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: Ing. 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	prof. Mgr. Petr Páta, Ph.D.
Head of department’s signature	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

LiDAR	Light Detection and Ranging
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
RBL	Rule-based Lloyd Algorithm
ToF	Time-of-Flight
PCL	Point Cloud Library

Contents

1 Introduction **1**
1.1 Related Works . . . 1
1.2 Problem Statement . . . 1
1.3 Contributions . . . 1
1.4 Mathematical Notation . . . 1

2 Extension of the Rule-Based Lloyd Algorithm to 3D **2**
2.1 Introduction . . . 2
2.2 Basic principles of Rule-Based Lloyd Algorithm . . . 2
2.3 Extension of the RBL algorithm to 3D . . . 5
2.4 Simulation and Results Analysis . . . 8
2.5 Summary and Key Insights . . . 11

3 UAV Navigation Using the RBL Algorithm and LiDAR Sensing **12**
3.1 Introduction . . . 12
3.2 LiDAR-Based Perception and Point Cloud Processing . . . 12
3.3 Using the found triangles for cell A partition . . . 14
3.4 Implementation and Integration on UAV . . . 15
3.5 Simulation and Experimental Results . . . 16
3.6 Conclusion . . . 16
3.7 Introduction . . . 16
3.8 3D LiDAR Sensor Model and Simulation Setup . . . 16
3.9 Object Detection and Approximation . . . 16
3.10 Integration with RBL Algorithm . . . 16
3.11 Simulation Results . . . 17
3.12 Discussion . . . 17
3.13 Summary . . . 17

4 Researching on possible enhancements of the algorithm for example using neural networks or learning-based techniques. **18**

5 Conclusion **19**

6 References **20**

A Appendix A **21**

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$^{\text{th}}$ 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$^{\text{st}}$, 2$^{\text{nd}}$, 3$^{\text{rd}}$, and 4$^{\text{th}}$ 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 relies on global positioning data together with onboard sensors that provide information about the surrounding environment. So far, the algorithm has mainly been tested in 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 3D navigation, where altitude and sensor orientation also play a role.

Problem Statement

The primary challenge that was addressed involved the extension of an existing 2D algorithm into a 3D space. To achieve improved results in this higher dimension, new rules controling vertical exploration were introduced. The subsequent sections of this chapter detail the fundamental principles of the RBL algorithm, with specific attention devoted to its application in both 2D and 3D environments. The distinct rules governing the algorithm's behavior in these two dimensions are described separately to provide clarity. 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.

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 [1], 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.

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. It relies on global positioning data, such as GPS or alternative methods for obtaining global coordinates, combined with sensor inputs that provide information about the agent's 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 [7], magnetic [10], or LiDAR-based navigation [9]. Additionally, 2D navigation is widely employed 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

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 is knows of 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 $||\mathbf{p}_i - \mathbf{p}_j|| \leq 2r_{s,i}$, where $r_{s,i}$ is denoted as half of the sensing radius of the i-th agent. 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 minimize the coverage cost function, which accounts for the distribution of agents and obstacles over the environment. This function is expressed as:

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

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, or 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 $\tilde{\mathbf{p}}_j = \mathbf{p}_j + 2(\Delta_{ij} - \frac{\|\mathbf{p}_i - \mathbf{p}_j\|}{2})\frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|}$. 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:

- **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 β_{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**

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

  where $\bar{\mathbf{c}}_{\mathcal{A}_i}$ 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 solely 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.

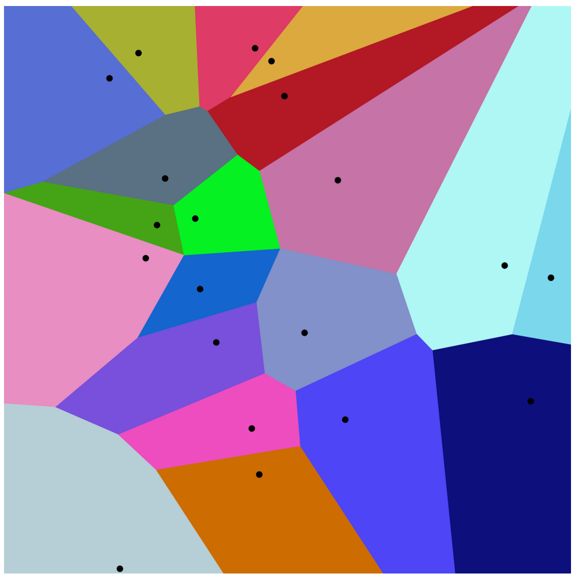
2.3 Extension of the RBL algorithm to 3D

Motivation for 3D Extension

In many practical applications, agents must operate in three-dimensional spaces, considering not only horizontal movement but also vertical positioning. 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 Unmanned Aerial Vehicle (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

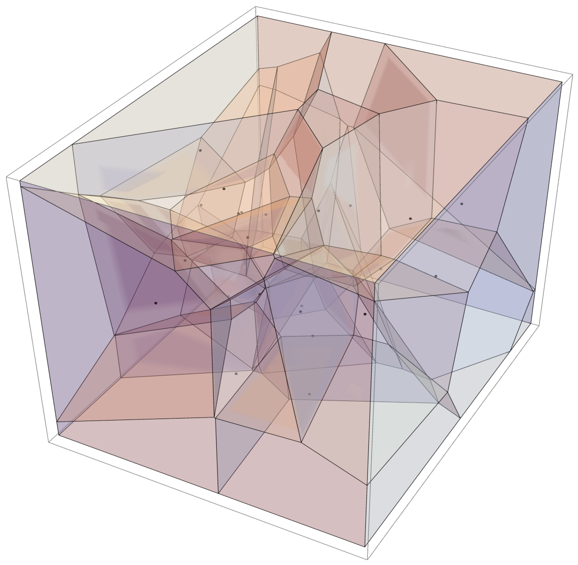

(b) Euclidean Voronoi Diagram in 3D

Figure 2.1: (a) an example of 20 Voronoi cells in 2D [5] (b) 25 Voronoi cells in 3D [6]

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**

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

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 $\max_z$:

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

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 direc 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 $D_{influence}$ value in the range [-1, 1]:

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

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 $D_{influence}$ is combined with the relative vertical displacement of the agent and $\mathbf{c}_{\mathcal{S}_i}$ Weighted average is used to combine them:

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

where w_1 w_2 are weighting factors. The resulting combined influence $C_{influence}$ 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 phi_i can be constructed as follows:

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

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).

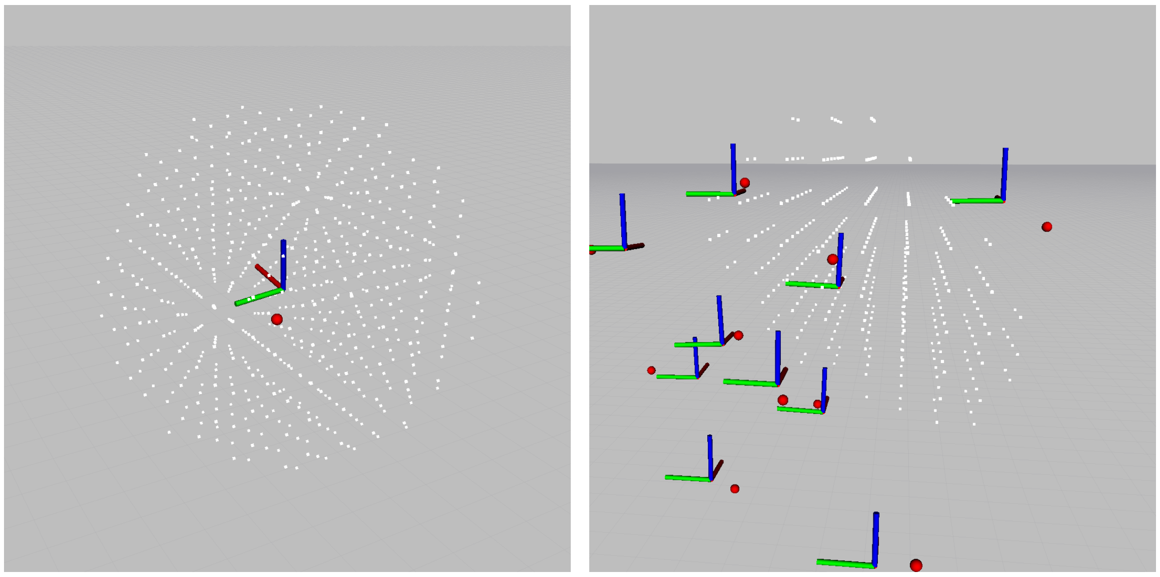
(iv) **Agent movement**
Finally, the agent is directed towards a centroid location $c_{\mathcal{A}}$.

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 [2]. 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 [12] and [3]. 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 eas conducted with N = 10, also repeated

10 times. After each UAV was flown from the ground to its designated starting position, the RBL algorithm was initiated.

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 $\bar{v}$ [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)	3.5 m
Update rate	10 Hz
Encumbrance	0.5 m
$d_1 = d_3 = d_5$	0.5 m
$d_2 = d_4 = d_6$	1.0 m
d_7	0.2
β_i^D	1.5
η	0.9
min_z	1.0 m
max_z	10.0 m

Experimental Results

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

	SR [%]	$\overline{L}$ [m]	$\bar{t}$ [s]	$\bar{t}_{\max}$ [s]	$\bar{v}$ [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]	$\bar{v}$ [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]	$\bar{t}_{\text{max}}$ [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]	$\bar{t}_{\text{max}}$ [s]	$\overline{v}$ [m/s]
RBL 2D	100.00	00.00 ± 0.00	00.00 ± 0.00	00.00 ± 0.00	0.00 ± 0.00
RBL 3D	100.00	00.00 ± 0.00	00.00 ± 0.00	00.00 ± 0.00	0.00 ± 0.00
RBL 3D$_{\text{clipped}}$	100.00	00.00 ± 0.00	00.00 ± 0.00	00.00 ± 0.00	0.00 ± 0.00
RBL 3D$_z$	100.00	00.00 ± 0.00	00.00 ± 0.00	00.00 ± 0.00	0.00 ± 0.00

2.5 Summary and Key Insights

Recap of modifications. Faced challenges and solutions applied

3 UAV Navigation Using the RBL Algorithm and LiDAR Sensing

3.1 Introduction

TODO

Motivation

The importance of UAV navigation in environments with obstacles

Problem Statement

Challenges of UAV movement using LiDAR-based sensing

Objectives

Implementing RBL with LiDAR for obstacle avoidance and path planning

Chapter Overview

Summary of what this chapter covers

3.2 LiDAR-Based Perception and Point Cloud Processing

Overview of LiDAR for UAV Navigation

Light Detection and Ranging (LiDAR) is a crucial sensing technology widely used in applications such as SLAM [14], autonomous vehicles [11], UAVs TODOcite, and precision agriculture [4]. 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 [8].

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) [13]. 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. Each point in the downsampled cloud is converted to an Eigen 3D vector for easy mathematical operations. The Euclidean distance between each point in pointcloud and the UAV's position $\mathbf{p}_i$ is computed. Points within a predefined encumbrance radius around the UAV are discarded to remove self-detected points.

TODO add images of not downsampled and downsampled and close up on the uav

The resulting filtered point cloud contains only relevant environmental features while eliminating unnecessary points, improving efficiency in decision-making processes.

Surface Reconstruction and Triangulation

To obtain a continuous surface representation from the point cloud, normal estimation and triangulation are performed. This process involves estimating surface normals, combining them with point positions, and applying a triangulation algorithm to generate a polygonal mesh.

- **Normal Estimation** – Surface normals are estimated using the Point Cloud Library (PCL) TODOcite. A k-d tree is employed to efficiently find neighboring points, and the normal at each point is computed based on its local neighborhood. This provides information about the surface curvature and orientation.
- **Point Cloud and Normal Merging** – Once the normals are estimated, they are combined with the original point cloud to create a representation that includes both spatial position and surface orientation.
- **Triangulation Using Greedy Projection** –The point cloud is converted into a polygonal mesh using the **Greedy Projection Triangulation (GP3)** method. This algorithm forms triangles between neighboring points while enforcing constraints on edge length, surface angles, and normal consistency. A k-d tree is used to accelerate the search for nearest neighbors, ensuring efficient mesh generation.
- **Results** – The output is a polygonal mesh that approximates the underlying surface of the point cloud. This mesh can be used for visualization, collision detection, or further geometric processing.

Greedy Projection Triangulation (GP3), pcl lib and parametres Generating triangular mesh - How GP3 constructs a surface from point cloud data

3.3 Using the found triangles for cell A partition

TODO

Overview

the next step is to utilize these triangles to partition the navigation space, specifically modifying cell A for the RBL algorithm. This process ensures that the UAV has a structured representation of obstacles and free space.

Finding the Closest Point on a Triangle

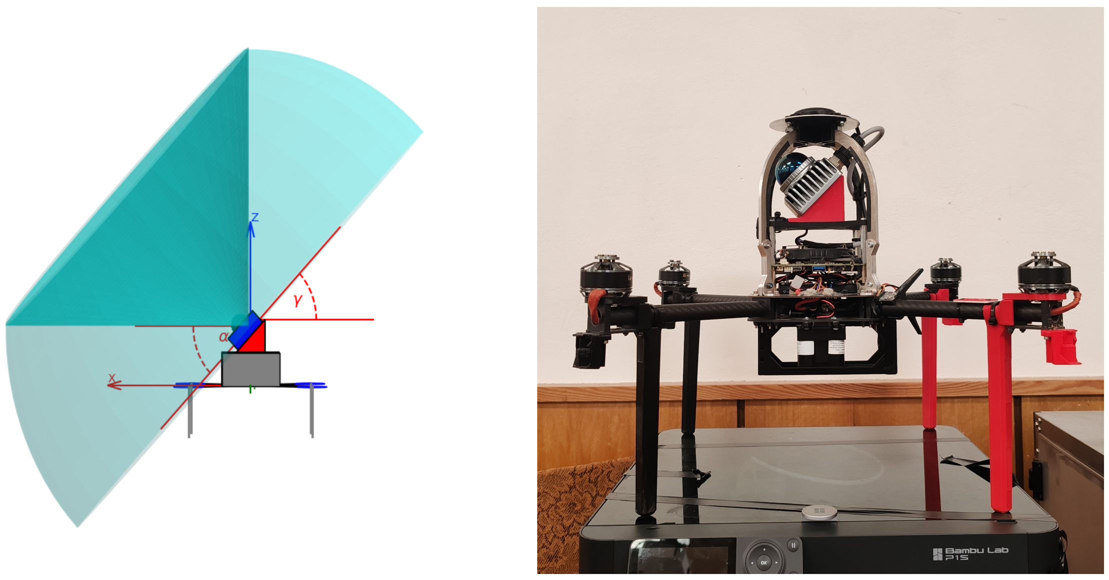
Plane Calculation for Slicing Cell A

Integration with the RBL Algorithm

3.4 Implementation and Integration on UAV

TODO

Challanges in Integration



(a) UAV model with visualized LiDAR mounting parameters. (b) UAV used in the experiment with mounted LiDAR

Figure 3.1: 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.

For modification of cell A 2 planes are defined using real knowledge of how LiDAR is mounted on the uav and livox vertical fov Let $\mathbf{e}_z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ be the unit vector along the z-axis, $R_{\text{off}} \in SO(3)$ the offset rotation, and $R_{\text{rpy}} \in SO(3)$ the roll-pitch-yaw rotation. α_1, α_2, α_3

Then the resulting vector is given by:

$$\mathbf{n} = R_{\text{rpy}} \cdot R_{\text{off}} \cdot \mathbf{e}_z$$

Modified cell S:

$$\mathcal{S}_i = \tag{3.1}$$

livox - 360 * 60 deg - account for centroid calculation

Software and Hardware setup

LiDAR sensor config, uav platform specifications

3.5 Simulation and Experimental Results

TODO

Simulation Setup

virtual environment modeling, tools used

Performance in Simulated Environments

Success rates, obstacle avoidance, efficiency metrics - same as in the rbl

Real-World Experiments in a Forest Environment

Comparative Analysis

Simulation vs. real-world performance

3.6 Conclusion

TODO Summary of contributions and some directions for future research

REDO after this

3.7 Introduction

Motivation for using 3d lidar and challenges

3.8 3D LiDAR Sensor Model and Simulation Setup

Overview of LiDAR used. Simulation configuration

3.9 Object Detection and Approximation

Methods for extracting objects from LiDAR point clouds. Approximating detected objects with simple shapes. Handling noisy or incomplete data

3D Surface Approximation from Point Clouds ← This one seems promising. I would like to try this.

Alpha Shape: A Generalization of the Convex Hull

3.10 Integration with RBL Algorithm

Modifications to ensure safe navigation and how approximated objects influence Voronoi cells

3.11 Simulation Results

Example scenarios - in rviz with drone and also in real life - me holding a branch with leafs or something like that

3.12 Discussion

Limitations and possible improvements

3.13 Summary

4 Researching on possible enhancements of the algorithm for example using neural networks or learning-based techniques.

tune the paramaters of RBL to suit 3D

I will have to read something first, but what I found is that I could try object detection and classification using models like PointNet, PointNet++ or VoxelNet

Pointcloud denoising

Segmentation of LiDAR data. RangeNet++, SalsaNext to differentiate between terrain trees and free space

NN approaches for approximating point clouds with simple convex shapes:

CvxNet: Learnable Convex Decomposition by Deng et al. (CVPR 2020)

Label-Efficient Learning on Point Clouds using Approximate Convex Decompositions by Gadelha et al. (Arxiv 2020)

Region Segmentation via Deep Learning and Convex Optimization by Sonntag and Morgenshtern (Arxiv 2019)

Point Density-Aware Voxels for LiDAR 3D Object Detection

5 Conclusion

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

6 References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] A. TODO, *Point-lio mrs*. [Online]. Available: https://github.com/ctu-mrs/Point-LIO.

A Appendix A