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

University M'Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering
Department of Power and Control

Final Year Project Report Presented in Partial Fulfilment of

the Requirements of the Degree of

'MASTER'

In Electrical and Electronic Engineering

Title:

Mobile robot LIDAR Based Path Planning and Tracking

Presented By:

- **BENDREF Mansour Djassem**
- **SAHI Yasser**

Supervisor:

Dr . BELAIDI Hadjira

Registration Number:...../2021

Dedication

It is with genuine gratitude and warm regards that this work is dedicated to my beloved brother who have been and always will be there Inchalah to encourage and support me. And great gratitude and love for our parents and sisters for their love and support.

This work is also dedicated to our beloved grandmother Meriam. And all our closest friends for their continuous encouragement; thank you.

Acknowledgement

At first, we would humbly thank Allah the Almighty for guiding us and giving us health, will and knowledge that helped us to accomplish this work in such a hard time.

Then, we would like to express our great gratitude to our supervisor: ***Dr.BELAIDI Hadjira*** *for her support and guidance and for always insisting to conduct our own work, and being there whenever we had that confusion or a question. We couldn't have done it without her.*

We also wish to thank the President of jury and jury members for accepting to take part of the examining committee. We are thankful that in the midst of all their activities, they dedicated time and effort to read and evaluate our project.

Abstract

This project deals with mobile robot path planning and tracking. The mobile robot is equipped with a Lidar sensor which scans the environment and returns cloud points constituting the obstacles coordinates. First, our work consists in building the robot environment using binary occupancy grid strategy. Then, an algorithm which allows the robot to reach its target starting from any initial coordinate is created (RAY based planner). Second a graph based strategy is used to extract the free optimal path, then the obtained path is smoothed using Cubic splines. Last, pure pursuit controller is implemented to allow the robot to track the reference optimal path and find a new path in case the robot encounters a sudden static obstacle that was not present before in the ma . The work is validated by simulation results using MATLAB and Simulink. Moreover, the obtained results are compared with previous work in the literature (PRM and RRT-star) to prove their efficiency, and the obtained results are discussed.

ملخص

يتعامل هذا المشروع مع تخطيط مسار الروبوت المتحرك وتتبعه. تم تجهيز الروبوت المتحرك بجهاز استشعار Lidar الذي يقوم بمسح البيئة وإرجاع سحابة نقاط التي تشكل إحداثيات العوائق. أولاً ، يتمثل عملنا في بناء بيئة الروبوت باستخدام استراتيجية Binary occupancy grid . بعد ذلك، يتم إنشاء خوارزمية تسمح للروبوت بالوصول إلى هدفه بدءًا من أي إحداثيات أولية (Ray based planner)، ثانيا تم استخدام إستراتيجية تعتمد على نظرية المخططات (Graph) لاستخراج المسار الأمثل ، ثم يتم تنعيم المسار الذي تم الحصول عليه باستخدام (Cubic splines). في الأخير يتم استعمال متحكم (pure pursuit) لتتبع المسار للسماح للروبوت المتحرك بتتبع المسار المرجعي الأمثل المخطط سابقا و انشاء مسار جديد للوصول الى الهدف في حالة وجود عائق جامد على السار المخطط مسبقا. تم التحقق من صحة العمل من خلال نتائج المحاكاة باستخدام MATLAB و Simulink ؛ علاوة على ذلك ، تتم مقارنة النتائج التي تم الحصول عليها مع الأعمال السابقة (PRM and RRT planners) لإثبات كفاءتها، كما يتم مناقشة النتائج التي تم الحصول عليها .*

Table of contents

List of figures

List of tables

List of abbreviations

AAV	Autonomous Aerial Vehicle
AGV	Autonomous Ground Vehicle
AI	Artificial Intelligence
AS	Autonomy Spectrum
AUV	Autonomous Underwater Vehicle
EM	Electromagnetic
EO	Electro-Optical
GA	Genetic Algorithm
ICR	Instantaneous Center of Rotation
LiDAR	Light Detection And Ranging
PF	Potential Field
PRM	Probabilistic Road-Maps
ROV	Remotely-Operated Vehicle
RPA	Remotely-Piloted Aircraft
RRT	Rapidly exploring Random Tree
UAS	Unmanned Aircraft System

General Introduction

Nowadays, the adoption and demand on autonomous mobile robot is growing from industrial robots, hospitals, autonomous transportation systems and tremendous application which are adopted in daily life. So the problem of autonomy and decision making and planning for robot has been an area of interest for research for many decades [37] [38] [39], then path planning is considered as an important task for autonomous robots along-side with environment modelling and motion planning. This report deals with creating a simple environment modelling strategy based on binary occupancy grid and proposing a new path planning strategy (RAY based planner) to solve the problem of global path planning and optimal path extraction for mobile robots, in addition to the implementation of motion controller for differential robot to track the panned path. These strategies are simulated and validated using MATLAB and Simulink.

The rest of the report is organized as follow. Chapter 1 introduces mobile robots and the problem of path planning and environment modelling; then, it states path planning problem and some relevant works. Chapter 2 examines the binary occupancy grid modelling strategy and proposes path planning approach and optimal path extraction, last path smoothing is applied. In Chapter 3, the kinematics of differential drive robot are examined along-side with the pure pursuit controller. Chapter 4 is dedicated for discussing the obtained results in different environment complexities and results comparison for validation. The report ends up by conclusion and future work proposition.

Chapter I

Generalities and state-of-the-art

Robot navigation and path planning has been a very active research topic in the last decade. This report is one of the contributions to this field. In this chapter, autonomous robots are discussed, and the need for proper path planning along-side with a convenient environment modeling techniques are stated, and lidar system as a range sensing device is presented. In addition, different techniques related to robot path planning are discussed along with some related works to the subject.

1.1 Generalities on mobile robots

1.1.1 Definition and history of robots

Robots! Robots on Mars and in seas, in emergency clinics and homes, in processing plants and schools; robots battling fires, making products and items, saving time and lives … Robots today have an extensive effect on numerous parts of present day life, from modern assembling to medical services, transportation, and investigation of the profound space and ocean. Tomorrow, robots will be as unavoidable furthermore, individual as the present Smartphone [1].

The robot “concept” was established by those many creative historical realizations. Nonetheless, the emergence of the “physical” robot had to await the advent of its underlying technologies during the course of the twentieth century. In 1920, the term robot derived from “robota” which means subordinate labor in Slav languages [1]. Around the middle of the twentieth century, the first robots were realized. In 1961 the first industrial robot, “Unimate”, created by American inventor George Devol, was in operation on a General Motors assembly line[2]. They benefited from advances in the different technologies of mechanics, controls, computers and electronics. As always, new designs motivate new research and discoveries, which in turn lead to enhanced solutions and thus to novel concepts. This circle over time produced that knowledge and understanding which gave birth to the field of “robotics”[1]. over time as the science developed in all research areas and new ones emerged the field of robotics gained a lot of attention in terms of investments and research and robotics became combination of various disciplines, electrical and electronics, computer science and engineering, mechatronics and mechanical design, control and automation systems, AI and cognitive systems ... etc.

By the time robots became fully capable of providing help and usefulness and precision to human tasks and activities under human control and supervision, the need for the robotic systems

to be more independent and to possess the ability to accomplish tasks and provide services by their own became a necessity and vital part of the future robots, after which the term of autonomous robots emerged. An autonomous robot is a robot that performs behaviors or tasks with a high degree of autonomy (without external influence). Autonomous robotics is usually considered to be a subfield of artificial intelligence, robotics, and information engineering [3]. Early versions were proposed and demonstrated by author/inventor David L. Heiserman [4][5][6]. After that the robotics systems where classified according to their location on what is called "Autonomy Spectrum" AS from manual to fully autonomous system as in figure 1.1.

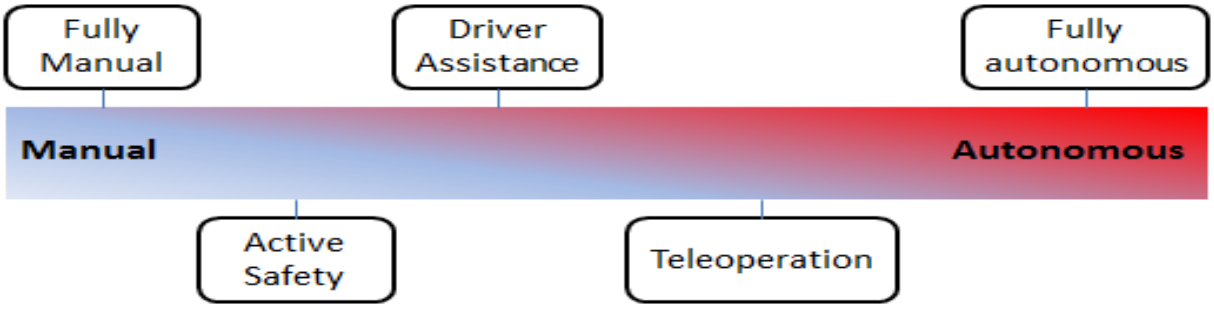


Figure 1.1 The autonomy spectrum [7]

1.1.2 Types of autonomous robots

The autonomous systems umbrella is so vast and variant from robotic arms and manipulators to mobiles robots or combination of both. In this report we will focus on mobile robots. By excluding the space robots, on earth mobile robots are considered to belong to one of three categories according to operating environment: AGVs, AAVs and AUVs.

a) Autonomous Ground Vehicles and robots

An autonomous ground vehicle (AGV) is a vehicle that operates while in contact with the ground and with or without an onboard human presence [8]. AGVs are widely used in many applications where it may be not suitable, dangerous or impossible for the task to be accomplished by human operator presence [9]. Generally, the vehicle will have a set of sensors to observe the environment, and will either autonomously make decisions about its behavior or pass the information to a human operator at a different location who will control the vehicle through teleoperation [10].

Figure 1.2 An autonomous LG cleaner [11].

Figure 1.3 An automated tour-guide robot [12].

b) Autonomous Aerial Vehicles

An autonomous aerial vehicle (AAV), commonly known as a drone, is an aircraft without any human pilot or crew or passengers on board. AAVs are a component of an unmanned aircraft system (UAS), which include additionally a ground-based controller and a system of communications with the AAVs [13]. The flight of AAVs may operate under remote control by a human operator, as remotely-piloted aircraft (RPA), or with various degrees of autonomy, such as autopilot assistance, up to fully autonomous aircraft that have no provision for human intervention [13].

Figure 1.4 Delivery drone by Amazon [14].

Figure 1.5 Autonomous drone in agriculture [15].

c) Autonomous Underwater Vehicles

An autonomous underwater vehicle (AUV) is a robot that operates underwater without requiring input from an operator. AUVs constitute part of a larger group of undersea systems known as unmanned underwater vehicles, a classification that includes non-autonomous remotely operated underwater vehicles (ROVs) controlled and powered from the surface by an operator/pilot via an umbilical or using remote control [16]. Some examples are shown in figure 1.6 and 1.7.

Figure 1.6 Sea exploring robot by Kawasaki [17]. **Figure 1.7** Aquanaute a subsea robot [18].

By far all the presented autonomous systems seem to be very unlike; however, their core functions or the autonomous abilities given to these systems are all the same they just differ in the complexity of their tasks and the complexity of the robot operating environment. The main abilities that should be present in a fully autonomous robot are:

1. Perception:
   - Mapping and environment modeling.
   - Localization.
2. Decision:
   - Planning for paths and tasks.
   - Collision avoidance.
3. Actuation:
   - Path execution.
   - Task accomplishment.

Among the previously mentioned functions contained in an autonomous robot, decision making (path planning and obstacle avoidance) is the most important one, because it is the ability that gives the robot independency from the human control and supervision and make the robot take his own decisions without human intervention; so, it is a crucial criteria in order to make a fully autonomous robot. In the upcoming chapters of this report we will focus on the path planning for AGVs and discuss the problem of path planning and our proposed approach to extract an optimal path for AGV.

1.2 Environment Modeling

Environment or terrain modeling is the task of recreation of the robot workspace by choosing the most efficient means suitable for the modeled environment. According to the work done in [19] world modeling or terrain mapping can be categorized into two main types: Topological maps and Metric maps. However in many practical cases a hybrid methods that employs those two types are often used, as in the work done in[20].

1.2.1 Topological maps

Topological map is a type of diagram that has been simplified so that only vital information remains and unnecessary detail has been removed. This type of maps lacks for measurements, distance and direction information. The most frequently example is the map for a City Subway. This type of maps can be considered as a graph, where vertices or nodes are associated to fixed places while vertex edges are related to possible physical routes between these locations [21].

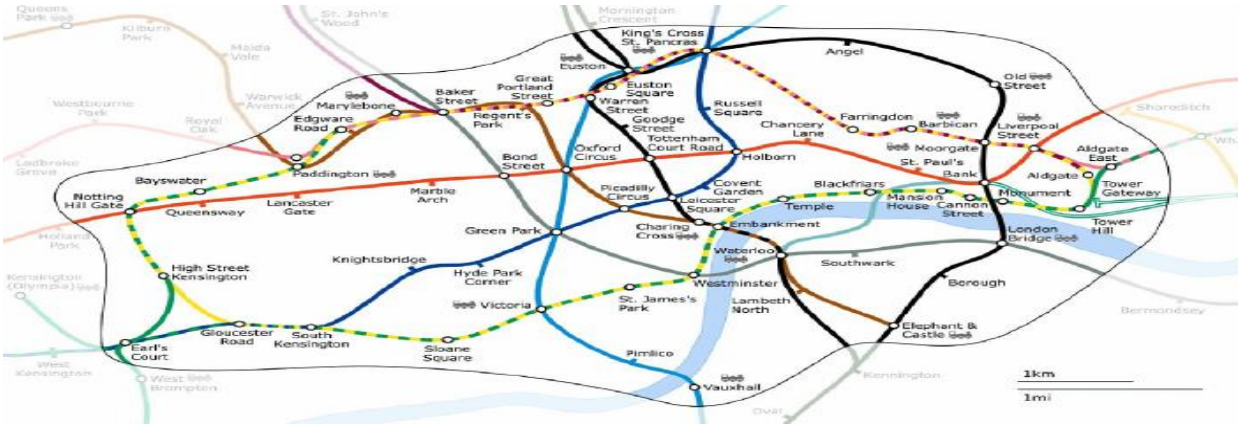


Figure 1.8 Topological map for London underground central zone[22].

1.2.2 Metric maps

According to the way of representation, all metric maps can be divide into two main categories, features based maps or locations based maps.

a) Features based maps

A feature based map is described as a list of N landmarks referred to as map features.

$$M=\{F_1,F_2,\ldots,F_N\}^T \quad (1.1)$$

This type of maps is well suited for large scale environment because it stores only already discovered obstacles as landmarks with updates performed during exploration by appending newly discovered landmarks into the list that represents a map.

Figure 1.9 An example of feature based map with extracted landmarks—trees [21]

b) Locations based maps

The second type of metric maps are location based maps which are described as a list of N elements related to each coordinates x, y and z including occupied as well as free spaces .

$$M=\{L_{x1,y1,z1}, L_{x2,y2,z2},\ldots,L_{xn,yn,zn}\} \quad (1.2)$$

The most widely used metric location-based map family is occupancy or binary grid maps. Where, each cell or location is associated with an occupancy value either ‘0’ or ‘1’. First,

binary grids model mapping algorithms were presented in late 80s by Elfes. A [23]. The idea is to divide observed space into binary cells forming rigid grid. The accuracy in this approach is directly depended on grid size and is a subject of compromises. The small cells give more accurate world representation with tiny details but cost a lot of computing loading and often are just not worthwhile in large empty spaces.

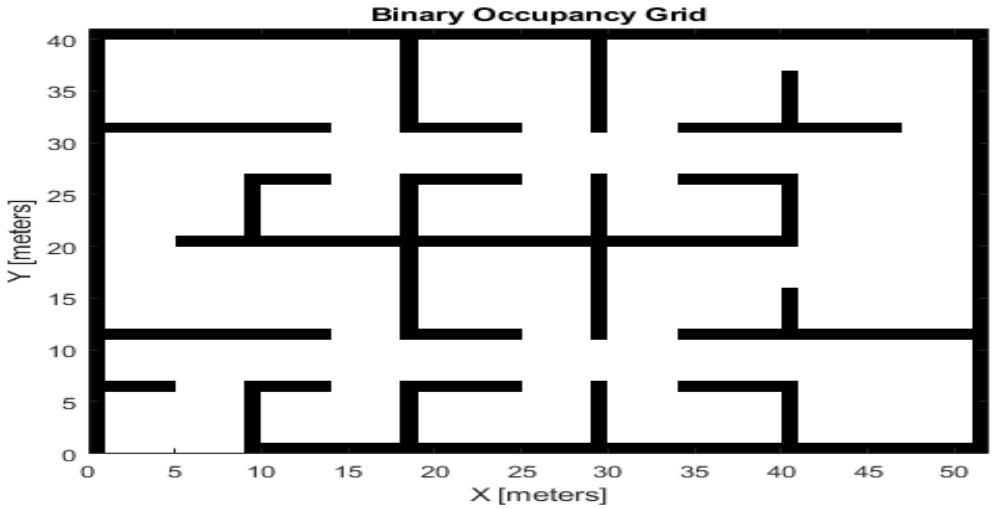


Figure 1.10 A binary occupancy map from MATLAB examples.

The above presented types of world representation are necessary for an efficient navigation of mobile robots and define the type of possible path planning approaches that could be applied in the target workspaces.

1.3 LIDAR data extraction

As seen before, perception or map building and environment sensing is a crucial for the autonomy of robots, and environment modelling is done starting from a data-set collected about the modelled workspace; so remote sensing and ranging devices such as camera, radar, lidar and ultrasound ranging are used to collect this data. In the last decade lidar technology has become more advanced and a set of low-cost lidar systems has emerged to markets; thus, a 2D lidar is considered as a ranging sensor in this report.

Lidar (Light Detection and Ranging) is a remote sensing and ranging technology uses the pulses from a laser to collect measurements. These are used to create 2D and 3D models and

maps of objects and environments. Such sensor utilizes the laser used in everyday applications. This Laser emits an electromagnetic waves (EM) in the optical and infrared wavelengths. That is why it is called an active sensor, meaning that it sends out an EM wave and receives the reflected signal back [24]. It is similar to microwave radar, except at a much shorter wavelength. This means that it will have much better angular resolution than radar but will not see through fog or clouds [24]. It is similar to passive electro-optical (EO) sensors in wavelengths, except that it provides its own radiation rather than using existing radiation, and has many more sensing modes due to control over the scene illumination and some brings its own flashlight and can therefore see at night using near-infrared wavelengths, whereas passive EO sensors have limited capability in the near infrared at night because of insufficient available near-infrared radiation. This means that lidar can have increased angular resolution associated with the shorter wavelengths and still operate 24 hours a day. Lidar will have a frequency of 200 THz (Terahertz) and a wavelength around 1.5 mm, a typical lidar will have a wavelength about 20,000 times smaller than the X-band tracking radar, and 200,000 times smaller than the L-band search radar, with corresponding increases in carrier frequency [24]. X rays and gamma rays will be orders of magnitude shorter in wavelength and higher in frequency than visible or infrared EM radiation as illustrated in figure 1.12. This kind of laser technology is safe to the eye and has been in use for decades [24].

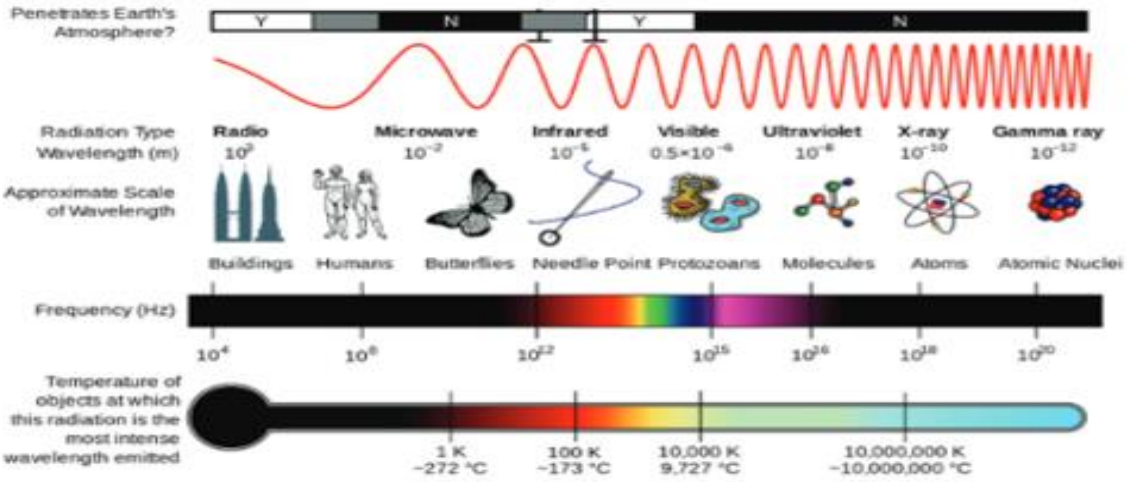


Figure 1.11 the electromagnetic spectrum and the size of common objects compared to the size of a wavelength (adapted from Wikipedia for NAS study) [24].

The lidar sensor on a vehicle consists of a laser transmitter and a light receiver, the transmitter emits light beam that strike nearby objects present in the range of the sensor, the beam of light will be reflected back to the sensor when hitting an object as shown in figure 1.13. The lidar system records each beam's roundtrip data, measuring time to every object in the vehicle's vicinity and the angle of the beam relative to the sensor frame (from the associated hardware in the lidar system as a DC motor, a servo-motor …).

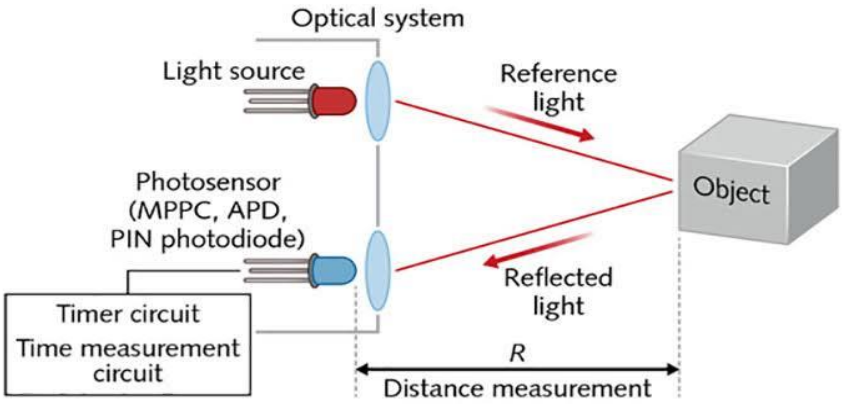


Figure 1.12 TOF ranging sensor schematic [25].

Using the measured time ‘t’ (between emitting and receiving the signal) and the known speed of light ‘c’ , the distance ‘d’ between sensor and target can be calculated with a high accuracy using the following formula 2.1 [26].

$$d = \frac{c \times t}{2} \tag{2.1}$$

A rotating or spinning of this sensor located at the top of the vehicle will capture a 360-degree field of view at a rapid rate of speed to provide a complete image of the vehicle's surroundings. Conversely, solid-state lidar sensors are fixed in place and point in a single direction with 90 to 120 degrees field of view [26]. The scan frequency of rotating system depends only on the TOF sensor element sampling rate. When the system starts a new scan, it returns an array of range and corresponding beam’s angle as shown in figure 1.13.

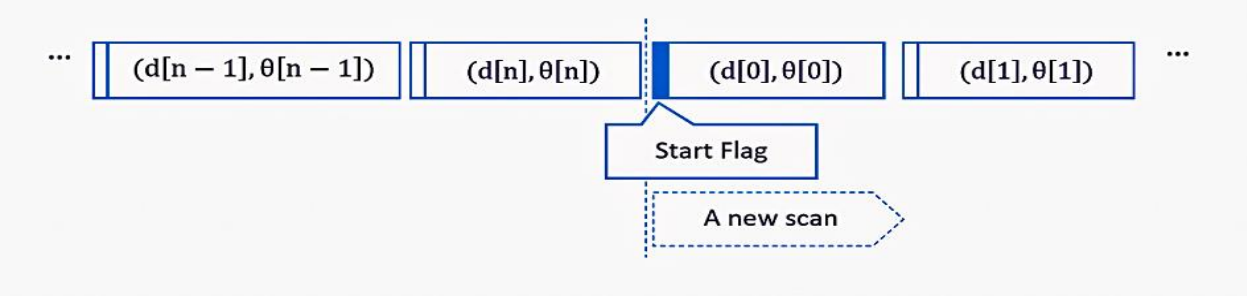


Figure 1.13 output data of a LIDAR sensor [27].

After getting a set polar coordinates points from the lidar as (d_i , θi) in the sensor's frame, it should be converted into cartesian coordinates (x_i , y_i) in the same frame. But first a median filter is applied to the range row to reduce the sensor error between individual samples. Then the conversion to cartesian coordinates is done by means of formulas 2.2.

$$\begin{cases} xi = di \times \cos \theta i & \theta \in \{0; 2\pi\} . \\ yi = di \times \sin \theta i & i = 1.2 \dots 360 . \end{cases} \quad (2.2)$$

So if we assume that the sensor will scan 360 samples per revolution then at each scan the sensor will return (x , y) coordinates of 360 points in the local frame of the sensor.

1.4 Path planning and previous relevant works

1.4.1 Path planning and problem formulation

Path planning from location A to location B, simultaneous obstacle avoidance and reacting to environment changes are simple tasks for humans but not so straightforward for an autonomous vehicle. These tasks present challenges that each autonomous mobile robot needs to overcome to become autonomous as mentioned previously. Path planning problem might be formulated in a sentence as follows: "finding safe obstacle-free road from initial state 'S' to target state 'G' ". This task solution highly depends on both word and robot configurations and their way of representations. In accordance with the world representation type, navigation may be divided into two large groups, structured and unstructured. The first is defined mainly by man-created structures like roads or buildings where navigation problem is limited to finding and following these possible roads. The unstructured type is more general problem and more complicated, in order to tackle this problem it has been subdivided into smaller and more specific

situations. Path planning can be used in fully known or partially known environments; as well as, in entirely unknown environments where sensed information and perception algorithms defines the robot surrounding and environment [21].

1.4.2 Path planning algorithms' types

Since the concept of autonomous robot has emerged to existence many theoretical and practical navigation approaches has been developed to tackle the problem of path planning.

a) Mathematical overview

Robot and word configurations are connected by Configuration Space concept. The definition of robot's Configuration Space Q, also called Q-space is critical for any efficient path planning algorithms. The Q-space is a n-dimensional space that includes all robot's possible configurations in an operating world W and it is represented as $q=[q_1,q_2,\ldots,q_n]^T$. Part of the configuration space that is occupied by the obstacles Oi is denoted $Q_{obs} = \cup_i O_i$. So the free part of the environment without the obstacles is defined by eq1.3

$$Q_{free} = Q - Q_{obs} \tag{1.3}$$

Q_{free} therefore contains space where the mobile system can plan its motion, hence the path planning task might be defined as finding a continuous trajectory in Q_{free} [28].

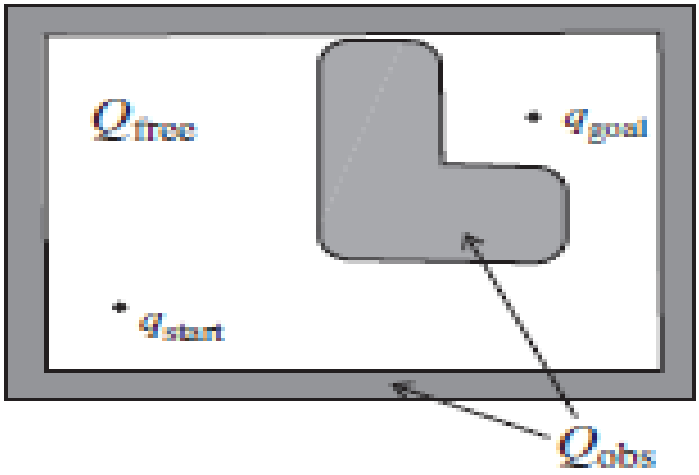


Figure 1.14 Representation of free and occupied spaces [28].

For an AGV the robot configuration can be expressed as q=[x y θ] where (x ,y) are the cartesian coordinates of the robot center in the world frame, and θ represents the robot orientation with respect to x-axis of the world frame, (positive angle is to be taken counter clock-wise). In most cases Q_{obs} is inflated by the robot radius and add some small extra safety distance in order to treat the robot as point in path planning algorithms as explained in eq1.4 and shown in figures 1.15 and 1.16 below:

$$r_{r+s} = r_{\text{robot radius}} + r_{\text{safety}} \quad (1.4)$$

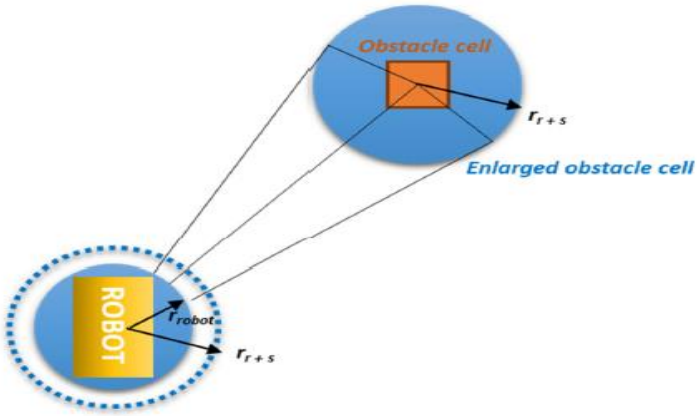


Figure 1.15 obstacle inflating process

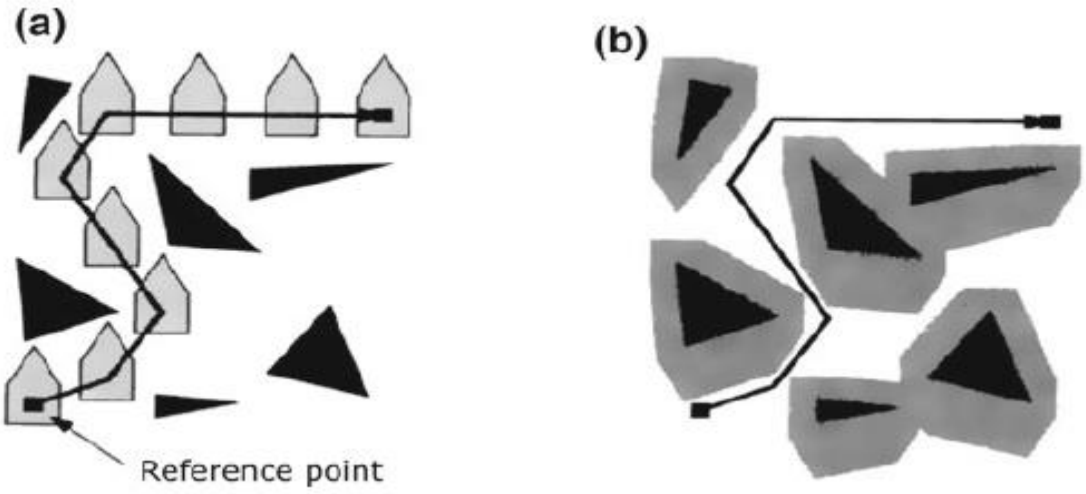


Figure 1.16 (a) Robot work space with obstacles. (b)Its configuration space representation [29].

b) Bug algorithms family

Perhaps the most straight forward path planning approach is to move toward the goal, unless an obstacle is encountered. The Bug1 and Bug2 algorithms [30] are among the earliest and simplest sensor-based planners. These algorithms assume the robot is a point operating in the plane with a contact sensor or a zero range sensor to detect obstacles. These algorithms require two behaviors: move on a straight line and follow a boundary. When the robot has a finite range (nonzero range) sensor, then the tangent Bug algorithm [31] is a Bug derivative that can use that sensor information to find shorter paths to the goal. This family of algorithms has witnessed several improvements but still lack the ability to find a solution in several practical cases and the generated path is not optimal.

c) Graph Based Path Planning

Unlike the previous algorithms that do not need map and are based on reacting behavior, the largest group in which the motion planning or roadmaps depend on graphs and a previously represented map of the workspace. In most cases, the possible path in Q can be considered as a graph G = (V, E, W_E). The graph consists of sets of vertices (nodes) V related to physical locations, and edges E ⊂ V × V which connects pair of vertices and are associated with each edge costs $W_{E;}$ costs are defined by cost function. In this type of path planning algorithms, the two problems of planning that need to be solved can be defined as: graph construction and shortest path search. In the first step is trying to connect the start-pose of the robot with the goal-pose thought a graph G; then, a search algorithm is used to find the optimal path between the two poses. Some of the well-known graph based approaches are the “visibility graph” and “Voronoi diagrams” and there are also many Hybrid Visibility-Voronoi solution applications that optimize path length and path smoothing [32]. But ones of the mostly used graph based algorithms is A*[33] and its dynamic version D* [34]. A* algorithm introduces a heuristic function given by eq1.5:

$$f(X) = g(X) + h(X) \tag{1.5}$$

to cost estimation in path searching. Where g(X) is related to actual cost and h(X) is the estimated cost. It allows to search only in nodes (vertices) with minimal costs during expansion

without costs calculation for already visited vertices. It means that the path calculation will be as fast as possible until better-informed heuristic is not included.

d) Potential field Based Path Planning

The potential field method describes the environment with the "potential field", which can be thought of as an imaginary height. The goal point is in the bottom, and the height increases with the distance to the goal point and is even higher at the obstacles (or seen as robot and obstacles have similar charges and goal has different charge to attract the robot), as shown in figure 1.17. A path planning procedure can then be explained as the motion of the ball that rolls downhill to the goal, A potential field is expressed as a sum of the attractive field due to the goal point $U_{attr}(q)$ and a repulsive field $U_{rep}(q)$ due to the obstacles:

$$U(q) = U_{attr}(q) + U_{rep}(q) \quad (1.6)$$

The goal point is the global minimum of the potential field. These methods suffer from trapping into local minima. Thus, potential field methods need additional effort to overcome the problem of local minima as well as to find a minimum length path to the goal point [35].

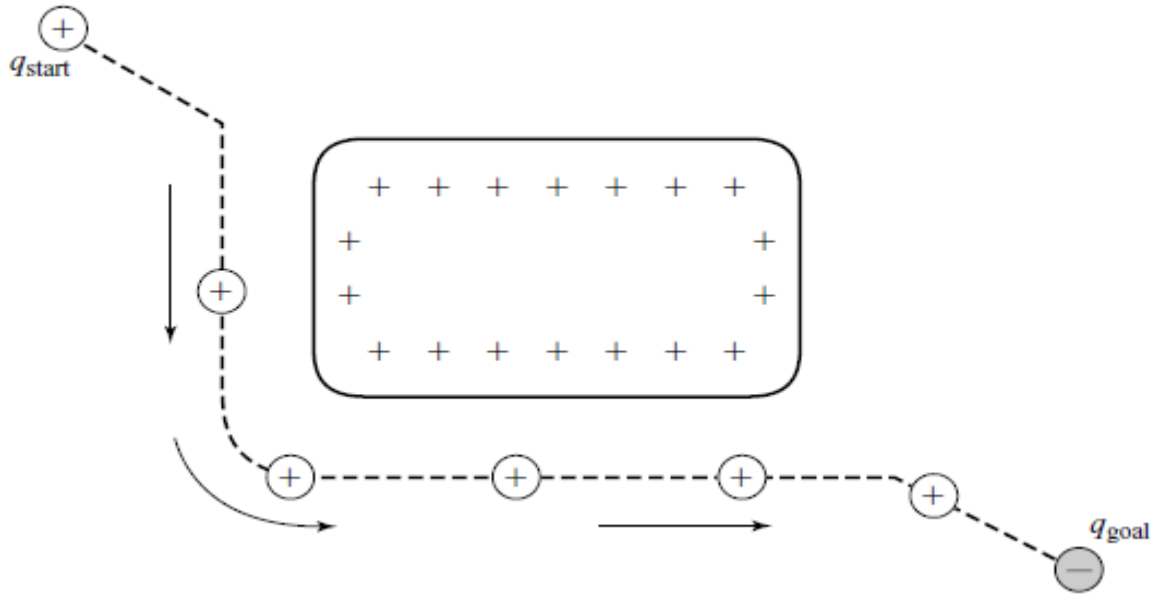


Figure 1.17 positive charges repels and the goal negative charge attract the robot

e) Sampling Based Path Planning

Up to now the presented path planning methods required explicit presentation of free environment configuration space. By increasing the dimension of the configuration space these methods tend to become time consuming. In sampling-based path planning, random points (robot configurations) are sampled; then, collision detection methods are applied in order to verify if these points belong to the free space. From a set of such sampled points and connections among them (connections must also lie in the free space) a path between the known start and the desired goal point is searched. Sampling-based methods do not require calculation of free configuration space Q_{free}, which is a time-consuming operation for complex obstacle shapes and for high dimensions Q. Instead, the random samples are used to present the configuration space, and independently from the environment geometry a path planning solution can be obtained for a wide variety of problems. In comparison to decomposition of the environment to cells, the sampling-based approaches do not require a high number of cells and time-consuming computation that is normally required to achieve accurate decomposition to cells. Due to inclusion of stochastic mechanisms (random walk) such as in random path planner the problem of being trapped in some local minimum (e.g., as in the potential field method) is unlikely, because motion is defined between randomly chosen samples from the free space. Sampling-based approaches can be divided into the ones that are appropriate for a single path search and those that can be used for many path search queries, an example of both approaches that are extensively used and have a lot of attention in the last decade are Rapidly Exploring Random Tree (RRT) and Probabilistic Roadmap (PRM) .

A. PRM

Creating a path in the configuration space using the Probabilistic Roadmap Planner is considered as a two phase process: planning, and query. In the planning phase, the planner keeps on sampling the configuration space until a certain number of collision free configurations has been generated. Each sampled configuration, or simply said point in Q, is connected to the neighbors lying within an area of predefined size. Here, the connection between two points is established by a straight line path that is only maintained by the planner if it is found to be collision-free. The resulting network, called a roadmap, stored by the planner is then used to solve all subsequent motion planning

queries. A motion planning query is specified by a start and goal configuration, q_s and q_g. In order to solve a query, both configurations are connected in a first step to the respective closest configuration of the roadmap. Then, starting from q_s a solution path is found by moving iteratively to the neighboring node providing the minimum distance to the goal configuration (see figure 1.18). Usually the distance metric used is the Euclidean norm[36].

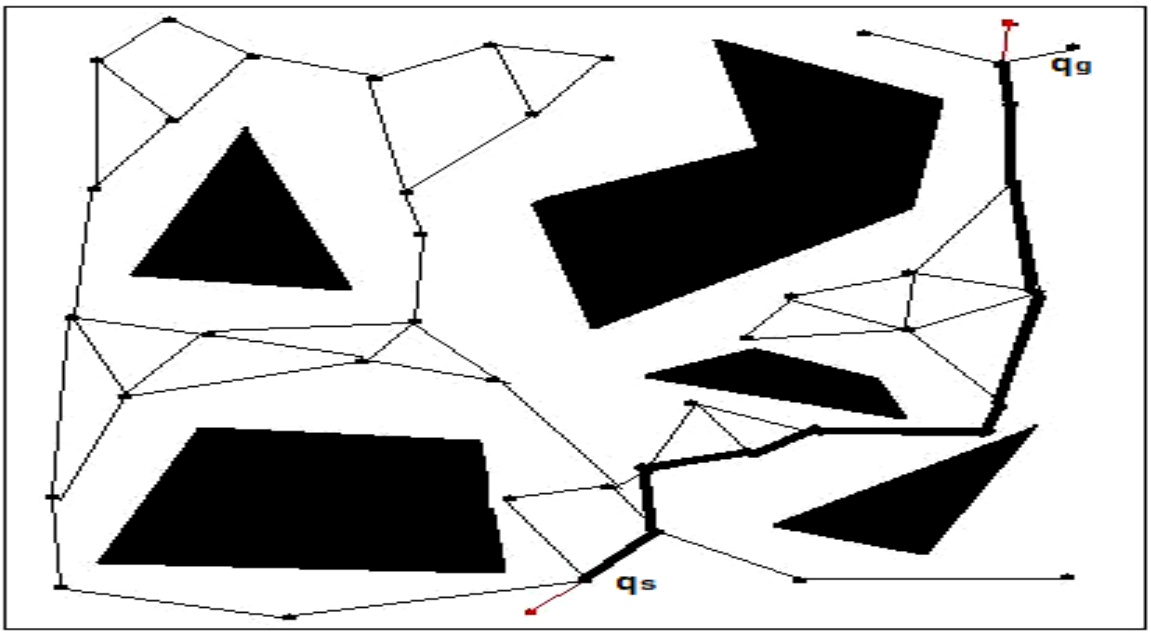


Figure 1.18 Probabilistic Roadmap Planner. Solution to a motion planning query (solid path).

B. RRT

RRT Planner is a single-query probabilistic motion planning algorithm that searches for a path by incrementally expanding trees T in the configuration space, also referred to as a Rapidly Exploring Random Tree. From an algorithmic point of view, trees are represented by a data structure. As opposed to the PRM planner, this kind of algorithm only explores a subset of the configuration space, relevant for solving the query in mind. Furthermore, the tree growing procedure is repeated for each new query, for which reason this approach is referred to as a single-query method. The routine to be repeatedly applied for growing the trees in Q proceeds as follows:

At the beginning of each iteration a random configuration q_{rand} is sampled from Q according to a uniform probability distribution. Then, the closest configuration to q_{rand} in

T, also called the nearest neighbor q_{near}, is found and the local planner generates a new configuration q_{new} at a distance ε from q_{near} along the segment connecting q_{near} and q_{rand}. Afterwards, both q_{new} and the segment joining it to q_{near} are checked for collisions. If they are found to be collision-free, the search tree T is expanded by the new configuration and that segment. The simplest form of a RRT Planner grows a single tree rooted at the start configuration q_{start}. Then, during the search the tree tries occasionally to connect to the goal configuration by using it as q_{rand} in the tree expansion procedure[36] as explained in figure 1.19.

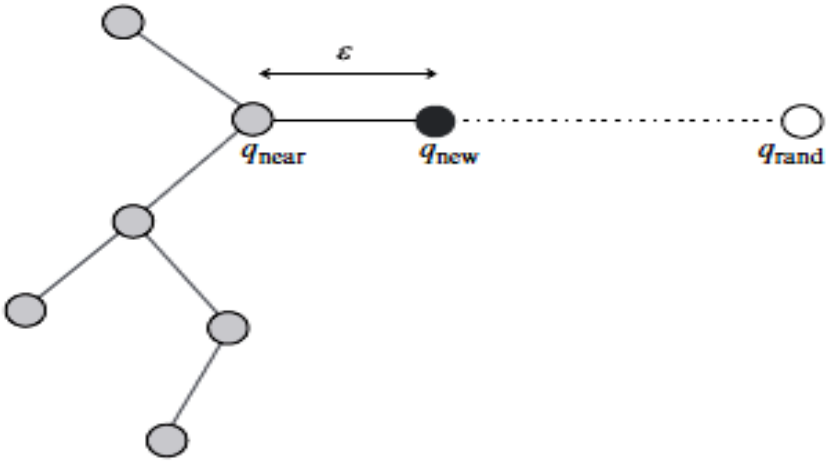


Figure 1.19 Example of a tree expansion step.

1.4 Some previous relevant works

Despite these previously discussed approaches in the last two decades many attempts to derive new planning approaches either by enhancing the existent ones or hybrid combinations of them or creating a totally new approaches, some of the related works are:

Laser simulator: A novel search graph-based path planning approach by Mohammed. AH Ali and Musa Mailah [37]. In this paper a novel technique called laser simulator approach for visibility search graph-based path planning has been developed to determine the optimum collision-free path in unknown environment in a two-dimensional map. In addition, a comparative study on the laser simulator approach, A* algorithm, Voronoi diagram with fast marching and Point Bug algorithms was performed to show the benefits and drawbacks of the

proposed approach. Another method was proposed in [38] where a new approach of path planning technique is presented; in which the virtual size of the obstacle present in the environment is assumed to be increased approximately (2n+1) times of the size of the cell. The experimental analysis of the proposed method shows the improvement in the path planning which reduces the chances of collisions. The authors in [39] present a hybrid algorithm to the problem of path planning that can be used to find global optimal collision-free paths. This approach relies on combining potential field (PF) method and genetic algorithm (GA) which takes the strengths of both and overcomes their inherent limitations. In this integrated frame, the PF method is designed as a gradient-based searching strategy to exploit local optimal, and the GA is used to explore over the whole problem space.

1.5 Conclusion

This chapter, presented the path planning problem formulation and the motivation for this work and other related works. In addition, some preliminary materials and necessary notions to conduct our proposed work. Chapter 2 will introduce our proposed approaches for envoronment modeling and path planning.

Chapter II

Environment Map building and path planning

In this chapter, a simple strategy to model a 2D environment for robot navigation based on extracted data from lidar sensor as presented in chapter 1 is explained. Then, our proposed strategy for robot path planning is presented in details along with optimal path extraction. Last, the need and motivation for a smooth path for robots and the implementation of a widely used smoothing technique is presented.

2.1 Map building using LIDAR data-Binary occupancy Map

In the previous section we have seen that Lidar presents the environment ranging data as (x, y) points in the local frame of the sensor. Before passing this data to represent a map we should first get rid of points that did not hit an obstacle and preserve only points that represent the presence of an obstacle as shown in figure 2.1.

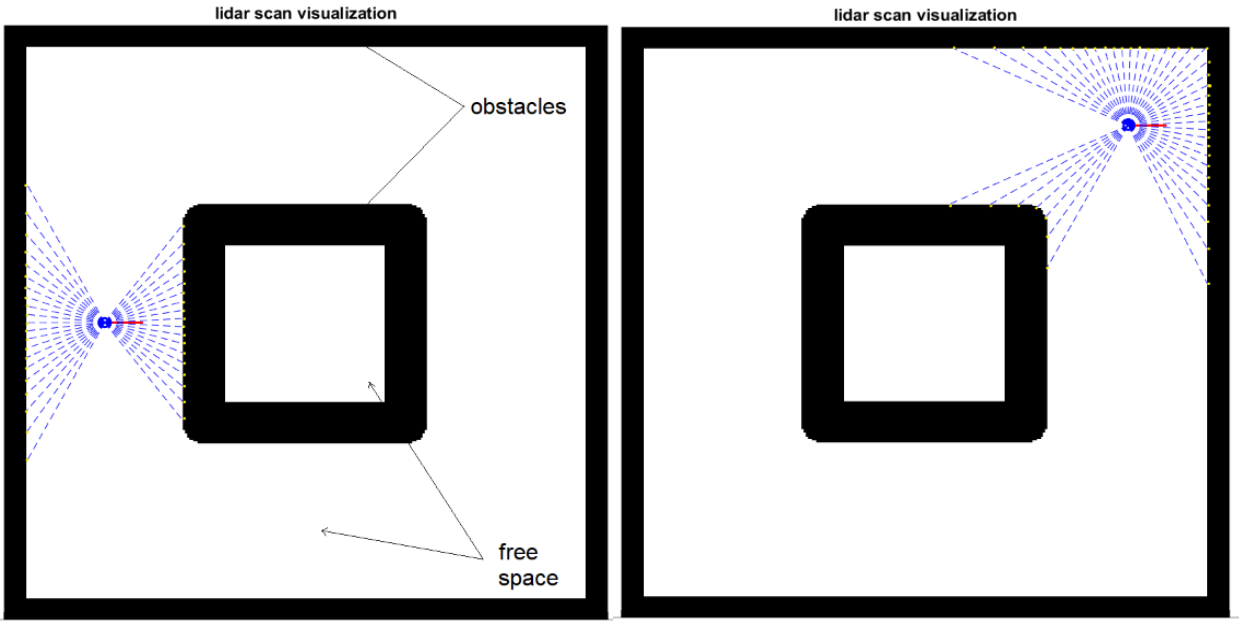


Figure 2.1 Lidar scan visualization, beams that did not hit an obstacle are discarded.

After this process we get set of points that represent the boundary of an obstacle in the vicinity of the robot but the coordinates of these points are defined with respect to the lidar sensor local frame; thus, the coordinates need to be converted to the global frame of the map as illustrated in figure 2.2. The relation between the two frames is defined by the translation vector $[X_t, Y_t]^T$ and the rotation matrix R. where (X_t , Y_t) is the coordinates of the lidar in the map frame assumed to be pre-defined.

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

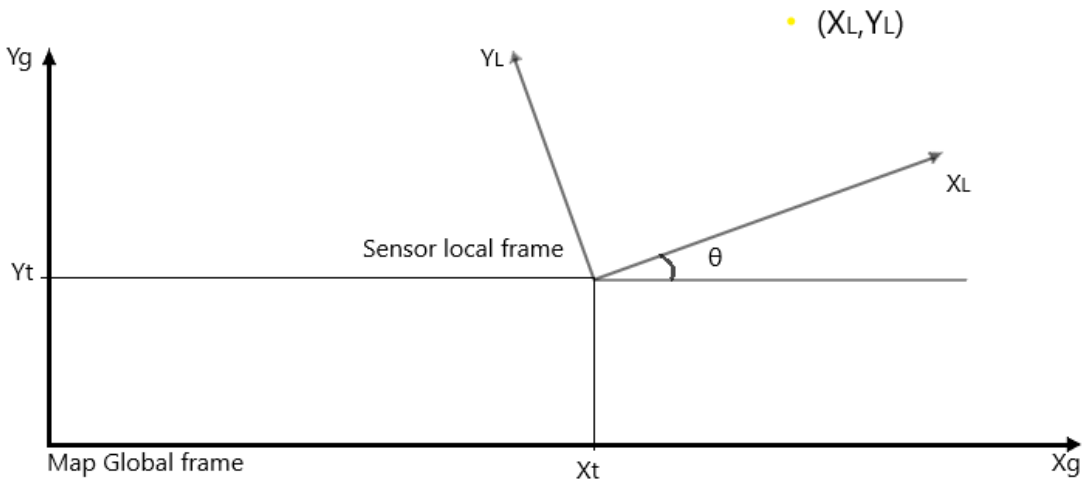


Figure 2.2 Global and Local coordinates transformation .

At this point we are able to read a single scan data and transform its coordinates to the global frame of the map; now these coordinates are to be converted into indices in the map grid instead of length in meters. First we define the map size as [length , width] in meters and the resolution as "N" grids per meter and we define an occupancy matrix with size of length$\times N$ by width$\times N$ with all elements initially '0' (means free cell).

$$\begin{cases} X = X_g \times N \\ Y = Y_g \times N \end{cases} \tag{2.2}$$

Then, we set the values of the entries (X, Y) in the occupancy matrix to '1' (means occupied cell). This whole process is repeated for multiple predefined sensor's locations and the occupancy matrix is updated along the process as shown in figures 2.3 and 2.4.

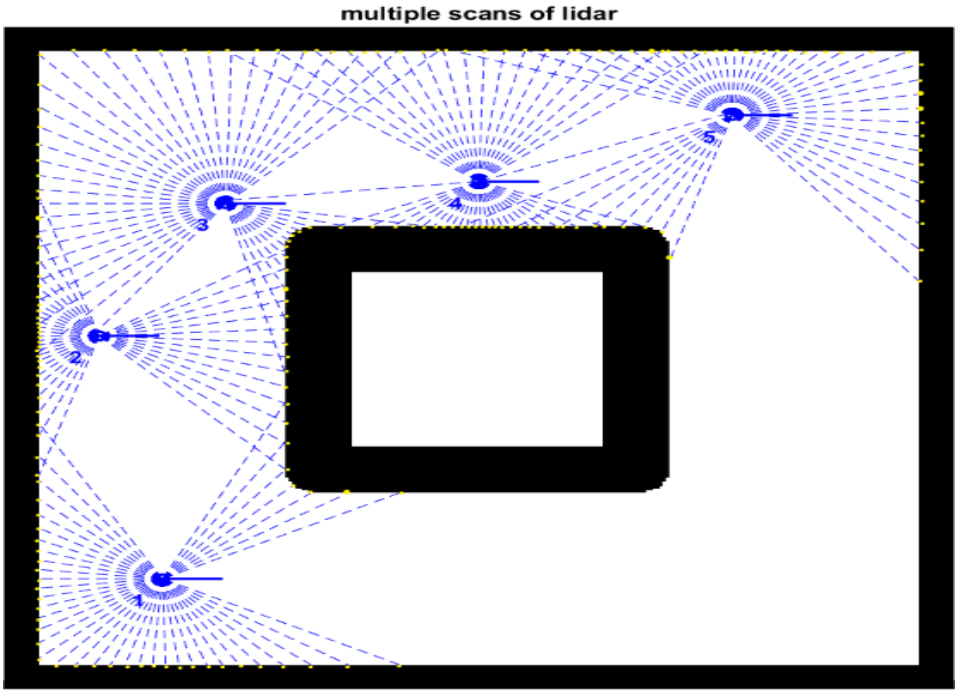


Figure 2.3 Multiple scans of a lidar at different locations.

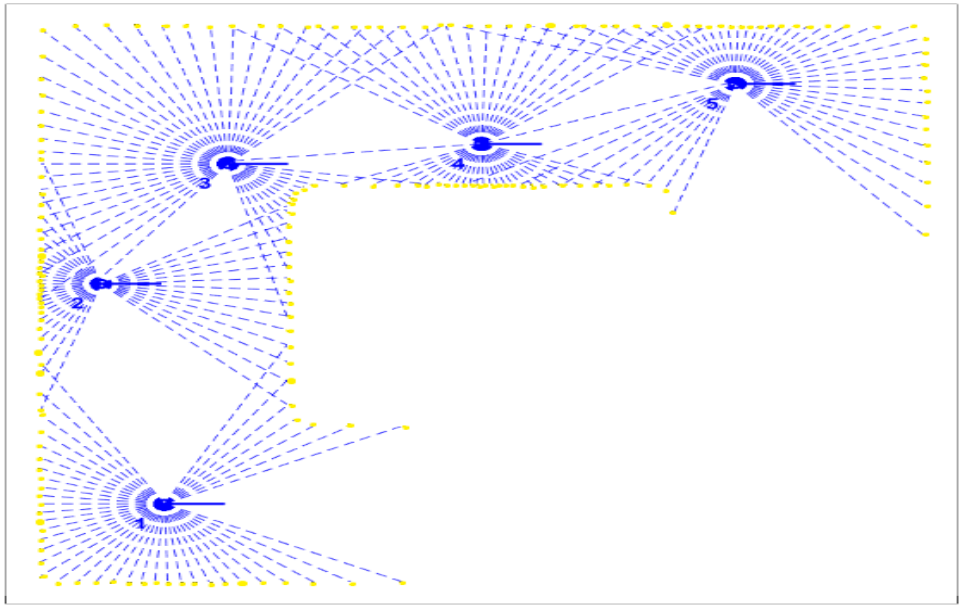


Figure 2.4 The reflection points returned by the lidar sensor.

After setting the occupancy of the reflected points to ‘1’ we get the occupancy map shown in figure 2.5.

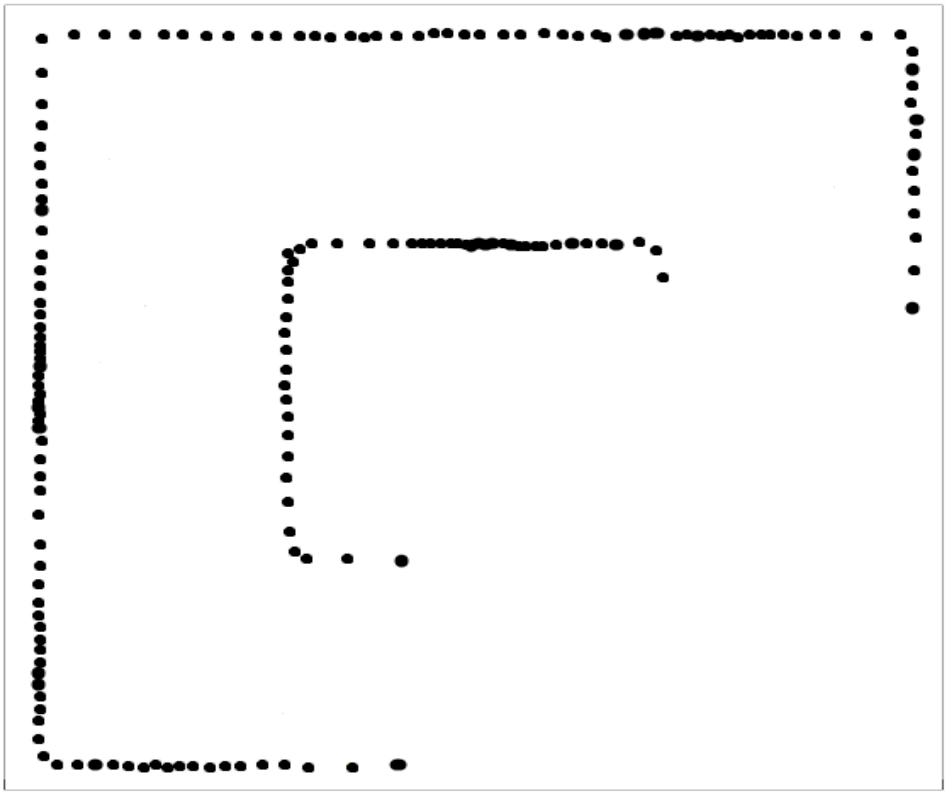


Figure 2.5 The resultant binary occupancy map.

At this point the resultant binary occupancy map is inflated by a distance equals to the robot radius and some extra safety distance as was illustrated in figure 2.6 to reconstruct the original map with some error, as the number of lidar scans is increased the final map will be more accurate.

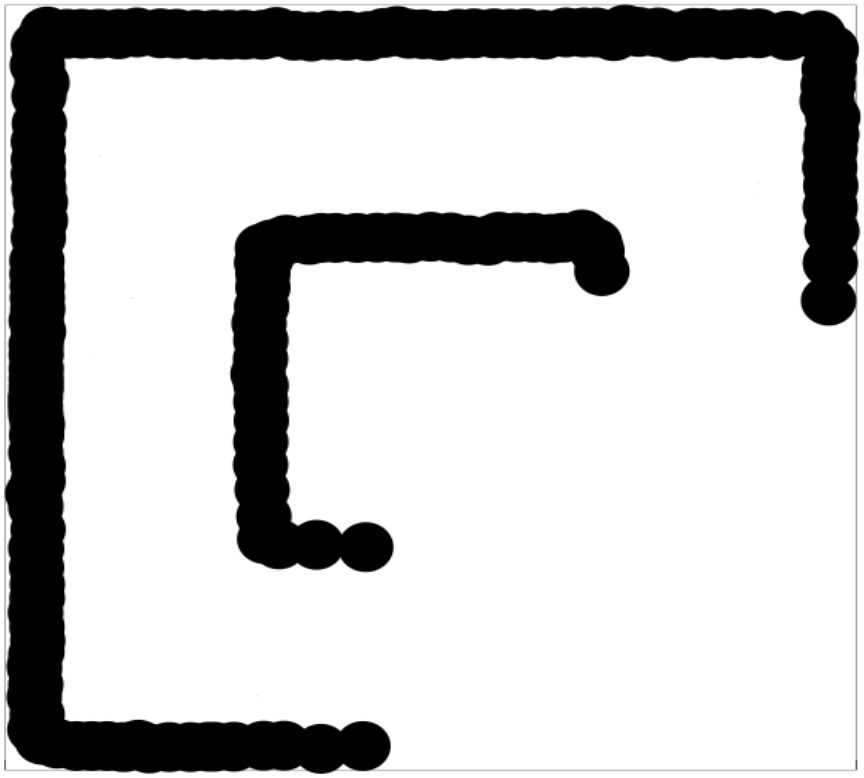


Figure 2.6 The inflated binary occupancy map.

2.2 Environment representation

In this section, we will present some custom binary occupancy maps that were created; they will be used through the simulation to simulate the proposed path planning algorithm and in the comparison and validation of the obtained results. These maps were built with different complexities and obstacles shapes.

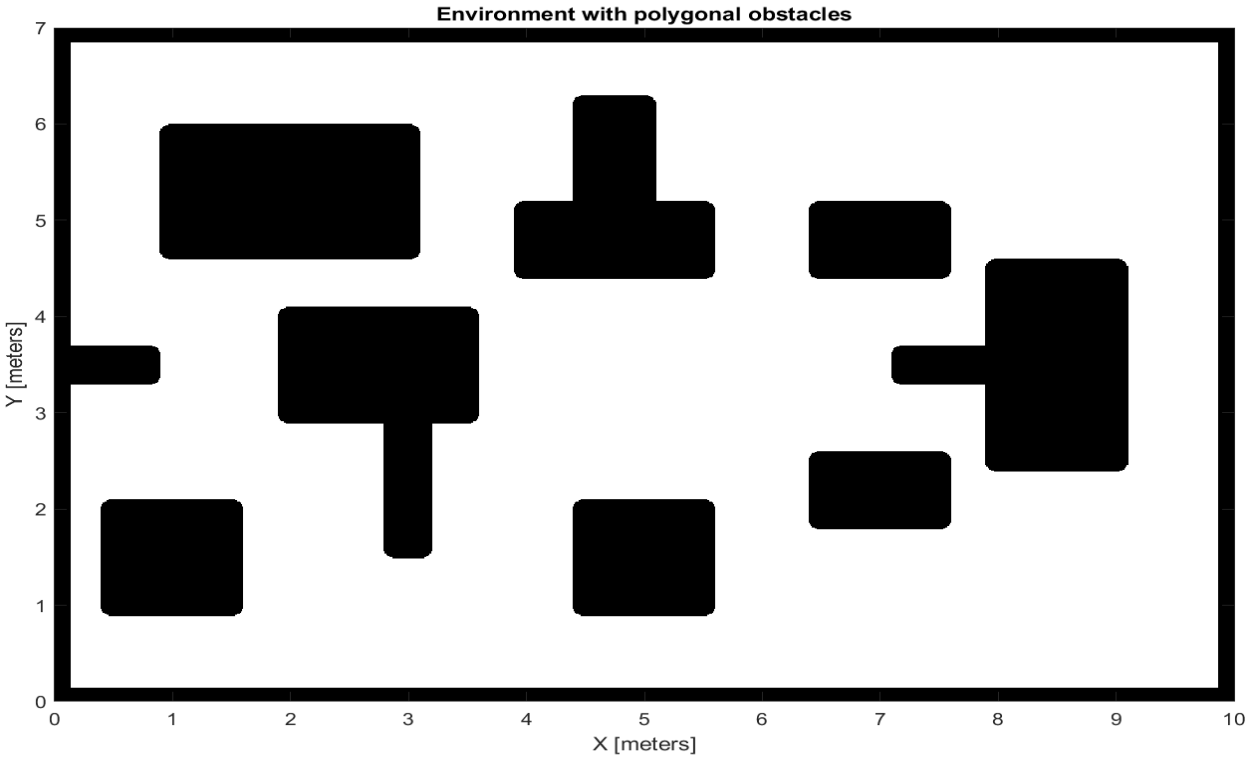


Figure 2.7 Environment with polygonal obstacles.

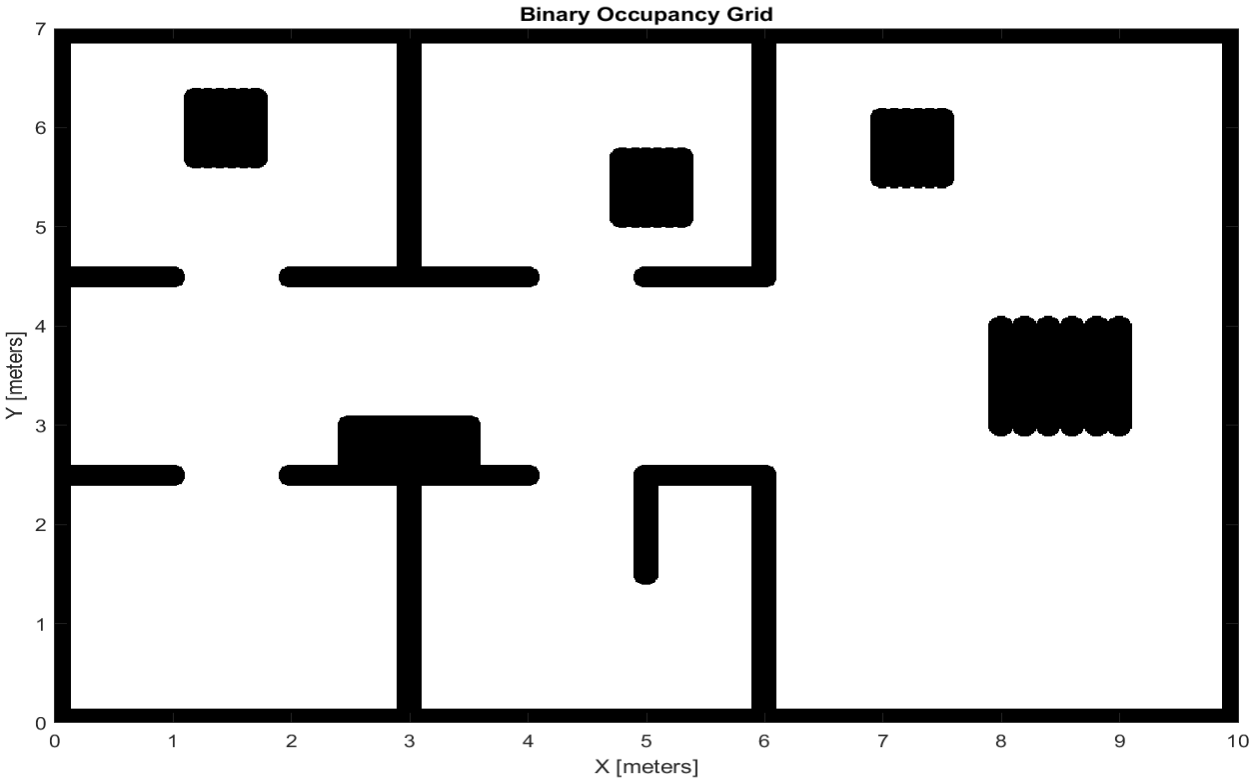


Figure 2.8 Home shaped environment.

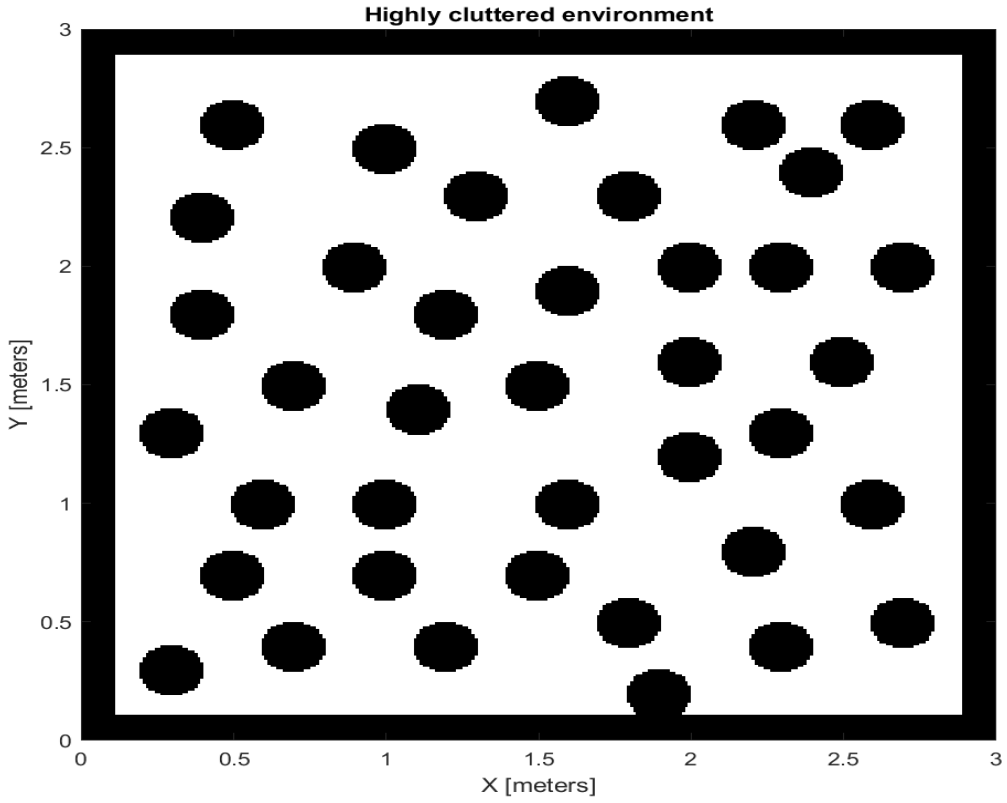


Figure 2.9 Highly cluttered environment.

2.3 Path planning strategy (ray based planner)

A new approach has been introduced in this work to search for the optimum path in unknown and terrains while avoiding obstacles. It is emulating the lidar sensor when it is used to detect the robot environment boundaries and build a map. The main advantage of this approach is its ability to be used for both global and local path planning with the presence of obstacles in unknown environments. And it is a goal oriented approach, means that it uses a criteria (Euclidean distance) that always guide the search process towards the goal unlike other approaches that explores all the map such as sampling based approaches (RRT , PRM) that may explore areas that are not necessary for reaching the goal.

The working principle of RAY planner is described in the following steps:

- The robot environment is modeled as 2D binary occupancy map.
- The start location (X_s, Y_s) and goal location (X_g, Y_g) are well defined before starting planning.
- First check if the entered start and goal points are valid poses, if not return an error.
- The planner start by initiating set of 'N' rays in all directions with specified length 'd' and equally spaced angles as described in figure 2.10.

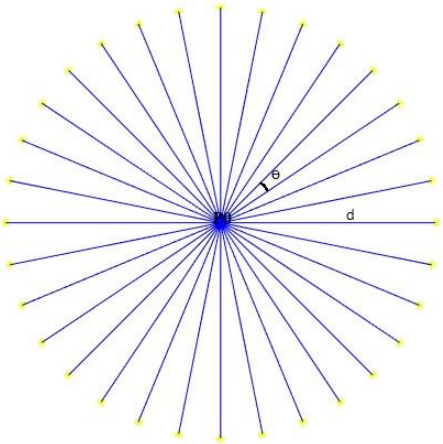


Figure 2.10 rays generating at point 'P'.

Where 'd' and 'Ɵ' are calculated as:

$$d_i = \sqrt{(X_g - X_{pi})^2 + (Y_g - Y_{pi})^2} \quad (2.3)$$

$$\Theta_n = \frac{2\pi}{N} \times n \qquad n = 0.1 \ldots N - 1 \quad (2.4)$$

Where (X_{pi} , Y_{pi}) is the current point in the search process; at the beginning it is the start point (X_s ,Y_s).

- Some of the generated rays will hits an obstacle if one is present within the rays range and some will not, we find the coordinates of the points that represent the end of the rays in both cases using eq2.5 as illustrated in figure 2.11

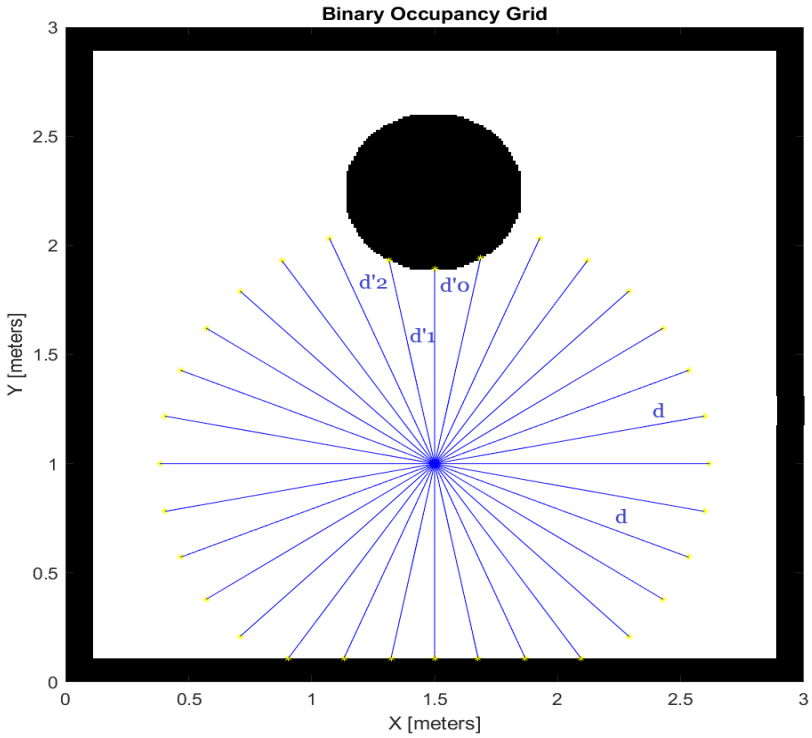


Figure 2.11 Illustration of rays hitting obstacles.

$$\begin{cases} Xn = d\cos(\Theta n) \\ Yn = d\sin(\Theta n) \end{cases} \tag{2.5}$$

- Next the distance between all calculated points and the goal is determined and the point with the smallest distance is selected to be the next point 'p'.
- If the selected point is within a tolerance distance 'ε' from the goal, then the process stops and the algorithm returns the set of the previously visited points. Otherwise, the process continues.
- If the new selected point P_{i+1} is the same as the parent point P_i; then, this will be a critical case because this point will always give the smallest distance to the goal. Here the algorithm enters in a loop and will generate the same point iteration after iteration; because the same point will be chosen to be the nearest one. In this case a random point is selected from the previous set, the process follows normally as illustrated in figures 2.12.

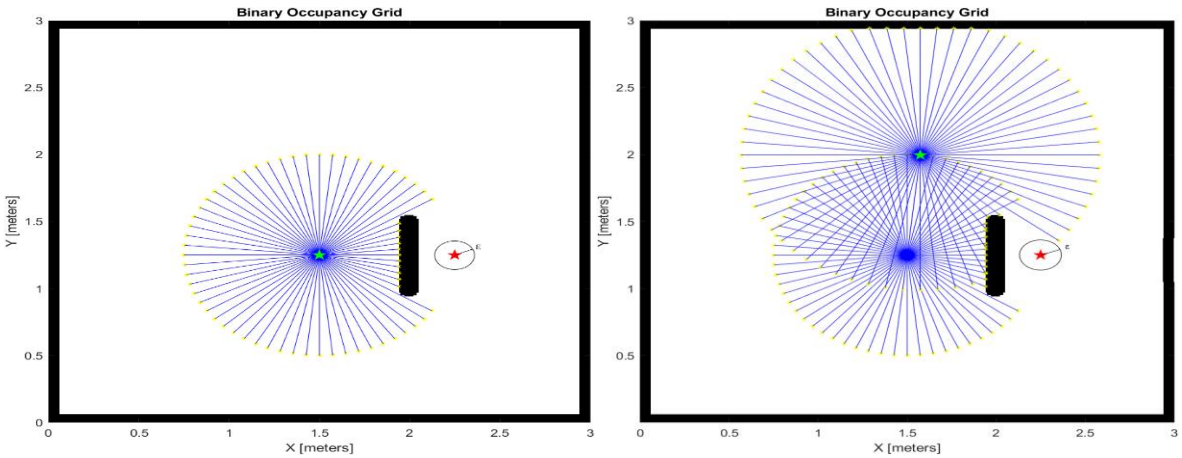


Figure 2.12 The planner selects a random point to ignore the closest point

- Last, when the goal is reached, the algorithm returns $(i + 1) \times 2$ matrix of way-points.

An example of the search process is illustrated in figure 2.13 and a detailed flowchart for the proposed algorithm is shown in figure 2.14.

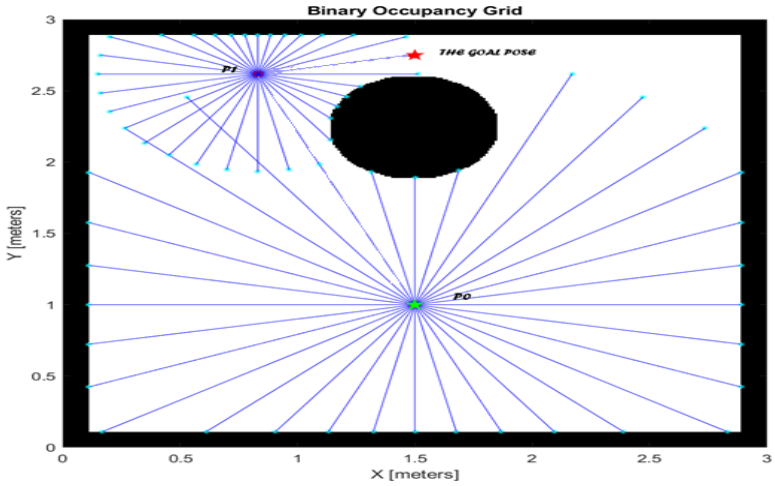


Figure 2.13 An example of the Ray planner search process.

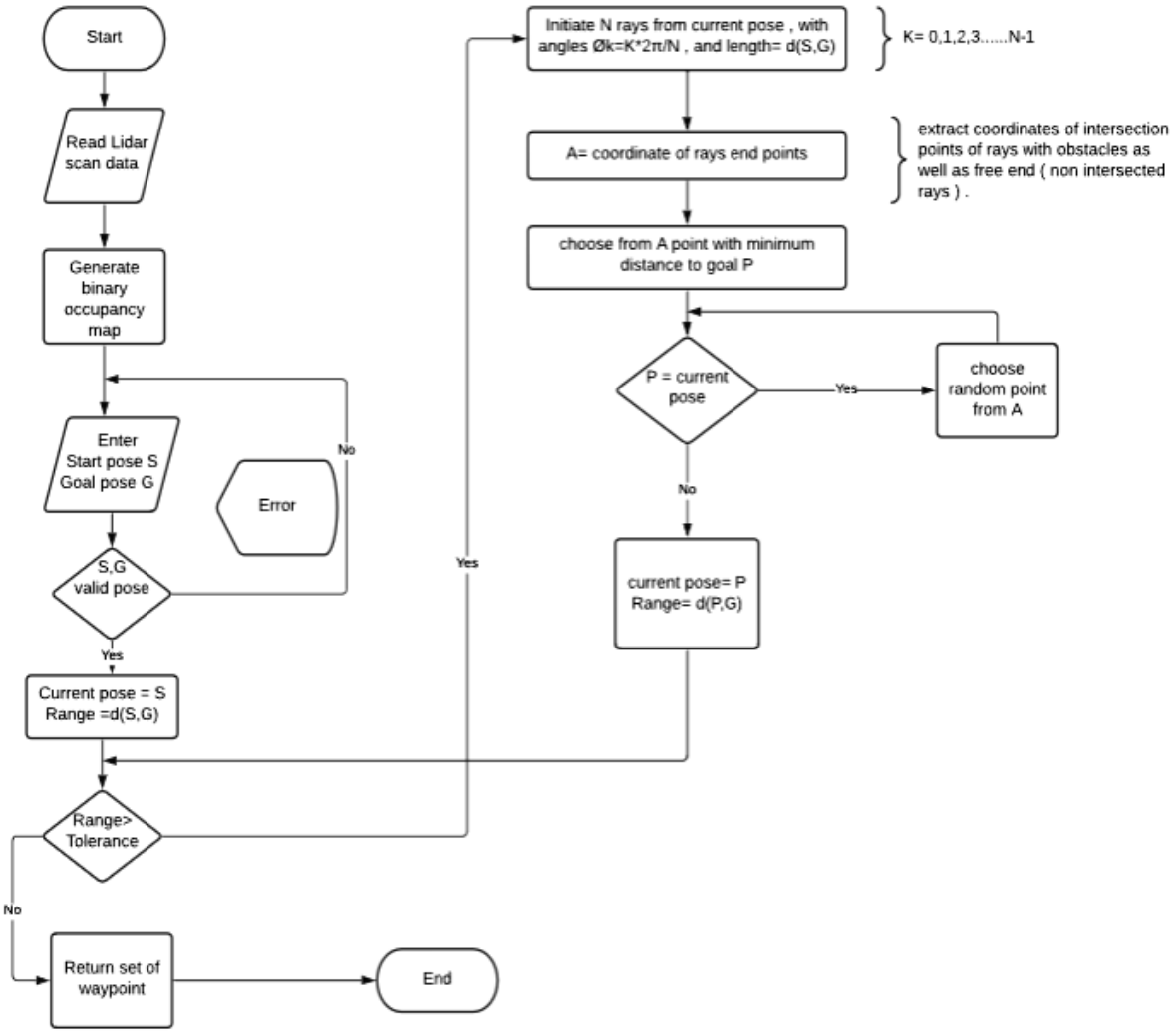


Figure 2.14 Flowchart of Ray planner algorithm.

2.4 Optimal path extraction

The set of waypoints generated by the Ray planner when connected does not give and optimal path to the goal. In more complicated environment it gives something that is nowhere near optimal; so a strategy to extract and optimal path from the generated waypoints is applied. This method uses graph theory in which we construct a graph that represents the planned path [40].

Each node in such a graph is called a waypoint and specifies a special location in the region, an edge between two waypoints denotes a possible path between those two waypoints.

Based on waypoint graph, the path finding can be easily performed. The proposed approach mainly consists of the following steps:

- Starting by getting the set of waypoints $P_i(x_i,y_i)$ from the RAY planner, create 'M'equally spaced intermediate points between every two successive points from this set using eqs 2.6 and 2.7 as shown in figure 2.15.

$$\begin{cases} \Delta x = \frac{|x_{i-1}-x_i|}{M} \\ \Delta y = \frac{|y_{i-1}-y_i|}{M} \end{cases} \quad (2.6)$$

$$\begin{cases} x_{i-1} = x_i + k\,\Delta x \\ y_{i-1} = y_i + k\,\Delta y \end{cases} \quad k=0.1\ldots M-1 \quad (2.7)$$

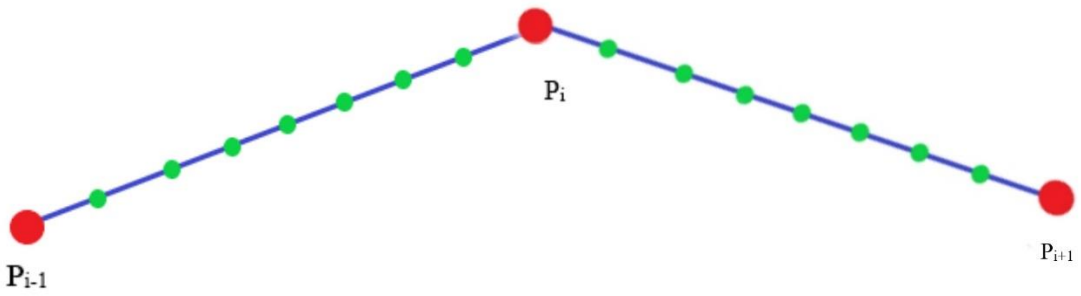


Figure 2.15 Intermediate points.

- Next an undirected graph 'G' is created with nodes represent new waypoints and the edges are the connection between them.
- Check all the valid connections of the nodes and eliminate the invalid connection, this mean when we have connection between two nodes pass through an obstacle we take it as invalid edge. The process is illustrated in figure 2.16.

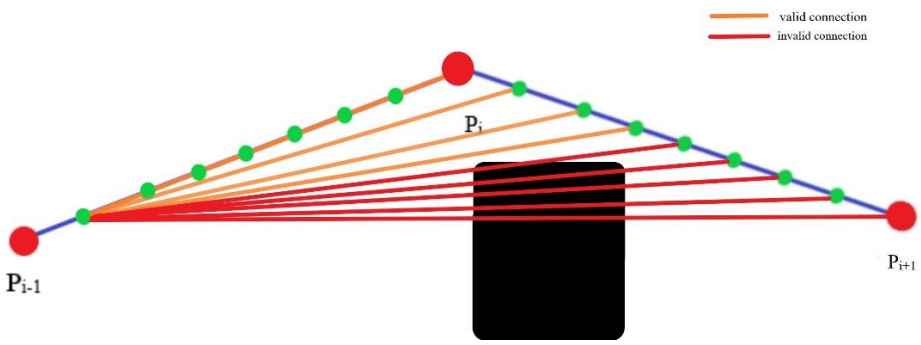


Figure 2.16 Removing the invalid connections for single point.

- The new graph will be simpler and easy to traverse. An example of how this step can simplify the graph is shown in figure 2.17.

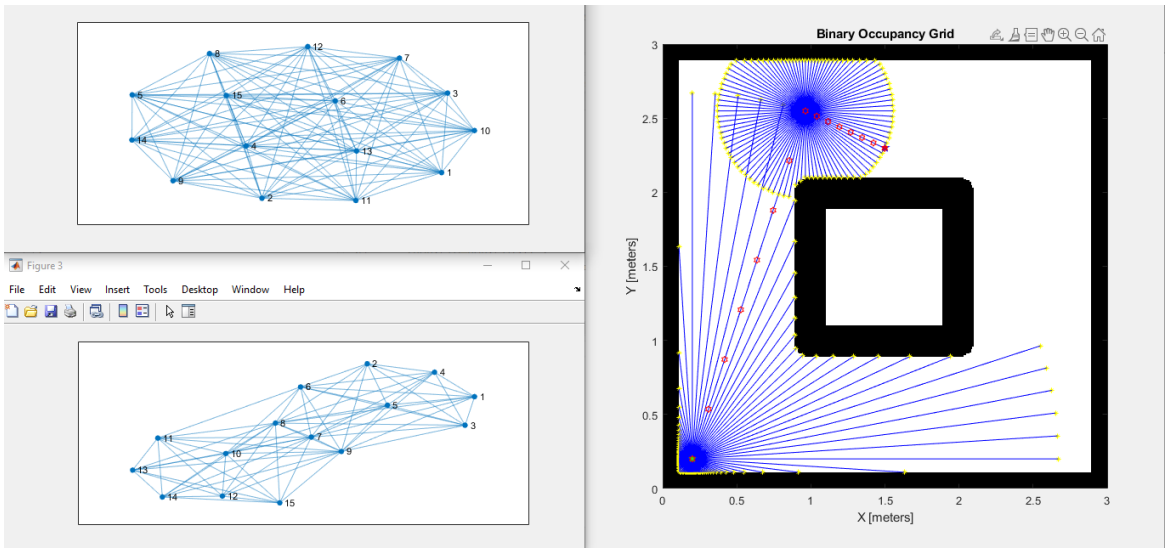


Figure 2.17 Simplification of the graph.

- Then, the weights are added to the edges which represent the Euclidean distance between the nodes, these are the costs to travel from one node to the other. An example of such graph is shown in figure 2.18.

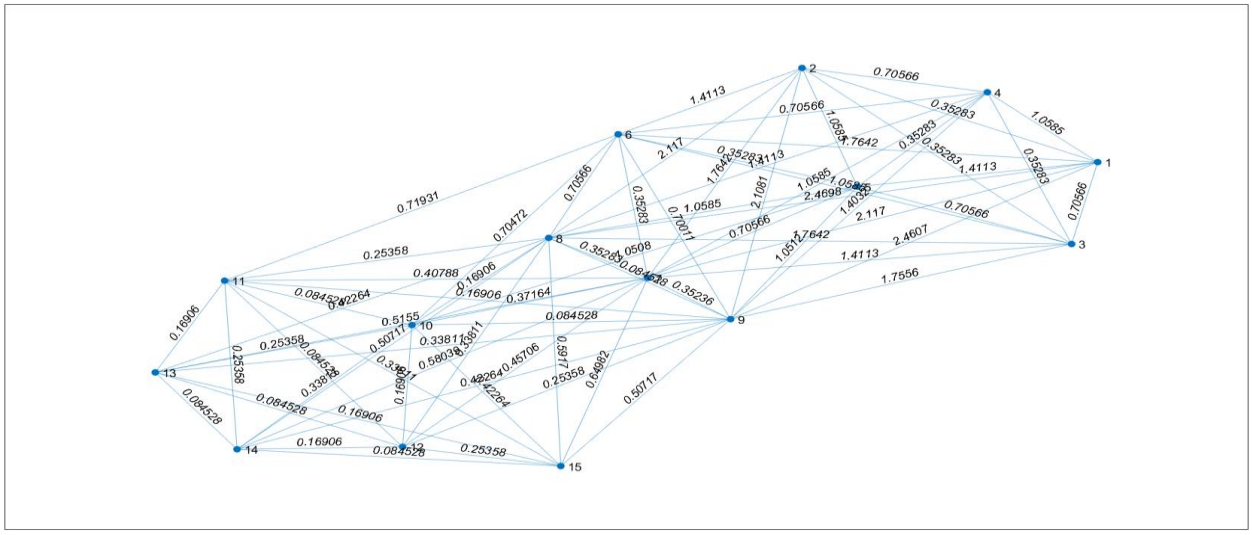


Figure 2.18 A weighted graph, weight are the length of edges.

- As a last step in our method to optimize the distance between the goal pose and the start pose a graph search algorithm is used to find the shortest path between the start and goal nodes in the graph.

The flowchart of the process is illustrated in figure 2.19 bellow:

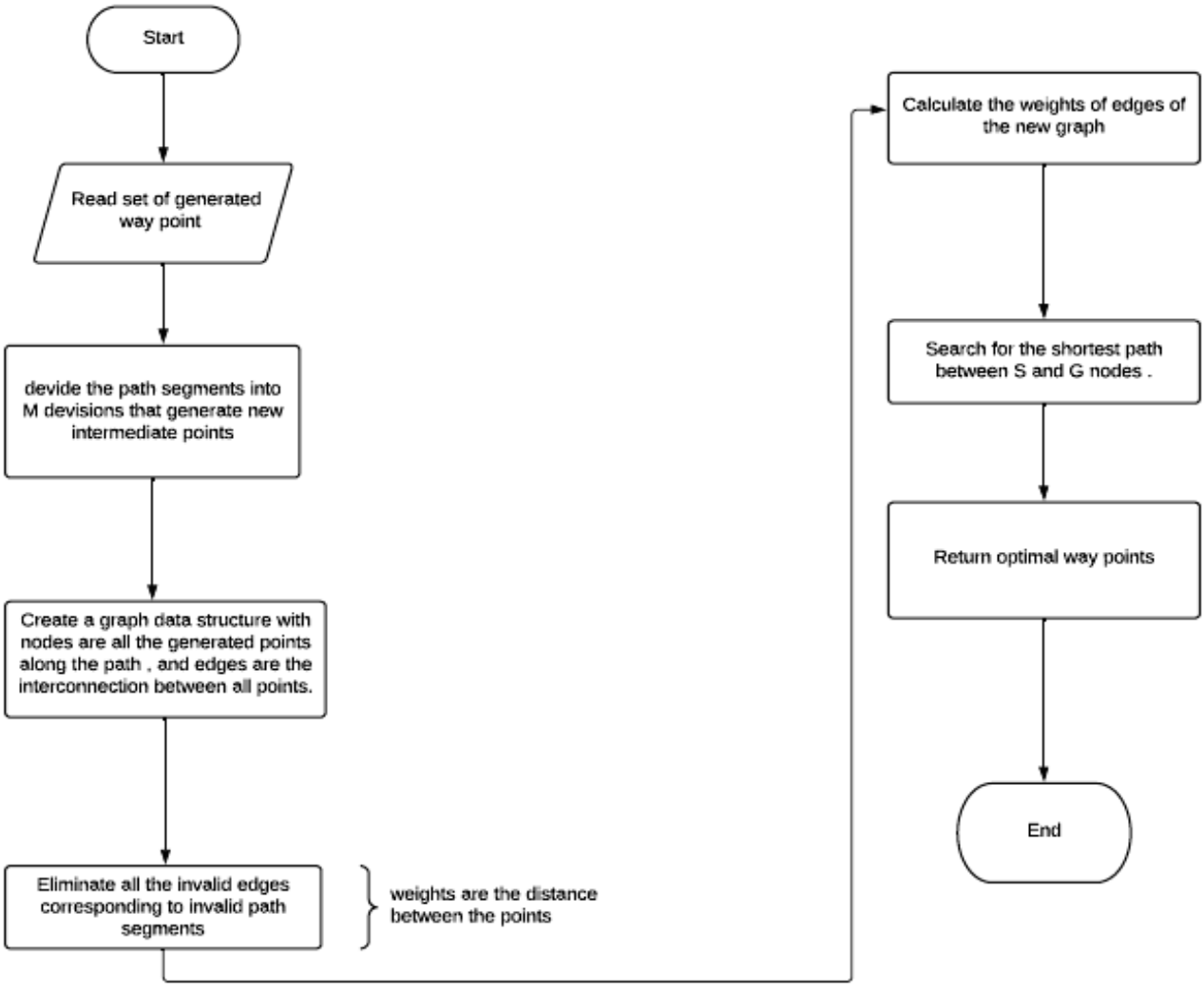


Figure 2.19 Graph based path optimization flowchart.

2.5 Path smoothing

Many path planning algorithms generate a path that is represented as set of waypoints (or control points) which has many sharp turns at those points. Such paths are not fit for mobile robot as it has to slow down at these sharp turns in which this action may not be suitable for robot's kinematics or not desired in a certain application. These robots could be carrying delicate, dangerous, or precious items or even passengers. On the contrary, smooth trajectories are often desired for robot motion and must be generated while planning the robot's path.

2.5.1 Path continuity

The smoothness of a path is generally expressed in terms of continuity. Continuity is of two types: (a) Geometric continuity (G^i), and (b) Parametric continuity (C^i). Geometric continuity ensures that the endpoints of the various segments of the path meet, and the tangent vector's directions are equal. Parametric continuity ensure that the endpoints of the various path segments meet, and tangent vector's direction and magnitudes both are equal. Roughly speaking, parametric continuity (C^i) implies geometric continuity (G^i), but not vice-versa. In general, a curve s has C^n continuity if it's n^{th} derivative $\frac{d^n s}{dt^n}$ is also continuous [41] as shown in figure 2.20.

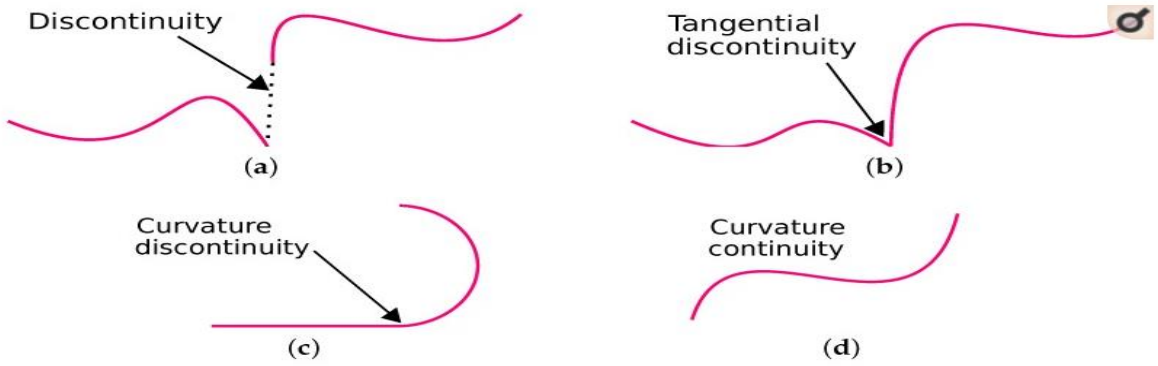


Figure 2.20 Parametric continuity. (a) Discontinuous curve segments. (b) C^0 continuity. (c) C^1 continuity. (d) C^2 continuity[41].

2.5.2 Cubic splines

A spline is a piecewise polynomial function that can have a locally very simple form, yet at the same time be globally flexible and smooth. Splines are very useful for modeling arbitrary functions, and are used extensively in computer graphics. In interpolating problems, spline interpolation is often preferred to polynomial interpolation [41].

The two important characteristics of interpolator cubic splines are: (1) they are the splines of minimum degree that yields C^2 approximations; and (2) they are sufficiently smooth in the presence of small curvatures. Let us thus consider in [a , b] "n+1" ordered nodes $a=x_0<\cdots< x_n = b$

and the corresponding evaluations f_i $i = 0,\cdots, n$. The aim is to provide an efficient procedure for constructing the cubic spline interpolating those values. Since the spline is of degree 3, its second-order derivative must be continuous. By introducing the following notation:

$f_i = s_3(x_i), m_i = s'_3(x_i), M_i = s''_3(x_i)$. i = 0 1 ... n the cubic spline interpolation is given by,

$$s_{3,i-1} = M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x_i - x)^3}{6h_i} + C_{i-1}(x - x_{i-1}) + \hat{C}_{i-1} \quad (2.8)$$

Where $h_i = x_i - x_{i-1}$ i = 0 1 ... n

$$\hat{C}_{i-1} = f_{i-1} - M_{i-1}\frac{{h_i}^2}{6} \quad , C_{i-1} = \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(M_i - M_{i-1}) \quad (2.9)$$

Mi can be calculated from the M continuity system

$$u_i M_{i-1} + 2M_i + \lambda_i M_{i+1} = d_i \qquad i = 0\ 1 \ldots n \quad (2.10)$$

Using the cubic splines to smooth the path generated bay RAY planner yields the results shown in figures 2.21 and 2.22.

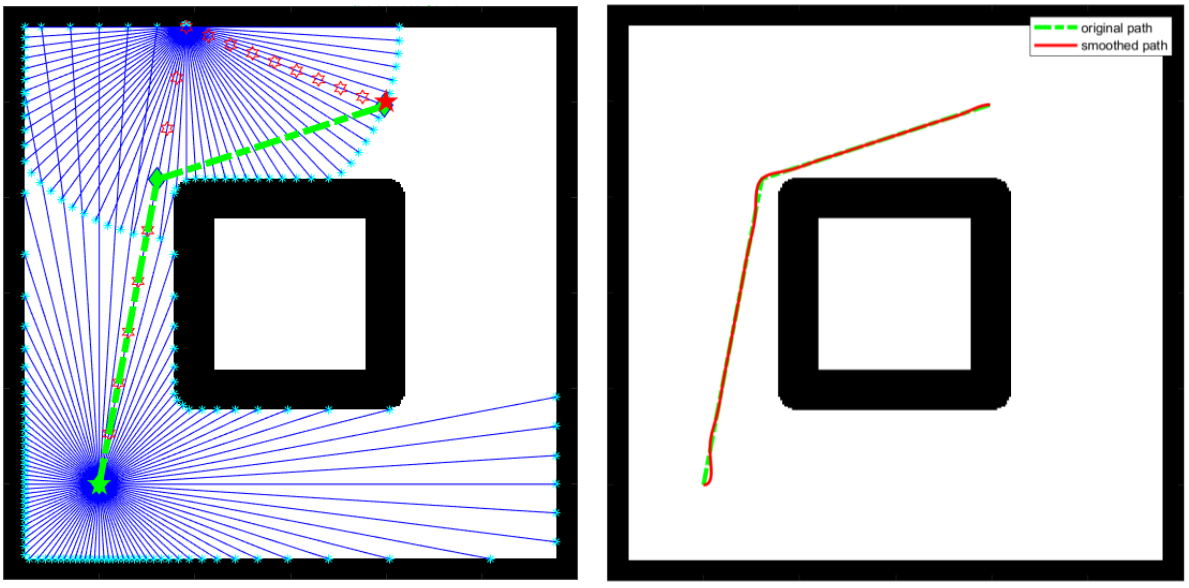


Figure 2.21 On the left: the planned path. On the right: the smoothed path.

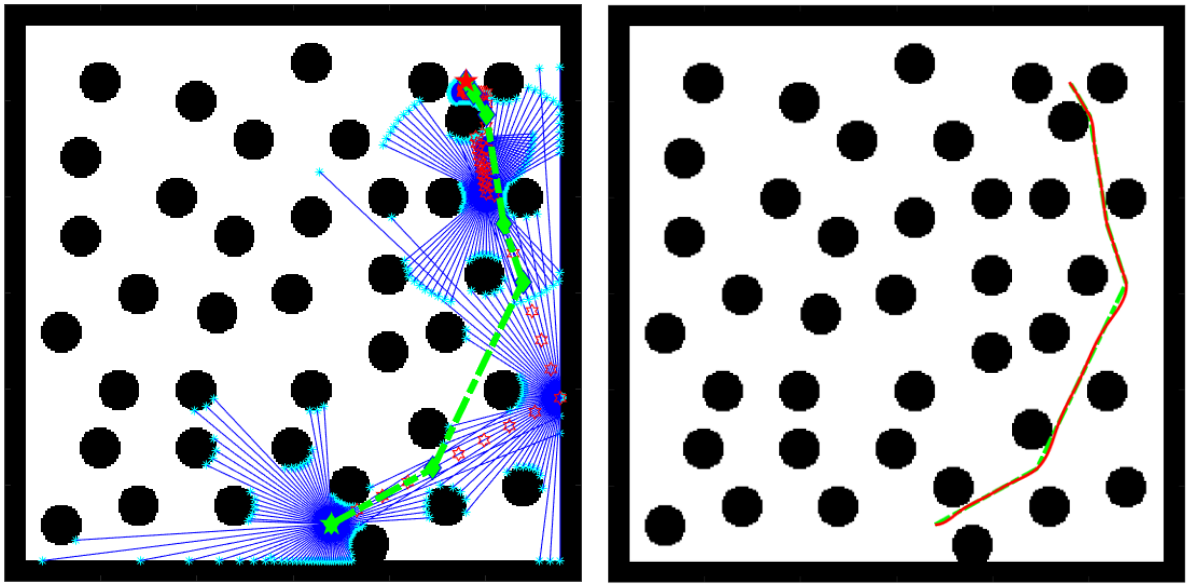


Figure 2.22 On the left: the planned path. On the: right the smoothed path.

2.6 Conclusion

Along this chapter, we have covered how to build a binary occupancy map starting from raw data given by the lidar, and proposed some environments of different complexities to test the proposed path planning algorithm. Second, RAY path planning algorithm is explained and how this approach search for a feasible path starting from any arbitrary initial position to the specified goal, then optimal path extraction from the search feasible path using graph theory is implemented. Last, we have seen that smooth paths are highly desirable for mobile robots and how piece-wise paths may introduce implications for the kinematics of the robot. Thus, path smoothing using cubic splines were implemented and results in C^2 continuous path.

Chapter III

Trajectory tracking and path re-planning

In chapter 2, the proposed path planning approach (Ray based planner) was discussed. However, the path generated must be followed by the robot in order to reach the goal starting from initial position, where motion planner is needed. Motion planning is the process by which we define the set of actions needed to execute and follow the planned path. A differential drive robot model is considered.

3.1 Differential drive and inverse kinematics model

Motion models that describe robot kinematics are interested in mathematics of robot motion without considering its causes, such as forces or torques. Kinematic model describes geometric relationships that are present in the system.

Differential drive is a very simple driving mechanism that is quite often used in practice, especially for smaller mobile robots. Robots with this drive mechanism usually have one or more castor wheels to support the vehicle and prevent tilting. Both main wheels are placed on a common axis. The velocity of each wheel is controlled by a separate motor. The input (control) variables are the velocity of the right wheel $V_R(t)$ and the velocity of the left wheel $V_L(t)$. As shown in figure 3.1 the associated quantities and variables are; r is the wheel radius; L is the distance between the wheels and R(t) is the instantaneous radius of the vehicle driving trajectory (the distance between the vehicle center (middle point between the wheels) and the Instantaneous Center of Rotation point (ICR). In each instance of time both wheels have the same angular velocity ω(t) around the ICR.

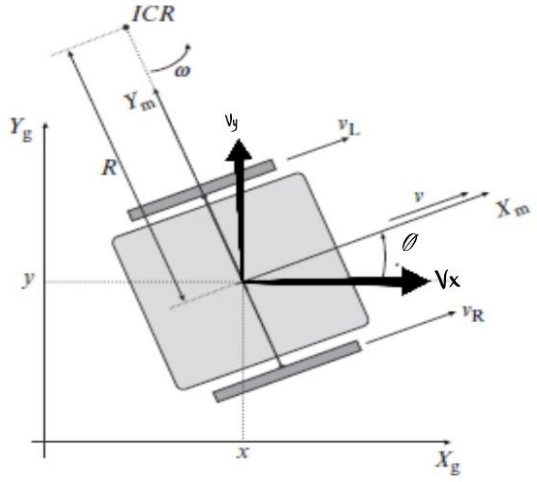


Figure 3.1 Differential drive kinematics [36].

These set of motion equations are used to derive the robot forward and inverse kinematics [36].

$$\omega_L=\omega = \frac{VL(t)}{R(t)-\frac{L}{2}} \tag{3.1}$$

$$\omega_R=\omega = \frac{VL(t)}{R(t)+\frac{L}{2}} \tag{3.2}$$

from where ω(t) and R(t) are expressed as follows:

$$\omega(t) = \frac{VR(t)-VL(t)}{L} \tag{3.3}$$

$$R(t) = \frac{L*VR(t)+VL(t)}{2VR(t)-VL(t)} \tag{3.4}$$

Tangential vehicle velocity is then calculated as :

$$V(t)= \omega(t)\times R(t)= \frac{VR(t)+VL(t)}{2} \tag{3.5}$$

Since the robot model will be simulated in accordance with the pure pursuit controller which gives the robot angular and tangential velocities as required outputs to follow the path; the set of presented equations (eq3.6 and eq3.7) can be used to derive the required wheels tangential velocities to give the robot the specified tangential and angular velocities as:

$$\omega_{R/L}=(1\pm \frac{L}{2R(t)})\times \frac{V}{r} \tag{3.6}$$

$$V_{R/L}= r\times\omega_{R/L} \tag{3.7}$$

where The notation of "+" and "-" inside the bracket indicate the direction of robot turning. The robot will turn left if the notation of ω_R and ω_L are "+" and "-" respectively.

3.2 pure pursuit controller

The pure pursuit algorithm was originally devised as a method for calculating the arc necessary to get a robot back onto a path. This first application of the method came with the Terragator, a six wheeled skid steered robot that was used for outdoor vision experimentation in the early 80's. The standard references for the original derivations of the wok go to Wallace [42], and found it to show the greatest promise as a general purpose tracking algorithm.

The pure pursuit approach is a method of geometrically determining the curvature that will drive the vehicle to a chosen path point, termed the goal point. This goal point is a point on the path that is one look ahead distance from the current vehicle position. An arc that joins the current point and the goal point is constructed. The chord length of this arc is the lookahead distance [43]. The point (x , y) is constrained to be on the path. The objective is to calculate the curvature of the arc that joins the origin to (x , y) and whose chord length is "L". As shown in Figure 3.2, the following two equations (eq3.8) hold. The first is from the geometry of the smaller right triangle and the second from the summing of line segments on the x axis.

$$\begin{cases} x^2 + y^2 = L^2 \\ x + d = r \end{cases} \tag{3.8}$$

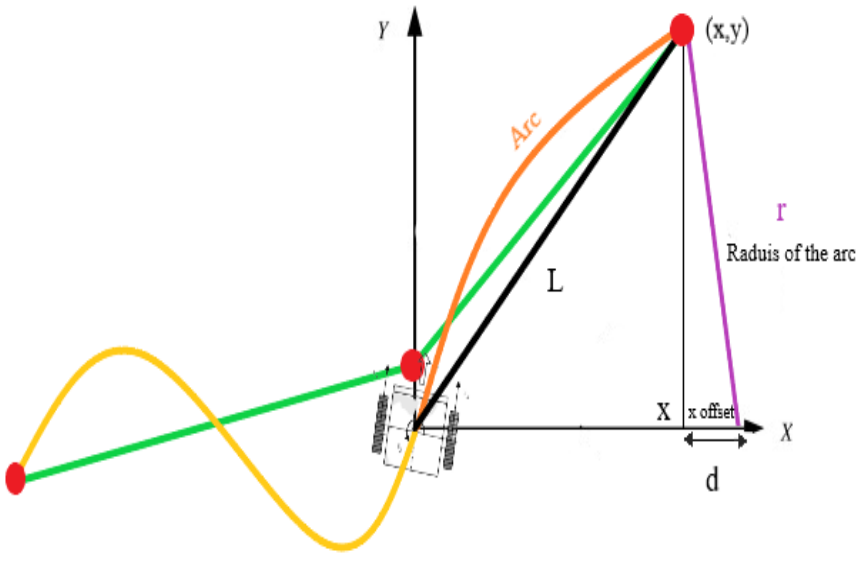


Figure 3.2 Geometric relations of pure pursuit algorithm.

The next series of equations relate the curvature of the arc to the lookahead distance.

$$\begin{cases} d = r - x \\ (r-x)^2 + y^2 = r^2 \\ r^2 - 2rx + x^2 + y^2 = r^2 \\ 2rx = L^2 \\ r = \frac{L^2}{2x} \\ \gamma = \frac{2x}{L^2} \end{cases} \tag{3.9}$$

The implementation of the pure pursuit algorithm itself is fairly straightforward. The pure pursuit algorithm can be outlined as follows:

- Determine the current location of the vehicle.
- Find the path point closest to the vehicle.
- Find the goal point.
- Transform the goal point to vehicle coordinates.
- Calculate the curvature and request the vehicle to set the steering to that curvature.
- Update the vehicle's position.
- Repeat the process.

The look ahead distance property is the main tuning property for the controller. The look ahead distance is how far along the path the robot should look from the current location to compute the angular velocity commands. The effect of changing this parameter can change how your robot tracks the path and there are two major goals: regaining the path and maintaining the path. In order to quickly regain the path between waypoints, a small Look ahead distance will cause the robot to move quickly towards the path. However, as can be seen in figure 3.3 below, the robot overshoots the path and oscillates along the desired path. In order to reduce the oscillations along the path, a larger look ahead distance can be chosen. However, it might result in larger curvatures near the corners [44].

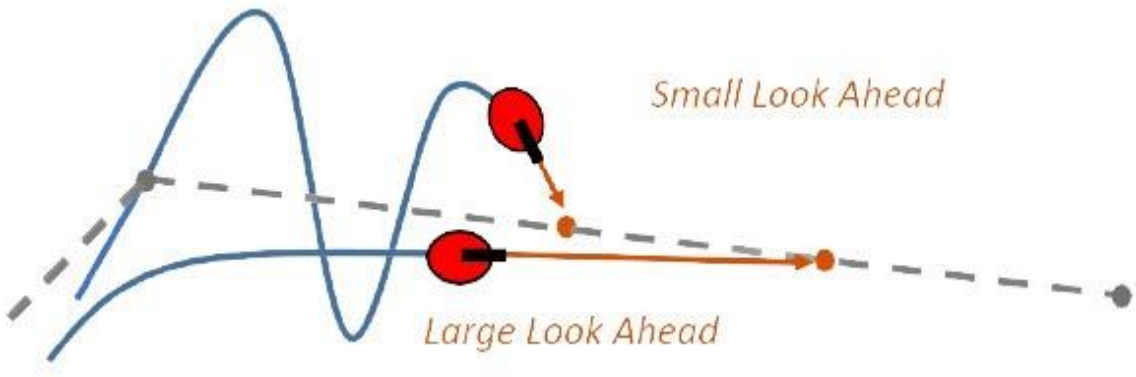


Figure 3.3 The effect of small and long lock ahead distances.

3.3 Simulation

Now the whole system is simulated using MATLAB Simulink, in addition to the controller and the robot model, we added a lidar sensor model in order to detect new encountered obstacles in the generated path that were not present in the map and respond by re-planning a new path avoiding the encountered obstacle to the goal. In addition a slowing down condition near goal to provide a smooth stop at goal by introducing a gain factor given by eq3.10 to the velocities generated by the controller when the robot reaches the goal at a distance "ε".

$$\text{Gain} = \frac{\sqrt{(Gx-Px)2+(Gy-Py)2}}{\varepsilon} \tag{3.10}$$

The process is presented using the flowchart in figure 3.4 and the corresponding Simulink model is presented in figure 3.5.

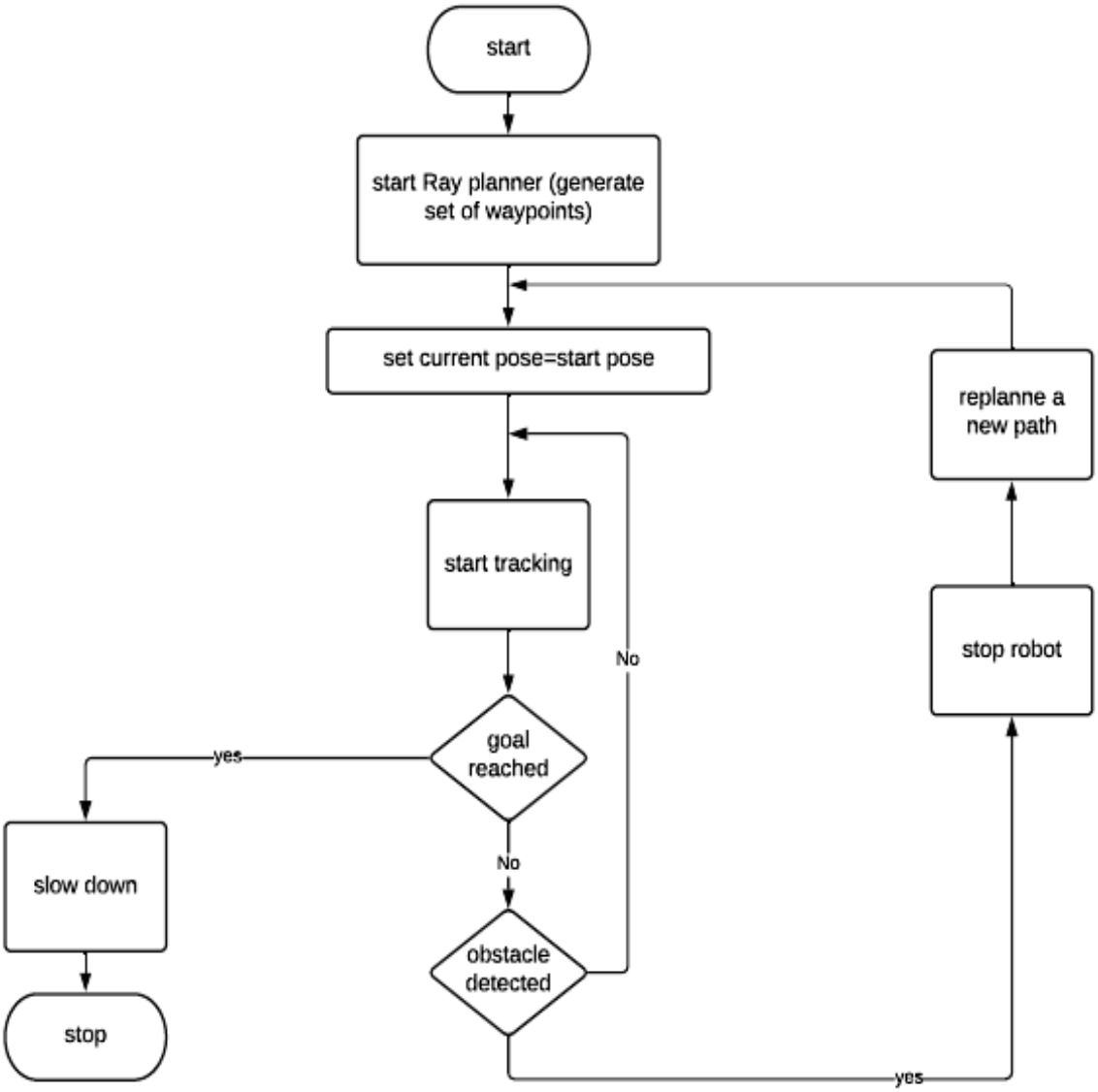


Figure 3.4 Flowchart of path tracking and re-planning process.

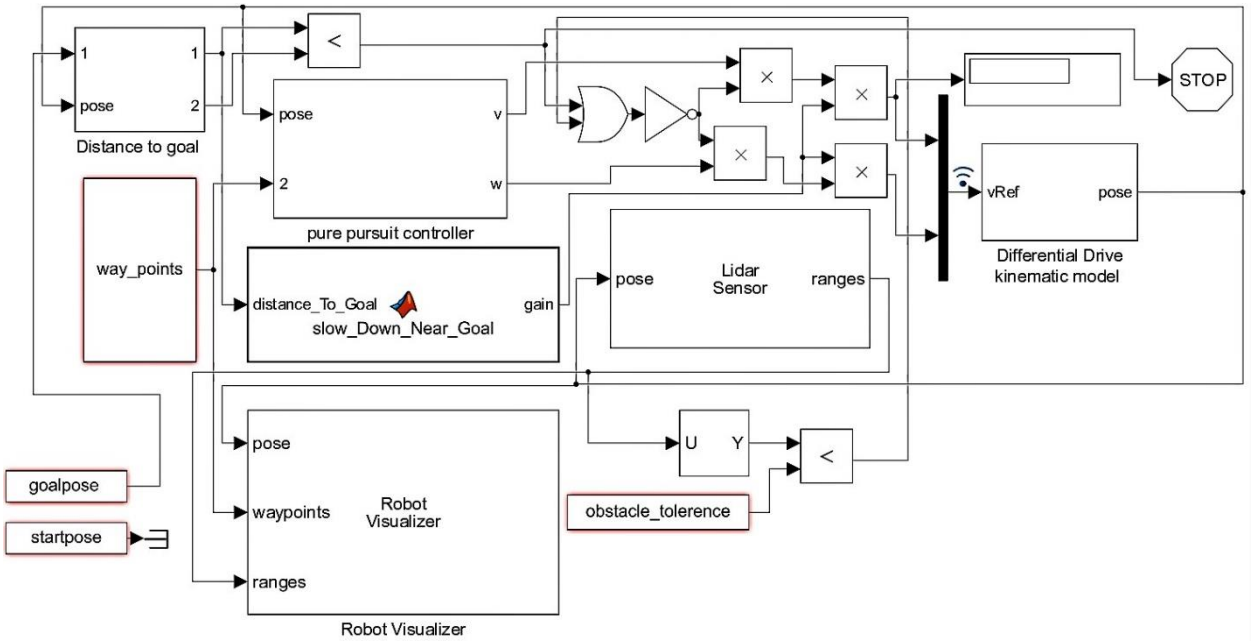
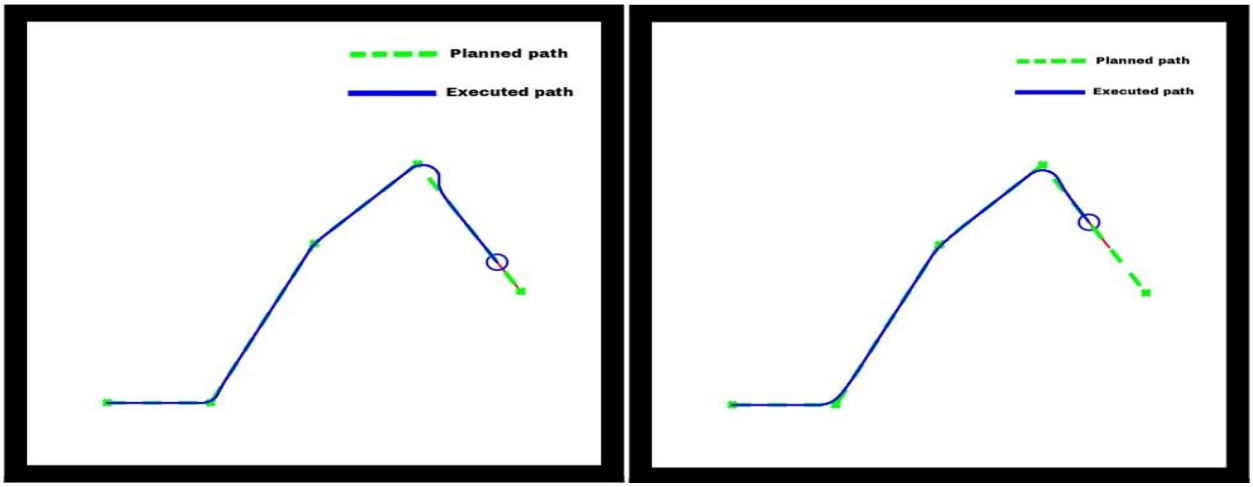


Figure 3.5 Simulink model of the path tracking and path re-planning process.

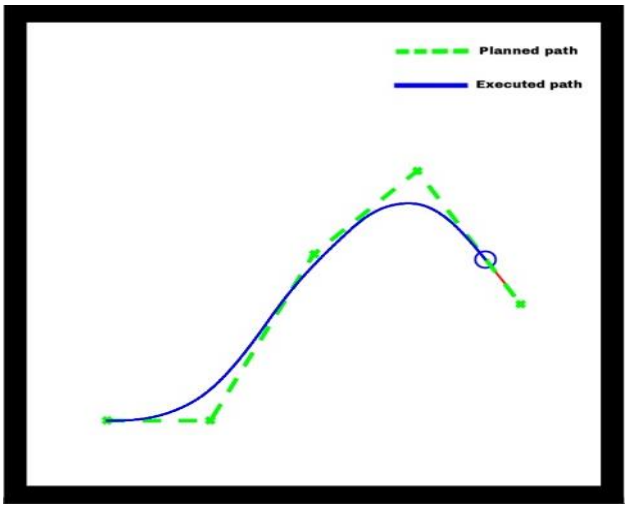
Here is a brief explanation for the functions of the main Simulink model blocks:

- Way_points: hold the path generated by RAY path planner as N×2 array.
- Pure pursuit controller block: accepts as inputs the planned path from way_points block and the current position of the robot generated by the forward kinematic model of the differential drive robot. It uses eqs3.8 and 3.9 and the designated lookahead distance to calculate the required arc parameters to reach the lookahead point from the current position. The outputs of this block are the linear and angular velocities required to follow the calculated arc. An example illustrating the affect of lookahead distance on the executed path is shown in figure 3.6 below.
- Differential drive kinematic model: this block uses the velocity commands from the controller to calculate the future position of the robot starting from current location.
- Lidar sensor: this model simulates a range sensor to detect object within its range, the block has single input which is the robot location and its outputs are distances to objects which are used to detect objects in the robot's path.
- Distance to goal: this block provide the instantaneous distance between robot location and the goal.

- Slow down near goal: this block modulates the robot velocity when it is within a distance $\varepsilon = 0.3$ meter.
- Robot visualizer: this is the block responsible for plotting and visualizing the robot.



(a) (b)



(c)

Figure 3.6 The effect of lookahead distance on executed path, (a) L=0.05m,(b) L=0.1m,(c) L=0.5m.

3.4 Conclusion

In this chapter, we have presented motion model for differential drive robot. This model describes how the robot behaves in respond to the inputs. In addition, we have seen that pure pursuit controller is fairly simple algorithm for path following yet, a very effective one if its main parameter (lookahead distance) is tuned properly. Simulation results are presented in chapter 4.

Chapter IV

Results evaluation and validation

In this chapter, simulation results of the RAY path planner and the pure pursuit tracker and path re-planning are demonstrated. In addition, for performance comparison of the obtained results with some of the widely known and used planners (PRM and RRT*) are discussed. This chapter is organized as:

In the first section, the effect of some planner parameters are discussed; then, the results of path planning using the proposed approach are demonstrated for different scenarios in several maps configurations with different types of obstacles and shapes. The next section is to be specified to demonstrate the path tracking algorithm simulation results. In addition, a local planning is done to avoid new obstacles in the pre-planned path.

After that, the comparison of paths generated using RAY planner and other algorithms are demonstrated, the criteria are; (1) the length of the generated path, because it is a critical for any path planning algorithm as the shortest path is always desirable. (2) the number of control points contained in the generated path, because this criterion will affect the performance of the path smoother algorithm if the generated path is to be smoothed, so a minimum number of way-points in the path is desirable also. (3) The execution time of the planning process, this time must be small to be practically applicable. Last the obtained results are discussed; in addition, some critical scenarios observed regarding the RAY planner algorithm are highlighted.

All the computations were performed on a laptop with a fourth generation Intel Core i3 processor clocked at 1.7 GHz equipped with 8GB of RAM.

4.1 Obtained results

The proposed path planning algorithm (Ray based planner) as was discussed in chapter 2 uses rays to find the goal; then, it uses graph theory to extract the optimal path by the help of the introduced intermediate points. These two parameters (number of lunched rays R and number of intermediate points N) have different effects on the length of the generated path and the execution time of the planning process. Before passing to the obtained results these two parameters must be investigated. For this purpose, ray planner was used to plan a path in the map illustrated in chapter 2 figure 2.7, with starting and goal locations are (5, 0.5) and (5.1, 6.5) respectively. The different parameters values that were tested and the obtained results are

demonstrated in table 4.1. The visualization of the different cases is illustrated bellow in figures 4.1 and 4.2.

Criteria / Rays	Path length (meters)	Execution time (seconds)
Rays = 20	7.52	13.47
Rays = 50	6.57	7.69
Rays = 100	6.55	2.01
Rays = 200	6.52	1.98

(a)

Criteria / points	Path length (meters)	Execution time (seconds)
N=3	6.77	0.17
N=6	6.61	0.65
N=12	6.56	2.89
N=24	6.50	12.56

(b)

Table 4.1 (a) The effect of changing number of rays on generated path length and execution time of the planner**. (b)** The effect of changing the number of intermediate points on path length and planner execution time.

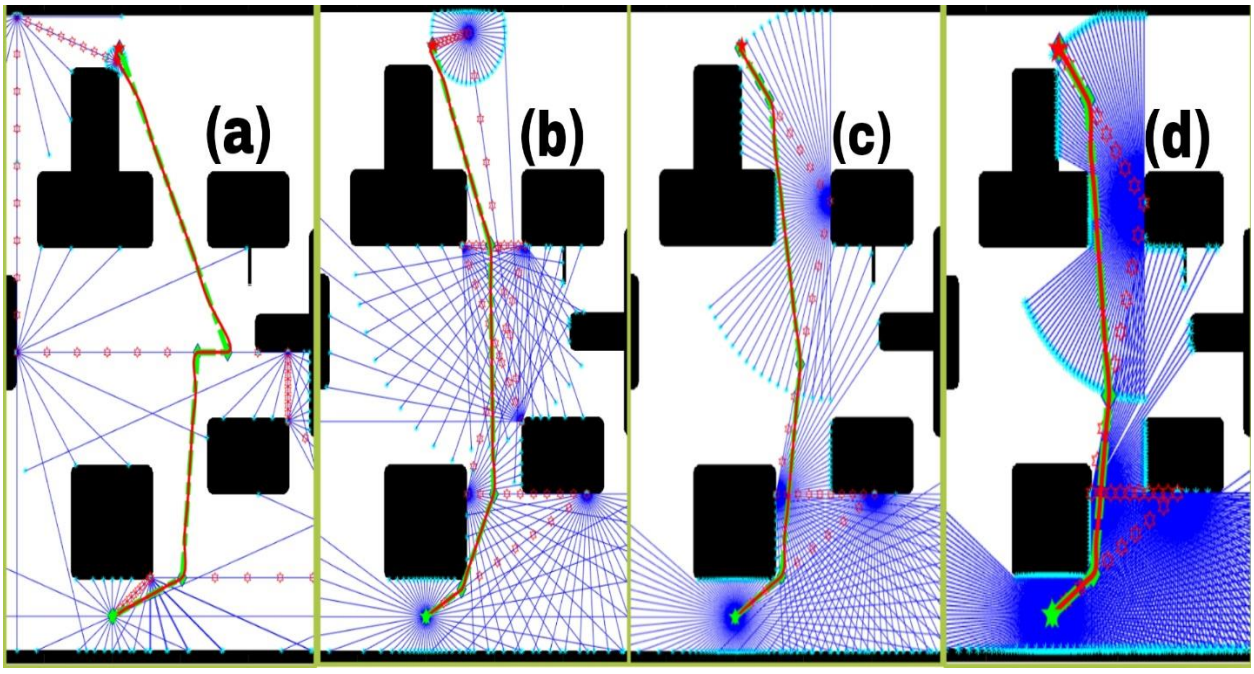


Figure 4.1 The effect of changing the number of rays on the planned path. (a) 20 rays,(b) 50 rays,(c) 100 rays and (d) 200 rays. With the same number of intermediate points N=10.

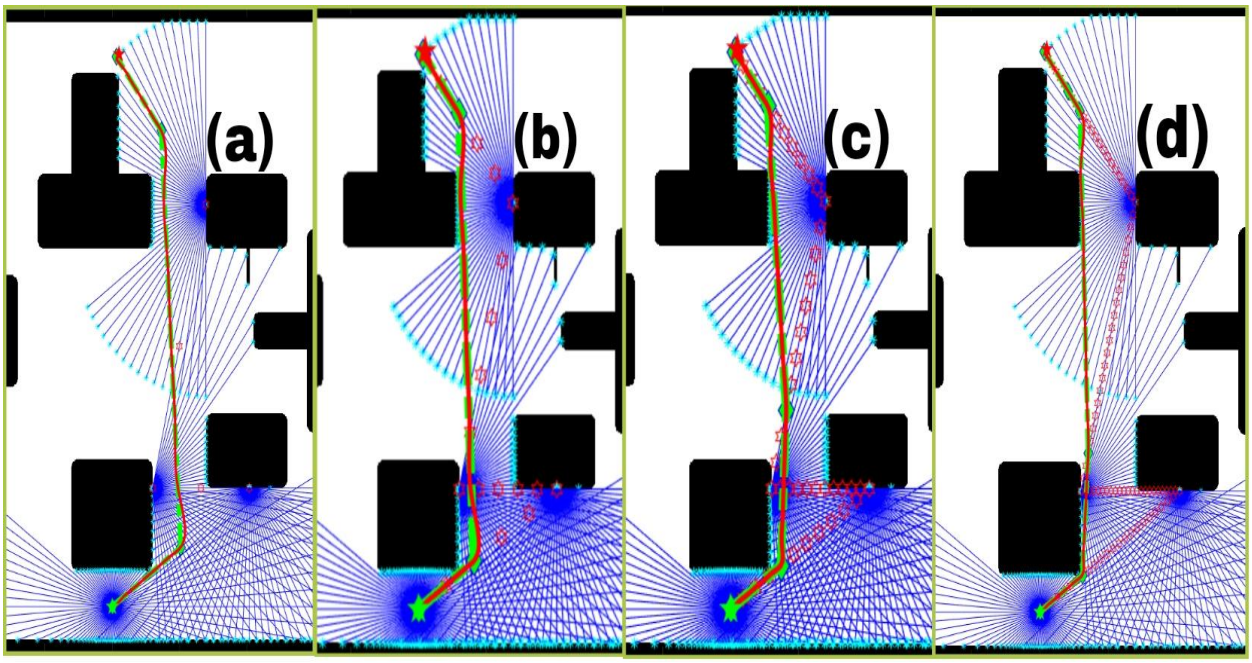


Figure 4.2 The effect of changing the number of intermediate points on the planned path. (a) 3 points,(b) 6 points ,(c) 12 points and (d) 24 points. With the same number of rays R=100.

From the obtained results, we can see that increasing the number of rays reduces the length of the path approximately one meter going from 20 to 100 rays, but no tangible additional improvements if we increase the number any further as illustrated in figure 4.1, in the other hand we observe that the running time has decreased dramatically from 13.47 seconds with 20 rays to 2.01 seconds using 100 rays.

For the effect of N (number of points), by increasing the number of points we do not get large improvements in the path length. Instead, it has a refining effect on the path as shown in figure 4.2, but we see that the execution time increase rapidly after N=6.

For the next section, we set these two parameters to R=100 rays and N=6 points throught the simulation, because at these two values we can compromise between generating short path with small running time.

4.1.1 Path planning

The path planning simulation is conducted in 2D maps and are presented in figures 2.7 2.8 2.9. In addition, to a simple map shown in figure 4.3 that contains only the borders of the

map. The simulation results are displayed in a simple N×M (meters) discretized environment with resolution of 100 grids/meter. The work space obstacles are illustrated in dark color, as for the robot size is treated as point in the configuration space that occupies a single grid cell in the map.

Through the displayed results the green and red stars represent the start and goal positions respectively, the green dashed lines represents the optimal generated path and the continuous red line is the smoothed path. As for the planner parameters we set the number of initiated rays to 100, number of intermediate point to 6 and the goal tolerance to 0.05 meters. The results are demonstrated bellow in figures 4.3, 4.4, 4.5 and 4.6.

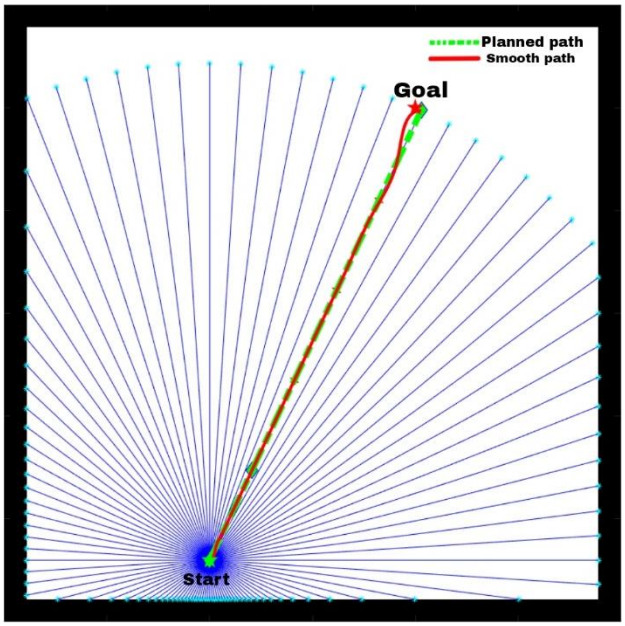


Figure 4.3 RAY Path planning in free map. The map is 3×3 meters with no obstacles the start And goal position are (1 , 0.3) and (2 , 2.5) in meters respectively.

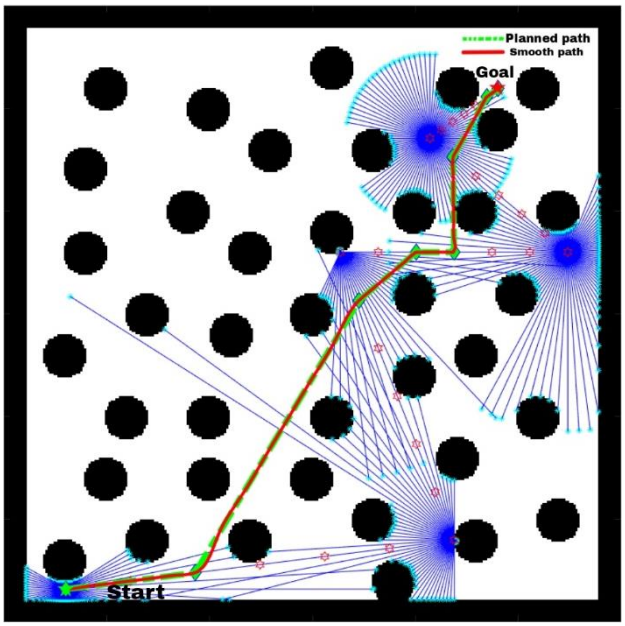


Figure 4.4 RAY Path planning in a cluttered map, with size 3×3 meters with 36 randomly placed obstacles of radius 0.1 meter, the start and goal position are (0.3 , 0.16) and (2.4 , 2.6) respectively.

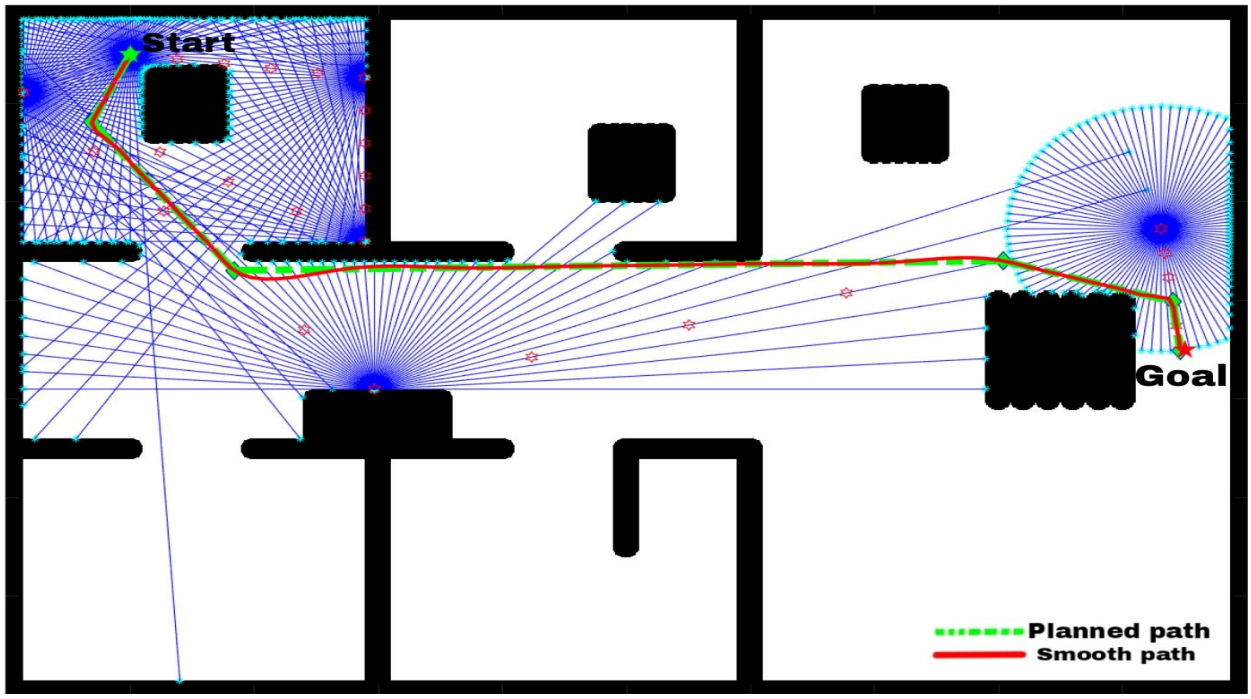


Figure 4.5 RAY Path planning in a home shaped environment with 5 squares obstacles. The map size is 10×7 meters, the start and goal positions are (1 , 6.5) and (9.5 , 3.5) in meters respectively.

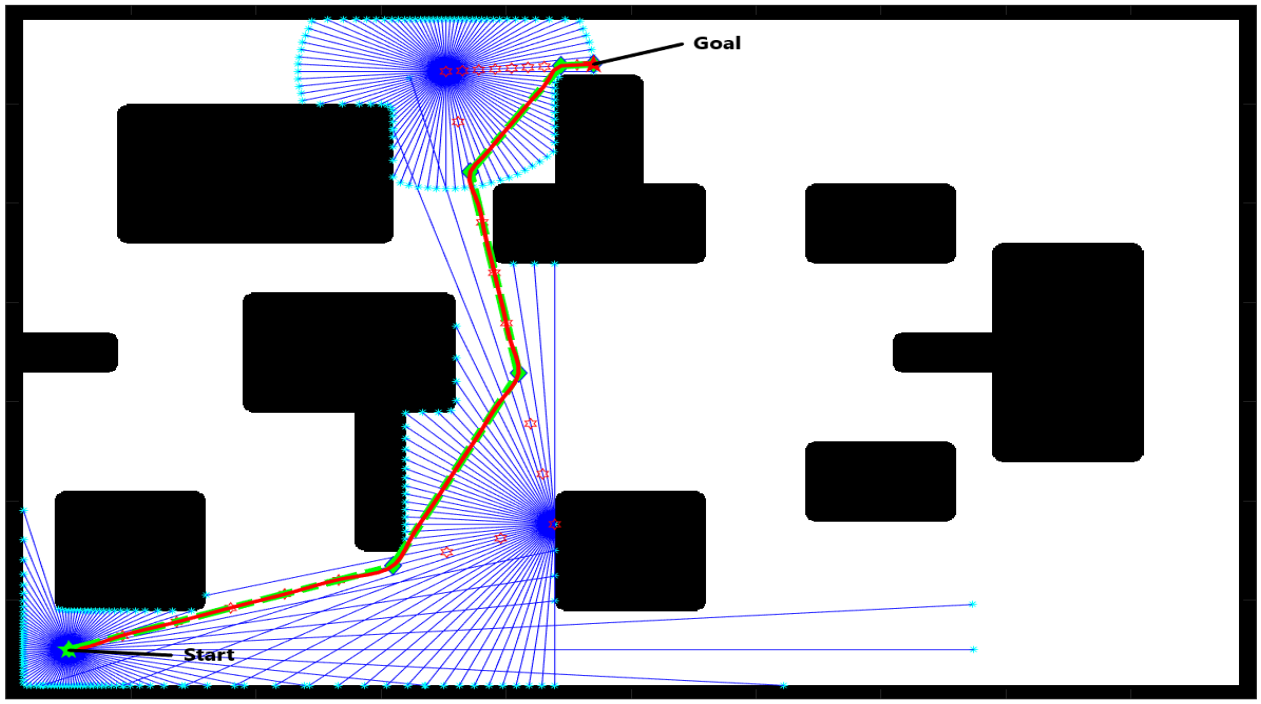


Figure 4.6 RAY Path planning in a map with 9 polygonal shaped obstacles of different sizes and configurations. The map size is 10×7 meters, the start and goal positions are (0.5 , 0.5) and (4.7 , 6.4) in meters respectively.

The proposed approach finds the goal in different environments and different start and goal locations effectively as was illustrated in figures 4.3, 4.4, 4.5 and 4.6. Starting by the empty map we see that the goal is reached within a single step and the Ray planner yields the shortest path possible because the search starts with radius equals to distance between the start and goal positions.

For the results in figures 4.4 4.5 4.6 the proposed algorithm find the goal always with optimal or sub-optimal path. The search start by lunching rays from start points then choose the closest point to the goal and repeat the process. If the chosen point is not repeated then the planner follows simple steps as in case of scenarios illustrated in figures 4.4 and 4.6 till it reaches the goal within the specified tolerance.

For the case where the chosen point is repeated, as in the third step of the scenario described by figure 4.5. The planner takes a random point from the current set of point instead of the closest one to the goal and lunches the rays. This random step prevents the algorithm from getting stuck at the corner and continues the search.

The search is always restricted in small areas depending on the map complexity and how far the goal is. This is another advantage because the search radius is modulated over time depending on how far is the goal; so, if the current point has a visual contact with the goal, the goal can be reached within the next step of search as shown in the case of empty environment in figure 4.3. In all other cases the search terminates once the planner has visual contact with the goal.

After the ray planner reaches the goal position in the search phase, it starts the extraction of the optimal path using the approach discussed in chapter 2. The intermediate points are illustrated as small red stars; whereas, the number of new intermediate points is directly proportional to the number of random steps in the planning process. This will increase the execution time of the algorithm as it was demonstrated in table 4.1(b).

The final path is smoothed based on the approach presented previously in chapter 2 (cubic splines smoother). We see that the smoothed path follows exactly the original path in straight lines segments without any oscillations, and at the control points the path is well curved. The smoother does not add much additional distance to the path.

4.1.2 Path tracking and local planning

Path re-planning in case of encountering new obstacle while tracking the pre-planned path is simulated using the Simulink block illustrated in chapter 3 figure 3.5 under the same workspaces used for path planning tests. For the robot kinematics we set the track width to (L=0.2 meter) and wheels radius is (r =3.5 cm). The controller lookahead distance is set to (d = 0.05 meter). The simulation results are illustrated in figures 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12 bellow. The added lidar model has 90 degrees field of view in front the robot with scanning range of 1 meter which is used to detect new obstacles in the robot's path. The first set of images shows how the robot create a new path to reached the same goal, while the second set (the paths in blue color) illustrates the corresponding path generated by the tracker.

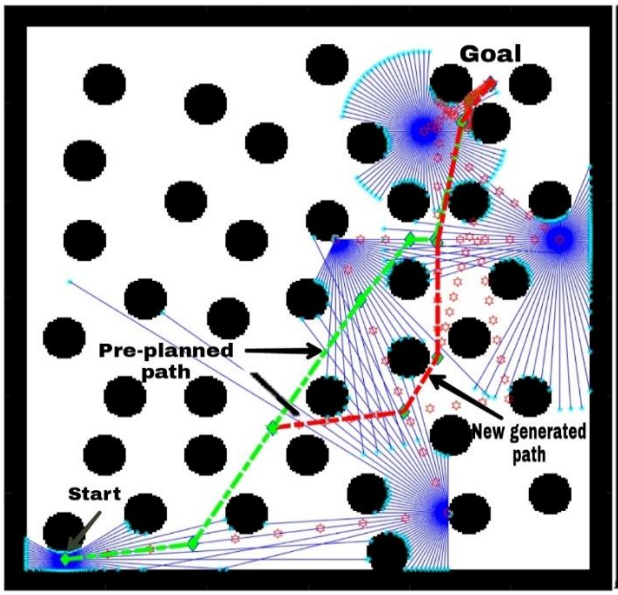


Figure 4.7 Path re-planning in cluttered map

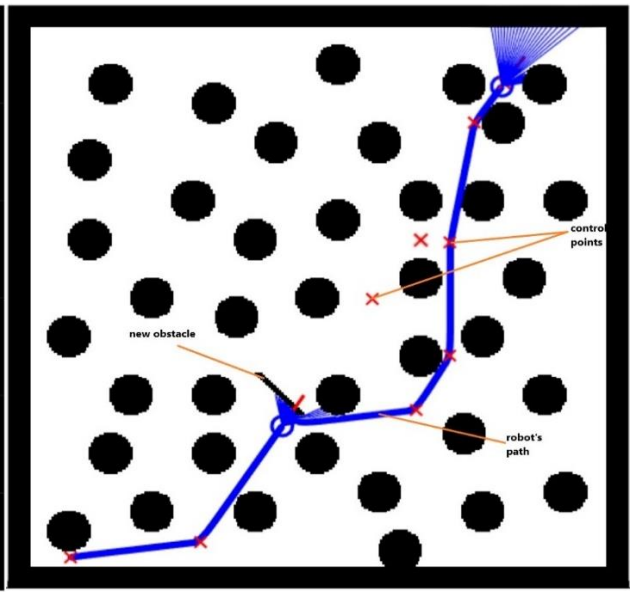


Figure 4.8 Path tracking in cluttered map

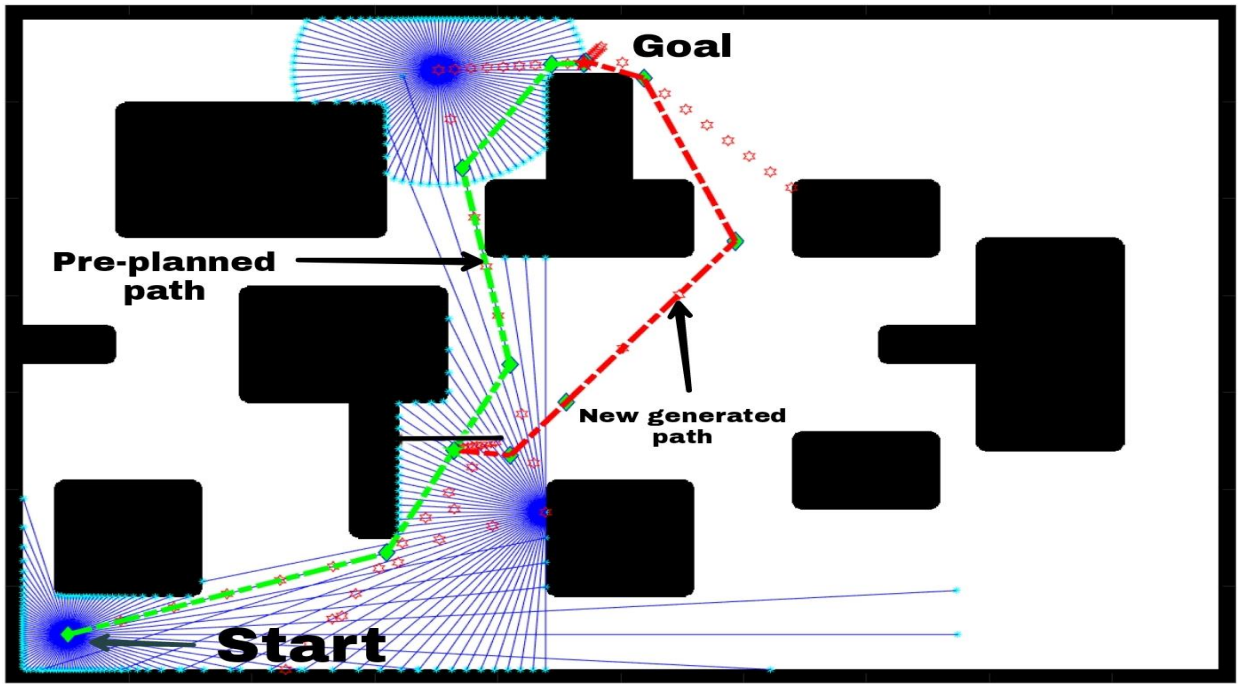


Figure 4.9 Path re-planning in a map with polygonal shaped obstacles.

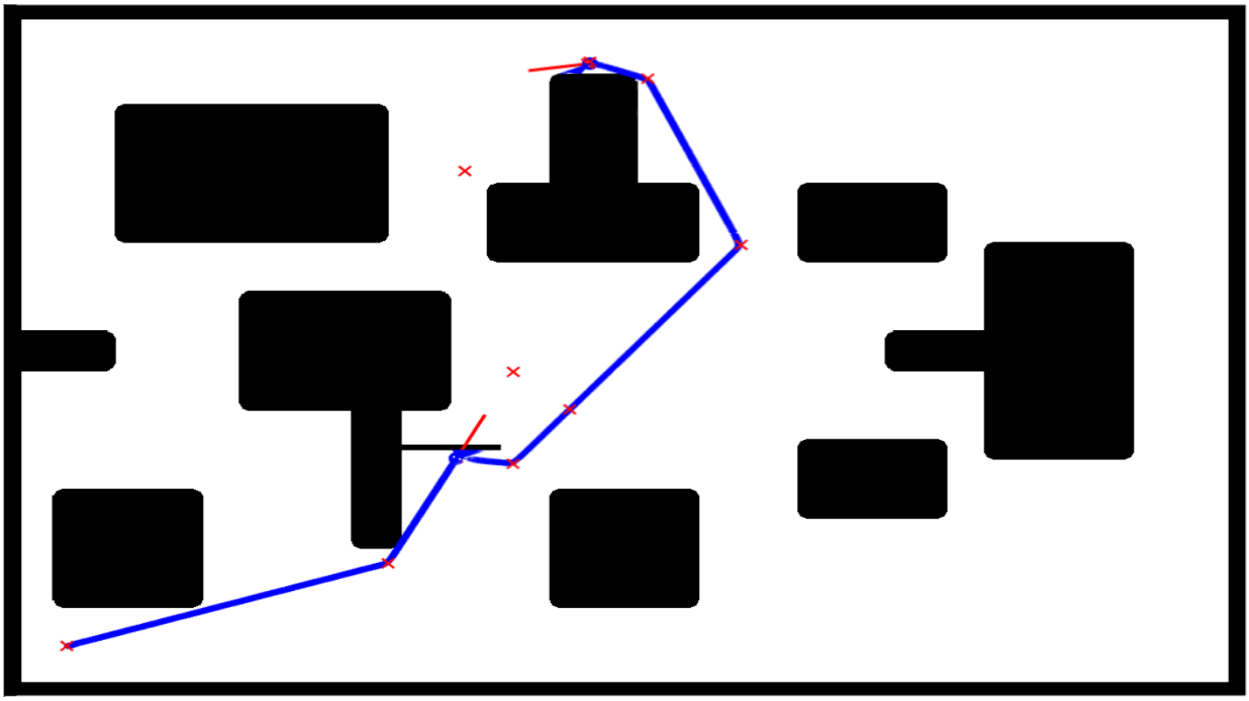


Figure 4.10 Path tracking in a map with polygonal shaped obstacles illustrated in blue color.

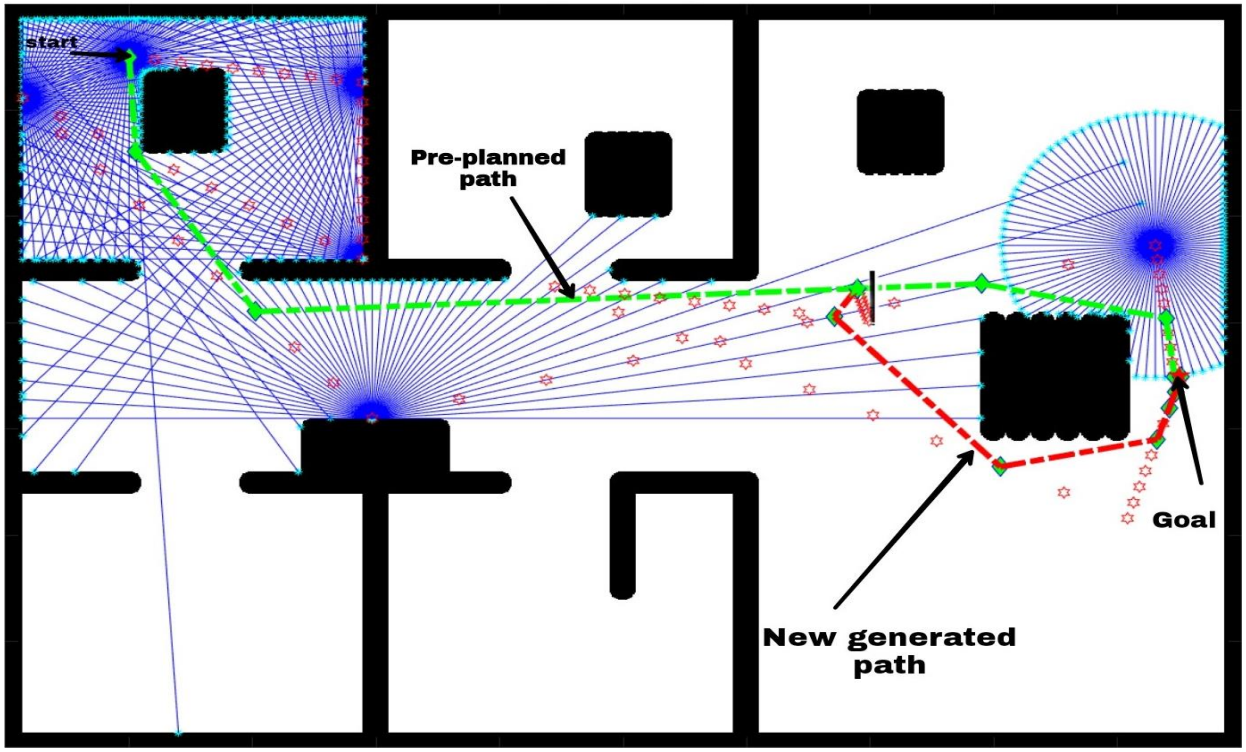


Figure 4.11 Path re-planning in a home shaped map.

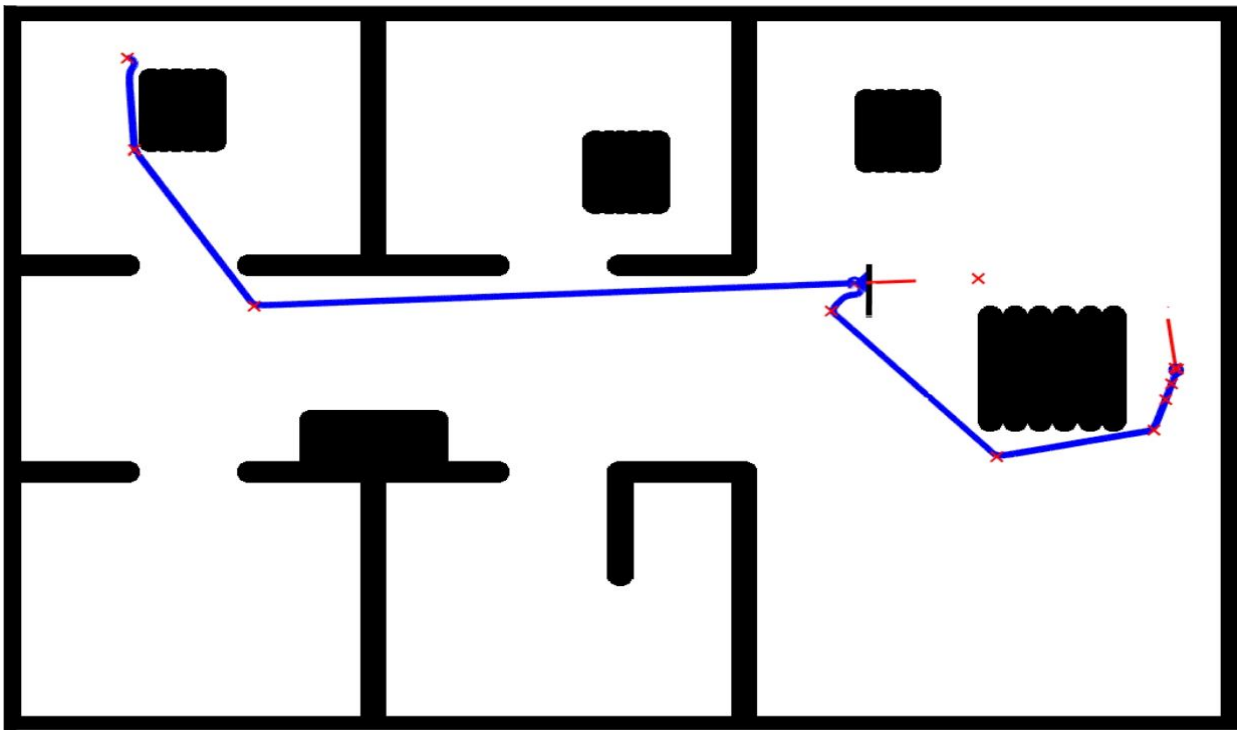


Figure 4.12 Path tracking in a home shaped map illustrated in blue color.

As stated earlier the simulation was conducted in the same environments as for path planning and the maps are the same size with the same earlier defined parameters (dimensions, resolution, number of obstacles and their shapes), in the simulation the robot is visualized as a point with little red line to show the orientation of the robot. After that the optimal path, from start to goal locations, is planned as illustrated in green color in figures 4.7, 4.9 and 4.11. Before the robot starts following the path, a new static obstacle randomly placed on the pre-planned path is added. As the robot starts moving as shown in figures 4.8, 4.10 and 4.12, the lidar will detect if an obstacle is on the path. If one is detected the robot will stop at distance 0.2 meters from the detected obstacle and generates a totally new path from its current position to the goal. The new path is illustrated in red in figures 4.7, 4.9 and 4.11, where we see that the new path is totally re-planned to give an optimal path, sometimes this new path will avoid the new obstacle and come back to initial path at some point as it is the case shown in figure 4.7.

4.2 Results comparison and validation

For comparison purposes, all the previously generated and tested scenarios using our proposed approach were also tested under PRM and RRT-star planners on the same machine; the set of generated paths using PRM and RRT* are illustrated in figures 4.13, 4.14, 4.15 and 4.16 bellow. Where for RRT* results the algorithm is set to perform 7000 iterations so the generated path is optimal, the blue points represent the randomly sampled robot configurations and the short blue lines are the connection between them and the optimal generated paths are highlighted in red. The same for PRM planner we set 100 as maximum points number that the planner samples randomly, the blue points are the randomly sampled points from the free space and thin blue lines represent valid connections between the sampled points, the optimal generated path is illustrated by bold dashed blue line. For PRM planner, since the path planning is a stochastic process, the planning approach was performed three times for each scenario and the average results (from path length prospective) are summarized in table 4.2.

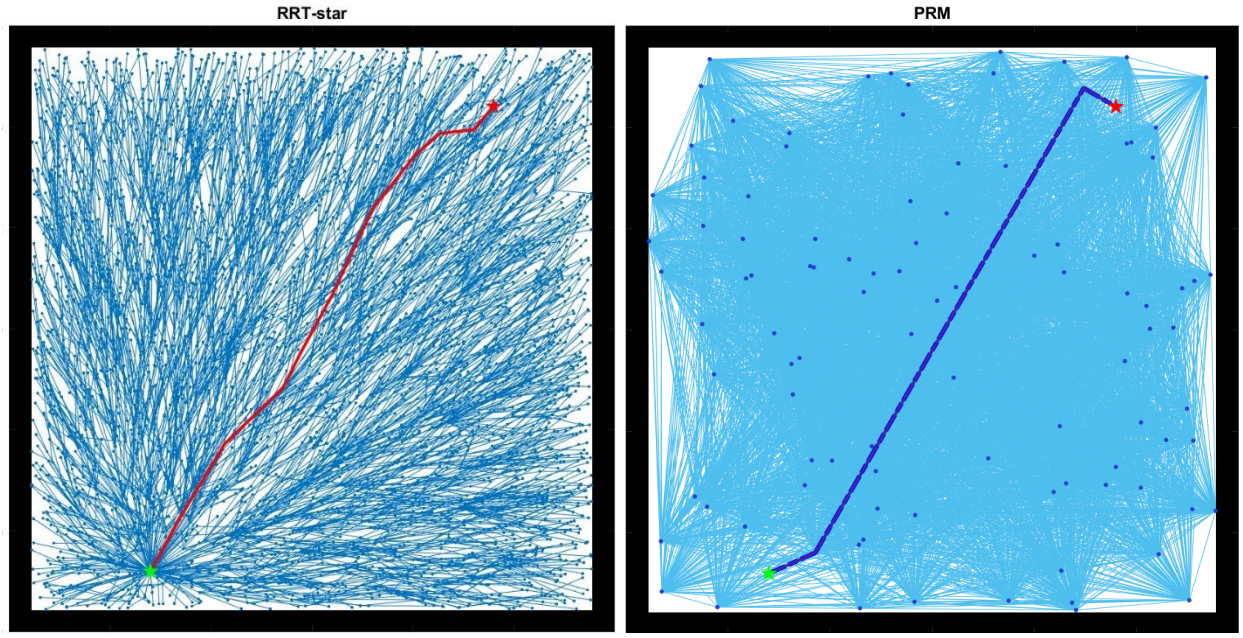


Figure 4.13 RRT* and PRM in an empty map left and right respectively start and goal positions are green and red start respectively.

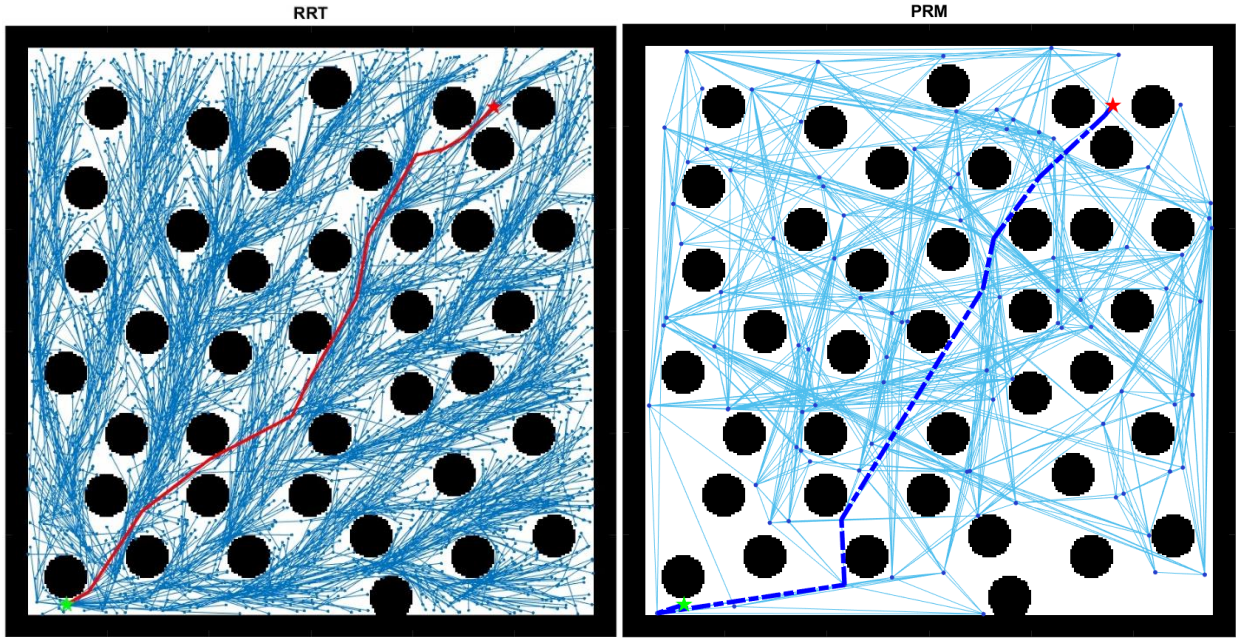


Figure 4.14 RRT* and PRM in a cluttered map left and right respectively. Start and goal positions are green and red start respectively.

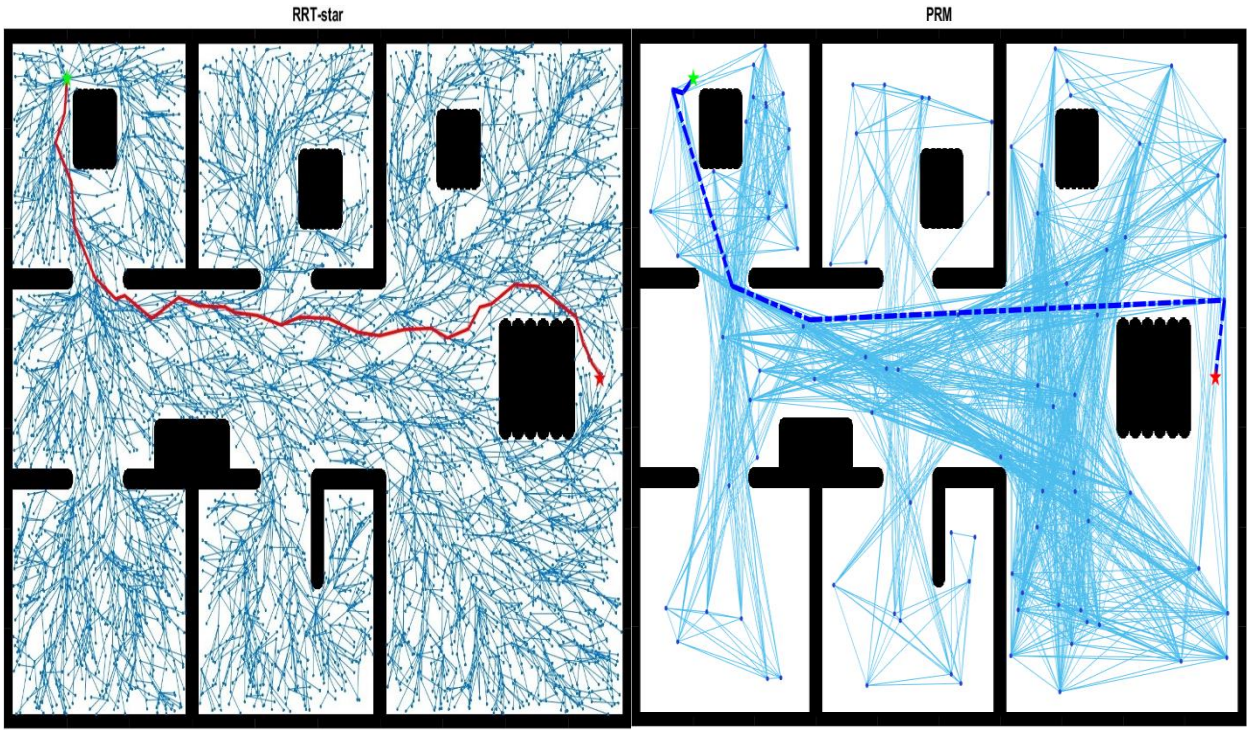


Figure 4.15 RRT* and PRM in a home shaped map left and right respectively. start and goal positions are green and red start respectively.

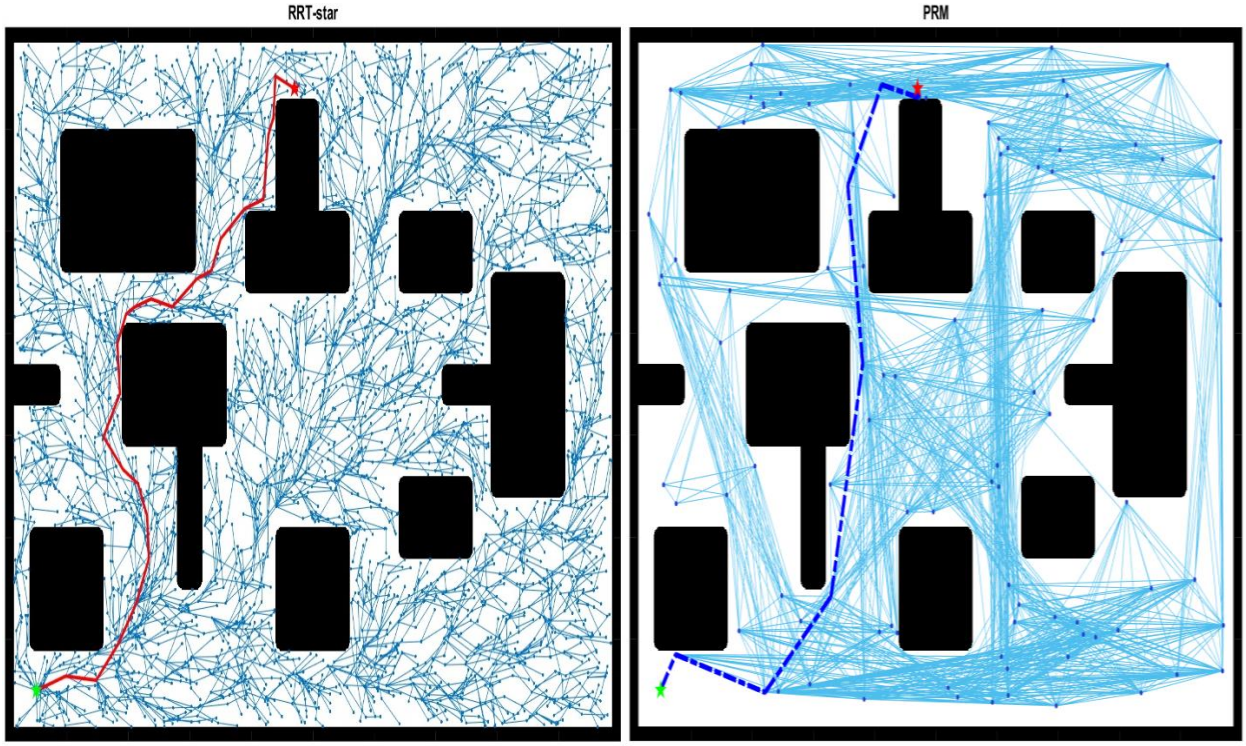


Figure 4.16 RRT* and PRM in a map with polygonal obstacles left and right respectively. start and goal positions are green and red start respectively.

Table 4.2 Comparison of path length and execution time for RAY,PRM,RRT* in different maps.

Map	Free map			Home shaped map			Map with polygonal obstacles			Cluttered map		
Planners	RAY	PRM	RRT*	RAY	PRM	RRT*	RAY	PRM	RRT*	RAY	PRM	RRT*
Path length (meters)	2.41	3.02	2.99	10.77	11.44	11.76	8.65	9.32	9.86	3.59	3.84	2.99
Number of control points	1	2	8	6	8	29	6	8	22	8	8	10
Execution time (seconds)	0.08	0.72	18.01	2.28	1.03	22.10	0.42	0.74	20.07	1.37	0.67	20.02

In the first map,(empty map) we see that the goal is reached easily in the first step and the Ray planner yields the shortest and optimal path possible (D=2.41 meters) in a very small time (T= 0.08 Seconds), because the search starts with radius equals to distance between the start and goal positions . Unlike PRM and RRT-star that generated a longer path as illustrated in table 4.2 and explored the whole map even when it is as empty as this one, in which PRM and RRT* execution times were comparable to the cases of more complex maps.

For the results all other maps the proposed algorithm find the goal always with optimal path and returned the shortest path and minimum number of control points, except in the cluttered map where RRT* generated the shortest path (D = 2.99 meters). But from execution time prospective we see that RAY planner is highly dependent on map complexity as illustrated in table 4.2, the running time is directly proportional to environment complexity which another advantage for this planner. In the other hand PRM and RRT* has approximately the same time execution because the planning is pre-determined by a fixed number of iterations or sampled points.

4.3 Critical cases

In this sub-section, we will discuss some critical scenarios that the proposed approach has difficulty to solve them.

First case; because of the search criteria in choosing the next point is goal oriented; then, if the start pose is in nearly closed location and the goal is separated just with a wall as illustrated in figure 4.15 bellow then the algorithm will take a long time or even enters in loop when trying to choose a random point suitable for continuing the search.

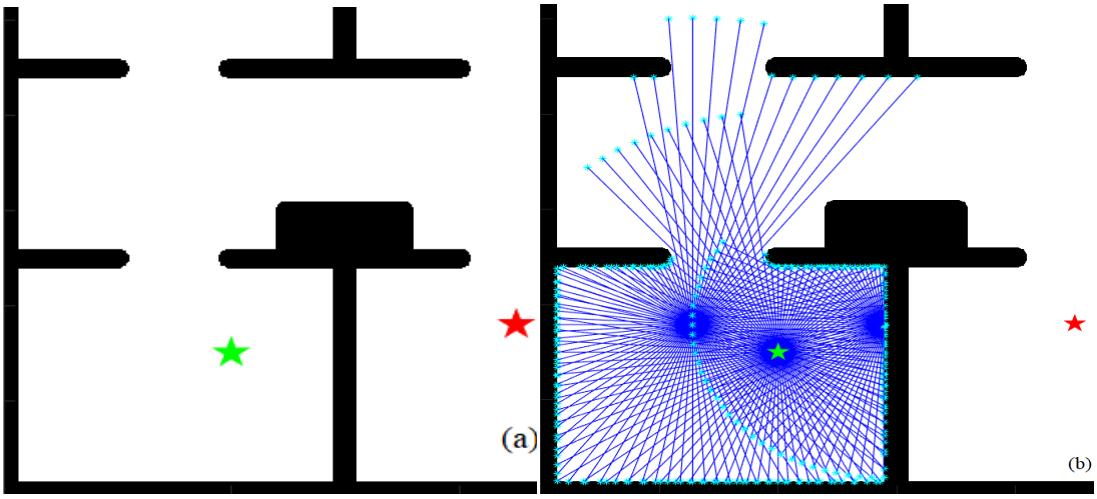


Figure 4.17 critical case scenarios (a) and the obtained results (b).

Second case; is when the start or goal locations is totally enclosed by obstacles. In this case the algorithm will return an error message “the goal position cannot be reached” after certain pre-defined N number of iterations. The case and obtained results are demonstrated in figure 4.16 below. The same thing if the existing free spaces are smaller than robot size as shown in figure 4.19, where the security zones of the obstacles overlap and the distance between the obstacles will be less then the robot diameter. In this case, the two obstacles are considered as a single obstacle and no free path is found between them.

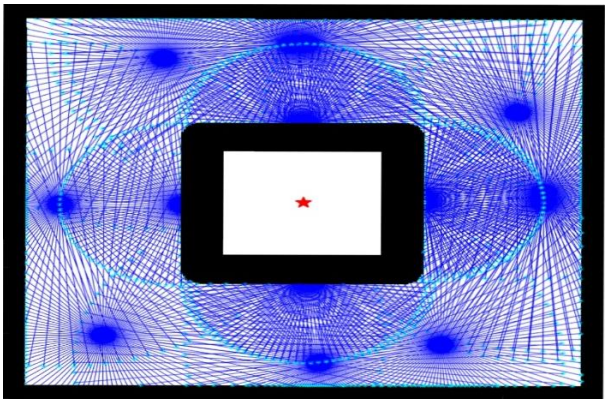


Figure 4.18 Critical case; goal is totally enclosed by obstacle.

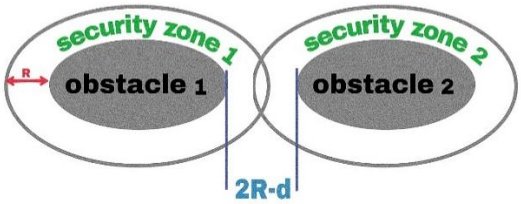


Figure 4.19 Critical case; free space smaller than robot size; d≥ 0.

Third case; when two obstacle are close to each other the approach can find free path between them in one condition, if the distance between the security zones is equal or greater than the dimension of grid cell (which is 1cm in our simulation) as shown in figure 4.20 bellow.

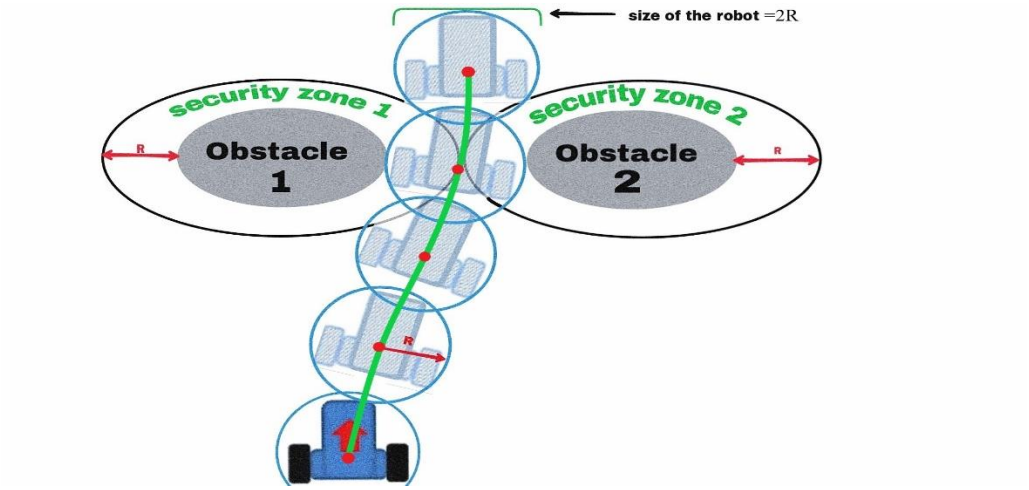


Figure 4.20 Third case critical scenario.

4.4 Conclusion

Through this chapter, the results of our work has been demonstrated. Starting by investigating the effect of Ray planner's different parameters on the generated path as well on the running time of the proposed algorithm; then, these parameters are tuned to compromise between the generated path length and the execution time of the algorithm. Furthermore, the results given by

Ray planner and the path follower algorithm and local planning are illustrated and discussed for different environment scenarios and complexities. These obtained results are further compared with results given by PRM and RRT* algorithms that are well known and used planners, this comparison yielded very good results in which Ray planner generated the shortest path almost in all cases with a good running time. Last some critical cases in which Ray planner has difficulties to deal with are highlighted.

Conclusion and perspectives

In this report a path planning method based on casting rays is proposed to find an optimal path for a mobile robot; this technique uses a modulated length rays to search for new points that are closer to the goal. In addition this algorithm takes advantages of graph theory to extract the optimal path from pre-planned feasible paths. This algorithm was further used along side well known smoothing technique and path tracking and a proposed simple approach to construct a binary occupancy map. The obtained results were satisfactory regarding the objectives of the project.

For future work, we will first concentrate on solving the current issues of the algorithm, which are the difficulties that are encounter in the mentioned critical scenarios especially the first critical case where the planner get stuck. In addition, to further optimizing the algorithm running time so it could be applied online; then, extend the approach for mobile robot in 3D environment that are suitable for AAVs and AUVs.

References and Bibliography

[1]Bruno Siciliano, Oussama Khatib (Eds). Springer Handbook of Robotics(Springer Handbooks: Springer-Verlag Berlin Heidelberg 2008),page number: 01.

[2]JeremyNorman's,historyofinformation.com.8/14/2021.https://www.historyofinformation.com/detail.php?entryid=4071.

[3]"Information Engineering Main/Home Page". www.robots.ox.ac.uk. Retrieved 2018-10-03.

[4]Heiserman. David (1976). Build Your Own Working Robot. TAB Books. ISBN 0830668411.

[5]Heiserman, David (1979). How to Build Your Own Self-Programming Robot. TAB Books. ISBN 0830612416

[6]Heiserman, David (1981). Robot Intelligence with Experiments. TAB Books. ISBN 0830696857.

[7] Z. H. Khan, Arsalan Khan, "Perspectives in Automotive Embedded Systems: From manual to fully autonomous vehicles", First International Symposium on Automotive and Manufacturing Engineering (SAME)At: SMME, NUST, Islamabad, Pakistan, Nov 2015..

[8]Hu, J. Bhowmick, P. Jang, I. Arvin, F. Lanzon, A. "A Decentralized Cluster Formation Containment Framework for Multirobot Systems" IEEE Transactions on Robotics, 2021.

[9]Hu, J.; Turgut, A.; Lennox, B.; Arvin, F., "Robust Formation Coordination of Robot Swarms with Nonlinear Dynamics and Unknown Disturbances: Design and Experiments" IEEE Transactions on Circuits and Systems II: Express Briefs, 2021.

[10]Hu, J.; Bhowmick, P.; Lanzon, A., "Group Coordinated Control of Networked Mobile Robots with Applications to Object Transportation" IEEE Transactions on Vehicular Technology, 2021.

[11] https://interestingengineering.com/lg-airport-robots-assisting-travelers-korea-airport

[12] Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. "Principles of Robot Motion Theory, Algorithms, and Implementation",2005.

[13]Sharma, Abhishek; Basnayaka, Chathuranga M.Wijerathna; Jayakody, Dushantha Nalin K.

"Communication and networking technologies for UAVs: A survey". Journal of Network and Computer Applications. 168:102739. arXiv: 2009.02280. doi:10.1016/j. jnca. 2020.102739. S2CID 221507920.

[14]https://www.tampabay.com/news/business/2020/08/31/amazon-wins-faa-approval-to deliver-packages-by-drone/

[15] https://www.propellertechnologies.in/index1.php

[16]"Sea glider: Autonomous Underwater Vehicle". www.apl.washington.edu.

[17] https://www.oedigital.com/news/477084-stinger-prepping-kawasaki-s-spice-auv

[18] http://chinaplus.cri.cn/photo/world/19/20190821/336520_1.html

[19]Thrun, S. et al.: Robotic mapping: a survey. In: Exploring artificial intelligence in the new millennium 1, page. 1–35.

[20]Alberto, P., et al.: Efficient integration of metric and topological maps for directed exploration of unknown environments. Robot. Auton. Syst. 41(1),page 21–39.

[21] Andrii Kudriashov , SLAM Techniques Application for Mobile Robot in Rough Terrain.

[22] http://hotellitlennotmatkat.fi/lontoo/.

[23] Elfes, A.: Using occupancy grids for mobile robot perception and navigation. Computer 22(6),46–57 (1989).

[24]LiDAR technologies and systems / Paul McManamon.

[25] https://www.semanticscholar.org/paper/TOF-Lidar-Development-in-Autonomous-Vehicle-Liu-Sun/2c92395e5f81e4ea6cb9f33475f9ead7417ba6ac

[26] https://www.jdpower.com/cars/shopping-guides/what-is-lidar-and-how-does-it-work .

[27] Noelia Llamazares Álvarez, "Study of path following algorithms for LIDAR obstacle detection and collision avoidance". Escola Tècnica Superior d'Enginyeria Industrial de Barcelona. June 2018. Page 16.

[28] LaValle, S.M. "Planning Algorithms". Cambridge University Press, Cambridge (2006).

[29]Burgard. W, et al. "Introduction to Mobile Robotics" Robot Motion Planning (2018). http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motionplanning.

[30]V. Lumelsky and A. Stepanov. "Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitrary shape". Algorithmica, 2.

[31]I. Kamon, E. Rivlin, and E. Rimon. "A new range-sensor based globally convergent navigation for mobile robots". In IEEE Int'l. Conf. on Robotics and Automation, Minneapolis, MN, April 1996.

[32]Wein, R., Van den Berg , J.P. Halperin, D . "The visibility-Voronoi complex and its applications". Comput. Geom. 36(1), 66–87.

[33]Hart, P.E. Nilsson, N.J., Raphael, B." A formal basis for the heuristic determination of minimum cost paths". IEEE Trans. Syst. Sci. Cybern. 4(2), 100–107.

[34]Stentz, A.: "The focussed D^* algorithm for real-time re-planning". In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), vol. 95, pp. 1652–1659.

[35]Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun, " Principles of Robot Motion: Theory, Algorithms, and Implementations", MIT Press, June 2005.

[36]G. Klanˇcar, A. Zdešar, S. Blažiˇc, I. Škrjanc." Wheeled mobile robotics From Fundamentals Towards Autonomous Systems ".

[37]Laser simulator: A novel search graph-based path planning approach by Mohammed. AH Ali and Musa Mailah , International Journal of Advanced Robotic Systems 2018: 1–16.

[38]Jitin Kumar Goyal ,K.S. Nagla " A New Approach of Path Planning for Mobile Robots ". 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI).

[39]Yun-Qian Miao, Alaa M. Khamis, Fakhri Karray, and Mohamed S. Kamel, A novel approach to path planning for autonomous mobile robots.

[40]N. M. Wardhana, H. Johan, and H. S. Seah, "Enhanced waypoint graph for surface and volumetric path planning in virtual worlds," The Visual Computer, vol. 29, no. 10, pp. 1051–1062.

[41]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6165411/ ,accessed on 08/28/2021.

[42]Wallace, R. et al. "First Results in Robot Road-Following", repon witbio CMU-RI-TR-8&4.

[43]R. Craig Conlter. Implementation of the Pure Pursuit Path 'hcking Algorithm. January 1992.

[44]www.mathworks.com,https://fr.mathworks.com/help/robotics/ug/pure-pursuit controller.html ,accessed 08/23/2021