collaborative Multi-Robot Coordination

Bachelor’s thesis title in Czech:

Vývoj bezpečného algoritmu pro koordinaci UAV pomocí 3D lidaru a koordinace více robotů

Guidelines:

(1) Develop a safe flocking algorithm for UAVs in C++ within the MRS system framework . The algorithm must ensure collision-free and efficient movement of UAV.
(2) Compare your solution to some of the most promising algorithms in the literature within the MRS simulator, in particular [1],[2],[3].
(3) The novel contributions of the thesis will include
i) Extend existing multi-robot algorithm from 2d to 3d.
ii) Use 3d lidar to localize, sense, process, and react to environments such as a forest.
iii) Research on possible enhancements of the algorithm for example using neural networks or learning-based techniques.
(4) Conduct an experimental campaign to validate the theoretical findings. Include different real-world scenarios, such as navigating through forests or crowded spaces.

Bibliography / sources:

[1] Boldrer, M., Serra-Gomez, A., Lyons, L., Alonso-Mora, J., & Ferranti, L. (2024). Rule-Based Lloyd Algorithm for Multi-Robot Motion Planning and Control with Safety and Convergence Guarantees. arXiv preprint arXiv:2310.19511v2 (2023).
[2] Mezey, D., Bastien, R., Zheng, Y., McKee, N., Stoll, D., Hamann, H., & Romanczuk, P. (2024). Purely vision-based collective movement of robots. arXiv preprint arXiv:2406.17106.
[3] Ahmad, A., Licea, D. B., Silano, G., Báča, T., & Saska, M. (2022). PACNav: a collective navigation approach for UAV swarms deprived of communication and external localization. Bioinspiration & Biomimetics, 17(6), 066019

Name and workplace of bachelor’s thesis supervisor:

Manuel Boldrer, Ph.D. Multi-robot Systems FEE

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

Date of bachelor’s thesis assignment: **30.01.2025** Deadline for bachelor thesis submission: _____

Assignment valid until: **20.09.2026**

_____
prof. Dr. Ing. Jan Kybic
Head of department’s signature

_____
prof. Mgr. Petr Páta, Ph.D.
Vice-dean´s signature on behalf of the Dean

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

Declaration

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

Date ....

Abstract

The study of autonomous Unmanned Aerial Vehicles (UAVs) has become a prominent sub-field of mobile robotics.

Keywords TODOUnmanned Aerial Vehicles, Automatic Control

Abbreviations

FOV Field of View

GNSS Global Navigation Satellite System

GPS Global Positioning System

IMU Inertial Measurement Unit

LiDAR Light Detection and Ranging

MRS Multi-robot Systems Group

RTK Real-time Kinematics

SLAM Simultaneous Localization And Mapping

UAV Unmanned Aerial Vehicle

UGV Unmanned Ground Vehicle

RBL Rule-based Lloyd Algorithm

ToF Time-of-Flight

PCL Point Cloud Library

GNSS Global Navigation Satellite System

3D three-dimensional

2D two-dimensional

EMI Electromagnetic Interference

IMU Inertial Measurement Unit

RL Reinforcement Learning

NN Neural Network

Contents

1 Introduction

TODO

1.1 Related Works

TODO

1.2 Problem Statement

1.3 Contributions

TODO

1.4 Mathematical Notation

TODO

$\mathcal{N}_i$	set of neighboring agents for the i-th agent
$\mathbf{x}$, $\boldsymbol{\alpha}$	vector, pseudo-vector, or tuple
$\hat{\mathbf{x}}$, $\hat{\boldsymbol{\omega}}$	unit vector or unit pseudo-vector
$\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3$	elements of the *standard basis*
$\mathbf{X}, \boldsymbol{\Omega}$	matrix
$\mathbf{I}$	identity matrix
$x = \mathbf{a}^\intercal \mathbf{b}$	inner product of $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^3$
$\mathbf{x} = \mathbf{a} \times \mathbf{b}$	cross product of $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^3$
$\mathbf{x} = \mathbf{a} \circ \mathbf{b}$	element-wise product of $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^3$
$\mathbf{x}_{(n)} = \mathbf{x}^\intercal \hat{\mathbf{e}}_n$	n^{th} vector element (row), $\mathbf{x}, \mathbf{e} \in \mathbb{R}^3$
$\mathbf{X}_{(a,b)}$	matrix element, (row, column)
x_d	x_d is *desired*, a reference
$\dot{x}, \ddot{x}, \dddot{x}, \ddddot{x}$	1^{st}, 2^{nd}, 3^{rd}, and 4^{th} time derivative of x
$x_{[n]}$	x at the sample n
$\mathbf{A}, \mathbf{B}, \mathbf{x}$	LTI system matrix, input matrix and input vector
SO(3)	3D special orthogonal group of rotations
SE(3)	*SO(3)* $\times \mathbb{R}^3$, special Euclidean group

Table 1.1: Mathematical notation, nomenclature and notable symbols.

2 Extension of the Rule-Based Lloyd Algorithm to 3D

2.1 Introduction

Motivation

Motivation in using Rule-based Lloyd Algorithm (RBL) algorithm offers communication-less approach for guiding multiple agents from point A to point B. It requires an estimate of its position—whether through Global Positioning System (GPS), Real-time Kinematics (RTK) or methods like Simultaneous Localization And Mapping (SLAM)—along with onboard sensors that provide information about the surrounding environment, enabling it to avoid obstacles and move toward the goal. So far, the algorithm has mainly been tested in two-dimensional (2D) scenarios with a fixed altitude, and the results have looked promising. This were the main motivation factors to try and extend it to full three-dimensional (3D) navigation, where altitude and sensor orientation also play a role.

Problem Statement

Many real-world robotic applications, particularly involving Unmanned Aerial Vehicle (UAV), require navigation within 3D environments containing obstacles. While algorithms like the RBL [4] have proven effective in 2D goal convergence, they are inherently limited when applied directly to 3D space. The core problem arises because the existing RBL algorithm is fundamentally designed for 2D planar navigation, making it unsuitable for applications like UAV operations that require navigating in 3D space. Specifically, there is a need to extend existing 2D algorithms to 3D while smartly incorporating vertical exploration to ensure safe, efficient, and effective navigation through crowded 3D environments without significantly compromising performance or requiring excessive computational resources. This work addresses the challenge of adapting the RBL algorithm for 3D space by developing and evaluating specific rules to control vertical movement, aiming to achieve robust 3D navigation capabilities.

Objectives

- Enhance the convergence speed of agents towards their goals by promoting exploration, with focus on the vertical exploration.
- Demonstrate the algorithm's enhanced applicability, achieved by extending goal placement to the z-axis.

Chapter Overview

The principles of the RBL algorithm, presented in [4], form the basis of the work in this chapter. These principles are clarified and modified to address the challenges of 3D extension. The chapter details the fundamental principles of RBL in 2D and 3D spaces, describes the specific rules applied to enhance convergence, and concludes with a series of simulation scenarios designed to demonstrate the performance of the proposed solution. Finally, the chapter concludes in the presentation and evaluation of simulation experiments designed to assess the

efficiency of the proposed 3D extension and the impact of the introduced vertical exploration rules.

2.2 Basic principles of Rule-Based Lloyd Algorithm

Overview

The algorithm is a communication-less approach designed to navigate agents from point A to point B. The algorithm's ability to determine each agent's position relative to its goal depends on the availability of positioning data. This can be either global positioning data, such as from GPS, or relative positioning data obtained through techniques like SLAM in GNSS-denied environments, combined with sensor inputs that provide information about the agent's surrounding environment. These sensors can include LiDAR, depth cameras, or standard cameras with estimation techniques, allowing the agent to detect and avoid obstacles or other agents. The algorithm enables autonomous navigation without requiring direct communication between agents, making it suitable for scalable and decentralized applications.

Applications and Limitations in 2D

Modern robotics relies on the capability to navigate from point A to point B. Navigation plays a crucial role in various robotic applications, such as Unmanned Ground Vehicle (UGV)s, which are commonly used in manufacturing and logistics. UGVs typically follow predefined 2D trajectories guided by visual [16], magnetic [19], or LiDAR-based navigation [18]. Additionally, 2D navigation is widely used in robotic vacuum cleaners, enabling them to systematically cover an area while avoiding obstacles.

An obvious limitation for algorithms in 2D is scalability. As the number of agents in a system increases, the complexity of managing their movements and coordination also grows significantly. Obstacle avoidance in 2D can also be less efficient compared to 3D environments, as agents have fewer options for evading obstacles. In 3D, agents can change their altitude in addition to their horizontal trajectory, giving them more freedom to maneuver around obstacles.

Key Principles in 2D

It should be noted that most of the infromation presented here is adapted from the work detailed in [4]. RBL ensures convergence to the goal and provides sufficient conditions for achieving it. The problem involves individual control of N agents from their initial position $\mathbf{p}_i(0)$ toward a goal region, represented as circle. This goal region is denoted as $G(\mathbf{g}_i, r_g)$, where $\mathbf{g}_i$ is center and r_g is radius of goal region. The agent is progressing towards its designated destination, $\mathbf{d}_i$. Each agent knows its current position $\mathbf{p}_i$, encumbrance δ_i, which determines safe space around agent. Additionally, each agent also knows the positions and encumbrances of its neighboring agents $\mathcal{N}_\mathbf{i}$, agent $j \in \mathcal{N}_\mathbf{i}$ if $||\mathbf{p}_i - \mathbf{p}_j|| \leq \frac{r_{s,i}}{\eta}$, where $r_{s,i}$ is sensing radius of the i-th agent and η is a scaling parameter of cells. For simplicity $r_{s,i}$ is considered to be same for all agents, therefore $r_{s,i} = r_s$.

The core objective of the algorithm is to maximize the coverage cost function, which accounts for the distribution of agents and obstacles over the environment. This function is

expressed as:

$$J_{\text{cov}}(\mathbf{p}) = \sum_{i=1}^{N} \int_{\mathcal{V}_i} \|\mathbf{q} - \mathbf{p}_i\|^2 \varphi_i(\mathbf{q}) d\mathbf{q}, \tag{2.1}$$

where $\mathbf{p}_i$ is the position of agent i, $\mathcal{V}_i$ is the Voronoi cell of the i-th robot, $\|\mathbf{q}-\mathbf{p_i}\|^2$ is squared Euclidian distance between point in the mission space $\mathbf{q} \in \mathcal{Q}$ and agent's position p_i, and $\varphi_i(\mathbf{q})$ is the weighting function.

Voronoi cell is defined as:

$$\mathcal{V}_i = \{q \in \mathcal{Q} | \|\mathbf{q} - \mathbf{p}_i\| \leq \|q - \mathbf{p}_j\|, \forall j \neq i\}. \tag{2.2}$$

For visual representation see Fig. 2.1a.

However, this standard definition of Voronoi cells does not take into account the physical space occupied by the agents, their encumbrances. To address this, a Modified Voronoi cell is introduced, which takes into account the encumbrances of agents. This modified version adjusts the boundaries of each Voronoi cell to account for the encumbrances of neighboring agents. Also to enhance the algorithm performance, the Voronoi cell are scaled using a scaling parameter $\eta \in [0, 1]$. The modified Voronoi cell definition is as follows:

$$\tilde{V}_i = \begin{cases} \{\mathbf{q} \in Q \mid \|\mathbf{q} - \mathbf{p}_i\| \leq \eta \|\mathbf{q} - \mathbf{p}_j\|\}, & \text{if } \Delta_{ij} \leq \frac{\|\mathbf{p}_i - \mathbf{p}_j\|}{2} \\ \{\mathbf{q} \in Q \mid \|\mathbf{q} - \mathbf{p}_i\| \leq \eta \|\mathbf{q} - \tilde{\mathbf{p}}_j\|\}, & \text{otherwise.} \end{cases} \tag{2.3}$$

$\forall j \in \mathcal{N}_i$, where $\Delta_{ij} = \delta_i + \delta_j$ and $\tilde{\mathbf{p}}_j = \mathbf{p}_j + 2(\Delta_{ij} - \frac{\|\mathbf{p}_i - \mathbf{p}_j\|}{2}) \frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|}$. The parameter η controls the degree of ovelap or empty space between cells. Specifically, for the classical Voronoi cell definition ($\eta = 0.5$), each agent needs to detect its neighbors within a distance of $2r_s$. In general each agent needs to know its neighbors within distance of $\frac{r_s}{\eta}$.

Together with cell $\mathcal{S}_i$ defined as:

$$\mathcal{S}_i = \{\mathbf{q} \in \mathcal{Q} | \|\mathbf{q} - \mathbf{p}_i\| \leq r_{s,i}\}, \tag{2.4}$$

the cell $\mathcal{A}_i$ is obtained as $\mathcal{A}_i = \tilde{V}_i \cap \mathcal{S}_i$.

Convergence to goal region $G(\mathbf{g}_i, r_g)$ depends on the choice of weighting function that assigns weights to points $\mathbf{q}$ in the mission space $\mathcal{Q}$. The weighting function $\varphi_i(\mathbf{q})$ is defined as follows:

$$\varphi_i(\mathbf{q}) = \exp\left(-\frac{\|\mathbf{q} - \mathbf{d}_i\|}{\beta_i}\right), \tag{2.5}$$

where β_i is the weighting factor for points $\mathbf{q}$, and $\mathbf{d}_i$ represents the current destination of the agent. The destination is computed as follows:

$$\mathbf{d}_i = \mathbf{p}_i + R(\theta)(\mathbf{g}_i - \mathbf{p}_i), \tag{2.6}$$

where R is the azimuthal rotation. Rules for changing weighting function and rotation of destination:

- **Weighting rule**

$$\dot{\beta}_i(A_i) = \begin{cases} -k & \text{if } \beta_i > \beta_{\min} \wedge \|\mathbf{c}_{A_i} - \mathbf{p}_i\| < d_1 \wedge \|\mathbf{c}_{A_i} - \mathbf{c}_{\mathcal{S}_i}\| > d_2 \\ 0 & \text{if } \beta_i \leq \beta_{\min} \wedge \|\mathbf{c}_{A_i} - \mathbf{p}_i\| < d_1 \wedge \|\mathbf{c}_{A_i} - \mathbf{c}_{\mathcal{S}_i}\| > d_2 \\ -(\beta_i - \beta_i^D) & \text{otherwise,} \end{cases} \tag{2.7}$$

where the first case decreases β_i over time, the second case ensures that β_i does not decrease below its minimum threshold $\beta_{\min}$ (saturation) and the third case provides a general update rule when the previous conditions are not met.

- **Azimuth update rule**

$$\dot{\theta}_i = \begin{cases} k & \text{if } \theta < \frac{\pi}{2} \wedge \|\mathbf{c}_{\mathcal{A}_i} - \mathbf{c}_{\mathcal{S}_i}\| > d_4 \wedge \|\mathbf{p}_i - \mathbf{c}_{\mathcal{A}_i}\| > d_3 \\ -k & \text{if } \theta > 0 \wedge \neg(\|\mathbf{c}_{\mathcal{A}_i} - \mathbf{c}_{\mathcal{S}_i}\| > d_4 \wedge \|\mathbf{p}_i - \mathbf{c}_{\mathcal{A}_i}\| > d_3) \\ 0 & \text{otherwise,} \end{cases} \tag{2.8}$$

where the first case increases θ_i over time, the second case ensures that θ_i converges back when the distance constraints are not satisfied, and the third case keeps θ_i unchanged.

- **Azimuth reset rule**

$$\theta = 0 \quad \text{if } \theta = \frac{\pi}{2} \wedge \|\mathbf{p}_i - \overline{\mathbf{c}}_{\mathcal{A}_i}\|, \tag{2.9}$$

where $\overline{\mathbf{c}}_{\mathcal{A}_i}$ represents the centroid computed from the cell $\mathcal{A}_i$, which is weighted using the unrotated destination, meaning $\mathbf{d}_i = \mathbf{g}_i$.

Weighted centroid is computed as follows:

$$\mathbf{c}_{\mathcal{A}_i} = \frac{\int_{\mathcal{A}_i} \mathbf{q}\varphi_i(\mathbf{q})\, d\mathbf{q}}{\int_{\mathcal{A}_i} \varphi_i(\mathbf{q})\, d\mathbf{q}}, \tag{2.10}$$

where $\mathbf{q}$ represents the point in mission space, and $\varphi_i(\mathbf{q})$ is a weighting function. The centroids for other relevant cells are computed using a similar approach, but over different sets.

By applying the previously defined rules and computations, the RBL algorithm successfully guides agents movement toward the goal. However, these rules focus on rotating the centroid $\mathcal{A}_i$ in the azimuthal plane. To enhance performance in a fully three-dimensional space, additional rules are required to account for elevation adjustments.

2.3 Extension of the RBL algorithm to 3D

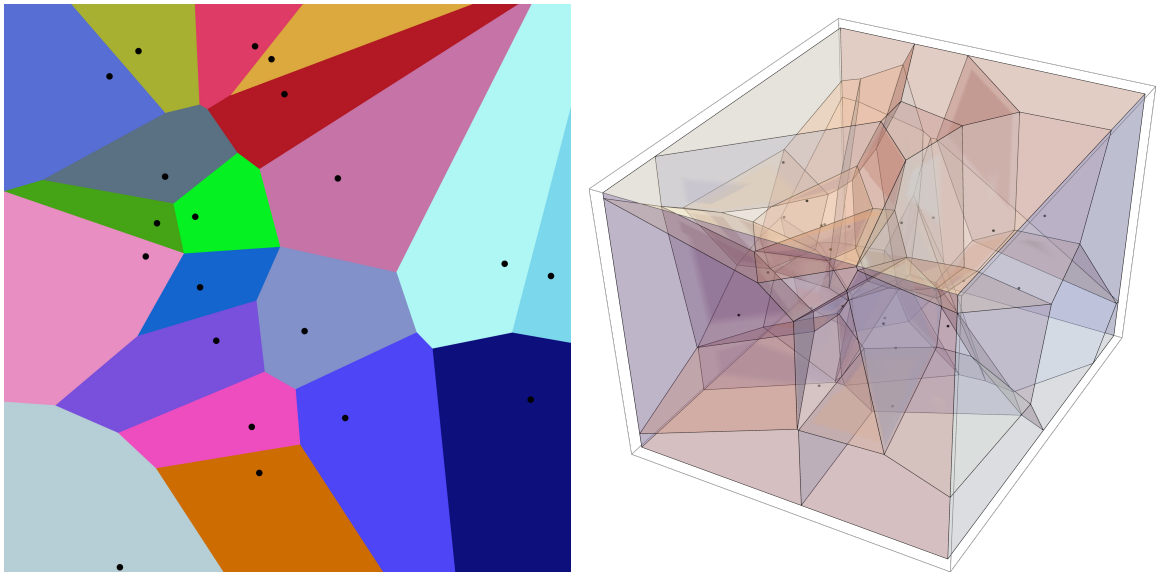
Motivation for 3D Extension

In many practical applications, agents must operate in three-dimensional spaces, considering not only horizontal movement but also vertical. A 3D extension is necessary to navigate complex environments that feature obstacles in all directions. In 2D, agents are restricted to a flat plane, which simplifies navigation but limits the ability to interact with objects and environments that exist in the third dimension. The ability to utilize vertical space can enhance energy efficiency, as agents can optimize their paths by ascending or descending to avoid obstacles or to find more favorable environmental conditions, such as discovering more open space. The transition from 2D to 3D also opens up possibilities for more advanced movement strategies, such as navigating through multi-level environments or optimizing trajectories by utilizing vertical space. Moreover, the use of 3D models allows for more accurate representations of real-world scenarios, where elevation plays a crucial role in decision-making and task execution.

Differences between 2D and 3D

The primary distinction in the 3D extension is that each goal region is now represented as a sphere rather than a circle. Similarly, the sensing cell $\mathcal{S}_i$ is also modeled as a sphere

instead of a circle, allowing for a more accurate representation of the UAV's perception in three-dimensional space. This change introduces a key modification when defining $\tilde{V}_i$: in the 2D case, a line was sufficient to slice the sensing region, whereas in 3D, a plane must be computed to properly segment the spherical sensing cell.



(a) Euclidean Voronoi Diagram in 2D (b) Euclidean Voronoi Diagram in 3D

Figure 2.1: (a) an example of 20 Voronoi cells in 2D [10] (b) 25 Voronoi cells in 3D [14]

Additional Constraints and Modifications

To extend the approach to the 3D case, several adjustments are made. The weighting rule (2.7) remains unchanged. However, in both the azimuth update rule (2.8) and the azimuth reset rule (2.9), only the displacement of centroids projected onto the xy-plane is taken into account. The modified formulations for the 3D case are as follow:

- **Azimuth update rule 3D**

$$\dot{\theta}_i = \begin{cases} k & \text{if } \theta < \frac{\pi}{2} \wedge \|\mathbf{c}_{\mathcal{A}_i} - \mathbf{c}_{\mathcal{S}_i}\|_{xy} > d_4 \wedge \|\mathbf{p}_i - \mathbf{c}_{\mathcal{A}_i}\|_{xy} > d_3 \\ -k & \text{if } \theta > 0 \wedge \neg(\|\mathbf{c}_{\mathcal{A}_i} - \mathbf{c}_{\mathcal{S}_i}\|_{xy} > d_4 \wedge \|\mathbf{p}_i - \mathbf{c}_{\mathcal{A}_i}\|_{xy} > d_3) \\ 0 & \text{otherwise.} \end{cases} \tag{2.11}$$

- **Azimuth reset rule 3D**

$$\theta = 0 \quad \text{if } \theta = \frac{\pi}{2} \wedge \|\mathbf{p}_i - \overline{\mathbf{c}}_{\mathcal{A}_i}\|_{xy}. \tag{2.12}$$

The notation $\|\cdot\|_{xy}$ denotes the Euclidean norm computed in the xy-plane only, defined as:

$$\|\mathbf{p}_i - \mathbf{p}_j\|_{xy} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{2.13}$$

Additionally, $Z_{clipping}$ is applied to each sensing cell $\mathcal{S}_i$, constraining it within the vertical limits defined by $\min_z$ and $\max_z$:

$$\mathcal{S}_i = \left\{ \mathbf{q} \in \mathcal{Q} \mid \|\mathbf{q} - \mathbf{p}_i\| \leq r_{s,i}, \quad \min_z \leq q_z \leq \max_z \right\}, \tag{2.14}$$

where $\min_z$ and $\max_z$ define the vertical bounds within which the sensing region $\mathcal{S}_i$ is restricted. This ensures that the agent cannot exceed these limits, as it follows the computed centroid $\mathbf{c}_{V_i}$. By constraining the sensing radius, the agent remains confined within the specified region, preventing it from moving outside the vertical interval $\min_z$ to $\max_z$.

The destination rotation rule Z_{rule} is introduced to enhance agents avoidance by rotating the computed destination $\mathbf{d}_i$ by an angle ϕ. For vertical rotation angle ϕ, the following condition is intorduced

$$\Omega = \|\mathbf{c}_{\mathcal{A}_i} - \mathbf{c}_{\mathcal{S}_i}\|_z < d_6 \wedge \|\mathbf{p}_i - \mathbf{c}_{\mathcal{A}_i}\|_z \vee |\|\mathbf{p}_i - \mathbf{c}_{\mathcal{S}_i}\|_{xy} - \|\mathbf{p}_i - \mathbf{c}_{\mathcal{A}_i}\|_{xy}| > d_7. \tag{2.15}$$

This condition has two main parts, combined by a logical or. The first part evaluates the vertical displacement of the centroid of the partitioned cell $\mathbf{c}_{\mathcal{A}_i}$ from the centroid of the sensing cell $\mathbf{c}_{\mathcal{S}_i}$ and the displacement of the agents position $\mathbf{p}_i$ from $\mathbf{c}_{\mathcal{A}_i}$. The second part considers the difference in the horizontal plane between the agent's position and the centroids $\mathbf{c}_{\mathcal{A}_i}$, $\mathbf{c}_{\mathcal{S}_i}$.

To improve agent distriution in the vertical dimension, a directional influence on vertical exploration is introduced. Given that each agent's position $\mathbf{p}_i$ and final goal $\mathbf{g}_i$, are known, this directional influence can be included into the update rule for ϕ_i.

First, the global heading angle, θ_{goal}, towards the goal is calculated:

$$\theta_{goal} = \text{atan2}(g_{i,y} - p_{i,y}, g_{i,x} - p_{i,x}), \tag{2.16}$$

where $\mathbf{g}_{i,x}$, $\mathbf{g}_{i,y}$ represent the x and y coordinates of the goal, and $\mathbf{p}_{i,x}$, $\mathbf{p}_{i,y}$ represent the x and y coordinates of the agent. The resulting angle is in the range $[-\pi, \pi]$. Next, the heading angle is linearly mapped to a direction influence $D_{\text{influence}}$ value in the range [-1, 1]:

$$D_{\text{influence}} = \frac{\theta_{goal}}{\pi}. \tag{2.17}$$

This mapping ensures that agents traveling directly northward (from South to North) have a directional influence close to one, that will promote upward vertical exploration. Similarly, agents traveling southward have a direction influence close to -1, promoting downward exploration. Agents moving primarily eastward or westward have a directional influence close to 0, resulting in minimal vertical exploration. This strategy encourages a more uniform distribution of agents throughout the 3D space.

To determine the adjustment to the agent's vertical orientation, the directional influence $D_{\text{influence}}$ is combined with the relative vertical displacement of the agent and $\mathbf{c}_{\mathcal{S}_i}$ Weighted average is used to combine them:

$$C_{\text{influence}} = \frac{w_1 \cdot (\mathbf{c}_{\mathcal{S}_i,z} - \mathbf{p}_{i,z}) + w_2 \cdot D_{influence}}{w_1 + w_2}, \tag{2.18}$$

where w_1 w_2 are weighting factors. The resulting combined influence $C_{\text{influence}}$ is then used to update the agent's rotation of it's destination $\mathbf{d}_i$ by ϕ_i.

Based on condition (2.15) and (2.18) update rule for ϕ_i can be constructed as follows:

$$\dot{\phi}_i = \begin{cases} k & \text{if } \phi_i < \frac{\pi}{4} > \wedge C_{\text{influence}} > 0 \wedge \Omega \\ 0 & \text{if } \phi_i > \frac{\pi}{4} > \wedge C_{\text{influence}} > 0 \wedge \Omega \\ -k & \text{if } \phi_i > -\frac{\pi}{4} > \wedge C_{\text{influence}} \leq 0 \wedge \Omega \\ 0 & \text{if } \phi_i < -\frac{\pi}{4} > \wedge C_{\text{influence}} \leq 0 \wedge \Omega \\ -k & \text{if } \phi_i > 0 \wedge \neg\Omega \\ k & \text{if } \phi_i < 0 \wedge \neg\Omega \\ 0 & \text{otherwise,} \end{cases} \tag{2.19}$$

where the first four cases control the rotation of the agent's destination $\mathbf{d}_i$ and the final three cases handle the smooth convergence of ϕ_i back to 0, when the condition Ω is not met.

After the application of the rules for θ_i and ϕ_i, the agent's destination is updated as:

$$\mathbf{d}_i = \begin{pmatrix} p_{i,x} + \|\mathbf{p}_i - \mathbf{g}_i\| \cdot \sin(\phi_{\mathbf{g}_i} + \phi_i) \cdot \cos(\theta_{\mathbf{g}_i} - \theta_i) \\ p_{i,y} + \|\mathbf{p}_i - \mathbf{g}_i\| \cdot \sin(\phi_{\mathbf{g}_i} + \phi_i) \cdot \sin(\theta_{\mathbf{g}_i} - \theta_i) \\ p_{i,z} + \|\mathbf{p}_i - \mathbf{g}_i\| \cdot \cos(\phi_{\mathbf{g}_i} + \phi_i) \end{pmatrix}, \tag{2.20}$$

where $\phi_{\mathbf{g}_i} = \arccos(\frac{\mathbf{g}_{i,z} - \mathbf{p}_{i,z}}{\|\mathbf{p}_i - \mathbf{g}_i\|})$ is the polar angle from z-axis to the goal, $\theta_{\mathbf{g}_i} = atan2(\mathbf{g}_{i,y} - \mathbf{p}_{i,y}, \mathbf{g}_{i,x} - \mathbf{p}_{i,x})$ is the azimuthal angle in the xy-plane to the goal.

Algorithm Walkthrough

At each iteration, the proposed algorithm performs a sequence of actions to control agent movement. These actions can be summarized as follows:

(i) **Cell Generation:**
For each agent, two cells are generated: a partitioned cell, denoted as $\mathcal{A}$ and a sensing cell, denoted as $\mathcal{S}$.

(ii) **Centroid Computation:**
- The centroid of cell $\mathcal{A}$, $c_{\mathcal{A}}$ is computed using a weighted function that considers both the agent's goal position, $\mathbf{g}_i$, and its current destination, $\mathbf{d}_i$.
- The centroid of cell $\mathcal{S}$, $c_{\mathcal{S}}$ is computed using a weighted function that considers only the agent's current destination, $\mathbf{d}_i$.

(iii) **Rule Application and Destination Update:**
A set of rules is applied to:
- Adjust weighting function usied in the centroid computations (2.7).
- Rules to update θ_i (2.11) and ϕ_i (2.19) are applied.
- Update the agent's destination $\mathbf{d}_i$ (2.20).

(iv) **Agent movement**
Finally, the agent is directed towards a centroid location $c_{\mathcal{A}}$.

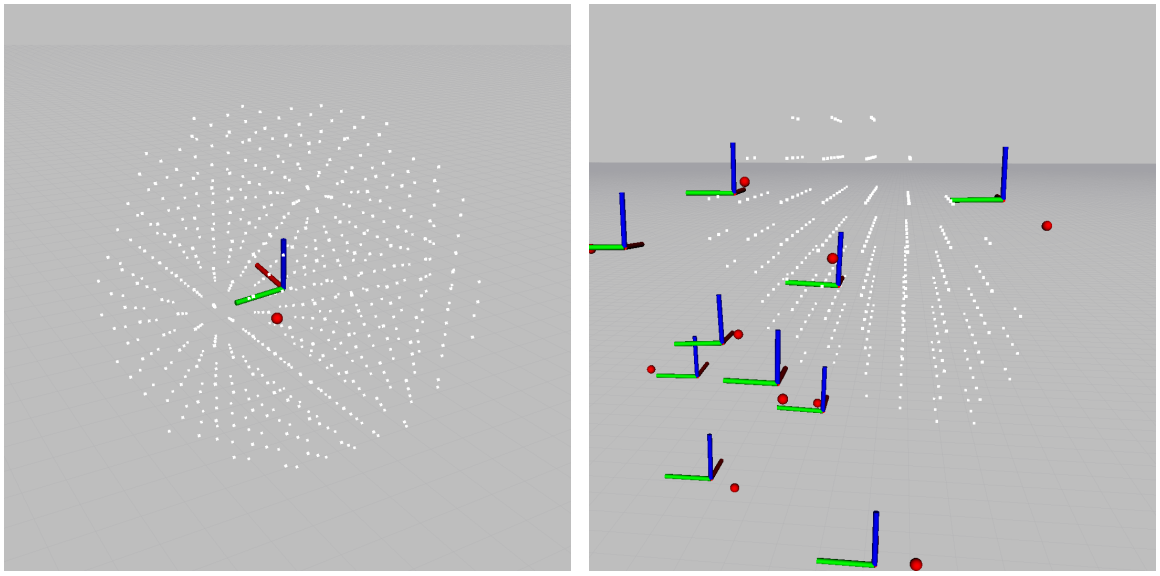
This process is repeated until all agents reach their goals.

2.4 Simulation and Results Analysis

Computational Implementation Details

The continuous algorithm presented in the previous section can be easily discretized for implementation in a computational enviroment. Instead of continuous sets, the cells $\mathcal{A}$ and

$\mathcal{S}$ are aproximated using a set of discrete points. Discrete points are partitioned out when definition (2.3) is not met.



(a) Example of discretized cells ($\mathcal{A} = \mathcal{S}$)

(b) Example of a partitioned cell $\mathcal{A}$

Figure 2.2: Discrete approximation of Voronoi cells, with centroid $c_{\mathcal{A}}$ shown as a red dot.(a) Example where the partitioned cell $\mathcal{A}$ and the sensing cell $\mathcal{S}$ are the same. (b) Example where the partitioned cell $\mathcal{A}$ is partitioned by other UAVs.

Simulation Enviroment

In this simulation, the term 'agent' refers to the UAV, which were simulated using the framework provided by Multi-robot Systems Group (MRS) [7]. Each UAV obtains its global ground truth position from the ROS simulator. The positions of neighboring UAVs were estimated using simulated blinking ultraviolet markers, following the method described in [21] and [8]. The following constraints are relevant for each UAV:

Parameter	**Value**
Maximal horizontal velocity $[\mathrm{m\,s^{-1}}]$	4.0
Horizontal acceleration $[\mathrm{m\,s^{-2}}]$	2.0
Maximal ascending velocity $[\mathrm{m\,s^{-1}}]$	2.0
Vertical ascending acceleration $[\mathrm{m\,s^{-2}}]$	1.0
Maximal descending velocity $[\mathrm{m\,s^{-1}}]$	2.0
Vertical descending acceleration $[\mathrm{m\,s^{-2}}]$	1.0

Table 2.1: Motion constraints of the UAV.

Simulation Scenarios

To evaluate the safety and performance of the UAVs during interactions, a series of experiments was conducted. Experiments were performed primarily using circular formations,

with UAV counts of N = 5, 10, and 15. Each of these circular formation experiments was repeated 10 times. Additionally, to extend the analysis to 3D and assess the algorithm's performance, a set of spherical formation experiments was conducted with N = 10, also repeated 10 times. After each UAV was flown from the ground to its starting position, the RBL algorithm was activated.

These formations provide a controlled enviroment compared to random initial position and goal locations. In both circular and spherical formations, the UAVs were evenly distributed along the circle or the surface of the sphere, with the goal position located on the opposite side. The radius of both circle and the sphere was set to 5 meters.

The following metrics were measured across the 10 repetitions of each experiment, along with their standard deviations: success rate SR [%], average trajectory length $\overline{L}$ [m], average time to reach the goal $\bar{t}$ [s], average of the maximal time for UAV to reach the goal $\bar{t}_{\max}$ [s], and average velocity $\overline{v}$ [m/s].

The specific values of the parameters used in the experiments are summarized in following table 2.2:

Table 2.2: Parameters Used in Experiments

Parameter	Value
Sensing radius r_s [m]	3.5
Update rate [Hz]	10
Encumbrance δ_i [m]	0.5
$d_1 = d_3 = d_5$ [m]	0.5
$d_2 = d_4 = d_6$ [m]	1.0
d_7 [m]	0.2
$\min_z$ [m]	1.0
$\max_z$ [m]	10.0
β_i^D [ ]	1.5
η [ ]	0.9
w_1 [ ]	0.7
w_2 [ ]	0.3

- **Experimental Results**

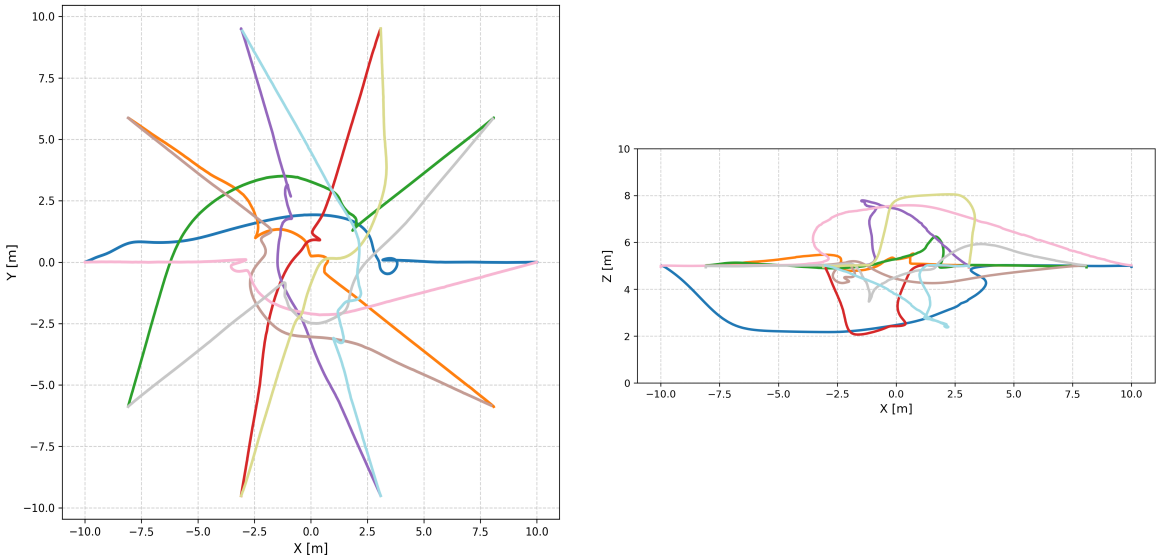
Full experimental results and trajectory visualizations are presented in Chapter A. The following tables highlight key results for the N=10 agent scenarios:

- $N = 10$ **Circular Formation:**

	SR [%]	$\overline{L}$ [m]	$\bar{t}$ [s]	$\bar{t}_{\max}$ [s]	$\overline{v}$ [m/s]
RBL 2D	100.00	22.95 ± 1.64	30.79 ± 2.28	34.73 ± 0.77	0.74 ± 0.07
RBL 3D	100.00	22.22 ± 0.89	30.05 ± 2.61	34.39 ± 4.19	0.73 ± 0.05
RBL $3D_{\text{clipped}}$	100.00	22.31 ± 0.69	30.22 ± 1.83	33.22 ± 0.93	0.73 ± 0.04
RBL $3D_z$	100.00	22.38 ± 0.88	28.80 ± 2.30	32.35 ± 1.25	0.77 ± 0.05

- $N = 10$ **Spherical Formation:**

	SR [%]	$\overline{L}$ [m]	$\overline{t}$ [s]	$\overline{t}_{\max}$ [s]	$\overline{v}$ [m/s]
RBL 3D	100.00	14.32 ± 1.52	27.76 ± 3.06	32.48 ± 2.26	0.51 ± 0.05
RBL 3D$_z$	100.00	14.06 ± 0.97	27.17 ± 2.66	31.86 ± 1.24	0.55 ± 0.06



(a) Top-Down View (X-Y)

(b) Side View (X-Z)

Figure 2.3: Effect of the Z-axis rule on the circular crossing.

2.5 Comparison with State-of-the-Art Method

As an alternative approach developed within the MRS framework, the 3D RBL algorithm presented here can be compared to other state-of-the-art methods like PACNav [15]. Both algorithms target communication-less UAV coordination, a feature desirable for scalability. Furthermore, implementations described for both approaches have utilized UVDAR to determine the positions of neighbouring UAVs.

A key requirement for both is some form of self-position estimation. PACNav achieves this in GNSS-denied settings using SLAM techniques. Similarly, RBL can also operate without GNSS provided a robust odometry source like SLAM is available to determine the UAV's own position within a consistent frame.

Where the approaches differ notably is in their coordination strategy. PACNav utilizes a leader-follower dynamic: 'uninformed' UAVs identify and follow 'informed' (or otherwise reliable) leaders based on observed motion characteristics like path persistence and similarity, encouraging group cohesion. In contrast, the RBL algorithm assigns an individual goal to each UAV. Coordination emerges as each agent calculates its path based on its local modified Voronoi cell, moving towards its computed centroid while implicitly accounting for neighbours.

Regarding validation, PACNav has demonstrated performance through real-world experiments, including navigation in a forest environment. Validation of the multi-agent 3D RBL algorithm was conducted through simulation. The integration and experimental validation of 3D LiDAR for environmental perception and single-agent navigation within the RBL framework are addressed in the subsequent chapter.

Another distinct approach in decentralized, communication-less coordination is the purely vision-based model presented by Mezey [6] Unlike RBL, which relies on explicit position information, the Mezey et al. model derives control actions from a 1D visual projection field generated by detecting neighbours via onboard camera. It requires no localization data. Coordination emerges from low-level visual attraction-repulsion forces, resulting in patterns like flocking or swarming rather than directed motion towards individual goals. While effectively demonstrated in 2D with UGVs, extending this purely vision-based model to complex 3D environments presents challenges regarding goal achievement. Furthermore, the use of toroidal simulation environments, which mitigate group fragmentation and can facilitate specific emergent patterns like leader-follower dynamics, potentially limits the direct applicability of some observed behaviors to real-world 3D scenarios. The 3D RBL, by utilizing explicit position information, aims for more predictable goal-oriented navigation in 3D, although with different sensing requirements compared to the purely reactive, vision-only model of Mezey et al.

2.6 Summary and Key Insights

This chpater detailed teh extension of RBL algorithm, originally tested for 2D coordination, to operate in a full three-dimensional space. Key modification included adapting The Voronoi cell partitioning for 3D, utilizing projections for already existing azimuth update rules, and introducing a new elevation rotation angle. This new rule dynamically adjusts the agent's vertical destination based on relative centroid positions and directional influence towards the goal, aiming to imporve distribution. Note that for the directional influence to work the UAVs need to have some sort of information about frame of reference between each other at the start of the algorithm, such as Earth's magnetic field. Vertical movement has also been constrained within defined interval ($\min_z$, $\max_z$).

The proposed 3D RBL extensions were evaluated through simulations within the MRS framework, comparing performance UAVs crossing circle and sphere. The results indicate that the extension variants successfully enabled multi-agent coordination in three dimensions. Specifically, analysis of the simulation results reveals a trend where the performance of the 3D approach with applied rules compared to the basic 2D appears to increase with scaling number of agents. For instance, based on the average completion times, the 3D variant was calculated to be aproximately 1.99% faster than the 2D variant for N=5 agents. This performance is scales for larger swarms. For N=15 the 3D approach was calculated to be 4.22% faster than the 2D. This suggests that the extension becomes increasingly beneficial in more crowded scenarios.

While the magnitude of this observed improvement was perhaps less than initially anticipated, it is important to consider the conservative motion constraints for UAVs on the Z-axis (velocity/acceleration), reflecting realistic UAV dynamics but limiting vertical exploration. Furthermore, the parameters used for testing simulation were also conservative, suggesting that finer tuning - particularly of vertical exploration weights and weighting function agressiveness - could potentially yield better performance. Nonetheless, the trend indicated by these results supports the conclusion that the 3D extension effectiveness crowdeness rises. Furthermore, simulations in the spherical formation scenario confirmed the algorithm's capability to manage 3D navigation and coordination.

In conclusion, the simulations presented in this chapter demonstrate the feasibility of extending the RBL algorithm to 3D. The introduced elevation control rule appears particularly beneficial, enhancing the efficiency and convergence speed of the agents towards their goals in

simulated 3D environments. This provides a promising foundation for applying the 3D RBL algorithm to real-world UAV navigation tasks, explored further in subsequent chapter.

2.7 Future Work

Several ideas for future research and development emerge from the work presented in this chapter:

- **Real-World Multi-Agent Testing:**
  While the subsequent chapter explores single-agent navigation in a real forest, the multi-agent coordination aspects of the 3D RBL algorithm lack real-world validation. Initial plan to conduct such tests were set back by practical challenges, including the lack of full $360° \times 180°$ sensor coverage, which resulted in blind spots, particularly above and below the UAV. Specifically, hardware limitations on the MRS UAV for swarming [22], including emitter and camera placement that created blind spots (especially directly above and below), restricting the UVDAR system [8] from providing the complete spherical coverage needed for 3D coordination experiments. Resolving these hardware and sensing limitations to facilitate robust multi-agent 3D experiments remains a key step for future research.
- **Parameter Tuning using Reinforcement Learning:**
  The performance of the RBL algorithm is sensitive to several parameters. Reinforcement Learning (RL) could be used to automatically tune these parameters for optimal performance in specific environments or scenarios such as crossing circle scenario. Parameters suitable for RL-based tuning include the vertical exploration weights (w_1, w_2), the scale parameter (η), aggressiveness parameter (β_i^D for $\varphi_i(\mathbf{q})$), the sensing radius (r_s), and potentially the distance thresholds (d_1 through d_7) for centroid displacement.
- **Neural Network-Based Navigation Policy:**
  An alternative direction would be to replace the rule-based approach entirely with a learned policy using Neural Network (NN).
  - Input Representation: The primary NN input could be a fixed-size 3D tensor representing a discretized voxel grid centered on the UAV's current position (e.g., n×n×n, where n is an odd integer defining the sensing region). Each element in this tensor would hold a binary value: '1' indicating free space and '0' indicating occupied or unknown space. This provides the network with a structured representation of the current surroundings. Additionally, the goal position (x_g, y_g, z_g) would serve as a separate input, directing the policy towards the desired destination.
  - Output: The NN's output could directly be the next desired local waypoint (a point within free-space part of the input array).
  - Training: Training such an NN, likely using RL, would require a suitable simulation environment. This environment should feature dynamic obstacles (other agents) and static obstacles, along with randomized initial and goal positions to promote generalization. The reward structure should be designed to guide the learning effectively by penalizing collisions (with obstacles or other agents), penalizing the selection of invalid next waypoints (those outside the determined free-space), and rewarding progress towards the goal.

3 UAV Navigation Using the RBL Algorithm and LiDAR Sensing

3.1 Introduction

Motivation

UAVs are increasingly being deployed in challenging, unstructured environments like dense forests, moving beyond their traditional use in open areas. This shift is driven by a growing demand for autonomous solutions in sectors like environmental monitoring, forestry management (health assessment [13]), and search and rescue, where ground-based access is difficult. Furthermore, this work aligns with the broader trend of deploying UAVs for increasingly complex tasks, such as detailed infrastructure inspection (e.g., power lines, where autonomous navigation in cluttered, potentially high Electromagnetic Interference (EMI) environments is essential (Fly4Future [23])). Enabling UAVs to reliably navigate point-to-point within these challenging settings is a foundational step towards realizing these advanced applications safely and efficiently.

While the primary objective of this work is to enable successful point-to-point navigation within a forest using the RBL algorithm, a significant secondary benefit emerges from this process. By successfully navigating from point A to point B while simultaneously building a map of the traversed environment, the UAV generates valuable spatial data about the forest structure. This autonomously generated map can then serve a vital purpose: enabling enhanced planning for future missions within the same area. Once a map exists, subsequent UAV operations could potentially transition from purely reactive navigation strategies to more efficient, globally informed path planning algorithms such as TODO find some. Leveraging prior knowledge of obstacle locations could significantly improve the safety and efficiency of future routine tasks.

Problem Statement

Operating UAVs effectively within complex, three-dimensional environments like forests presents significant navigational challenges that hinder widespread autonomous deployment. The primary problems addressed in this work stem from:

(i) **Global Navigation Satellite System (GNSS)-Denied Conditions:**
Within forests, the dense canopy and other obstructions frequently block or scatter GNSS signals, leading to unreliable or completely absent reception. This necessitates reliance on onboard sensors and algorithms (like SLAM based on Light Detection and Ranging (LiDAR) and Inertial Measurement Unit (IMU) data) for accurate localization and state estimation.

(ii) **Cluttered and Unpredictable Environments:**
Forests are inherently cluttered with numerous static obstacles (trees, trunks, branches) and potentially dynamic ones (animals, leafs, falling branches). The navigation system must be capable of perceiving these obstacles in real-time and planning safe paths around them without prior knowledge of their exact layout.

Therefore, the core problem is to develop and validate a robust autonomous navigation system that allows a UAV to reliably traverse between specified points in a cluttered, GNSS-denied

forest environment using only onboard sensing and computation.

Objectives

The primary objectives within this chapter are:

- **Integrate the RBL with Onboard Sensing:**
  Adapt the core of the RBL to utilize real-time sensor data. This includes modification of sensing cell $\mathcal{S}$, because used LiDAR doesnt see in all directions. Sensing cell needs to remain convex so the centroid computed from partitioned cell $\mathcal{A}$ is never computed outside of the set - for safety convergence towards the goal.
- **Process Point Cloud Data:**
  Implement neccessary filtration to refine the raw point cloud data acquired from the LiDAR.
- **Integrate external packages on the UAV:**
  Integrate and configure external software packages for simultaneous environmental mapping and robust state estimation, suitable for operation within the GNSS-denied forest environment.
- **Experimental Validation:**
  Conduct experiments to evaluate the performance, robustness, and effectiveness of the complete navigation solution. This validation will be performed through real-world flight tests in an actual forest.

Chapter Overview

This chapter covers the LiDAR perception process, covering data acquisition, preprocessing, mapping, and the method used to integrate mapped voxel data into the RBL algorithm. Subsequently, the practical challenges of implementing the system on a UAV with sensor limitations are discussed, along with the software solutions employed. The chapter proceeds to present the experimental validation, outlining the simulation setup and results, followed by the real-world forest experiment methodology, challenges, and performance analysis, illustrated with videos of successful flights [2], [1] and a mapping failure case [3]. Finally, a comparative analysis, discussion of limitations, and summary of key findings are presented.

3.2 LiDAR-Based Perception and Point Cloud Processing

Overview of LiDAR for UAV Navigation

LiDAR is a crucial sensing technology widely used in applications such as SLAM [11], autonomous vehicles [20], UAVs TODOcite, and precision agriculture [9]. It provides high-resolution spatial data about the surrounding environment, making it a valuable tool for perception and navigation in dynamic and complex environments. For UAV applications, LiDAR serves several essential functions:

- **3D Mapping** – Capturing a detailed representation of terrain, structures, and obstacles.
- **Obstacle Detection** – Identifying objects and estimating their position relative to the UAV for collision avoidance.
- **Autonomous Path Planning** – Assisting navigation algorithms by providing spatial information for decision-making.
- **Terrain Following** – Helping the UAV maintain a safe altitude by detecting variations in ground elevation.

LiDAR offers several benefits that make it an attractive choice for UAV-based navigation:

- **High Accuracy** – Provides precise distance measurements, crucial for obstacle avoidance and localization.
- **Environment Agnostic** – Functions effectively in various conditions, including low-light environments and featureless terrain where cameras may fail.
- **Fast Data Acquisition** – Captures thousands to millions of points per second, enabling real-time processing.
- **Rich Depth Information** – Unlike cameras that provide only 2D images, LiDAR generates accurate depth data, improving spatial awareness and 3D perception.

Despite its advantages, LiDAR also presents certain challenges and limitations:

- **Computational Complexity** – Processing large point clouds in real-time requires significant computational power, which may be a limitation for UAVs with low processing resources.
- **Sensor Noise and Artifacts** – External factors such as vibrations and motion of UAV can introduce errors in point cloud data.
- **Limited Field of View (FoV)** – The placement of the LiDAR sensor on the UAV affects its coverage, requiring strategies to compensate for blind spots.
- **Environmental Interference** – Performance may degrade in challenging conditions such as fog, rain, or dense vegetation due light deviation.
- **Power Consumption** – LiDAR sensors can consume a significant amount of power, which reduces the overall flight time of the UAV.
- **Interference with Other LiDARs** – LiDAR sensors can experience interference when multiple units are used nearby, potentially leading to faulty measurements.

Point Cloud Data Acquisition

LiDAR systems determine object distances by emitting laser pulses and measuring the time it takes for the reflected light to return. This process, known as Time-of-Flight (ToF), involves scanning the environment with laser beams directed at varying horizontal and vertical angles. The reflected light, modulated in intensity, phase, or frequency, is captured by a receiver, which uses a lens to focus the signal onto a photodetector. This detector converts the light into an electrical signal via the photoelectric effect [17].

The system calculates distance based on the light's travel time, considering its near-light-speed propagation. To distinguish transmitted from received signals, the laser's wavelength is often adjusted. Subsequent signal processing filters and analyzes the electrical signal, accounting for surface material and environmental variations. The output is a 3D point cloud representing the scanned environment, along with reflected laser energy intensities. All these data points are stored in a ROS message of type $sensor\_msgs :: PointCloud2$

Preprocessing Techniques

To efficiently process LiDAR data and reduce computational complexity, the raw point cloud undergoes downsampling and filtering. The point cloud density is reduced using a voxel grid filter. Subsequently, points associated with the UAV's structure are removed based on its known encumbrance.

- **Voxel Grid Downsampling** – The raw LiDAR point cloud often contains a large number of points, which can be computationally expensive to process in real-time. To

address this, we apply a voxel grid filter using the Point Cloud Library (PCL) [24]. This method partitions the 3D space into a grid of voxels with a given resolution (leafSize) and retains a single representative point per voxel. The filtering process reduces the number of points while preserving the overall structure of the environment.

- **Filtering Points Corresponding to the UAV Structure** – LiDAR sensors mounted on UAVs can capture unwanted points originating from the UAV itself, such as reflections from its frame or rotor rods. To prevent these points from interfering with navigation, additional filtering step has been applied. Points falling outside a specified distance range (closer than a minimum threshold or farther than a maximum threshold) are filtered out.

The resulting filtered point cloud contains only relevant environmental features while eliminating unnecessary points, improving efficiency for future processes. The point cloud is than fed into Point lio state estimator and bonxai mapping.

Bonxai Mapping

As will be shown later in this chapter, prior terrain knowledge is beneficial for navigation. For this purpose, environmental mapping was accomplished using a simple package developed by the MRS group, implementing the Bonxai mapping approach. Specific details and a citation for this package are omitted as it represents an internal MRS solution currently under development, however, various publicly available packages offer similar voxel-based mapping capabilities. The resulting map is represented as a voxel grid, which is then used by the RBL algorithm for navigation planning.

Initially, an approach involving surface reconstruction from this voxel data was investigated. This involved estimating surface normals from the point cloud combined with Greedy Projection Triangulation (GP3) method from the PCL library to generate polygonal mesh approximating the environment's surface. While this method could produce relatively accurate surface representation, it turned out to be computationally expensive and too slow for real-time execution on the UAV's onboard computer.

Due to these limitations, the surface reconstruction approach was abandoned. Instead a simpler and more efficient method was adopted, which involves directly using the information about the environment stored in voxel grid map. This direct voxel usage approach is detailed in the subsequent section. Consequently, further development of the mesh-based surface reconstruction method is not recommended for this application.

Voxel-Based Modification of Cell $\mathcal{A}$

Given that the size of the voxels is known, this infromation can be used to refine the partitioning of cell $\mathcal{A}$ by treating voxels as obstacles. First, it is necessary to determine which voxels are relevant to the RBL algorithm. This can be achieved by considering the scaling parameter η. A voxel is considered relevant if it lies within radius of $\frac{r_s}{\eta}$ of the agent. For each discrete point in in the set composing cell $\mathcal{S}$, the nearest voxel is identified using k-d tree algorithm from the PCL library. Because the found point is the voxel's center, the closest pint within that voxel's boundaries is calculated to accurate partition the cell $\mathcal{A}$ from cell $\mathcal{S}$.

Given a point $\mathbf{p}_\mathcal{S}$, the voxel center $\mathbf{v_c}$ and the voxel edge length $\mathbf{e}$, $\mathbf{p_{closest}}$, is computed

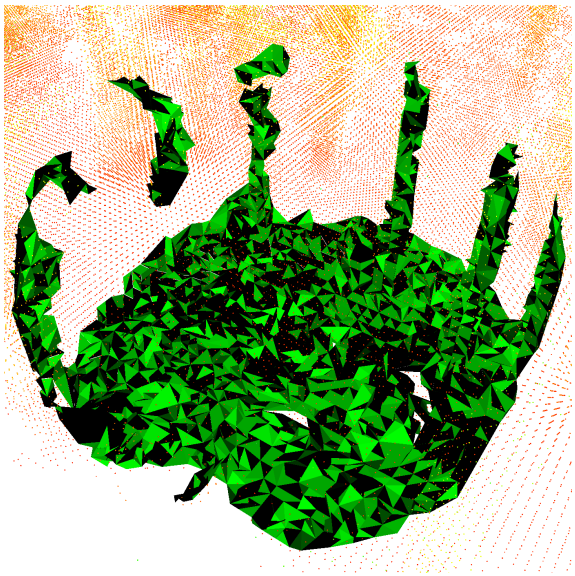


Figure 3.1: Surface approximation using Greedy Projection Triangulation.

as:

$$\mathbf{p}_{closest} = \begin{pmatrix} \text{clamp}(p_{\mathcal{S}}, x, v_{c,x} - \frac{\mathbf{e}}{2}, v_{c,x} + \frac{\mathbf{e}}{2}) \\ \text{clamp}(p_{\mathcal{S},y}, v_{c,y} - \frac{\mathbf{e}}{2}, v_{c,y} + \frac{\mathbf{e}}{2}) \\ \text{clamp}(p_{\mathcal{S},z}, v_{c,z} - \frac{\mathbf{e}}{2}, v_{c,z} + \frac{\mathbf{e}}{2}) \end{pmatrix}, \tag{3.1}$$

where function clamp(x, a, b) constrains the value x to the range [a, b]. This constrains $\mathbf{p}_{\mathcal{S}}$ to the voxel boundaries.

A point is excluded from the set if $\|\mathbf{p_i} - \mathbf{p}_{\mathcal{S}}\| \leq \eta \cdot \|\mathbf{p_i} - \mathbf{p}_{closest}\|$.

3.3 Implementation and Integration on UAV

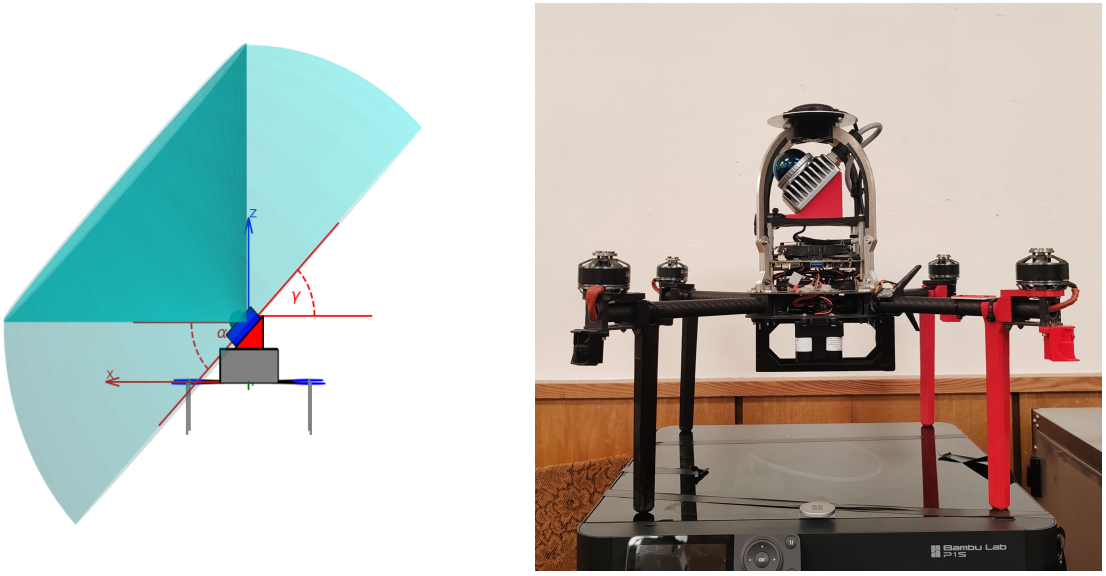
Challanges in Integration

The algorithm's performance is influenced by the limitations of the LiDAR sensor used for environmental sensing. While the algorithm functions optimally with a full 360° horizontal and 180° vertical Field of View (FOV), which would require multiple sensors, the experiments used a single Livox Mid 360 LiDAR [12]. This LiDAR provides a 360° horizontal FOV and only a 59° vertical FOV, resulting in a sensing blind spot.

To solve this limitation, several modification were implemented. The LiDAR was mounted at an angle γ, as shown in Figure Fig. 3.2, to enhance ground sensing and increase forward visibility.

In addition to the LiDAR mounting angle, a software modification is implemented to account fo the LiDAR's limited FOV. The algorithm uses map to partition cell $\mathcal{S}$ into cell $\mathcal{A}$, from which the centroid is computed to guide UAV movement. To ensure safety, UAV should only move only within their visible FOV. However, map information from areas outside the current FOV can still be valuable.

Therefore, a constraint is intorduced - the UAV maintains its yaw rotation towards the current centroid $\mathbf{c_A}$ and moves towards it only if the centroid lies within a defined angular range in fron of the UAV.



(a) UAV model with visualized LiDAR mounting parameters.

(b) UAV used in the experiment with mounted LiDAR

Figure 3.2: UAV and LiDAR mounting scheme. Subfigure (a) shows a modeled UAV with visualized LiDAR mounting parameters, including the LiDAR elevation field of view α and tilt angle γ. Subfigure (b) displays the real UAV used in the experiment with the LiDAR mounted according to the same configuration.

To incorporate the LiDAR's FOV into cell $\mathcal{S}$, two planes are defined based on the LiDAR's mounting configuration and vertical FOV. Let $\mathbf{e}_z$ be unit vector alog the z-axis, $R_{off} \in SO(3)$ the offset rotation, and $R_{rpy} \in SO(3)$ the roll-pitch-yaw rotation. The normal vectors, $\mathbf{n}_1$ and $\mathbf{n}_2$ defining two planes are given by:

$$\mathbf{n}_1 = R_{rpy} \cdot R_y(\gamma) \cdot \mathbf{e}_z \tag{3.2}$$

$$\mathbf{n}_2 = R_{rpy} \cdot R_y(\gamma - \alpha) \cdot \mathbf{e}_z, \tag{3.3}$$

where α is the LiDAR's FOV and γ is the the mounting angle of the LiDAR.

Cell $\mathcal{S}$ is then modified by excluding points that lie outside the LiDAR's FOV, defined by these two planes:

$$\mathcal{S}'_i = \{\mathbf{q} \in \mathcal{S}_i \mid \mathbf{n}1 \cdot (\mathbf{q} - \mathbf{p}_{LiDAR}) \geq 0 \wedge \mathbf{n}2 \cdot (\mathbf{q} - \mathbf{p}_{LiDAR}) \leq 0\}, \tag{3.4}$$

where $\mathcal{S}'_i$ is the modified cell $\mathcal{S}$ and $\mathbf{p}_{LiDAR}$ is the exact position of the LiDAR sensor.

This modification effectively restricts the points considered in the calculation of the centroid to only those within the LiDAR's FOV.

To incorporate map information from the surrounding area outside of the LiDAR's current FOV, the following procedure is used. Firstly, both cells $\mathcal{S}_i$ and $\mathcal{S}'_i$ are created. These cells are partitioned into cells $\mathcal{A}_i$ and $\mathcal{A}'_i$ using voxels. Cell $\mathcal{A}'_i$ is partitioned using voxels taht are actively sensed, while cell $\mathcal{A}_i$ is partitioned using the voxels from the whole surrounding.

If the centroid computed from cell $\mathcal{A}_i$, $\mathbf{c}_{\mathcal{A}_i}$, is within the cell $\mathcal{A}'_i$, the UAV is commanded to move towards $\mathbf{c}_{\mathcal{A}_i}$ while simu rotating its yaw towards it. However, if $\mathbf{c}_{\mathcal{A}_i}$ lies outside cell

$\mathcal{A}'_i$, it is projected onto the closest point on the boundary of cell $\mathcal{A}'_i$. As the UAV only moves if the centroid is within a certain angle in front of it, this projection onto the boundary of $\mathcal{A}'_i$ ensures that the UAV primarily rotates towards the centroid until the centorid is within an angle in front of the UAV.

This procedure effectively utilizes the information fro mthe map while ensuring that the UAV moves only within its actively sensed FOV.

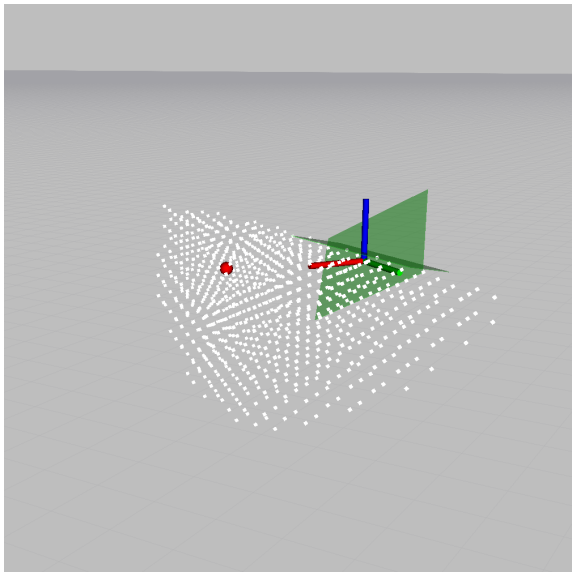


Figure 3.3: RViz visualization of LiDAR field-of-view (FOV) planes. The UAV is represented by the XYZ coordinate axes. The green squares represent the two planes, defined by normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$, used to slice cell $\mathcal{S}_i$ and determine the modified cell $\mathcal{S}'_i$. The red sphere represents the centroid $\mathbf{c}_{\mathcal{A}_i}$.

3.4 Experimental Results

This section presents the findings from both simulated and real-world experiments conducted to evaluate the proposed approach.

Simulation Setup

In the simulations, a virtual forest environment was created. Initially, a raycasting approach from the UAV to the simulated trees was considered for obstacle detection. However, due to its computational cost, a static point cloud representation of the entire forest was generated and published instead. The forest was modeled as a collection of trees, with trunks represented by cylinders and crowns by spheres.

The forest was contained within a rectangular area defined by the following boundaries:

- x_{min} = -20.0 m
- x_{max} = 20.0 m
- y_{min} = -32.0 m
- y_{max} = 32.0 m

A total of 100 trees were randomly generated within this area, with a minimum separation distance of 3 meters between them. The trees were generated with the following parameter

variances:

- Trunk radius: 0.5 ± 0.2 m
- Tree height: 7.5 ± 2.0 m
- Crown radius: 2.5 ± 0.5 m

The UAV's starting position was set at one corner of the rectangular area (-22.0, -33.0, 2.0), and the goal position was set at the opposite corner (22.0, 33.0, 5.0), resulting in an approximate distance of 80 meters between them.

It is important to note that the right-hand rule and z rule, described in the previous chapter, are not required for effective coordination in this static forest environment. Also, the point-lio estimator and Bonxai mapping were not simulated.

The specific values of the parameters used in the experiments are summarized in following table 3.1:

Table 3.1: Parameters Used in Experiments

Parameter	Value
Sensing radius r_s [m]	3.5
Update rate [Hz]	10
Encumbrance δ_i [m]	0.5
$d_1 = d_3 = d_5$ [m]	0.5
$d_2 = d_4 = d_6$ [m]	1.0
β_i^D [ ]	0.5
η [ ]	0.9

The UAV dynamics are the same as described in the previous chapter and detailed in Table 2.1. Each simulation was run 10 times.

- **Performance in Simulated Environments**

The simulation results highlight the proposed solution capability in navigating inside dense forest environments. The UAV demonstrated effective obstacle avoidance, reacting appropriately to simulated trees while maintaining efficient trajectory without unnecessary detours. Convergence towards the designated goal waypoint was consistently achieved in a stable manner.

	SR [%]	$\overline{L}$ [m]	$\overline{t}$ [s]	$\overline{v}$ [m/s]
Results	100.00	91.99 ± 0.96	150.89 ± 0.88	0.85 ± 0.00

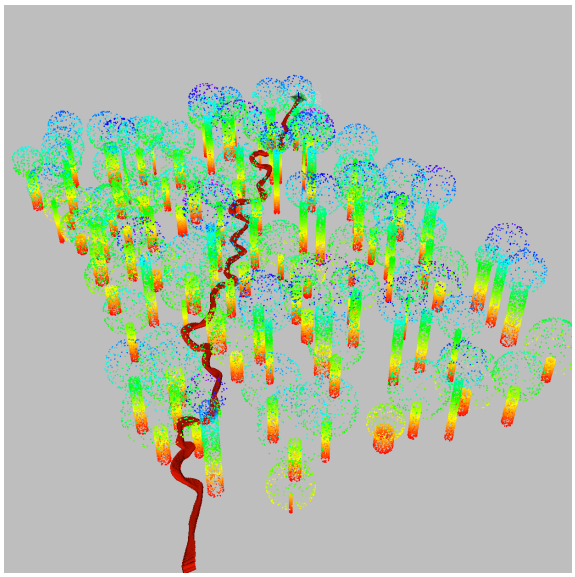


Figure 3.4: Visualization of simulated forest and the corresponding path of the UAV, indicated by a red line.

- **Real-World Experiments in a Forest Environment**

Following the successful simulation validation, the RBL algorithm was tested through real-world experiments conducted in a forest environment.

- **Location:**
  The chosen forest site presented a relatively clear environment, considered 'friendly' for initial UAV flight tests. To increase the navigational challenge and better test the obstacle avoidance capabilities, additional branches were placed near trees within the flight area after a few successfull flights. The point cloud filtration parameters were set to discard points closer than 1.0 meter (to avoid detecting the UAV's own propellers) and farther than 40 meters.
- **UAV Platform:**
  A custom-built multirotor UAV from the MRS group was used for this experiment (TODO ask for specifications). It was equipped with a primaryly with LiDAR mounted on top for environmental perception and state estimation. While other sensors like GPS (unreliable in the forest), a Garmin altimeter, and a barometer were present on the platform, they were not directly used by the UAV.
- **Experimental Procedure:**
  Each experimental run followed a defined procedure. First, the necessary software components were launched within a tmux session on the ground. A safety pilot then armed the UAV and performed a manual takeoff to a safe altitude. Once airborne and stable, the RBL algorithm was initiated, commanding the UAV towards a predefined goal. The flight was continuously monitored via visualization in Rviz and observing the UAV. If the UAV approached an obstacle too closely or exhibited unsafe behavior, the safety pilot immediately terminated the autonomous mode, took manual control, and landed the UAV. Between runs, key algorithm parameters, such as those controlling navigation 'aggressivity' (e.g., the scale of cells in the RBL and the weighting function used for path planning), were adjusted based on observations from the previous flight, and the experiment was repeated.

- **Safety Measures:**
  Safety was paramount throughout the experiments. A trained safety pilot maintained visual line-of-sight with the UAV at all times and was prepared to immediately take manual control via the remote controller should any unsafe condition arise.
- **Data Collected:**
  For safety reasons, data logging was initiated only after the manual takeoff was complete. During the autonomous flight part and subsequent landing, all relevant ROS topics (including sensor data, estimated state, RBL cells, centroid) were recorded into rosbag files for detailed post-flight analysis.
- **Challenges Encountered:**
  Several challenges emerged during real-world deployment. A significant set-back involved correctly managing coordinate frame transformations, particularly due to the slight tilt of the top-mounted LiDAR sensor. Ensuring accurate alignment between the LiDAR's point cloud data, the Point-LIO state estimator, and the navigation algorithm's reference frame required careful configuration. Another challenge, which warrants further investigation (discussed in Future Work), related to the Bonxai mapping part. Occasionally, the mapper would map dynamic objects, such as leaves disturbed by propellers, into the static map. Due to the mapping algorithm's configuration, which was set to be conservative about removing potentially static obstacles to ensure safety, these incorrectly mapped 'ghost' obstacles were sometimes not removed even when subsequent scans showed they were no longer present. In some instances, this led to the UAV perceiving itself as surrounded by obstacles (effectively 'locking' itself), requiring the safety pilot to intervene and land.

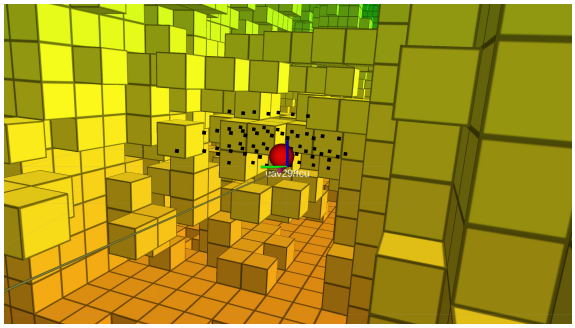


Figure 3.5: Visualization of a moment where the UAV incorrectly mapped leaves as obstacles directly in its path.

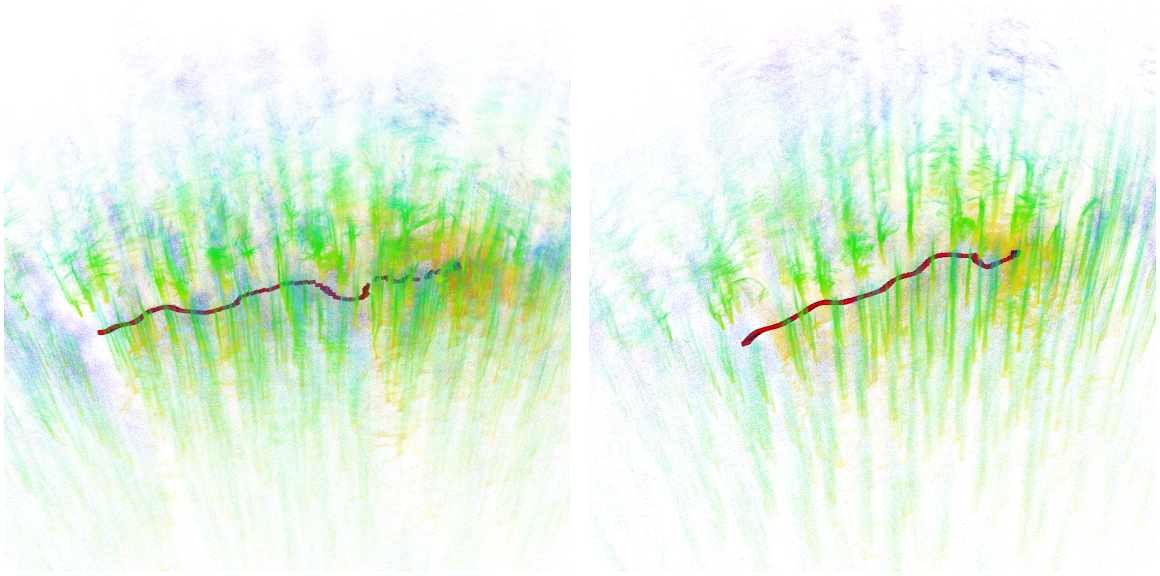
- **Performance in Real-World Experiment**

For the real world experiment I have chosen 2 flights to describe and one fail flight. For each of the flight I uploaded a video One conservative:

Table 3.2: Parameters Used in Experiments

Parameter	Values Conservative	Values Aggressive
Sensing radius (r_s)	2.0 m	2.0 m
Update rate	10 Hz	10 Hz
Encumbrance	0.5 m	0.5 m
$d_1 = d_3 = d_5$	0.5 m	0.5 m
$d_2 = d_4 = d_6$	1.0 m	1.0 m
β_i^D	0.5	0.1
η	0.8	0.8

	L [m]	t [s]	v [m/s]
Conservative Flight	53.84	116.40	0.46
Agressive Flight	35.56	51.64	0.69



(a) (b)

Figure 3.6: Visualization of the path the uav took during the flights a is conservative, b is agressinve.

- **Comparative Analysis**

The real-world flight experiments conducted in the forest environment successfully validated the navigation capabilities of the proposed setup, despite initial setbacks during setup related to odometry and mapping configuration. The UAV generally performed as expected based on simulation results, demonstrating its ability to navigate autonomously between points A and B while avoiding static obstacles.

Performance and Limitations

A primary limitation observed during real-world testing originated from the environmental mapping component, specifically the Bonxai mapping package used. While effective for static elements, the system occasionally struggled with dynamic obstacles, such as moving leaves. These obstacles were sometimes included into the voxel grid map and, due to the map's update policy (configured conservatively for safety), were not always removed immediately even when no longer present. This could lead to situations where the UAV perceived itself as blocked (as shown in the failure case video [3]), requiring the safety pilot to land. Addressing this mapping behavior is crucial for future testing.

Comparison with 2D Baseline

Direct comparison with other state-of-the-art 3D navigation methods is challenging due to the specific nature of the RBL-based approach. However, a meaningful comparison can be done with the original 2D RBL implementation [4]. The key differences lie in the sensing and dimensionality:

- The 2D version relied on a 2D LiDAR for planar obstacle detection and a separate optical distance measurement sensor to maintain a constant flight altitude. Remapping obstacles in 2D is often simpler.
- 3D implementation utilizes a 3D LiDAR for perception in three dimensions and must actively manage altitude based on the perceived environment, treating the ground itself as an obstacle.

A notable behavior observed in the 3D tests was the UAV's tendency to initially increase altitude after takeoff, effectively moving away from the ground obstacle, before potentially descending again as it neared the goal. This behavior highlights the algorithm's capability to manage its vertical position based on the 3D environment. Illustrative examples of successful flights showcasing different parameter settings (aggressive vs. conservative) are provided in [2] and [1].

Future Work

The mapping limitations highlight key areas for future work. Potential solutions include:

- **Dynamic Obstacle Handling:**
  Implementing techniques to segment and filter out dynamic or transient objects from the map representation such as [5].
- **Reactive Navigation:**
  Exploring strategies that rely more on direct sensor input for immediate obstacle avoidance, removing dependence on the map, though this introduces challenges in perceiving areas outside the current sensor view (e.g. behind the UAV).
- **Map Update Logic:**
  Investigating alternative mapping packages or implementing mechanisms like voxel decay instead of mapping (where unoccupied voxels gradually fade if not persistently observed) into the existing framework. While probabilistic mapping should ideally handle this, the developmental stage of the Bonxai package currently limit this capability.

Resolving these mapping challenges, particularly the robust handling of dynamic elements, is a prerequisite for advancing to more complex multi-agent experiments utilizing only onboard lidar sensing.

3.5 Conclusion

This chapter detailed the implementation and real-world validation of the (RBL) algorithm, adapted for autonomous UAV navigation within a cluttered, GNSS-denied forest environment using onboard 3D LiDAR sensing. The primary objective was to implement RBL principles with real-time perception, state estimation (Point-LIO), and mapping (Bonxai) to achieve reliable point-to-point navigation.

Key technical contributions included adapting the RBL algorithm's cell partitioning to directly utilize voxelized map data created from processed LiDAR point clouds. Modifications were also introduced to effectively handle the practical limitations of a single LiDAR sensor with a restricted vertical field of view, ensuring safe convergence by constraining movement based on actively sensed area while still leveraging map information. An alternative approach using surface reconstruction via mesh generation was investigated but considered unsuitable due to computational complexity on the UAV onboard computer.

The proposed solution effectiveness was demonstrated through both simulation and real-world experiments conducted in a forest. The UAV successfully navigated between designated start and goal points, avoiding static obstacles like trees and adapting its altitude based on the LiDAR's perception.

Despite the overall success, the experiments identified a key limitation related to the Bonxai mapping package's handling of dynamic environmental elements, such as moving leaves, which occasionally led to navigation deadlocks. This emphasises the importance of robust mapping with dynamic obstacles.

In conclusion, this chapter successfully demonstrated the practical application and feasibility of using a 3D RBL algorithm integrated with LiDAR sensing for autonomous UAV navigation in a challenging forest setting. This capability can be observed in these videos [2], [1].

4 Conclusion

Summarize the achieved results. Can be similar as an abstract or an introduction, however, it should be written in past tense.

5 References

[1] M. Kamler, *Drone forest navigation - conservative parameters (slow and safe flight)*, 2025. [Online]. Available: https://www.youtube.com/watch?v=AJPk0yVCPUo&ab_channel=MichalKamler.

[2] ——, *Drone forest navigation – aggressive parameters (best performance)*, 2025. [Online]. Available: https://www.youtube.com/watch?v=DFt222gnA_w&ab_channel=MichalKamler.

[3] ——, *Drone forest navigation – mapping failure (remapping error)*, 2025. [Online]. Available: https://www.youtube.com/watch?v=JVmW0qfwP3c&ab_channel=MichalKamler.

[4] M. Boldrer, A. Serra-Gomez, L. Lyons, V. Kratky, J. Alonso-Mora, and L. Ferranti, "Rule-based lloyd algorithm for multi-robot motion planning and control with safety and convergence guarantees," *arxiv*, 2024. [Online]. Available: https://arxiv.org/abs/2310.19511.

[5] Y. Jia, T. Wang, X. Chen, and S. Shao, *Trlo: An efficient lidar odometry with 3d dynamic object tracking and removal*, 2024. arXiv: 2410.13240 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2410.13240.

[6] D. Mezey, R. Bastien, Y. Zheng, N. McKee, D. Stoll, H. Hamann, and P. Romanczuk, *Purely vision-based collective movement of robots*, 2024. arXiv: 2406.17106. [Online]. Available: https://arxiv.org/abs/2406.17106.

[7] C. i. P. Multi-Robot Systems Group, *Mrs uav system*, Accessed: 2025-03-10, 2024. [Online]. Available: https://github.com/ctu-mrs/mrs_uav_system.

[8] V. Walter and M. Group, *Uvdar_multirobot_simulator*, https://github.com/ctu-mrs/uvdar_multirobot_simulator, Accessed: 2025-04-24, 2024.

[9] S. Debnath, M. Paul, and T. Debnath, "Applications of lidar in agriculture and future research directions," *Journal of Imaging*, vol. 9, no. 3, 2023, ISSN: 2313-433X. DOI: 10.3390/jimaging9030057. [Online]. Available: https://www.mdpi.com/2313-433X/9/3/57.

[10] B. Ertl, *3d voronoi mesh of 25 random points with 0.3 opacity and points*, Own work, licensed under CC BY-SA 4.0, Accessed: 2025-02-27, 2023. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=38534275.

[11] D. He, W. Xu, N. Chen, F. Kong, C. Yuan, and F. Zhang, "Point-lio: Robust high-bandwidth light detection and ranging inertial odometry," *Advanced Intelligent Systems*, vol. 5, no. 7, p. 2 200 459, 2023. DOI: https://doi.org/10.1002/aisy.202200459. eprint: https://advanced.onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202200459. [Online]. Available: https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202200459.

[12] Livox Technology Company Limited, *Mid-360 LiDAR Sensor*, Accessed: 2025-05-02, 2023. [Online]. Available: https://www.livoxtech.com/mid-360.

[13] F. e. České vysoké učení technické v Praze, *Detekce kůrovce, obrana proti nepřátelskému převzetí dronů a roje úzce spolupracujících vzdušných robotů.* 2023. [Online]. Available: https://fel.cvut.cz/import/tz/2023-0426-tz_fel_cvut_mrs_temesvar.pdf.

[14] WalkingRadiance, *3d voronoi mesh of 25 random points with 0.3 opacity and points*, Own work, licensed under CC BY-SA 4.0, Accessed: 2025-02-27, 2023. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=127768757.

[15] A. Ahmad, D. Bonilla Licea, G. Silano, T. Báča, and M. Saska, "Pacnav: A collective navigation approach for uav swarms deprived of communication and external localization," *Bioinspiration & Biomimetics*, no. 6, 2022. DOI: 10.1088/1748-3190/ac98e6. [Online]. Available: http://dx.doi.org/10.1088/1748-3190/ac98e6.

[16] R. Jin, Y. Wang, Z. Gao, X. Niu, L.-T. Hsu, and J. Liu, "Dynavig: Monocular vision/ins/gnss integrated navigation and object tracking for agv in dynamic scenes," 2022. arXiv: 2211.14478 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2211.14478.

[17] N. Lopac, I. Jurdana, A. Brnelić, and T. Krljan, "Application of laser systems for detection and ranging in the modern road transportation and maritime sector," *Sensors*, vol. 22, no. 16, 2022, ISSN: 1424-8220. DOI: 10.3390/s22165946. [Online]. Available: https://www.mdpi.com/1424-8220/22/16/5946.

[18] Z. Liu, A. Amini, S. Zhu, S. Karaman, S. Han, and D. Rus, "Efficient and robust lidar-based end-to-end navigation," 2021. arXiv: 2105.09932 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2105.09932.

[19] T. Takebayashi, R. Miyagusuku, and K. Ozaki, "Development of magnetic-based navigation by constructing maps using machine learning for autonomous mobile robots in real environments," *Sensors*, vol. 21, no. 12, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/12/3972.

[20] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, 50–61, Jul. 2020, ISSN: 1558-0792. DOI: 10.1109/msp.2020.2973615. [Online]. Available: http://dx.doi.org/10.1109/MSP.2020.2973615.

[21] V. Walter, M. Saska, and A. Franchi, "Fast mutual relative localization of uavs using ultraviolet led markers," *2018 International Conference on Unmanned Aircraft System (ICUAS 2018)*, 2018. [Online]. Available: https://dronument.cz/sites/default/files/uploaded/mutual_localizations_leds_naki.pdf.

[22] *F4f robofly swarm*, Fly4Future. [Online]. Available: https://shop.fly4future.com/product/f4f-robofly-swarm/.

[23] Fly4Future, *Drones for inspection of power lines*. [Online]. Available: https://fly4future.com/custom-drones/drones-for-inspection-of-power-lines/.

[24] B. S. Radu B. Rusu, *The voxelgrid class, point cloud library*, Accessed: 2025-03-28. [Online]. Available: https://pointclouds.org/documentation/classpcl_1_1_voxel_grid.html.

A Experimental Results

- $N = 5$ **Circular Formation:**

	SR [%]	$\overline{L}$ [m]	$\bar{t}$ [s]	$\bar{t}_{\max}$ [s]	$\overline{v}$ [m/s]
RBL 2D	100.00	21.06 ± 0.10	25.15 ± 0.21	25.15 ± 0.19	0.83 ± 0.01
RBL 3D	100.00	20.77 ± 0.29	26.04 ± 0.51	26.79 ± 0.27	0.79 ± 0.02
RBL $3D_{\text{clipped}}$	100.00	20.60 ± 0.24	26.73 ± 0.47	27.39 ± 0.28	0.77 ± 0.02
RBL $3D_z$	100.00	20.97 ± 0.52	25.54 ± 0.97	26.72 ± 0.60	0.81 ± 0.03

- $N = 10$ **Circular Formation:**

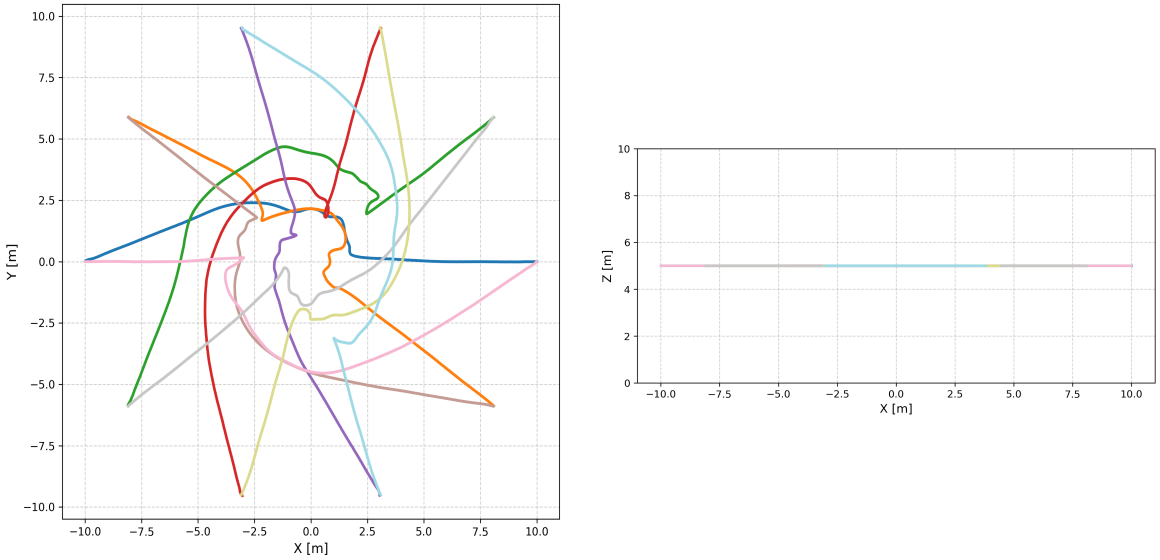
	SR [%]	$\overline{L}$ [m]	$\bar{t}$ [s]	$\bar{t}_{\max}$ [s]	$\overline{v}$ [m/s]
RBL 2D	100.00	22.95 ± 1.64	30.79 ± 2.28	34.73 ± 0.77	0.74 ± 0.07
RBL 3D	100.00	22.22 ± 0.89	30.05 ± 2.61	34.39 ± 4.19	0.73 ± 0.05
RBL $3D_{\text{clipped}}$	100.00	22.31 ± 0.69	30.22 ± 1.83	33.22 ± 0.93	0.73 ± 0.04
RBL $3D_z$	100.00	22.38 ± 0.88	28.80 ± 2.30	32.35 ± 1.25	0.77 ± 0.05

- $N = 15$ **Circular Formation:**

	SR [%]	$\overline{L}$ [m]	$\bar{t}$ [s]	$\bar{t}_{\max}$ [s]	$\overline{v}$ [m/s]
RBL 2D	100.00	23.56 ± 1.75	33.89 ± 2.48	38.75 ± 1.45	0.69 ± 0.06
RBL 3D	100.00	22.36 ± 0.97	30.65 ± 3.07	35.77 ± 3.72	0.72 ± 0.07
RBL $3D_{\text{clipped}}$	100.00	22.32 ± 0.81	30.69 ± 2.61	34.50 ± 0.47	0.72 ± 0.06
RBL $3D_z$	100.00	22.64 ± 0.95	29.67 ± 2.54	34.27 ± 0.88	0.76 ± 0.06

- $N = 10$ **Spherical Formation:**

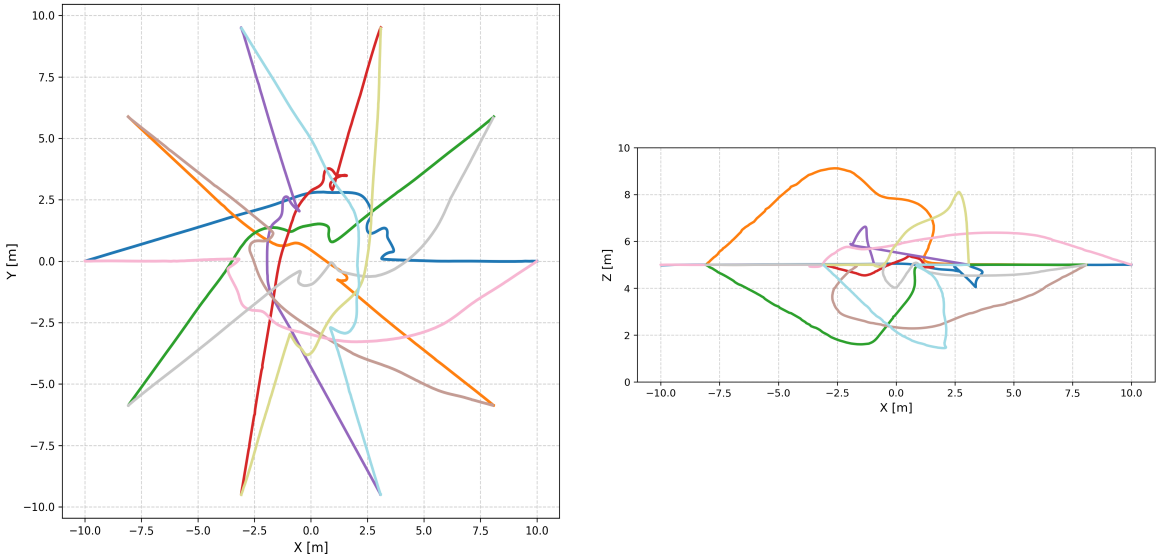
	SR [%]	$\overline{L}$ [m]	$\bar{t}$ [s]	$\bar{t}_{\max}$ [s]	$\overline{v}$ [m/s]
RBL 3D	100.00	14.32 ± 1.52	27.76 ± 3.06	32.48 ± 2.26	0.51 ± 0.05
RBL $3D_z$	100.00	14.06 ± 0.97	27.17 ± 2.66	31.86 ± 1.24	0.55 ± 0.06



(a) Top-Down View (X-Y)

(b) Side View (X-Z)

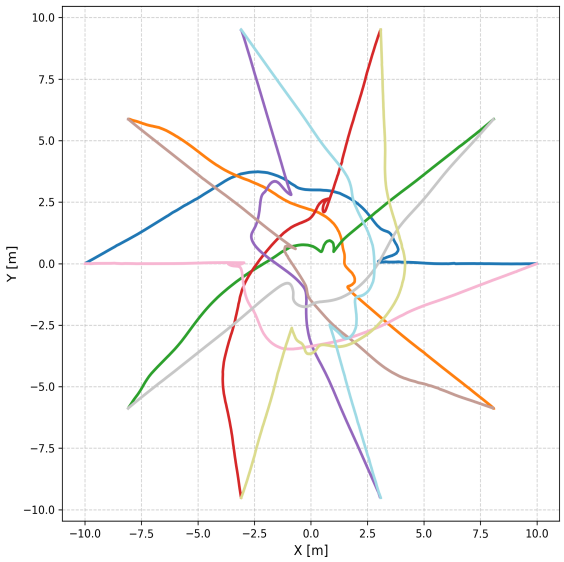
Figure A.1: Trajectories in a 2D circular crossing.



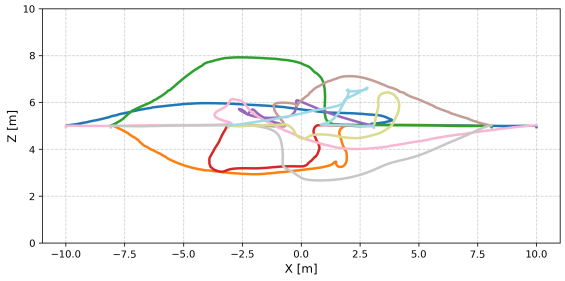
(a) Top-Down View (X-Y)

(b) Side View (X-Z)

Figure A.2: Trajectories in a 3D circular crossing.

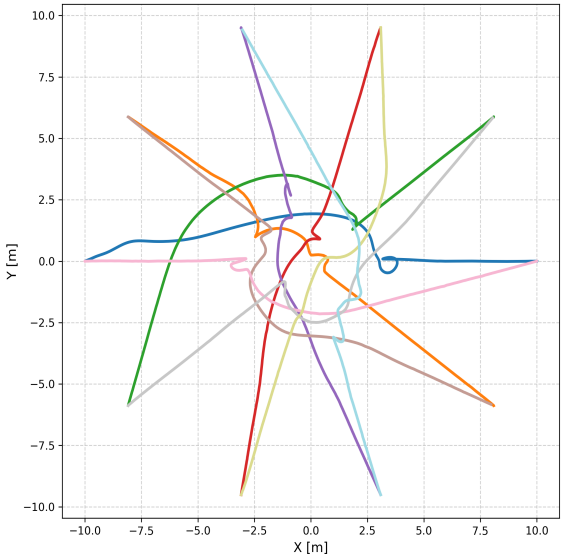


(a) Top-Down View (X-Y)

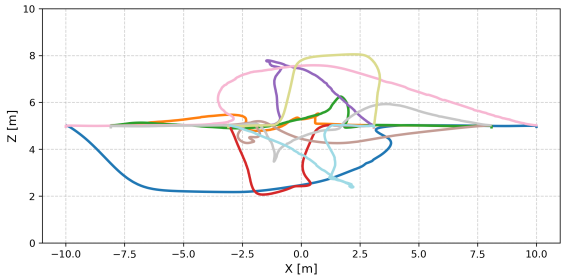


(b) Side View (X-Z)

Figure A.3: Effect of Z-axis clipping on the circular crossing.

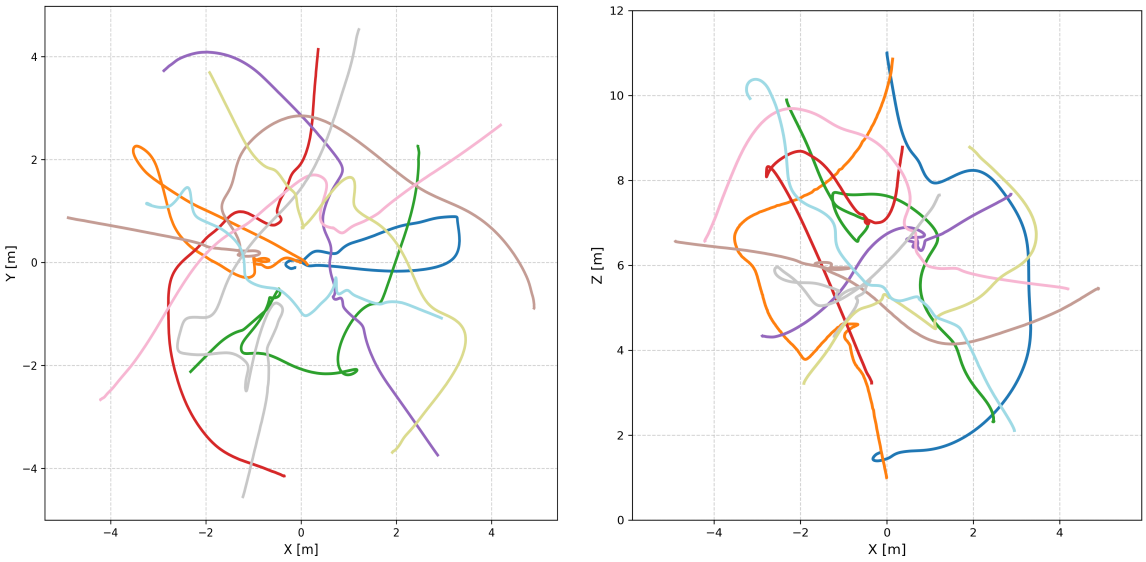


(a) Top-Down View (X-Y)



(b) Side View (X-Z)

Figure A.4: Effect of the Z-axis rule on the circular crossing.



(a) Top-Down View (X-Y)

(b) Side View (X-Z)

Figure A.5: Trajectories in a 3D spherical crossing.