Multi-robot coordination

Autonomous exploration of gallery networks

Engineering Graduation Project

Seatech 3A - MOCA

Auteur :
Fabien MATHÉ

Enseignant :
M. Mehmet ERSOY

Abstract

Mots-clés:

Mots-clés

Abstract

Keywords:

Keywords

Remerciements

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

L'un des ouvrages de référence utilisés dans ce travail est le livre de S. M. LaValle, *Planning Algorithms* [1], que je remercie particulièrement pour son engagement à rendre ses travaux accessibles au public (`https://lavalle.pl/`).

I.1 État de l'Art : Exploration Multi-Robot de Cavités

L'exploration des cavités est essentielle pour la cartographie souterraine, la recherche scientifique et les interventions d'urgence, tout en posant des défis uniques pour la robotique autonome.[2] Leur exploration multi-robot est un domaine de recherche en plein essor dans le domaine de la robotique, avec des applications dans des environnements variés tels que l'exploration spatiale, l'exploration de grottes, les missions de sauvetage dans des environnements urbains sinistrés, ou encore l'extraction minière.[3, 4] Ces environnements, souvent complexes et dynamiques, présentent de nombreux défis pour la planification de mission, notamment des obstacles imprévisibles, des zones inaccessibles pour les robots mobiles, et l'absence de signal GPS. De ce fait, l'exploration multi-robot permet de tirer parti de la coopération entre robots pour surmonter ces difficultés.

I.1.1 Défis dans l'exploration multi-robot de cavités

L'exploration de cavités avec plusieurs robots implique plusieurs défis techniques. En particulier, la gestion de la coopération entre robots et la gestion de l'information dans des environnements complexes sont deux aspects fondamentaux de ce type de mission. Les défis peuvent être classés en plusieurs catégories :

I.2 État de l'Art des Méthodes de Communication Inter-Robot dans un Environnement de Cavité

La communication inter-robot (CIR) dans un environnement de cavité est un défi majeur pour les systèmes multi-robots, en raison des conditions particulières que ces environnements présentent, telles que des espaces confinés, des obstacles physiques et des perturbations qui peuvent affecter les signaux de communication. Les méthodes de CIR peuvent être classées en différentes catégories selon la technologie utilisée et la stratégie de communication adoptée. Les approches classiques reposent principalement sur des réseaux sans fil, tels que la communication par radiofréquence (RF), qui est couramment utilisée pour des applications de communication à longue portée mais qui peut souffrir de limitations dans les cavités où les signaux sont atténués par des parois solides. Dans ce contexte, des méthodes de communication ad hoc sont souvent employées, où les robots créent un réseau dynamique de relais pour échanger des informations. Une approche courante est l'utilisation de la *communication par maillage*, dans laquelle chaque robot agit comme un relais, permettant ainsi une couverture étendue et une transmission de données entre robots même lorsque les obstacles interfèrent avec les signaux directs. La *communication acoustique* ou *ultrasonique*, qui repose sur des ondes sonores, est une alternative viable dans les environnements de cavité, où elle peut transmettre des informations de manière robuste, notamment pour des distances plus courtes ou dans des environnements très confinés. De plus, la communication optique (par exemple, la *communication par lumière visible ou infrarouge*) devient de plus en plus populaire dans des applications de haute précision dans des cavités étroites, car elle est peu sensible aux interférences électromagnétiques et permet des transmissions rapides et sécurisées. Une autre stratégie consiste à utiliser des protocoles de communication coopérative, où les robots collaborent pour optimiser le flux d'informations. Des techniques de *routage adaptatif* et de *partitionnement dynamique de réseau* sont souvent utilisées pour gérer les obstacles et les interférences dans les réseaux multi-robots. Enfin, les méthodes de *communication opportuniste*, basées sur des échanges de données ponctuels lorsque la ligne de visée est dégagée, sont particulièrement adaptées aux environnements où la connectivité est intermittente et où les robots doivent s'organiser pour minimiser les interruptions dans le flux de données. L'un des principaux défis reste de maintenir une communication fiable et efficace, notamment en optimisant les stratégies de répartition de la bande passante, d'allocation des ressources et de gestion des interférences dans des espaces très contraints. Des approches récentes cherchent à intégrer des systèmes hybrides combinant ces technologies pour assurer une meilleure résilience face aux conditions environnementales changeantes et aux mouvements des robots.

I.2.1 Planification de chemin, de trajectoire et évitement d'obstacle

La planification de chemin pour un système méchatronique constitue le fondement de tous les systèmes mobiles autonomes, qu'il s'agisse de drones ou de bras de manutention et d'assemblage. Le principe de la planification de chemin ou de trajectoire est de déterminer une solution - un chemin ou une trajectoire - reliant un point de départ à un point cible.

La distinction entre planification de chemin et planification de trajectoire réside dans le fait qu'un chemin planifié n'est pas nécessairement réalisable par un robot. En effet, la planification de chemin ne prend pas toujours en compte la faisabilité physique ou cinématique pour un robot mobile.

L'évitement d'obstacles est une contrainte essentielle dans ces deux approches. Les obstacles définissent les zones inaccessibles, et comme nous le verrons par la suite, leur nature - mobile ou immobile - détermine en grande partie la méthode à employer pour résoudre le problème de planification.

Parmis les méthodes déterministe, on trouve une large variete de méthodes principalement basées sur trois approches différentes. Les approches par graphs, celle de décomposition célullaire et enfin les celles utilisant des champs potentiels.[5, 6]

La méthodes des graphs consistent à construire une carte des chemins empruntable en partant des obstacles de la scène. Parmis ces méthodes utilisant des graphs, on peut distinguer quatres types differents : Les graph de visibilité[7], les diagrammes de Voronoï [8] ou encore la méthode des Silhouette[6].

Les méthodes associés au decomposition celulaire consistent à diviser l'espace libre du robot en régions simples, appelées cellules, où il est facile de générer un chemin entre deux configurations. Un graphe représentant les relations d'adjacence entre les cellules est ensuite construit et exploré.[5, 9, 10, 11]

Une autre méthode repose sur une subdivision fine de l'espace afin de repérer les zones libres. La méthode des champs potentiels s'appuie sur cette idée en définissant des potentiels qui traduisent des forces d'attraction, dirigées vers les coordonnées cibles, et de répulsion, correspondant par exemple aux obstacles. Le chemin est ensuite déterminé en suivant l'opposé du gradient du potentiel total ainsi calculé.[5, 12]

Des approches alternatives ont été développées dans les années 1990 et 2000, notamment les méthodes stochastiques de planification de chemin RPP (*Random Path Planners*) et PRM (*Probabilistic Roadmap Planners*). Ces méthodes consistent à échantillonner l'espace de manière aléatoire afin de créer un graphe de chemins possibles (*roadmap*) dans cet espace.[13, 14, 15]

L'un des principaux atouts de cette méthode de planification de chemin est qu'elle ne dépend ni de la structure de l'espace, ni du nombre d'obstacles, ni de leur disposition. Une fois la *roadmap* créée, il suffit d'utiliser une méthode de recherche de chemin dans un graphe pour trouver le chemin menant au point objectif.[16]

II Partie 2

II.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.

II.2 Creation of the map

The map consist in succession of straight lines generate by a simple cellular automata.

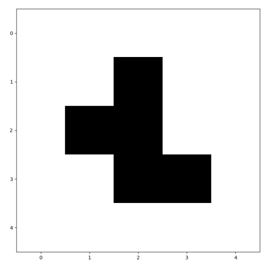
A cellular automaton is a grid of cells, where each cell can exist in different states based on predefined rules. The concept was developed by Stanislaw Ulam and John von Neumann in the 1940s.

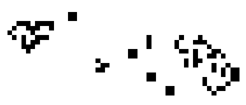
The most famous cellular automaton, which helped popularize its use, was developed by John Conway: *Conway's Game of Life*. This model follows a set of four simple rules, which can be found here. The rules are based on the properties of neighboring cells.

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 II.1a, the system evolves into the states illustrated in Figure II.1 at generations 87 and 263.


(a) Iteration 0


(b) Generation 87

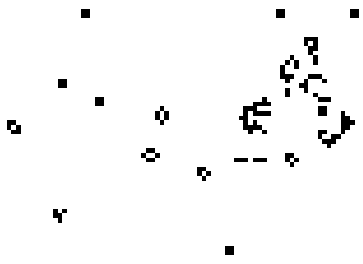

(c) Generation 263

Figure II.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 II.2. The maps generated after this transformation improved in quality.

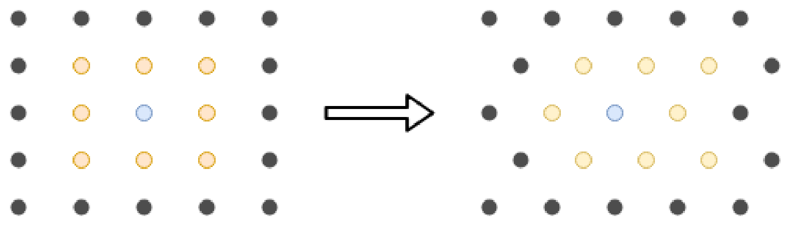


Figure II.2: Grid transformation

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

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 II.3 illustrates all possible states of an element composed of three cells.

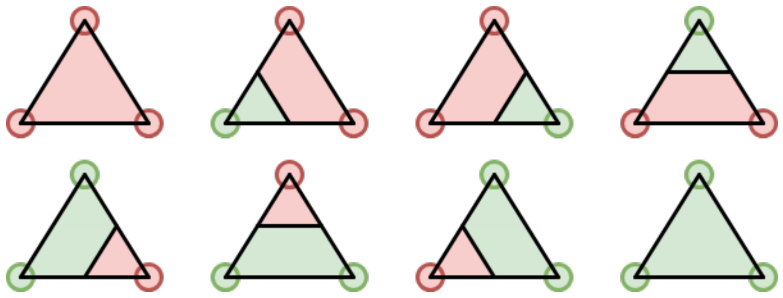


Figure II.3: Isolines, possible states of a triangular element

At the end of both steps—transformation and Marching Squares—we obtain the following schematic representation:

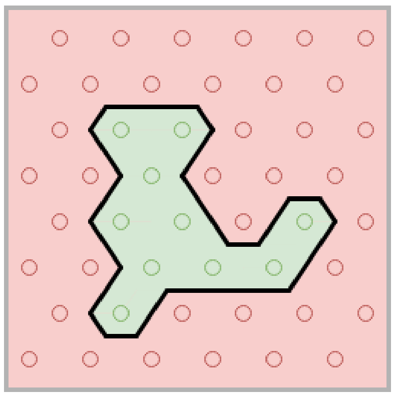
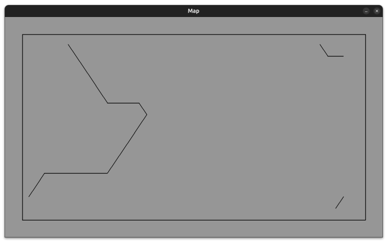
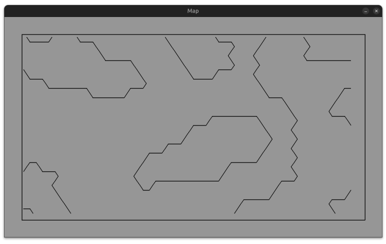


Figure II.4: Exemple scheme of map generated

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 II.5 illustrate some examples.


(a) $\delta x = 100$ mu


(b) $\delta x = 40$ mu

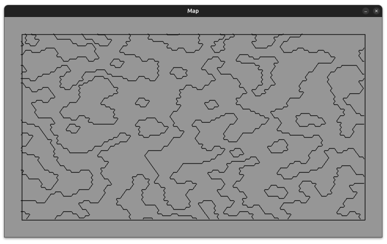

(c) $\delta x = 10$ mu

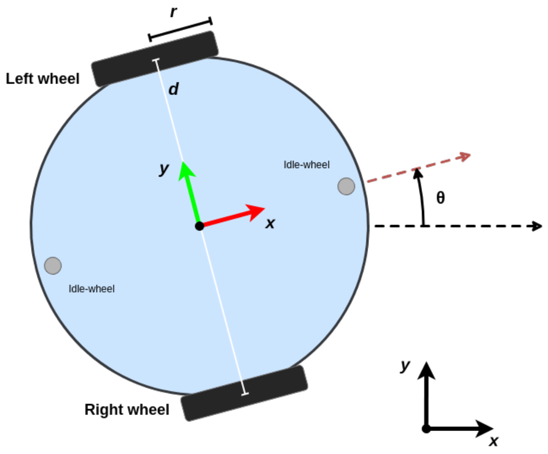
Figure II.5: Different maps made with equilateral triangles

TO BE DONE

II.3 Robot implementation

II.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 II.6a and Figure II.6b.



(a) Differential drive robot diagram

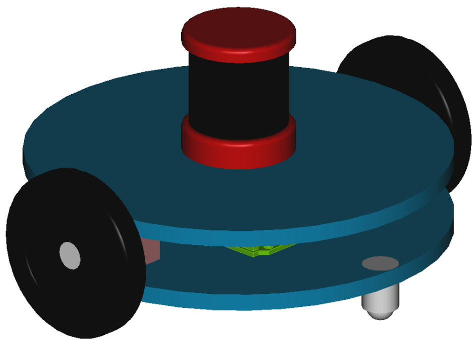

(b) Differential drive robot 3D model

Figure II.6: Robot models

Notations:

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

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 :

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

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 diveded by two of the two gives the angular speed of the robot.

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

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

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

II.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 II.7.

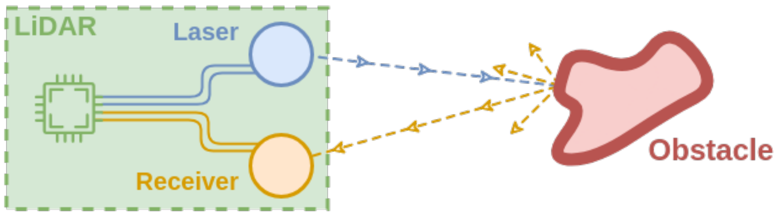


Figure II.7: LiDAR operating diagram

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.

II.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.[17]

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 II.8. 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.

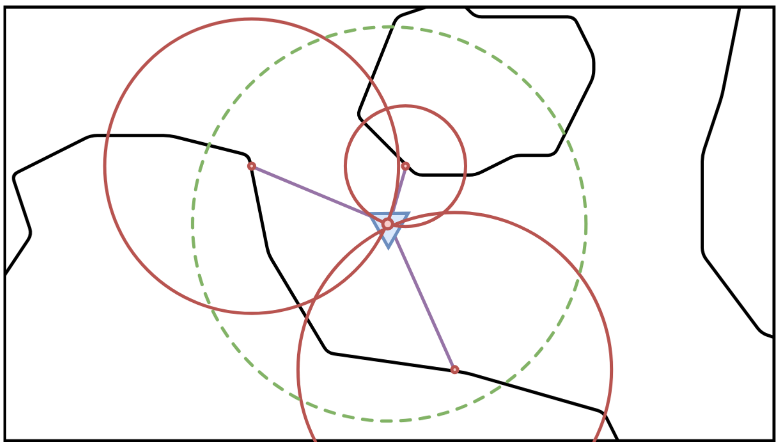


Figure II.8: Correct robot position using LiDAR data diagram

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 II.9, 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.

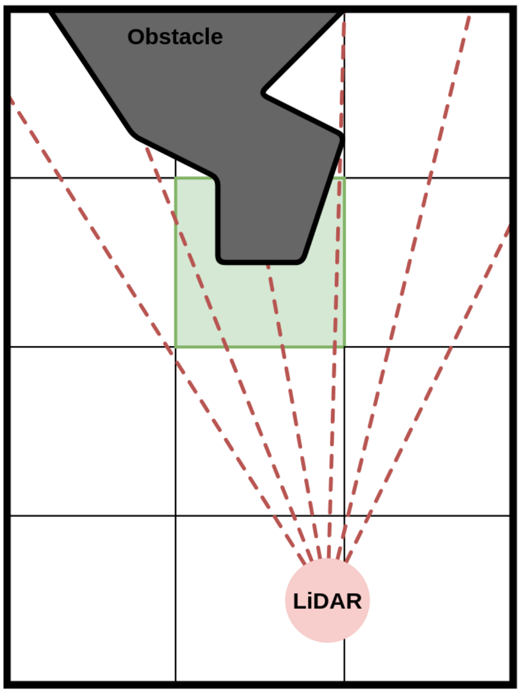
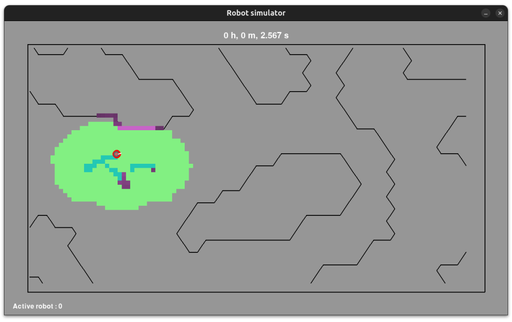
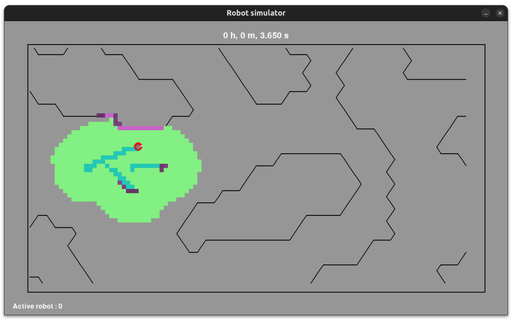


Figure II.9: Importance of asymetry in adding and subtracting in the occurence map

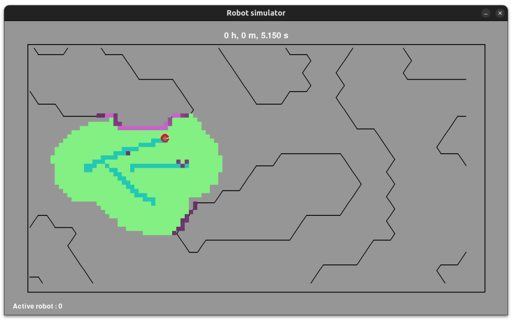
Figure II.10 illustrates the map updates over time with three robots. The purple squares represent occupied cells in the live grid map (dark purple = 100, light purple = 200). Green areas indicate free spaces, gray areas are unknown, black lines outline the map, and the small red dots represent the robots:



(a) Map update at time t_1



(b) Map update at time t_2



(c) Map update at time t_3

Figure II.10: Map updates over time

III Partie 3

III.1 Global path theory

In 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 $\Omega \in \mathbb{R}^2$, and let $\mathbf{X}(t)$ be the coordinates of the robot at time t in this space. Let $FV(t)$ be the field of vision of the robot at time t.

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

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

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

Here,

$$R(t,\theta) = \min(\text{distance}(\,\mathbf{X}(t),\ L(\theta)\cap W\,))$$

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

$$W = \{\,\text{Segment}(\Omega)\,\}$$

We define the field of vision of the robot as:

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

In other words, $FV(t)$ is the set of points in Ω present in a disk of radius R_{max} and located between the robot and the nearest intersection with a wall.

We now construct the functional that we will later seek to optimize, which is related to the movement of the robot.

We define KM (*Known Map*) as the space of the map known by the robot and EM (*Explorable Map*) as the part of Ω explorable by 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 $t = T$.

T is the time at which the map is explored to the maximum capacity of the robot, i.e., $KM = EM$.

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.

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

III.2 Local path theory

III.2.1 First approach: mathematical approach

In this section, we focus on determining the optimal commands for navigating to a local waypoint obstacle-free while adhering to the robot's constraints.

Optimal in this context means using the least amount of energy to move from point A to point B. Considering the robot moves using DC motors, the power consumption is proportional to the voltage for rotation speed and the current for the torque applied to the wheel. Assuming the floor is completely flat and the torque is constant, the current remains the same. Therefore, the energy consumption is only related to the wheel speed. We define the energy spent for moving as:

as a reminder, ω_L and ω_R are the speed of the left and right wheel.

$$E(T) = \int_0^T \|\omega(t)\|\, dt$$

with $\|\omega(t)\| = \sqrt{\omega_L^2(t) + \omega_R^2(t)}$ and $\omega_{L,R} : \mathbb{R}_+ \longrightarrow [-\omega_{max}, \omega_{max}]$

Two constraint remains, we want $X(0) = X_R$ and $X(T) = X_{WP}$

In the part **FIGURE ...**, we calculated $\dot{X} = F(\omega)$.

Note: If the current were not constant, we would need to account for the terrain characteristics and define the energy consumption as the product of voltage and current. In that case, $E(T)$ would become $\tilde{E}(T) = \int_0^T \|U(t)I(t)\| dt$, where $U(t)$ and $I(t)$ are functions dependent on the map characteristics. This would significantly complicate the problem.

III.2.2 Second approach: empirical approach

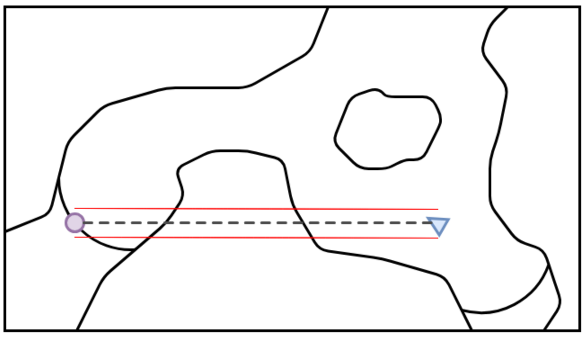
This second approach consists of determining the set of all possible maximal paths achieved by the robot in a certain amount of time. Reversing the problem should give us the optimal commands to reach the waypoint. In the first attempt, I focus on finding the maximal path using stochastic methods. Subsequently, I compute all possible combinations of paths with a given number of different commands.

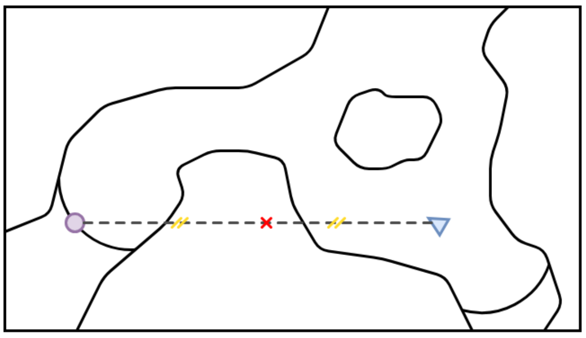
III.3 Introducing a new method for dynamic pathfinding

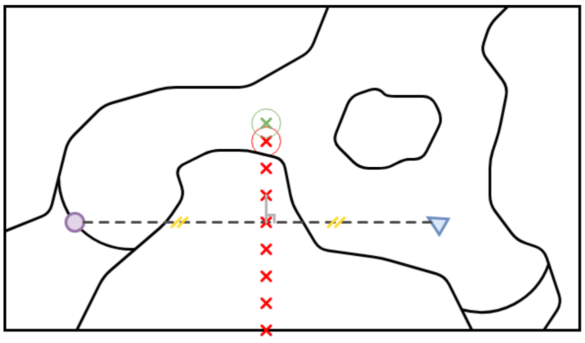
In this section, we introduce a new method for real-time dynamic pathfinding. This method involves inflating the path perpendicular to the shortest path to a point, i.e., a line. The line is split if it is not free from obstacles within a given safe range.

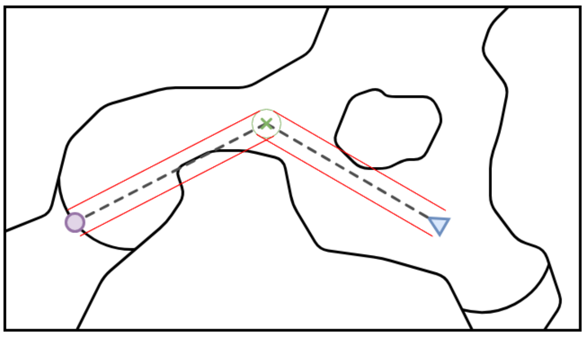
For our study, it is crucial that this algorithm runs in real-time to avoid any obstacles. Since the goal is to explore a map using multiple robots, we must ensure that the path computation time for each robot is minimal so that they can avoid each other.

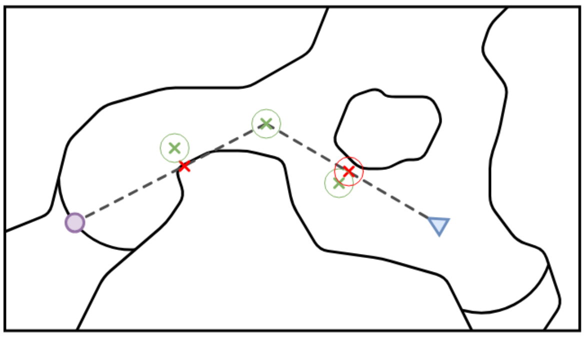
The principle is simple: draw a straight line between the robot and the waypoint (Figure III.1a). If the path is not free of obstacles within the defined safe range (Figure III.1b), split the path in the middle and shoot points perpendicular to the line (Figure III.1c). When a shot point is free from obstacles, validate the point and check if the two resulting lines are free from obstacles (Figure III.1d). If not, repeat the steps described above on the new lines (Figure III.1e). Once all points are safe, simplify the path using a straightforward algorithm. The process can be visualized on Figure III.1.


(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

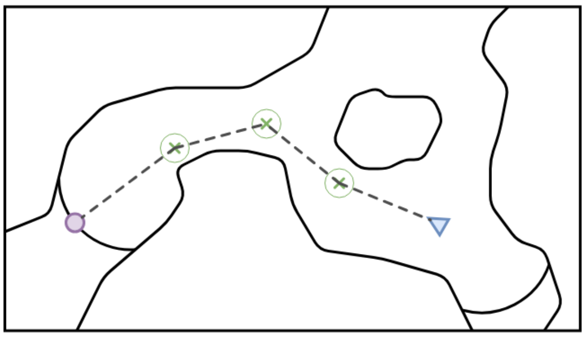

(f) Connect points

Figure III.1: Visualization of the dynamic pathfinding method

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.2.

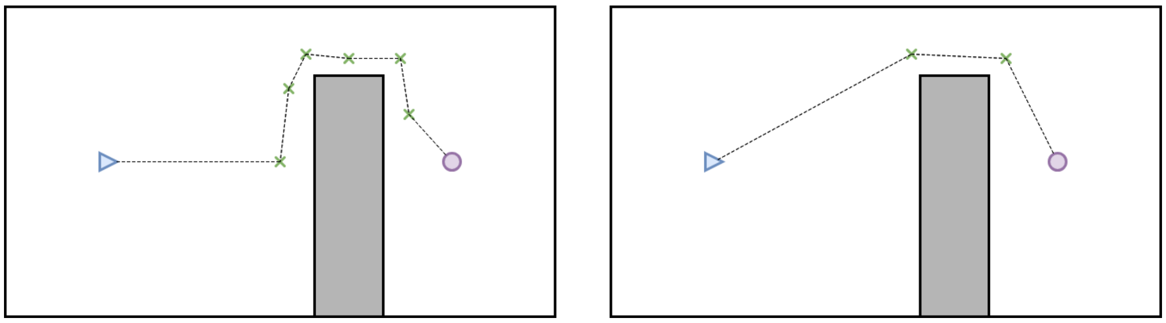


Figure III.2: Path simplification process

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.[18]

Differently from the part **FIGURE ...**, 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)[19]

Based on the work of *Calvo Tavia* and *al.*[18], 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:

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

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. $W_t \neq \emptyset$.:

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

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:

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

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

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.*[18]. 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.3. 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).

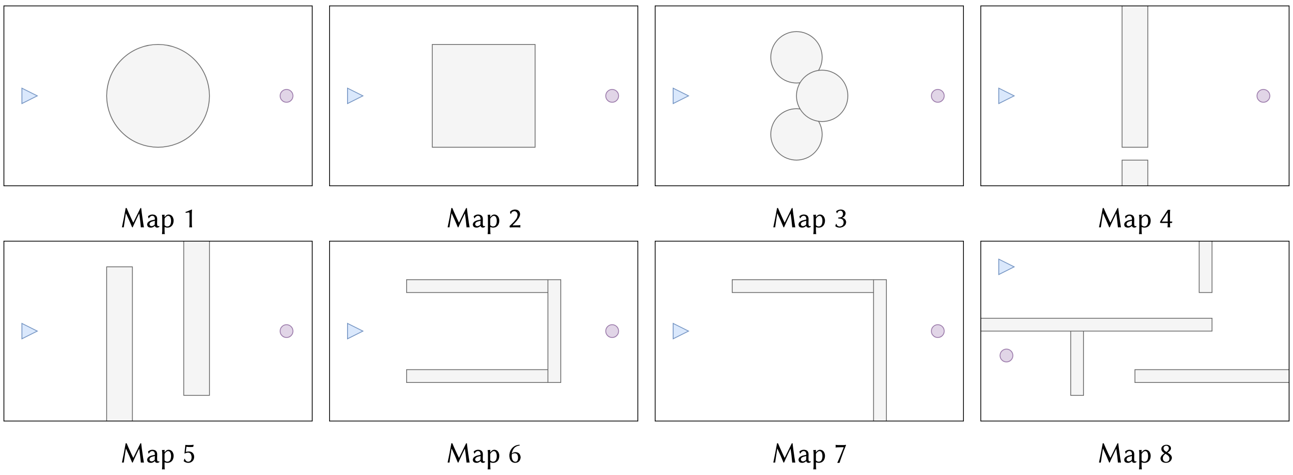


Figure III.3: Benchmark maps

Results

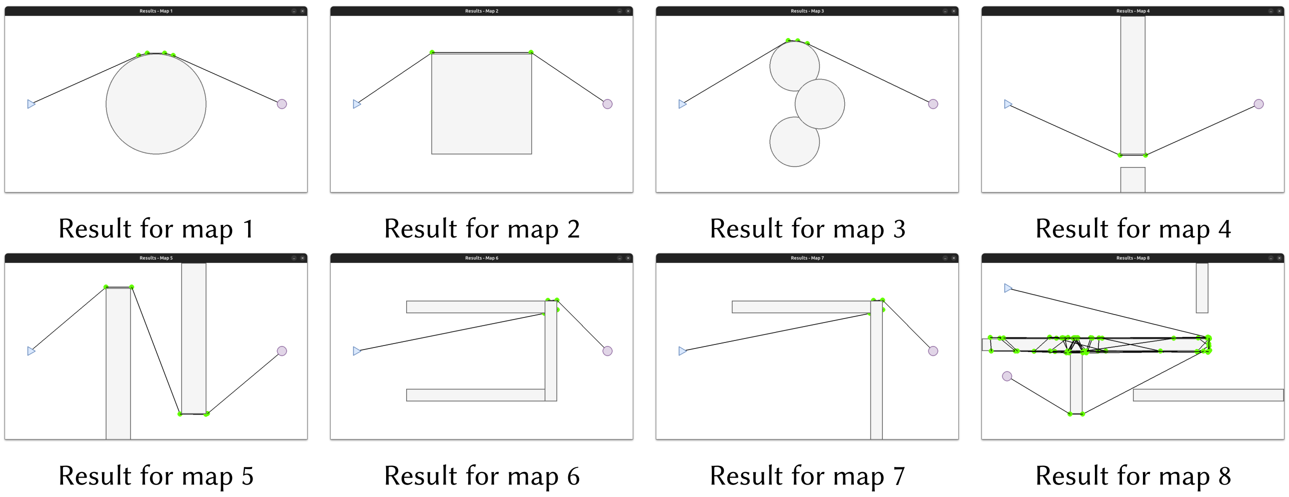


Figure III.4: Results on the benchmark maps

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.5 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.

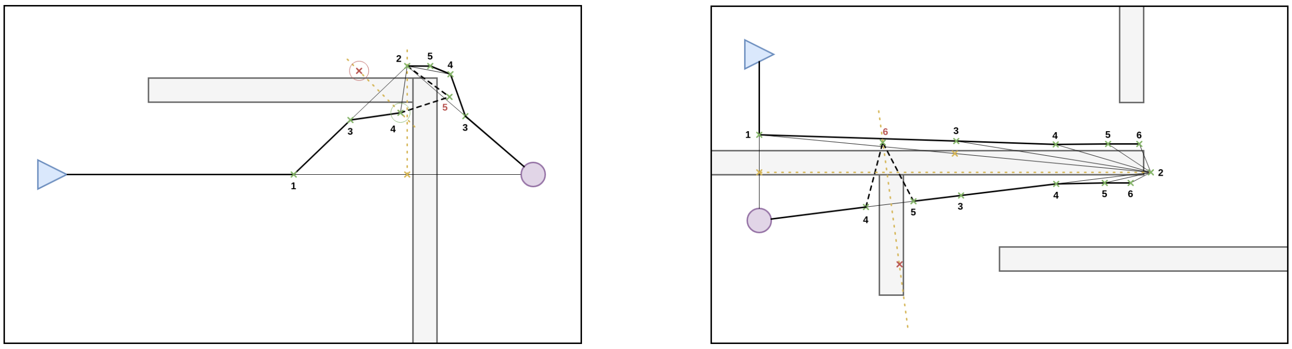

(a) Failure on map 7 (b) Failure on map 8

Figure III.5: Failure explanation for introduced method

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 uncertanty of ± 2 mu. The granularity is set to 5 for all cases without circular shape and 20 otherwise. **TO BE DONE**

Lenght of the path found in metric unit (mu)

Map	Theoretically	Method	Relative difference (%)	Cellular Automata (± 2 mu)	Error (%)
Map 1	1081	1084	0.3		
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	-			

Table III.1: Benchmark results for the new method

The Table III.1 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 precisly 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, the lattice approach for finding the shortest algorith is very good in this benchmark, within the range of uncertancy, the algorith always found the shortest path distance.

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.

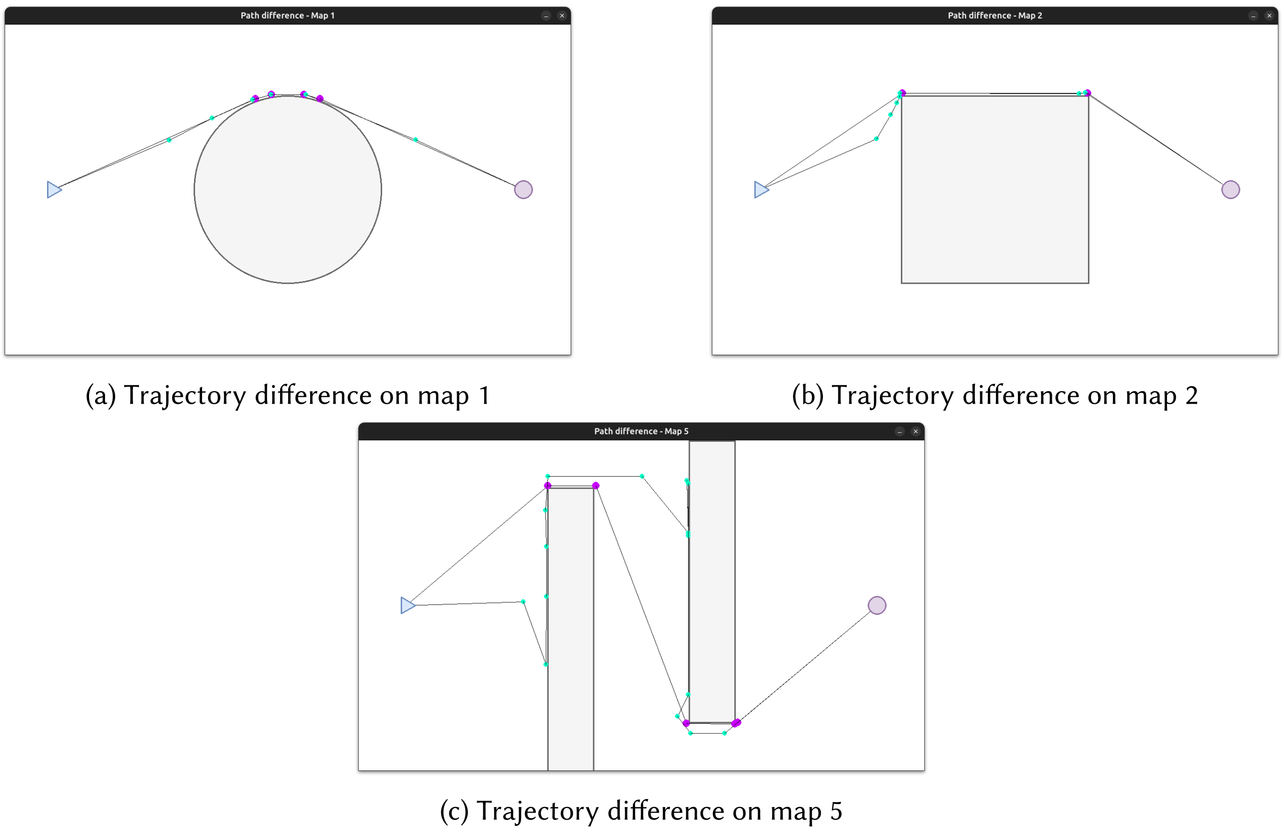
Dijkstra's algorithm is an algorithm to find the shortest path from a source to a final node in a weighted graph.[20]. It works by iteratively selecting the node with the smallest tentative distance, updating the distances to its neighbors, and marking it as visited. This process continues until all nodes have been visited or the shortest path to the target node is determined.[21]

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.

Perfomances

On an AMD Ryzen™ 5 3500U, using *Python* and the map in Figure II.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.6. This approach sacrifices the optimal path for reduced computational time. The difference in path can be significant, as shown in Figure III.6c. The path with the blue-green dots can illustrate how the robot navigates without prior knowledge of its environment before reaching the target point.



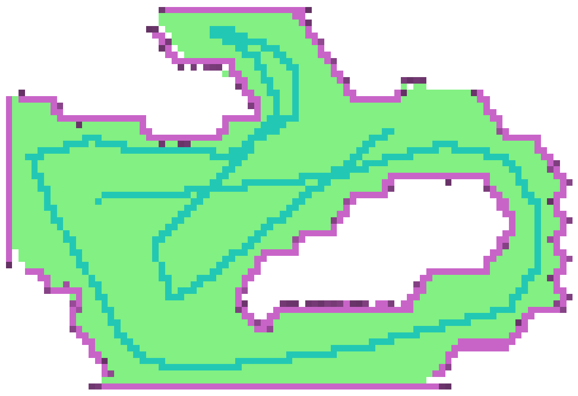
(a) Trajectory difference on map 1

(b) Trajectory difference on map 2

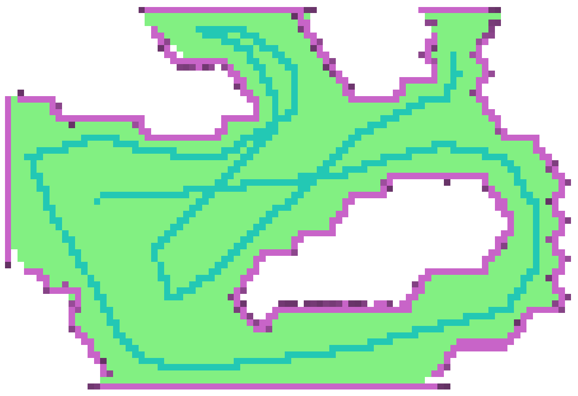
(c) Trajectory difference on map 5

Figure III.6: 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.7 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.7: Comparison of map exploration using global and local waypoint paths

III.7 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.7.1 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 suggested 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.

III.7.2 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.7.3 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.8 Global waypoint computing

Now the local navigation is set, we need to define how to select the next waypoint according to the map. First of all this is done by computing the frontiers of the know domains using image 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.9 shows an example of the method described above on the Figure III.8

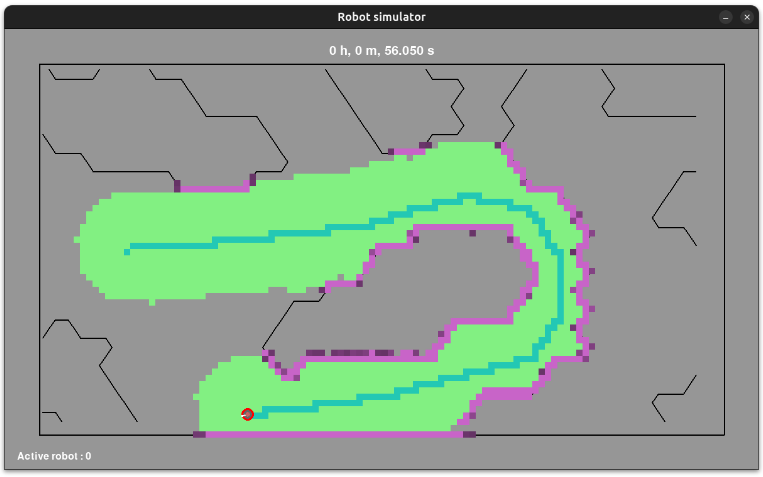
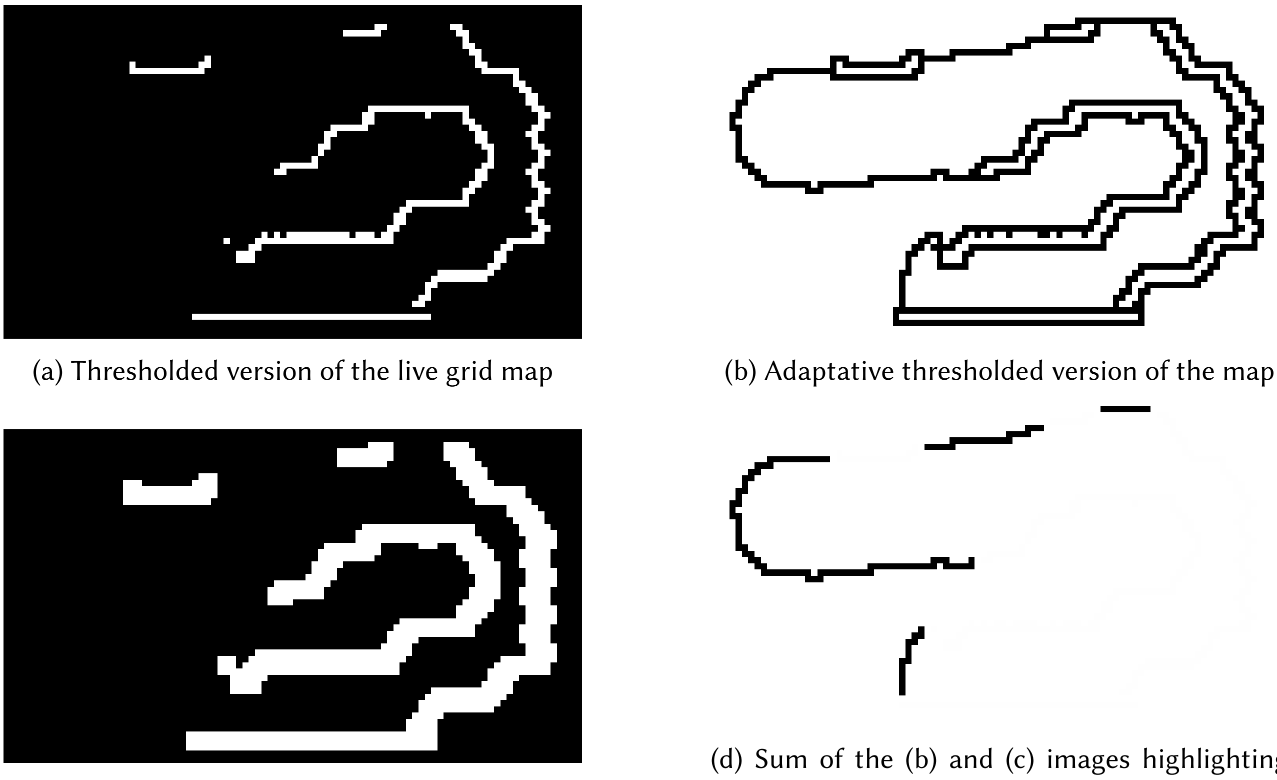


Figure III.8: Open frontiers of the map example



(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.9: 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.10 shows the weights, with only one out of every ten points displayed for clarity.

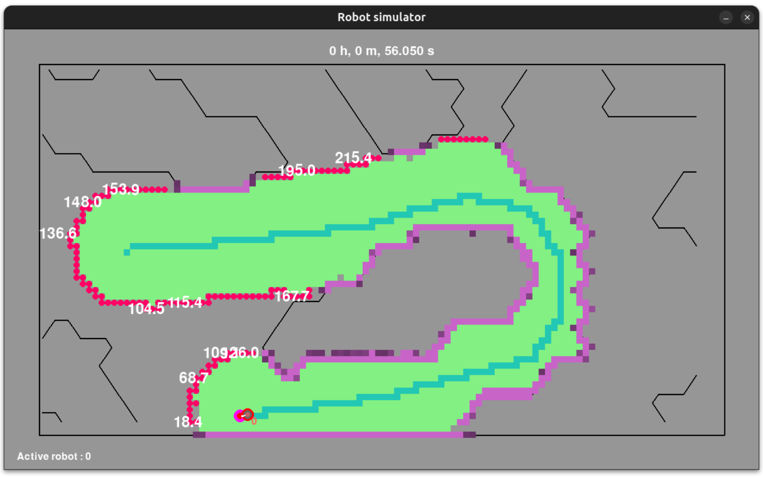


Figure III.10: Weighted frontiers of the map

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.

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.

$$\underbrace{\smile\smile\smile\smile}_{\text{ID of the sender}}\ \underbrace{\smile\smile\smile\smile}_{\text{Number of Receivers}}\ \underbrace{\smile\smile\smile\smile\cdots\smile\smile\smile\smile}_{4\times\text{Number of Receivers}}\ \underbrace{\smile\smile\smile\smile\cdots\smile\smile\smile\smile}_{\text{Message}}\ \cdots$$

$$\cdots\ \underbrace{\smile\smile\smile\smile}_{\text{Message end indicator}}\ \underbrace{\smile\smile\smile\smile}_{\text{Units sum}}\ \underbrace{\smile\smile\smile\smile}_{\text{Sender ID with parity bit}}$$

IV.3.1 Description

Emitter ID: $\smile\smile\smile\smile$

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.

IV.4 Dencoding

IV.5 Implementation

To implement it, we 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

Information sent for each message type could be found in the **Appendix ....**.

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.6 Possible robotics implementation

Robots communicate using different color channels. For instance, white could be used for broadcasting messages to all other robots. The overlapping of colors should not affect communication as long as each robot's color identifier is not a linear combination of the others.

We could use the SEN10656 4-channel color sensor for precise detection.

Each robot would be capable of emitting on all channels and receiving on a specific one.

V Partie 5

VI Partie à développer

Création du monde

- Automate cellulaire
- Méthode de square marching appliquée à des triangles

Capteurs utilisés

Minimalisme, quels capteurs sont nécessaire et suffisant pour accomplir la mission donnée ?

- Fonctionnement d'un lidar
- Accéléromètre

Planification de trajectoire

- **Différence avec la plannification de chemin (si existe ?)**
- Définition de la trajectoire optimale
- Description mathématique dans notre cas
- Manière de le résoudre, temps nécessaire, possibilités

Évitement d'obstacles

Essais

- Diagramme de Voronoi
- DFS (Depth-First Search)

Méthode par subdivision successive

- Isotropic waves generator to validate or reject the method
- Cellular automata

Création d'une carte de caractéristique sur un maillage cartésien uniforme

Planification dynamique de trajectoire

Contrôle du robot

- Cinématique du robot utilisé
- Modèle intégré

Temps de calcul

- Liste des optimisations
  - Subdivision de la carte
  - Détection d'obstacles par le Lidar
    - Fonction `move_on_line` (parallèle avec le computer graphics)

Non encore implémenté

- Sauvegarde dynamique de l'exploration

Conclusion

HUGE NOTE ON PERFORMANCES

Perspectives

ROS (Robot Operating System) implementation coupled with Rviz or Gazebo. Implementation on robot either to have both simulation and experimentation

Ouverture thèse...

References

[1] S. M. LaValle, Planning Algorithms. 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, consulté le 10 janvier 2025. [Online]. Available: https://www.nationalgeographic.com/science/article/greenland-secret-caves-exploration

[3] 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

[4] P. Kambesis, “The importance of cave exploration to scientific research,” Journal of Cave and Karst Studies, vol. 69, 04 2007.

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

[6] B. Aneeta, S. Ekta, and D. Bhaskar, “Robot path planning using silhouette method,” in 13th National Conference on Mechanisms and Machines, January 2008, pp. 12–13.

[7] L.-P. Tomás and W. M. A., “An algorithm for planning collision-free paths among polyhedral obstacles,” Communications of the ACM, 1979.

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

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

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

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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

[18] 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.

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

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

[21] Wikipedia, “Dijkstra’s algorithm,” 2025, consulté le 17 Fevrier 2025. [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra’s_algorithm#Algorithm

Annexes