

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 Power and Control

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

MASTER

In Electrical and Electronic Engineering

Option: Control

Title:

**Autonomous Mobile Robot Path
Planning Using the A* “A star”
Algorithm**

Presented by:

- **MASSOUM Mohamed Samy**
- **LAKHDARI Chabane**

Supervisors:

Dr, O. Hachour

2020/2021

Dedications

*We dedicate this work to our parents, who helped us through
our academic endeavors and nourished our dreams and wishes.
To our brothers and sisters whom we shared our success and failures.
To our friends whom we learned and experienced life with.
To the staff of IGEE.
Last and not least to whoever was there when we needed.*

Samy & Chabane

Acknowledgments

First and foremost we thank Allah all mighty for all the blessing and strength in working and completing this modest project.

*We would like to express our outmost gratitude to our supervisor **Dr.Hachour** for guidance, encouragement and support through the period of working to complete this project, where we learned and developed new experience in the field.*

Abstract

The ability to search and plan a path for a robot is highly essential especially that path planning is one of the main steps for complete robot navigation; this project and its report oversee the necessary steps and grounds for efficient path planning through a preset workspace, to perform this, choosing the best-suited type of map through many possible models is the first step, after the working space is set, a range of different versions of the A* algorithm not similar in their heuristics are tested in a Python IDE to generate the optimal path alternated through selected cases of obstacles placements and map sizes, the best version was chosen then is tested in a MATLAB environment for the final simulation to search and derive the optimal path, where random obstacles are placed in the map after setting the start and goal position by choice, allowing for a solution fitting for a mobile robot navigation problem.

Table of Content

Dedications.....	I
Acknowledgements.....	II
Abstract	III
Table of Content	IV
List of Figures	VIII
List of Tables.....	X
List of Abbreviations	XI
General Introduction.....	XII
 Chapter 1: Introduction to Autonomous Mobile Robots	
1.1 introduction	1
1.2 Automated and Autonomous Robot.....	2
1.3 The See Think Act Cycle.....	3
1.4 Perception.....	4
1.4.1.Sensor classification.....	4
1.5 Localization.....	6
1.5.1.Some localization methods.....	6
1.6 Path planning.....	7
1.6.1. Graph search.....	8
1.6.2.Potential Field Path Planning Strategies.....	9
1.6.3.Obstacle avoidance.....	9
1.7 Motion Control.....	9
1.7.1. Kinematics.....	9

1.7.2. Dynamics	10
1.7.3 Degree of Freedom.....	10
1.8 Conclusion	11

Chapter 2 : Model Building and Path Planning Method

2.1. Introduction.....	12
2.2. Mobile Robot Model.....	12
2.2.1. Differential Drive Robot.....	13
2.3. Path planning.....	16
2.3.1.Map Building	16
2.3.1.1. Configuration Space (C-space).....	16
2.3.1.2. Discretization.....	19
a. Combinatorial Planning.....	19
b. Sampling-based Planning.....	19
2.3.1.3. Graphs.....	20
a.Grid graphs.....	22
b.Visibility Graph.....	23
c.Voronoi Diagram.....	23
2.3.2. Graph Search.....	25
2.3.2.1 Uninformed Search Algorithms.....	25
a.Breadth First Algorithm.....	26
b.Depth First Algorithm.....	26

c.Dijkstra/Uniform Cost Search (UCS).....	27
2.3.2.2 Informed Search Algorithms.....	28
a.A Star (A*) Algorithm.....	29
b.Greedy Best-first Search Algorithm	30
2.3.3. Criteria to compare algorithms.....	30
2.4 A Star Search Algorithm.....	31
2.4.1.Description.....	31
2.4.2. Use of Heuristics and Advantages.....	31
2.4.3.Pure Heuristic Search.....	32
2.4.4.Algorithm of our A* search.....	32
2.4.5. Optimality.....	35
2.5 .Conclusion.....	35

Chapter 3 : Simulation and results

3.1 Introduction :.....	36
3.2 Simulation Apparatus:.....	36
3.2.1 Hardware:.....	36
3.2.2 Software:.....	36
3.3 Simulation:.....	36
3.3.1 Simulation I:.....	36
a.First Scenario:.....	37
b. Second Scenario:.....	41
3.3.2. Cross examination of Simulation I and Discussion:.....	45
3.3.3. Simulation II:.....	49

3.4 Conclusion:.....51

General conclusion52

1. Notes to conclude:.....52

2. Future work:..... 53

 a. Concerning the environment:..... 53

 b. Concerning the search heuristic:..... 53

Appendix

References

List of Abbreviations

GPS	Global Positioning System
Lidar	Light Detection and Ranging
Ψ	State space matrix representing Robot in the global reference frame
Ψ_R	State space matrix representing Robot in the local reference frame
C-Space	configuration space
RRT	Rapidly exploring Random Trees
PRM	Probabilistic Roadmaps
(A^*, h_i)	A star algorithm with specified heuristic h
P_o	Optimal Resulting Path

List of Figures

Fig.1.1- Nasa Perseverance Rover from March mission 2020 [5].....	2
Fig.1.2- Industrial welding Robot.....	3
Fig.1.3- Autonomous Robot See Think Act cycle	3
Fig.1.4 - Lifelines of some important contributors to estimation technology [9].....	7
Fig.1.5 - The office chair example of a Holonomic System.....	9
Fig.2.1 – Presentation of robot in 2D axis.....	13
Fig.2.2 – Environment and Robot axis configuration and rotation in 3D space.....	14
Fig.2.3– configuration space of the robot environment.....	17
Fig.2.4– path planning in the configuration space of the robot environment.....	18
Fig.2.5– Graph configuration, edges and nodes.....	20
Fig.2.6– Weighted edges and nodes in a directed Graph configuration.....	21
Fig.2.7– Grid map with cells centers as nodes.....	22
Fig.2.8– Grid map with cells corners as nodes.....	22
Fig.2.9- (a) obstacles occupying Min cell space, (b) resolution augmentation for obstacles to fit cells.....	23
Fig.2.10 - (a) The two-dimensional map of a robot's test area. (b) The Voronoi diagram after the elimination step,(c) general representation of Voronoi diagram	24
Fig.2.11 - Breadth First Algorithm example.....	26
Fig.2.12 - Depth First Algorithm example.....	27
Fig.2.13 – examples of the Dijkstra algorithm (a)(b).....	28
Fig.2.14- example of grid map searched using a A*	29

Fig.2.15- explanation of the A star (A*) cost function	31
Fig .2. 16- Graph Diagram incorporating the typical A* search Algorithm.	33
Fig..3.1- simulation in 25*25 grid maze structured obstacles ,(a) Dijkstra algorithm’s path planning result,(b)A* Algorithm with h_1 heuristic ,(c) A* Algorithm with h_2 heuristic ,(d) A* Algorithm with h_3 heuristic.....	38
Fig.3.2- simulation in 25*25 grid randomly structured obstacles ,(a) Dijkstra algorithm’s path planning result,(b)A* Algorithm with h_1 heuristic ,(c) A* Algorithm with h_2 heuristic ,(d) A* Algorithm with h_3 heuristic.....	40
Fig.3.3- simulation in 50*50 grid maze structured obstacles ,(a) Dijkstra algorithm’s path planning result,(b)A* Algorithm with h_1 heuristic ,(c) A* Algorithm with h_2 heuristic ,(d) A* Algorithm with h_3 heuristic.....	42
Fig.3.4- simulation in 50*50 grid maze structured obstacles ,(a) Dijkstra algorithm’s path planning result,(b)A* Algorithm with h_1 heuristic ,(c) A* Algorithm with h_2 heuristic ,(d) A* Algorithm with h_3 heuristic.....	44
Fig.3.5.- clustered bar chart of Scenario I case 1 results.....	46
Fig.3.6.- clustered bar chart of Scenario I case 2 results.....	46
Fig.3.7- clustered bar chart of Scenario II case 1 results.....	47
Fig.3.8- clustered bar chart of Scenario II case 2 results.....	47
Fig.3.9- Matlab simulation of (A*, h_2) in 25*25 grid.....	49
Fig.3.10. Matlab simulation of (A*, h_2) in a 50*50 grid.....	49

List of Tables

Table 1.1 Classification of sensors used in mobile robots.....	5
Table 3. 1 Time, explored nodes and path length of each algorithm in a maze like map of 25*25 grid.....	39
Table 3. 2 Time, explored nodes and path length of each algorithm in a random set obstacles map of 25*25 grid.....	41
Table 3. 3: Time, explored nodes and path length of each algorithm in a maze like map of 50*50 grid.....	43
Table 3. 4: Time, explored nodes and path length of each algorithm in a random set obstacles map of 50*50 grid.....	45
Table 3. 5: Average running time of 10 repetitions of each grid size.....	50



General introduction

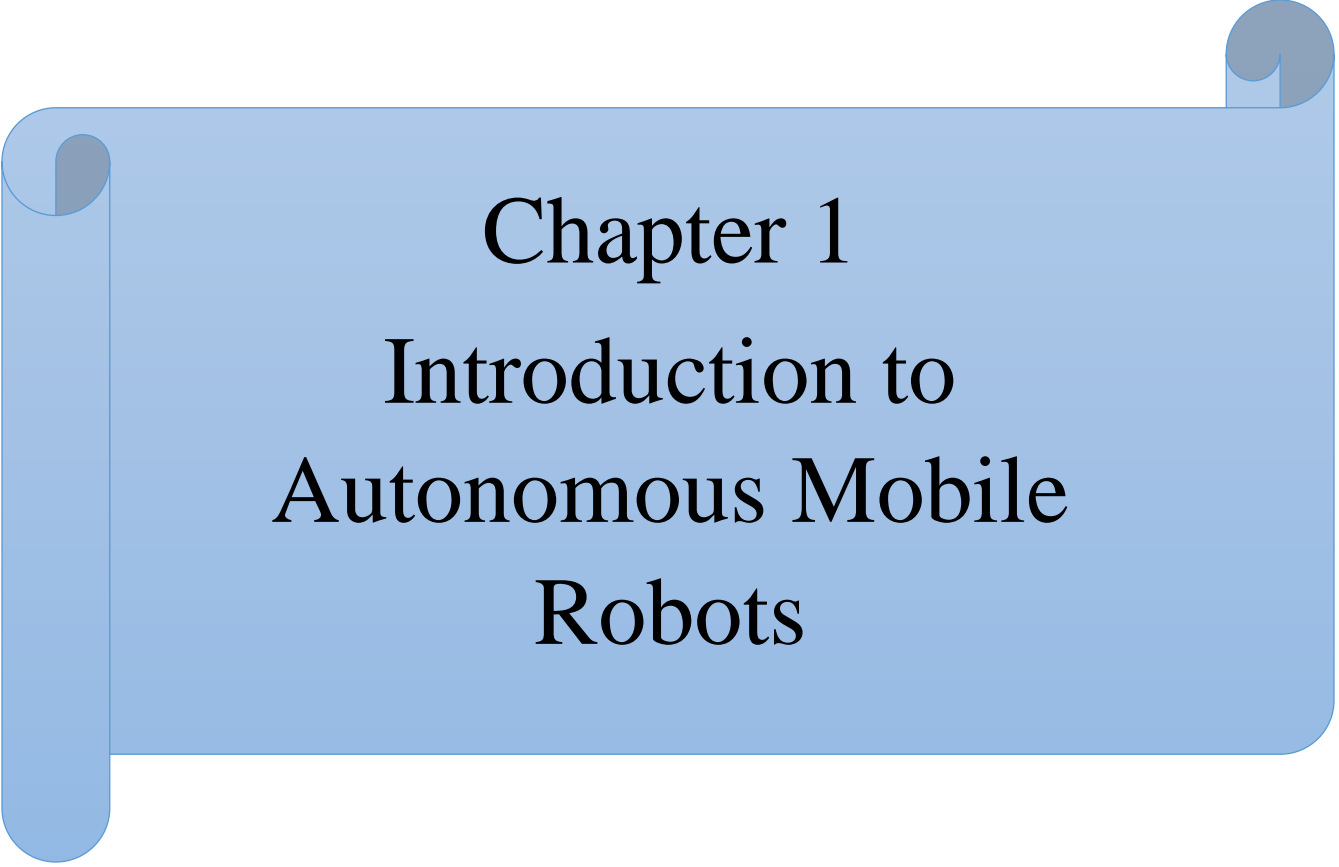
General Introduction

The field of Robotics merges many disciplines from electrical and electronics engineering to computer science and mechanics, control, and automation engineering, this can be a consequence of the variety of robots types, in addition to the different environments to which the robots are exposed. Since the creation of the first robot, its application has become more than just a fictional idea but the dawn of new advancements in modern life.

One of the types that are on unceasing development is the mobile robot: Typically, this last knows where it is and it knows where it should go to, it has the map of the environment and it can navigate without bumping into obstacles. Making smarter robots is now a race! Autonomous mobile robots are one of the most discussed subjects in the scientific community. Implicitly trained robots are the future of robotics thus creating high precision sensors and writing fast adaptive algorithms is a crucial activity.

Rapid algorithms are quite a profound idea, it is the mirror of how fast the robot acts to find the optimal path, actually this what brought the idea of motion and path planning too many algorithms were designed to solve this problem like depth first search, breadth first search, Dijkstra algorithm, A star ...

This work aims to clarify the difference between the different algorithms used in graph search problems, also the effectiveness of heuristics to guide the differential drive robot to its goal location in random obstacles map and maze type map. Finally, the discussion of each heuristic of A star algorithms to reduce the number of iterations through the nodes and edges to refine the rate of improvement in avoiding the obstacles and finding the optimal path.



Chapter 1
Introduction to
Autonomous Mobile
Robots

1.1 Introduction:

A Robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks[2], where Autonomous Mobile Robots have a slightly different definition meaning a kind of Robot with the ability to move and navigate in different mediums “Water Air and land “.

The ability of a mobile robot to travel makes it useful in many fields, which grants it access to a variety of disciplines military, industrial ,transportation ,medical, agriculture, and mining, but for a well depiction of Navigation a mobile robot need a certain steps looped to insure a much concrete one, these steps can be summarized in these questions:

1. Where the Robot is?
2. What is the Robot destination?
3. How is the Robot getting there?

To answer these three questions the Robot need to have:

1. A model of the environment (built or predesigned).
2. Perception and analysis of the environment.
3. Define its situation in the environment.
4. Plan and execute its movements.

To describe a Robot kinematics is the most fundamental aspect of design, analysis, control, and simulation. The robotics community has focused on efficiently applying different representations of position and orientation and their derivatives with respect to time to solve foundational kinematics problems [3], whereas the dynamic equations of motion provide the relationships between actuation and contact forces acting on robot mechanisms, and the acceleration and motion trajectories that result [4].

1.2 Automated and Autonomous Robot:

An autonomous robot in contrast to an automated one is apparent in the ability to spawn its own decisions regarding motion, future position and speed which grants it a high level of autonomy, while an automated robot is a slave to the orders of the master in case an engineer or a technician, which means it requires constant intervention and monitoring with a slight or no autonomy and even if it doesn't require tinkering the processes it performs must be knowledgeable and repeated.

As an example of the two kinds, we can refer to these two scenarios:

Autonomous Robot: A Humanoid Robot able to navigate indoor and outdoor environments with no help from outside factors, meaning it can sense its terrain and interact with it on its pure cognition.

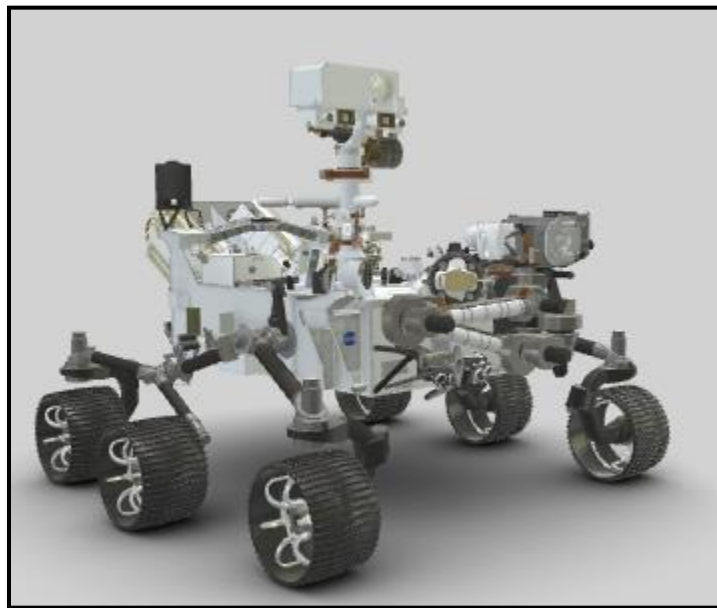


Fig. 1.1- Nasa Perseverance Rover from March mission 2020 [5].

Automated Robot: A robotic arm in a factory responsible for soldering parts in an assembly line, the ability to perform these tasks must be linked to these tasks high repetitiveness and predictability, while constantly monitoring the robot performance.



Fig. 1.2- Industrial welding Robot.

1.3 The See Think Act Cycle:

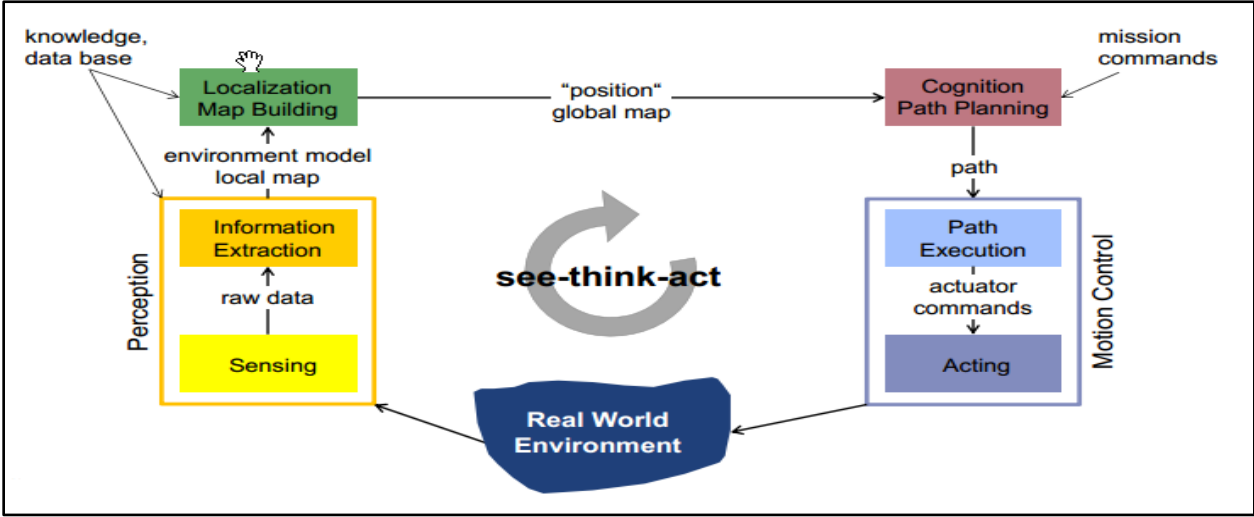


Fig. 1.3- Autonomous Robot See Think Act cycle.

The whole block referred to as robot navigation, which is a pivotal part of the autonomy of the mobile robot, can be dissolved to its main building blocks as shown in (Fig -3-) Perception, Localization, Cognition or Path Planning, Execution or Motion Control.

1.4 Perception:

One of the most important tasks of an autonomous system of any kind is to acquire knowledge about its environment [6]; this can be performed using information fed through measurements from different sensors.

Since the robot is moving in an unforeseen environment it may encounter new characteristics, it is only logical to use a variety of sensors, some measure simple values such as wheel encoders, others are complex and sophisticated sensors able to give precise and critical information about the robot position for example Lidar sensor, which uses laser to determine objects ranges based on light reflection time upon return to the receiver.

1.4.1 Sensor classification:

We classify sensors using two important functional axes: proprioceptive/exteroceptive and Passive/active [7].

Proprioceptive sensors is a term assigned to sensors responsible for the measurement of values related to the robot (internal to the system) such as battery status, motor speed.

Exteroceptive sensors acquire measurements from the robot exterior environment for example range measurement, orientation of the robot, in short these sensors can help the robot in building environment features based on their measurements.

Passive Sensors measure energy related to the environment meaning it reacts to energy imposed by the environment with no kind of interference with the phenomena as can any heat probe show.

Active Sensors are sensor that emit energy to the environment, then measure the environment reaction to that energy in order to produce measurements, this type of sensors is considered superior performance wise, but since the sensor function is to emit to receive it can interfere with the phenomena it tries to measure.

Table 1.1 Classification of sensors used in mobile robots

General classification (typical use)	Sensor (Sensor System)	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers	EC	P
	Optical barriers	EC	A
	Noncontact proximity sensors	EC	A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders	PC	P
	Potentiometers	PC	P
	Synchros, resolvers	PC	A
	Optical encoders	PC	A
	Magnetic encoders	PC	A
	Inductive encoders	PC	A
	Capacitive encoders	PC	A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A/P
Acceleration sensor	Accelerometer	PC	P
Ground beacons (localization in a fixed reference frame)	GPS	EC	A
	Active optical or RF beacons	EC	A
	Active ultrasonic beacons	EC	A
	Reflective beacons	EC	A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors	EC	A
	Ultrasonic sensor	EC	A
	Laser rangefinder	EC	A
	Optical triangulation (1D)	EC	A
	Structured light (2D)	EC	A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s)	EC	P
	Visual ranging packages		
	Object tracking packages		

A: active , P: passive, P/A: passive/active. -- PC: Proprioceptive, EC: exteroceptive.

1.5 Localization:

After a successful extraction of the environment features, a robot must be successful in determining its position in its surrounding environment, although this part may seem a straightforward method it cannot be further from the truth, since localization is highly related to the type of Robot used and the tasks it's supposed to perform, if the robot is indoor mobile robot then its localization has no use for GPS in building its map, whereas if it needs indoor and outdoor navigation the robot localization will be based on GPS for global localization and a more precise sensor for local localization, since a GPS module is inaccurate in range of few meters.

Although for most autonomous mobile robots localization is more than simply just determining position in space, but rather building a map then determine their position in that map, nevertheless It is because of the inaccuracy and incompleteness of the sensors and effectors that localization poses difficult challenges more than most parts of the navigation process.

Some localization methods:

- a) **Monte Carlo localization:** which is a localization method based on Particle Filter

Instead of describing the probability density function itself, it represents it by maintaining a set of samples that are randomly drawn from it. To update this density representation over time [8], using a sampling-based representation obtains a localization method that has several key advantages.

- b) **Kalman Filter Localization:** Theoretically, the Kalman filter is an estimator for what is called the linear-quadratic problem, which is the problem of estimating the instantaneous “state” of a linear dynamic system perturbed by white noise—by using measurements linearly related to the state but corrupted by white noise[9], the Kalman Filter is also used to predict future courses of uncontrollable dynamic systems if it's: floods, falling celestial debris or controlled dynamic systems such as rocket/missile trajectory tracking, process control in chemical systems.

According to Kalman Filter definition, it can be a powerful localization method if the concerned localization is specially a local one, this method is mainly a mathematical tool, can only model Gaussian distributions.

- c) **Markov localization:** The key idea of Markov localization is to maintain a probability density over the space of all locations of a robot in its environment [10].

So to answer the question “where am i?” one or a combination of the above localization methods is used, this does not mean these methods are all there is ,but rather the best there is in local or global localization concerning our robot.

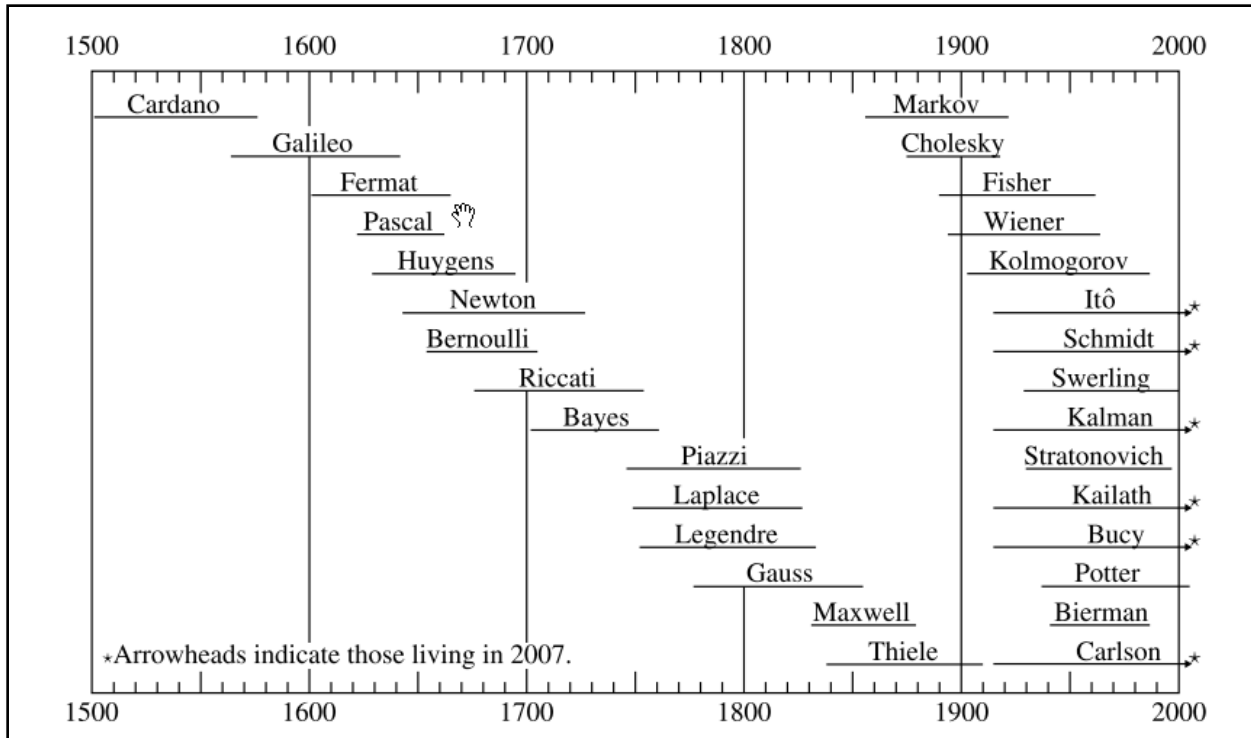


Fig.1.4 - Lifelines of some important contributors to estimation technology [11].

1.6 Path planning:

Path planning is the ability of a robot to reach a preset destination or rather move through an environment with a successful collision avoidance path between its depart and arrival, so the Robot need to apply local path planning from the data the sensors can provide which can be key in avoiding obstacles, in addition a global path planning can be necessary if the location is static and already known on a vast terrain (outdoor and indoor).

Path planning can be performed relaying on a variety of ways:

1.6.1. Graph search

Graph search techniques have been used in the field of mathematics to solve the shortest path problem between two states on a directed graph

a. Graph construction

Starting from a representation of free and occupied space, several methods are known to decompose this representation into a graph that can then be searched using any of the methods (Visibility graph, Voronoi diagram, exact cell decomposition, approximate cell decomposition). The challenge lies in constructing a set of nodes and edges that enable the robot to go anywhere in its free space while limiting the total size of the graph [12].

b. Deterministic graph search :

suppose that our environment map has been converted into a connectivity graph using one of the graph generation methods, path planning objective is to find the optimized path in the map connectivity graph, which leads to several search algorithms that have become quite popular in mobile robotics such as (Breadth-first search, Dijkstra's algorithm, A* algorithm, D* algorithm).

c. Randomized graph search:

When encountering complex high-dimensional path planning problems solving the problem can be enduring and hard to finish in a fixed frame of time, that's when randomized search becomes useful, since it forgoes solution optimality for faster solution computation [13], for example Rapidly Exploring Random Trees (RRTs) this method allows to search non-convex high-dimensional spaces by randomly building a space-filling tree [14].,

Both Graph search methods need to fulfill a set of requirements according to the application:

- Optimality.
- Completeness: Can it find all solutions for the path/ problem at hand?
- Accuracy/ precision.
- Execution Time.

1.6.2 Potential Field Path Planning Strategies :

Potential field is commonly used in the field of robot path planning. The most prominent advantage of PF algorithm is briefness. PF model consists of two kind of potential field, the gravitational and repulsion field, which based on the obstacles and the goal respectively corporately attract robot complete path planning, and the robot is treated as a point under the influence of an artificial potential field $U(q)$. The robot moves by following the field.

1.6.3.Obstacle avoidance:

During robot motion, its trajectory is mainly altered using local obstacle avoidance based on sensor feedback, because of current sensor reading and goal position in addition to robot location with regard to the goal position the robot motion can be totally functional of these variables. The obstacle avoidance algorithms depend to varying degrees on the existence of a global map and on the robot's precise knowledge of its location relative to the goal position.

1.7. Motion Control:

A robot needs certain mechanisms to allow it to move in certain terrains, the relation of robot to its environment via mechanics is called locomotion, which portrays the interaction robot and it's environment, issues regarding locomotion can be drawn as Stability(dynamic/kinematic stability, number of contact points) ,characteristics of contact (angle ,shape ,and size of contact) ,type of medium(water ,air ,land ,soft ,hard).

Since balance is not a real issue in wheeled mobile robots, as all robots with wheels tend to have a constantly achievable balance with a minimum of three wheels, instead the focus should be on controllability and maneuverability.

1.7.1 Kinematics:

Kinematics pertains to the motion of bodies in a robotic mechanism without regard to the forces/torques that cause the motion [15]; it can be used to determine how a mechanical body behaves in this case a Mobile Robot.

- a. Forward Kinematics model: if given a set of actuator positions for a robot arm, determine the corresponding pose.
- b. Reverse Kinematics model: if given a desired pose for a robot arm, determine the corresponding actuator positions.

1.7.2. Dynamics

Dynamic modeling can be performed using the laws of mechanics that are based on the three physical elements: inertia, elasticity, and friction that are present in any real mechanical system such as the robot.

1.7.3. Degree of Freedom:

Depending on the number of degrees of freedom available for motion planning, we distinguish between three types of systems:

- Holonomic Systems where the robot is able to move instantaneously in any direction in the space of its degrees of freedom.

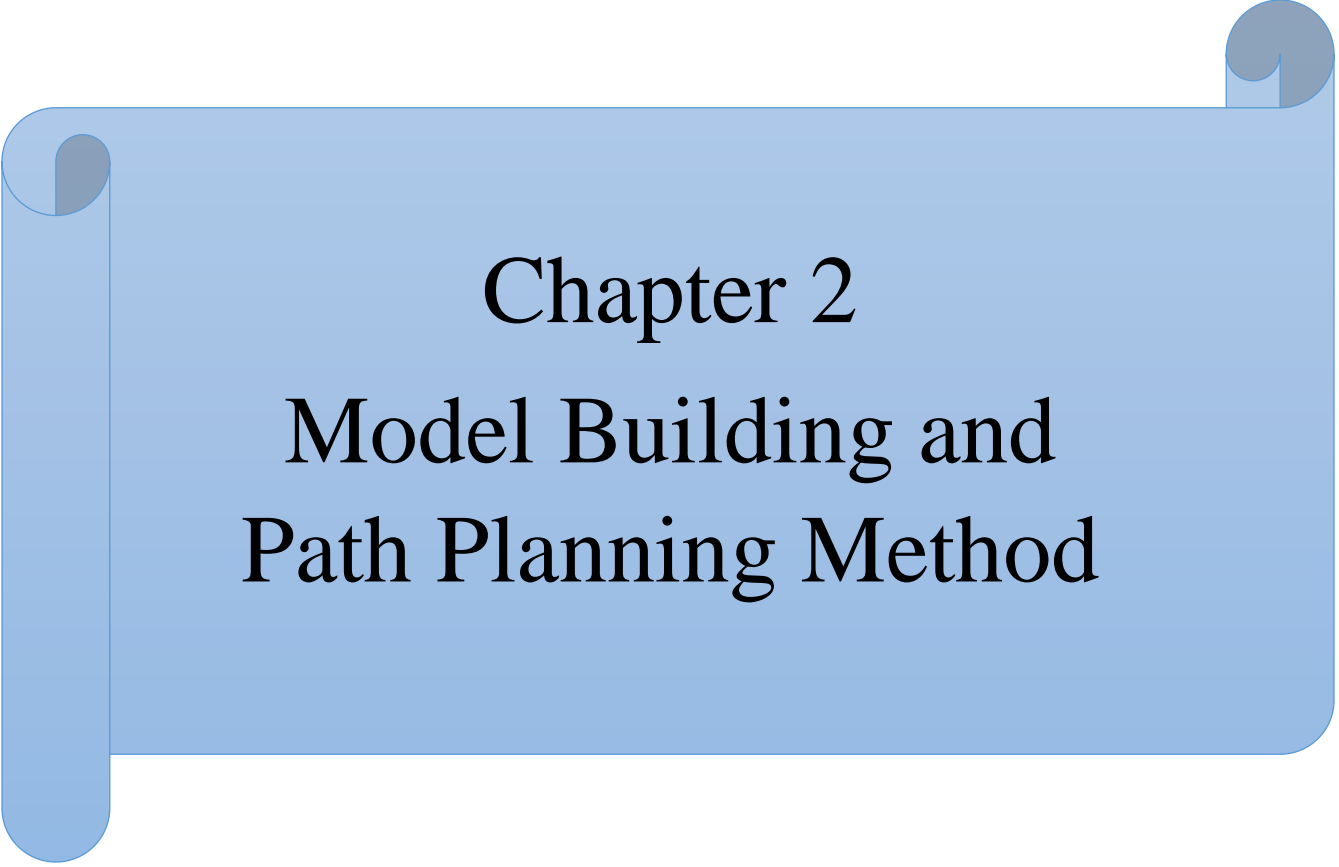


Fig.1.5 -The office chair example of a Holonomic System.

- Non-holonomic Systems where the robot is not able to move instantaneously in every direction in the space of its degrees of freedom.
- Redundant Systems with the number of DOF well above the minimum necessary for holonomic motion.

1.8.Conclusion:

This chapter introduces the key concepts in the process of robot navigation, which are crucial to understand the work presented in this project, from choosing a suited sensor mesh, solving a problem concerning localization in foreseen or unforeseen environment, to planning a path and executing that scheme through the constant control of robots position understood from its kinematics and dynamic studies.



Chapter 2

Model Building and Path Planning Method

2.1. Introduction:

As introduced in the previous chapter a robot need a set of steps to perform a fluid navigation, in this chapter we present the necessary materials to model a mobile robot mathematically, furthermore we clarify the methods we can rely on for map building and path planning from the ones previously sited.

Since our map and path-planning method is based on predefined environment and configuration, information on the functionality and approaches used will be given in this part.

A part of this chapter is the algorithm used to search for the optimal path in the path planning section, because there is a variety of algorithms it is only logical to choose the most convenient for our scenario, thus a comparison to pic the most suited approach is detailed.

2.2. Mobile Robot Model:

Selecting the right type of mobile robot is important since locomotion defines the interaction with environment, because wheeled robots are usually designed so that all wheels are in ground contact at all times. Thus, three wheels are sufficient to guarantee stable balance [16].

To be able to control a mobile robot it's important to determine the number of wheels and their maneuverability in the (X,Y,Θ) frame where x and y are the axis representing position in 2D ,while Θ is the orientation angle of the robot frame axis according to the general motion axis, the best robot for total maneuverability is the omnidirectional robot, but to achieve that the robot need an immense complex control status over the wheels ,for example a three wheeled robot with spherical wheels has a complex kinematic and dynamic equations which deliver an insight to the robot control, another example is the application of Swedish wheel in a three wheel robot setup such as all degrees of freedom are considered which requires a very precise and complex control to deliver the right navigation, thus we need a more simpler kind of robot able to perform the desired motion ,while needing moderate control ,the type of robot suitable for our applications is a Differential Drive Robot.

2.2.1. Differential Drive Robot:

A differential drive robot is robot where the two wheels do not share the same motor coupling axel and it's possible to add a dummy wheel to aide in the rotating process and stability of the frame, which mean every wheel has its own speed and motor, the difference in the two wheels speed controls the angle and speed of rotation on the robot center of gravity.

If we consider a differential drive robot with two wheels of radius R and speeds v_l and v_r , in addition to L as the length between the two wheels, Θ is the rotation angle of the robot body axis from the main motion axis, we can construct the following shape:

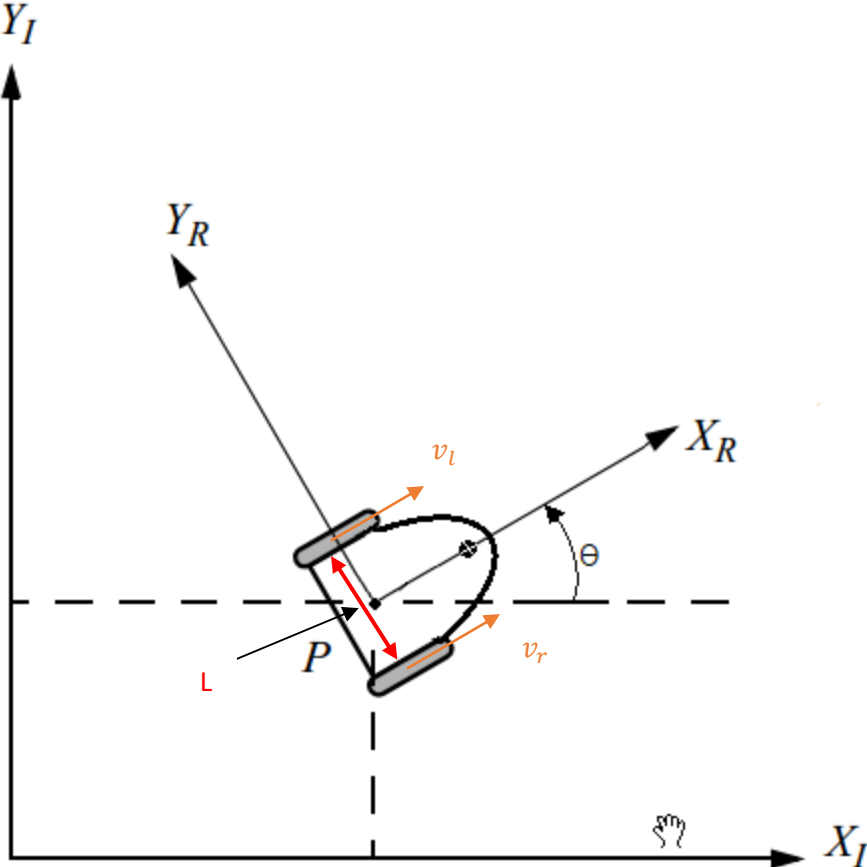


Fig.2.1 – Presentation of robot in 2D axis.

Chapter 2

Model Building and Path Planning Method

To derive the equations describing the robot we need some basic vector transformation, since we are dealing with two set of axis, to do so we use rotation matrix which transfers values from one set to the other set.

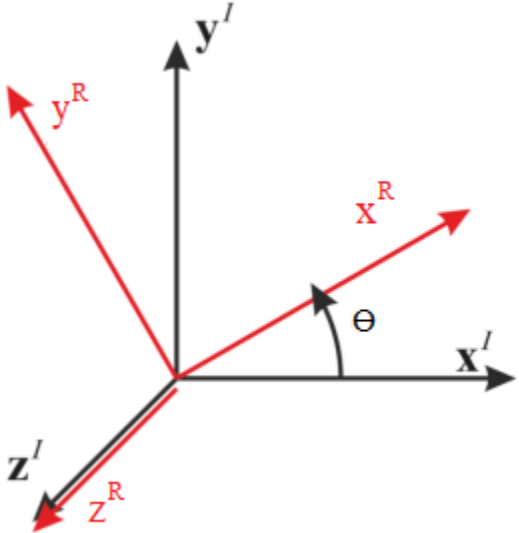


Fig.2.2 – Environment and Robot axis configuration and rotation in 3D space

To efficiently control the robot we need a constant mapping of the world coordinates to the Robot coordinates or vice versa, so in order to perform this we use a conventional rotation matrix $R(\theta)$ such that it is presented as :

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

If we consider a robot state matrix representing its position in the global reference frame, which will also lead to the representation of robot’s velocity in the global frame represented as another state space matrix as follows:

$$\Psi = \begin{bmatrix} X \\ Y \\ \theta \end{bmatrix} \text{ Leading to, by derivation } \Psi' = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} \tag{2.2}$$

The local state matrix of the robot from (2.1) and (2.2) as a mapping from the global reference frame is deduced as follows:

$$\Psi_R = R(\Theta)\Psi \quad , \quad \dot{\Psi}_R = R(\Theta)\dot{\Psi} \quad (2.3)$$

When returning to the global frame we simply multiply by the rotation matrix inverse, these equations are very helpful in forward and inverse kinematics calculations.

The mapping back to the global frame is calculated by multiplying by the inverse of the rotation matrix $R(\Theta)$, which results in equation bellow:

$$\Psi = R(\Theta)^{-1}\Psi_R \quad , \quad \dot{\Psi} = R(\Theta)^{-1}\dot{\Psi}_R \quad (2.4)$$

Ψ : State space matrix representing Robot in the global reference frame.

Ψ_R : State space matrix representing Robot in the local reference frame.

The speed of each wheel is $\omega * r$, where ω is the angular velocity of the wheel multiplied by r the radius of the wheel.

$$v_R = r * \omega_R \quad , \quad v_L = r * \omega_L \quad (2.5)$$

The above equation is a description of each wheel speed, since the speed of the robot is the average speed both wheels then:

$$v_{Robot} = \frac{v_R + v_L}{2} = \frac{r(\omega_R + \omega_L)}{2} \quad (2.6)$$

Whereas for the rotational velocity is given by:

$$\omega_{Robot} = \frac{v_R - v_L}{L} = \frac{r(\omega_R - \omega_L)}{L} \quad (2.6)$$

If we rely on equation (2.4) which is a solution for the velocity of the robot in the global plane .

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r(\omega_R + \omega_L)}{2} \\ 0 \\ \frac{r(\omega_R - \omega_L)}{L} \end{bmatrix}, [17] \quad (2.7)$$

2.3. Path planning:

Path planning is considered one of the most important parts of robot navigation, robots motion in the beginning when there was no advanced algorithms and sensors for actual precise path planning, were exclusively reliant on sensor data for constant obstacle avoidance which rendered them obsolete since many terrains need 3d sensing and some tasks are more time and distance dependent ,so the modern discipline of robotics has an abundance of algorithms with positive and negative aspects ,which makes cross-examining and choosing each one directly linked to the purpose of the robot, the optimization problem, and the environments.

2.3.1 Map Building :

To be able to construct a map where we will be able to successfully plan a path for the considered mobile robot, first we need to have some ground basis for mapping the actual world to a virtual one where calculations of path and representation of figures is more refined and measurable, then we consider the methods to construct our map which will present a variety of solutions chosen according to the type of algorithm presented.

2.3.1.1. Configuration Space (C-space):

Configuration space describes the positions of all robot points relative to fixed coordinate system, the C-Space defines a mapping which concerns a motion planning in the real world, it encloses two regions “free space” and “obstacle region”, free space is the region a robot can roam freely and contains the path we are supposed to build, obstacle region defines the region in which The robot has no access this is to avoid collisions.

Often the C-Space is discretized and it is the case in our project, which will be discussed in (2.4), we define our work space as $W = R^n$, where $O \in W$ representing the set of obstacles and

Chapter 2

Model Building and Path Planning Method

A(q) the robot in the configuration such that $q \in C$, the function A(q) is the representation of the robot in the C-Space, it can be considered as a mapping of the robot to the configuration space.

We can then consider:

$$C_{Free} = \{q \in C \mid A(q) \cap O = \emptyset\} \tag{2.8}$$

This equation guarantees no overlapping of the robot with the obstacle region.

$$C_{Obst} = C / C_{Free} \tag{2.9}$$

This equation simply defines that the obstacle space is the configuration space minus the free space and that is what the operator ‘/’ signifies in set operations.

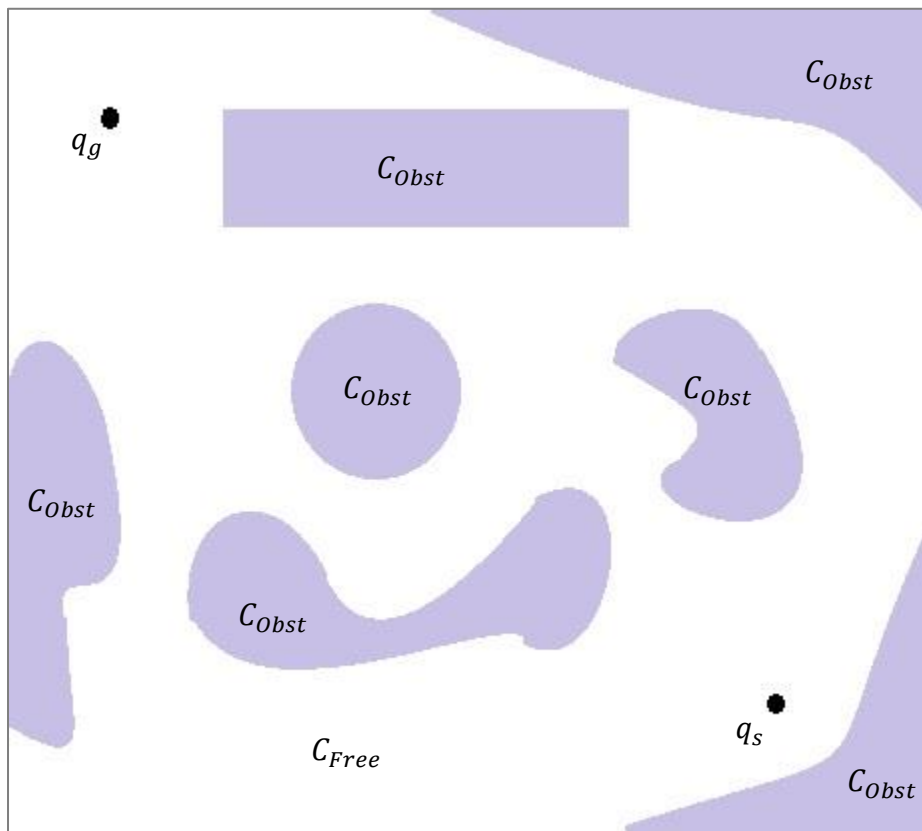


Fig.2.3– configuration space of the robot environment.

We define:

Chapter 2

Model Building and Path Planning Method

C: as the configuration space.

q_s : As the start configuration.

q_g : As the Goal configuration.

The path planning then amounts to finding a path that connects the start point and goal point in the free space defined by equation (2.8), which is defined by the function:

$$\lambda : [0,1] \longrightarrow C_{Free} \quad \text{the function is only able to map to the free space exclusively.}$$

With $\lambda(0) = q_s$ and $\lambda(1) = q_g$

After considering the setting, we can do the planning by considering the robot as a point in C-Space.

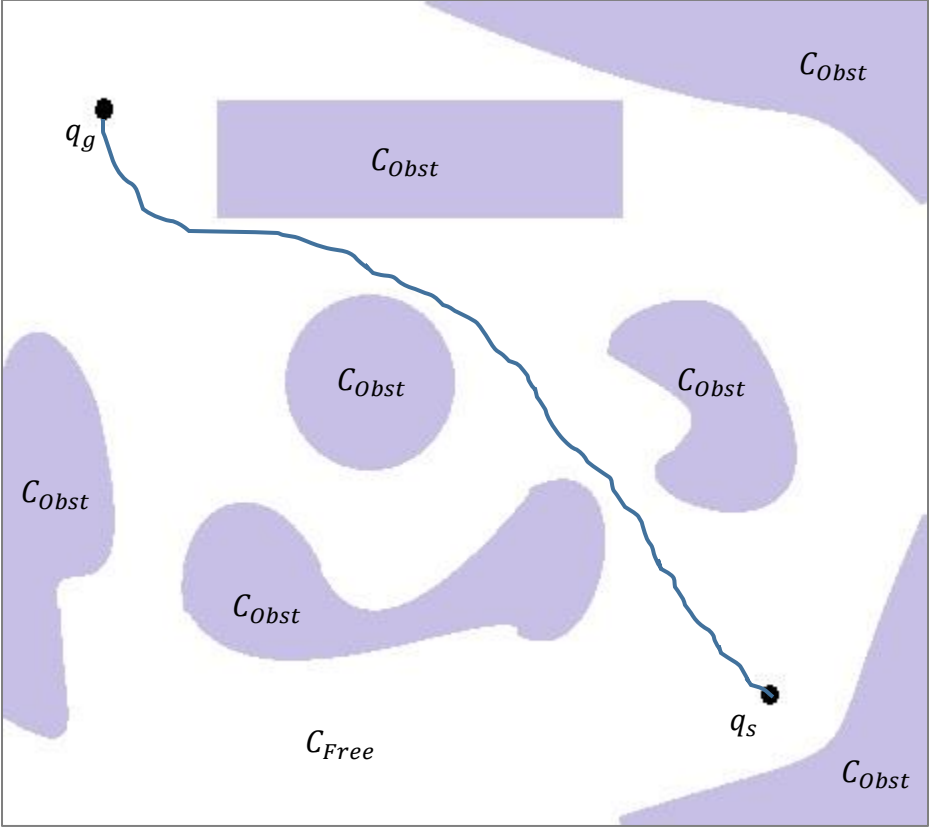


Fig.2.4– path planning in the configuration space of the robot environment

2.3.1.2. Discretization:

The configuration space is often discretized for path planning, a continuous terrain can pose a hard obstacle if we are trying to find a path, that is why we tend to discretize it for smooth path planning, in general there are two approaches to discretize C-Space:

a. Combinatorial Planning :

This method characterizes the free space C_{Free} where it captures the connectivity of the C_{Free} into a graph and finds solutions that formulate a path, this is very efficient in low-dimensional spaces 2D even 3D space; an example of this method can be an occupancy grid.

b. Sampling-based Planning :

This method incrementally searches the C-Space for solutions by using collision avoidance heuristics, where we sample possible configurations so we are constantly expanding and sampling.

After choosing the method to discretize the C-Space, it is then possible to choose the type of algorithms compatible with it, for combinatorial planning graph search algorithms are an inherited solution, but for Sampling-based planning we can refer to RRT, RRT*, PRM algorithms as a fraction of sampling based algorithms, but these algorithms have many drawbacks :

- A sampling-based planning algorithm finds paths by sampling random points in the environment. Heuristics are used to maximize the exploration of space and bias the direction of search. This makes these algorithms fast, but neither optimal nor complete.
- As the resulting paths are random, multiple trials might lead to very different results.

There is no one-size-fits-all algorithm for a path-planning algorithm and care must be taken to select the right paradigm (single-query vs. multi-query), heuristics, and parameters [18].

Chapter 2

Model Building and Path Planning Method

So since, we are considering a combinatorial planning method, which is compatible with our type of problem, it is necessary to define graphs and graph search algorithms concerning occupancy grids.

2.3.1.3. Graphs:

Graphs are mainly an abstract representation of connection between multiple objects, they are very important as they can describe different phenomenon and represent many problems for tailored solutions.

Graphs are a collection of nodes N and edges E , pairs of nodes are connected using edges which forms a network, we represent a graph based on its edges and nodes $G(N, E)$ which is commonly constructed from pre-defined maps or sensor data which in our case a predefined map that can be put as random as the user sees fit.

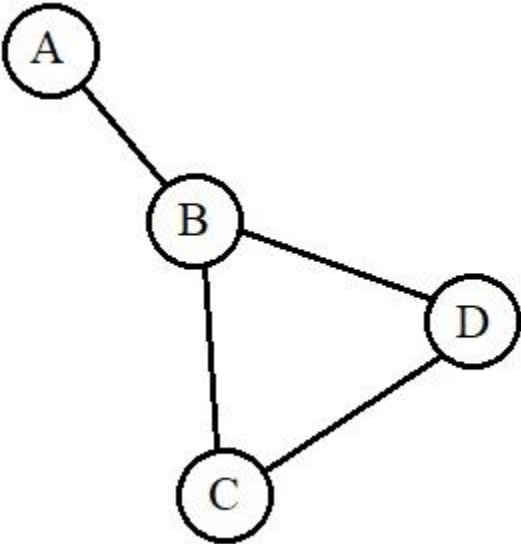


Fig.2.5– Graph configuration, edges and nodes.

Chapter 2

Model Building and Path Planning Method

In the example of **Fig.2.5**, we can notice an illustration of a simple graph, which we can define as:

Nodes: A, B, C, D

Edges: (A, B), (B, C), (B, D), (C, D).

Graph edges can be weighted or unweighted meaning they can have values such as distance, time..., resulting in uniform graphs with equal edge values or uninform graphs having unequal edge values, or no values at all which are less common in path planning, nodes also have values which indicate position or edge related results such that each node has relation to other nodes indicating their interconnections.

In addition, edges can have direction in one or both ways; this method simplifies the problem and directs the solution.

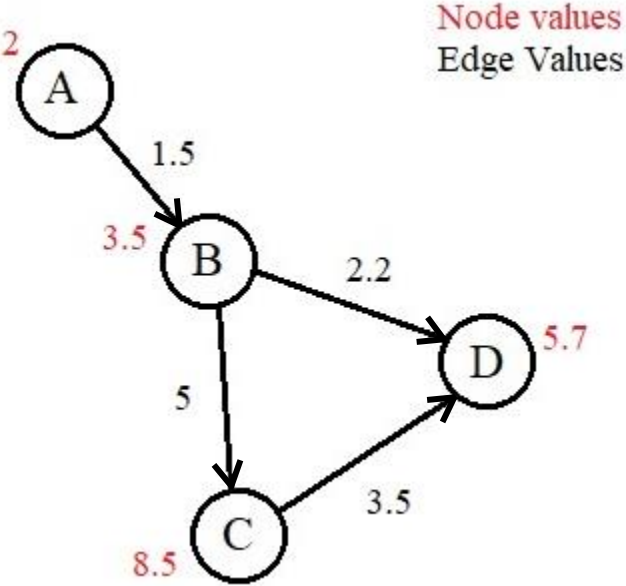


Fig.2.6– Weighted edges and nodes in a directed Graph configuration.

a. Grid graphs:

Grids are the most popular of graphs, most commonly lattice graphs, which are geometric structure that show a repetitive property, often used to describe atoms in solids so it originated from chemistry, a grid is one of the most used structures in 2D and highly used representation in robotics, which is conducted by forming square discretization of the workspace, where nodes can be the center points of the cells or each of the corners.

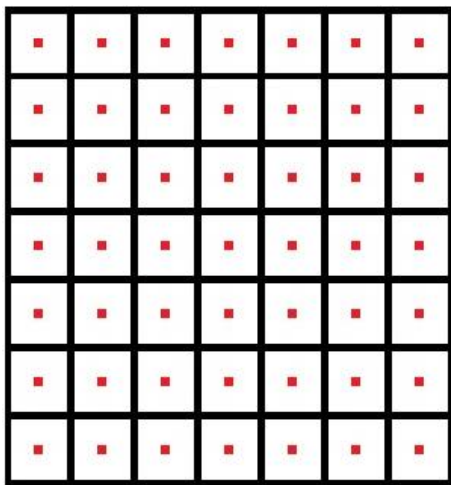


Fig.2.7–Grid map with cells centers as nodes.

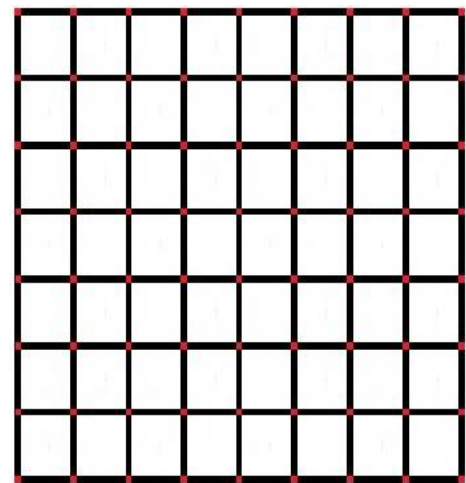


Fig.2.8–Grid map with cells corners as nodes.

When we discretize a map, we might lose accuracy that is why we can solve this by manipulating resolution so for cells with minimum occupancy can be divided into smaller cells for a representation that is more accurate as it is show in **Fig.2.9**.

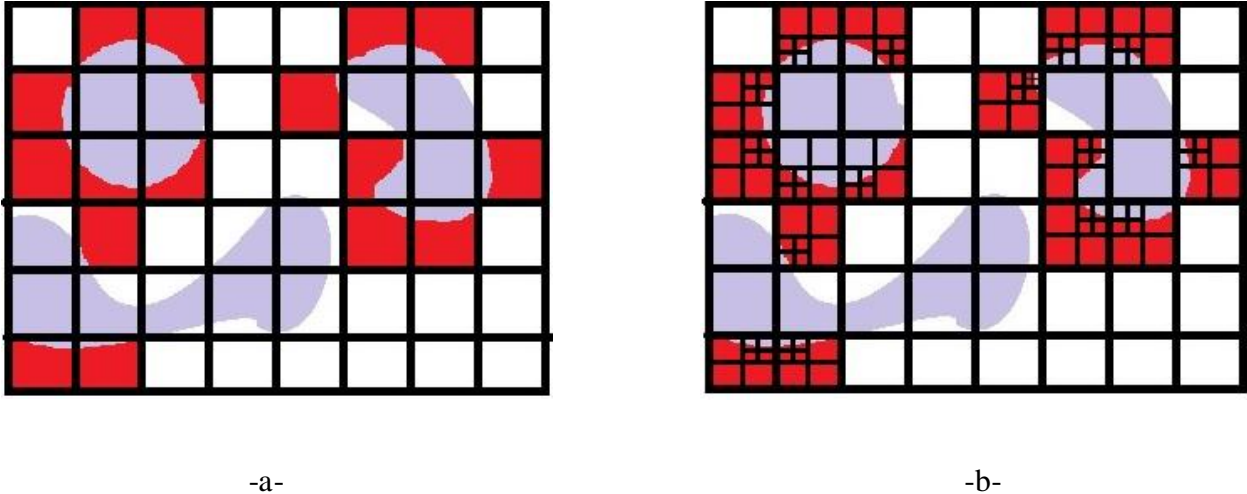


Fig.2.9 - (a) obstacles occupying Min cell space, (b) resolution augmentation for obstacles to fit cells.

b. Visibility Graph:

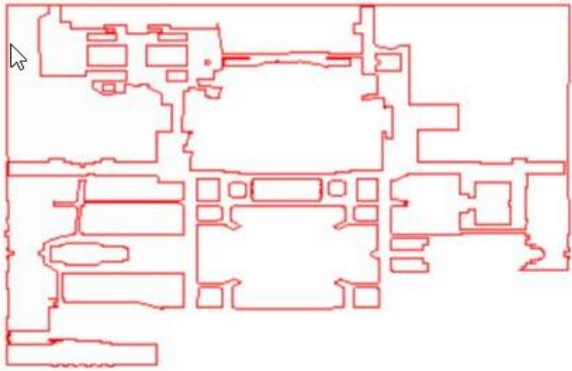
Visibility graph works effectively on polygonal workspaces, in this type of graph each obstacle corner is denoted as a node where all corners that are mutually visible are connected through an edge, this is where the method derives its name.

c. Voronoi Diagram:

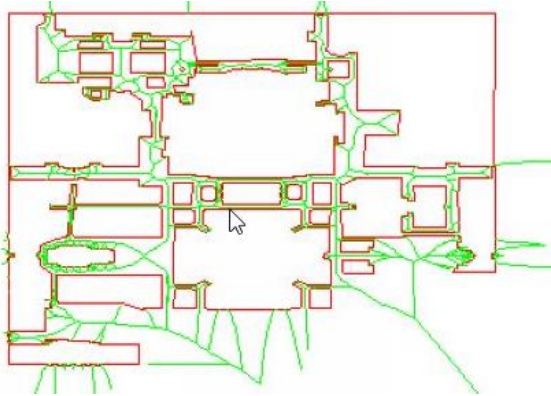
Georgy Voronoi first introduced the Voronoi diagram in 1908. Such a diagram demonstrates the distances between sets of points in a multi-dimensional space. In the area of path planning for mobile robots, a two dimensional space is used. The diagram is formed by connecting lines equidistant to the closest pair of points in the given set [19].

In a path-planning problem, a set of points represent the polygonal obstacles. This set is used to construct a Voronoi diagram, which is then snipped to remove edges colliding with obstacles or leading to dead-ends. The resulting graph is then searched to find the shortest path to the goal.

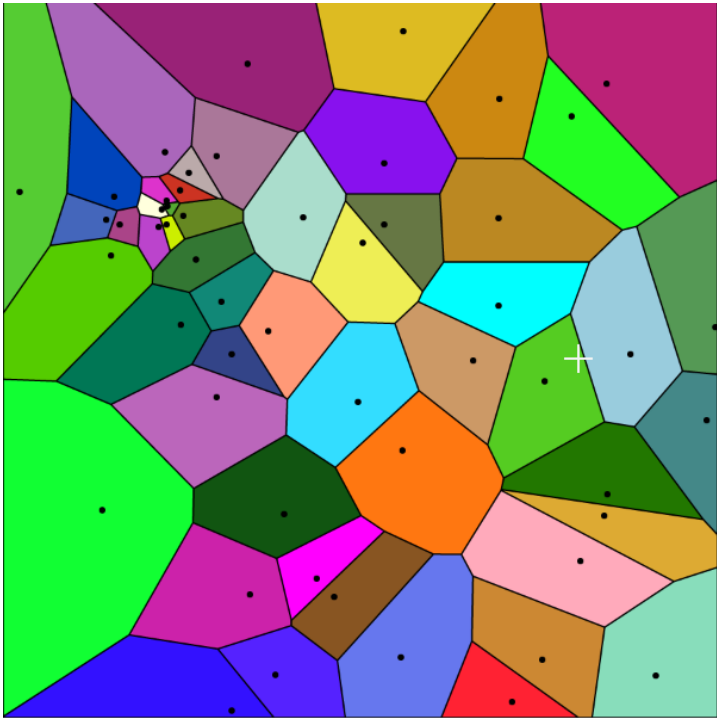
However, although it is a good representation the Voronoi diagram does not guarantee the Euclidian distance, which will reduce the optimality of graph search.



-a-



-b-



-c-

Fig.2.10 - (a) The two-dimensional map of a robot's test area. (b) The Voronoi diagram after the Elimination step, (c) general representation of Voronoi diagram.

Since our path planning need of heuristics to calculate the optimal path, we can only rely on the grid-based graphs, which will be demonstrated in the upcoming sections.

2.3.2. Graph Search:

After the construction of graph based map ,which in our case a grid based graph “occupancy grid”, we need to identify the type of graph search algorithm best suited for our map, to do this we need first to identify some of the famous algorithms and there uses:

2.3.2.1.Uninformed Search Algorithms:

This type of algorithms explore the graph with no sort of information about the goal ,so they are blindly exploring graphs or in a more sophisticated manner a brute force approach called the blind search, to find the desired state we expand different nodes and search through all the possible states that we are expanding in our configuration space. Then we start from our initial point and hope to at some point in time we end up at our goal configuration

The uninformed search algorithms are the simplest form of search algorithms, they are blindly searching through the configuration space they need to take into account how many steps they already have been taken from their start location until they reach a certain state and then they always try to make the decision of expanding the cheapest node in order to know to explore the space further until they find the goal configuration.

There are different ways how these costs can be taken into account for example if it is a search tree we expand it by looking to the depth of that tree we may have a cost function associated with every state in every individual transitions between the states, so we should take into account all the variants of this uninformed search techniques.

On contrast, the uninformed search technique does not exploit any additional information about the problem or additional knowledge about the goal or even an upper bounds or lower bounds that generate how cheap or how expensive a trajectory is.

The popular algorithms in this context are Breadth first search, Depth first search, and Uniform-cost search.

Chapter 2

Model Building and Path Planning Method

a. Breadth First Algorithm:

Here we are expanding our configuration space in a certain order not considering any cost so we are expanding the nodes and its neighbors row by row until we reach the wanted state.

This is kind of a tree where we start from our initial configuration expanding the neighboring states we stop once we find our goal configuration

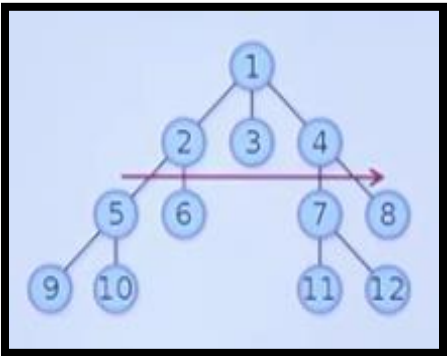


Fig.2.11 - Breadth First Algorithm example.

For example here our starting point is “1” and our goal is “12” using breadth first search we start from our initial configuration (1) then we first inspect all the neighbors so “2” ,”3” and “4” and then we inspect the next level then the neighbors of “2” which are “5” and “6”, “3” has no neighbors, neighbors of “4” is “7” and “8” we inspect the next level till we arrive at “12”which is our goal configuration

b. Depth First Algorithm:

Depth first search is somewhat different is breadth first search here we are going level by level checking for the first neighbor and then its first neighbor until there are no states left, then we back trace back and expand all the other states .

It is definitely not an optimal solution because we may find a path by just going into the depths to a goal configuration although there may be a shorter path just by going through a different neighbor.

Chapter 2

Model Building and Path Planning Method

It is a complete algorithm if we are in finite spaces so if there is just a limited number of states we can explore it otherwise if this is an infinite number of spaces we may run into a kind of an endless loop.

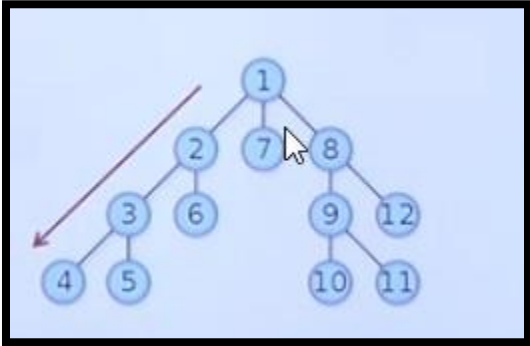


Fig.2.12 -Depth First Algorithm example.

b. Dijkstra/Uniform Cost Search (UCS):

We often do not have equal cost for all those expansions so some trajectories are more expensive and some trajectories/transitions are cheaper and this leads us into a so-called “cost sensitive search” which is still an uninformed search because it only uses this cost function and it doesn't take any further information into account about the goal.

In this example what we have is that all nodes are treated uniformly: In the same way and independent of their level but they are distinguished based on a cost function or the cost of transition from one node to another node.

Path finding algorithms is very important in real life; Dijkstra is a smart way for root planning in networks to get to our destination the quickest possible, it is used to find the shortest path between nodes in a graph [20].

In the following example we want to go from place “S” to place “E” passing through the necessary nodes, dodging all the obstacles represented by the numbers between the nodes but keeping in mind that the only path that matters is the shortest one.

Chapter 2

Model Building and Path Planning Method

To find the shortest path between “S” and “E” each time we pick the visited vertex with the lowest distance, we calculate the distance through it to each unvisited neighbor, and update the neighbor’s distance if smaller.

Dijkstra’s algorithm requires a priority queue at each of N iterations, where N is the number of network nodes. In order for the algorithm to run you must have a complete representation of the graph.

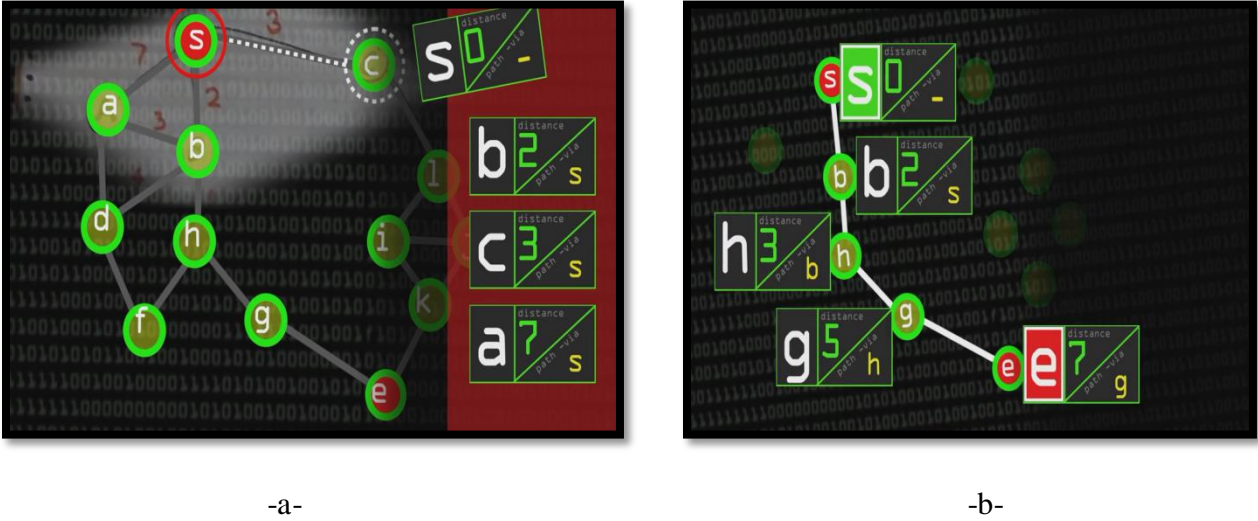


Fig.2.13 –examples of the Dijkstra algorithm (a) (b).

2.3.2.2. Informed Search Algorithms:

An informed search algorithm is designed to search the graph with a kind of hint about the goal destination; this hint is called a Heuristic, heuristic is an estimate of the cost to know how expensive is it in order to reach the goal configuration starting from the current configuration.

The heuristic provides a vision to exploit an estimate and still come up with a good solution or maybe even an optimal solution and that is what the informed search techniques are about, they are using this heuristic to speed up the search to find the goal configuration quickly but not all of these techniques are optimal.

The popular algorithms in this context are greedy best-first search, A*, D*.to define better this type of search algorithms we explore some examples:

a. A Star (A*) Algorithm:

A star is one of the best-informed search algorithms, with many uses even after many years of invention, the algorithm uses heuristics to search for the goal point this mean with the constant comparison of start-to-node coast it adds a goal-to-node distance related coast as described by the following equation.

$$f(n) = g(n) + h(n) \quad (2.10)$$

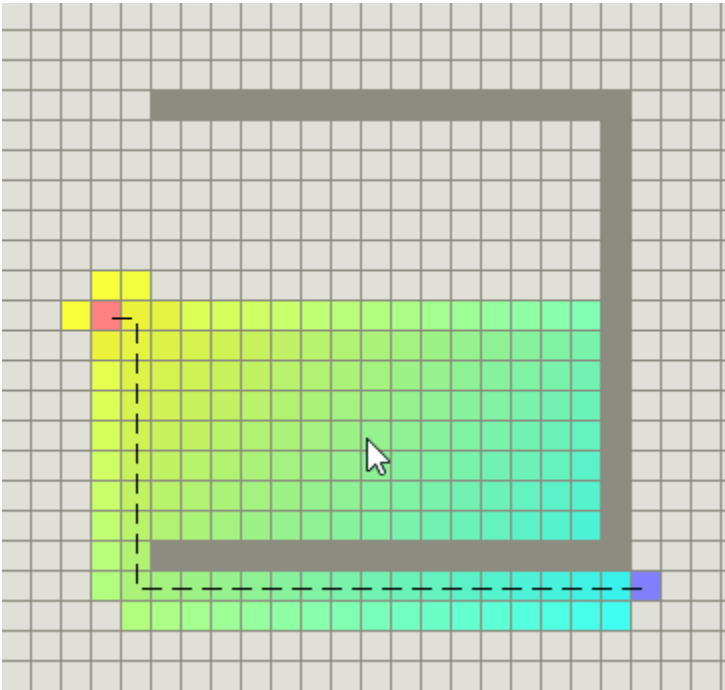


Fig.2.14- example of grid map searched using a A*.

b. Greedy Best-first Search Algorithm :

Best-first search is a search algorithm, which explores a graph by expanding the most promising node chosen according to a specified rule. This specific type of search is called greedy best-first search or pure heuristic search [21].

Greedy best-first search tries to expand the node that is closest to the goal, because the solution is likely led through this node. Thus, it evaluates nodes by using just the heuristic function; that is, $f(n) = h(n)$.

2.3.3. Criteria to compare algorithms:

The things that we are actually evaluating when we compare different search algorithms are generally four criteria:

- Completeness, optimality, time complexity and space complexity so what does that mean
- **Completeness** means that we have a valid sequence of configurations from the start to the goal location; in other words in case there is a path to reach our goal does our algorithm guarantee that will find it and this is
- **Optimality** in robot navigation problems, it is the algorithm, which generates the best possible solution under the assumption that has been made under certain cost functions.
- **Time complexity**: how long does it take to find a solution? Here we refer to the number of states or the number of transitions that the algorithm performs in order to find the path to final state.
- **Space complexity**: how much memory is needed to perform the search?

Because our search is based on a Grid Map with Static obstacles we, well seek the A* algorithm in designing our solution through multiple cases and environments.

2.4. A Star Search Algorithm:

2.4.1. Description

A* search is the most commonly known form of best-first search. It uses heuristic function $h(n)$, and cost to reach the node “n” from the start state $g(n)$. It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently.

A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses $g(n) + h(n)$ instead of $g(n)$. [22]

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence, we can combine both costs as following, and this sum is called as a **fitness number**.

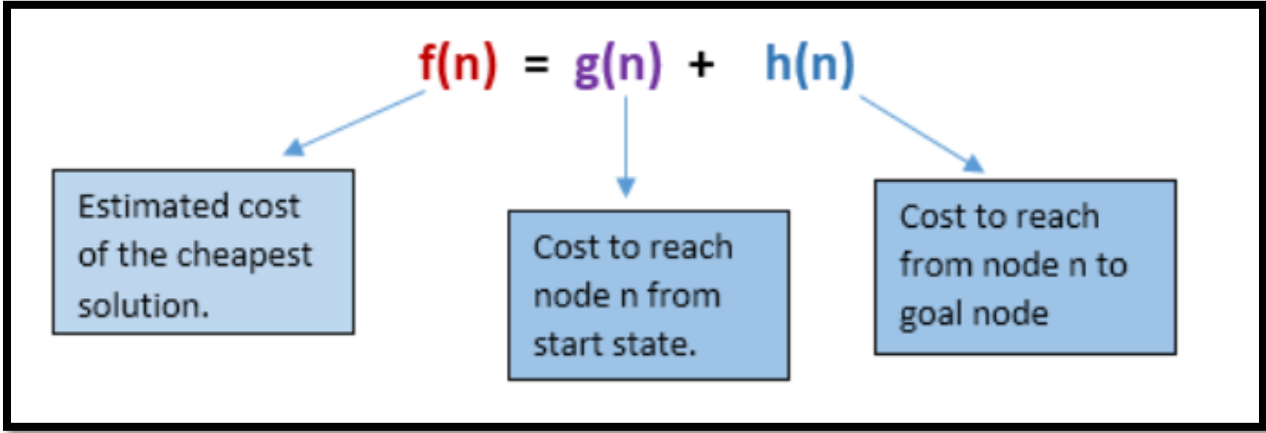


Fig.2.15- explanation of the A star (A*) cost function.

2.4.2. Use of Heuristics and Advantages:

Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by $h(n)$, and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

2.4.3. Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value $h(n)$. It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes that have already expanded and in the OPEN list, it places nodes, which have yet not been expanded. [23]

On each iteration, each node “n” with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues until a goal state is found

2.4.4. Algorithm of our A* search:

In our project, we used a classic algorithm describing the functionality of a typical A* star algorithm at work, our only variations are regarding the heuristics, which will be demonstrated, in the upcoming chapter.

- **Step 1:** Place the starting node in the OPEN list.
- **Step 2:** Check if the OPEN list is empty $OPEN \{ \} = \emptyset$, if the list is empty then return failure and stops.
- **Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function ($g + h$), if node n is goal node then return success and stop, otherwise
- **Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
- **Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer, which reflects the lowest $g(n')$ value.
- **Step 6:** Return to **Step 2**.

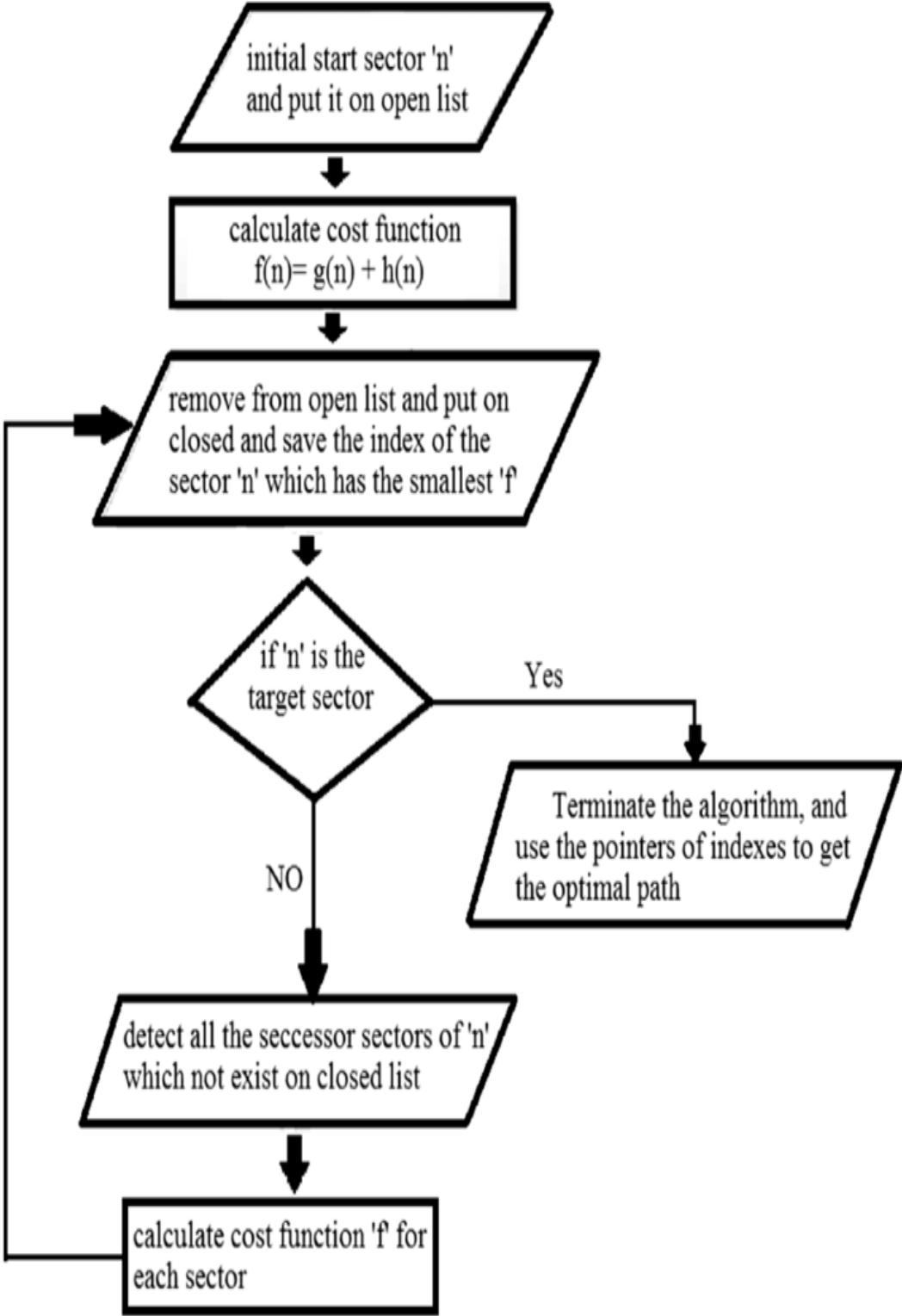


Fig .2.16- Graph Diagram incorporating the typical A* search Algorithm.

Algorithm: A* Path Search

Input : Start_position, End_position, Obstacles, size.

```
1  short_path ← empty
2  Cost ← equals 0
3  h(n) ← 0
4  g(n) ← 0
5  p ← list
6  size ← appointed by user in our case(25 or 50)
7  map() ← size, obstacles, Start_position, End_position
8  Open() ← open list receives map() only free space is permissible
9  Closed() ← Closed list has only the Start_position
10 If Open() = 0
11 |   Break .
12 Else
13   While shorth_path empty
14   |   N ← size(Open()) all possible search nodes
15   |   For i from 1 to N
16   |   |   (x,y)=Open.neighbor(i+1) append x and y of the neighboring nodes
17   |   |   h(n)=heuristic of selected node
18   |   |   g(n)=accumulated cost from start to node
19   |   |   Cost=g(n)+h(n)
20   |   |   If Coast (Open.neighbor(i+1)) < Coast (Open.neighbor(i))
21   |   |   |   Closed() ← Open.neighbor(i+1)
22   |   |   |   P(i) ← Open.neighbor(i+1)
23   |   |   |   If Open.neighbor(i+1)= End_position
24   |   |   |   |   Break,
25   |   |   |   |   Else continue,
26   |   |   |   Else discard
27   |   short_path ← P(i)
```

28 Return short_path

Output: Short path

2.4.5. Optimality

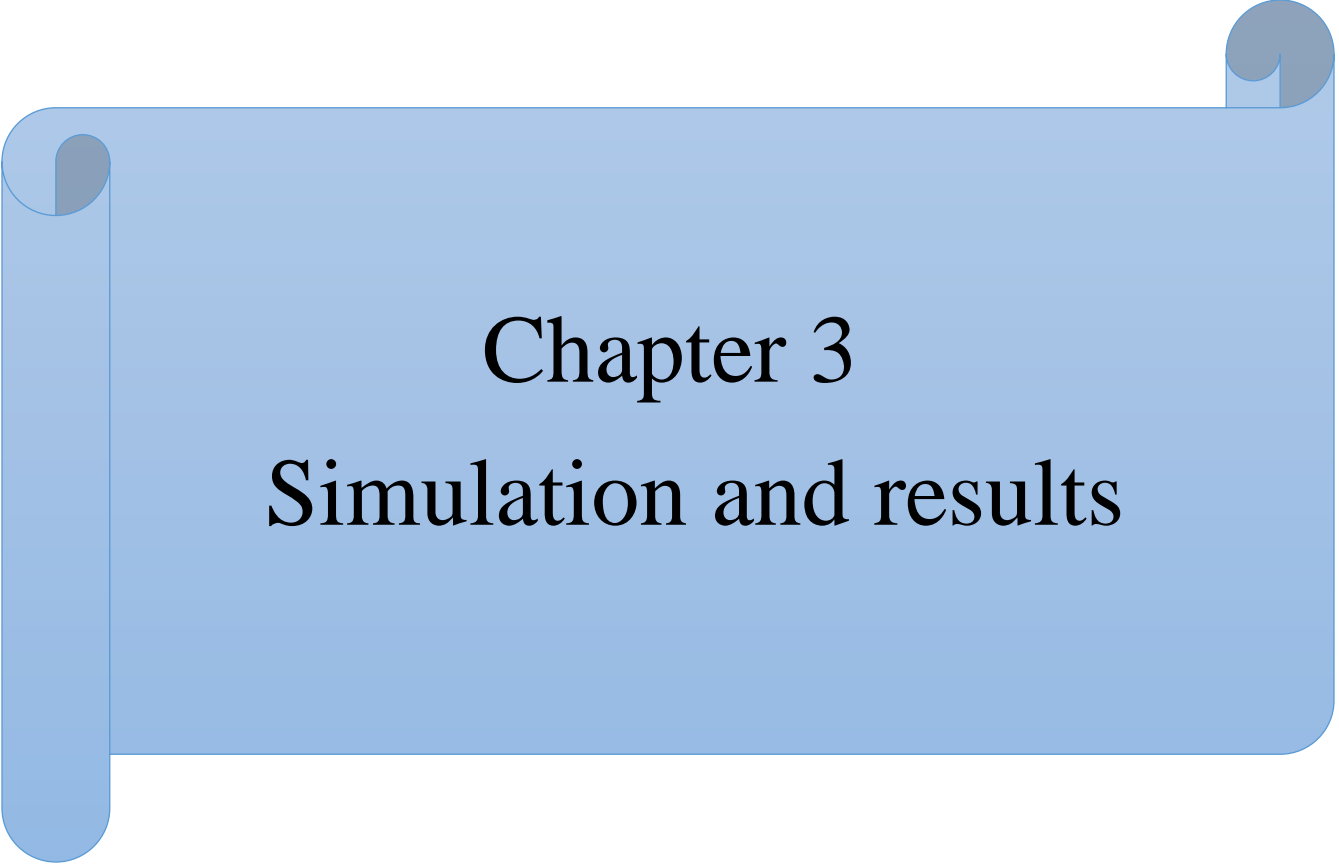
The algorithm described so far only gives us the length of the shortest path. To find the actual sequence of steps, the algorithm can be easily revised so that each node on the path follows its predecessor. After running this algorithm, the end node points to its predecessor, and so on, until the previous node is the start node.

For example, when finding the shortest route on a map, $h(x)$ might represent the straight-line distance to the target, since it is actually the smallest possible distance between two points. For a grid map, the use of the Manhattan distance becomes better depending on the set of movements available (the difference between these methods is discussed in chapter 3).

A* ensures that the optimal path is found without processing a node more than once. It is the better algorithm than other search algorithms. This algorithm can solve very complex problems

2.5. Conclusion:

In this chapter we represented the methodology was followed in choosing our type of work space and different types we choose from, in addition to choosing the A* algorithm as our base for finding the short path between a start and goal point, put in mind that there is a variety of heuristics bounding our results in finding the desired solution.



Chapter 3

Simulation and results

3.1 Introduction:

In this chapter using findings and algorithm described in the previous chapter, we test our algorithms in different simulations and different programming environments, in order to find a solution to shortest path problem; our work consists of testing different heuristics and trying to identify the best possible one to use.

Because we are using multiple environments, we will show different results and different scenarios, which will be discussed and analyzed.

3.2 Simulation Apparatus:

In our simulation, we will see different versions and scenarios so to start we have to talk about the hardware and software used.

3.2.1 Hardware:

All simulations were performed on a laptop with an Intel Core I3 3rd generation CPU with clock frequency of 2.4 GHz processor, an 8 GB DDR3 RAM of 665 MHz frequency and an Intel Graphics 4000 GPU of 648 MHz frequency.

3.2.2 Software:

In the simulation multiple software were used:

- For python language we used the Anaconda Navigator with Spider 5.0.5 IDE a powerful python development environment.
- For the second simulation, we used Matlab software.

3.3 Simulation:

3.3.1 Simulation I:

In this simulation, the start position is assigned an orange color where as for the destination position we assign it a blue color, all obstacles are in white in a work space randomly black which

mean after assigning obstacle cells free space is in black, green cells are an indication of open list exploration, red cells are already explored cells, after the algorithm finish searching the optimal path is then drawn using purple color as a final path.

The obstacles are considered inflated so the path is adjacent to each obstacle when drawing, which let us assign the robot as a cell, and each cell is connected using four edges.

We try in this simulation two algorithms A* and Dijkstra, the latter is used as reference example to the best next thing.

We have:

- Dijkstra has no heuristic only a cost function from the start node to the explored nodes, it can be considered as an A* if we put $h(n) = 0$. $F(n) = g(n)$.
- A* has a heuristic and can be represented as $f(n) = g(n) + h(n)$, we test tree heuristics:
- $h_1 = x + y$ Which is the Manhattan distance from each node to the goal position, considered as the real distance.
- $h_2 = \sqrt{x^2 + y^2}$ Which is the Euclidian distance from each node to the goal position, considered in some cases an under estimate of the real heuristic $h_2 \leq h_1$
- $h_3 = x^2 + y^2$ Which is the square of the Euclidian distance and considered an over estimation of the real distance $h_3 > h_1$.
- Where x and y are distance between two points (x1, y1) and (x2, y2) is given by:

$|x1 - x2| + |y1 - y2|$ such that (x2, y2) are coordinates of each explored cell and (x1, y1) coordinates of the goal point.

a.First Scenario:

In this scenario, we take the grid size to be equal to 25, which is the number of cells on the x and y-axis, denoted as $S(x, y) = [25, 25]$, we consider this as a medium sized area.

We try the two algorithms with all A* heuristics:

Case 1: maze type obstacles map

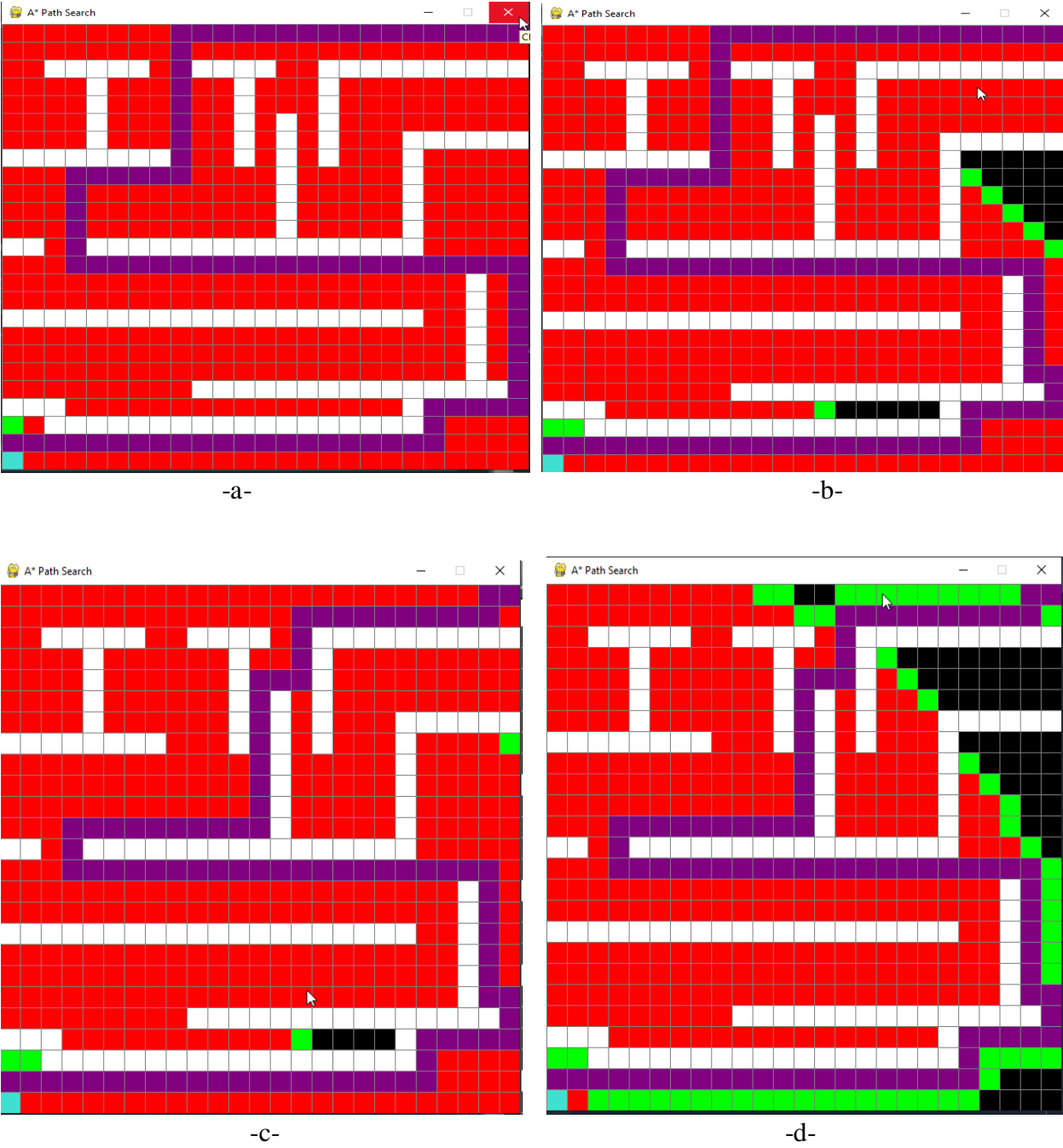


Fig.3.1- simulation in 25*25 grid maze structured obstacles ,(a) Dijkstra algorithm’s path planning result,(b)A* Algorithm with h_1 heuristic ,(c) A* Algorithm with h_2 heuristic ,(d) A* Algorithm with h_3 heuristic.

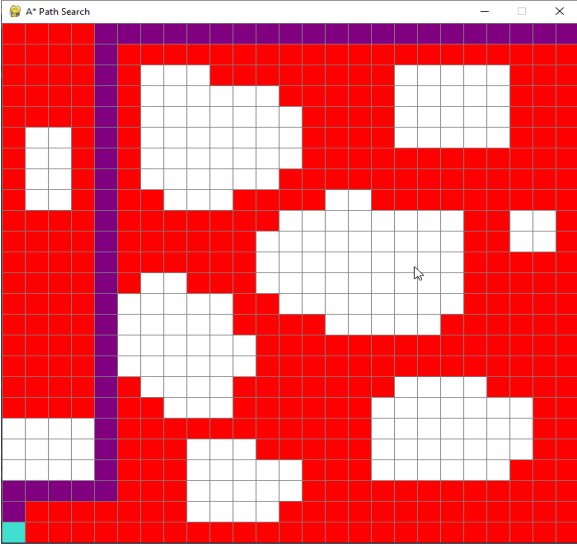
The simulation start with a start position in the top right corner constantly searching for the goal position in the left bottom corner ,the algorithms constantly search a maze demonstrated in **Fig.3.1** which is quite tricking and hard ,each algorithm resulting path is almost identical to the other, but for the resulting paths different results are shown where Dijkstra and (A^*,h_1) have the same path traveled ,while (A^*,h_2) and (A^*,h_3) have the same path this difference is not considered according to **Table 3.1** ,although in the next simulation some differences will appear in time of search ,explored nodes and path length as shown in **Table 3.1** bellow:

Table 3. 1 Time, explored nodes and path length of each algorithm in a maze like map of 25*25 grid.

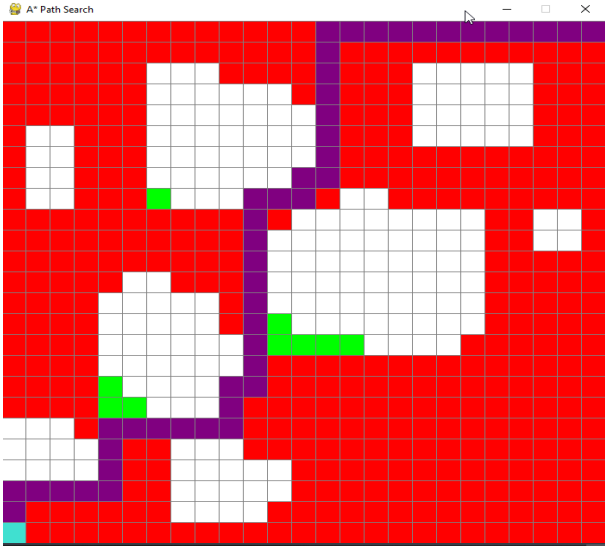
Algorithms	Time (s)	Explored nodes	Path length (nodes)
Dijkstra	5.96343	484	90
A^*,h_1	5.12706	464	90
A^*,h_2	5.40627	480	90
A^*,h_3	5.43103	535	97

We clearly see from the data that all algorithms got exactly the same length of path except for the h_3 which is an over estimate ; although time differ from one to another but still it's remotely close between them the shortest search time being 5.12706 seconds and the longest 5.96343 seconds, however node exploration is somewhat different and this is due to the heuristics at work, these results are considered for a 25*25 grid which is almost a medium sized grid.

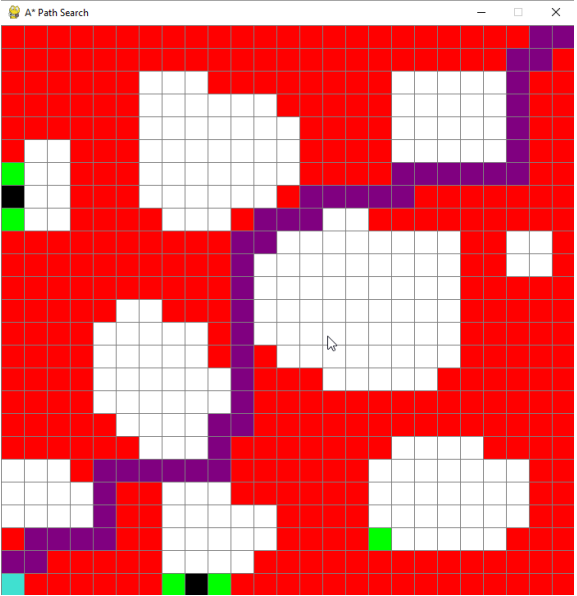
Case 2: random obstacles map:



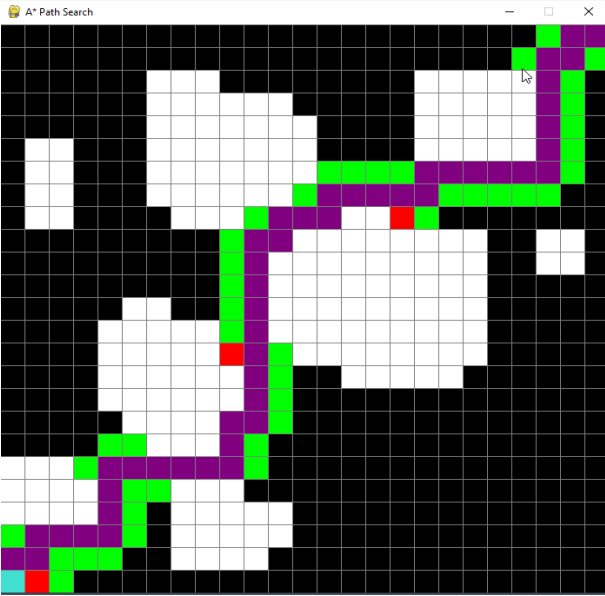
-a-



-b-



-c-



-d-

Fig.3.2- simulation in 25*25 grid randomly structured obstacles ,(a) Dijkstra algorithm's path planning result,(b)A* Algorithm with h_1 heuristic ,(c) A* Algorithm with h_2 heuristic ,(d) A* Algorithm with h_3 heuristic.

The simulation as the previous one is a search from the start position at the top right corner, to the bottom left corner searching through the corresponding nodes ,according to each algorithm setting ,this case is also a 25*25 grid but with randomized obstacles, from **Fig .3.2** we can clearly notice a difference in the way the graph was searched to path plan ,as in **Fig.3.2 a,b,c** the grid was quite searched in reason of finding a solution ,but in **d** the algorithm went straight to the target with no exploration of neighboring nodes.

Table 3. 2 Time, explored nodes and path length of each algorithm in a random set obstacles map of 25*25 grid.

Algorithms	Time (s)	Explored nodes	Path length (nodes)
Dijkstra	6.00000	412	49
A*, h_1	5.53570	436	49
A*, h_2	5.76106	403	49
A*, h_3	0.82848	94	50

Clearly in a 25*25 random obstacles map a (A*, h_3) algorithm is faster than the rest with less explored nodes, but when considering the optimality of the path (A*, h_2), (A*, h_1) and Dijkstra are better than $P_0=49$, which signifies the optima path in this setting.

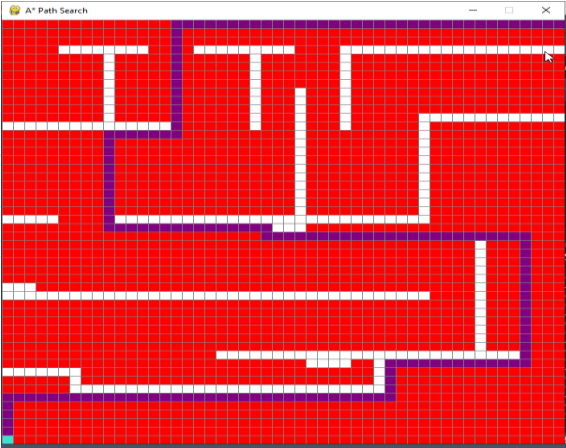
b. Second Scenario:

In this scenario, we take the grid size to be equal to 50, which is the number of cells on the x and y-axis, denoted as $S(x, y) = [50, 50]$, we consider this as a large sized area.

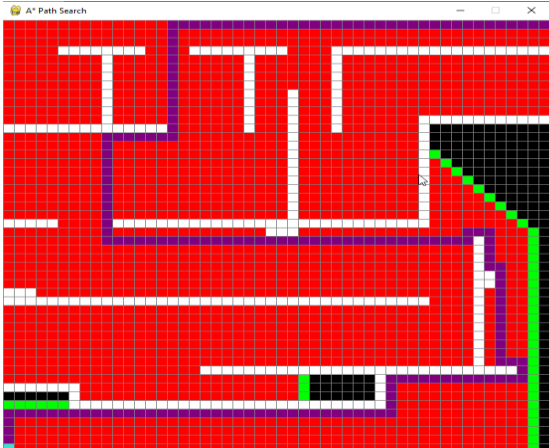
We try the two algorithms with all A* heuristics:

Case 1: maze type obstacles Map.

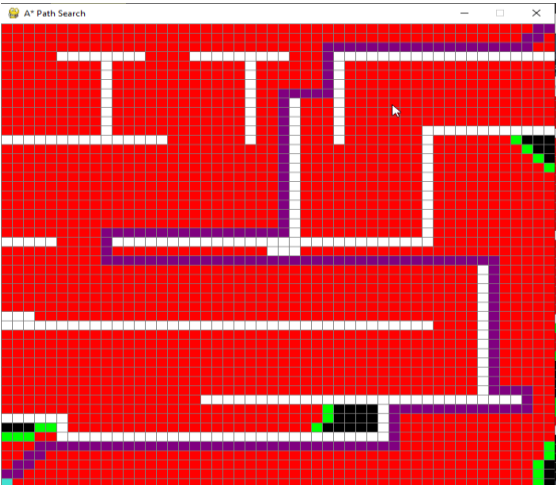
The simulation here is the same as the simulation in scenario I case1, with the difference in grid size equal to 50*50, we can in the next figure see the results:



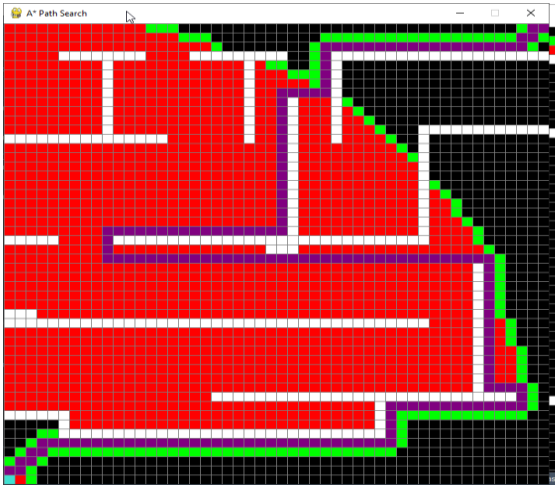
-a-



-b-



-c-



-d-

Fig.3.3- simulation in 50*50 grid maze structured obstacles ,(a) Dijkstra algorithm’s path planning result,(b)A* Algorithm with h_1 heuristic ,(c) A* Algorithm with h_2 heuristic ,(d) A* Algorithm with h_3 heuristic.

The simulation in the an augmented plane dons not differ much as we can see in **Fig.3.3**.meaning in regard to the resulting path but obviously regarding time, explored nodes we can expect to have a variety of results.

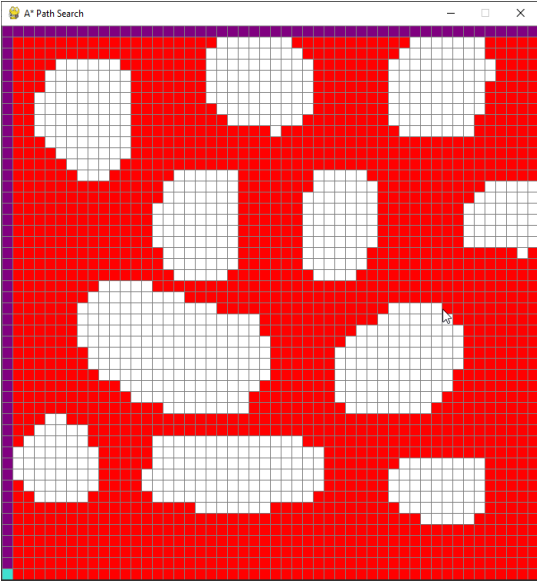
Table 3. 3: Time, explored nodes and path length of each algorithm in a maze like map of 50*50 grid.

Algorithms	Time (s)	Explored nodes	Path length (nodes)
Dijkstra	103.30925	2222	172
A*, h_1	91.94993	2081	172
A*, h_2	97.67163	2196	172
A*, h_3	204.64825	4595	176

Algorithms Dijkstra, (A*, h_1),(A*, h_2) preformed good regarding time referred to (A*, h_3) which had a very long time spent searching for the goal position, although most algorithms had P_o between 172 ,which is considered close ,with (A*, h_1) as the optimal solution regarding time node explored and resulting path length.

Case 2: random obstacles map.

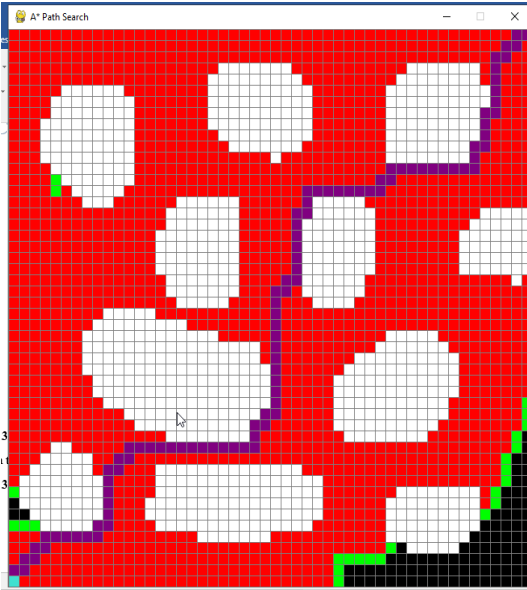
The same as simulation I a.2 case 2 this simulation only increased the grid size to 50*50 and kept the random property of the obstacles, as we can see in **Fig.3.4**.



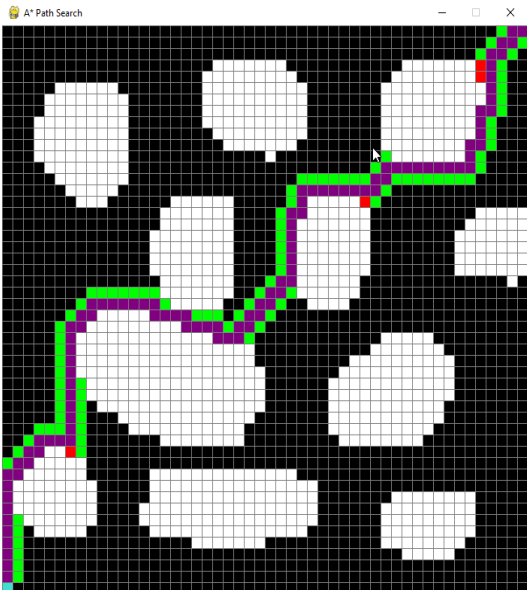
-a-



-b-



-c-



-d-

Fig.3.4- simulation in 50*50 grid maze structured obstacles ,(a) Dijkstra algorithm’s path planning result,(b)A* Algorithm with h_1 heuristic ,(c) A* Algorithm with h_2 heuristic ,(d) A* Algorithm with h_3 heuristic.

As seen before some algorithms work best in this type of environment specially (A^*,h_3) , where we can see the difference from the rest of algorithms, concerning explored nodes, (A^*,h_1) is best where (A^*,h_2) as second, Dijkstra is using brute force to explore the graph.

Table 3. 4: Time, explored nodes and path length of each algorithm in a random set obstacles map of 50*50 grid.

Algorithms	Time (s)	Explored nodes	Path length (nodes)
Dijkstra	73.28393	1638	96
A^*,h_1	51.79499	1230	96
A^*,h_2	72.88451	1547	96
A^*,h_3	4.641179	201	102

As described in Table we can see that by increasing the grid size algorithms start to differ in a random obstacle area ,as we can notice (A^*,h_3) has the optimum time and number of nodes explored ,while the path is not optimal ,considering (A^*,h_1) (A^*,h_2) and Dijkstra has $P_O = 96$ as the best path planed but with more time .

3.3.2. Cross examination of Simulation I and Discussion:

We are going to represent all cases in a clustered bar chart, which will allow as comparing algorithms and deducing the best possible algorithm in searching for our optimal path.

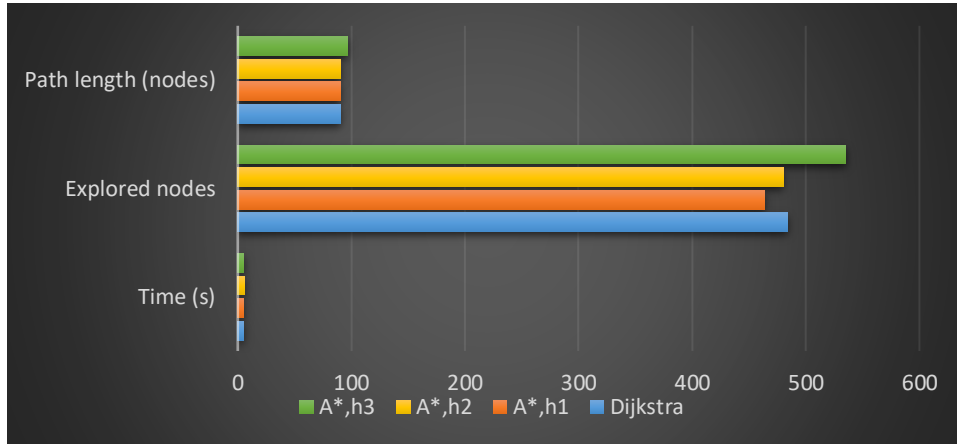


Fig.3.5.-clustered bar chart of Scenario I case 1 results.

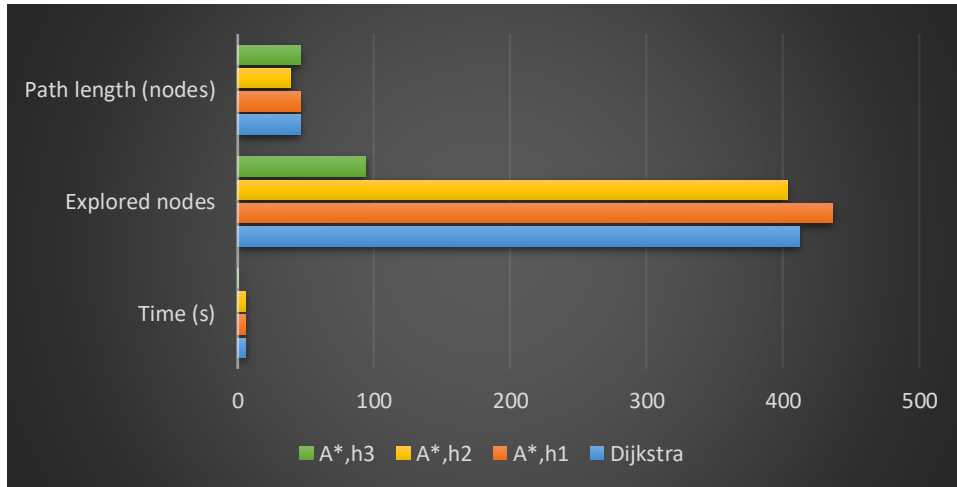


Fig.3.6.-clustered bar chart of Scenario I case 2 results.

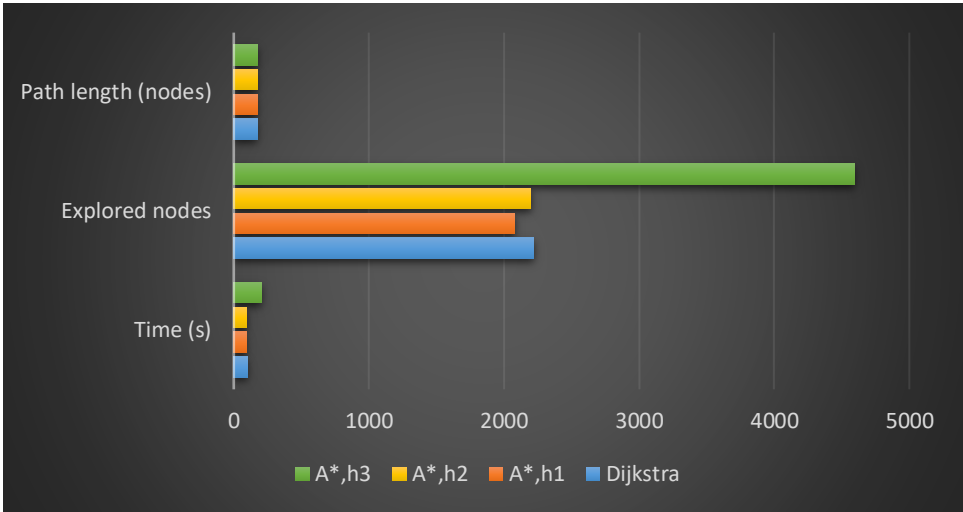


Fig.3.7-clustered bar chart of Scenario II case 1 results.

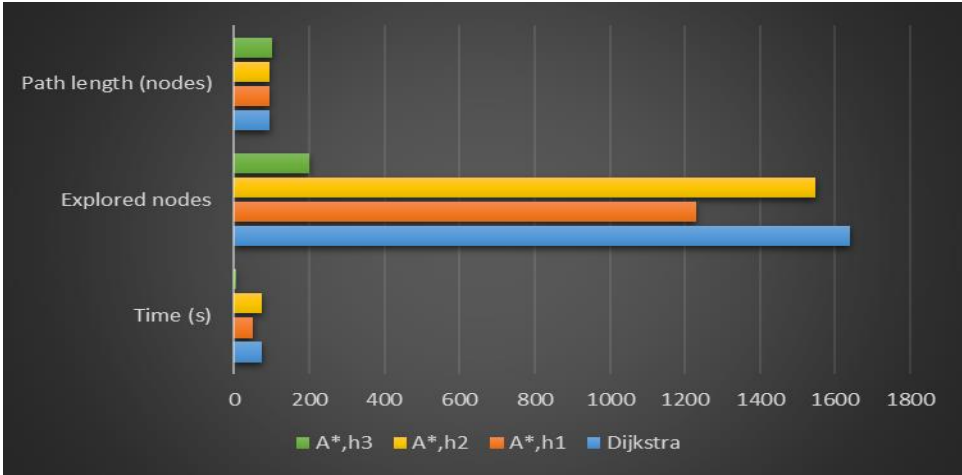


Fig.3.8-clustered bar chart of Scenario II case 2 results.

- From the four charts, we can see that concerning the time in both grids and only in random obstacle environments we can see that (A*,h₃) is the best suitable algorithm as in both scenarios it has a 46.6% less than the average of time in the first scenario, and 99.1% less in the second scenario, while (A*,h₃) does not increase much regarding time, where (A*,h₂) and Dijkstra have improved in larger grids but not much further from average but (A*,h₁) remains the fastest algorithm with an optimal resulting path in all cases.

- Although in maze based map (A^*,h_3) is the worst in regards to time if we increase the grid size, but the rest of the algorithms increase slightly in proportion with the size.
- When we consider explored nodes with Dijkstra being a brute force algorithm in reference to A^* , it has the highest number of explored nodes in most cases, where (A^*,h_3) has a very low number in random obstacle map environments and a high number in maze based map, where (A^*,h_1) and (A^*,h_2) kept fairly an average number in each case.
- Regarding path optimality (A^*,h_2) , (A^*,h_1) and Dijkstra are the best in 3 cases which make them the best algorithms concerning short path problem, (A^*,h_3) although fast has from medium to bad path optimization, especially if we increase our grid size.

From these notes, we can say our best choices is:

(A^*,h_1) if we wish for an optimal path in all scenarios and cases with a slight time increase and average number of explored nodes, this can be the result of underestimation or equal estimate which always produces the best path plan with slight compromise in nodes explored and time spanned.

Where:

(A^*,h_3) is only good if we are considering large size grids and we wish for short time and less explored nodes, but with a the worst results in regard to our planed path length, this is due to overestimation of the heuristic which leads to no optimal paths and high number of nodes explored in some cases.

(A^*,h_2) and Dijkstra have average results in small sized grids, but second best in all areas in regard to time and explored nodes, this is due to Dijkstra lack of informed search and (A^*,h_1) average heuristic regarding path optimality.

3.3.3. Simulation II:

In this simulation, we use the Matlab software where we have chosen (A^*, h_1) as the optimal algorithm for path planning, we consider here an eight edged nodes for better optimization and ,access to lateral movement ,the grid size is by choice so we are going to use two preset sizes 50*50 and 25 *25 .

We consider the target position in green color and the robot start position in blue, both are small circles this is because the obstacles are inflated green rectangle in darker blue for ease of simulation, and we can see the detailed path at the end of the simulation .

All objects are randomly put and can have any place on the grid.

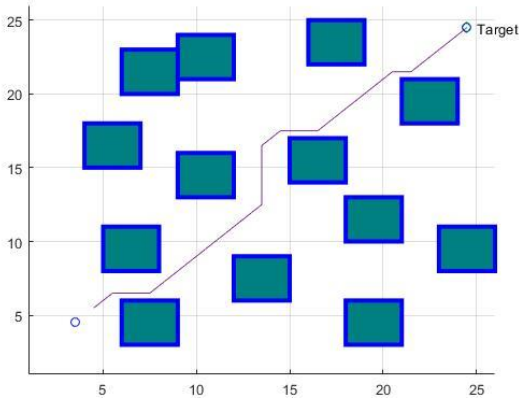


Fig.3.9- Matlab simulation of (A^*, h_2) in 25*25 grid.

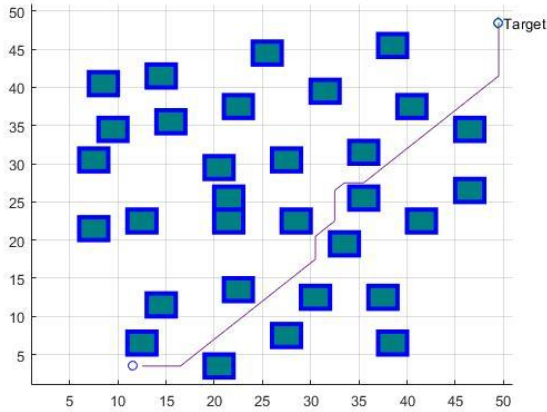


Fig.3.10. Matlab simulation of (A^*, h_2) in a 50*50 grid.

In both simulation we clocked the time needed for path generation ten times each trying the same exact simulation with scattered obstacles and same begging and start, we had an averaged running time in each size of the grid to come up with an average time.

Table 3. 5: Average running time of 10 repetitions of each grid size.

Grid size 25*25 time (sec)	Grid size 50*50 time (sec)
21.215	37.668
20.945	36.132
20.005	36.200
19.785	37.974
22.178	38.038
23.345	35.121
21.111	35.845
21.450	34.221
18.963	38.003
20.432	37.020
22.374	37.112

The average of the 25*25-grid simulation time is: 21.073 seconds.

The average of the 50*50-grid simulation time is: 36.667 seconds.

We can notice from these results that our average time in these simulations with grid sizes is highly better than that of the ones in our python environment this is due to the string conversion in python which render compiling slightly higher in time size, in addition to the simple pre built “Distance” function which minimizes the time .

After testing the Manhattan distance heuristic in different settings and environments we have concluded that for a highly optimal path planning with good to better-explored nodes and less time spanned we can always refer to this algorithm slight change from the Euclidian based one to determine the best path for a robot.

3.4 Conclusion:

Testing the A* star algorithm with different Heuristics with a dummy reference as the Dijkstra algorithm in different setting and environments for (Time, explored nodes, path length), has shown how all algorithms are in servitude of some kind of drawback, but to choose a good algorithm a compromise need to be made for the best results in all areas of our tests, time, explored nodes and optimal path.



General Conclusion

General Conclusion

Notes to conclude:

In this project, we worked to find a solution to the path planning problem in a grid-based map or occupancy grid through a cross-examination and elimination process, the algorithms depicted can successfully solve the path planning problem of any robot model, but in deferring results, we had to choose the better suited for balanced results close to total optimality in different scenarios and cases, to conduct the project we assume:

- The obstacles are static and not dynamic.
- The goal, start, and obstacles are users put anywhere on any map.
- Obstacles can be of any size and shape.
- The maps have maze type and random type obstacles to cover all scenarios.

The goal is to plan the path through a map in different sizes and configured obstacles to have a better understanding of the many algorithms, our algorithm solves the path planning problem by using the A* star algorithm with a variety of heuristics.

Our work mainly is to choose the best heuristic formula, which will be the base for the A* optimal search algorithm, this is after trying the proposed heuristics in a maze than a random based obstacles map then analyzing the different data of “time, explored nodes, path length” to choose the heuristic that explores the path with the highest efficiency in all or majority of scenario’s cases, concerning our binding data as criteria to judge, all this is performed in the python environment for its ease of list creation and the dictionaries properties of choosing and sorting.

After finally pairing the best heuristic with our A* algorithm we start our Matlab simulation where we try a random obstacle map using the resulting algorithm, in this setting is faster than the previous setting since Matlab does not have a string conversion drawback and all tests are internally shown.

The obtained results have served the set objective, this simulation confirms that graph search algorithm A* is a capable algorithm in path planning problems if the heuristic is carefully chosen confirming our proposition of being that A * is as good as its heuristic, our algorithm

showed that its path planning optimality in accordance to time spanned, nodes explored and path size is related to grid size and obstacle setting chosen.

Future work:

Since the robotics arena is diverse and interdisciplinary, while working on the project you find and formulate all sorts of possible problematics, which exert the necessity of further improvements in the future:

Concerning the environment:

Our work can adhere to the dynamics of the environment where we can introduce dynamic obstacles, which will inject necessary properties in designing the algorithm, this will need a portion search algorithm with the same properties of A* and the concerned heuristic.

Regarding search heuristic:

This can be achieved by breaking down a single heuristic function into simpler parts, which can be optimized using dynamic programming. In addition, applying machine learning models to the heuristic part such as gradient descent can be promising.



Appendix

Appendix

Manhattan distance

A taxicab geometry is a form of [geometry](#) in which the usual distance function or [metric](#) of [Euclidean geometry](#) is replaced by a new metric in which the [distance](#) between two points is the sum of the [absolute differences](#) of their [Cartesian coordinates](#). The taxicab metric is also known as [snake](#) distance, city block distance, Manhattan distance, with corresponding variations in the name of the geometry. The latter names allude to the [grid layout of most streets](#) on the island of [Manhattan](#), which causes the shortest path a car could take between two intersections in the [borough](#) to have length equal to the intersections' distance in taxicab geometry.

Krause, Eugene F. (1987). [Taxicab Geometry](#)

The taxicab distance, DI , between two vectors P and Q in an n -dimensional real vector space with fixed Cartesian coordinate system, is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. More formally,

$$DI(P,Q)=\|P-Q\|=\sum_{i=1}^n |P_i - Q_i| \text{ where } P \text{ and } Q \text{ are vectors}$$

For example, in the plane, the taxicab distance between :

$$(P_1, P_2) \text{ and } (Q_1, Q_2) \text{ is } |P_1 - Q_1| + |P_2 - Q_2|$$

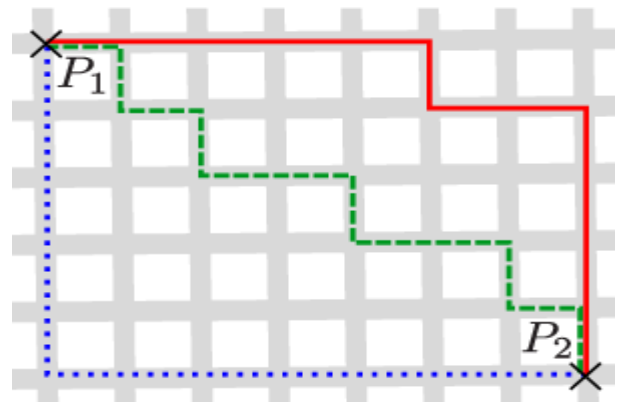


Fig A1: Taxicab geometry distances. All three pictured lines have the same length for the route between P 1 and P 2 [23] .

Pygame 2.0.1 library:

programming language library for making multimedia applications like games pygame (the library) is a Free and Open Source python

Pygame is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language.

Pygame is highly portable and runs on nearly every platform and operating system.

Pygame itself has been downloaded millions of times.

Pygame is free. Released under the LGPL licence, you can create open source, freeware, shareware, and commercial games with it. See the licence for full details.

For a nice introduction to pygame, examine the line-by-line chimp tutorial, and the introduction for python programmers. buffer, and many other different backends... including an ASCII art backend! OpenGL is often broken on linux systems, and also on windows systems - which is why professional games use multiple backends.[24]



Fig A2.Pygame symbol.

Bibliography

- [1] cs.stanford.edu, Origins of "robot" and "robotics", Robotics: A Brief History Stanford Computer Science. [Online]. Available: <https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/history.html>.
- [2] S K Saha, Introduction to Robotics 2014. MC GRAW HILL INDIA. P. 2.
- [3] Bruno Siciliano, Oussama Khatib, Springer Handbook of Robotics .Springer International Publishing 2016. P. 12.
- [4] Bruno Siciliano, Oussama Khatib, Springer Handbook of Robotics. Springer International Publishing, 2016. P.37.
- [5] mars.nasa.gov, Nasa Science, Mars 2020 Mission Perseverance Rover. [Online]. Available: <https://mars.nasa.gov/mars2020/>.
- [6] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots. 2nd ed. P. 101.
- [7] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots. 2nd ed. P. 101.
- [8] Frank Dellaert, Dieter Fox, Wolfram Burgard, Sebastian Thrun, Monte Carlo Localization for Mobile Robots, IEEE International Conference on Robotics & Automation. Detroit, Michigan. May 1999. p.1
- [9] MOHINDER S. GREWAL, ANGUS P. ANDREWS, KALMAN FILTERING, Theory and Practice Using MATLAB. Third Edition. John Wiley & Sons publications. 2008. p.1.
- [10] D. Fox, W. Bureard, - and S. Thrun. Markov localization for mobile robots in dynamic environments. Journal of Artificial Intelligence Research, 11, 1999. p.392
- [11] MOHINDER S. GREWAL, ANGUS P. ANDREWS, KALMAN FILTERING, Theory and Practice Using MATLAB. Third Edition. John Wiley & Sons publications. 2008. p.5.
- [12] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots. 2nd ed. P. 373.
- [13] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots. 2nd ed.
- [14] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, Renato Vidoni, Path Planning and Trajectory Planning Algorithms: a General Overview. ResearchGate. 2015.

- [15] Bruno Siciliano, Oussama Khatib, Springer Handbook of Robotics. Springer International Publishing, 2016. P.11.
- [16] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots. 2nd ed. P. 37.
- [17] N.Leena, K.K.Saju, Modelling and trajectory tracking of wheeled mobile robots, International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST - 2015). P. 540
- [18] Advanced Robotics #5: Sampling-based Path Planning, Correll Lab, CU Computer Science, <http://correll.cs.colorado.edu/?p=2012> . [Online].
- [19] Robert Hult, Reza Sadeghi Tabar, Path Planning for Highly Automated Vehicles. Master's Thesis in Systems, Control and Mechatronics. P. 103.
- [20] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". Introduction to Algorithms (Second ed.).
- [21] What is greedy best first search in artificial intelligence?, Everythingwhat , <https://everythingwhat.com/what-is-greedy-best-first-search-in-artificial-intelligence>.
- [22] Delling, D.; Sanders, P.; Schultes, D.; Wagner, D. (2009). "Engineering Route Planning Algorithms". Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation
- [22] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths,"
- [23] Secure Geographic Routing Protocol for DTN November 2015, Computer Communications. Author: [Adrián Sánchez-Carmona](#)
- [24] About Pygame , <https://www.pygame.org/wiki/about>.