Multi-robot coordination
Autonomous exploration of gallery networks

Engineering Graduation Project
Seatech 3A - MOCA

Auteur :
Fabien MATHÉ

Enseignant :
M. Mehmet ERSOY

Abstract

This work focuses on cavity exploration using drone swarms, addressing issues ranging from trajectory planning to communication between robots. A new trajectory planning method has been developed, based on the successive division of the shortest path leading to the target point, represented by a straight line. This algorithm has proven to be particularly suited for quick and dynamic adaptation of the robots' trajectories. Additionally, an approach based on a master robot has been developed for coordination between the robots, combining low data transfer and basic communication error management using the ARQ method. Finally, all these methods were tested on a simulator developed in *Python* as part of this research.

Keywords:

Robotics, Path Planning, Inter-Robots Communication, Multi-robots, Cave Exploration

Résumé

Ce travail porte sur l'exploration de cavités par essaim de drones, en abordant des problématiques allant de la planification de trajectoire à la communication entre les robots. Une nouvelle méthode de planification de trajectoire a été conçue, reposant sur un découpage successif du chemin le plus court menant au point cible, qui est représenté par une droite. Cet algorithme s'est révélé particulièrement adapté à une adaptation rapide et dynamique des trajectoires des robots. Par ailleurs, une approche basée sur un robot maître a été développée pour la coordination entre les robots, combinant un faible transfert de données et une gestion des erreurs de communication par la méthode ARQ. Enfin, toutes ces méthodes ont été testées sur un simulateur développé en *Python* dans le cadre de cette recherche.

Mots-clés:

Robotique, Planification de trajectoire, Communication Inter-Robots, Multi-Robots, Exploration de cavités

Acknowledgement

I would like to take this opportunity in this final project report to express my deep gratitude to all my professors who have supported and guided me throughout my journey.

I would also like to thank Mr. Frédéric Golay and Mr. Cédric Galusinsky, the two course directors of the MOCA program, for the quality of their teaching, their expertise, and their valuable advice over the past two years.

Finally, I am especially grateful to Mr. Mehmet Ersoy, who has always been present and available to assist me with all the projects I have undertaken during these three years at SeaTech. Whether it was for HydroSharks, my personal research, my student projects, or soon for my PhD, his support has been invaluable.

Contents

Introduction

The motivation behind this work lies in the pursuit of the unknown and a deeper understanding of our planet. Underground cavities remain among the least explored places on Earth due to their inaccessibility. The challenges inherent in such exploration are numerous, ranging from communication difficulties to trajectory planning in confined spaces, which are often steeply inclined and require advanced robotic design for navigation. Whether in the depths of the oceans or beneath the Earth's surface, these hidden environments remain largely unexplored. The potential discovery of unknown biological entities, as well as ancient geological formations untouched by human activity, adds further significance to this research. Indeed, caves are among the last places on Earth that have remained largely free from human influence and pollution.

This work focuses on developing exploration algorithms for multi-robot cave navigation. The robots used in this study are two-wheeled, differentially driven systems. A key challenge is ensuring full autonomy, meaning the robots must operate without external control while effectively communicating with each other.

To support this development, we will first explore the theoretical aspects of trajectory optimization in robotics. Given the challenges of navigating complex and uneven terrain, robust and efficient algorithms are essential. However, to simplify the initial approach, the robots will be tested on a flat and smooth surface.

At the core of this project, I will propose an algorithm that enables communication between robots without relying on an external master or centralized synchronization. This approach aims to enhance coordination and decision-making in a fully decentralized manner, making the system more adaptable to real-world exploration scenarios.

A crucial component of this project is the development of a Python-based simulator. This simulator will allow real-time testing of inter-robot communication and algorithm performance, providing a valuable platform for exploring innovative and interdisciplinary techniques.

I State of the art

One of the key reference works used in this study is S. M. LaValle's book, Planning Algorithms [1], for which I am particularly grateful for his commitment to making his work accessible to the public (https://lavalle.pl/).

I.1 Multi-Robot Exploration of Cavities

The exploration of cavities is essential for underground mapping, scientific research, and emergency interventions. However, it poses unique challenges for autonomous robotics.[2, 3] Multi-robot exploration of such environments is a growing research field with applications in various domains, including space exploration, cave exploration, urban search and rescue missions, and mining operations.[4, 5] These environments are often very complex, presenting significant challenges for mission planning, such as unpredictable obstacles, inaccessible areas for mobile robots, and the absence of GPS signals. Multi-robot cooperation can help overcome these difficulties by leveraging distributed sensing and coordinated decision-making.

I.1.1 Challenges in Multi-Robot Cave Exploration

Exploring cavities with multiple robots involves several technical challenges. Two fundamental aspects are managing cooperation between robots and handling information in complex environments. The technical difficulties encountered are numerous.[3]

A particularly successful research field of multi-robot exploration has been in extraterrestrial environments. On earth, the DARPA Subterranean Challenge, launched in 2019, tested autonomous robot teams in complex underground environments, highlighting challenges related to mapping, navigation, and search under severe communication constraints. This challenge further increased interest in autonomous supervision approaches for groups of robots operating in tunnels and mines.[6]

Several robotic approaches have been developed to tackle these challenges. Some methods involve hybrid robotic teams consisting of wheeled or legged robots combined with aerial drones.[7] In other cases, solely using flying drones has proven sufficient for exploring and mapping cavities.[8, 9, 10, 11]

I.1.2 Underwater Cave Exploration

Underwater cave exploration is another area of active research. Water provides a stable medium, reducing the difficulty of maintaining position compared to aerial exploration. However, underwater exploration remains highly challenging due to complex communication constraints and the difficulty of localizing robots relative to external references. Ongoing research focuses on semantic segmentation for underwater cave exploration to enhance perception and navigation capabilities.[12, 13]

I.2 Path Planning for a Mechatronic System

Path planning for a mechatronic system is the foundation of all autonomous mobile systems, whether they involve drones or handling and assembly arms. The principle of path or trajectory planning is to determine a solution —a path or a trajectory— connecting a starting point to a target point.

The distinction between path planning and trajectory planning lies in the fact that a planned path is not necessarily feasible for a robot. Indeed, path planning does not always take into account the physical or kinematic constraints of the mobile robot.

Obstacle avoidance is an essential constraint in both approaches. Obstacles define inaccessible areas, and as we will see later, their nature—mobile or static—largely determines the method to be used to solve the planning problem.

Among deterministic methods, there is a wide variety of approaches mainly based on three different strategies: graph-based methods, cellular decomposition methods, and potential field methods [14, 15].

Graph-based methods consist of constructing a map of feasible paths by considering the obstacles in the scene. Among these graph-based methods, four different types can be distinguished: visibility graphs [16], Voronoi diagrams [17], and the Silhouette method [15].

Methods associated with cellular decomposition consist of dividing the robot's free space into simple regions, called cells, where it is easy to generate a path between two configurations. A graph representing the adjacency relationships between the cells is then constructed and explored [14, 18, 19, 20].

Another method relies on a fine subdivision of the space to identify free zones. The potential field method is based on this idea by defining potentials that represent attractive forces directed toward the target coordinates and repulsive forces corresponding, for example, to obstacles. The path is then determined by following the opposite of the gradient of the total computed potential [14, 21].

Alternative approaches were developed in the 1990s and 2000s, particularly stochastic path planning methods such as Random Path Planners (RPP) and Probabilistic Roadmap Planners (PRM). These methods involve randomly sampling the space to create a graph of possible paths (*roadmap*) within this space [22, 23, 24].

One of the main advantages of this path planning method is that it does not depend on the structure of the space, the number of obstacles, or their arrangement. Once the *roadmap* is created, it is sufficient to use a graph-based path search method to find the path leading to the target point [25].

I.3   Inter-Robot Communication in Cave Environments

Inter-robot communication (IRC) in cave environments poses challenges due to confined spaces, obstacles, and signal disruptions. IRC methods vary based on technology and communication strategies.

Traditional approaches rely on wireless networks, such as radio frequency (RF) communication, but suffer from signal attenuation in caves. Ad hoc methods, like mesh networks, allow robots to relay data dynamically, extending coverage despite obstacles [26]. The Autonomous and Collaborative High-Bandwidth Operations with Radio Droppables (ACHORD) framework enhances multi-robot coordination by adapting to intermittent connectivity [26].

Acoustic and ultrasonic communication offer robust short-range alternatives, while optical meth-

ods (visible light or infrared) provide interference-resistant, high-precision transmissions [27]. Techniques like real-time signal prediction improve communication-aware exploration [28].

Cooperative protocols optimize information flow through adaptive routing and dynamic network partitioning. Opportunistic communication, based on intermittent data exchanges, is particularly useful in environments with connectivity constraints [29]. Large-scale cave exploration studies highlight the importance of maintaining reliable links for coordination [11].

A major challenge remains optimizing bandwidth, resource management, and interference control. Hybrid systems combining multiple technologies enhance resilience in dynamic environments [27]. The ACHORD framework integrates network design with high-level decision-making to improve subterranean communication [26].

II   Notations

- X Coordinates vector of the robot, $\begin{pmatrix} x \\ y \end{pmatrix}$

- X_I Initial robot coordinates

- X_{WP} Waypoint coordinates

- $\dot{X}$ Velocity vector, $\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}$

- ω_L Left wheel rotation speed

- ω_R Right wheel rotation speed

- ω Wheels rotation speed vector,

- ω_{max} Maximal wheels rotation speed, $\begin{pmatrix} \omega_L \\ \omega_R \end{pmatrix}$

- θ Heading of the robot

- $\dot{\theta}$ Angular speed of the robot

- r Wheels radius

- d Distance between the two wheels

- t Time

- T End time

- T_{WP} Time at which the robot reached the waypoint

III   Path planning

III.1   Global path theory

In this section, we introduce the preliminary for mathematical optimization of the path. Several challenging problems arise, which are described at the end of this section.

we consider the theoretical path that the robot should follow in a space with obstacles, without taking into account the feasibility constraints imposed by the robot's physical limitations. This theoretical path is derived based on the shortest distance to the target while avoiding obstacles.

Let us define the following variables:

- $\Omega \in \mathbb{R}^2$: the world

- $\mathbf{X}(t)$: the coordinates of the robot at time t in this space

- $FV(t)$: the field of vision of the robot at time t

- W: the set of all walls of the map represented by segments, $W = \{ \text{WallSegment}(\Omega) \}$

- $R(t, \theta)$: the minimale distance between the robot and all the intersection points between a segment starting from the robot and the map walls.

- R_{max}: the radius of the field of vision

- $\mathbf{M}(t, \theta)$: the first intersection point between a segment and the walls

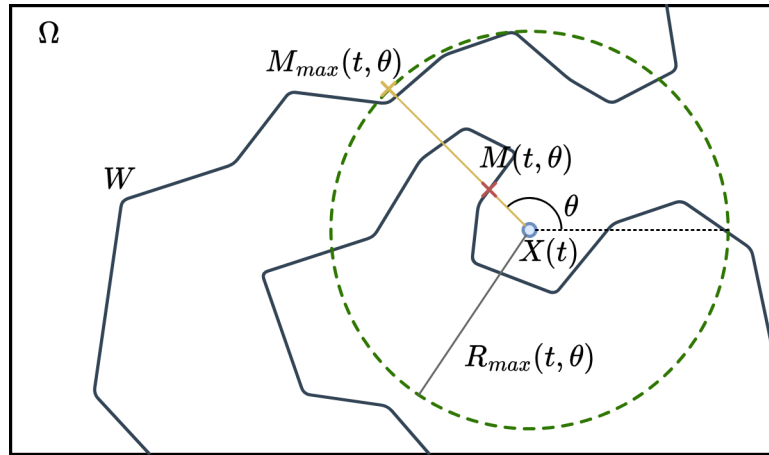The Figure III.1 visualize the notation on a diagram.

Figure III.1: Notation diagram

We can define R as:

$$R(t, \theta) = \begin{cases} \min(\text{distance}(\mathbf{X}(t), P_i(t, \theta))), & \forall i \in \{0, \ldots, \text{Card}(P) - 1\} \text{ if } \text{Card}(P) > 0, \\ R_{max}, & \text{otherwise.} \end{cases}$$

where $P(t, \theta) = \{L(t, \theta) \cap W\}$ is the set of all intersection point between the segment L and the segment of the map and $Card(P)$ the number of element in this set.

L is defined as follows:

$$L(t, \theta) = \{\, (1 - l)\,\mathbf{X}(t) + l\,\mathbf{M}_{max}(t, \theta) \mid l \in [0, 1] \,\}$$

We then have:

$$\mathbf{M}(t, \theta) = \mathbf{X}(t) + R(t, \theta) \begin{pmatrix} \cos\theta & 0 \\ 0 & \sin\theta \end{pmatrix} \mathbf{X}(t)$$

We denote $\mathbf{M}_{max}(t, \theta)$ as the point such that $R(t, \theta) = R_{max}$.

Furthermore, we define the field of vision of the robot as:

$$FV(t) = \{\, (1 - l)\,\mathbf{X}(t) + l\,\mathbf{M}(t, \theta) \mid l \in [0, 1], \theta \in [0, 2\pi[\, \} \tag{1}$$

In other words, $FV(t)$ is the star domain of Ω with a maximum radius R_{max}, within which all frontier points are reachable by a straight line from the robot.

We now construct the functional to optimize, which is related to the movement of the robot.

$$J(\mathbf{X}(t)) = \int_0^T |\dot{\mathbf{X}}(t)| dt \tag{2}$$

J is a functional function of the initial position of the robot, giving the length of the path traveled by the robot before $t = T$.

We define:

- KM: the space of the map known by the robot

- EM: the part of Ω explorable by the robot.

The time T represents a time at which the map is explored to the maximum capacity of the robot, meaning that $KM = EM$.

The Figure III.2 illustrates the notation used for the map. The green region represents the known map (KM), while the light green area corresponds to the explorable map (EM). The light red region denotes the non-achievable part of the map, which complements Ω. Additionally, W represents the walls.
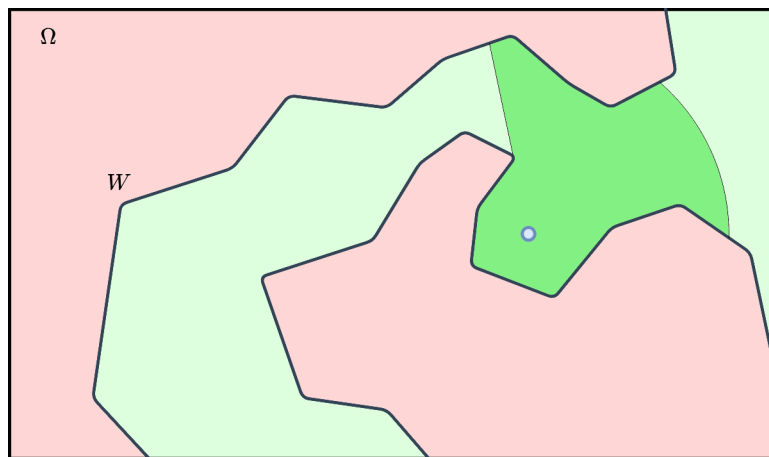

Figure III.2: Map notation diagram

To determine if $KM = EM$, we calculate the contours of the known domain. If all contours are closed, then $KM \subset EM$. Additionally, if the measures of KM and EM, namely their areas, are equal, then we can reasonably say that $KM = EM$.

For simplicity in studying the functional, we slightly modify it. This form simplifies the computation of the gradient.

$$J(\mathbf{X}(t)) = \frac{1}{2} \int_0^T \dot{\mathbf{X}}(t)^2 dt \tag{3}$$

The goal is to incorporate two constraints: $X(0) = X_R$ and $X(T) = X_{WP}$.

One major challenge is ensuring that the robot reaches position X_{WP} at time T. In other words, how can we constrain the shape of $\dot{X}(t)$ so that $X(T) = X_{WP}$?

Another issue is choosing an appropriate value for T. Should it be finite or not? Upon reflection, T must be significantly greater than the time the robot requires to reach the target position. This is because if the robot stops upon reaching the target at time t_1, then

$$\int_{t_1}^T \dot{\mathbf{X}}(t)^2 dt = 0.$$

The last major issue is how to account for the behavior of not knowing the map in advance. A first implementation could involve assuming the map is known and addressing the entire problem later.

I didn't have the time to properly address the issues described, but the optimization process and the encountered problems will be tackled in future work. Nonetheless, the method for solving such an optimization problem are given below.

III.2   Optimisation methods

Several methods can be used to solve this optimization problem, such as[30]:

- **Direct optimization methods**: These methods discretize the trajectory and solve the resulting finite-dimensional optimization problem. Examples include collocation and shooting methods.

- **Indirect methods**: These rely on deriving the necessary optimality conditions using the Pontryagin Maximum Principle (PMP) and solving the resulting boundary value problem.

The pros and cons of theses two methods are[30]:

Method	Pros	Cons
Direct	Easy to implement, no prior theoretical study needed	Less precise than indirect methods
	More robust and less sensitive to initial conditions	Can converge to local (suboptimal) minima
	Easily handle state constraints	High memory consumption, inefficient for high-dimensional problems
	Compute optimal controls in closed-loop (feedback) form	
Indirect	Very high numerical precision	Require precise initial conditions to converge
	Multiple shooting method allows parallel computation	Based on the maximum principle, which provides only local optimality
	Difficult to handle state constraints	
	Require prior knowledge of the optimal trajectory structure	

Table III.1: Comparison of Direct and Indirect Methods in Optimal Control

Another method would be to use **Machine learning-based approaches**: Reinforcement learning or neural networks can be employed to approximate optimal control policies.

Each approach has its advantages and trade-offs, depending on computational efficiency, robustness, and the nature of the constraints.

Since the optimization was not completed, which would have conditioned the continuation of this work, we cannot define an algorithm that proposes an optimal trajectory in the mathematical sense. However, this inability to solve the problem led me to consider a new dynamic trajectory planning method, which is described below.

III.3   Introducing a new method for dynamic pathfinding

In this section, we introduce a new method for real-time dynamic pathfinding. This method involves inflating the path perpendicular to the shortest path to a point, i.e., a line. The line is split if it is not free from obstacles within a given safe range.

For our study, it is crucial that this algorithm runs in real-time to avoid any obstacles. Since the goal is to explore a map using multiple robots, we must ensure that the path computation time for each robot is minimal so that they can avoid each other.

The principle is simple: draw a straight line between the robot and the waypoint (Figure V.8). If the path is not free of obstacles within the defined safe range (Figure III.3b), split the path in the middle and shoot points perpendicular to the line (Figure III.3c). When a shot point is free from obstacles, validate the point and check if the two resulting lines are free from obstacles (Figure III.3d). If not, repeat the steps described above on the new lines (Figure III.3e). Once all points are safe, simplify the path using a straightforward algorithm. The process can be visualized on Figure III.3.
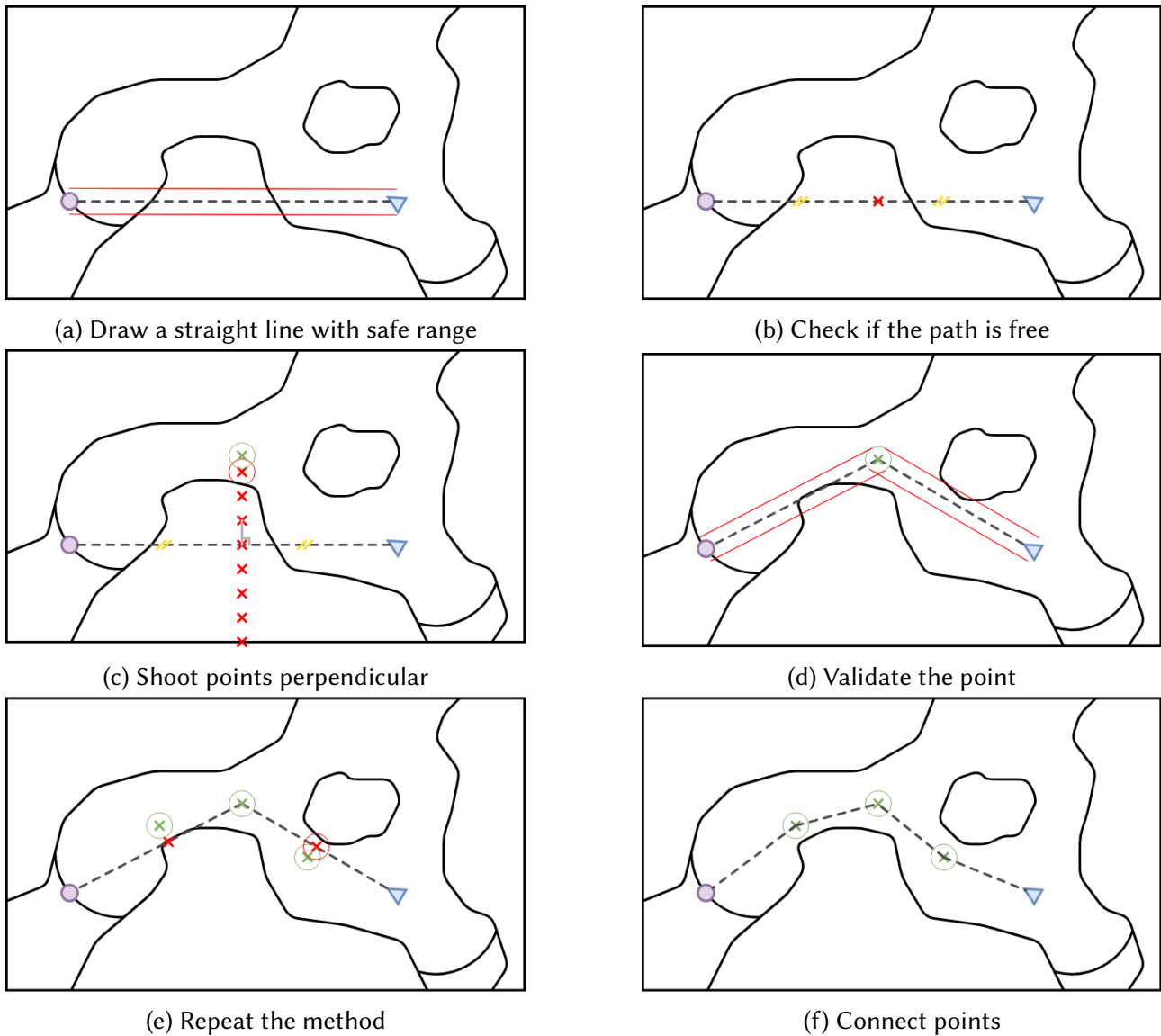
(a) Draw a straight line with safe range

(b) Check if the path is free

(c) Shoot points perpendicular

(d) Validate the point

(e) Repeat the method

(f) Connect points

Figure III.3: Visualization of the dynamic pathfinding method

III.3.1 Path simplification

The goal of this algorithm is to ensure that the path is the shortest possible by simplifying it. The algorithm iterates through all the points in the path. For each point, it checks if the path to the next point is obstacle-free. If it is, the algorithm continues to the next point. If the path is not free, the algorithm keeps the last valid point and starts the process again from there. This way, the path is simplified by removing unnecessary intermediate points while ensuring it remains obstacle-free. The method can be visualized on the Figure III.4.
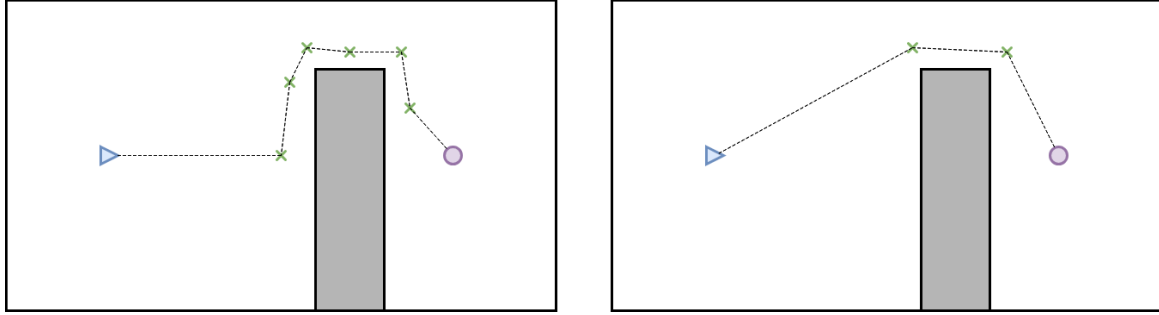
Figure III.4: Path simplification process

To ensure this algorithm works and to measure its performance, I conducted a small benchmark. While the benchmark can be theoretically calculated, I also explored a numerical method for computing the shortest path to handle various scenarios. The benchmark will serve as a performance estimator for the numerical method.

III.4 Finding the shorthest path distance

To find the shortest path, I like to take inspiration from nature by simulating a wave propagating through a medium. This way, the shortest path naturally emerges as the one the wave follows.

The wave equation describes how information spreads at a certain speed, but challenges arise—how to model refraction, how to ensure the wave propagates at a constant speed. To tackle this, I used a cellular automaton once again. Made for propagating information, they are an interesting tool for this approach.[31]

Differently from the subsection V.2, I will use cellular automata as a computational tool for simulating phenomenon. Application were found in various domain from physics to biology. Lattice Gaz Cellular Automata (LGCA) for instance are used to simulate gaz fluid flows, it is the precursor of the lattice Boltzman Method (LBM)[32]

Based on the work of *Calvo Tavia* and *al.*[31], the process is describe below:

Given a lattice $\Lambda = \{(i, j) \in \mathbb{N}^2 \ : \ 1 \leq i, j \leq L\}$ with L the size of the lattice, we define the following sets:

- $A_t = \{(i, j) \in \Lambda \ : \ a_t(i, j) > 0\}$: the set of activated cells at time t

- $\mathcal{B}$: the set of obstacles

- $\Gamma \subset \Lambda$: the set of secondary wave sources

- E_t: the set of empty spaces

- $\mathcal{M}_{ij} = \{(k, l) \in \Lambda \ : \ \|(k - i, l - j)\|_\infty = 1\}$: the Moore neighborhood of a cell (i, j)

Each cell of the lattice carries 2 variables:

- $a_t(i, j)$: the state of the cell (i, j) at time t

- $z_t(i,j)$: the distance vector of the wave from the source to the cell (i,j) at time t, defined as:

$$z_t(i,j) = \begin{pmatrix} \text{Total number of steps taken to arrive at cell } (i,j) \\ \text{Number of diagonal steps among them} \end{pmatrix}$$

To update z_t, we introduce the following variable that track the distance of the wave from the source to the cell (i,j) at time t:

$$r_{t_{ij}}(k,l) = \begin{cases} (0,0) & \text{if } (i,j) \in E_t \text{ or } (k,l) \notin A_t \\ z(k,l) + (1, \mathbb{1}_{D_{ij}}(k,l)) & \text{otherwise} \end{cases}$$

where $\mathbb{1}_{D_{ij}}$ the diagonal function indicator, i.e. 1 if (k,l) is a diagonal neighbor of (i,j) and 0 otherwise.

$$D_{ij} = (k,l) \in \Lambda \; : \; |k-i||l-j| = 1 \subset M_{ij}$$

Note that $\|r_{t_{ij}}\|_2$ is the distance mesurement from the wave source to the cell (i,j).

Moreover, we define the set of cells in the Moore neighborhood that could be a source of activation for the cell (i,j) at time t as:

$$W_t = \{(k,l) \in M_{ij} \; : \; t < a_t(k,l) + \|r_{t_{ij}}\|_2 \le t+1\}$$

Finally, we define the pair (k,l) where the distance $\|r_{t_{ij}}\|_2$ is minimal if the set of potential source of activation for the cell (i,j) at time t is not empty, i.e. $W_t \ne \emptyset$.:

$$(i_t^*, j_t^*) = \begin{cases} argmin\{\|r_{t_{ij}}(k,l)\|_2 \; : \; (k,l) \in W_{t_{ij}}\} & \text{if } W_t \ne \emptyset \\ (i,j) & \text{otherwise} \end{cases}$$

After defining all the sets and variables above, the wave propagation is governed by the following rules:

At each iteration of time, we compute the two variables $a_t(i,j)$ and $z(i,j)$ for each cell $(i,j) \in \Lambda$ as follows:

$$a_t(i,j) = \begin{cases} t+1 & \text{if } (i,j) \in \Gamma \text{ or } (M_{ij} \cap A_t) \ne \emptyset \\ a_t(i_t^*, j_t^*) & \text{otherwise} \end{cases}$$

$$z_t(i,j) = \begin{cases} z_t(i,j) & \text{if } (i,j) \notin E_t \backslash \Gamma \text{ or } W_t = \emptyset \\ r_t(i_t^*, j_t^*) & \text{otherwise} \end{cases}$$

This implementation does not allow to track the refraction of the waves, these are wrongly reflected by the obstacles. To prevent front breaking in wave propagation near obstacles, the concept of "additional secondary" wave sources is introduced, inspired by Huygens' principle, where selected cells on the obstacle boundary generate secondary waves. An algorithm determines these sources based on geometric conditions, ensuring correct wave propagation by maintaining the expected wavefront shape even when interacting with obstacles. This simple algorithm is describe in the work of *Calvo Tavia* and *al.*[31]. We will not go into the details of the algorithm here.

III.5   Analysis of the new methode

Now I have an numerical method for computing the shortest path length, we can focus on performance of the algorithm introduced. Despite this algorithm is simple and fast, making it suitable for implementation on a large number of small chips, they have some limitations that can be critical for certain cases. A small benchmark was conducted with the following results.

Eight maps were introduced in this benchmark, as shown in Figure III.5. The maps are 1200 by 700 metric units, with the robot starting at the triangle located at coordinates (100, 350) and the target point at (1100, 350).
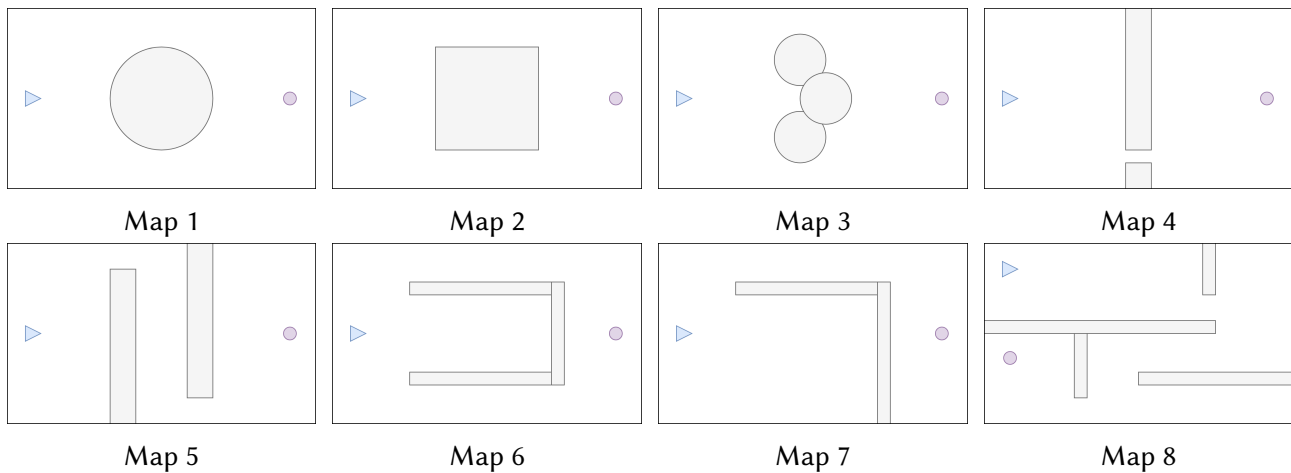
Figure III.5: Benchmark maps
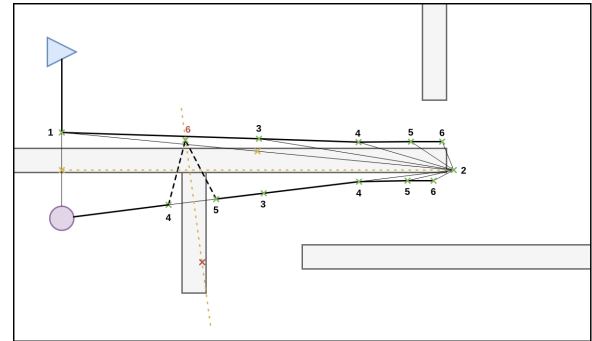
Results
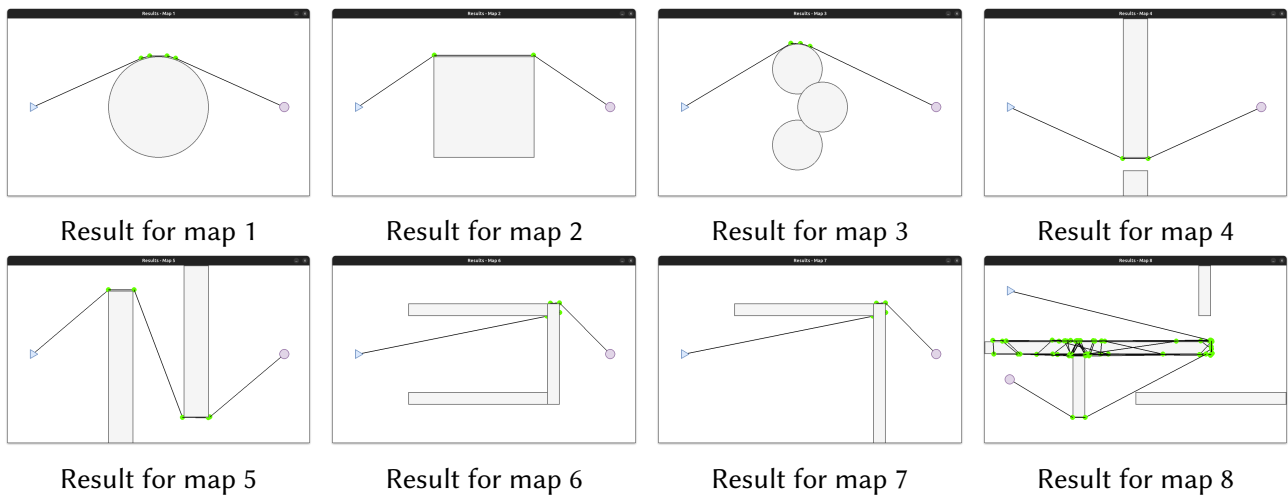
Figure III.6: Results on the benchmark maps

On maps 6 and 7, the robot is blocked by the angles of the walls. On map 8, although it appears that the robot finds a path, the lines within the wall indicate otherwise. The method is blocked by the wall just before reaching the waypoint. This outcome can be anticipated by examining the shape of the walls near the waypoint. Figure III.7 illustrates why the new method fails in the situations presented by maps 6, 7, and 8. The numbers represent the iteration at which each point is created, with red numbers indicating the iteration when the method is certain to fail due to the geometry and the way the method is implemented. While this is not tragic, it is important to note

that my goal is to explore a cave, and the robot is likely to occur only once and not repeatedly when a part of the map is fully explored. For these cases, I implemented another path planning algorithm introduced in the next secion.
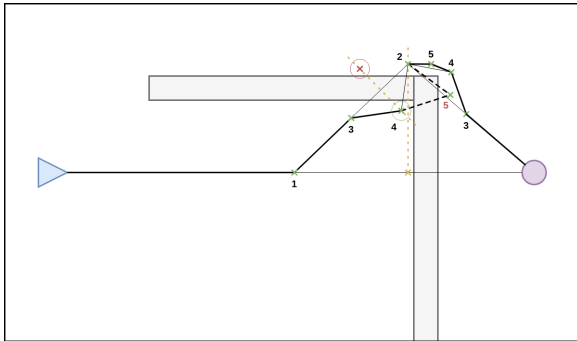
(a) Failure on map 7

(b) Failure on map 8

Figure III.7: Failure explanation for introduced method

III.5.1 Results

The calculation with the cellular automata is done with the following parameters, the size of the cells are 2 by 2 metric unit (mu) giving a uncertainty of ± 2 mu. The granularity is set to 5 for all cases without circular shape and 20 otherwise.

Map	Theoretically	Method	Relative difference (%)	Cellular Automata (± 2 mu)	Error (%)
Map 1	1081	1084	0.3	1086	0.5
Map 2	1121	1125	0.4	1122	0.1
Map 3	1122	1125	0.4	1122	0
Map 4	1084	1088	0.4	1086	0.2
Map 5	1521	1531	0.7	1522	0.1
Map 6	1166	-		1168	0.2
Map 7	1166	-		1168	0.2
Map 8	1776	-		1780	0.2

Table III.2: Benchmark results for the new method

The Table III.2 gives us the relative difference between the method introduced and the theoretical value for the shortest path. As we can see the path when found is always under 1% longer than the shortest one, moreover, we can imagine less to 1% if the compute precisely the point on the border, due to the iterative process, the point can be place up to a distance of zero to the step we move the point to check if it is safe.

Moreover, except for the 1st and 8th maps, the lattice approach for finding the shortest algorith is very good in this benchmark, within the range of uncertainty, the algorith always found the shortest path distance.
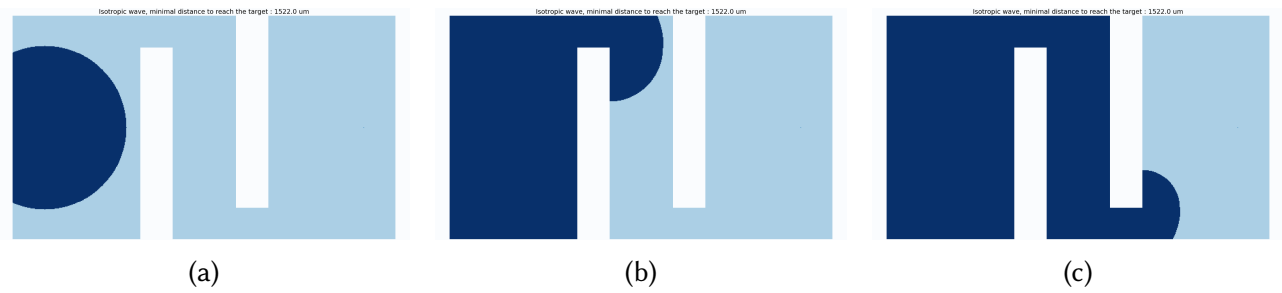
The Figure III.8 shows the waves growing on map 5.

Figure III.8: Wave propagation steps using cellular automata

III.6   Dijkstra's algorithm

The second algorithm for path planning uses Dijkstra's method. The goal is to navigate to the nearest point of the map where a robot already passed to the waypoint. The waypoints are located on the frontiers of the robot's explored area, calculated using lidar data. This ensures that the path from the nearest location to the waypoint is a straight line free from obstacles.

Dijkstra's algorithm is an algorithm to find the shortest path from a source to a final node in a weighted graph.[33]. It works by iteratively selecting the node with the smallest tentative distance, updating the distances to its neighbors, and marking it as visited. This process continues until all nodes have been visited or the shortest path to the target node is determined.[34]

To ensure the path to the nearest point exists, the robot will move only on cells previously traversed by a robot. This path will be shortened by the algorithm described earlier. If the robot remains in a certain area for too long, indicating a deadlock, the Dijkstra path is computed to guide the robot out. This emergency path is only calculated when the robot is in a deadlock.

III.7   Perfomances

On an AMD Ryzen™ 5 3500U, using *Python* and the map in Figure V.5b, the robot computes the next global path in an average of 1.02×10^{-1} seconds, approximately 98 times per second. This performance is suitable for real-time navigation and obstacle avoidance.

In the simulator, to be more computationally friendly, I simplified the method during implementation by computing only the next point. The difference in trajectory can be seen in Figure III.9. This approach sacrifices the optimal path for reduced computational time. The difference in path can be significant, as shown in Figure III.9c. The path with the blue-green dots can illustrate how the robot navigates without prior knowledge of its environment before reaching the target point.
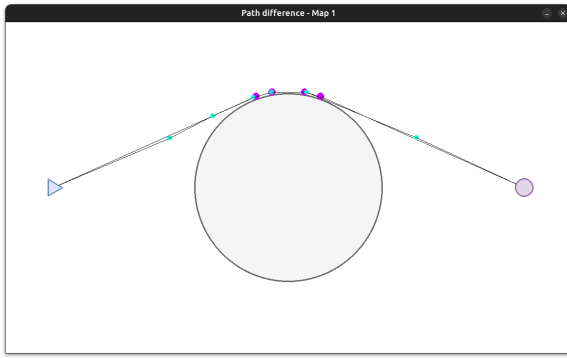
(a) Trajectory difference on map 1
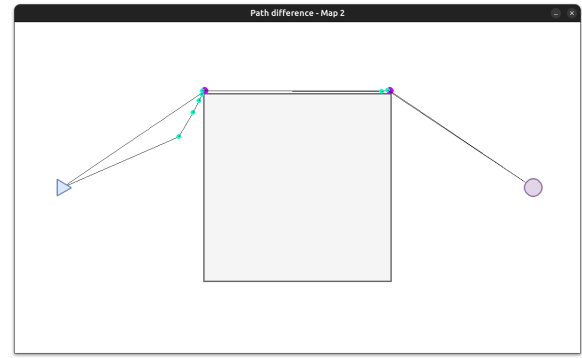
(b) Trajectory difference on map 2
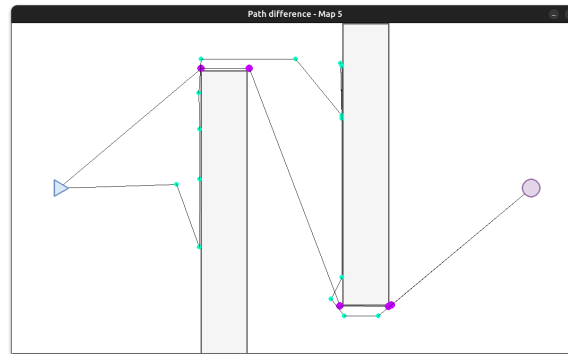
(c) Trajectory difference on map 5

Figure III.9: Trajectory differences on maps 1, 2, and 5

On the same map and laptop, the local approach for path planning is significantly more efficient in terms of computational time. The average computation time is 8.20×10^{-4} seconds, allowing for approximately 1220 executions per second. Despite the method's poor performance in the benchmark, it performs well in the simulator, resulting in better map exploration and improved path quality. The Figure III.10 shows the two maps after exploration of only one robot.
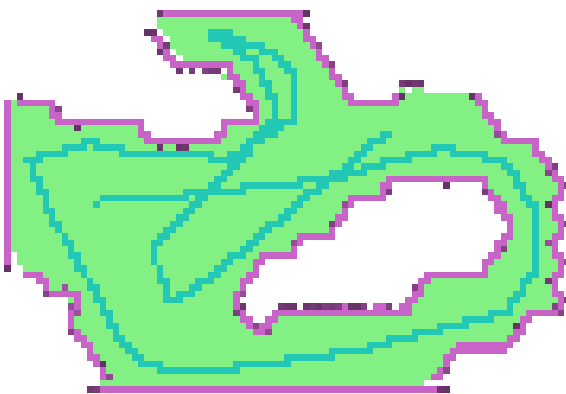
(a) Map explored using global waypoint path
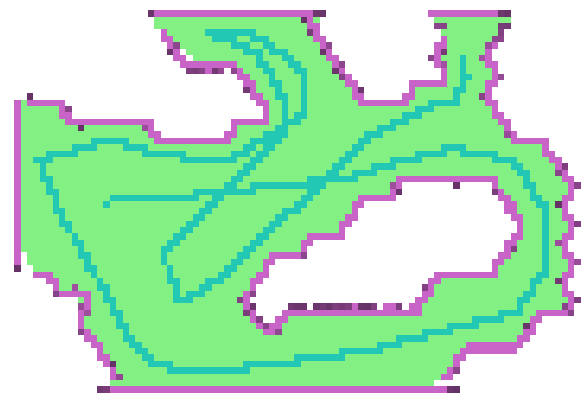
(b) Map explored using local waypoint path

Figure III.10: Comparison of map exploration using global and local waypoint paths

III.8 Performance Analysis Methodology

To evaluate the performance of the algorithm, we will use the following criteria: range, robustness, and completeness. These criteria are part of a broader methodology that also includes effectiveness and speed.

III.8.1 Delimited Range of Application

The combination of the two algorithms is well-suited for both open spaces and medium-sized cavities. However, the method performs significantly better in open spaces, as indicated by the benchmark results. This approach could be considered for aeronautical applications. It should be noted that in narrow cavities, Dijkstra's algorithm is likely to be executed each time a new direction is chosen. The method introduced can be implemented in either 2D or 3D, which is a significant advantage for applications in cavities.

III.8.2 Robustness to Failure and Uncertainty

Robustness to failure and uncertainty means that the robot must handle all possible cases, such as being in a deadlock and failing to exit it. The second part of the method, Dijkstra's algorithm, ensures that the robot can exit any situation by moving back along its path.

III.8.3 Completeness

Completeness means that the robot must explore all the domain possible to be explored or accomplish all the possible tasks given to it. To study completeness, a vast range of tests must be conducted. At this stage of the study, it is not certain that the algorithm is complete.

III.9 Global waypoint computing

Now the local navigation is set, we need to define how to select the next waypoint occording to the map. First of all this is done by computing the frontiers of the know domains using computer vision detection method.

To extract the open frontiers of the map, I propose the following method:

1. Obtain the live grid map of the environment.

2. Generate a thresholded version of the map to distinguish wall areas from other regions (unknown and free spaces).

3. Identify the contours of the map by appliying an Adaptive Gaussian Treshold.

4. Apply a single iteration of dilation on the wall treshold.

5. Add the contour map from the dilated wall image to isolate the open frontiers.

The Figure III.12 shows an example of the method described above on the Figure III.11
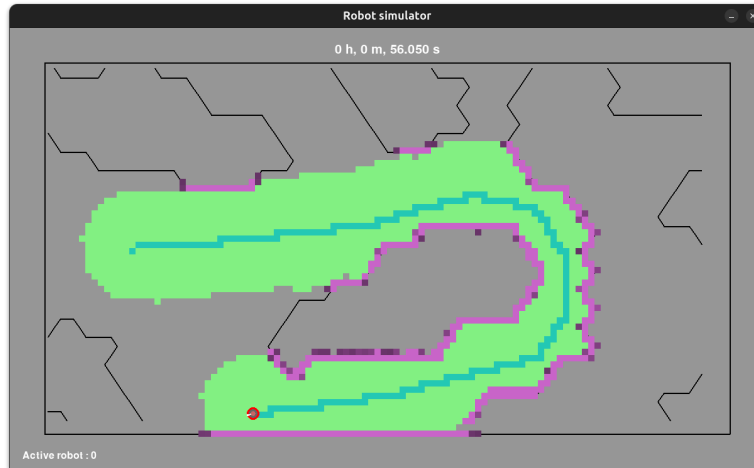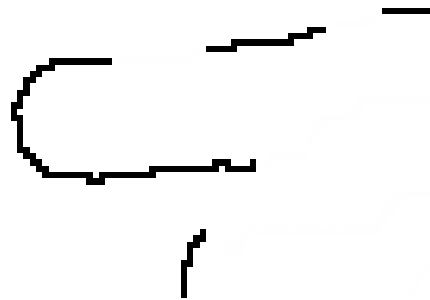
Figure III.11: Open frontiers of the map example

(a) Thresholded version of the live grid map

(b) Adaptative thresholded version of the map

(c) Dilatation of the wall thresholded map

(d) Sum of the (b) and (c) images highlighting open frontiers

Figure III.12: Steps to extract the open frontiers from the live grid map

Once all frontiers are computed, an algorithm assigns a weight to each cell, and the robot selects the cell with the minimal weight. Figure III.13 shows the weights, with only one out of every ten points displayed for clarity.
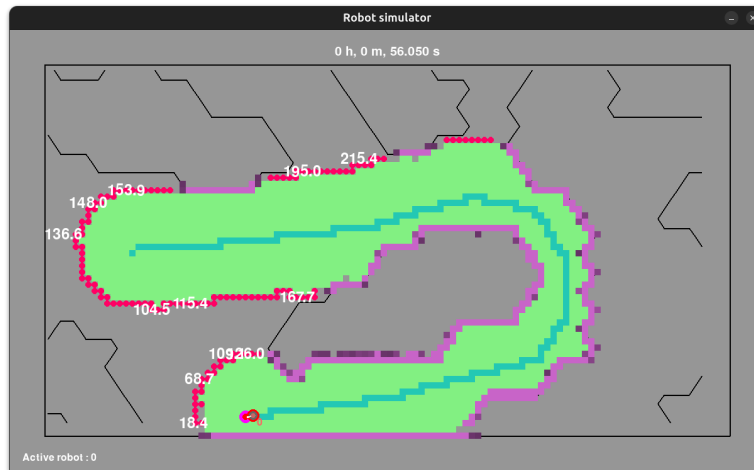
Figure III.13: Weighted frontiers of the map

The weights are calculated using four parameters of the robot: its position, heading, speed, and angular speed. Each of these parameters is assigned a different weight to prioritize one constraint over the others, such as position, heading, speed, or angular speed.

Now that both local and global path planning are established, the next step in our journey is to develop an efficient communication system between robots, ensuring seamless coordination and collaboration.

IV   Communication

This algorithm aims to coordinate the exploration of unknown areas by a swarm of robots without a central entity to manage the coordination.

IV.1   Operation

A the beginning of the exploration, we assign a number to each robot, the smallest one is the master. Another implementation would be to choose the master accordingly to their battery level, the one with the higher battery level is the master. Indeed, the master robot is the one that consume more due to all the calculation it will do. To keep it simple, we choose the smallest number.

The communication is bases on a strong hierarchical structure, the master robot give the direction to all robot link with it with a bigger number.

If the group split, the master change to the strongest robot in the group. Robot communicate with each other using light, detecting if the robot can communicate with another is made, in a first time, by ensuring visual contact. A more powerful approach is to simulate the travel of the light in the medium.

The master of a group is a hub for communication, each new direction is given by him using this method :

- Each robot in a group sends the master any part of the map that is new to it

- Each robot of a group sends the master the waypoint with the open frontier index it wants to explore.

- Once all robot of the group send its information to the master, the protocol for gathering information is given later, it compute the normalized cost table for each combination robot-waypoint. Values are given between 0 and 255 to send only one byte information ensuring low communication volume.

	Robot 1	Robot 2	Robot 3	Robot 4	Robot 5
WP 1	**23**	87	234	56	192
WP 2	245	76	**11**	68	39
WP 3	90	**21**	73	**50**	164
WP 4	132	58	49	77	**25**
WP 5	181	**13**	66	**39**	70

Table IV.1: Example of costs table for a group of 5 robots with 5 differents waypoints

In the case, there is less waypoint than robot, the group split in two groups.

	Robot 1	Robot 2	Robot 3	Robot 4	Robot 5
WP 1	**61**	125	93	47	**59**
WP 2	88	**12**	**53**	**29**	174

Table IV.2: Example of costs table for a group of 5 robots with 2 differents waypoints

In the case, there is less robot than waypoint, each robot explore a zone.

	Robot 1	Robot 2	Robot 3	Robot 4	Robot 5
WP 1	**42**	134	212	63	189
WP 2	215	98	**19**	75	48
WP 3	102	**32**	95	68	141
WP 4	142	74	58	89	**31**
WP 5	193	**27**	78	**54**	83
WP 6	156	63	**17**	99	115
WP 7	173	49	132	82	**23**
WP 8	204	**57**	146	71	94

Table IV.3: Example of costs table for a group of 5 robots with 8 differents waypoints

- The master, after computing the cost table, distribute the waypoint across the group minimizing the cost combination. For instance, in the Table IV.1, Table IV.2 and Table IV.3, bold number are the minimum cost for each robot and each waypoint, however the minimun combination of cost is given by the blue one.

IV.1.1 Meeting of two groups

If 2 groups met, both master share all the information they have without moving. Once the trasfert is done, the strongest master take control of all the group and continue the exploration.

IV.2 Inter-robot communication protocol

For each robot one at a time in the group. The master asked for informations. Listen for the answer. If an answer is given, the master validate the tranfert of information. Else, retry, retry and retry and skip it. Compute thing. Give an order to the robot. The robot listen for the order. If an answer is received, the robot validate the tranfert of information. Else, retry, retry and retry and skip it.

IV.3 Encoding

To encode the message, we use a specific message frame. This frame ensures that if part of the message is incorrectly transmitted, such as when a bit in the message is flipped, the receiver will detect the error and request the message to be resent until it is received correctly. This method is known as Automatic Repeat reQuest (ARQ).

The message includes two verification mechanisms, unit sum check and parity bit: one to check the integrity of the transmitted message and another to confirm the identity of the sender.
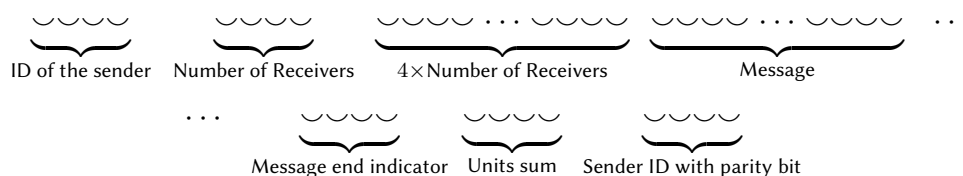
IV.3.1 Description

Emitter ID: ⌣⌣⌣⌣
The unique 4-bit identifier for the sender of the message (range: 0-15).

Number of Receivers: ⌣⌣⌣⌣

Optional 4-bit field indicating the number of receivers. If there is only one receiver, this field is set to *0000*.

Receiver IDs: ⌣⌣⌣⌣ ⋯ ⌣⌣⌣⌣

Each receiver's ID is encoded in 4 bits. For multiple receivers, these IDs are listed sequentially.

Message Content: ⌣⌣⌣⌣ ⋯ ⌣⌣⌣⌣

Type indicators (up to 15 indicator types, 0000 being already reserved):

Type indicator	Type	Data size
0000	Message end	4 bits
0001	Integer	8 bits
0010	Integer	32 bits
0011	Float	32 bits
0100	String	8 bits for length + 8 bits per character
0101	List of int8	16 bits for length + 8 bits per integer
0110	List of float	16 bits for length + 32 bits per float
0111	Message type ID	8 bits
1000	Robot info	168 bits
1001		
1010	Binary message	16 bits for length + 1 bit per binary bit
1011		
1100		
1101		
1110		
1111		

Table IV.4

Units sum: ⌣⌣⌣⌣

4 bits representing the sum of the bit in the message frame modulo 16.

Emitter ID with Parity: ⌣⌣⌣⌣

The emitter ID is repeated with a parity bit. The parity ensures the total number of 1s in the message frame up to the emitter ID with parity is correct. If odd, the last bit is flipped.

We also define a table of message with their specification, for each message received, there is a frame of response with some test.

Message ID table:

Message ID	Description
00000000	Ask to repeat the last message
00000001	Last message received correctly
00010000	Ask the connected robot set
00010001	Send the connected robot set
00100000	Ask to all where they want to go
00100001	Send next waypoint position with frontier ID
00100010	Send robot info
00100011	Send next position to reach
00110000	Ask for live grid map synchronisation
00110001	Send updated live grid map
01010101	Transmit message from another robot

Robot info: 2 *float32* for position, 1 *float32* for velocity, 1 *float32* for rotation, 1 *float32* for angular velocity, 1 *int8* for energy

Connected robot set: 4 bits for set length + 4 bits per robot id

IV.4 Performance Analysis Methodology

To evaluate the performance of the algorithm, we will use same criterion basis than those used for the path planning analysis: range, robustness, speed.

IV.4.1 Delimited Range of Application

The communication protocol is designed for swarm-based robotic exploration, enabling robots to exchange map data, coordinate movements, and share status updates without relying on external infrastructure. It is applicable in both indoor and outdoor environments, such as disaster response, planetary exploration, and search-and-rescue operations. Communication is initially based on direct visual contact (light-based communication) but can be extended to simulate light travel for improved reliability. The protocol is optimized for scalability, ensuring efficient data exchange in small to medium-sized robotic swarms.

IV.4.2 Robustness to Failure and Uncertainty

To ensure robustness, the protocol incorporates error detection and recovery mechanisms, such as Automatic Repeat reQuest (ARQ), unit sum check, and parity bit verification to detect corrupted messages and request retransmission. If communication fails, the system automatically retries multiple times before skipping a failed recipient, ensuring overall progression. Additionally, the master role is dynamically reassigned if a group splits or if a master robot fails, maintaining coordination despite potential disruptions. The compact encoding of messages also minimizes bandwidth usage, making the system resilient to network constraints.

IV.4.3 Speed

An important aspect of the method is its speed and efficiency. The use of compact one-byte encoded cost values ensures that communication remains lightweight, reducing transmission delays. The protocol follows a strict request-response cycle, preventing unnecessary retransmissions and

minimizing wait times. Moreover, by enabling parallel communication between different groups, it ensures that multiple robots can process and execute tasks simultaneously, leading to faster exploration and real-time adaptability to environmental changes.

With the path planning method established, effective communication ensures coordination among robots. We have analyzed its range, robustness and speed, showing its role in efficient exploration.

Next, we focus on the Python-based simulator, which will test and refine both the path planning and communication protocols before real-world deployment.

V Simulator implementation

V.1 Purpose and range of the simulator

The simulator is designed to test various algorithms and methods for multi-robot exploration of caves. The robots operate in the air while moving along the floor. To simplify the problem, we approximate the floor as a 2D plane without any surface irregularities.

V.2 Creation of the map

The map consist in succession of straight lines generate by marching square method on a simple cellular automata.

V.2.1 Cellular Automata

A cellular automaton is a grid of cells, where each cell can exist in different states based on predefined rules. The concept was developed by Stanislaw Ulam and John von Neumann in the 1940s.

The most famous cellular automaton, which helped popularize its use, was developed by John Conway: *Conway's Game of Life*. This model follows a set of four simple rules, which can be found here. The rules are based on the properties of neighboring cells.

There are two commonly used neighborhood types in cellular automata:

- **Von Neumann neighborhood**, which considers the four direct neighbors (left, top, right and bottom).

- **Moore neighborhood**, an extension that includes all eight surrounding cells, both diagonal and direct neighbors.
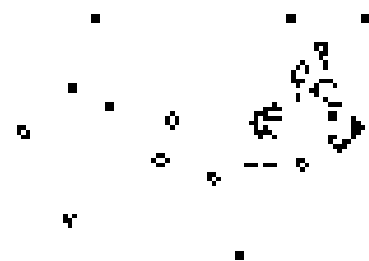
The beauty of this system lies in the complexity that emerges from such simple rules. Beginning with the configuration shown in Figure V.1a, the system evolves into the states illustrated in Figure V.1 at generations 87 and 263.
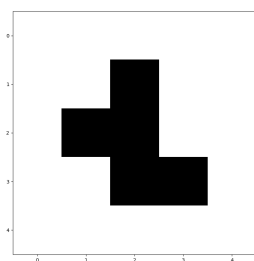
(a) Iteration 0   (b) Generation 87   (c) Generation 263

Figure V.1: Conway's Game of Life

Programmers, computer scientists, and mathematicians began cataloging all the patterns they encountered and constructed remarkably complex machines.

In our case, the rules are defined as follows:

- If there are more than 4 activated cells in the Moore neighborhood, the cell activates.

- Otherwise, the cell deactivates.

I implement the method on a Cartesian grid and then transform it into a triangular mesh, as shown in Figure V.2. The maps generated after this transformation improved in quality.
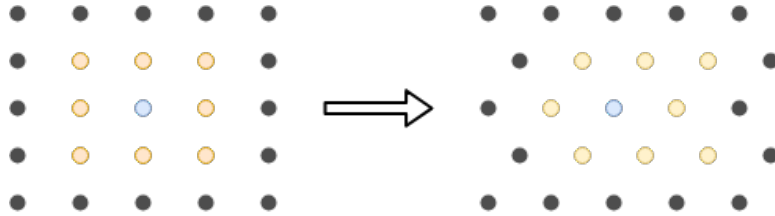

Figure V.2: Grid transformation

Using the Marching Squares method, I draw the boundaries between occupied cells (red dots) and unoccupied cells (green dots).

V.2.2 Marching squares

Marching Squares is a technique for generating the contours of a two-dimensional grid, which, in our case, is a triangular mesh. As we traverse the domain, the boundaries are drawn accordingly. Figure V.3 illustrates all possible states of an element composed of three cells.
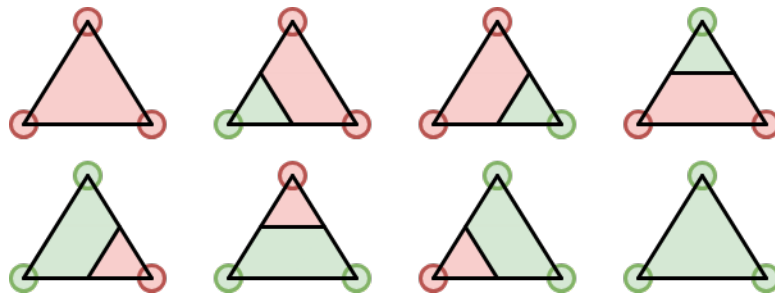
Figure V.3: Isolines, possible states of a triangular element

At the end of both steps—transformation and Marching Squares—we obtain the following schematic representation:
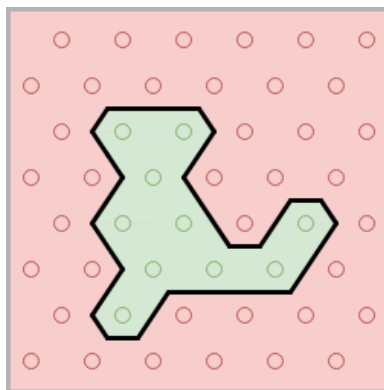
Figure V.4: Exemple scheme of map generated

In the simulator, maps are generated based on a seed and a random generator. The shape of each map is controlled by three parameters: the step size along the x-axis, the step size along the y-axis, and the overall map size.

This allows for a vast variety of map configurations. Figure V.5 illustrate some examples.
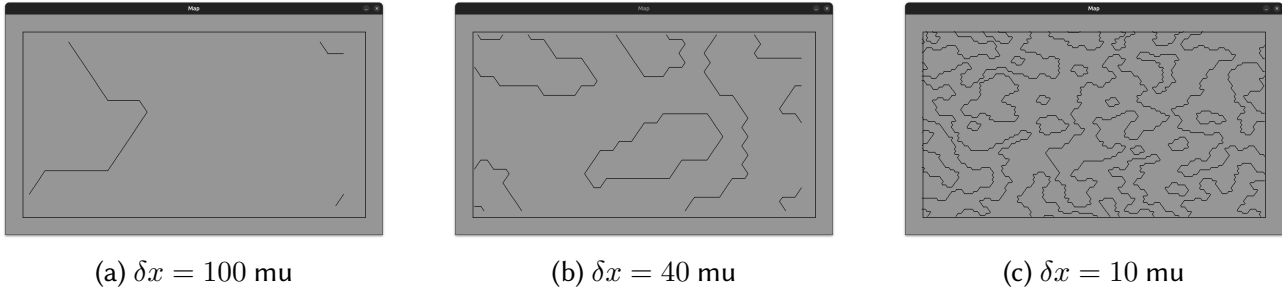

(a) $\delta x = 100$ mu   (b) $\delta x = 40$ mu   (c) $\delta x = 10$ mu

Figure V.5: Different maps made with equilateral triangles

The advantage of this technique for generating maps is its adaptability to 3D environments, allowing for the creation of detailed cave maps.
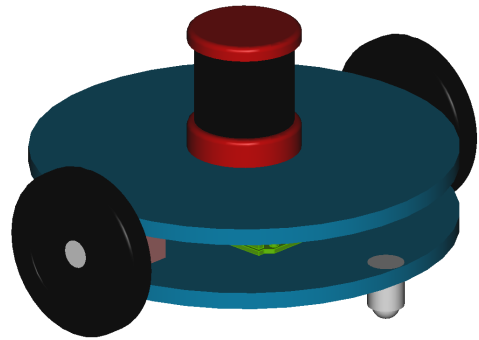
V.3 Robot implementation

V.3.1 Robot model

As a robot, I used a simple diferential drive robot with two idle-wheels and 2 driven wheels.

A model of the robot can be found on Figure V.6a and Figure V.6b.
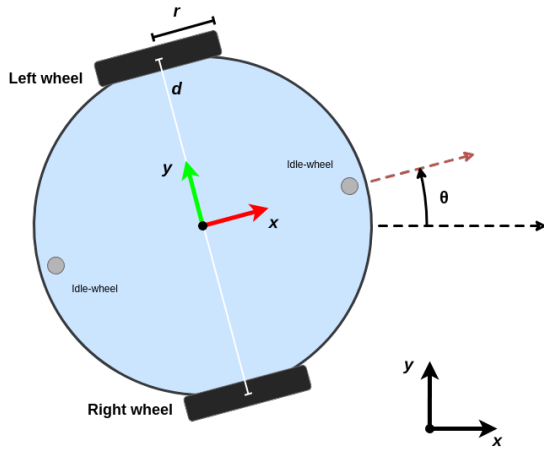
(a) Differential drive robot diagram   (b) Differential drive robot 3D model

Figure V.6: Robot models

The kinematics of the robot is given by :

We assume that the grip is perfect between the road and the robot wheels, the robot wheels roll without sliding. Under this condition, V_R and V_L, the speed of the wheel in the ground reference is given by :

$$V_R = r \times \omega_R \text{ and } V_L = r \times \omega_L$$

So that, V the velocity of the robot is the sum of both speed divided by two. Decomposition of the velocities in the ground reference (O, x, y), we have :

$$\dot{x} = \frac{r}{2}\left(\omega_R + \omega_L\right)\cos\theta$$
$$\dot{y} = \frac{r}{2}\left(\omega_R + \omega_L\right)\sin\theta$$

Moreover, we note φ the angular velocity of the robot in the robot reference isolating one wheel.

$$\varphi_R = \frac{r}{d} \times \omega_R \text{ and } \varphi_L = \frac{r}{d} \times \omega_L$$

In this case, the difference divided by two of the two gives the angular speed of the robot.

$$\dot{\theta} = \frac{1}{2}\left(\varphi_R - \varphi_L\right)$$

The difference is taken as is to ensure that the angular speed is positive when the angle increases in the anticlockwise direction.

$$\dot{\theta} = \frac{r}{2d}\left(\omega_R - \omega_L\right)$$

This model for the robot isn't yet integrated due to the ongoing optimization of the path, which is discussed in the perspectives section.

V.3.2 Sensors

The robot is equiped with 2 sensors by default, an accelerometer that calculates acceleration in x and y axis and the rotation along z axis. The second sensor is a LiDAR sensor.

A LiDAR stand for Light Detection And Ranging is a sensor that emits laser beams and measures the time it takes for the beams to return after hitting an object. This time-of-flight measurement is used to calculate the distance between the sensor and the object. LiDAR is commonly used in robotics for exploration applications. The operation of a LiDAR sensor is illustrated in Figure V.7.
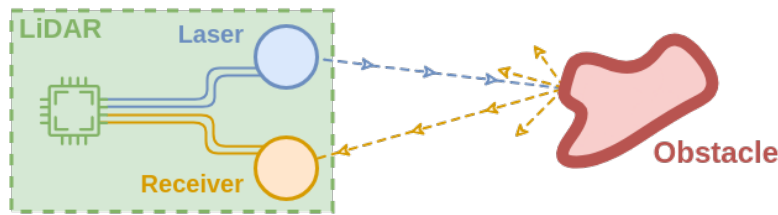

Figure V.7: LiDAR operating diagram

In our case, we work in the air, which is why we chose LiDAR. For underwater exploration applications, sonar is used and works on the same principle. Instead of emitting laser beams, sonar emits sound waves.

For estimating inertial parameters, more advanced sensors such as Doppler Velocity Logs (DVL) can be used. They work by emitting sound waves and calculating the Doppler effect to estimate the position and rotation of the robot.

Simulating the LiDAR was challenging in terms of performance. Initially, I attempted to simulate it by shooting rays and computing collisions with any walls in the map. However, this approach severely impacted the simulator's performance. To improve efficiency, I divided the map into multiple cells, storing walls that intersect each cell. By only checking the cells that the LiDAR beam passes through, rather than the entire map, performance was significantly enhanced.

To determine the indices that the LiDAR beam crosses, I implemented a function similar to the Bresenham's line algorithm, which is used to draw lines on a screen. This task was also complex because it was crucial not to miss any cells that the beam passes through. I used a modified version of Bresenham's line algorithm that expands from one cell in all directions. This method, illustrated in Figure V.8, ensures that all relevant cells are checked, saving a considerable amount of computational time for other tasks.[35]
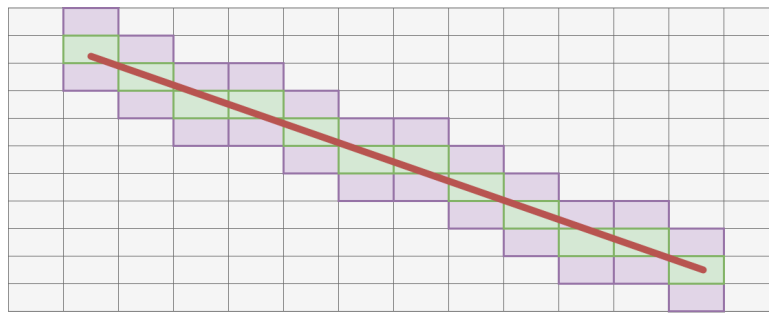
Figure V.8: Drawing a line using Bresenham's algorithm

V.3.3   Mapping

Simultaneous Localization and Mapping (SLAM) is a key problem in robotics, enabling a robot to determine its position while simultaneously building a map of its environment. Various techniques have been developed to tackle this challenge. For instance, EKF-SLAM uses a Kalman filter to estimate both the localization and the map but faces scalability limitations in large environments. FastSLAM, which relies on a particle filter, enhances scalability by handling world features through multiple hypotheses.

Graph-based approaches, such as Graph-SLAM, are effective for large-scale optimization problems but require complex data management. Visual SLAM (V-SLAM) leverages cameras to estimate localization and construct maps, whereas LiDAR-based SLAM relies on laser sensors for precise depth measurements, making it particularly useful for outdoor environments. Dynamic SLAM variants manage environments with moving objects by excluding them from map updates.[36]

In this work, I used LiDAR-based SLAM. The robot scans its environment using a LiDAR sensor and builds a map based on the collected data. The map is then used to localize the robot within the environment.

The localization process involves determining the collision points where the LiDAR beams intersect with walls. At each collision point, a circle is drawn with a radius equal to the measured LiDAR distance. The robot's position is then estimated by calculating the intersection of multiple such circles. The method is visualized in Figure V.9. Here, the central red dot represents the intersection of the circles, the red beams are the LiDAR beams, and the green dotted line indicates the maximum range of the LiDAR.
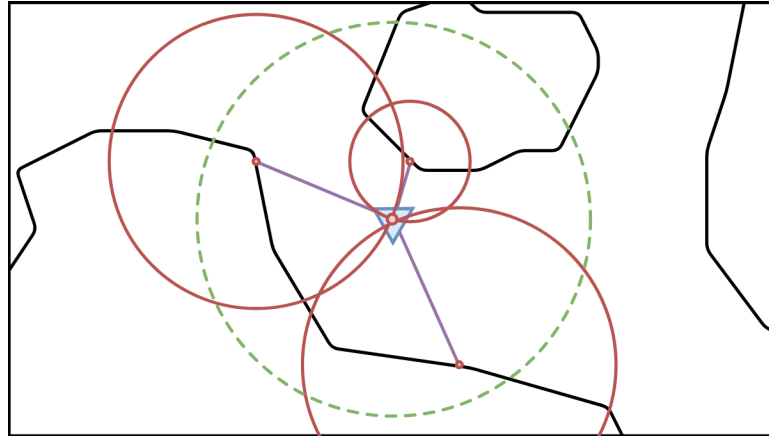
Figure V.9: Correct robot position using LiDAR data diagram

For mapping, I use a feature map, referred to as the live grid map in this section. Each cell in the map can have one over multiple states. When the LiDAR detects a collision, the corresponding cell in the map is assigned an integer value between 100 and 200. The closer the value is to 200, the more certain it is that the cell represents a wall. When the robot passes through a cell, its value is set to 19. If the cell is free, its value is set to 20. A value of 0 indicates that the state of the cell is unknown.

Since the robot operates alongside other robots, the map must account for the presence of multiple dynamic objects. To achieve this, a twin of the live grid map is created, which tracks the number of times a LiDAR beam collides with an object in each cell. Each time the LiDAR detects a collision, 3 is added to the cell in the occurrence map. Conversely, each time the LiDAR beam passes through a cell without detecting a collision, the occurrence value of that cell is decreased by 1. Any cell with an occurrence value above 0 is considered to be occupied by a wall. The higher the occurrence value, the more likely the cell is occupied.

The asymmetry between adding and subtracting values in the occurrence map makes the map more reliable. If the values were symmetric (i.e., adding 1 for a collision and subtracting 1 for no collision), the live grid map would be less secure and more sensitive to changes. For example, in Figure V.10, if the values were symmetric, the green cell would be considered free of obstacles. However, with the current method, the green cell would have a value of $3 - 1 - 1 = 1$, indicating that it is occupied.

This way, if the occurrence value is high, we can be certain that there is a non-moving obstacle in that area. If the occurrence value is low, the cell is less likely to be occupied.
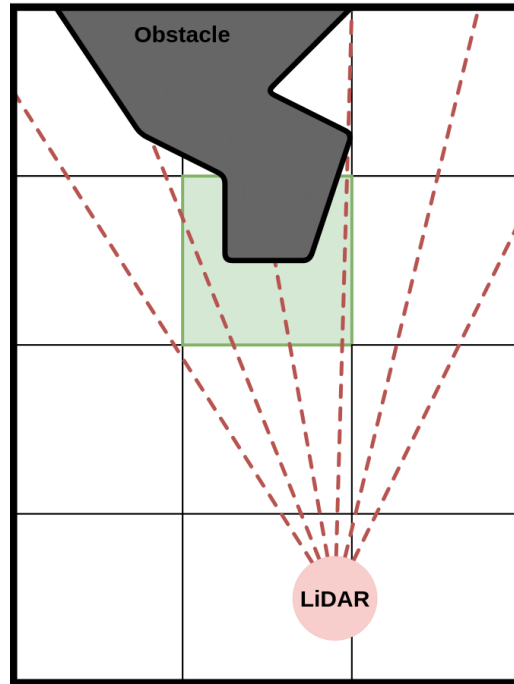
Figure V.10: Importance of asymetry in adding and subtracting in the occurence map
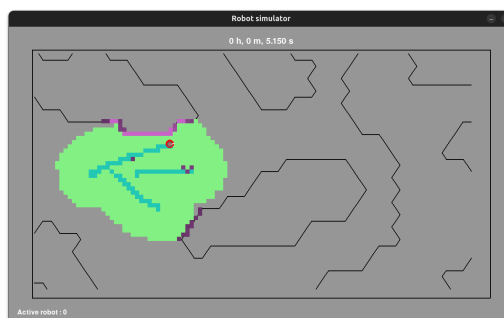
Figure V.11 illustrates the map updates over time with three robots. The purple squares represent occupied cells in the live grid map (dark purple = 100, light purple = 200). Green areas indicate free spaces, gray areas are unknown, black lines outline the map, and the small red dots represent the robots:
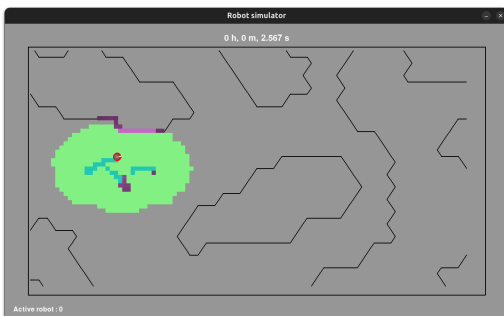
(a) Map update at time t_1
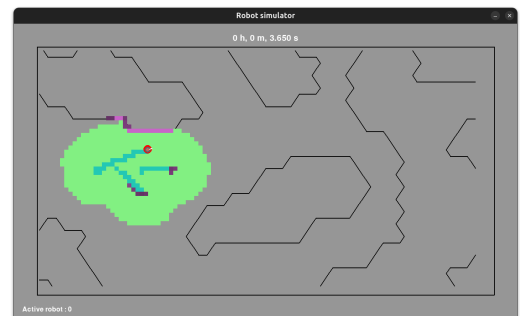
(b) Map update at time t_2

(c) Map update at time t_3

Figure V.11: Map updates over time

V.4   Communication implementation

Robots communicate using different color channels to distinguish message types. For example, white could be used for broadcasting messages to all robots. Communication remains unaffected by overlapping colors as long as each robots color identifier is not a linear combination of the others. For precise detection, we could use the SEN10656 4-channel color sensor. Each robot can emit on all channels while receiving on a specific one.

Currently, communication has been implemented up to the waypoint decision-making stage. However, the master robot does not yet issue movement orders to others. Despite this, the transmission of map data and robot status information functions effectively.

Beyond the previously developed method, several improvements have been introduced. For instance, a conversation history has been added, and the transmission and reception rates can be adjusted to simulate different conditions. Every second, each robot sends a status update to all others, ensuring a continuous exchange of information.

These adjustments were crucial for refining communication, making it more adaptable and realistic for various scenarios.

V.5   Final results

This section presents the final results of this work, evaluating the performance of the proposed multi-robot exploration method. The results reflect the current state of implementation, considering that some adjustments and features are not yet fully realized as initially conceived.

The results of testing the simulator on different maps are summarized in Table V.1.

Note:

During some tests, robots occasionally collided with each other. After investigating the issue, I found that the LIDAR frequency needed to be significantly higher for real-time collision avoidance. Otherwise, a Kalman Filter should be implemented to predict robot movement and adjust their paths accordingly. A Kalman Filter is an optimal recursive estimation algorithm that fuses sensor data with a predictive model to estimate the true state of a system, even in the presence of noise and uncertainties. In this context, it would help predict the future positions of robots based on their past trajectories, allowing them to proactively avoid collisions rather than relying solely on real-time sensor data.

Since the final part of the communication protocol has not been implemented, the paths of the robots frequently overlap, resulting in a suboptimal solution. This observation is seen in the Table V.1.

Map Size	Number of Robots	Exploration Time	Relative Improvement
30	1	2'20"	-
30	2	1'3"	55%
40	1	2'35"	-
40	2	1'32"	41%
55	1	2'7"	-
55	2	54"	57%
100	1	3'16"	-
100	3	1'28"	55%
100	5	1'12"	63%

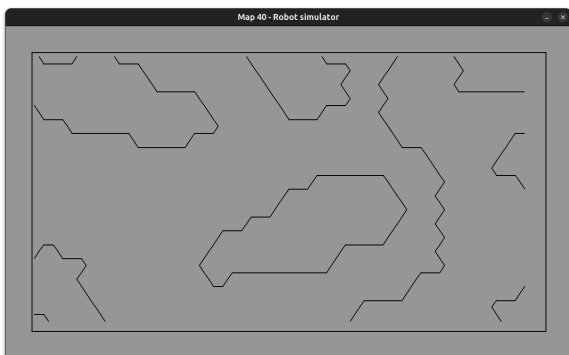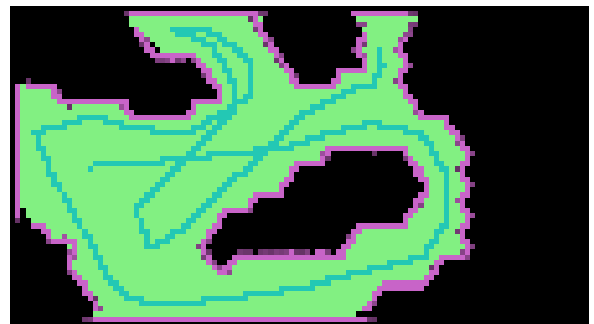Table V.1: Exploration times for different map sizes and number of robots
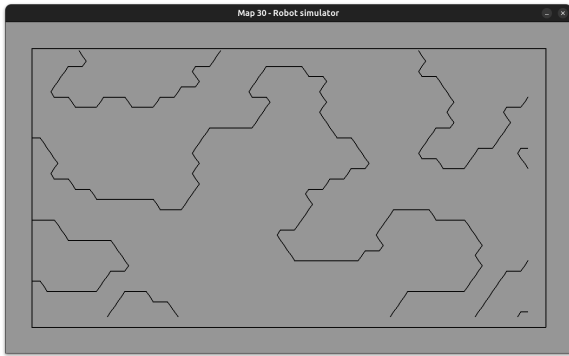
(a) Map 30
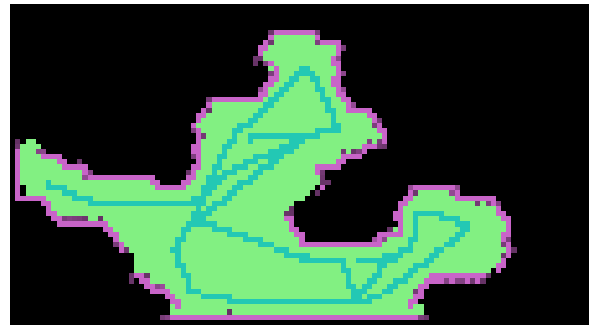
(b) Map 30 - End of the exploration
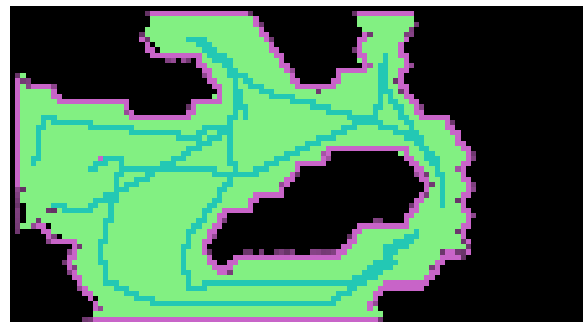
Figure V.12: Exploration on map 30
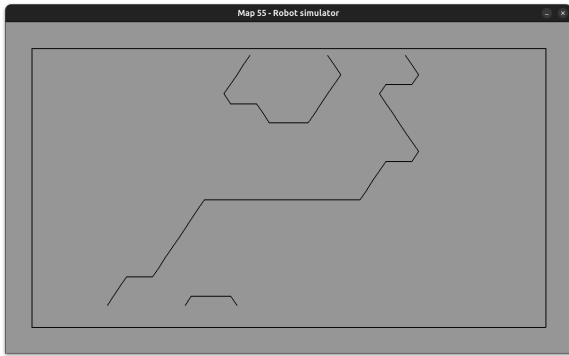

(a) Map 40

(b) Map 40 - End of the exploration
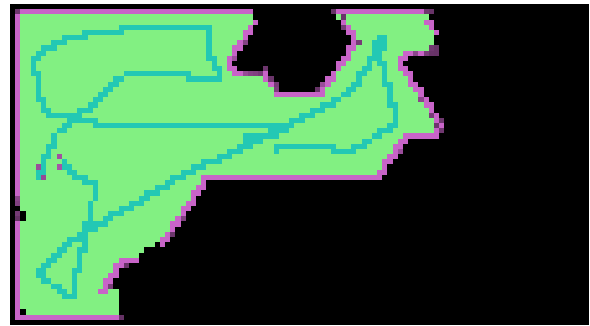
Figure V.13: Exploration on map 40

(a) Map 40 - End of the exploration with 2 robots
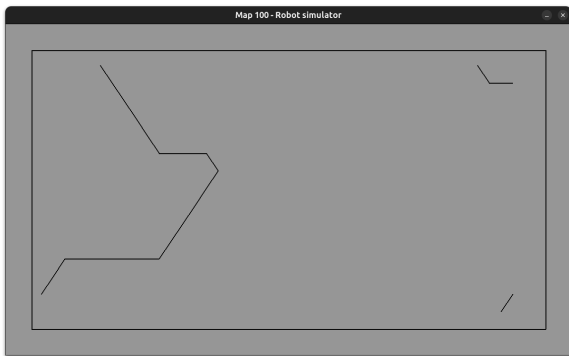
Figure V.14: Exploration on map 40
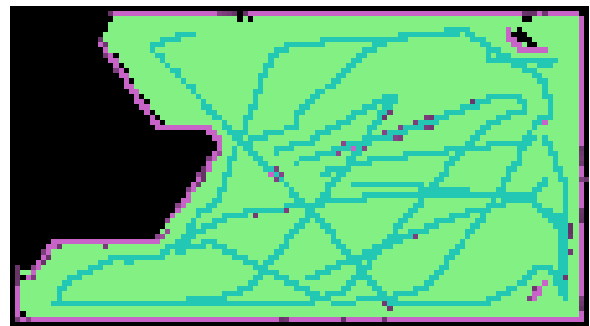
(a) Map 55

(b) Map 55 - End of the exploration

Figure V.15: Exploration on map 55

(a) Map 100

(b) Map 100 - End of the exploration

Figure V.16: Exploration on map 100

Conclusion

In conclusion, this work has successfully proposed and tested novel methods for trajectory planning and communication within drone swarms for cavity exploration. The proposed trajectory planning algorithm, which divides the shortest path to the target into successive segments, has demonstrated its effectiveness in enabling dynamic and rapid adjustments to the robots' paths. Furthermore, the master robot-based coordination approach has proven efficient in minimizing data transfer while effectively managing communication errors through the ARQ method. All these strategies were tested within a Python-based simulator, providing valuable insights into the practical application of these methods in real-world scenarios. The results of this research open up new possibilities for advancing multi-robot systems in complex environments, offering a solid foundation for further developments in this area.

Moreover, the current simulation and algorithms were implemented in Python, but performance could be significantly improved by switching to C++. Python, while suitable for prototyping, is slower due to its interpreted nature. Using C++, a compiled language, would enhance computational efficiency, especially for real-time calculations, trajectory planning, and multi-robot coordination, and better handle resource-intensive tasks like path recalculations and communication. Additionally, it is important to note that the communication protocol was not fully implemented in this work, which means that its complete functionality and impact on system performance could not be fully evaluated.

Finally, while significant progress has been made in trajectory planning, much remains to be done in the mathematical optimization of path planning. Future work will focus on refining the mathematical models to ensure better accuracy, scalability, and optimality of the planned paths. This includes further work on optimizing energy consumption, minimizing travel time, and addressing constraints imposed by the environment or robot capabilities.

Perspectives

To further develop this work, I aim to explore another way to acheive optimal path. As seen earlier, global path optimization was not entirely successful. To address this, I simplified the problem and focused on a new approach that frames it as a control problem from both a mathematical and empirical perspective. Both aspects are still under development and must be implemented in the simulator later.

Local Path Theory

First Approach: Mathematical Optimization

In this section, we seek to determine the optimal control commands that allow the robot to navigate toward a local waypoint without collisions while adhering to its physical constraints.

Here, "optimal" refers to minimizing energy consumption when moving from point A to point B. Since the robot operates with DC motors, power consumption is proportional to voltage (which affects rotation speed) and current (which influences torque). Assuming a flat surface and constant torque, current remains unchanged, meaning energy consumption is primarily determined by wheel speed. The energy expenditure for movement is defined as:

$$E(T) = \int_0^T \|\omega(t)\| \, dt \tag{4}$$

where

$$\|\omega(t)\| = \sqrt{\omega_L^2(t) + \omega_R^2(t)} \tag{5}$$

and

$$\omega_{L,R} : \mathbb{R}_+ \longrightarrow [-\omega_{max}, \omega_{max}] \tag{6}$$

Two constraints must be satisfied: the robot's initial position $X(0)$ must match its current position X_R, and its final position $X(T)$ must correspond to the waypoint X_{WP}.

In subsubsection V.3.1, we derived the system dynamics as $\dot{X} = F(\omega)$.

The goal is then to optimize $E(T)$.

Note: If the current were not constant, energy consumption would depend on the terrain characteristics. In this case, energy would be defined as:

$$\tilde{E}(T) = \int_0^T \|U(t)I(t)\| dt \tag{7}$$

where $U(t)$ and $I(t)$ vary based on environmental conditions. This would significantly increase the problem's complexity.

The challenge remains identical to that in global path optimization: How can we ensure that $X(T) = X_{WP}$?

Second Approach: Empirical Optimization

The second approach involves determining the set of all possible maximal paths that the robot can follow within a given time frame. By reversing the problem, we can infer the optimal control commands needed to reach the waypoint efficiently.

Initially, I focus on identifying the maximal paths using stochastic methods, see Figure V.17a. Then, I compute all feasible path combinations for a given number of distinct control inputs, see Figure V.17b. This approach provides a data-driven way to refine the navigation strategy, complementing the theoretical optimization.
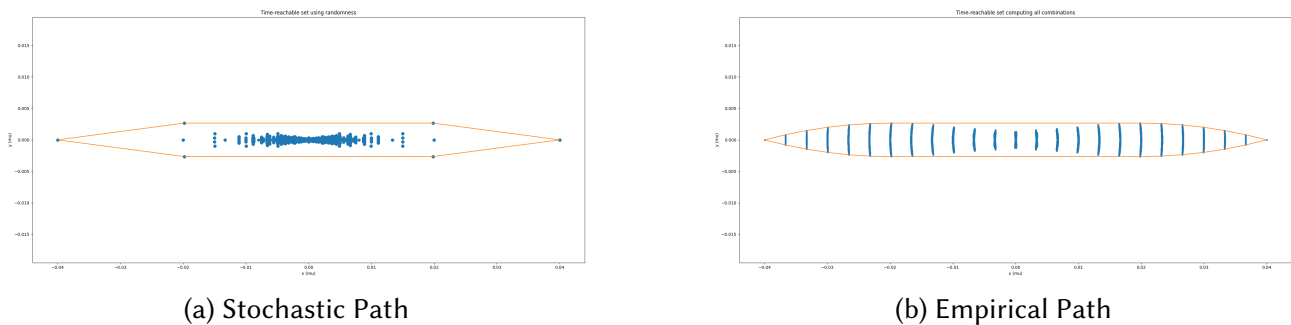

(a) Stochastic Path

(b) Empirical Path

Figure V.17: Comparison of Stochastic and Empirical Path Optimization Approaches

Results were better simulating all combinations possible.

ROS Implementation

Possible future work includes implementing the simulator within a Robot Operating System (ROS) architecture, integrated with RViz or Gazebo for visualization and testing. ROS is a middleware that provides a flexible framework for developing robotic applications by offering tools, libraries, and conventions to simplify communication between different robotic components.

By implementing the simulator in ROS, we can enhance modularity, allowing seamless integration of additional sensors, actuators, and real-world robotic hardware. Gazebo, as a high-fidelity physics simulator, would enable realistic testing of the robot's movement and interaction with dynamic environments, while RViz would facilitate real-time visualization of sensor data and path planning.

Moreover, integrating ROS would allow for better compatibility with existing robotic platforms, making it easier to transition from simulation to real-world deployment. Future work could also explore ROS2 for improved performance, scalability, and real-time capabilities, further enhancing multi-robot coordination and communication.

The energy consumption shoulf also be implemented to account for the robot's real-time power usage, considering factors such as motor efficiency, battery limitations, and terrain characteristics.

PhD oppening

All these open areas in my work will be essential components of my future research within the framework of my PhD. The main objective will be to design optimal trajectory planning algorithms for the recovery of underwater drone swarms. This work represents the first step into the world of robotics and mathematics research, a field I will continue to explore and develop during my PhD, which is currently in the process of securing funding.

References

[1] S. M. LaValle, <u>Planning Algorithms</u>. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[2] N. Geographic, "Greenland's secret caves: Inside the hidden world of ice," 2025, accessed on 10 January 2025. [Online]. Available: https://www.nationalgeographic.com/science/article/greenland-secret-caves-exploration

[3] R. Scalea, M. Rodrigues, D. P. Moya Osorio, C. Lima, R. Souza, H. Alves, and K. Castelo Branco, "Opportunities for autonomous uav in harsh environments," 08 2019, pp. 227–232.

[4] H. t. Dang, "Underwater robots for karst and marine exploration : A study ofredundant auvs," Ph.D. dissertation, 2021, thèse de doctorat dirigée par Lapierre, Lionel SYAM - Systèmes Automatiques et Micro-Électroniques Montpellier 2021. [Online]. Available: http://www.theses.fr/2021MONTS038

[5] P. Kambesis, "The importance of cave exploration to scientific research," <u>Journal of Cave and Karst Studies</u>, vol. 69, 04 2007.

[6] K. Otsu, S. Tepsuporn, R. Thakker, T. Vaquero, J. Edlund, W. Walsh, G. Miles, T. Heywood, M. Wolf, and A.-A. Agha-Mohammadi, "Supervised autonomy for communication-degraded subterranean exploration by a robot team," 03 2020, pp. 1–9.

[7] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, "Graph-based subterranean exploration path planning using aerial and legged robots," <u>Journal of Field Robotics</u>, vol. 37, no. 8, pp. 1363–1388, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21993

[8] M. Dharmadhikari, H. Nguyen, F. Mascarich, N. Khedekar, and K. Alexis, "Autonomous cave exploration using aerial robots," in <u>2021 International Conference on Unmanned Aircraft Systems (ICUAS)</u>, 2021, pp. 942–949.

[9] G. Zhang, B. Shang, Y. Chen, and H. Moyes, "Smartcavedrone: 3d cave mapping using uavs as robotic co-archaeologists," 06 2017, pp. 1052–1057.

[10] A. Husain, H. Jones, B. Kannan, U. Wong, T. Pimentel, S. Tang, S. Daftry, S. Huber, and W. Whittaker, "Mapping planetary caves with an autonomous, heterogeneous robot team," 03 2013, pp. 1–13.

[11] P. Petráček, V. Krátký, M. Petrlík, T. Báča, R. Kratochvil, and M. Saska, "Large-scale exploration of cave environments by unmanned aerial vehicles," <u>IEEE Robotics and Automation Letters</u>, vol. PP, pp. 1–1, 07 2021.

[12] A. Gupta, A. Abdullah, X. Li, V. Ramesh, I. Rekleitis, and M. Islam, "Demonstrating cavepi: Autonomous exploration of underwater caves by semantic guidance," 02 2025.

[13] A. Abdullah, T. Barua, R. Tibbetts, Z. Chen, M. Islam, and I. Rekleitis, "Caveseg: Deep semantic segmentation and scene parsing for autonomous underwater cave exploration," 05 2024, pp. 3781–3788.

[14] L. Jean-Claude, <u>Robot Motion Planning</u>. Boston, MA: Springer US, 1991. [Online]. Available: http://link.springer.com/10.1007/978-1-4615-4022-9

[15] B. Aneeta, S. Ekta, and D. Bhaskar, "Robot path planning using silhouette method," in <u>13th National Conference on Mechanisms and Machines</u>, January 2008, pp. 12–13.

[16] L.-P. Tomás and W. M. A., "An algorithm for planning collision-free paths among polyhedral obstacles," <u>Communications of the ACM</u>, 1979.

[17] G. Santiago, M. Luis, A. Mohamed, and M. Fernando, "Path planning for mobile robot navigation using voronoi diagram and fast marching," in <u>2006 IEEE/RSJ International Conference on Intelligent Robots and Systems</u>, 2006, pp. 2376–2381.

[18] Z. David and L. Jean-Claude, "New heuristic algorithms for efficient hierarchical path planning," <u>IEEE Transactions on Robotics and Automation</u>, vol. 7, pp. 9–20, 1991. [Online]. Available: https://api.semanticscholar.org/CorpusID:21438079

[19] K. K. and S. M., "An efficient motion-planning algorithm for a convex polygonal object in two-dimensional polygonal space," <u>Discrete and Computational Geometry</u>, vol. 5, no. 1, pp. 43–76, 1990. [Online]. Available: http://eudml.org/doc/131106

[20] A. Francis, B. Jean-Daniel, and F. Bernard, "A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles," in <u>Proceedings of the 1988 IEEE International Conference on Robotics and Automation</u>, vol. 3, 1988, pp. 1656–1661. [Online]. Available: https://api.semanticscholar.org/CorpusID:37779065

[21] K. Y. and B. J., "Potential field methods and their inherent limitations for mobile robot navigation," in Proceedings of the 1991 IEEE International Conference on Robotics and Automation, vol. 2, 1991, pp. 1398–1404.

[22] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). IEEE, 1996, pp. 113–120.

[23] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," The International Journal of Robotics Research, vol. 21, no. 3, pp. 233–255, 2002.

[24] C. Nissoux, T. Siméon, and J.-P. Laumond, "Visibility based probabilistic roadmaps," Advanced Robotics, vol. 13, no. 2, pp. 223–244, 1999.

[25] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," Mechanisms and Machine Science, vol. 29, pp. 3–27, 03 2015.

[26] M. S. d. Silva, L. Clark, V. Thangavelu, J. Edlund, K. Otsu, G. Correa, V. Varadharajan, A. Santamaria, T. Touma, A. Bouman, H. Melikyan, T. Pailevanian, S.-K. Kim, A. Archanian, T. Vaquero, G. Beltrame, N. Napp, G. Pessin, and A.-a. Agha-mohammadi, "Achord: Communication-aware multi-robot coordination with intermittent connectivity," IEEE Robotics and Automation Letters, vol. 7, pp. 1–8, 10 2022.

[27] F. Klaesson, P. Nilsson, T. S. Vaquero, S. Tepsuporn, A. Ames, and R. M. Murray, "Planning and optimization for multi-robot planetary cave exploration under intermittent connectivity constraints," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:231797539

[28] L. Clark, J. A. Edlund, M. S. Net, T. S. Vaquero, and A. akbar Agha-mohammadi, "Propem-l: Radio propagation environment modeling and learning for communication-aware multi-robot exploration," 2022. [Online]. Available: https://arxiv.org/abs/2205.01267

[29] J. Banfi, A. Quattrini Li, I. Rekleitis, F. Amigoni, and N. Basilico, "Strategies for coordinated multirobot exploration with recurrent connectivity constraints," Autonomous Robots, vol. 42, 04 2018.

[30] E. Trélat, Contrôle optimal : théorie et applications, 2nd ed., ser. Mathématiques Concrètes. Paris, France: Vuibert, 2008, version électronique mise à jour en octobre 2023 avec ajout d'exercices et correction de misprints.

[31] C. Calvo Tapia, J. Villacorta-Atienza, V. Mironov, V. Gallego, and V. Makarov, "Waves in isotropic totalistic cellular automata: Application to real-time robot navigation," Advances in Complex Systems, vol. 19, p. 1650012, 12 2016.

[32] S. Chen and G. D. Doolen, "Lattice Boltzmann Method for Fluid Flows," Annual Review of Fluid Mechanics, vol. 30, pp. 329–364, Jan. 1998.

[33] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische mathematik, vol. 1, no. 1, pp. 269–271, 1959.

[34] Wikipedia, "Dijkstra's algorithm," 2025, accessed on 17 February 2025. [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Algorithm

[35] ——, "Bresenham's line algorithm," 2025, last accessed on 19 February 2025. [Online]. Available: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

[36] S. Ding, T. Zhang, M. Lei, H. Chai, and F. Jia, "Robust visual-based localization and mapping for underwater vehicles: A survey," Ocean Engineering, vol. 312, p. 119274, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002980182402612X

Annexes