People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of
the Requirements for the Degree of

MASTER

In **Electronics**

Option: **Computer Engineering**

Title:

Static and dynamic collision avoidance for multi-robots in crowded environment

Presented by:

- **ABAD Malik**
- **BOUHAMIDI Ahmed Essadik**

Supervisor:

Dr. BELAIDI Hadjira

Registration Number:……/2019

Abstract:

A team of multiple mobile robots that work in parallel offers a number of advantages over single robot systems. Multiple robots have the potential to finish a given task faster than a single robot and they are able to perform some tasks that are outside the scope of a single-robot system.

As a primary problem, motion-level conflict resolution requires that the robots avoid collisions not only with static obstacles but also with other robots. If a robot regards its neighboring robots as static obstacles, the conflict among the robots is inevitable in some cases. It becomes worse in crowded multi-robot environments.

Therefore, the use of multiple robots in the same workspace requires the necessity to coordinate between them. Coordination among the robots may be of two types, centralized and decentralized. In the centralized approach, a single robot acts as the coordinator which monitors the movement of robots and hence the goal accomplishment is centered round the coordinator. In the decentralized strategy, there is no single coordinator in the environment. Each robot coordinates its own movement and ensures that it does not collide with any other robot while goal accomplishment.

In this project, a new hybrid multi-robots navigation strategy in crowded environment (while avoiding static and dynamic obstacles) will be proposed. Thus, the possibility of collision between the pre-planned trajectories will be calculated by a centralized coordinator. Hence, the time to reach the possible collision points by each robot is calculated according to their speed. Consequently, the robots concerned by the collision have to take decision to avoid the collision by applying the technique of priority between robots.

Dedication

To my lovely parents, grandparents, my ante and my wife who always supported me in my studies.
To all my family and friends.
To all my friends who have been there for me, especially: Malik, Aimen and Younes.
To all people who helped and gave me the chance to be the person I am now.

BOUAHMIDI Ahmed Essadik

To all my family, especially my parents and my sister who supported me during my studies.

To all my friends, particularly to Fouad, Ahmed, Lokman, Aimen and Walid who helped me during hard times.

To all peoples who have been there for me and made me a better person.

ABAD Malik

Acknowledgement

We would like to express our gratitude to the many people who helped us during this project, to all those who provided support, talked things over, read, wrote, offered comments, allowed us to quote their remarks and assisted in the editing, proofreading and design.

We would like to thank our supervisor, Dr. Hadjira BELAIDI, for the patient guidance, encouragement and advices she has provided throughout our time as her students. We have been extremely lucky to have a supervisor who cared so much about our work, and who responded to our questions and queries so promptly. We would also like to thank all IGEE club members. In particular we would like to thank signal and systems lab's responsible, Pr. BENTARZI H., who made this project realizable.

And without forgetting Dr. Med SAHNOUN for his precious advices.

Last and not least, we beg forgiveness of all those who have been with us over the course of the years and whose names we have failed to mention.

Table of contents

List of tables

List of figures

GENERAL INTRODUCTION

Autonomous mobile robots has become significant nowadays. Mobile robots are a locomotive system, capable of moving from starting points to another. It may use different kinds of displacement systems such as wheels, tracks, legs or combination of them. This system should be equipped with sensors and actuators to execute different complex tasks, tasks that could be too precise, repetitive or dangerous where human beings can't be effective.

One of the most characteristics of mobile robots is moving safely, in other words, navigating along its trajectory without collision or crash with all kinds of obstacles. An intelligent control strategy is required, this strategy is called algorithms, what gives to the robots the ability of moving from one point to another by calculating optimal[1] safe path. This is why we focus on path planning to make an autonomous mobile robots system.

In another hand, another problem occurs. That problem is the dynamic or unknown obstacles that can't be avoided using a pre-planned algorithm. However, by developing the previous algorithm to take into consideration other robots in the same system, the risk of crash between robots will be reduced. This method gives us the possibility to pre-plan an optimal path without an inter-system[2] collision.

For outer-system dynamic/static obstacles, the problem can't be fixed with same algorithm, that why a hybrid-system[3] is used to ensure a safe navigation.

Swarm robot which is used in many warehouse automation is a perfect example to illustrate that. One major problem is faced during normal process which is optimal path finder, robots generally lack to deliver a good service in term of perfect placement or boxes pick and place through all the mazes of shelves and rooms for instance and its time execution.

One other problem which is strongly faced is intelligent parking where the system is exposed to abrupt and intense parking conditions, many cars may flow instantly and have

[1] The lowest cost path
[2] Robots running the same system
[3] A system that use pre-planned algorithm plus real time detection algorithm

to be guided to particular parking spots. In busy places like shopping or industrial districts, such a solution will help prevent traffic jams caused due to parking lot entrance, specially seen in densely populated cities. Also, there is a need to have a robust collision avoidance algorithm in place for safety of humans and robots.

The work done before used different approaches, such as, potential field merged with bug0 algorithm applied on one robot system [1], and also, multi-robot collision avoidance using RFID tags [2] Those works have solved some major problems of the path planning in single/multi-robots system, but it can be improved using a different algorithm which is more cost-effective[4].

To develop a cost-effective hybrid system, we propose a method that takes into account goal assignment for the robots and collision points between them in order to assign a better instruction that allows robots to take decision to avoid collisions. This is often proven to be beneficial as preassigned goals result in circuitous path and interchanging them with robots at a closer location is advantageous. Additionally, some tasks require a robot to operate in territories with unknown obstacles. These can result in completely new paths which are longer than previously accounted for. Therefore, priority assignment can be introduced to tasks with multiple robots, allowing robots to decrease speed or stop in order to respect priority path among themselves and minimize the time requires to find new path. Thus we attempt to solve a dynamic collision based on time to reach collision points.

This project is composed of four chapters. The first chapter introduces generalities about multi-robots navigation systems, path planning and the different obstacle avoidance algorithms. The second chapter introduces the hardware used for building the robots. The third chapter explains how the simulation part was performed by giving flowcharts and some examples. Finally, the last chapter presents the implementation part and discusses the results, in addition some features will be proposed for the aim of making a better system.

[4] Calculate the shortest path in a minimum time

CHAPTER I

Multi-robot generalities

Multi-robot are nowadays more sought in order to accomplish repetitive and difficult tasks (such as in warehouses, which will increase efficiency), since the more delicate task is, the more accuracy and efforts it needs.

That's why the choice of algorithm and resolution of path planning problems is crucial in order to achieve our needs.

1.1 Mobile Robots generalities

1.1.1 Definition

A mobile robot is an automatic system that is capable of locomotion; mobile robots have the capability to move around in their environment and are not fixed to one physical location. It may have wheels, tracks, legs or a combination of them [3].

This complex mechanical device may be equipped with sensors and actuators designed to perform complex tasks either autonomously or under supervision, such as tasks that are too precise for human beings, repetitive or dangerous [4].

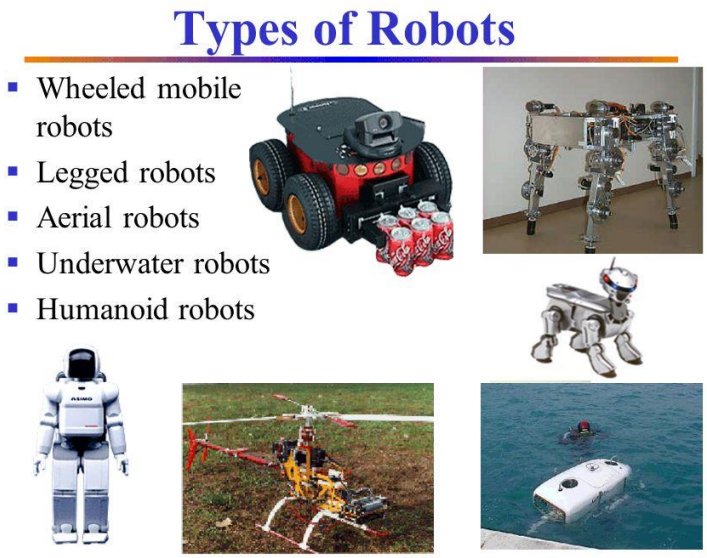


Figure 1.1: Different types of Robots

The most important characteristic of a mobile robot is, of course, to move, and be capable to guarantee its safety, to do that, it is necessary for the robot to move and navigate without colliding or clashing with any kind of obstacle. This safe navigation requires an intelligent control strategy, called Algorithms, capable of overcoming the

uncertainties presented by the real world. So, the robot must be able to move from one point to another by finding an efficient and safe path to avoid collision with the obstacles, that's why it is important to focus on path planning to guarantee the autonomy of mobile robots.

The purpose of path planning aims to generate a free path without collisions between a robot's starting and final configuration.

1.2 Path Planning Problem

The main task for motion planning is to compute a possible path for a robot from one configuration to another while avoiding obstacles [5].

For a mobile robot, in addition to avoid obstacles, the planner may be required to optimize certain objectives such as computing a path of the shortest length, shortest time or lowest energy consumption from one point to another while satisfying constraints determined by the vehicle dynamics or the environment.

Therefore, it is an essential task in the field of mobile robotics, which can be classified into two types: global path planning and local path planning.

For the Global Path Planning, the prior knowledge of the environment should be known. Many methods have been developed for global path planning, i.e., Voronoi graph, artificial potential field method, Dijkstra's algorithm, visibility search, grids, cell decomposition method, and so on.

In the other hand, for Local Path Planning, the robot can decide or control its motion and orientation autonomously using equipped sensors such as ultrasonic sensors, infrared range sensors, vision (camera) sensors, LiDAR … etc.

Since we are working in unknown obstacles (additional dynamic & static obstacles can be found) fuzzy logic, neural network, neuro-fuzzy, PSO algorithm, ant colony optimization algorithm, and simulated annealing algorithm, etc., are successfully employed by various researchers to solve the local navigation problem, some of them are used in sensor based systems in order to complete missing information using sensors mentioned before.

Here we are going to see the most used methods in path planning systems:

1.2.1 Artificial Potential Field

This approach considers that the robots rolls in a field of virtual forces. The latter is composed of two fields: the repulsive potential field around the obstacles and the field of attractive potential produced by the respective goals point for each robot.

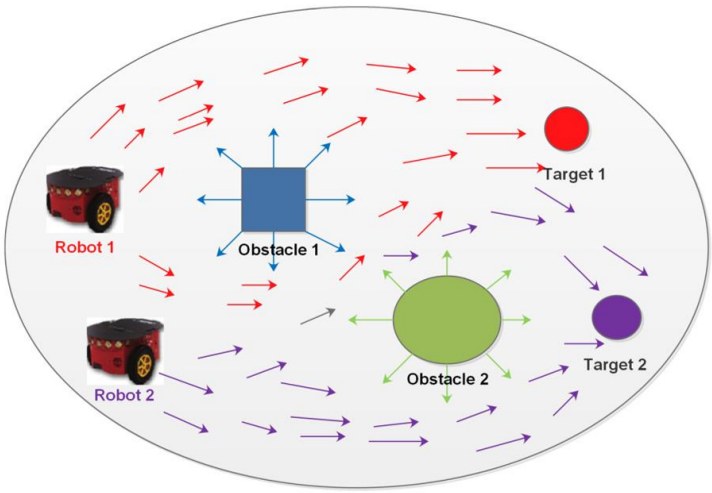


Figure 1.2: Attractive and repulsive potential fields.

The following figure shows the attractive force which is used to pull each robot toward its respective goal. Its expression is calculated depending on the actual position of each robot and the position of the targets while the repulsive force is used in order to ensure the safety of robots (each robot/goal is repulsive for other robots to ensure that each robot will be attracted only by its own goal). Its expression is written depending on the actual position of robots at each position of the workspace and the obstacles found in their neighborhood [6].

1.2.2 Graph Search

1.2.2.1 Dijkstra's Algorithm and Best-First-Search

Dijkstra's Algorithm works by visiting nodes in the graph starting with the object's starting point. It then repeatedly examines the closest not-yet-examined node, adding its nodes to the set of the ones to be examined. It expands outwards from the starting point until it reaches the goal [7]. From **Figure 1.3**, we can see that Dijkstra's Algorithm is guaranteed to find a shortest path from the starting point to the goal, as long as none of the edges have a negative cost.

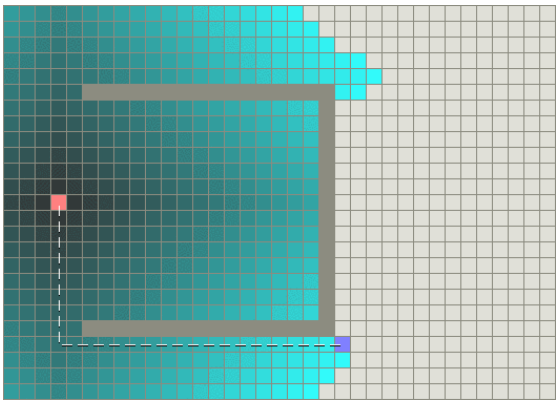


Figure 1.3[7]: Dijkstra's Algorithm

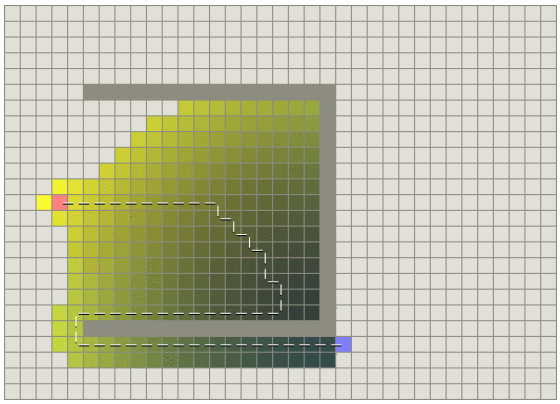


Figure 1.4[7]: BFS Algorithm

The Greedy Best-First-Search algorithm works in a similar way as Dijkstra's Algorithm, except that it has some estimate (called a heuristic) of how far from the goal any node is. From Figure 1.4 we can see that instead of selecting the closest node to the starting point, it selects the closest one to the goal. Compared to Dijkstra's Algorithm, Greedy Best-First-Search doesn't guarantee the shortest path. However, it runs much quicker than Dijkstra's Algorithm because it uses the heuristic function to guide its way towards the goal very quickly.

In the other hand, we can remark that Dijkstra's Algorithm guarantee the shortest path despite of working hard and wasting time exploring useless cells.

On the other hand, BFS Algorithm does less work but its final path isn't optimal in case of some kind of obstacles composition since it only considers the cost to get to the goal and ignores the cost of the path.

1.2.2.2 The A*(A-Star) Algorithm

Developed in 1968 to combine heuristic and formal approaches, the A* is like Dijkstra's Algorithm where it can be used to find a shortest path, and like Greedy Best-First-Search such that it can use a heuristic to guide itself. Therefore, it is as fast as Greedy Best-First-Search and finds a path as good as what Dijkstra's Algorithm does, which means, taking the shortest path by exploring the less possible cells, **Figure 1.5** shows an example of a concave obstacles combination:

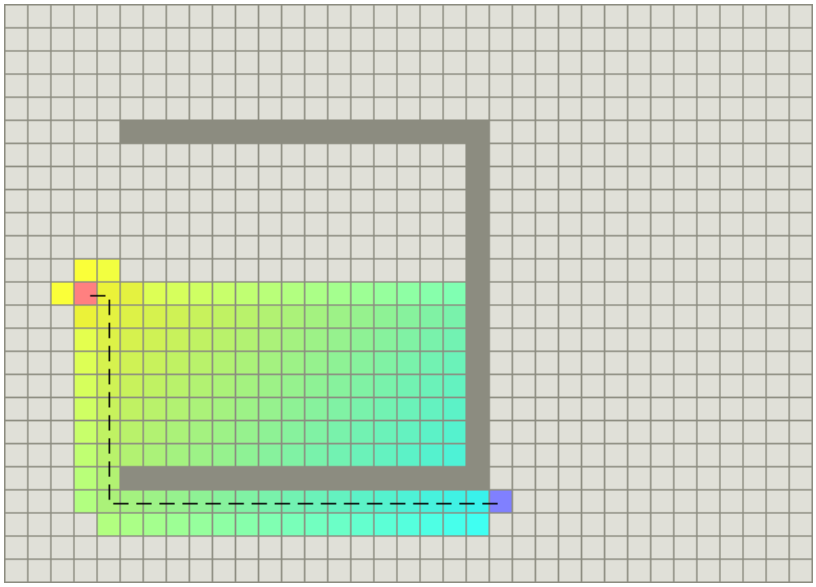


Figure 1.5[7]: A* Algorithm

A* uses the distance between the current location and the target and moves to the square that has the smallest distance. It evaluates squares (henceforth called a “node”) by combining g(n), the *exact cost* of the path from the starting point to any vertex **n** and h(n), the heuristic *estimated cost* from vertex **n** to the goal.

The total cost f (n) = g (n) + h (n) is calculated for each successor vertex and the vertex with the smallest cost f (n) is selected as a successor.

1.2.3 Sensor Based Method

Robot motion path planning revolves around two models that are based on different assumptions about the information available for planning. In the first model called "path planning with complete information", perfect information about the robot and the obstacles is assumed. Under the second model called "path planning with incomplete information", an element of uncertainty is present, and the missing data are provided by some source of local information such as a laser range finder or vision sensor. This model introduces a notion of sensor feedback and transforms the operation of motion planning into a continuous dynamic process. Under this approach, sensing becomes an active process, the robot decides at each step of its path what sensory information is required for generating its next step. The range sensor provides the robot with coordinates of those points of obstacle boundaries that lie within a limited radius of vision around the robot.

1.2.3.1 Bug Algorithm's family

1.2.3.1.1 Bug 1 Algorithm

This algorithm make the robots move in the direction of the goal until an obstacle is encountered. A canonical direction is followed (clockwise) until the location of the initial encounter is reached. The robot then follows the boundary to reach the point along the boundary that is closest to the goal. At this location, the robot moves directly toward the goal. If another obstacle is encountered, the same procedure is applied.

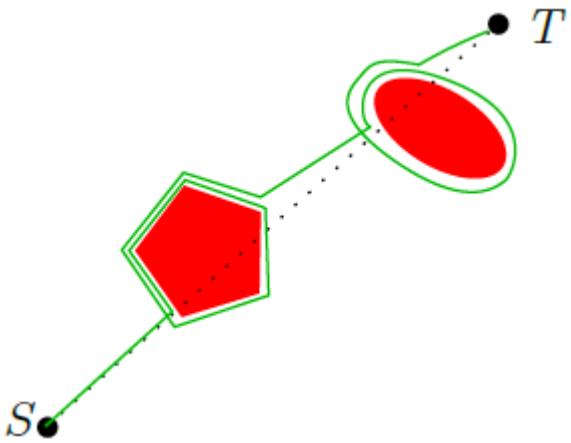


Figure 1.6[8]: Bug 1 Algorithm example

1.2.3.1.2 Bug 2 Algorithm

In this algorithm, the robot always attempts to move along the line of sight toward the goal. If an obstacle is encountered, a canonical direction is followed until the line of sight is encountered.

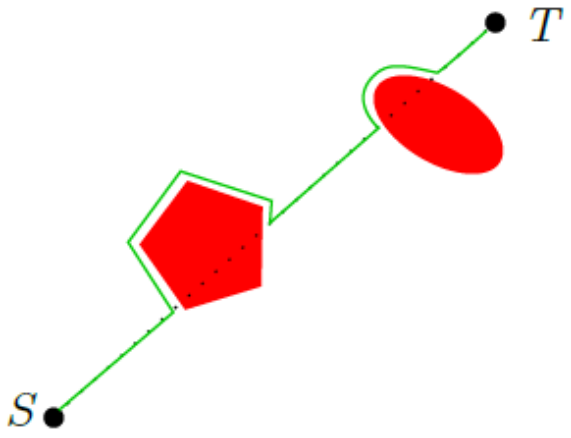


Figure 1.7[8]: Bug 2 Algorithm example

1.2.3.1.3 Bug 0 Algorithm

The main contribution of this algorithm is a new leaving condition which allows the robot to abandon obstacle boundaries as soon as global convergence is guaranteed, based on range data in the direction of the target. It is observed that moving along a straight path is faster than moving along the boundary of the obstacle, the leaving condition is designed to abandon the boundary as soon as convergence is guaranteed. The generated path is closer to the optimal path since the leaving condition is not based on the line connecting the start and goal point. Increasing the range sensor allows the robot to leave the obstacle boundaries earlier.[8]

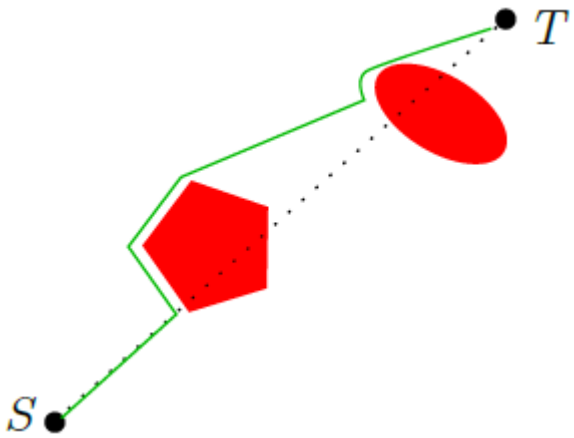


Figure 1.8[8]: Bug 0 Algorithm example

1.2.3.2 D* Algorithm

D*, Incremental A*, and D* Lite are extensions of A* that incrementally repair solution paths when changes occur in the underlying graph. These incremental algorithms have been used extensively in robotics for mobile robot navigation in unknown or dynamic environments.

D* functionality is equivalent to the A* re-planner, it initially plans using the Dijkstra's algorithm and allows intelligently caching intermediate data for speedy re-planning [9], its benefits are:

- Optimal
- Complete
- More efficient than A* re-planner in expansive and complex environments.
  - Local changes in the world do not impact on the path much.
  - Most costs to goal remain the same.

- It avoids high computational costs of backtracking.

1.3 Robot Formation

Planning for multiple robots is a broad field with application-specific methods, that's why Taxonomies are needed to:

- allow comparing different methods
- identify key issues
- identify trade-offs

In this project, the most useful taxonomies (proposed by Dudek et al. 1993) are:

1.3.1 Communication

The objective of communication is enabling robots to exchange state and environmental information with a minimum bandwidth requirement.

Of course, a higher communication exchange means a higher group performance, but it usually involves intermittent requests, status information and updates of sensory or model information, that's why we need to determine What, When, How and to whom communicate.

We should keep in mind that:

- Communication is not free, and can be unreliable
- In hostile environments, electronic countermeasures may be in effect.

The Major roles of communication are:

- Synchronization of action: ensuring coordination in task ordering
- Information exchange: sharing different information gained from different perspectives
- Negotiations: who does what?

1.3.2 Control Distribution

There are 3 types of control Distribution:

- **Centralized:** All control processing occurs in a single agent.

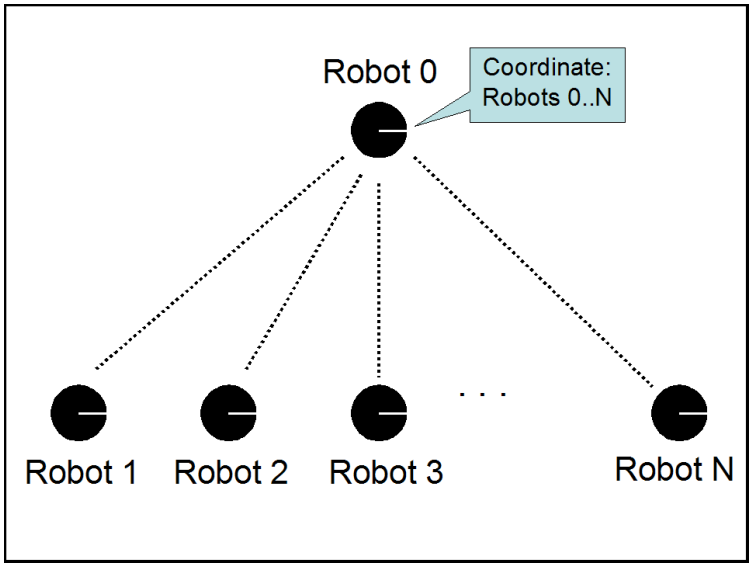


Figure 1.9: Centralized group Architecture

The advantage of a Centralized Multi-Robot Planning Approach is that Off-the-shelf path-planning algorithms can be directly applied, but, if the Dimensionality of configuration space increases, it leads to an increasing running time.

- **Decentralized:** Control processing is distributed among agents.

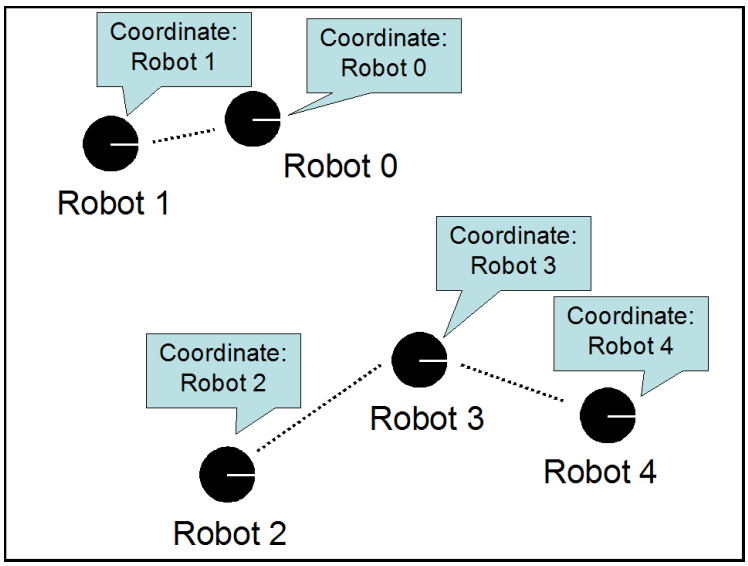


Figure 1.10: Decentralized group Architecture

Proposed by O'Donnell and Lozano-Perez 1989[10], this method plans paths for each robot independently of other robots, and coordinate them so that collision among robots are avoided.

The Advantage is that Dimensionality of configuration space doesn't increase running time since each robot is independent from others, but in the other hand, coordination is not always possible, which doesn't complete the decoupled planning.

Here are types of Decoupled approaches:

- ❖ Path coordination:
  - Plan independent paths for each robot
  - Plan velocities to avoid collisions (velocity tuning)
- ❖ Prioritized planning
  - Consider robots one at a time, in priority order
  - Plan for robot **i** by considering previous **i-1** robots as moving obstacles

➢ **Hierarchical:** Use groups of centralized systems.

1.3.3 Coordination and Cooperation

Coordination is when many robots share common resources (e.g. workspace, materials), they must coordinate their actions to resolve conflicts (e.g. collision).

Cooperation is when many systems strive to incorporate cooperation where robots are working together towards common goals (Cooperation requires coordination).

1.3.4 Size

Define size of the multi-robot system, for example, if it is a single robot, pair of robots, a limited number of robots or an infinite number of robots.

Scalability: this describes how amenable the system is to adding more robots.

It can result in a continuous degradation in performance as opposed to discrete.

Performance: the performance of a system can be characterized based on the number of robots (e.g. the number of tasks that can be accomplished in 1 hour).

Interference: Given limited resources, there is often a plateau or even decrease in performance once a certain threshold of robots is reached.

1.3.5 Composition:

The composition defines if the System is Homogenous or Heterogeneous, for example:

Homogeneous: All robots in the system have similar functionality and hardware.

Heterogeneous: Robots have varying functionality and hardware, affects maneuverability, tasks achievable, control possibilities or can lead to robots having roles.

1.4 Conclusion

In this chapter, we saw a general introduction about mobile robots, how they can be used more efficiently. The different types of systems, methods, algorithms and approaches used in order to solve different problems faced by one of the main applications of the mobile robot, which is motion planning.

We concluded that many methods are introduced in order to improve efficiency of mobile robots in crowded unknown environments, the objective is to reach the final destination by taking the shortest path without colliding with any static, dynamic obstacle or any other robot since it may be a multi-robot environment.

This is why a hybrid system of graph search using A* Algorithm and based sensor using an ultrasonic sensor for security has been chosen in order to improve our system in a multi-robot unknown environment with a prior knowledge of static obstacles.

CHAPTER II

Robots hardware design

In this chapter, the model of the robot which is implemented during our project is discussed. Including the sensors allowing the robot to collect DATA from its environment, to navigate safely and with high accuracy between dynamic and static obstacles, what make it useful in some delicate applications like rescue or fire stopping. As our project deals with a Hybrid Multi-robots platform; so, each robot needs to communicate with the other robots and with central unit (PC) to exchange DATA and collaborate their tasks. Hence, Wi-Fi modules are required and will be introduced at the end of this chapter.

2.1 Differential drive kinematics

Many mobile robots use a drive mechanism known as differential drive. It consists of 2 drive wheels mounted on a common axis, and each wheel can independently being driven either forward or backward.

Table 2.1. The spinning direction and speed control of each robot wheel

The direction of the robot	Direction of the Left wheel	Direction of the Right wheel	Speed of the left comparing to the right
Forward	Forward	Forward	Equal
Right	Forward	Forward/Backward	Greater
Left	Forward/Backward	Forward	Less
Backward	Backward	Backward	Equal

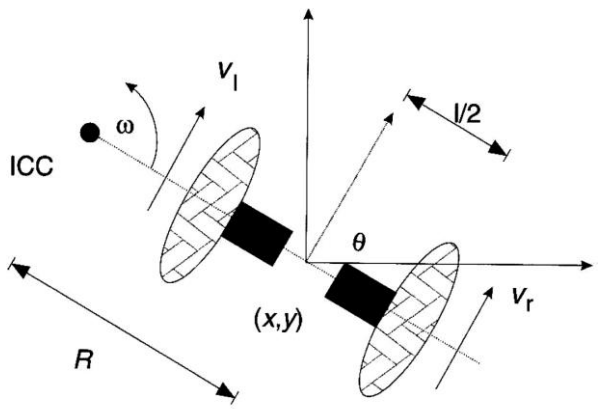


Figure 2.1: Differential Drive kinematics

While we can vary the velocity of each wheel, for the robot to perform rolling motion, the robot must rotate about a point that lies along their common left and right wheel axis. The point that the robot rotates about is known as the ICC[5] (see figure 2.1).

By varying the velocities of the two wheels, we can vary the trajectories that the robot takes.

Because the rate of rotation ω about the ICC must be the same for both wheels, we can write the following equations:

$$\omega\left(R + {}^{1}/_{2}\right) = Vr \quad (1)$$

$$w\left(R - {}^{1}/_{2}\right) = Vl \quad (2)$$

Where l is the distance between the centers of the two wheels, V_r , V_l are the right and left wheel velocities along the ground , and R is the signed distance from the ICC to the midpoint between the wheels. At any instance in time we can solve for R and ω:

$$R = \frac{l}{2}\frac{Vl + Vr}{Vr - Vl}\,; \omega = \frac{Vr - Vl}{l}; \quad (3)$$

There are three interesting cases with these kinds of drives:

- If $V_l = V_r$, then we have forward linear motion in a straight line. R becomes infinite, and there is effectively no rotation (ω is zero).
- If $V_l = -V_r$, then R = 0, and we have rotation about the midpoint of the wheel axis (we rotate in place).
- If $V_l = 0$, then we have rotation about the left wheel. In this case R = l/2 same is true if $V_r = 0$.

Note that a differential drive robot cannot move in the direction along the axis (this is a singularity). Differential drive vehicles are very sensitive to slight changes in velocity in each of the wheels. Small errors in the relative velocities between the wheels can affect

[5] *Instantaneous Center of Curvature*

the robot trajectory. They are also very sensitive to small variations in the ground plane, and may need extra wheels (castor wheels) for support.

2.1.1. Forward kinematics for differential drive robots

In figure 2.1, assume the robot is at some position (x, y), headed in a direction making an angle θ with the X axis. We assume the robot is centered at a point midway along the wheel axle. By manipulating the control parameters V_l, V_r, we can get the robot to move to different positions and orientations. (note: V_l , V_r) are wheel velocities along the ground). Knowing velocities V_l, V_r and using equation 3, we can find the ICC location:

$$\boldsymbol{ICC = [x - R\,sin(\theta); y + R\,cos(\theta)]} \tag{4}$$

And at time t + δt the robot's pose will be:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \tag{5}$$

This equation simply describes the motion of a robot rotating a distance R about its ICC with an angular Velocity of ω.

2.1.2. Inverse kinematics of a mobile robot

In general, we can describe the position of a robot capable of moving in a particular direction θt at a given velocity V (t) as:

$$\begin{cases} x(t) = \int_0^t V(t)\cos[\theta(t)]dt \\ y(t) = \int_0^t V(t)\sin[\theta(t)]dt \\ \theta(t) = \int_0^t \omega(t)dt \end{cases} \tag{6}$$

For the special case of a differential drive robot like the one we create, the equations become:

$$\begin{cases} x(t) = \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]\cos[\theta(t)]dt \\ y(t) = \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]\sin[\theta(t)]dt \\ \theta(t) = \frac{1}{l}\int_0^t [v_r(t) - v_l(t)]dt \end{cases} \tag{7}$$

A related question is: How can we control the robot to reach a given configuration (x, y, θ), this is known as the inverse kinematics problem.

Unfortunately, a differential drive robot imposes what are called non-holonomic[6] constraints on establishing its position. For example, the robot cannot move laterally along its axle. A similar nonholonomic constraint is a car that can only turn its front wheels. It cannot move directly sidewise, as parallel parking a car requires a more complicated set of steering maneuvers. So, we cannot simply specify an arbitrary robot pose (x, y, θ) and find the velocities that will get us there.

For the special cases of $v_l = v_r = v$ (robot moving in a straight line) the motion equations

become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + v\cos(\theta)\delta t \\ y + v\sin(\theta)\delta t \\ \theta \end{bmatrix} \quad \textbf{(8)}$$

If $v_r = -v_l = v$, then the robot rotates in place and the equations become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + 2v\delta t/l \end{bmatrix} \quad \textbf{(9)}$$

This motivates a strategy of moving the robot in a straight line, then rotating for a turn in place, and then moving straight again as a navigation strategy for differential drive robots.

2.1.3. Mapping angular wheel velocity to linear velocity

The left and right wheel velocities used above, V_l , V_r are linear velocities. We actually control the wheels by specifying an angular velocity V_{wheel} for a wheel specified in radians per second. Given V_{wheel}, we need to find out what the resulting linear velocity for that wheel's movement is. We define the following terms: r_{wheel}: wheel radius. D_{robot}: length of the differential drive wheel axle. V_{wheel}: magnitude of wheel velocity measured in radians/sec. If we want the robot base to rotate by ϕ degrees (the robot is turning in place), we need to find an equation for the amount of time t we need to run the wheel motor at velocity V_{wheel} to turn the robot an angle of ϕ degrees. The wheel turns a linear distance of $r_{wheel} . \theta$ along its arc where θ is simply $V_{wheel} . t$.

[6] In physics and mathematics is a system whose state depends on the path taken in order to achieve it.

The wheel will travel a distance equal to rθ along its arc (see figure 2.2). If we assume a wheel velocity of V_{wheel} = 10 radians/sec, then the wheel will travel 10*8 = 80 mm in 1 sec, which is also equivalent to 0.08mm in 1 msec.

To determine the time to turn the robot a specified angle in place, we note that the entire circumference C of the robot when it turns 360◦ is π D_{robot}.

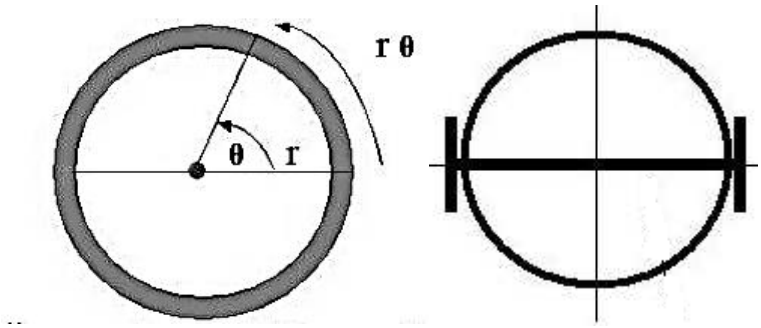
Given a time t, the wheel will turn:

$$Dist_{wheel} = r_{wheel} * V_{wheel} * t$$

$$\frac{Dist_{wheel}}{C} = \frac{\emptyset}{2\pi} \ (\emptyset \ measured \ in \ radians) \qquad (10)$$

$$\frac{V_{wheel} * r_{wheel} * t}{C} = \frac{\emptyset}{2\pi}$$

$$t = \frac{\emptyset C}{2\pi V_{wheel} * r_{wheel}}$$



a ***b***

Figure 2.2: a) Mapping angular velocity into linear velocity

b) Upper view of differential drive robot

2.1.4. Dimensions of robots we are using

We must take into consideration that we are using three similar robots with dimensions as mentioned shown in figure 2.4.

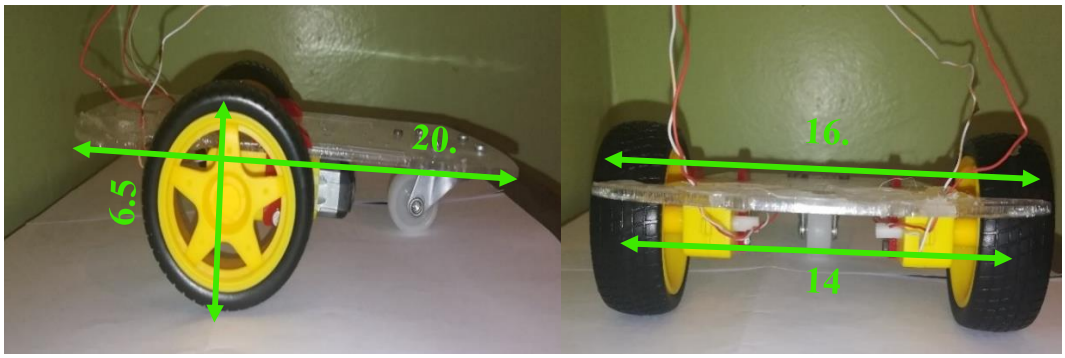


Figure 2.3: Measurement of the robot's body

$$D_{robot} = 14cm$$
$$r_{wheel} = {}^{6.5}/_{2}\ cm$$

2.2 Sensors

A sensor is a device which detects or measures a physical property and records, indicates, or otherwise responds to it [11].

To make a robot interact with its environment we need actuators and sensors, and in the following we are presenting some of the sensors that makes the robot see it surrounding and calculate the distances between the robot and the obstacles to design a map which will be the platform for planning a safe path from the starting point to the end point.

2.2.1 Ultra-Sonic sensor

From its name it is a sensor that measure distance by using ultrasonic waves. The sensor has an emitter that emit the wave and a receiver that receive the reflected wave from the obstacle. It measures the distance to the obstacle by calculating the time between the emission and reception.

a. Features of US sensor

- It detects even the transparent obstacles because the ultrasonic waves are reflected off a glass or liquid surface back to the receiver.

- Detection is not affected by accumulation of dust or dirt what make it resistant to dirt and mist.
- The detection is stable even for complex shaped obstacles.

b. Its functionality

It emits a 40000*Hz* ultrasonic signal which travels through the air and if there is an object or obstacle on its path it will bounce back to the module as explained in figure 2.4.

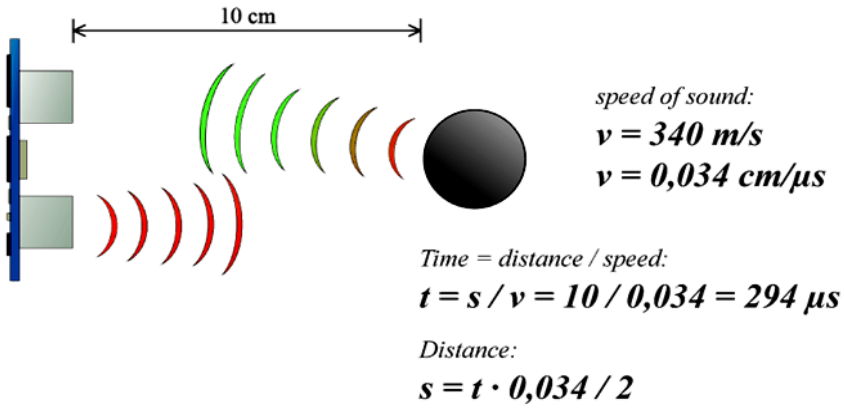


Figure 2.4: Features of Ultrasonic sensor HC-SR04

The distance can be calculated as follow:

$$D = T' * C$$

$$T' = {}^{1}/_{2} * T$$

where *D* is the distance, *T* is the time between the emission and reception, and *C* is the sonic speed. (*T'* is the time needed to travel half of the distance forth or back).

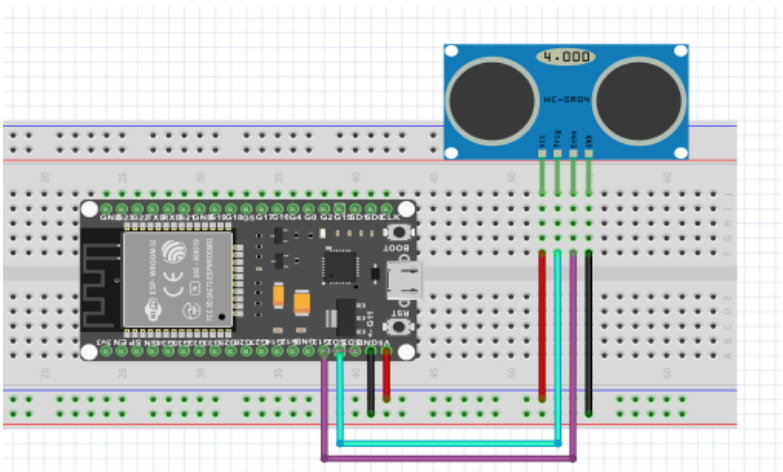


Fig. 2.5: Connecting ultrasonic sensor to the ESP32 card

The HC-SR04 Module has 4 pins, as illustrated in Fig 2.5, which are:

- VCC: +5VDC
- Trig: Trigger (INPUT)
- Echo: Echo (OUTPUT)
- GND: GND

The Ground and the VCC pins of the module needs to be connected to the Ground and the 5 volts pins on the controller Board respectively and the trig and echo pins to any Digital I/O pin on the controller Board [12].

In order to generate the ultrasonic signal we set the Trig on a High State for 10 µs. That will send out an 8 cycle sonic burst which will travel at the speed sound and it will be received in the Echo pin as explained in figure 2.6. The Echo pin will output the time in microseconds the sound wave traveled.

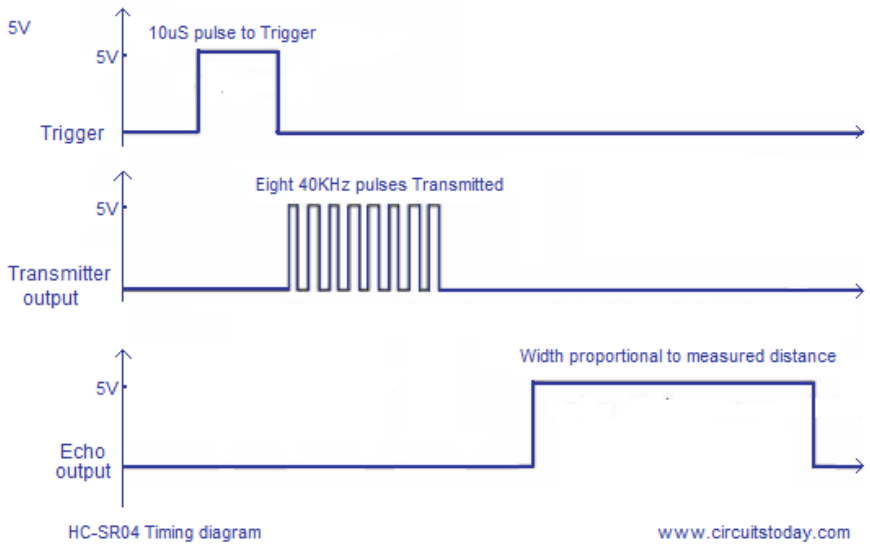


Figure 2.6: Generate the ultrasonic signal

2.2.2 Tachometer and encoder

To determine performance and efficiency of DC motors, it helps to have a tool that can measure or calculate the speed, angle, or count of a rotating object.

Tachometers measure rotational speed, which is particularly valuable for robot building. The speed of the motors determines the speed of the robot. Before building an entire robot around a pair of motors, you want to know how fast the motor shaft turns at the highest and lowest voltages supplied. And, if the motor speed is significantly reduced

under the load of the robot, that tells you the robot weighs too much for these motors (or gearheads) or that an unexpected source of friction (such as parts rubbing together) is degrading performance.

For very slow RPMs[7], you may need a device that counts the number of rotations over a longer period of time, because many digital tachometers aren't programmed to detect speeds slower than 10-60 RPM. An alternative is to use an encoder disk with many more marks, to fool the tachometer into thinking the wheel, gear, or shaft is spinning much faster. In either case, you simply divide the final number by the amount of time taken or the number of dark marks on the encoder disc.

Encoders are usually either encoder disks with visible marks and photosensors, or metal/magnets with magnetic field (Hall Effect) sensors.

As an encoder we are using a 20 holes disks printed using a 3D printer of the club (WAMEEDH) and FC-03 tachometer data sheet. [13]

2.2.2.1 FC-03 Tachometer

It is a widely used tachometer in robotics applications, with:

a) Features

- **Current:** Around 15mA
- **Operating voltage** DC 3.3V - 5V
- **Output signal:** Digital switching outputs (0 and 1)
- **Dimensions:** 3.2cm x 1.4cm
- Imported groove coupler sensor
- A fixed bolt hole for easy installation
- Used wide voltage LM393 comparator
- Groove width of 5mm

[7] Revolution per minute

b) Specifications

- DO output interface can be directly connected to a microcontroller IO port, if there is a block detection sensor, such as the speed of the motor encoder can detect.
- DO modules can be connected to the relay, limit switch, and other functions, it can also with the active buzzer module, compose alarm.

c) Pin-out

Table 2.2: FC-03 Pin Description

Pin	Function
Vcc	Connect to the positive 3.3V - 5V power supply
GND	Ground
DO	TTL switch signal output
AO	N/A

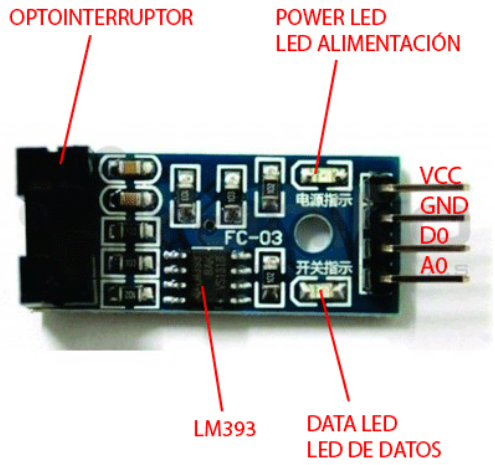


Figure 2.7 FC-03 Tachometer

2.3 Actuators

To make the robots move we need to use two dc motors each one associated with a wheel, and we need to add L293D motor driver module to amplify the current since the controller can't generate enough current to run the motors.

2.3.1 L293D

L293D is a dual H-bridge motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher-current signal. This higher current signal is used to drive the motors.

L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.

Enable pins 1 and 9 (corresponding to the two motors) must be high for motors to start operating. When an enable input is high, the associated driver gets enabled. As a result, the outputs become active and work in phase with their inputs. Similarly, when the enable input is low, that driver is disabled, and their outputs are off and in the high-impedance state. [14]

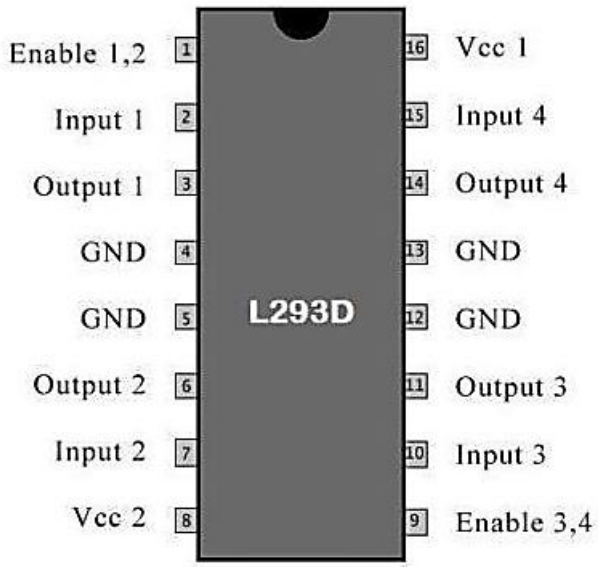


Figure 2.8: L293D PIN OUTS

Table 2.3: L293D Pin Description

PIN N°	Function	Name
1	Enable pin for motor 1, Active high	Enable 1, 2
2	Input 1 for Motor 1	Input 1
3	Output 1 for Motor 1	Output 1
4	Ground (0V)	Ground
5	Ground (0V)	Ground
6	Output 2 for Motor 1	Output 2
7	Input 2 for Motor1	Input 2
8	Supply voltage for Motors, 9-12V (up to 36V)	Vcc_2
9	Enable pin for Motor 2, active high	Enable 3,4
10	Input 1 for Motor 1	Input 3
11	Output 1 for Motor 1	Output 3
12	Ground (0V)	Ground
13	Ground (0V)	Ground
14	Output 2 for Motor 1	Output 4
15	Input 2 for Motor 1	Input 4
16	Supply voltage, 5V (up to 36V)	Vcc_1

2.3.2 DC motors

Figure 2.9: DC Motor

DC motor reducer, single axis, with DC 3V operating voltage and a RPM of 125R/minute, this gear box is applied for tracing car or robot. With plastic construction and colored in bright yellow, the DC gear motor measures approx. 2.5 inch long, 0.85 inch wide and 0.7 inch thick and a Shaft Size of 8mm x 5.4mm diameter and a Weight of 17gr.

Table 2.4: Specification of the DC motor that we are using

Motor specifications	
Motor type	Gearbox
Motor voltage	3-9 V
Gear Ratio	48:1
No-Load Current at 3V	0.12A
No-Load Speed at 3V	110 RPM
Stall Current at 3V	0.45 A
Stall Torque at 3V	0.26 kg-cm

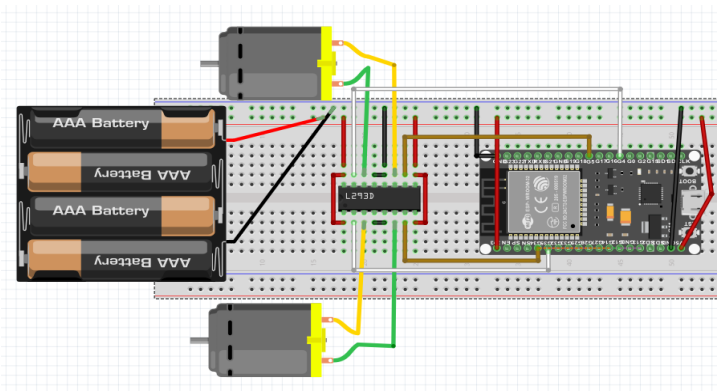


Figure 2.10: Interfacing L293D and 2 DC motors with ESP32

2.4 Wireless communication

A multi-robots system requires a communication support to transfer commands and exchange DATA between robots in both centralized and decentralized control. The best way to do that is by introducing a wireless communication module via Bluetooth or Wi-Fi.

2.4.1 Wi-Fi

Wi-Fi or IEEE 802.11x, shown in figure 2.7, is a technology for radio wave wireless local area networking that provide wireless high-speed Internet and network connections based on the IEEE 802.11 standards [15]. Wi-Fi module establishes connection between sender and receiver using radio frequencies (RF) within the electromagnetic spectrum (2.4GHz and 5GHz). When an RF current is supplied to an antenna it creates an electromagnetic field able to propagate through the space.

2.4.2 ESP 32 integrated Wi-Fi module

It is a hybrid Wi-Fi & Bluetooth Chip ESP32 can perform as a complete standalone system or as a slave device to a host MCU, reducing communication stack overhead on the main application processor. ESP32 can interface with other systems to provide Wi-Fi and Bluetooth functionality through its SPI / SDIO or I2C / UART interfaces.

2.5 The ESP32-WROOVER controller

ESP32-WROOM-32 is a powerful, generic Wi-Fi + BT + BLE MCU module that targets a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming and MP3 decoding.

At the core of this module is the ESP32-D0WDQ6 chip. The chip embedded is designed to be scalable and adaptive. There are two CPU cores that can be individually controlled, and the CPU clock frequency is adjustable from 80 MHz to 240 MHz The user may also power off the CPU and make use of the low-power co-processor to constantly monitor the peripherals for changes or crossing of thresholds. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI, UART, I²S and I²C. [16]

Table 2.5: ESP32-WROOM-32 Specifications

Categories	Items	specifications
Certification	RF certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Wi-Fi certification	Wi-Fi Alliance
	Bluetooth certification	BQB
	Green certification	RoHS/REACH
Test	Reliability	HTOL/HTSL/Uhast/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 µs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with -97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I^2C, LED PWM, Motor PWM, I^2S, IP, pulse counter, GPIO, capacitive touch sensor, ADC, DAC
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal

Integrated SPI flash	4 MB
Operating voltage/Power supply	2.7 V ~ 3.6 V
Operating current	Average: 80 Ma
Minimum current delivered by power supply	500mA
Recommended operating temperature range	-40 °C ~ +85 °C
Package size	(18.00±0.10) mm × (25.50±0.10) mm × (3.10±0.10) mm

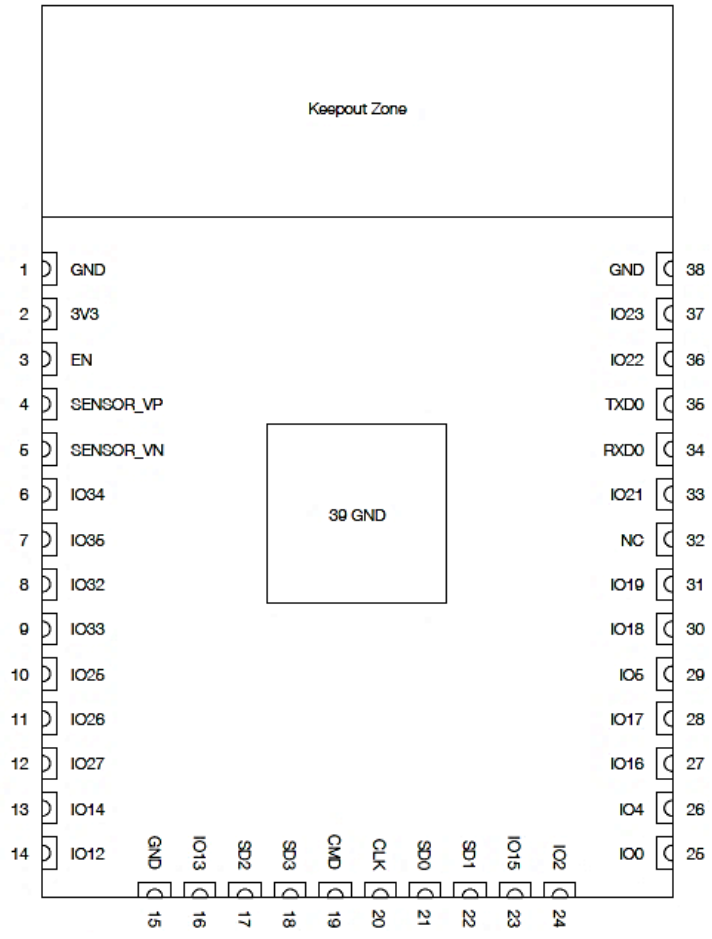


Figure 2.11: ESP32-WROOM-32 Pin Layout (Top View)

2.6 Building the robots circuit

After introducing each part alone and how to interface it with ESP32, now we introduce the final circuit of the robots taking in consideration that all the robots share the same circuit shown in the figure below:

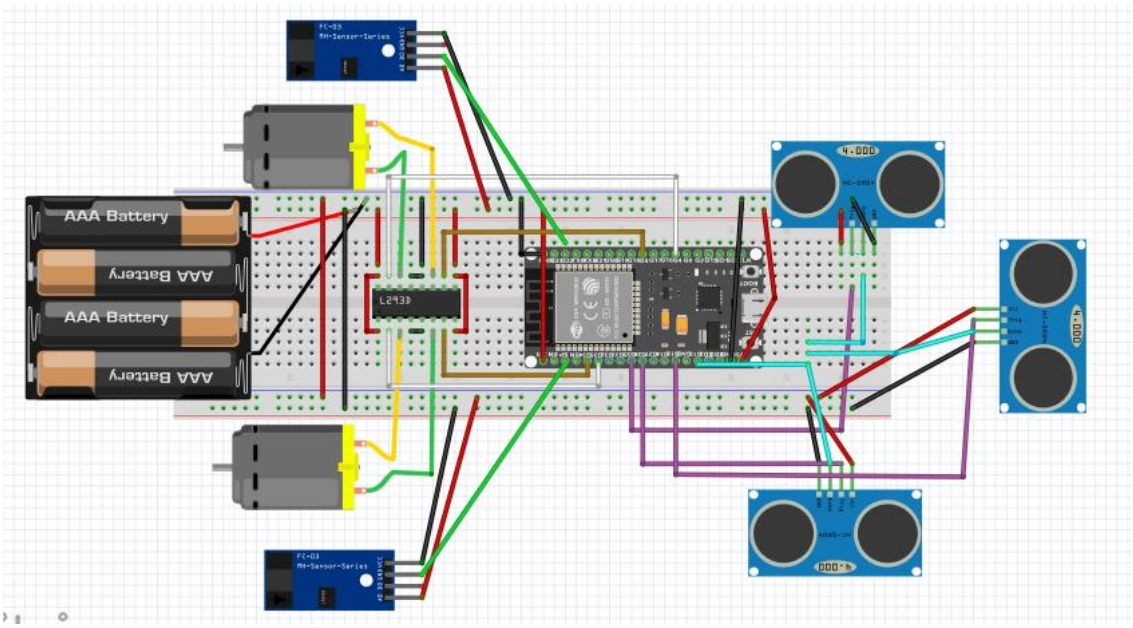


Figure 2.12: Final circuit

2.7 Conclusion

In this chapter we discussed in details the model of differential drive robot the one we are using for simulation and also the different sensors for obstacles detection, also the different communication technologies. The ultrasonic sensor is used because it is cheap and reliable for small indoor application where no interference with other devices, and the Wi-Fi module because it offers a faster full duplex DATA transfer.

CHAPTER III

Robots software design

In this chapter, the software used to run our system, the algorithm implemented to generate the optimal path for each robot and the collision avoidance system based on priorities will be introduced.

The connection between the two poles of our system, which are the Central unit (a computer running MATLAB) and robots (controlled by an ESP32) will be also established.

The last part will describe the robots behavior of robots after receiving the optimal paths using ultrasonic sensors to avoid any intruder obstacle.

3.1 Environment construction using MATLAB

There exist many approaches that allow us to construct a Map, plot robots/obstacles & run the chosen Algorithm in order to calculate and generate the best path for each robot. However, in order to ensure a quick calculation and stable communication between the software & hardware parts, we have used Matlab software that takes into account mathematical calculation and modeling part to allow optimal results; and the communication part to send/receive data from/to each robot.

Figure 3.1 gives an illustration example where a 10x10 Map with 3 start/end points for our multi-robot system and some random obstacles are generated.

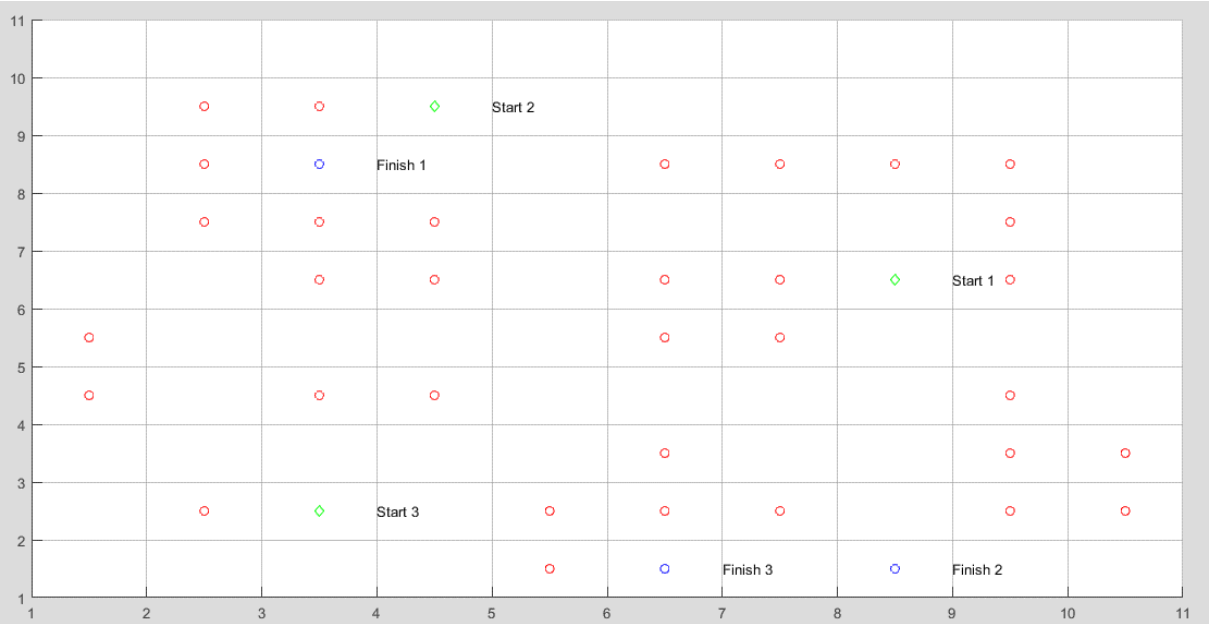


Figure 3.1: Multi robot formation in 2D Map plot

After entering the number of desired robots, the user can place any element (Robots, Goals, Obstacles) relative to the real world with a simple mouse click in the environment map. Since a warehouse is simulated, 2D environment case is generated.

At this moment, a Closed & Open Matrices are generated to store Occupied/free vertices as mentioned in chapter I.

3.2 Static/dynamic obstacles and Collision avoidance

There exist many approaches that allow a multi-robot system to navigate through a map without any collision. According to their obstacle detection and avoidance system algorithms and efficiencies, they can be classified from the simplest one (such as sensor based system) to the most complicated one (like image processing).

Since we are supposing that all obstacles are declared in the Map, an Ultrasonic sensor is added to each robot in order to deal with intruders and get high efficiency navigation.

This solution is generated by two parts:

i. The computer part, which represents the central unit and the Master part of our system, it executes the given algorithm and generates an optimal path for each robot by respecting priorities between robots (The priority system is given by order from the first robot with the highest one till the last robot with the lowest priority).

ii. The robot part, which represents the central unit's slave, will executes in parallel the given path and checks (using ultrasonic sensors) for intruder obstacles.

3.2.1 Computer level Obstacle Detection

The space configuration production

Since the map is in 2D integer array, the MATLAB code generates and assigns an integer number for each cell, so the example above shown in Figure 3.1 can be seen as a matrix as follow:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & -1 & -1 & 4 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & -1 & 0 & 2 & 2 & -1 & -1 & -1 & -1 & 2 \\ 2 & -1 & -1 & -1 & 2 & 2 & 2 & 2 & -1 & 2 \\ 2 & 2 & -1 & -1 & 2 & -1 & -1 & 1 & -1 & 2 \\ -1 & 2 & 2 & 2 & 2 & -1 & -1 & 2 & 2 & 2 \\ -1 & 2 & -1 & -1 & 2 & 2 & 2 & 2 & -1 & 2 \\ 2 & 2 & 2 & 2 & 2 & -1 & 2 & 2 & -1 & -1 \\ 2 & -1 & 6 & 2 & -1 & -1 & -1 & 2 & -1 & -1 \\ 2 & 2 & 2 & 2 & -1 & 5 & 2 & 3 & 2 & 2 \end{bmatrix}$$

Figure 3.2: Matrix representation of 10x10 Map

Such that integers inside the matrix can be interpreted as follow:

- Number -1 : Obstacle
- Number 0 : Goal 1
- Number 1 : Robot 1
- Number 2 : Empty
- Number 3 : Goal 2
- Number 4 : Robot 2
- Number 5 : Goal 3
- Number 6 : Robot 3

Once data is saved into a 10x10 matrix, Obstacles (Number -1 in our case) will be stored into a **2xn** CLOSED list before the algorithm can start to search for the shortest path from "Start" point (Number 1, 4 and 6 for Robot 1, 2 and 3 respectively) till the "Finish" point (Number 0, 3 and 5 for robot 1, 2 and 3 respectively).

The **2xn** CLOSED list represents all cells that have been explored including obstacles, while the OPEN list represents all successive cells that are yet to be explored.

3.2.2 Path planning algorithm

3.2.2.1 Case of single robot

In the case of one robot the A* algorithm is used. So, the "Start" cell is added to OPEN list as initial value, the function used to find the closest cell to the "Finish" point is $\{f(n) = g(n) + h(n)\}$, while **g (n)** is the exact cost of the path from the starting point to any cell **n**, and **h (n)** the heuristic estimated cost from the cell **n** to the goal as explained in Chapter I.

Once the closest cell with the smallest **f (n)** is found, the actual cell is removed from OPEN list and added to CLOSED list, from this point, the new cell is added to OPEN list and so on, until it reaches the Finish cell.

The flowchart given in figure 3.3 summarizes the algorithm A* used for space configuration in case of single robot as explained in this section.

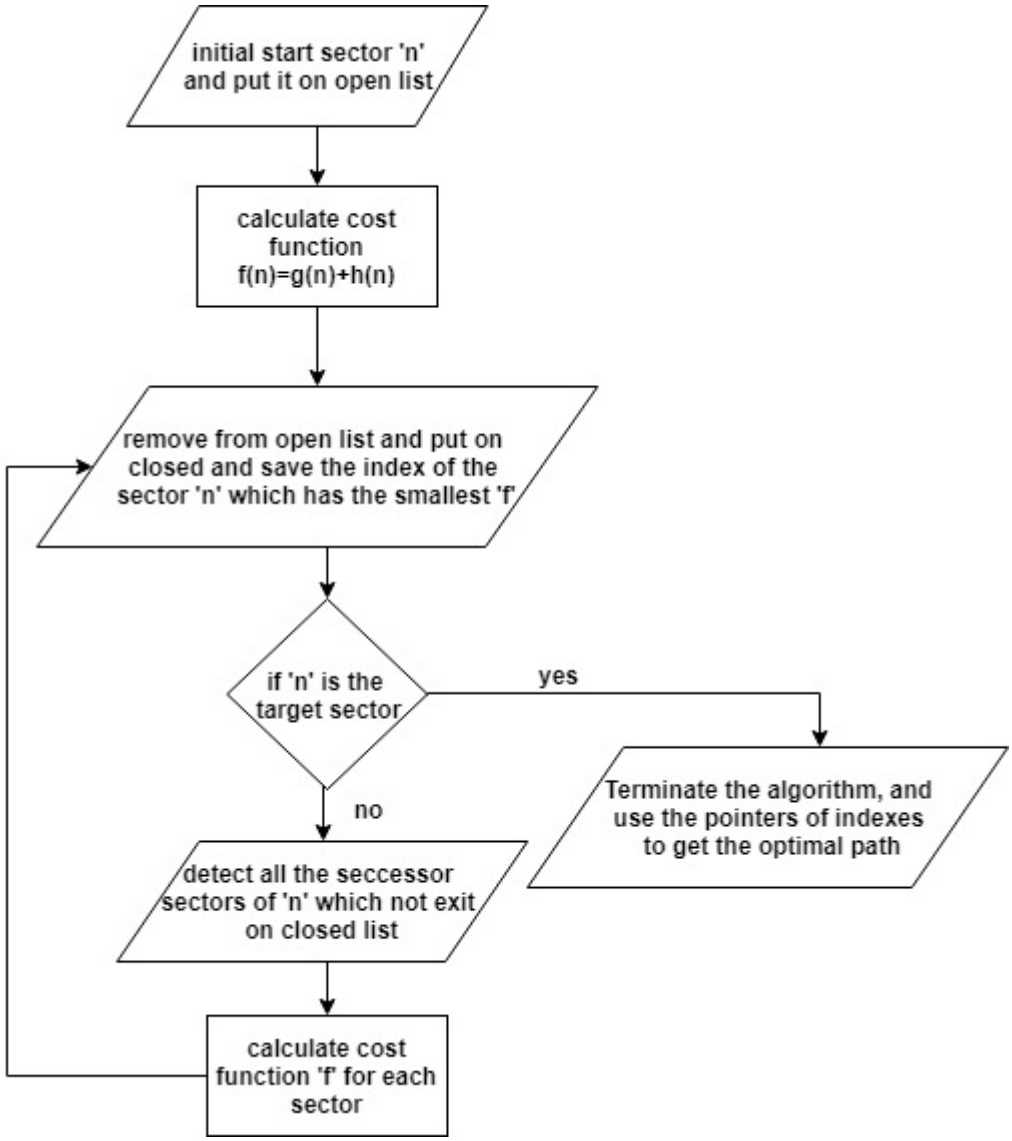


Figure 3.3: Flowchart of A* Algorithm

3.2.2.2 Case of Multi-robots

Here, the same thing is done as before, except that if two or more than two robots meet in a cell at the same time, a system of priorities takes action, the robots with a lower priority wait in the cell located before the intersection cell for a certain delay in order to let robots with higher priority keep going.

The result of our previous example using this method is shown in figure 3.4.

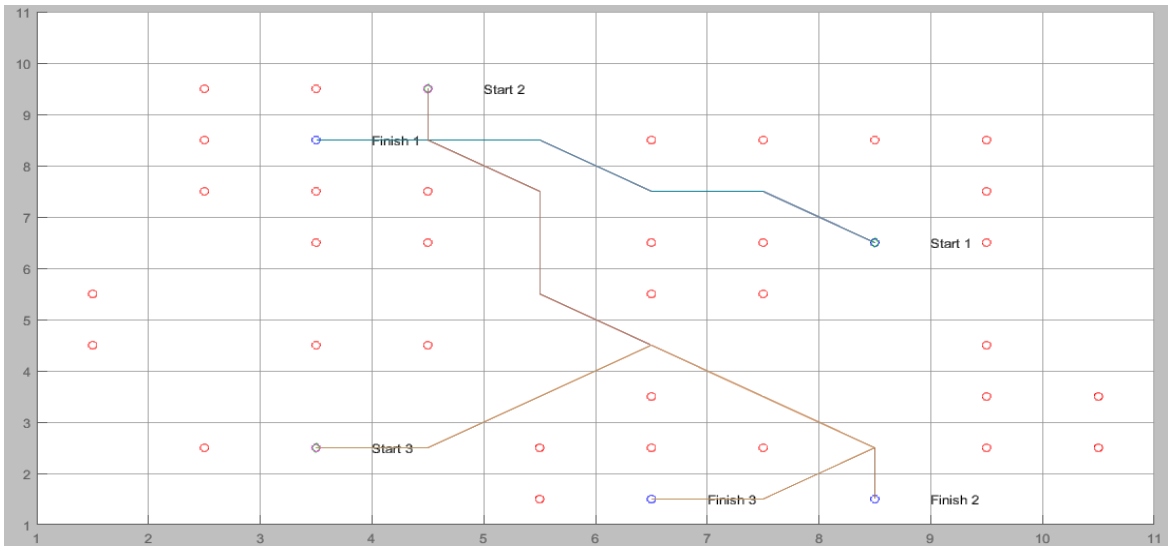


Figure 3.4: Generated path for Multi-robot formation

3.3 Communication Part

To make a Multi-robot hybrid system navigate without collision in an optimal time we need to make the robots communicate between them and with a computer to share the environment data and their positions to avoid conflict in following the paths generated by MATLAB.

First, the central unit has to be connected to all ESP32s via a Wi-Fi router, as shown in figure 3.5, which will assign IP addresses automatically.

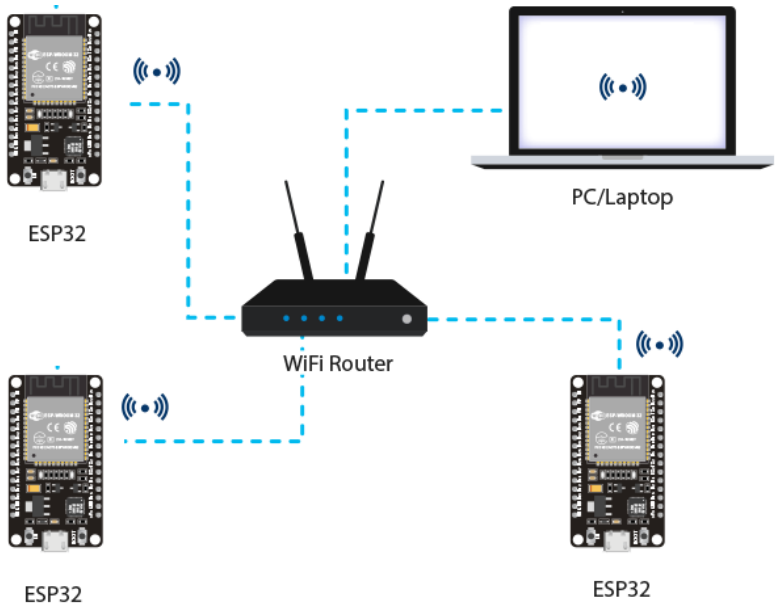


Figure 3.5: Multi-Robot Network model

Next, the connection between MATLAB and ESP32 using ARDUINO IDE is established using TCP/IP Protocol.[17]

Once the connection is established, each ESP32 will stay in listening mode waiting for TCP packet to be received from MATLAB containing matrix of the generated path associated with a start/stop signal.

Hence, figure 3.6 shows an ESP32 module in listening mode, before receiving data:

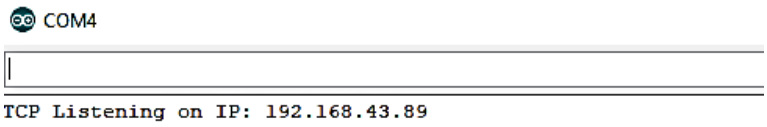


Figure 3.6: ESP-32 in listening mode

whereas, figure 3.7 show the matlab part when sending data and figure 3.8 shows an ESP32 module returning to a listening mode after receiving data.

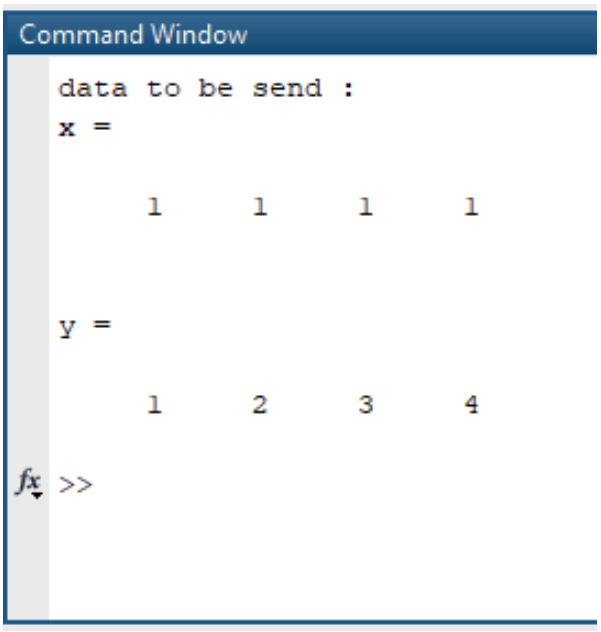


Figure 3.7: MATLAB command window after sending DATA.

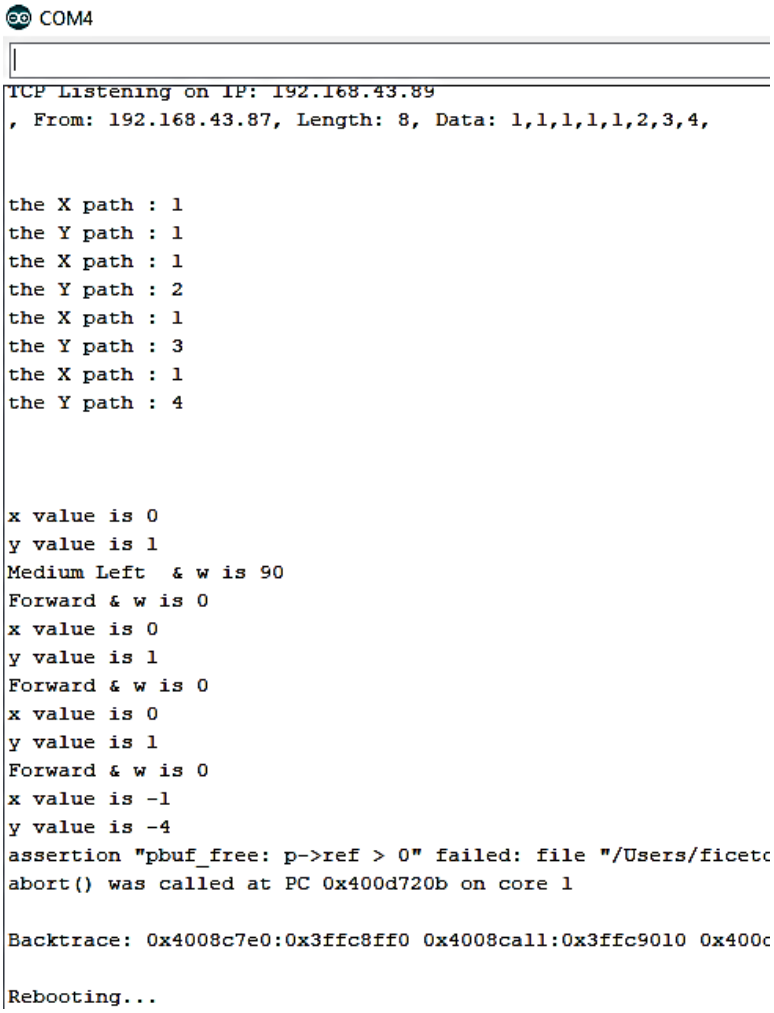


Figure 3.8: Returning to listening mode after reception/execution

Each packet is received and executed by ESP32 to start navigation while it is in listening mode. At that time, MATLAB enter in listening mode waiting for feedback.

3.4 Control part of robot

After explaining the environment construction strategy, path generation and how to send it to robots; now, the execution part will be discussed in this section.

The path is generated in the form of two vectors, the first one for the X coordinates and the second one for the Y coordinates, those two must be concatenated with Start/Stop byte to form the TCP packet to be sent from MATLAB to robots as shown in the table below.

Start/Stop	X values	Y values

The ESP32 of each robot stays in listening mode waiting for packet to be received. Once a packet arrives, the robot start processing it by checking the first byte if it matches the Start or Stop code, and it returns to listening mode. If the byte contain Stop code, the robot stops and sends back its position to MATLAB, otherwise it converts the data vector into a 2*N matrix and initialize a counter equal to the length of the path, to save the movements of the robot for future use.

All the robots can move in eight directions [-135° -90° -45° 0° 45° 90° 135° 180°], the starting one is always considered as 0° and at each iteration the robots calculate the angle to move with by subtracting the actual position/angle from the next position/angle in order to adjust their orientation and move to the next cell, until they reach end points, to make calculations easier and remove time factor, we made robots travel from each cell to the next one with same amount of time by making them move sqrt(2) faster when they move diagonally (Pythagoras theorem), with this operation, it takes the same amount of time traveling diagonally or Horizontally/Vertically.

At the same time, the robots keep looking for dynamic obstacles using ultrasonic sensors in front of them. When an obstacle is detected the robot with the lowest priority frees the path for the robot with higher priority, by checking the two other ultrasonic sensors on the right side and the left one.

The ultrasonic sensor is useful when the paths of two robots or more overlap, if the robots are going in the same direction the robot of lowest priority need to wait for the others to pass, but if they are going in opposite direction the robot with lower priority frees the road for the other robots by checking if there is enough space on the right side. The robot turns right and moves a sufficient distance and waits enough in order to lets the others pass, but if the right side of the robot is not free it will checks for the left one and do the same thing otherwise the robot goes back by reversing its path cell by cell and each time checks for the free space on both sides.

The flowcharts below (figure 3.9, figure 3.10 and figure 3.12) give a clear understanding of the program.

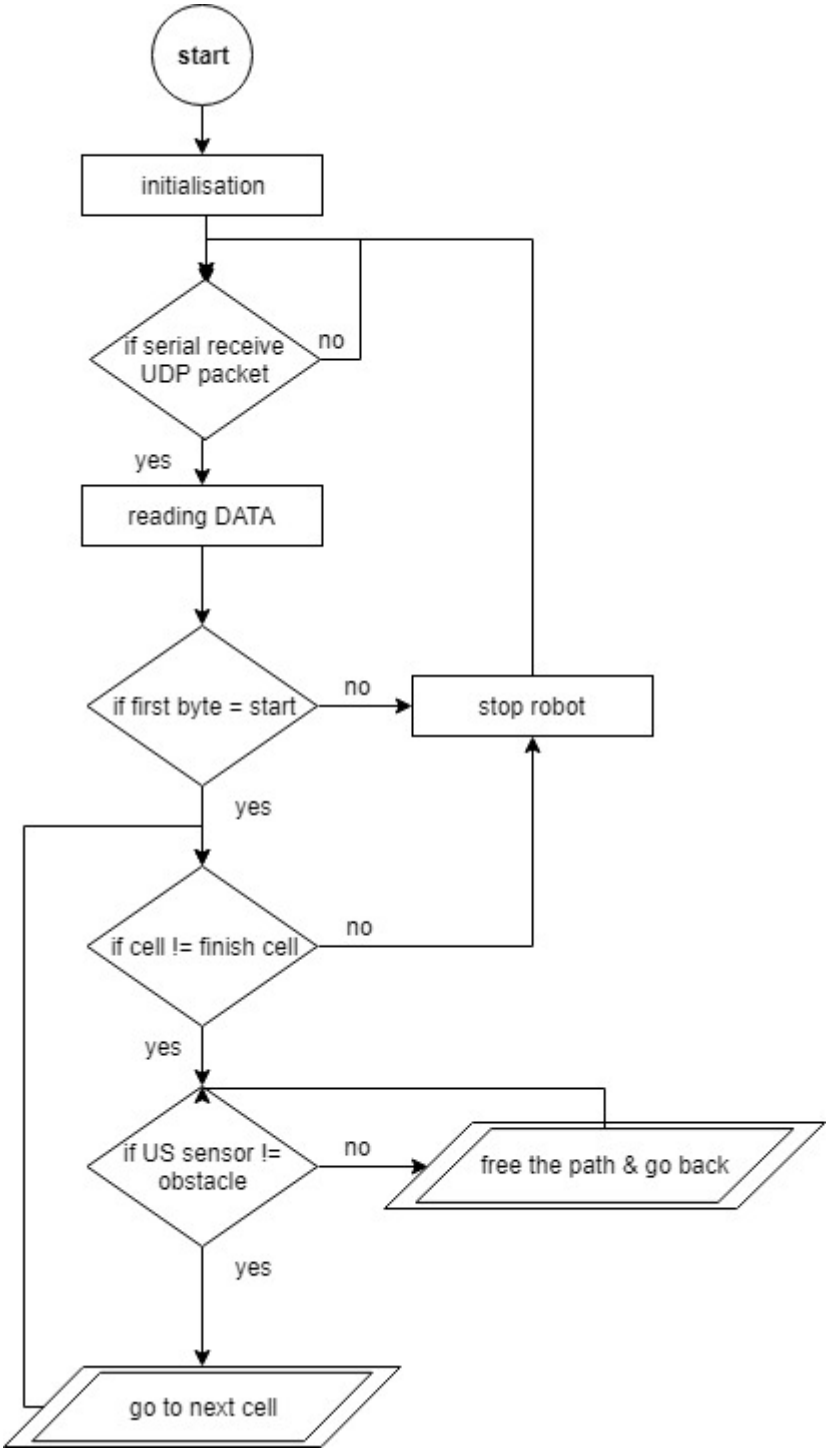


Figure 3.9: Flowchart of the entire program

The flowchart in figure 3.10 explain in more details how the robots with lower priority will free the path for the others and after that they return to their previous position. And the one in figure 3.12 explain how robots go to next cell by calculating the direction of the next move and moving forward for specific distance calculated using Pythagoras theorem.

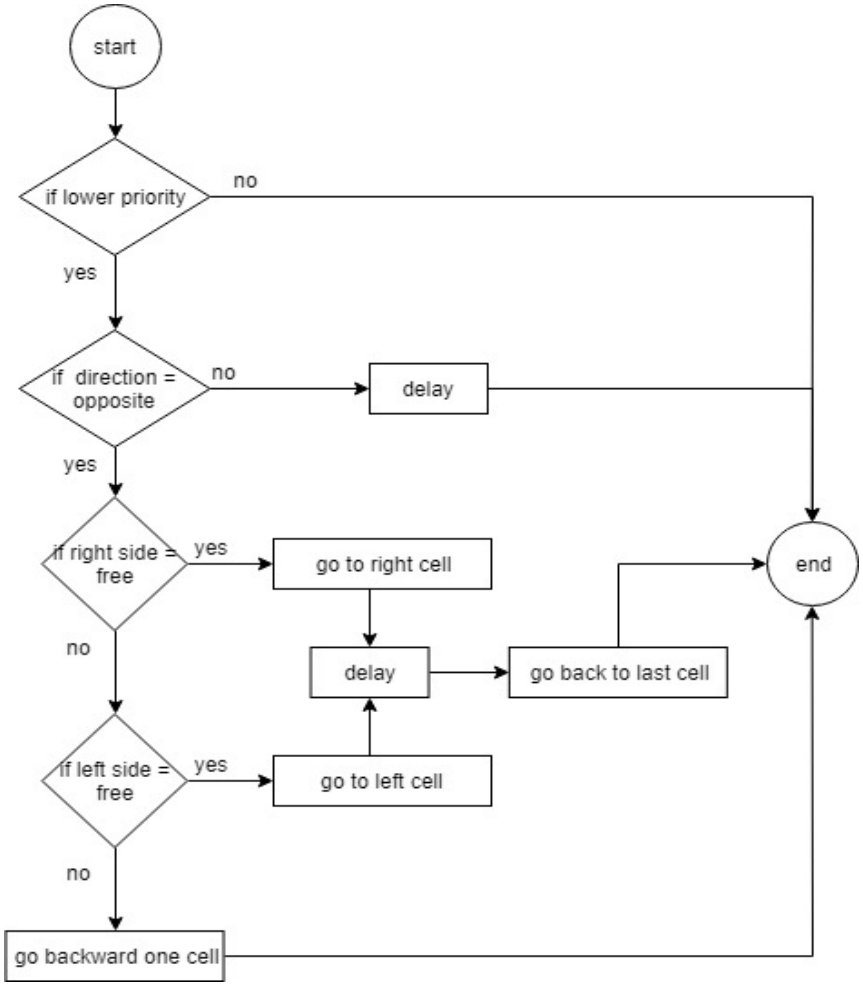


Figure 3.10: Flowchart of Path processing

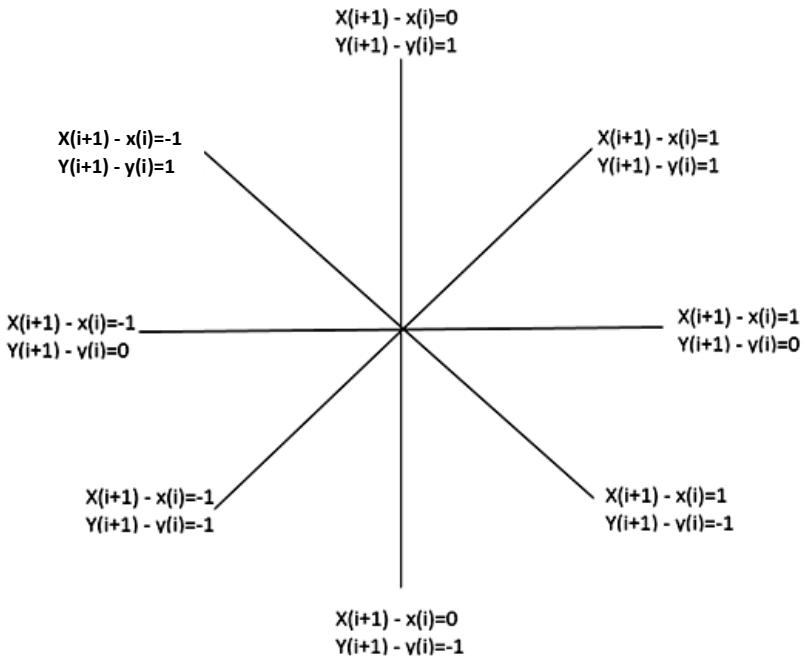


Figure 3.11: Calculation of future orientation

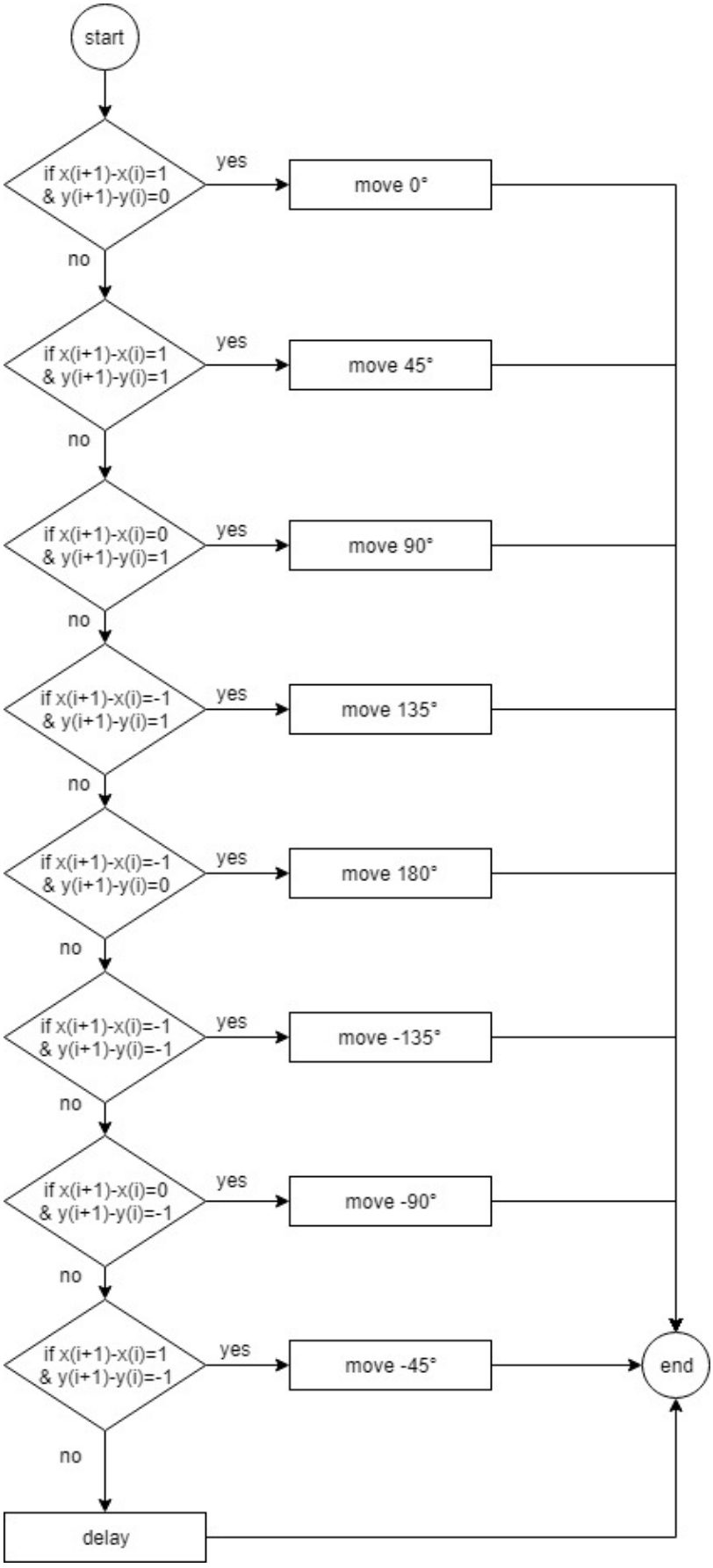


Figure 3.12: Flowchart of Movement/direction processing

3.5 Conclusion

In this chapter, we introduced the simulation part of our project, we saw how to proceed with a Multi-robot system by presenting flowchart that describe the A* Algorithm, the path generation and robot behavior after receiving the path, in order to eliminate several cases of possible collisions and conflicts.

Those optimizations will be discussed deeper in chapter IV, in order to compare the simulation results with the experimental ones.

CHAPTER IV

Results and discussions

In this chapter, we are presenting the results of the simulation and giving some examples associated with tests results to model some real life scenarios; so that, the efficiency of our system will be validated and in order to escape some critical situation. After that, we are giving some evaluation about the speed of our algorithm. At the end, we are suggesting some features to be added to make a more sophisticated system.

4.1. Software Implementation

The simulation part is working perfectly without any problem during the execution, it would be good to get the same results during the implementation. However in real world, it is not the case where a lot of constraint appears disallowing an optimal navigation from the starting points to the targets, and especially in a crowded environments. Thus, to get a very smooth navigation with high accuracy, we need to minimize the error factor.

In order to do so, both software and hardware (Computer and robots) must be well calibrated and optimized, the connection between the two parts must be stable and reliable using a determined router (Wireless Network is known to be more vulnerable than wired network).

The MATLAB Software guarantees a good flexibility for multiple Operating Systems, in addition to its reliability. All these factors will be presented in this chapter by showing how they have been applied.

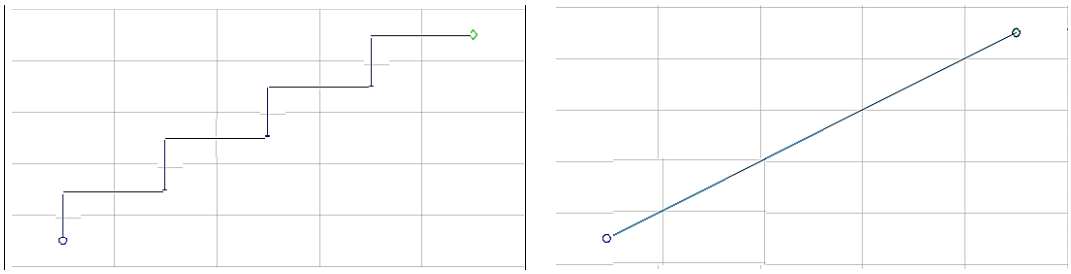
4.1.1. Multi-robot formation strategy

Some problems have been encountered during the implementation part, the most important ones were the collision point and obstacle avoidance in case of a non-declared obstacle facing one or several robots. Fortunately, some of them can be treated by adding some algorithms, such as collision point avoidance algorithm in order to reduce the work between robots.

4.1.1.1 Collision point treatment

Since we are modeling a warehouse, we suppose that our robots are moving in a 2D environment composed of cells. An algorithm must be introduced in order to expand the

original one and take into consideration dynamic obstacles (other robots in our case or humans). Moreover, in order to increase efficiency and flexibility of our work, we allowed the diagonal displacement that permits the system to win time and decrease number of rotations in case of a long diagonal displacement; hence, prevent robots from moving horizontally/vertically each time to reach the final cell giving a smooth navigation in 8 directions as shown in **figure 4.1**.



-a- **-b-**

Figure 4.1: a) Path planning using four directions

b) Path Planning using eight directions

In figure 4.1, the path cost using method -b- is 29.29% less than the path cost of method -a-, which includes an equivalent of saving one third of the time/power consumption.

However, allowing each robot to run in diagonal direction drives each of them to pass through a diagonal row of cells with a thickness of one cell as shown if **figure 4.2,** which will be considered as a path with an incremented risk of collisions.

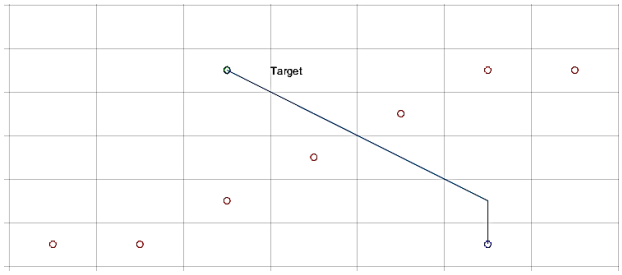


Figure 4.2: Diagonal obstacles with thickness of one cell

In order to fix this issue, we have to increment the thickness of the diagonal row by one as shown in **figure 4.3**.

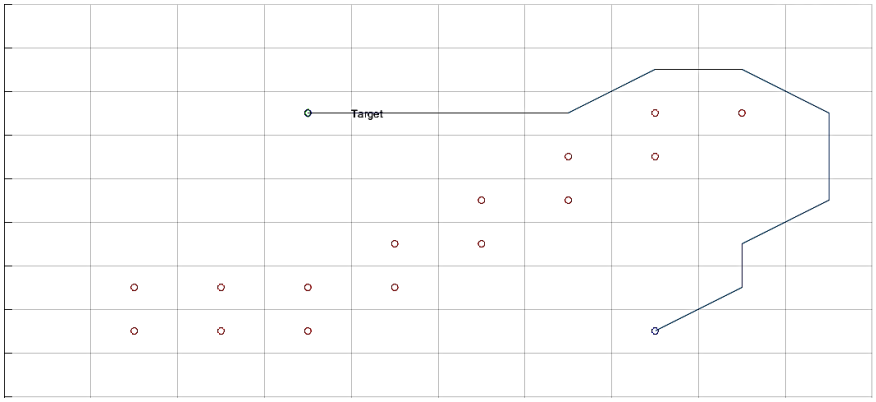


Figure 4.3: Diagonal obstacles with thickness of two cells

Now, one more problem at collision point treatment may happen for diagonal displacement, which will require 2 different algorithms in order to ensure a safe Optimal path.

1) First, let's suppose a horizontal/vertical collision:

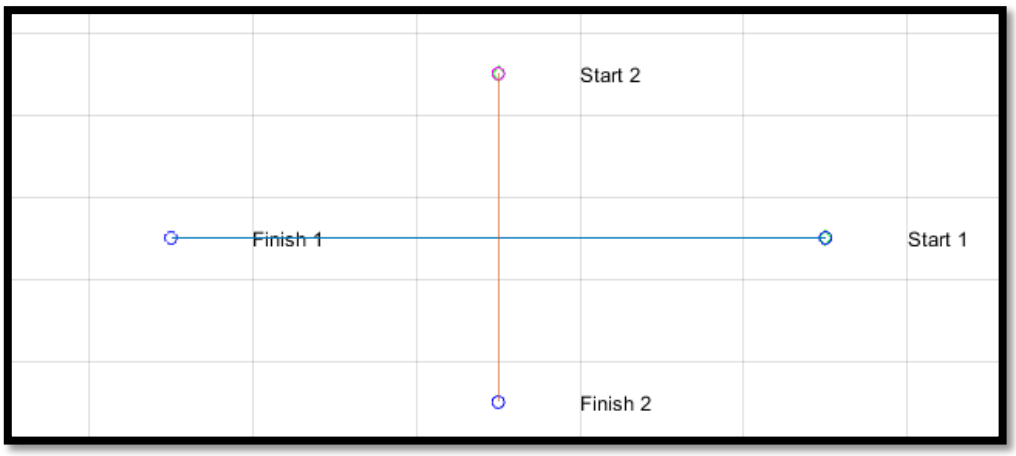


Figure 4.4: Horizontal/Vertical Multi-robot collision

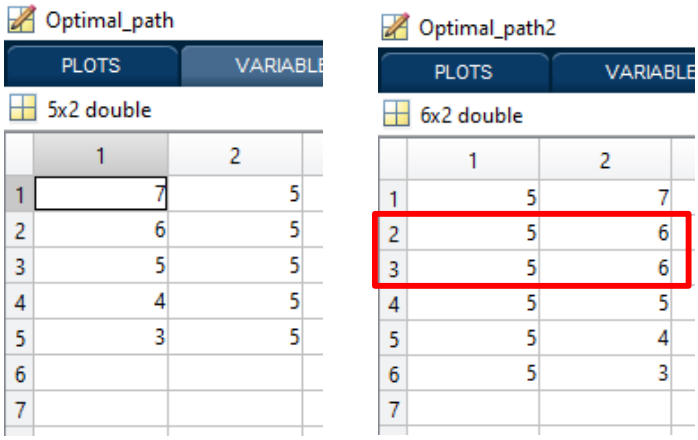
At first glance, the horizontal/vertical collision always occurs in a given cell which may be the same as the diagonal collision in some cases. From this, the following algorithm which takes as reference the collision cell will be executed in order to generate the Optimal Path by taking into account the priority of each robot.

```
a.x = OptimalPath.a[i][0]                //Robot 1's x position
a.y = OptimalPath.a[i][1]                //Robot 1's y position
b.x = OptimalPath.b[i][0]                //Robot 2's x position
b.y = OptimalPath.b[i][1]                //Robot 2's y position
i = 1
WHILE SIMULATION != END
        IF Priority.b < Priority.a
                IF ( a.x = b.x && a.y = b.y)
                        OptimalPath.b[i][0] = OptimalPath.b[i-1][0]
                        OptimalPath.b[i][1] = OptimalPath.b[i-1][1]
                        i++
                        OptimalPath.b[i][0] = b.x
                        OptimalPath.b[i][1] = b.y
                        i++
                 ELSE
                        OptimalPath.b[i][0] = b.x
                        OptimalPath.b[i][1] = b.y
                        i++

                 END IF
         END IF
END WHILE
```

Figure 4.5: Pseudo-code of the Horizontal/vertical collision avoidance Algorithm

The result of the two optimal paths is illustrated in **figure 4.6**.



	1	2
1	7	5
2	6	5
3	5	5
4	4	5
5	3	5
6		
7		

	1	2
1	5	7
2	5	6
3	5	6
4	5	5
5	5	4
6	5	3
7		

Figure 4.6: Optimal paths depending on Horizontal/Vertical collision cell

As shown in **figure 4.6**, when i = 3, the collision cell at [5,5] (i represents the length of each path generated, i_max = 5 for Optimal_path and i_max = 6 for Optimal_path2) is detected; hence, while the first robot (Optimal_path) continues its path without any stop due to its priority, the second robot (Optimal_path2) will wait one more time at i=3 ([x,y]=[5,6] at i = 3) since it has a lower priority than Robot 1.

2) Now, let's suppose a diagonal collision, but this time, the collision doesn't occure inside a cell, but between two diagonal cells, since no collision cell exists to use as reference, an alternative algorithm takes place to allow the good execution of system, let's take an example given in **figure 4.7**.

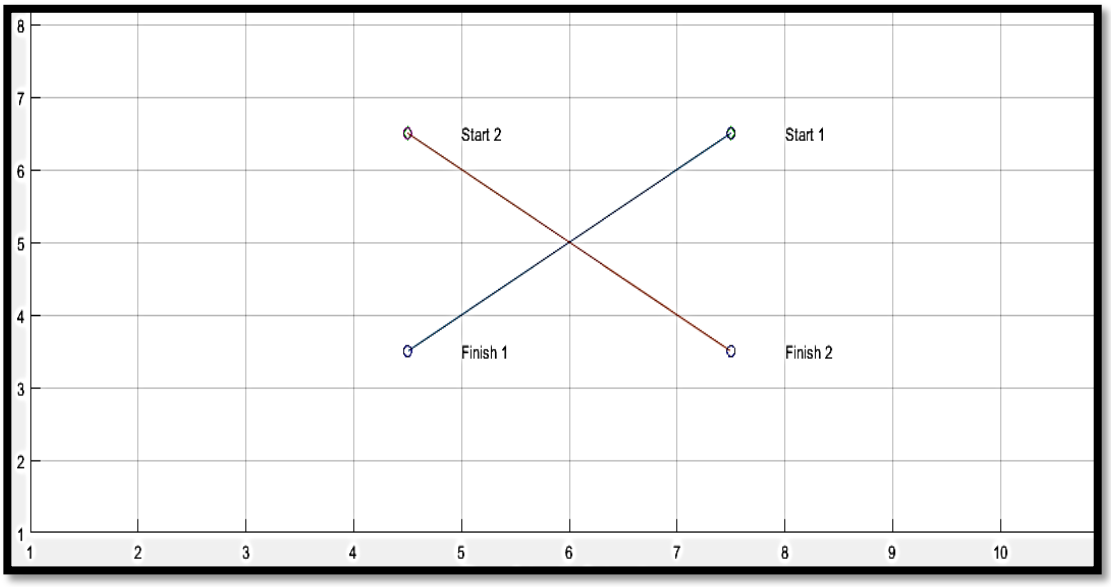


Figure 4.7: Diagonal Multi-robot collision

As we can see, the collision doesn't depend on a specific cell, the following figure represents an alternative solution that takes into account each cell before and after the collision regarding to each Optimal path is executed to optimize time and path generation algorithm.

```
A.x = OptimalPath.a[i][0]              //Robot 1's (x) position
A.y = OptimalPath.a[i][1]              //Robot 1's (y) position
a.x = OptimalPath.a[i-1][0]            //Robot 1's (x-1) position
a.y = OptimalPath.a[i-1][1]            //Robot 1's (y-1) position

B.x = OptimalPath.b[i][0]              //Robot 2's (x) position
B.y = OptimalPath.b[i][1]              //Robot 2's (y) position
b.x = OptimalPath.b[i-1][0]            //Robot 2's (x-1) position
b.y = OptimalPath.b[i-1][1]            //Robot 2's (y-1) position
i = 1
WHILE SIMULATION != END
        IF Priority.b < Priority.a     //if priority of robot "b" is smaller than priority of robot "a"

              IF (( (a.x = b.x + 1) && (A.x = B.x - 1) ) && ( a.y = b.y ))
                      OptimalPath.b[i][0] = OptimalPath.b[i-1][0]
                      OptimalPath.b[i][1] = OptimalPath.b[i-1][1]
                      i++
                      OptimalPath.b[i][0] = B.x
                      OptimalPath.b[i][1] = B.y
                      i++

               ELSE IF (( (a.y = b.y + 1) && (A.y = B.y - 1) ) && ( a.x = b.x ))
                      OptimalPath.b[i][0] = OptimalPath.b[i-1][0]
                      OptimalPath.b[i][1] = OptimalPath.b[i-1][1]
                      i++
                      OptimalPath.b[i][0] = B.x
                      OptimalPath.b[i][1] = B.y
                      i++

               ELSE   OptimalPath.b[i][0] = B.x
                      OptimalPath.b[i][1] = B.y
                      i++
               END IF
         END IF
END WHILE
```

Figure 4.8: Pseudo-code of the diagonal collision avoidance Algorithm

The result of this generated Optimal paths is given in **figure 4.9**.

Optimal_path

PLOTS VARIABLE

4x2 double

	1	2
1	7	6
2	6	5
3	5	4
4	4	3
5		
6		
7		

Optimal_path2

PLOTS VARIABLE

5x2 double

	1	2
1	4	6
2	5	5
3	5	5
4	6	4
5	7	3
6		
7		

Figure 4.9: Optimal paths depending on Diagonal collision

As we can see, the same method as Horizontal/Vertical collision is used in order to organize the traffic by taking into consideration priorities in a multi-robots environment.

3) The final collision case that may occur is the tunnel case, or to do simple, two robots facing each other, this case is more complex to be executed as an algorithm since it

is better to handle the collision in real-time and a very large map can make the system crash while searching for collision path. Fortunately, the following pseudo-code (**Figure 4.10)** which represents an alternative algorithm that should be executed by all robots using ultrasonic sensors while running their own optimal path is used to optimize the system.

```
Vector path = readFromESP32WirelessUDP()
RobotPosition.x = OptimalPath[0][0]
RobotPosition.y = OptimalPath[0][1]
Count = 1
WHILE Count <= packet.length
        ACTIVATE the Ultrasonic Sensor
        IF Priority Higher
           WHILE Distance > Minimum Distance required
               Check.x = (OptimaPath[Counter + 1][0]) - (optimalpath[Counter][0])
               Check.y = (OptimaPath[Counter + 1][1]) - (optimalpath[Counter][1])
           ASSIGN Next move corresponding to (Check.x) && (Check.y)
           END WHILE
        ELSE
           STOP Robot
           DODGE Mode
           HIGHER Priority robot pass
           RETURN and execute last move
        END IF
        Desactivate Ultrasonic Sensor
        Count++
END WHILE
```

Figure 4.10: Pseudo code of tunnel collision avoidance Algorithm

4.1.1.2 Execution time

In this part, the execution time spent by our algorithm is calculated. A* is used to find the optimal paths from starting point to the target for the three robots. To do that have considered a map with some random obstacles as shown in **table 4.1**.

Table 4.1: Samples of the time needed to calculate the paths

n*n cells	Length paths			Total number of traversed nodes			Time taken by the A* algorithm (ms)
	R1	**R2**	**R3**	**R1**	**R2**	**R3**	
6*6	6	6	6	20	19	16	103.394
20*20	19	19	17	93	129	114	351.554
50*50	48	33	19	318	259	161	560.853
100*100	57	11	10	977	73	50	$1.466x10^3$

- n*n cells represent the map size
- Length Paths represent the length path of each robot from the starting to the Final point.
- Total number of traversed nodes represent the OPEN list of how much nodes the algorithm traversed before reaching the Final point.
- The last column represents the time Taken by the A* algorithm to calculate and generate the Optimal Paths.

Remark: As we can see, the time needed to traverse and calculate the optimal path for each robot depend on the path length of each of them. The longer is the path, the bigger will be the time needed to execute the algorithm.

4.2. Implementation results

In order to ensure a good execution of the given path as explained before, each robot must translate the given information send from the unit center and received in form of packets, the vector containing information about the path inside each packet will be read by the algorithm being executed by each robot.

In **figure 4.11**, we are implementing a case where two robots are navigating in environment containing some random obstacles. We are giving a straight line path for the first one, the one with the highest priority, and a complex path for the second one which has lower priority. The generated path cells are given in **figure 4.12**.

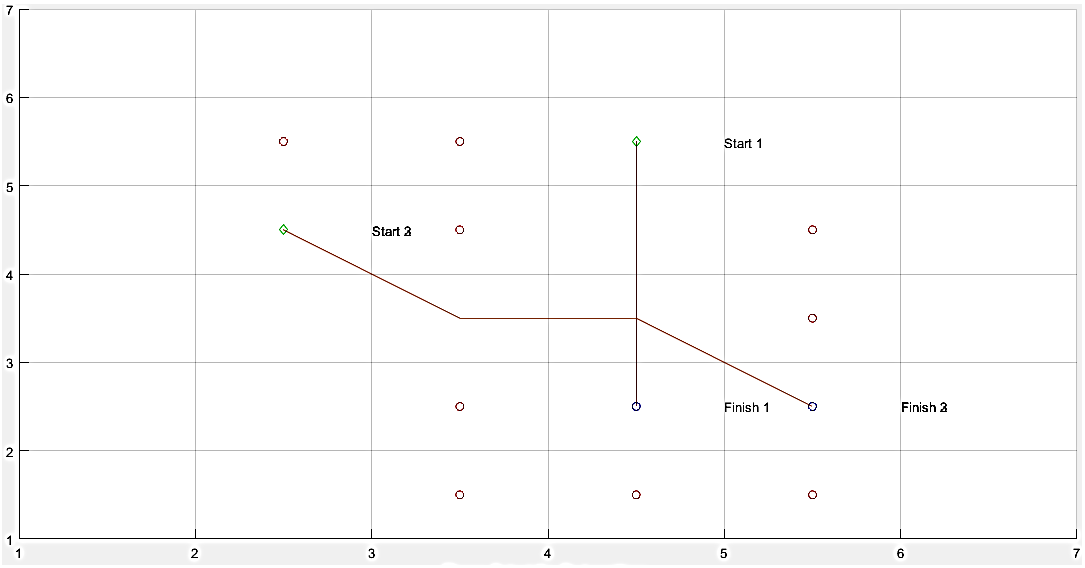


Figure 4.11: 2D environment with two robots and some random obstacles

Optimal_path

PLOTS | VARIABL

4x2 double

	1	2
1	4	5
2	4	4
3	4	3
4	4	2
5		
6		
7		
8		

Optimal_path2

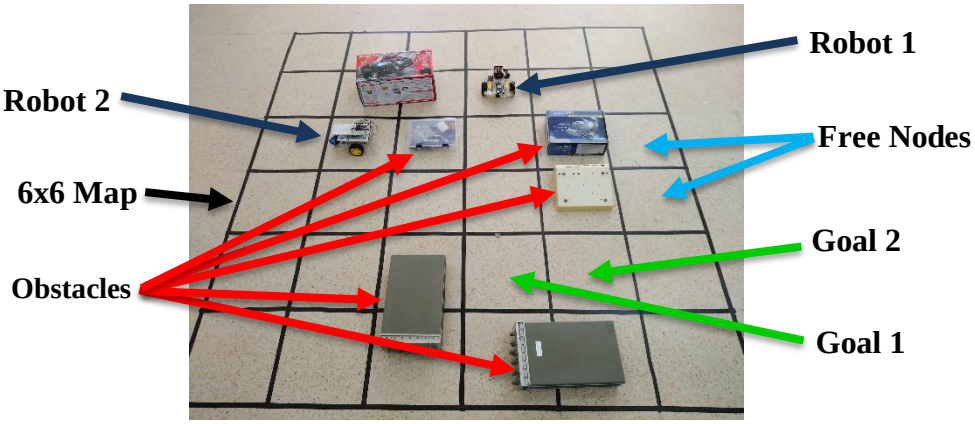
PLOTS | VARIABL

5x2 double

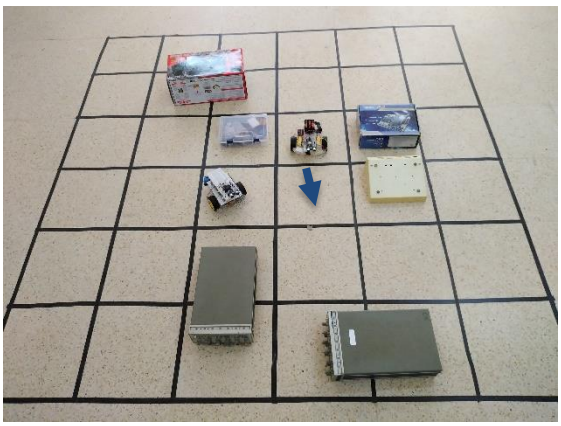
	1	2
1	2	4
2	3	3
3	3	3
4	4	3
5	5	2
6		
7		
8		

Figure 4.12: Optimal Paths including dynamic/static avoidance

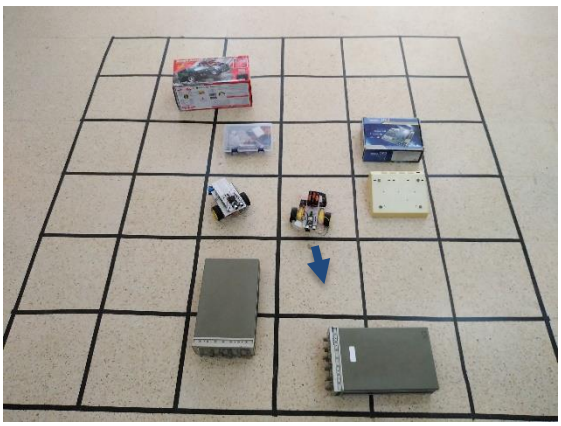
Applying this example in real world, we get the following results:



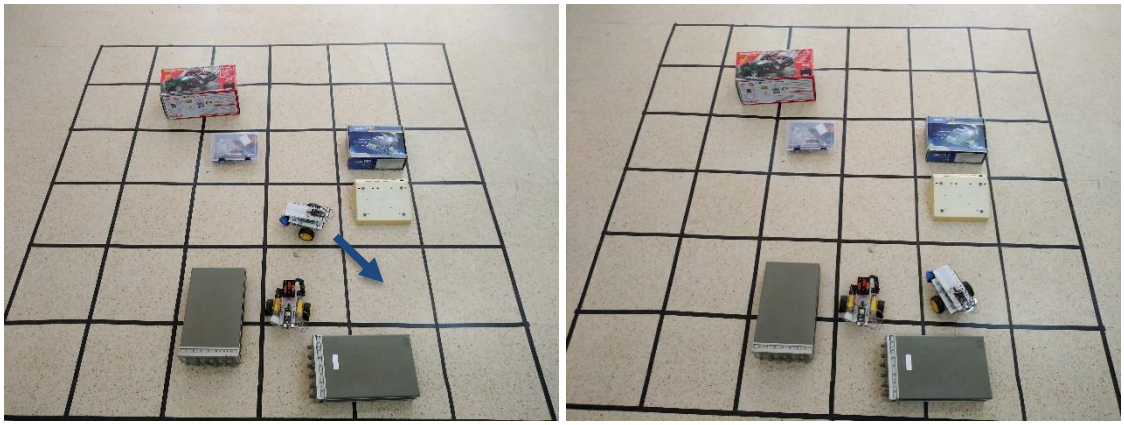
a



b



c



d **e**

Figure 4.13: pictures a, b, c, d and e show the behavior of the robots during the implementation

The following **Figure 4.13** represents the displacement of a multi-robot system from their starting point **"picture a"** successively until the final destination **"picture e"** with a collision avoidance case at **"picture c"**, while following table (**table 4.2**) represents the error factor at each iteration:

Table 4.2: comparison of implementation results with simulation results for Robot 1

Steps	X	Y	Distance error[1] (cm)
a	4	5	0
b	4	4	3.4
c	4	3	5.7
d	4	2	9.6

Table 4.3: comparison of implementation results with simulation results for Robot 2

Steps	X	Y	Distance error (cm)
a	2	4	0
b	3	3	3.8
c	3	3	3.8
d	4	3	6.1
e	5	2	8.6

As it is shown in **table 4.2** and **table 4.3**, the error factor from the center stay almost the same, however, the longer is the path, the bigger will be the error factor at the end of the Optimal Path execution.

The result is that the error factor is directly proportional to the direction changes and path length.

4.3. Problems and alternatives

Even if everything is working well in simulation part, some problems have been encountered during the implementation part. The two major problems were the lack of precision for the two encoders of the two robots, which drives the robot most of time to deviate from its real orientation, and the power supply which is not stable since we are using a battery for a better mobility. These two problems increase the error factor by increasing path length.

An alternative solution was introduced in order to decrease the error factor. A closed loop was added for the encoder part; so, each robot is actualizing its power in order to synchronize the two wheels and get an acceptable result even if it's not precise at 100%. For the battery part, the PWM (Pulse-width Modulation) is used to reduce battery consumption and get the desired supply. Since a low supply may not be sufficient to drive the dc motor because of the weight, and a higher supply leads to spinning wheels which is a major problem for orientation accuracy; so, PWM is a good solution to regulate the power consumption.

Eventually, the error factor can also be reduced using a more accurate/expensive equipment.

4.4. Conclusion

In this chapter, the implementation of the simulation part gave the expected results with an acceptable error since the robots are executing the preplanned optimal paths without collision, by dividing the software between the computer and robots, the system seems to be more autonomous and robust since robots can follow their own path even if the central unit is disconnected, hence losing the central unit or any robot doesn't mean losing the entire system.

General Conclusion

In this project, we focused on multi robot hybrid navigation system with unknown obstacles, Based on modified A* algorithm and ultrasonic sensors. We call it a hybrid system because, it starts as a centralized system where the PC is the central unit that do all the calculations and send it to robots. After that, the work of the PC stops and the relation between robots become a peer to peer relation, where the robots are autonomous but they need to communicate with each other to check the priorities.

As it seems, Artificial intelligence is broadening its path and use to excel more in future applications, and experts said that the machine will replace the humans, to perform more and more sophisticated tasks with high efficiency. What bring to our minds the word efficiency, which is the main subject of all the studies to make systems work with optimal resources and give perfect results to save time and money.

For this purpose, we used a modified A* algorithm, after searching for the optimal path from start to destination point for each robot, the modified part of the algorithm takes into account the collision point at the same moment “T” to makes changes on less prioritized robots as explain before, this algorithm could be associated with other algorithms to make the navigation smooth.

At the end, the results of the simulation where perfect, the program was executed in an optimal time without bugs and it could handle until 3 robots in a map of 100*100 and a great number of obstacles. The implementation part was also good, the robots were navigating from starting to finish point without collision with static obstacles/robots with respect of priorities. We had some error due to the hardware, the robots arrived at the target points with some deviation from the center of the cell, the reason for that was discussed before and we proposed some ideas to make it more precise.

We also proposed some features and algorithms that can be added or modified to make the system more intelligent and decrease the cost.

Future expansion

For future researches, we suggest making a decentralized system that could handle a very large number of robots, those robots are working together to increase the speed of scanning their environment quickly by combining the scan results of all of them, this work allows to design a map for unknown 2D/3D environments using high efficiency and wide range sensors that span 360° like the LiDAR. This feature will help for space discoveries and interventions in dangerous areas.

Since this system has been optimized for a 2D even environment, a RADAR can also be added in order to make the multi-robot system more flexible in a complex environment, such as 3D uneven environments with bumps.

The A* algorithm can also be combined with Potential Field/Bug 0 algorithms for smooth turns when passing near obstacles.

In the other hand, we need to make other studies to make a more stable control system for the motors and energy efficiency and speed.

References

[1] Development and Implementation of Even And Uneven Environment Navigation Strategy by BELKALEM JUGURTHA and ABED MOHAMMED AMINE in 2016 at IGEE (Boumerdes).

[2] Multi-robots navigation in communicating environment by MAOUCH Bilel and BOUCHAREB Omar Farouk in 2018 at IGEE (Boumerdes).

[3] Handbook of Research on Emerging Digital Tools for Architectural Surveying, Modeling, and Representation by Stefano Brusaporci –University of L'Aquila, Italy.

[4]Tzafestas SG. Introduction to mobile robot control. London: Elsevier, 2013.

[5] H. M Choset. Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT press, 2005.

[6] Guo J, Gao Y and Cui G. Path planning of mobile robot based on improved potential field. Inform Technol J 2013; 12(11): 2188–2194.

[7]Automated generation of Geometrically-Precise and Semantically-informed Virtual Geographic Environments Populated with Spatially-Reasoning Agents – Mehdi Mekni

[8] K. N. McGuire, G.C.H.E. de Croon and K. Tuyls, A Comparative Study of Bug Algorithms for Robot Navigation

[9] Dave Ferguson and Anthony Stentz, Technical Report CMU-TR-RI-05-19,The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-Uniform Cost Environments

[10] Robot Motion Planning, Jean-Claude Latombe, Stanford University.

[11] Pioneer 3 Operations Manual with MobileRobots Exclusive Advanced Robot Control & Operations Software, Pioneer 3 Operations Manual, Version 3, MobileRobots Inc, January 2006.

[12] Ultrasonic Ranging Module HC - SR04, datasheet, https://www.mouser.com/ds/2/813/ HCSR04-1022824.pdf

[13] http://androminarobot-english.blogspot.com/2017/03/encoder-and-arduinotutorial-about-ir.html.

[14] Datasheet L293, L293D QUADRUPLE HALF-H DRIVERS SLRS008C − SEPTEMBER 1986 − REVISED NOVEMBER 2004

[15] Garber, Megan (2014-06-23). "'Why-Fi' or 'Wiffy'? How Americans Pronounce Common Tech Terms". The Atlantic. Archived from the original on 2018-06-15.

[16] ESP32-WROOM-32 Datasheet Version 2.8 Espressif Systems Copyright © 2019/www.espressif.com

[17] https://community.cisco.com/t5/networking-documents/tcp/ta-p/3114870 .