Self-Parking Car

Submitted By:

Mohamed Hatem
Ahmed Mamdoh Mohamed
Thomas Medhat Mounir Botros
Youssef Mohamed Mahmoud
Hussam Elsayed
Andrew Boulos
Amr Anwar Amen

Mechatronics Engineering and Automation

Dr. Mohamed Ahmed Mahmoud AbdelWahab

Dr. Mohamed Essam

31/1/2023

1 Elsarayat St., Abbaseya, 11517 Cairo, Egypt
Fax: (+20 2) 26850617
www.eng.asu.edu.eg

DECLARATION

We/I hereby certify that this Project submitted as part of our/my partial fulfilment of BSc in ***(Mechatronics Engineering and Automation)*** is entirely our/my own work, that we/I have exercised reasonable care to ensure its originality, and does not to the best of our/my knowledge breach any copyrighted materials, and have not been taken from the work of others and to the extent that such work has been cited and acknowledged within the text of our/my work.

Signed: by all students

Mohamed Hatem	18P3449
Thomas Medhat Mounir Botros	18P8912
Ahmed Mamdoh Mohamed	19P5326
Hussam Elsayed	18P6670
Youssef Mohamed Mahmoud	18P6528
Amr Anwar Amen	18P4576
Andrew Boulos	18P7917

Date: 31/1/2023.

ACKNOWLEDGMENT

We would like to express our appreciation to Dr. Mohamed Abdelwahab and Dr. Mohamed Essam for all the effort and support they provided throughout the project. They taught us the correct methodology to conduct the research and to present the research works as clearly as possible.

Finally, we would like to thank our Faculty of Engineering, Ain Shams University, for their support and for providing us with their facilities.

ABSTRACT

This report will discuss the attempt of our team to make a toy car park in its designated parking slot , First we discuss the SLAM algorithms which gives us the location of the robot and the map surrounding ,Second we discuss the path planning algorithms that will make the car go to the parking slot ,Third we discuss the vision and sensor fusion to be able to detect the parking slot ,Then we will discuss how we control the motor , Finally we will discuss the tracking algorithms and the movement it will take to finally park in its slot.

Each Stage will discuss multiple algorithms and compare between them in addition to presenting simulation results of every algorithm then deciding on the algorithm that will be used in implementation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

Self-parking cars, also known as autonomous parking or park assist technology, have revolutionized the way we think about parking and driving. With the advent of advanced sensors, cameras, and algorithms, cars can now park themselves with minimal human intervention. This technology has the potential to make parking more convenient, efficient, and safer, as well as to address the growing demand for urban parking space.

Parallel parking is an ordeal for many drivers, but with parking space limited in big cities, squeezing your car into a tiny space is a vital skill. It's seldom an easy task, and it can lead to traffic tie-ups, frazzled nerves and bent fenders. Fortunately, technology has an answer - cars that park themselves. Imagine finding the perfect parking spot, but instead of struggling to maneuver your car back and forth, you simply press a button, sit back, and relax. The same technology used in self-parking cars can be used for collision avoidance systems and ultimately, self-driving cars.

Automakers are starting to market self-parking cars because they sense a consumer demand.

1.1 MOTIVATION:

Parallel parking is often the most feared part of the driver's test, and it's something almost everyone must do at some point. People who live in big cities may have to do it every day. Removing the difficulty, stress and uncertainty of this chore is very appealing.

Self-parking cars can also help to solve some of the parking and traffic problems in dense urban areas. Sometimes parking a car in a space is restricted by the driver's skill at parallel parking. A self-parking car can fit into smaller spaces than most drivers can manage on their own.

This makes it easier for people to find parking spaces and allows the same number of cars to take up fewer spaces. When someone parallel parks, they often block a lane of traffic for at least a few seconds. If they have problems getting to the spot, this can last for several minutes and seriously disrupt traffic.

1.2 OBJECTIVE:

Self-parking technology is mostly used in parallel parking situations (although BMW has a prototype that parks itself in horizontal spaces, like small garages). Parallel parking requires cars to park parallel to a curb, in line with the other parked cars. Most people need about six feet more space than the total length of their car to successfully parallel park, although some expert drivers can do it with less space.

To parallel park, the driver must follow these five basic steps:

1. He pulls ahead of the space and stops beside the car in front of it.
2. Turning the car's wheels towards the curb, he backs into the space at around a 45-degree angle.
3. When his front wheels are even with the rear wheels of the car in front of him, he straightens them and continues backing up.
4. While checking his rear view to be sure that he doesn't come too close to the car behind him, the driver turns his wheels away from the curb to swing the front end of his car into the space.
5. Finally, the driver pulls forward and backwards in the space until his car is about one foot away from the curb.

Our objective is to make all this autonomous, so the driver doesn’t have to do this movement every time.

1.3 THESIS:

Throughout this report we will talk about how we will determine our location and map in the second chapter, then we will talk about the path planning in the third chapter, then we will talk about how we will detect the parking slot in the fourth chapter, then the control of motor in the fifth chapter, then finally we will talk about the parking movement and tracking of the path in the last two chapters.

CHAPTER TWO: SLAM

2.1 INTRODUCTION:

Simultaneous localization and mapping (SLAM) are the computational problems of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. While this initially is a chicken-and-egg problem, there are several algorithms known for solving it in, at least approximately, tractable time for certain environments. Popular approximate solution methods include the particle filter, extended Kalman filter, covariance intersection, and Graph SLAM.

SLAM algorithms are based on concepts in computational geometry and computer vision, and are used in robot navigation, robotic mapping and odometry for virtual reality or augmented reality.

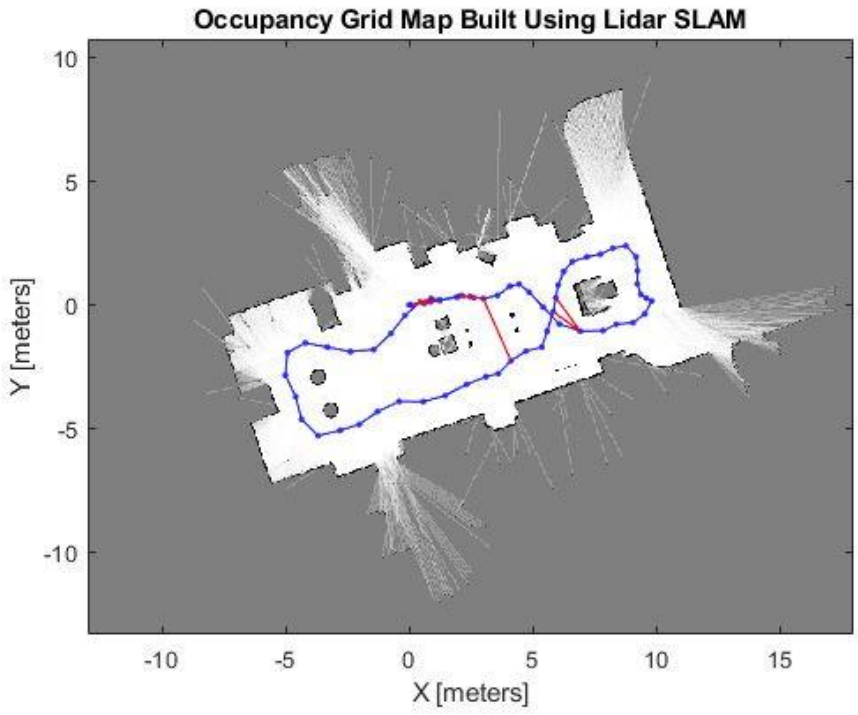


Figure 1 : Slam Using Lidar

2.2 ANGLE REPRESENTATION:

Rotation in 3D state-space can be represented in various forms, such as Euler angles, direction cosine matrices and unit quaternions. Different representations are more suitable for expressing an angular orientation or rotational motion than others at different situations. This section will devote itself to equip the reader with the different ways of expressing angles and rotations.

2.2.1 Euler angles:

As the easiest method to visualize orientation and rotation of a rigid body in state-space, Euler angles are constituted by a set of three elemental rotations about each axis in Euclidean space.

• roll, ϕ: rotation about the x-axis.

• pitch, θ: rotation about the y-axis.

• yaw, ψ: rotation about the z-axis.

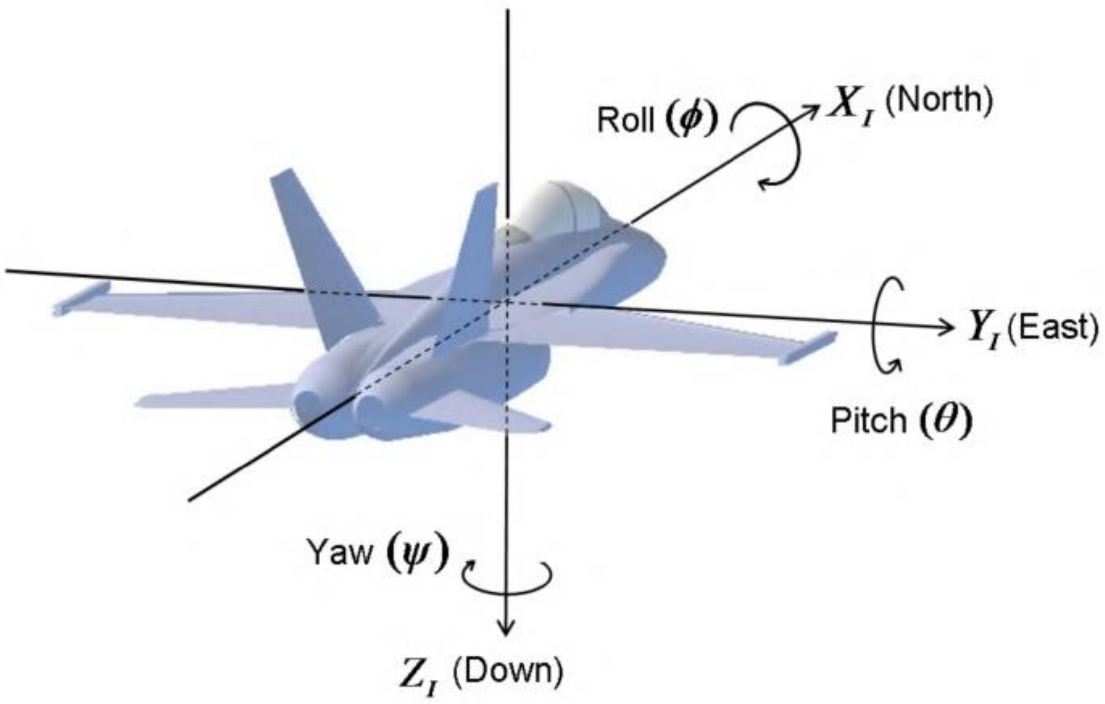


Figure 2 : Euler angles

2.2.2 Direction cosine matrix:

A direction cosine matrix (DCM) is a matrix representation of a space-state. Its vectors consist of the unit vectors that describe the current coordinate frame with respect to an origin coordinate frame It also allows for multiple successive rotations by matrix multiplying the rotation matrices with each other.

A 3D rotation about the x-axis, i.e., a roll motion ϕ, can be expressed as a DCM as such:

$$\boldsymbol{R_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

Similarly, for a pitch motion θ and a yaw motion ψ:

$$\boldsymbol{R_y} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\boldsymbol{R_z} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A generalized expression of a rotation about all axes can thus be described as:

$$\boldsymbol{R} = \boldsymbol{R_z R_y R_x} = \begin{bmatrix} \cos(\theta)\cos(\psi) & -\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) \\ \cos(\theta)\sin(\psi) & \cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

one can find the Euler angles with these formulae:

$$\phi = atan2(r_{32}, r_{33})$$
$$\theta = atan2\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right)$$
$$\psi = atan2(r_{21}, r_{11})$$

2.2.3 Unit quaternions:

Unit quaternions are very convenient tool to represent 3D orientations and rotations and are widely used in computer graphics, robotics, and flight dynamics. Unlike Euler angles, they avoid the problem of gimbal lock, and they are more numerically stable and easier to compose than both DCMs and Euler angles. A quaternion q is a fourtuple number system described by the real parameters $\{q\text{h}, q\text{N}, q\text{Y}, q\text{Z}\}$ as such:

$$q = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k}$$

The conversion from quaternions to Euler angles can be described with the following formulae:

$$\phi = atan2\left(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2)\right)$$
$$\theta = arcsin\left(2(q_w q_y - q_x q_z)\right)$$
$$\psi = atan2\left(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2)\right)$$

Conversely, Euler angles can be converted to quaternions:

$$q_w = \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)$$
$$q_x = \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)$$
$$q_y = \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)$$
$$q_z = \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right)$$

2.3 SENSORS :

2.3.1 Lidar:

LiDAR stands for Light Detection and Ranging and is a type of optical rangefinder device that performs laser scanning on an arbitrary scan plane of the immediate surrounding of the system. It provides point clouds of high resolution and precision in a wide range of directions at a high sampling rate. This can then be processed using various algorithms to estimate the trajectory taken by the moving device whilst mapping the environment.

The working principle of a LiDAR is to shoot short light pulses in each radial direction within its scan range from its scan origin, which travels towards the nearest distant object in its paths. It then bounces back and is registered by the LiDAR, which then measures the time it took for the light pulse to travel back and forth. By multiplying the travel time with the speed of light, the distance to the closest object in the given direction can be inferred. Every scan consists of a collection of scan points within the scan range, which comprise a so-called point cloud.

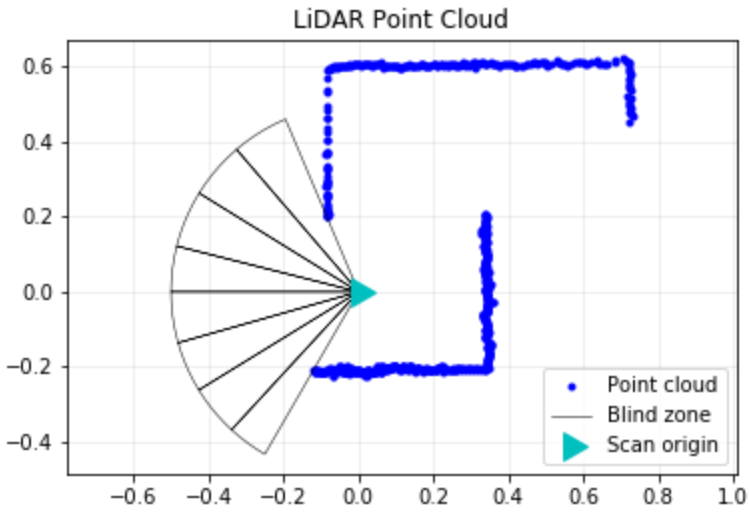


Figure 3 : Lidar Point Cloud

Scan Matching:

It is possible to estimate the trajectory of the LiDAR system by associating scan points, or features inferred from the scan points, between the point clouds of multiple scans. The

goal of laser scan matching is to find the translation and rotation of the current point cloud with respect to a reference scan such that the best overlap is achieved.

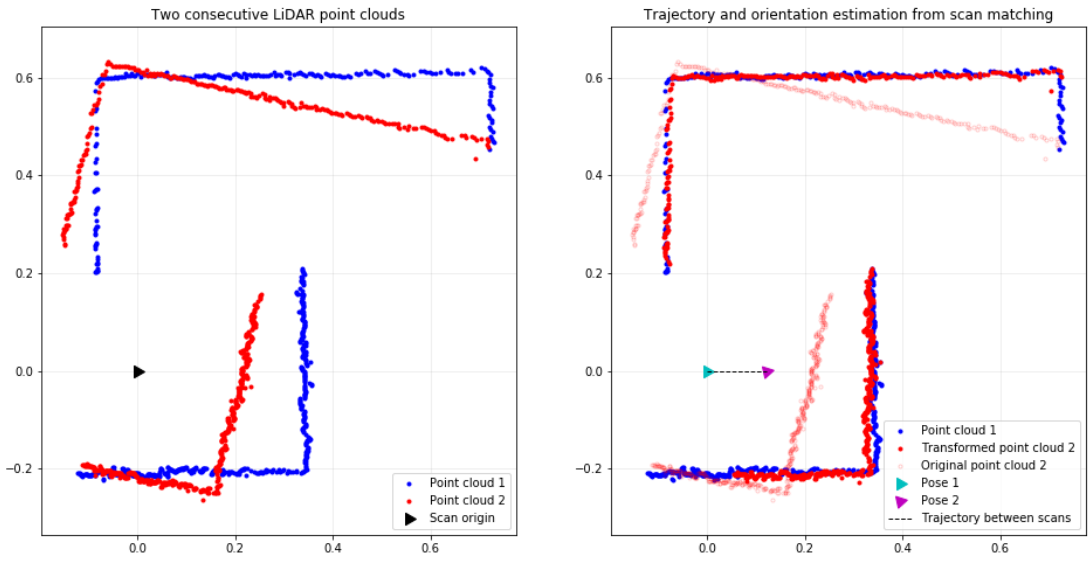


Figure 4 : Working Principle of LiDAR Scan Matching

Different scan matching algorithms will naturally yield different results in the accuracy of the trajectory and environment reconstruction and is also highly dependent on whether the environment is structured (i.e., contains extractable features) or complex.

2.4 SIMULATION:

In this project the simulation is classified to two parts which are the offline slam and the online slam

Online SLAM, as you can likely guess, is active while the robot is in flight and prioritizes object detection and avoidance over the quality of the map. Whereas offline SLAM takes place after the robot has landed and prioritizes map quality and loop closure now that we don't have to worry about robot control.

2.4.1 Offline Slam:

In this part, the objective was to compare between two scan matching algorithms using the same dataset and, those two algorithms are the ICP (Iterative Closest Point) and the Feature-Based.

Iterative Closest Point (ICP):

ICP is a point-to-point algorithm that has been widely used and studied in literatures, it seeks the optimal rigid transformation between two consecutive scans locally, which is the one that minimizes the least squares criterion between corresponding points.

the general procedure is as follows:

1. Correspondence search: for each scan point in P_k, choose and match the scan point in P_{k-1} that has the closest Euclidean distance to the point. The index of the corresponding point in P_{k-1} is assigned to $c(i)$. This can be done with a k-nearest neighbour code implementation.

2. Find the rotation matrix $\boldsymbol{R}$ and translation vector $\boldsymbol{t}$ that minimizes the least square error

3. Transform the scan points in P_k with the obtained transformation arrays.

4. Iterate steps 1-3 until the solution has converged.

Feature-based scan:

Lines are detected based on clustered sets in the point cloud that resembles a linear structure, whereas key points are unique objects consisting of corners and endpoints in the line environment that are tagged to a position in the global frame.

A line segment can be found by minimizing the orthogonal distance of each scan point from the line, by denoting it as a linear equation in 2D Euclidean space of the form.

$$ax + by + c = 0.$$

The line fitting procedure consists thus of finding the line parameters $\{a, b, c\}$ that minimize the sum of squares of distances from the subset of points with coordinates (X_p, Y_p) that form the line.

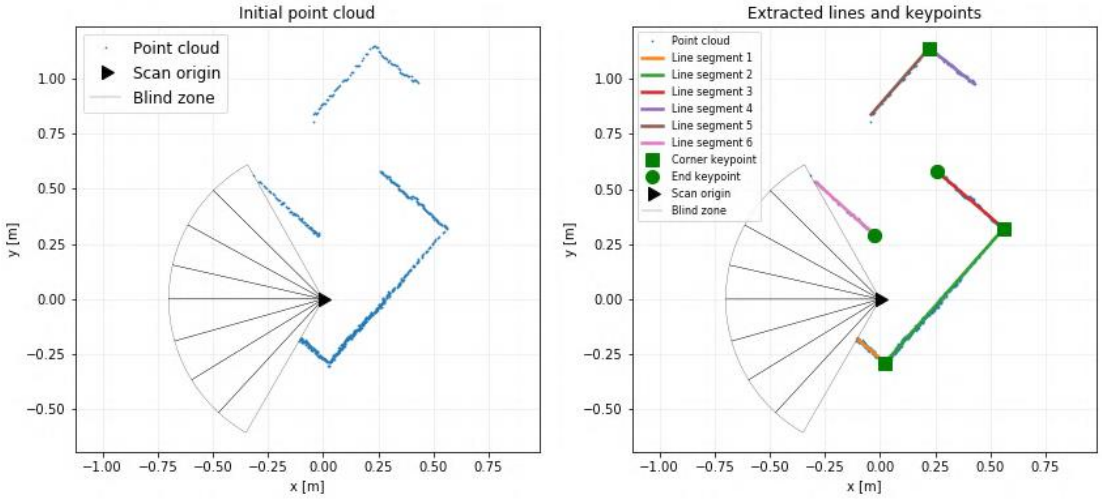


Figure 5 : Visualization of the line segment extraction

Results and Comparison:

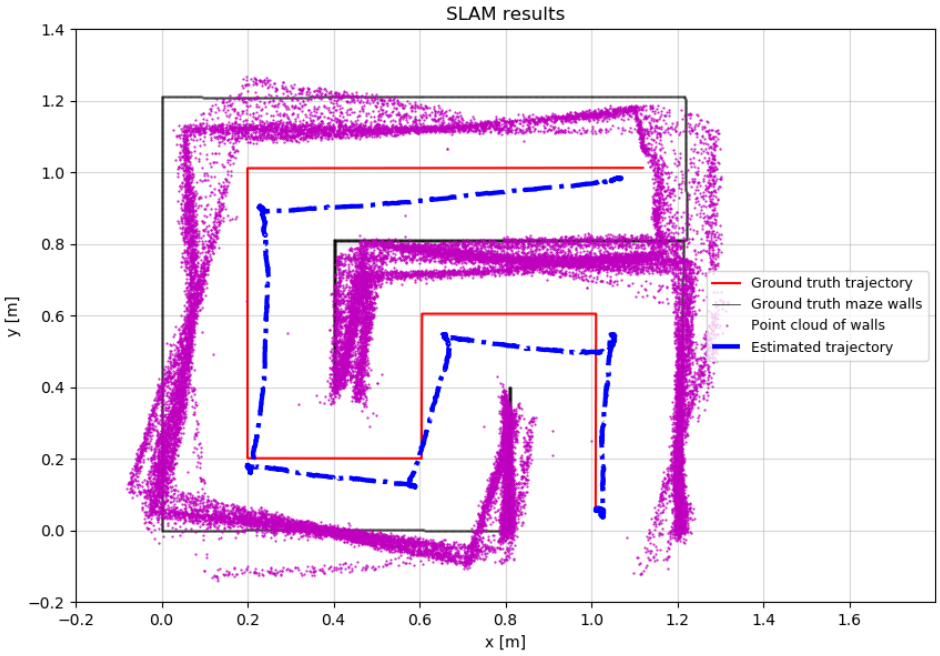


Figure 6 : ICP Output

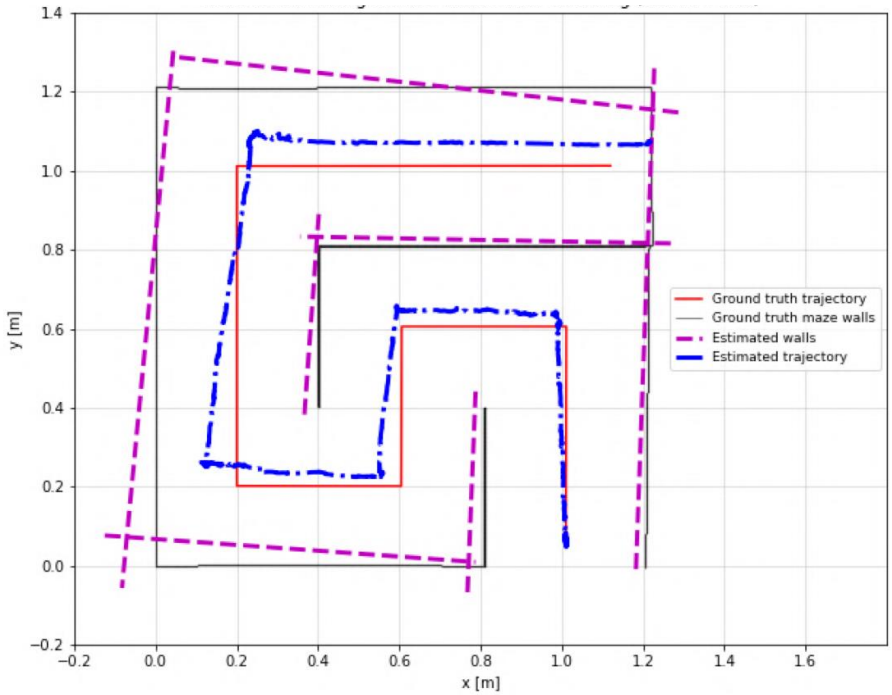


Figure 7 : Feature-Based Scan Output

Table 1 : deviation error (in meters) of estimated environment structure from ground truth

Test Run	ICP	Feature-Based
1	0.0324	0.0307
2	0.0560	0.0421
3	0.0374	0.1212
4	0.0439	0.0667
5	0.0555	0.0884
Average	0.0450	0.0698

Table 2 : deviation error (in meters) maze walls of estimated trajectory from ground truth trajectory

Test Run	ICP	Feature-Based
1	0.0443	0.0202
2	0.0654	0.0369
3	0.0456	0.1145
4	0.0550	0.0619
5	0.0561	0.0892
Average	0.0533	0.0645

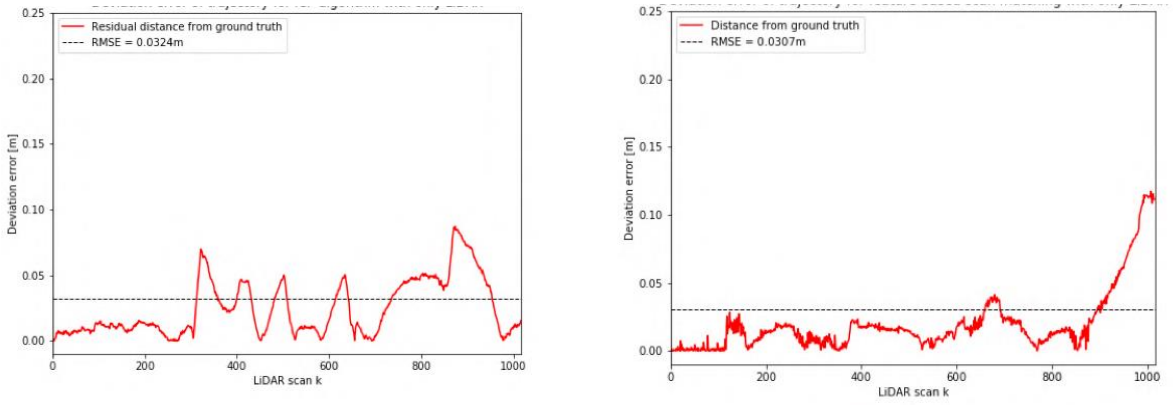


Figure 8 : Deviation Error Plots

The rest are in the Appendix.

```
Iteration 5000/5047: Time elapsed 1m31sec - Est. time remaining 0m0sec
Finished iterating after 1m31sec
Plotting results
==================================================
Deviation errors
==================================================
RMSE for trajectory estimate: 0.048269968191510346 m
```

Figure 9 : ICP Runtime

```
Iteration 5000/5047: Time elapsed 10m5sec - Est. time remaining 0m5sec
Finished iterating after 10m9sec
Plotting results
==================================================
Deviation errors
==================================================
RMSE for trajectory estimate: 0.03763732530972265 m
```

Figure 10 : Feature-Based Runtime

2.4.2 Online Slam:

After finishing the offline simulations, we had to find a way to perform the slam process in Realtime that is why we went to search for multiple online slam applications to use for our lidar.

After a lot of searching, the online slam was done on three main applications which are the Hector Slam, GMapping, and Google Cartographer

Hector Slam:

The Hector-SLAM algorithm differs from other grid-based mapping algorithms, as it does not require odometer information, but it needs laser data and a priori map.

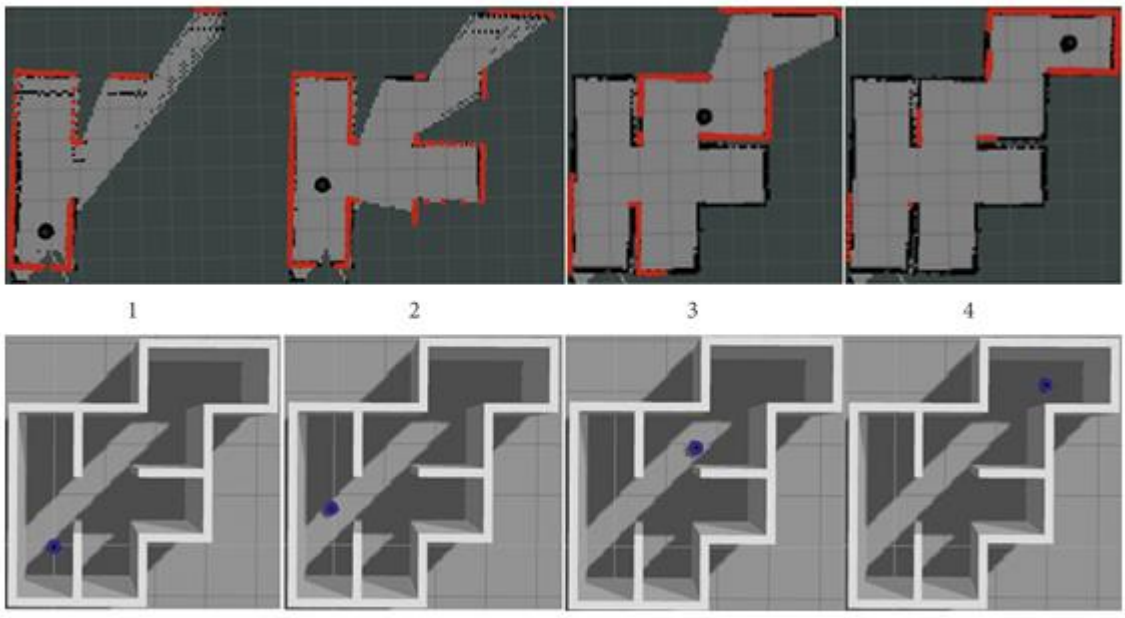


Figure 11 : Hector SLAM Simulation

GMapping:

The GMapping algorithm is a laser-based SLAM algorithm for grid mapping (G for grid), and the main idea is to use Rao–Blackwellized particle filters (RBPFs) to predict the state transition function. The algorithm is also known as the RBPF SLAM algorithm, named after the use of Rao–Blackwellized particle filters.

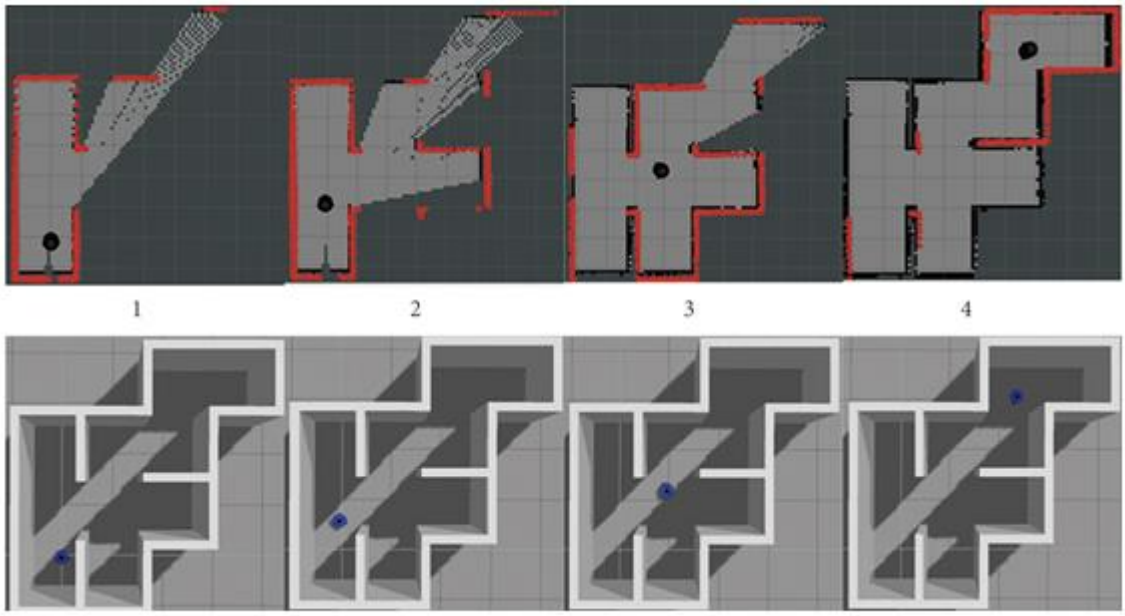


Figure 12 : GMapping Simulation

Assuming there are stairs and rugged surface which make the odometer inaccurate. It means we could not choose GMapping because it is very relied on odometer. Due to the rugged surface, so we choose cartographer.

Google Cartographer:

Google's solution to SLAM, called Cartographer, is a graph optimisation algorithm. The function of Cartographer is to process the data from Lidar, IMU, and odometers to build a map.

Cartographer ROS then acquires the sensor data through the ROS communication mechanism and converts them into the Cartographer format for processing by Cartographer, while the Cartographer processing result is released for display or storage.

It produces Impressive real-time results for solving SLAM in 2D.

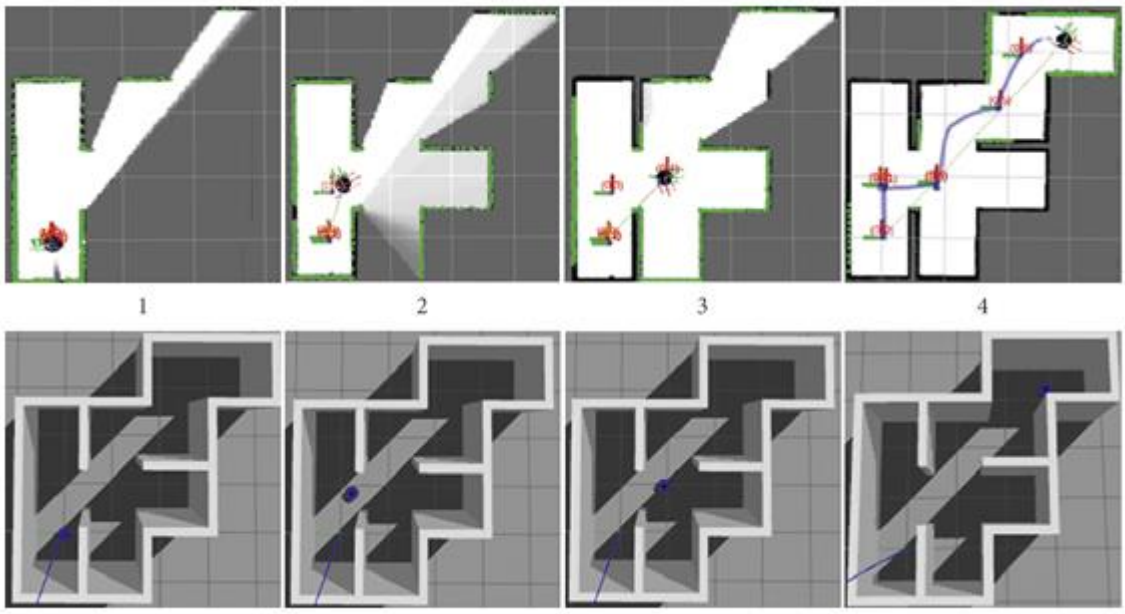


Figure 13 : Google Cartographer Simulation

2.5 CONCLUSION:

So, in the end, Google Cartographer was chosen to be the Slam Online Application After figuring out the map we need to draw our path in which the car will follow, we will talk about that in the next chapter.

CHAPTER THREE: PATH PLANNING

3.1 INTRODUCTION TO PATH PLANNING

Moving from one place to another is a trivial task, for humans. One decides how to move in a split second. For a robot, such an elementary and basic task is a major challenge. In autonomous robotics, path planning is a central problem in robotics. The typical problem is to find a path for a robot, whether it is a vacuum cleaning robot, a robotic arm, or a magically flying object, from a starting position to a goal position safely [1].

The problem consists in finding a path from a start position to a target position. This problem was addressed in multiple ways in the literature depending on the environment model, the type of robots, the nature of the application, etc. Safe and effective mobile robot navigation needs an efficient path planning algorithm since the quality of the generated path affects enormously the robotic application. Typically, the minimization of the travelled distance is the principal objective of the navigation process as it influences the other metrics such as the processing time and the energy consumption [1] [2].

This chapter presents an overview on mobile robot global path planning and provides the necessary background on this topic. It describes the different global path planning categories and presents a taxonomy of global path planning problem.

3.2 OVERVIEW OF THE ROBOT PATH PLANNING PROBLEM

Nowadays, we are at the cusp of a revolution in robotics. A variety of robotic systems have been developed, and they have shown their effectiveness in performing various kinds of tasks [2]. An intelligence must be embedded into robot to ensure (near)-optimal execution of the task under consideration and efficiently fulfil the mission. However, embedding intelligence into robotic system imposes the resolution of a huge number of research problems such as navigation which is one of the fundamental problems of mobile robotics systems [3].

To successfully finish the navigation task, a robot must know its position relatively to the position of its goal. Moreover, the robot must take into consideration the dangers of the

surrounding environment and adjust its actions to maximize the chance to reach the destination.

Putting it simply, to solve the robot navigation problem, we need to find answers to the three following questions: Where am I? Where am I going? How do I get there? These three questions are answered by the three fundamental navigation functions localization, mapping, and motion planning, respectively.

• Localization: It helps the robot to determine its location in the environment.

• Mapping: The robot requires a map of its environment in order to identify where it has been moving around so far. The map helps the robot to know the directions and locations. The map can be placed manually into the robot memory (i.e., graph representation, matrix representation) or can be gradually built while the robot discovers the new environment.

• Motion planning or path planning: To find a path for the mobile robot, the goal position must be known in advance by the robot, which requires an appropriate addressing scheme that the robot can follow. The addressing scheme serves to indicate to the robot where it will go starting from its starting position [1].

Planning is one obvious aspect of navigation that answers the question: What is the best way to go there? Indeed, for mobile robotic applications, a robot must be able to reach the goal position while avoiding the scattered obstacles in the environment and reducing the path length.

There are various issues need to be considered in the path planning of mobile robots due to various purposes and functions of the virtual robot itself as shown in **Figure 14: Path Planning issues**. Most of the proposed approaches are focusing on finding the shortest path from the initial position to goal position.

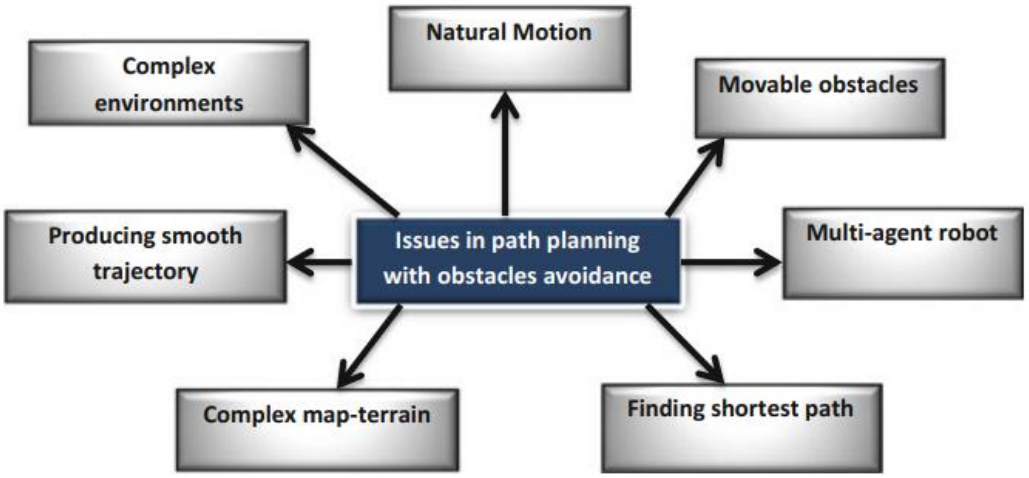


Figure 14: Path Planning issues

Recently, research is focusing on reducing the computational time and enhancing smooth trajectory of the virtual robot. Other ongoing issues include navigating the autonomous robots in complex environments. Some researchers consider movable obstacles and navigation of the multi-agent robot [4].

Whatever the issue considered in the path planning problem, three major concerns should be considered: efficiency, accuracy, and safety [1]. Any robot should find its path in a short amount of time and while consuming the minimum amount of energy. Besides that, a robot should avoid safely the obstacles that exist in the environment. It must also follow its optimal and obstacle-free route accurately.

Planning a path in large-scale environments is more challenging as the problem becomes more complex and time-consuming which is not convenient for robotic applications in which real-time aspect is needed.

In our work, we concentrate on finding the best approach to solve the path planning for finding shortest path in a minimum amount of time. We also considered that the robot operates in a large environment containing several obstacles having arbitrary positions.

3.3 PATH PLANNING CATEGORIES

In this section, we give a classification of the different problems related to mobile robot's path planning. It can be divided into three categories according to the robot's knowledge

that it has about the environment, the environment nature, and the approach used to solve the problem as depicted in **Figure 15: Path Planning Categories**.

Environment Nature: The path planning problem can be done in both static and dynamic environments: A static environment is unvarying, the source and destination positions are fixed, and obstacles do not vary locations over time. However, in a dynamic environment, the location of obstacles and goal position may vary during the search process. Typically, path planning in dynamic environments encompasses is more complex than that in static environments due to uncertainty of the environment. As such, the algorithms must adapt to any unexpected change such as the advent of new moving obstacles in the preplanned path or when the target is continuously moving. When both obstacles and targets are moving, the path planning problem becomes even more critical as it must effectively react in real time to both goal and obstacle movements. The path planning approaches applied in static environments are not appropriate for the dynamic problem [5].

Map knowledge: Mobile robots path planning basically relies on an existing map as a reference to identify initial and goal location and the link between them. The amount of knowledge to the map plays an important role for the design of the path planning algorithm. According to the robot's knowledge about the environment, path planning can be divided into two classes: In the first class, the robot has an a priori knowledge about the environment modelled as a map [5].

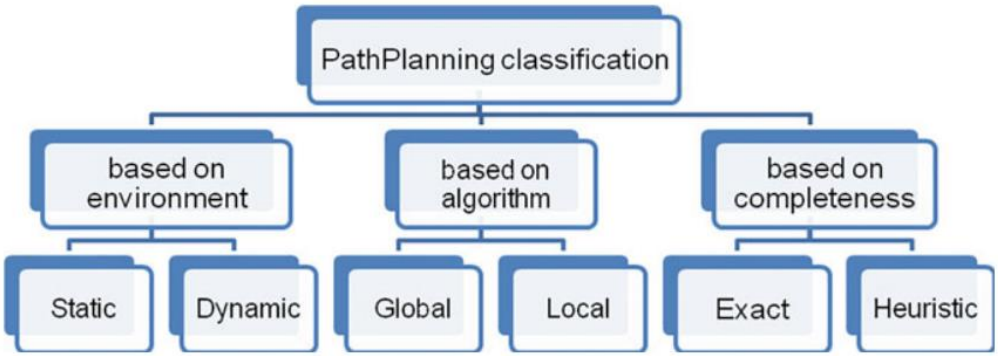


Figure 15: Path Planning Categories

Table 3: Global and local path planning

Local Path Planning	Global Path Planning
Sensor-based	Map-based
Reactive navigation	Deliberative navigation
Fast response	Slower response
Suppose that the workspace area is incomplete or partially incomplete	The workspace area is known
Generate the path and moving toward target while avoiding obstacles or objects	Generate a feasible path before moving toward the goal position
Done online	Done offline

This category of path planning is known as global path planning or deliberative path planning. The second class of path planning assumes that the robot does not have a priori information knowledge about its environment (i.e., uncertain environment). Consequently, it must sense the locations of the obstacles and construct an estimated map of the environment in real time during the search process to avoid obstacles and acquire a suitable path toward the goal state. This type of path planning is known as local path planning or reactive navigation. **Table 3: Global and local path planning** presents the differences between the two categories.

Completeness: Depending on its completeness, the path planning algorithm is classified as either exact or heuristic. An exact algorithm finds an optimal solution if one exists or proves that no feasible solution exists. Heuristic algorithms search for a good-quality solution in a shorter time [1].

3.4 GLOBAL PATH PLANNING

In the literature, numerous path planning algorithms have been proposed. Although the objective of these algorithms is to find the shortest path between two positions A and B in a particular environment, there are several algorithms based on a diversity of approaches to find a solution to this problem. The complexity of algorithms depends on the underlying techniques and on other external parameters, including the accuracy of the map and the number of obstacles [1].

The research on the path planning problem started in late 1960. Nevertheless, most of the efforts are more recent and have been conducted during the 80’s. Afterward, several research initiatives, aiming at providing different solutions in both static and dynamic environments, have emerged. Numerous approaches to design these solutions have been

attempted, which can be widely classified into three main categories: classical approaches, heuristic approaches, and graph search approaches. The classical methods are variations and/or combinations of a few general approaches such as Roadmap, potential field, cell decomposition. These methods dominated this field during the first 20 years. However, they were deemed to have some deficiencies in global optimization, time complexity, robustness, etc. The second category of methods used to solve the path planning problem is heuristic approaches which were designed to overcome the limits of classical methods. Moreover, numerous numbers of graph search algorithm developed over the last decades have been tested for path planning such as A∗, Dijkstra, Bellman Ford, etc [6].

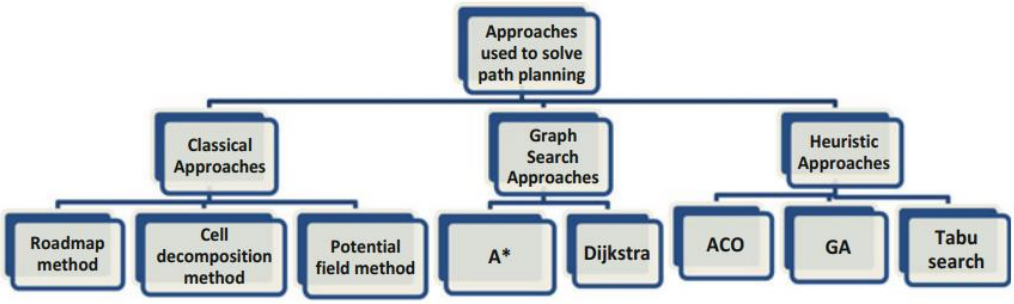


Figure 16: Path planning approaches

3.5 CLASSICAL APPROACHES

The three most successful classical approaches used to solve global robot path planning. For each approach, we present the basic ideas and a few different realizations. For these methods, an explicit representation of the configuration space is assumed to be known.

- The roadmap approach and its different methods such as visibility graphs method, freeway method, and silhouette method. The basic idea of roadmap methods is to create a roadmap that reflects the connectivity of C free. A set of lines, each of which connect two nodes of different polygonal obstacles, lie in the free space and represent a roadmap R. If a continuous path can be found in the free space of R, the initial and goal points are then connected to this path to arrive at the final solution, a free path. If more than one continuous path can be found and the number of nodes on the graph is relatively small, Dijkstra's shortest path algorithm is often used to find the best path [6].

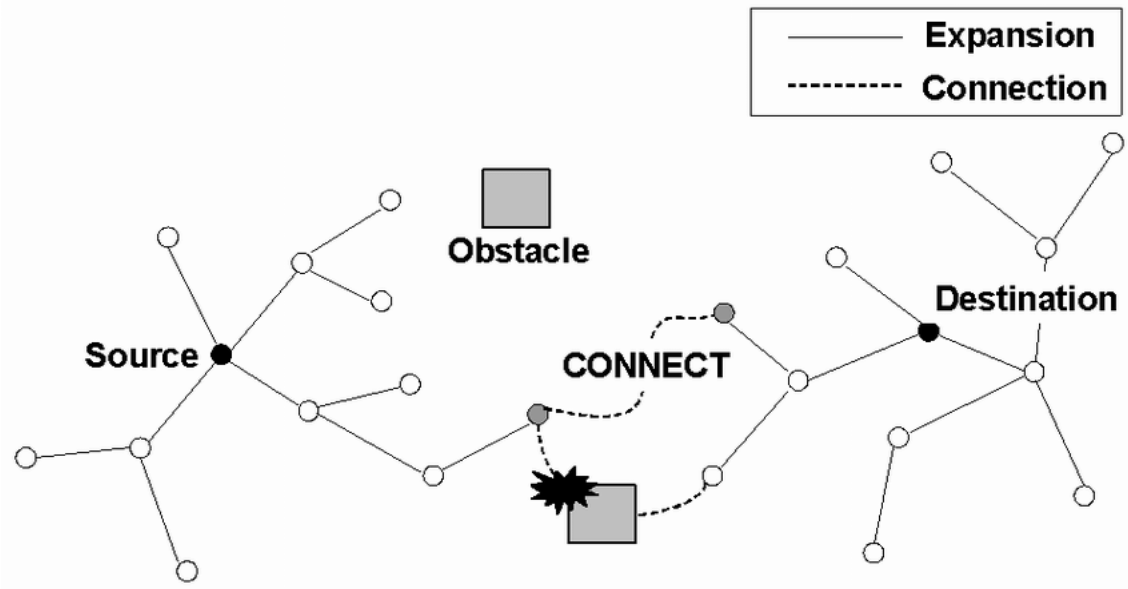


Figure 17: Roadmap

- The cell decomposition approach and its variants exact cell decomposition, approximate cell decomposition, and wave front planner. The basic idea behind this method is that a path between the initial node and the goal node can be determined by subdividing the free space of the robot's configuration into smaller regions called cells. The decomposition of the free space generates a graph called as a connectivity graph in which each cell is adjacent to other cells. From this connectivity graph, a continuous path can be determined by simply following adjacent free cells from the initial point to the goal point. The first step in cell decomposition is to decompose the free space, which is bounded both externally and internally by polygons, into trapezoidal and triangular cells by simply drawing parallel line segments from each vertex of each interior polygon in the configuration space to the exterior boundary. Then, each cell is numbered and represented as a node in the connectivity graph. Nodes that are adjacent in the configuration space are linked in the connectivity graph. A free path is constructed by connecting the initial configuration to the goal configuration through the midpoints of the intersections of the adjacent cells [6] [7].

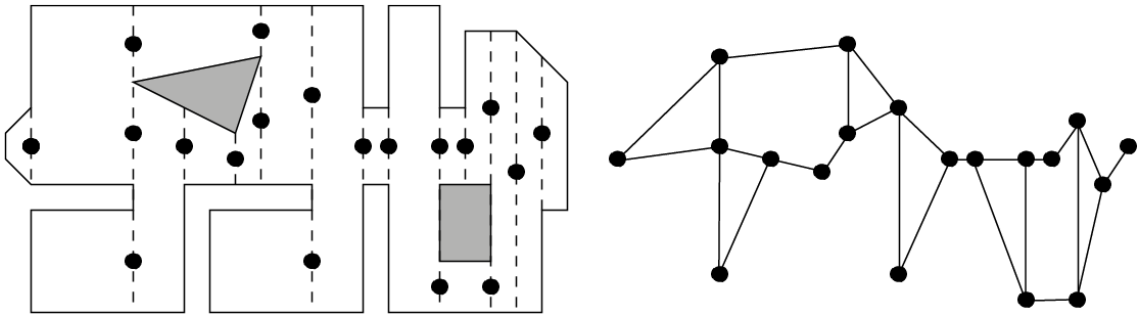


Figure 18: Cell decomposition

- Artificial potential field approach (PFM) involves modelling the robot as a particle moving under the influence of a potential field that is determined by the set of obstacles and the target destination. The obstacles and the goal are assigned repulsive and attractive forces, respectively, so that the robot is able to move toward the target while pushing away from obstacles.

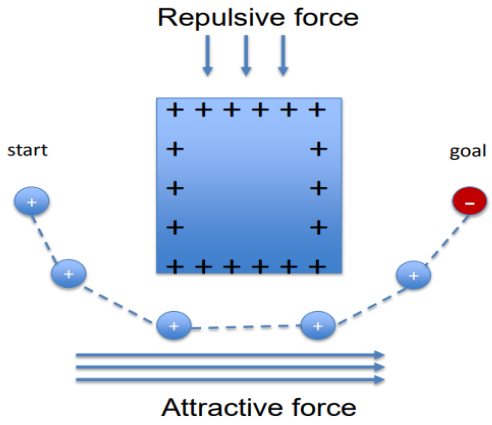


Figure 19: Artificial potential field

Limitations: The classical approaches were found to be effective and capable of solving the problem through providing feasible collision-free solutions. However, these approaches suffer from several drawbacks. Classical approaches consume much time to find the solution of the problem, which is considered as a significant drawback especially when dealing with difficult problems of large-scale and complex environments, since they tend to generate computationally expensive solutions. Another drawback of these classical approaches is that they might get trapped in local optimum solutions away from the global optimum solution, especially when dealing with large environments with several possible solutions.

3.6 GRAPH SEARCH APPROACHES

Graph search methods are other well-known approaches used to solve the path planning problem. Numerous graph search algorithms developed over the last decades have been evaluated for path planning of autonomous robots, for instance A star A∗, Dijkstra, breadth-first search (BFS), depth-first search (DFS).

3.6.1 The A Star (A*) Algorithm

Overview

The A∗ (A Star) algorithm is a path finding algorithm. It is an extension of Dijkstra's algorithm. A∗ achieves better performance (with respect to time), as compared to Dijkstra, by using heuristics. In the process of searching the shortest path, each cell in the grid is evaluated according to an evaluation function given by:

$$f(n) = h(n) + g(n) \qquad \text{Equation 1}$$

where g(n) is the accumulated cost of reaching the current cell n from the start position S:

$$g(n) = \begin{cases} g(S) = 0 \\ g(\text{parent}(n)) + \text{dist}(\text{parent}(n), n) \end{cases} \qquad \text{Equation 2}$$

h(n) is an estimate of the cost of the least path to reach the goal position G from the current cell n. The estimated cost is called heuristic. h(n) can be defined as the Euclidian distance from n to G. f (n) is the estimation of the minimum cost among all paths from the start cell S to the goal cell G. The Tie-breaking factor to Break multiplies the heuristic value (t Break ∗h(n)). When it is used, the algorithm favors a certain direction in case of ties. If we do not use tie-breaking, the algorithm would explore all equally likely paths at the same time, which can be very costly, especially when dealing with a grid environment. In a grid environment [7], the tiebreaking coefficient can be chosen as:

$$t\ Break = 1 + 1/(length(Grid) + width(Grid)) \qquad \text{Equation 3}$$

The algorithm relies on two lists: the open list which is a kind of a shopping list. It contains cells that might fall along the best path, but maybe not. Basically, this is a list of cells that need to be checked out. The closed list: It contains the cells already explored. Each cell saved in the list is characterized by five attributes: I D, parent Cell, g_Cost, h_Cost, and f _Cost. The search begins by expanding the neighbour cells of the start position S. The neighbour cell with the lowest f _Cost is selected from the open list, expanded, and added to the closed list. In each iteration, this process is repeated. Some conditions should be verified while exploring the neighbour cells of the current cell, and a neighbour cell is:

1. Ignored if it already exists in the closed list.

2. If it already exists in the open list, we should compare the g_cost of this path to the neighbour cell and the g_cost of the old path to the neighbour cell. If the g_cost of using the current cell to get to the neighbour cell is the lower cost, we change the parent cell of the neighbour cell to the current cell and recalculate g, h, and f costs of the neighbour cell.

This process is repeated until the goal position is reached. Working backward from the goal cell, we go from each cell to its parent cell until we reach the starting cell (the shortest path in the grid map is found).

A star Algorithm

Algorithm 1. The standard Astar Algorithm A^*

```
   input : Grid, Start, Goal
   // Initialisation:
 1 closedSet = empty set // Set of already evaluated nodes;
 2 openSet = Start // Set of nodes to be evaluated;
 3 came_from = the empty map // map of navigated nodes;
 4 tBreak = 1+1/(length(Grid)+width(Grid)); // coefficient for
   breaking ties;
 5 g_score[Start] = 0; // Cost from Start along best known
   path;
   // Estimated total cost from Start to Goal:
 6 f_score[Start] = heuristic_cost(Start, Goal);
 7 while openSet is not empty do
 8 |   current = the node in openSet having the lowest f_score;
 9 |   if current = Goal then
10 |   |   return reconstruct_path(came_from, Goal);
11 |   end
12 |   remove current from openSet;
13 |   add current to closedSet;
14 |   for each free neighbor v of current do
15 |   |   if v in closedSet then
16 |   |   |   continue;
17 |   |   end
18 |   |   tentative_g_score = g_score[current] + dist_edge(current, v);
19 |   |   if v not in openSet or tentative_g_score < g_score[v] then
20 |   |   |   came_from[v] = current;
21 |   |   |   g_score[v] = tentative_g_score;
22 |   |   |   f_score[v] = g_score[v] + tBreak * heuristic_cost(v, Goal);
23 |   |   |   if neighbor not in openSet then
24 |   |   |   |   add neighbor to openSet;
25 |   |   |   end
26 |   |   end
27 |   end
28 end
29 return failure;
```

Figure 20: A* algorithm

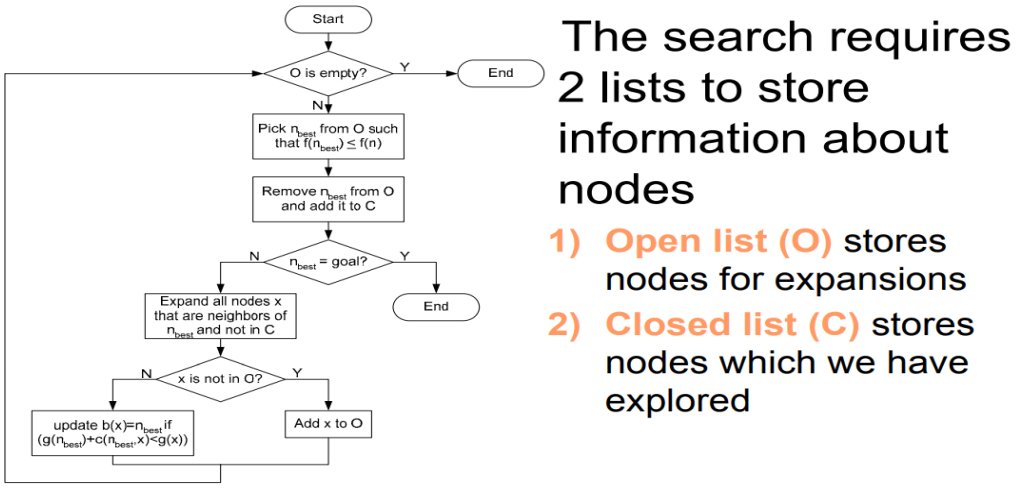


Figure 21: A* flowchart

A step by step A star example is presented in appendix **Astar**

3.6.2 Breadth First Search (BFS)

Overview

Breadth First search is known as an uninformed search because it does not use any information about how far the robot has travelled or how far the robot is from the goal. BFS begins at the starting position of the robot (root node) and begins looking for the goal by expanding all the successors of the root node.

The successors of a node are all allowable directions that the robot could travel next. Allowable means that directions causing the robot to crash into an obstacle, to move outside of the workspace will not be considered as successors of the node. Nodes that have already been visited by the robot will not be considered successors either. If those successors are not the goal, then BFS expands each of those nodes, and this loop continues until the goal node is reached. It is much easier to visualize BFS if we convert this scenario into a tree and search the tree with a fringe that is FIFO (first in first out) queue.

The queue is an array of expanded nodes from which we use to determine which node is to be expanded next. A FIFO queue means that the node that has been sitting on the queue for the longest amount of time is the next node to be expanded. The FIFO queue continues until the goal node is found. When found, the path leading to the goal node is traced back up the tree which maps out the directions that the robot must follow to reach the goal.

The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories:

- Visited and
- Not visited.

A Boolean visited array is used to mark the visited vertices. For simplicity, it is assumed that all vertices are reachable from the starting vertex. BFS uses a queue data structure for traversal.

There are many ways to traverse graphs. BFS is the most used approach.

Breadth first search is complete if the branching factor is finite. This means that the algorithm will always return a solution if a solution exists. In a path planning sense, breadth first search will always find the path from the starting position to the goal if there is an actual path that can be found.

Breadth first search is only optimal if the path cost is the same for each direction. In this case, the path cost would be the direction and optimal means the shortest distance path from the start to the goal.

BFS Algorithm

Every traversable neighbour cell is added to an array which is called OPEN LIST. OPEN LIST is the array of neighbour cells which must be reviewed in order to find the goal cell. OPEN LIST elements are reviewed if one is the goal cell or not. The steps of breadth-first algorithm can be listed as follows:

1. Define the starting and goal cells.
2. Load the map matrix.
3. Add the starting cell to OPEN LIST.
4. Add the neighbour cells to OPEN LIST.
5. If OPEN LIST is empty, no path.
6. If goal cell is added to OPEN LIST, define the PATH using map matrix. Else compute the cost of neighbour cells.
7. Pull out the reviewed cells from OPEN LIST. viii. Go to step iv.

A step by step BFS example is presented in appendix **BFS**

3.6.3 Depth First Search (DFS)

Overview

Depth first search is the complement to breadth first search. It is also an uninformed search method. Starting from the root node it expands all the successors of the start node

same as BFS. However, after the first node, DFS always expands the last successor added. The first node is expanded just like BFS and then the second node is expanded. These two steps look like BFS because there was only one successor to the start node. If DFS gets to a point where it cannot find any more successors for the node it is currently expanding, it will then move to the newest discovered node that has not been expanded.

As with BFS, DFS is easier to understand if we look at a tree and use a LIFO (last-in-first-out) stack. The stack contains the list of discovered nodes. The most recent discovered node is put on top of the LIFO stack. The next node to be expanded is then taken from the top of the stack and all its successors are added to the stack.

DFS Algorithm

Step 1: Create a stack with the total number of vertices in the graph as the size.

Step 2: Choose any vertex as the traversal's beginning point. Push a visit to that vertex and add it to the stack.

Step 3 - Push any non-visited adjacent vertices of a vertex at the top of the stack to the top of the stack.

Step 4 - Repeat steps 3 and 4 until there are no more vertices to visit from the vertex at the top of the stack.

Step 5 - If there are no new vertices to visit, go back and pop one from the stack using backtracking.

Step 6 - Continue using steps 3, 4, and 5 until the stack is empty.

Step 7 - When the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges.

A step by step DFS example is presented in appendix **DFS**

3.6.4 Dijkstra path planning

Overview

In graph theory, DJ algorithm is one of the classic algorithms to find the shortest path. For a connected weighted graph with n vertices, the time complexity of only source

shortest path algorithm is O (n^2), which can reduce the time complexity of DJ algorithm by using more efficient data structure.

DJ algorithm belongs to the algorithm of finding the shortest path from a single source point, and the result is the shortest path from each node to the source point. In the process of node expansion, there are two important operations: one is to relax each node; the other is to find a node with the smallest distance from the source point from the OPEN list as a new expansion node in each iteration, and delete it from the OPEN list as an expanded node and insert it into the CLOSE list. In this process, it is the most time-consuming to extract the node with the smallest distance from the OPEN list to the source point. In order to reduce the time consumption of DJ algorithm, we can design a new node expansion mechanism to avoid the above two operations [7].

On the basis of environment modelling based on grid method, the algorithm adopts four neighbourhood search. Assuming that point s is the starting point, we can see that the distance from the four nodes extending from point s to s is 1, and then add the four nodes to the OPEN list respectively. Then, select a node from the OPEN list with the smallest distance to s point. At this point, it can be found that each node can be used as the next expansion node, so it will be redundant to select the node with the smallest distance to s point from the OPEN list. Therefore, four nodes can be regarded as the next expansion nodes in a certain order, that is, the breadth first search algorithm can be used to expand the nodes.

Dijkstra's algorithm is a classic algorithm for finding the shortest path between two points due to its optimisation capability. The adjacency matrix is the naive storage structure of the algorithm. This storage structure has limited the use of the algorithm as it expands large storage space [7].

Dijkstra algorithm

This algorithm is like Breadth-first algorithm but adds the computation of different cost cells (not only the shortest path but also the lowest cost path). In this algorithm, again the neighbour cell array OPEN LIST exists. Like the previous algorithm here the steps are:

1. Define the starting and goal cells.

2. Load map matrix.
3. Add the starting cell to OPEN LIST.
4. Add the neighbour cells to OPEN LIST compute the costs, record their parent cell to PARENTS.
5. If OPEN LIST is empty, no path.
6. If goal cell is added to OPEN LIST define the PATH using PARENTS matrix. Else go on.
7. If neighbour cell is added OPEN LIST before finding its new cost and compare to its old cost. If it is lower, update the cost and PARENTS matrix.
8. Pull out the reviewed cells from OPEN LIST.
9. Go to step iv.

A step by step Dijkstra example is presented in appendix **Dijkstra**

3.7 HEURISTIC APPROACHES

Classical methods dominated the field of robot path planning before 2000 but had much lower percentage of usage after that year. Another category of approaches has been found more popular in the robot navigation field compared to classical techniques. This category was designed to overcome the limit of the classical and exact methods, and they are called metaheuristics [23]. The metaheuristics "as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions,".

It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low)-level procedures, or a simple local search, or just a construction method. This category of algorithms includes but is not limited to ant colony optimization (ACO), particle swarm optimization (PSO), genetic algorithm (GA), iterated local search (ILS), simulated annealing (SA), Tabu

Search (TS), neural network (NN), The greedy randomized adaptive search procedure (GRASP), and variable neighbourhood search (VNS) [1].

3.7.1 Rapidly exploring Random Tree (RRT) Path Planning

Overview

Another well-known searching technique is the RTT search family. RTT is a blind search technique which means it searches for the goal with no prior knowledge of its position. RRT is simple, a random point is generated from the current position. Each time a point is generated the algorithm checks if it reached the goal or not. When the goal point is reached the algorithms returns its tree back to the origin. But RRT has some limitations:

RTT is controlled by the number of iterations given by the user. If the number of iterations is reached before reaching the algorithm just return the tree of the last iteration. Since the path is controlled mainly by random point, it's not guaranteed that the path created is the best path the vehicle can take length wise.

The path doesn't take in consideration the dimensions of the vehicle. If the space the path generated is not wide enough for the vehicle to path through the path planner would not avoid it.

RTT can't operate in large areas as it needs an enormous number of iterations which takes up memory space more than given and operates slowly. For the past years, developers have been trying to optimize RTT searching technique to solve these problems.

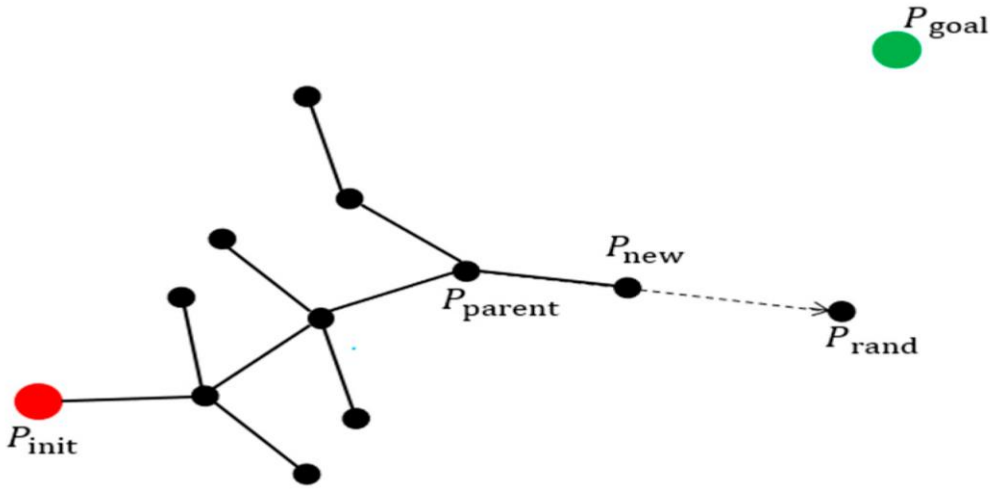


Figure 22: RRT

RRT Algorithms

Algorithm 1 Base RRT

Input:
Initial configuration q_{start} q_{goal} *obstacle*
Number of sampling point K
Step size δ

Output:
RRT graph G
Vertices of tree *nodes*

```
Initialize all Parameters ;
nodes = q_start;
for i = 1 to K do
    q_rand = Sample();
    q_near ← Nearest(nodes, q_rand);
    q_new ← Steer(q_near, q_rand, δ);
    if ObstacleFree( q_near, q_new) then
        nodes.add(q_new);
        return Advanced;
if dist(q_new − q_goal) < Error then
    return Reached;
else
    return Trapped
final ;
return Graph;
```

Figure 23: RRT Algorithm

A step by step RRT example is presented in appendix **RRT**

3.7.2 RRT Star Path Planning

Overview

RTT star initially operates as RTT at the beginning of the process. It randomly creates connected points that spread out through the map from the start point to the goal. While constantly checking if the nodes reached or not. But one of the main RTT problems was that it creates a very random path that most of the time is not optimal. The path created can be very unnecessary long and twisted and holds up precious processing time and memory.

RRT* was an important advance in optimum path planning for high-dimensional problems. RTT star is an optimized algorithm of the RTT. Where the nodes are shifted, deleted to save space and iterations. The RTT star tree is created by evaluating each new node and finding a better connection for it.

RRT star tree optimization steps:

- First a random point is generated newly for the tree to expand in the map.
- Then all the near points from other trees are selected as an optional parent
- The distance between each parent and the new node is calculated to choose best parent node.
- Unnecessary points or lines are deleted because a better fit is chosen.
- Repeat all steps till goal is found.

RRT* Algorithm

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$ **then** $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Figure 24: RRT* Algorithm

A step by step RRT* example is presented in appendix **RRT Star**

3.8 PATH PLANNING SUMMARY

Table 4: A star pros and cons

Advantages	Disadvantages
It is an optimal search algorithm in terms of heuristics. It is one of the best heuristic search techniques. It is used to solve complex search problems. There is no other optimal algorithm guaranteed to expand fewer nodes than A*	This algorithm is complete if the branching factor is finite, and every action has a fixed cost. The performance of A* search is dependent on accuracy of heuristic algorithm used to compute the function h(n).

Table 5: BFS pros and cons

Advantages	Disadvantages
The solution will be found by BFS If there is some solution. BFS will never get trapped in a blind alley, which means unwanted nodes. If there is more than one solution then it will find a solution with minimal steps.	Memory Constraints As it stores all the nodes of the present level to go for the next level. If a solution is far away, then it consumes time

Table 6: DFS pros and cons

Advantages	Disadvantages
The memory requirement is Linear WRT Nodes. Less time and space complexity rather than BFS. The solution can be found out without much more search.	Not Guaranteed that it will give you a solution. Cut-off depth is smaller, so time complexity is more. Determination of depth until the search has proceeded.

Table 7: Dijkstra pros and cons

Advantages	Disadvantages
Uninformed algorithm Dijkstra is an uninformed algorithm. This means that it does not need to know the target node beforehand. For this reason, it's optimal in cases where you don't have any prior knowledge of the graph when you cannot estimate the distance between each node and the target. Good when you have multiple target nodes Since Dijkstra picks edges with the smallest cost at each step it usually covers a large area of the graph. This is especially useful when you have multiple target nodes but you don't know which one is the closest.	Fails for negative edge weights If we take for example 3 Nodes (A, B and C) where they form an undirected graph with edges: AB = 3, AC = 4, BC=-2, the optimal path from A to C costs 1 and the optimal path from A to B costs 2. If we apply Dijkstra's algorithm: starting from A it will first examine B because it is the closest node. and will assign a cost of 3 to it and therefore mark it closed, which means that its cost will never be reevaluated. This means that Dijkstra's cannot evaluate negative edge weights.

Table 8: RRT pros and cons

Advantages	Disadvantages
The RRT algorithm can detect and evaluate obstacles in real time. The RRT algorithm is efficient. RRT algorithm still has superior performance in high dimensional space. RRT algorithm is admirably adapted to dynamic environments	The study of the RRT algorithm is random. The path generated by the RRT algorithm does not guarantee the best solution. Even under the same laboratory conditions, the RRT algorithm is still unstable The RRT algorithm must compare the distance between random and extended nodes. The overall efficiency of the algorithm will be reduced

Table 9: RRT* pros and cons

Advantages	Disadvantages
The main advantages of RRT are simplicity (implementation is straightforward and only needs simple computations), fast convergence and expansion toward unexplored regions of C-Space, and probabilistically completeness	The study of the RRT* algorithm is random.

3.9 EXPERIMENTAL RESULTS

3.9.1 A Star

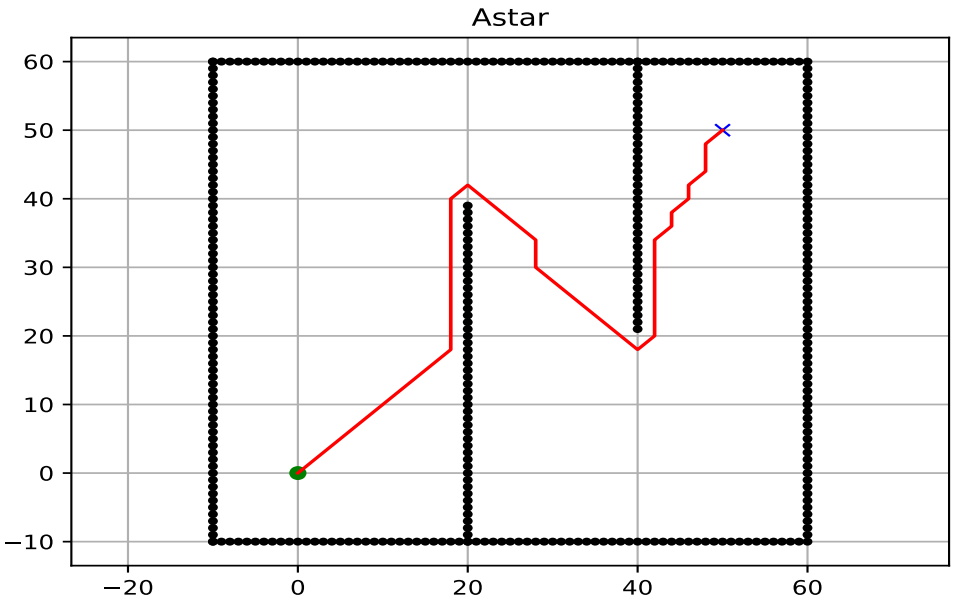


Figure 25: A star

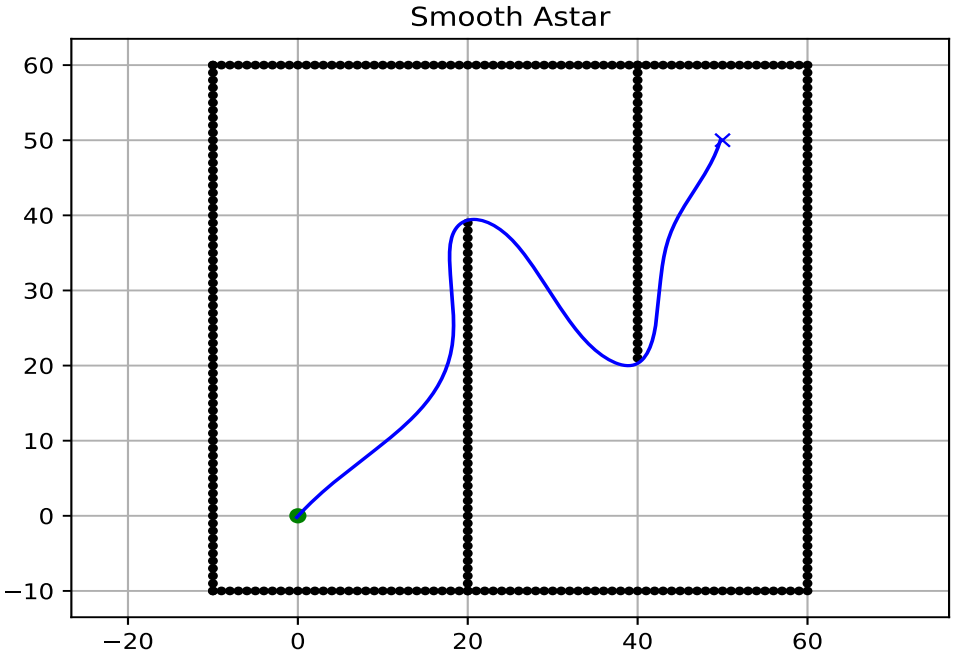


Figure 26: Smoothed A star

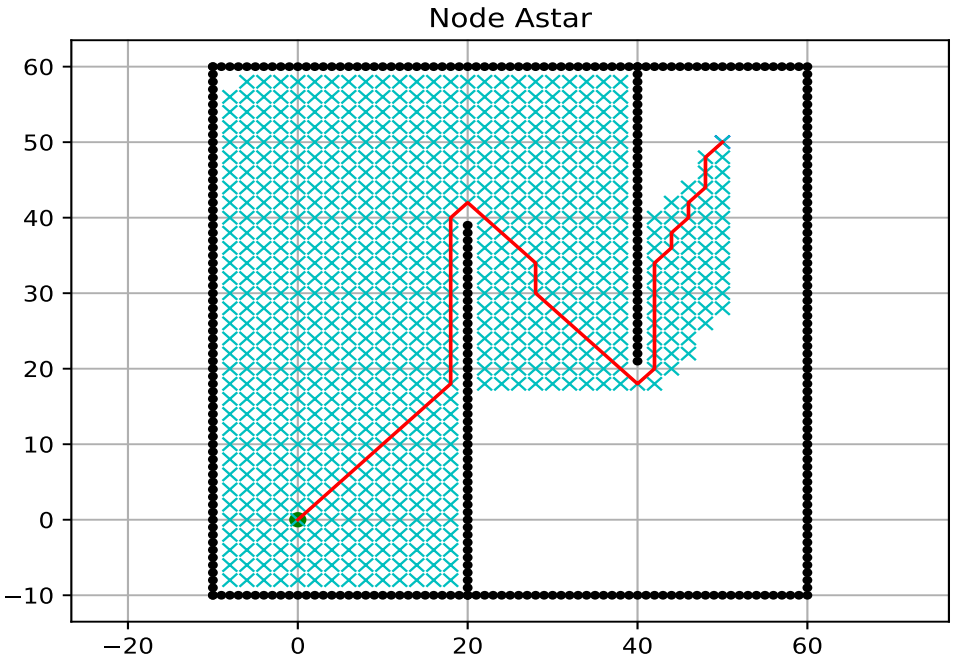


Figure 27: Node A star

3.9.2 BFS

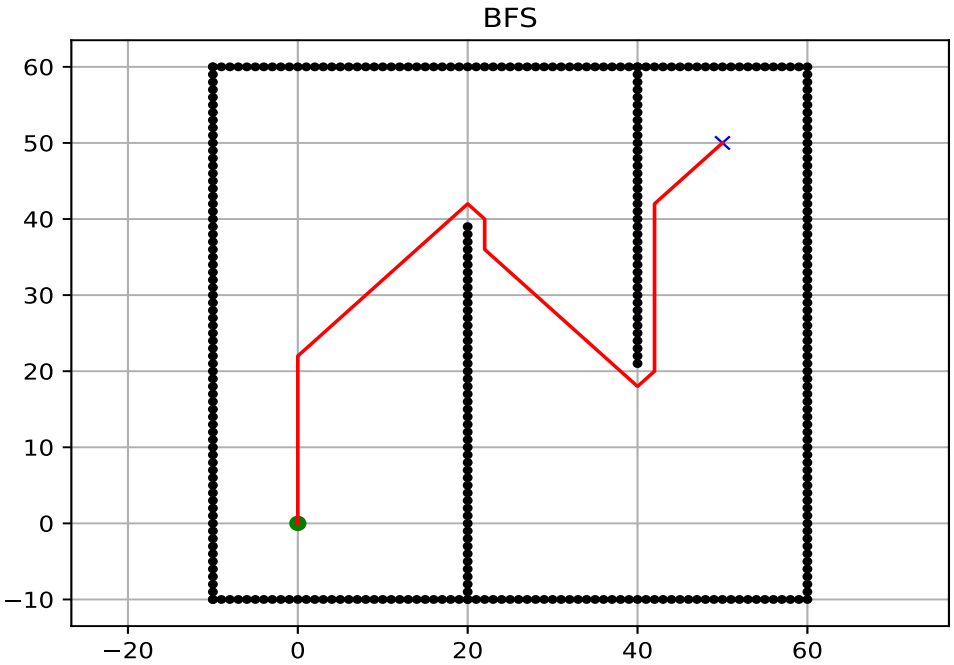


Figure 28: BFS

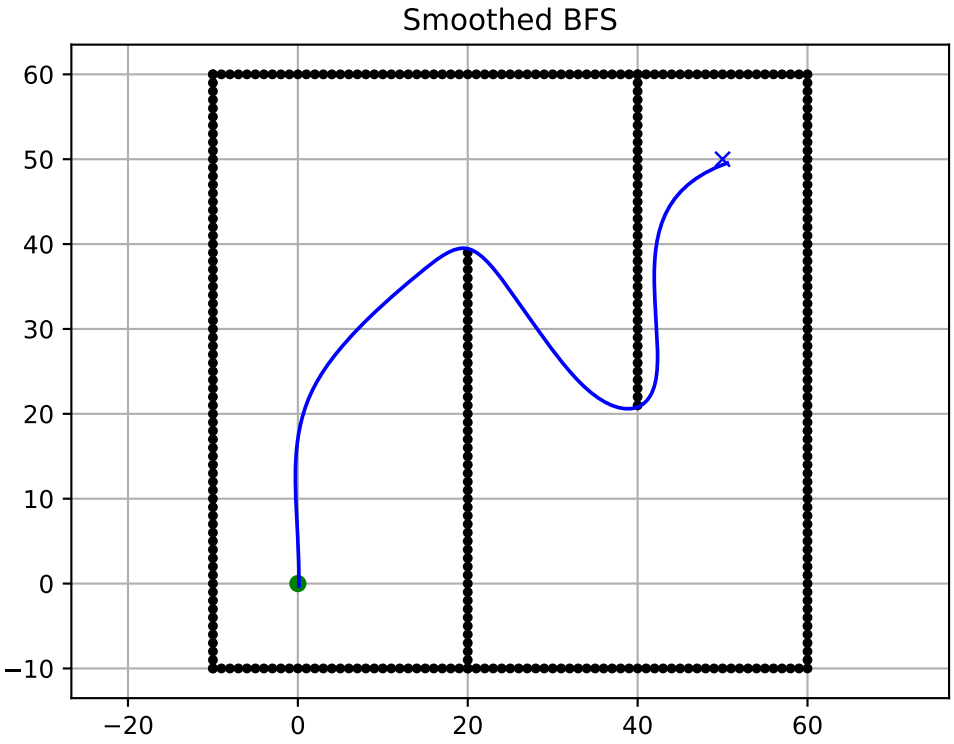


Figure 29: Smoothed BFS

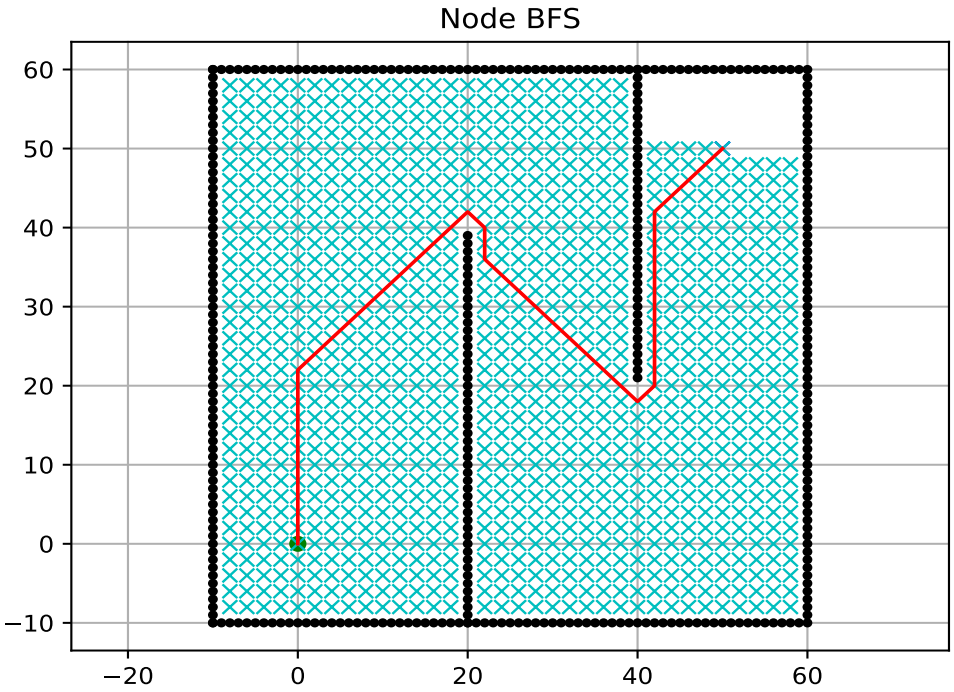


Figure 30: Node BFS

3.9.3 DFS

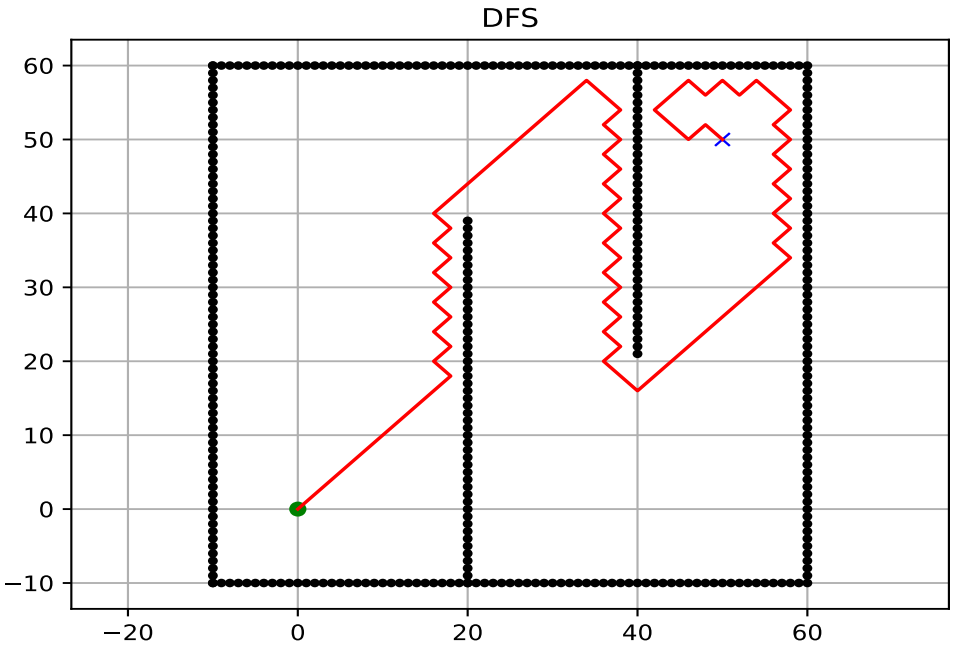


Figure 31: DFS

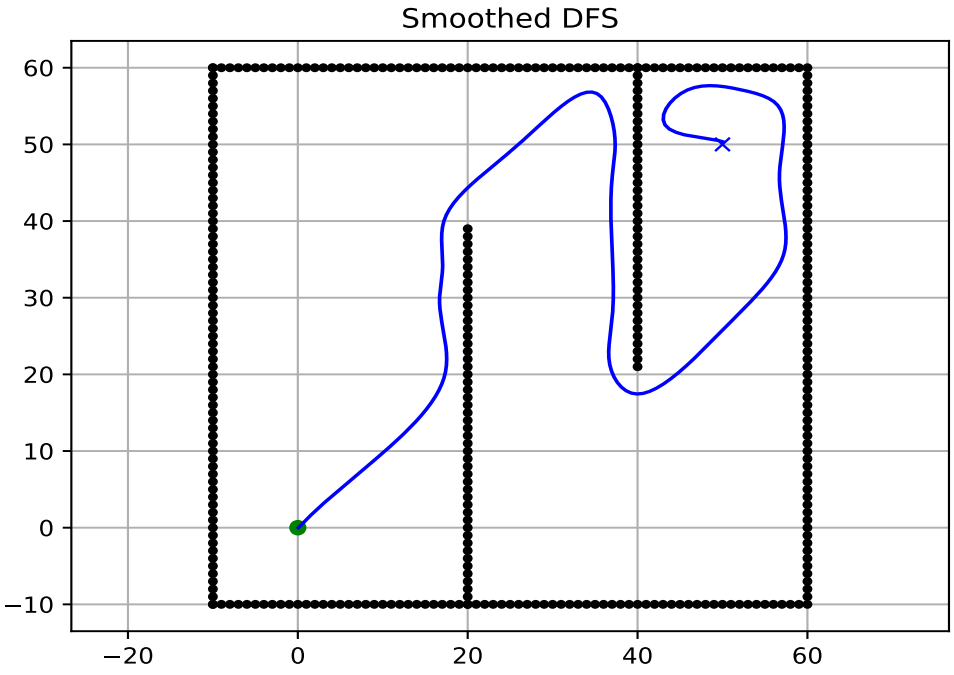


Figure 32: Smooth DFS

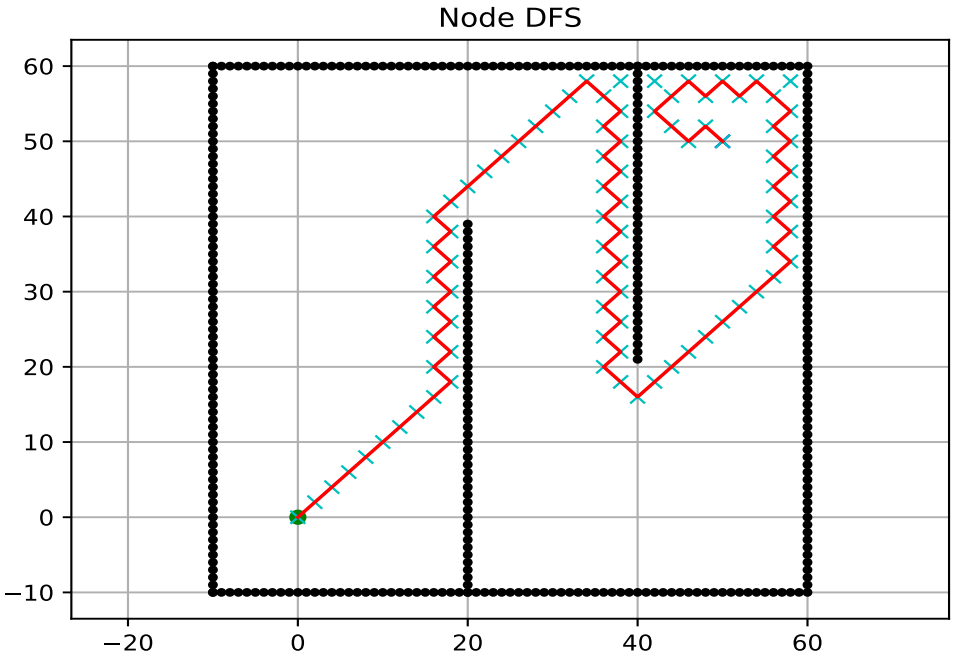


Figure 33: Node DFS

3.9.4 Dijkstra

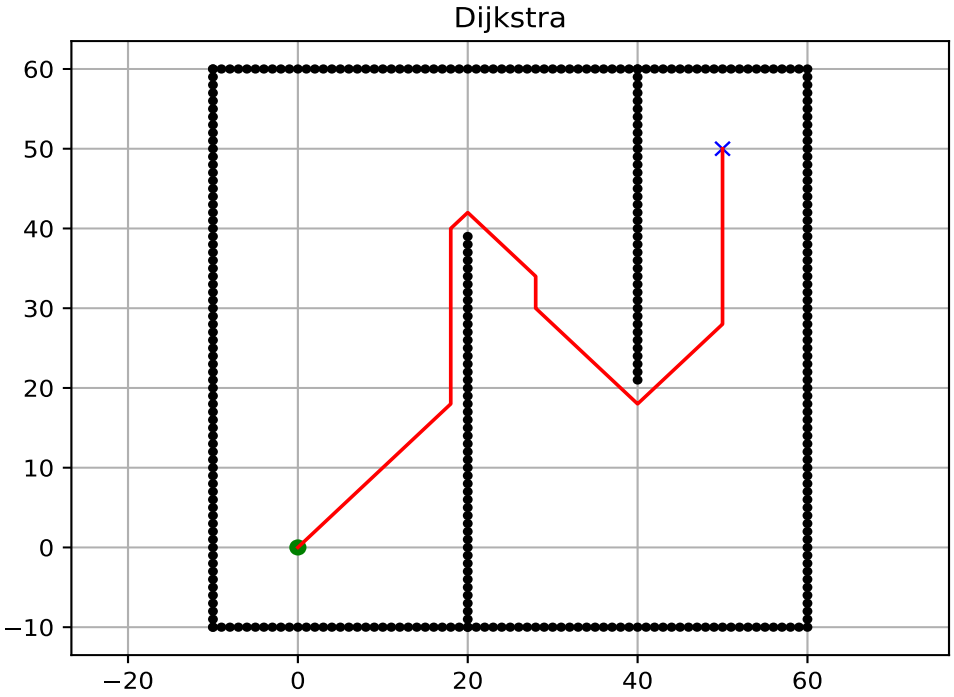


Figure 34: Dijkstra

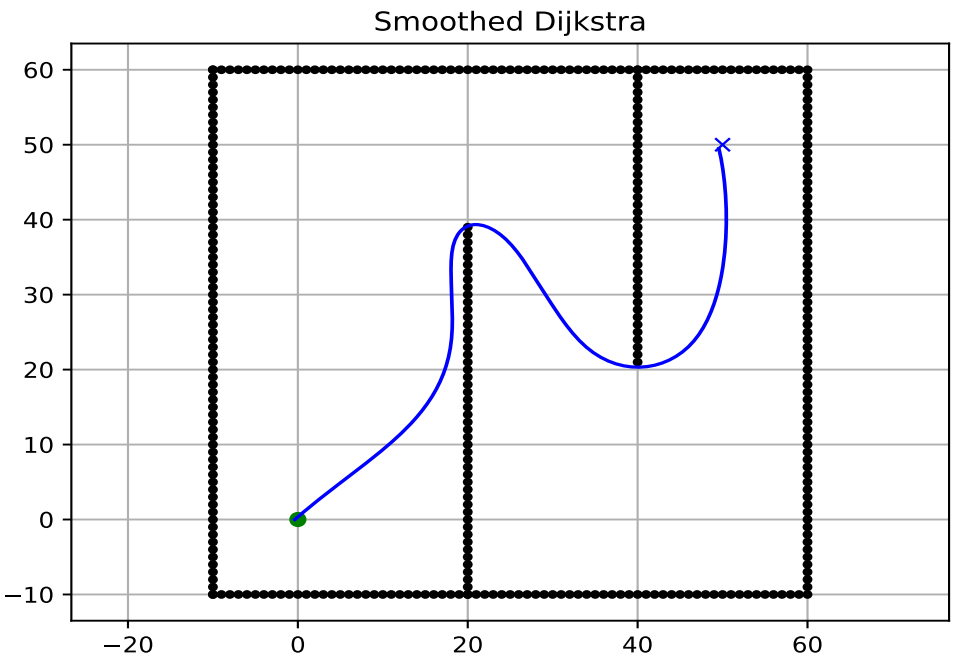


Figure 35: Smooth Dijkstra

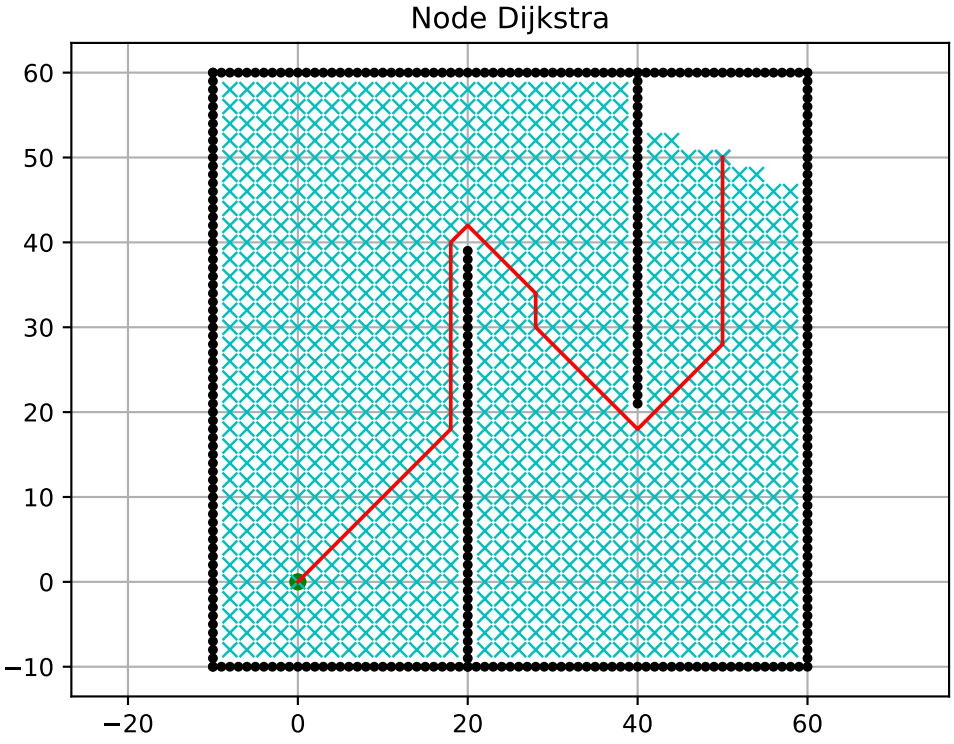


Figure 36: Node Dijkstra

3.9.5 RRT

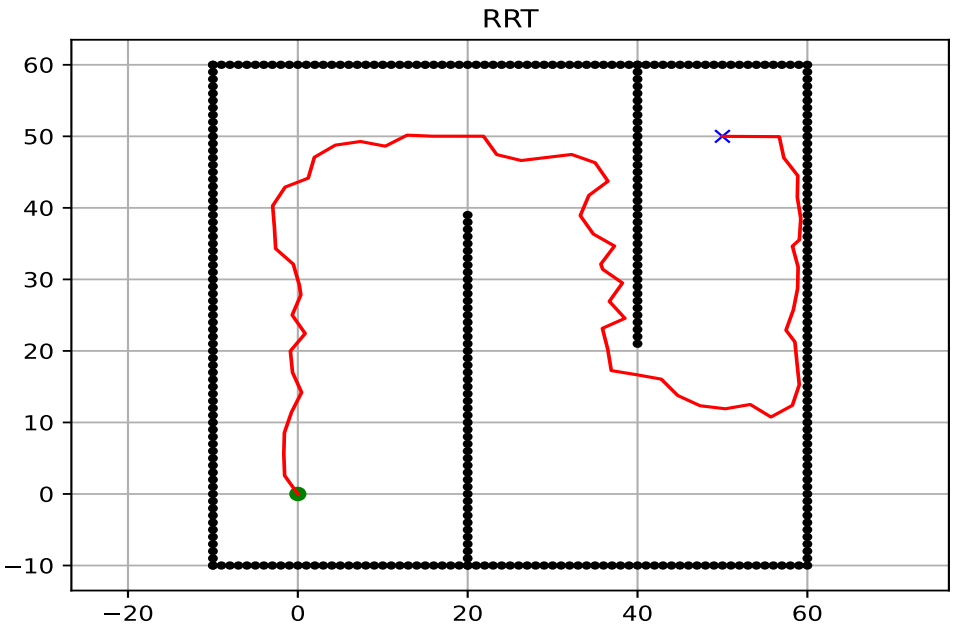


Figure 37: RRT

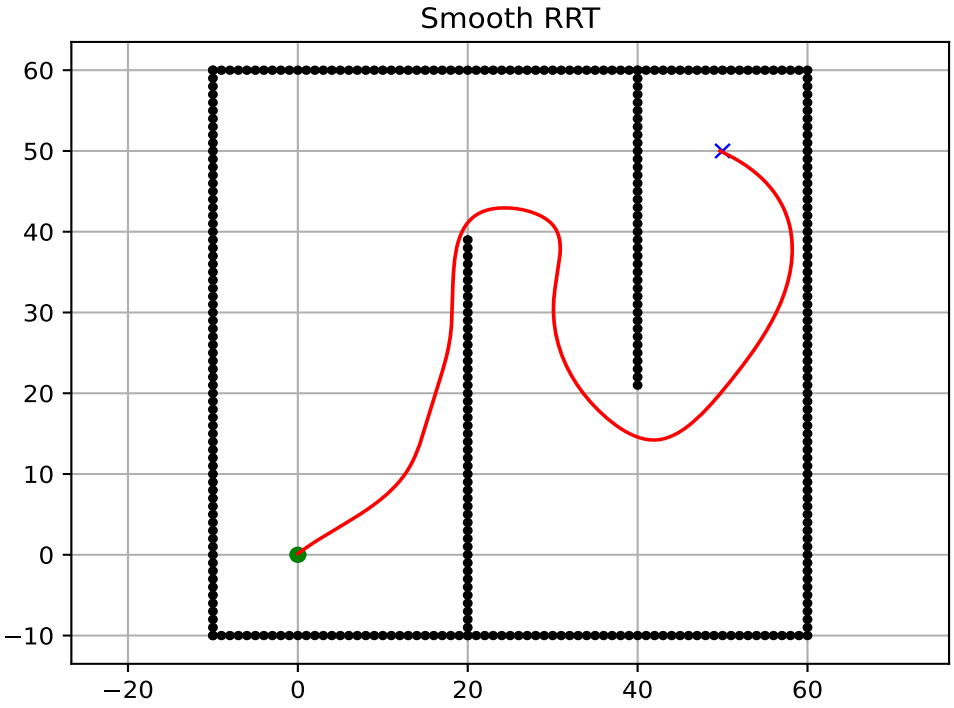


Figure 38: Smooth RRT

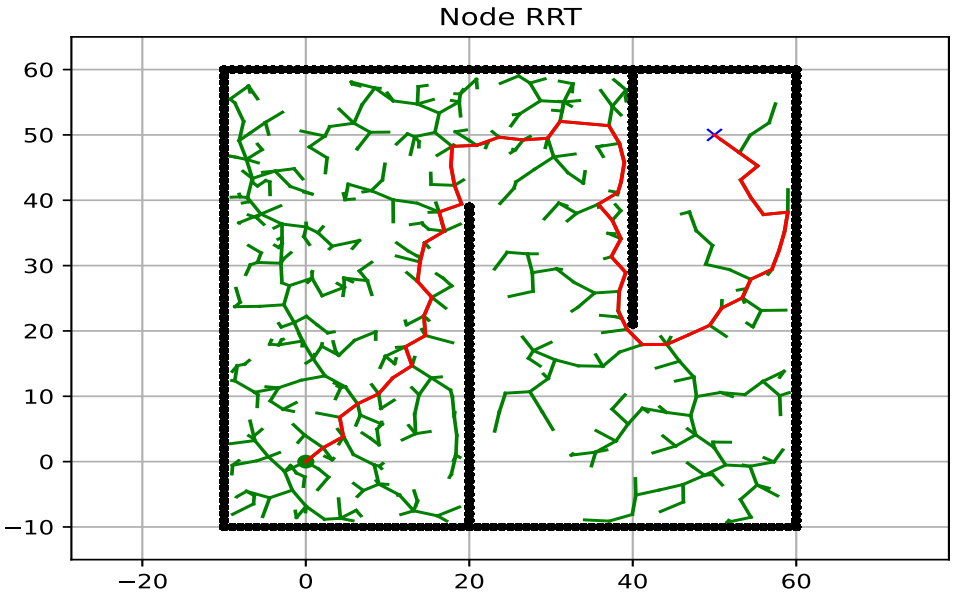


Figure 39: Node RRT

3.9.6 RRT Star

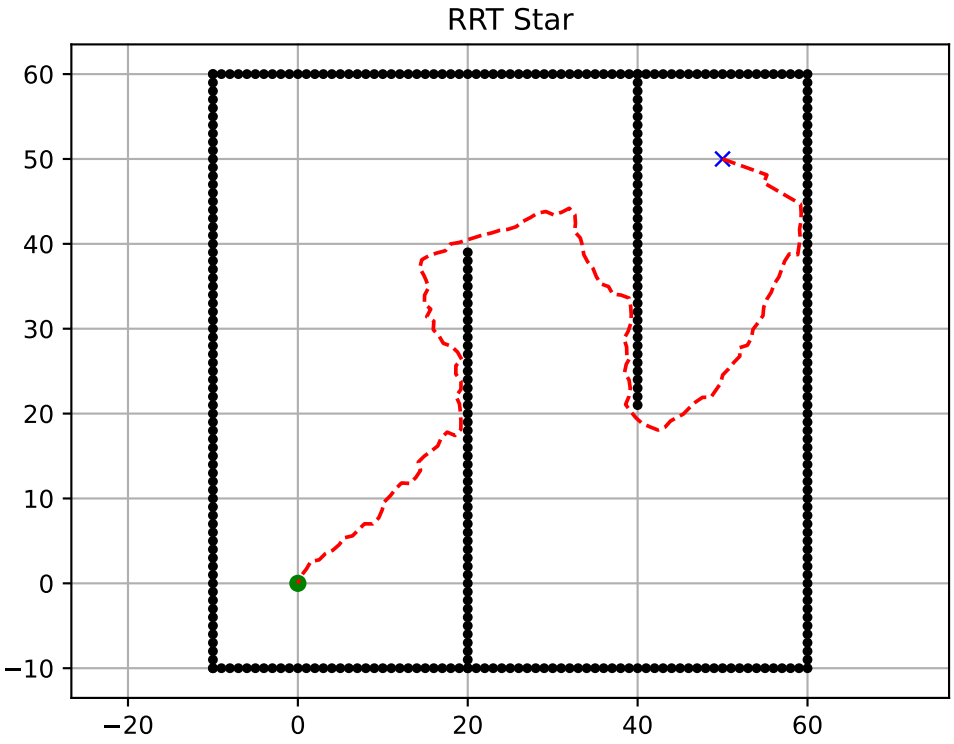


Figure 40: RRT*

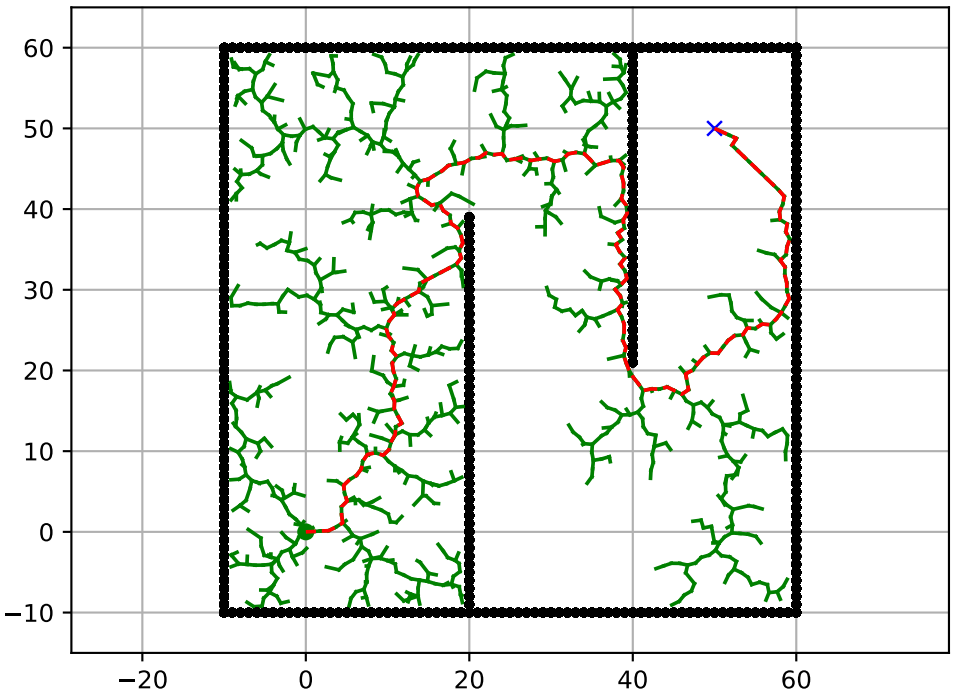


Figure 41: Node RRT*

3.9.7 Path Length

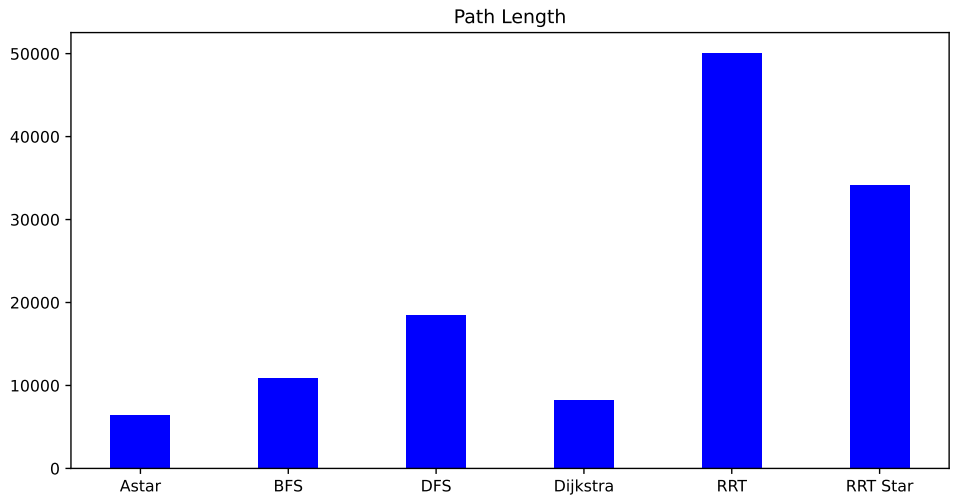


Figure 42: Path Length

3.9.8 Computational Time

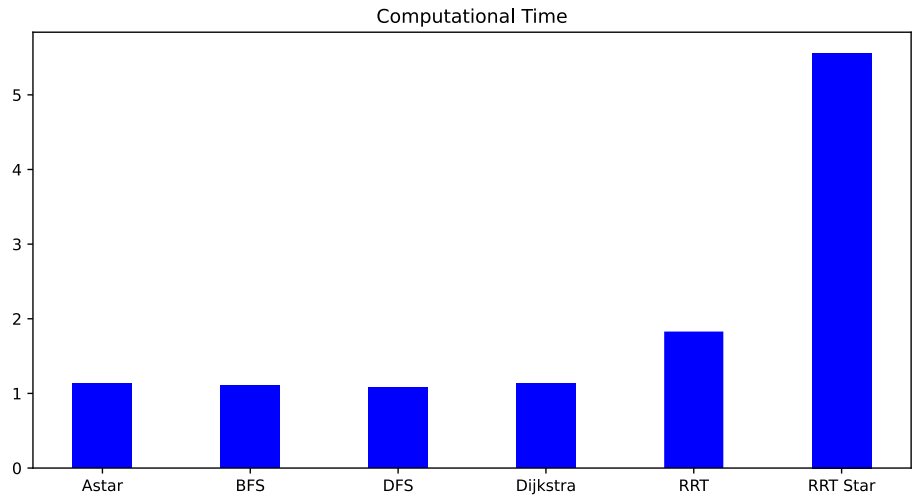


Figure 43: Computational Time

3.10 CONCLUSION

We need to find answers to the three following questions: Where am I? Where am I going? How do I get there? These three questions are answered by the three fundamental navigation functions localization, mapping, and motion planning, respectively.

We answered where am I? in the last chapter **Chapter two: Slam.** Now we answered the question of How do I get there? Using the discussed algorithms.

Every algorithm of the discussed algorithms which are A star, BFS, DFS, Dijkstra, RRT, and RRT star has it is advantages and disadvantages as discussed in **3.8**. what we do is using our experimental result we choose the algorithms that fits our need.

Based on our experimental result, we were able to get the shortest path with A star algorithm, with small computational time making it the best algorithms for our parking task. Using b spline smoothing method we were able to enhance the path to fit our car.

After determining the path we will follow we need to determine the slot in which we will park in. we will do this task using a camera and vision.

CHAPTER FOUR: VISION

Our project depends on the vision to interface with the world. More specifically, it will capture continuous image streams and extract features from these images. One of the most important features is the parking slot. Parking slots can be divided into two categories, empty slots that we can park in and non-empty slots. Computer Vision is the core of Artificial Intelligence technology. AI helps computers to decode and understand the visual data acquired from various sources. It involves automatic visual understanding using AI algorithms. The best example of computer vision is autonomous vehicles. Autonomous vehicles use sensor technology to identify people, cars, and other objects on the road.

4.1 CAMERA TYPES:

4.1.1 Raspberry Pi Camera:

The Raspberry Pi Camera Board is a custom-designed add-on module for Raspberry Pi hardware. It attaches to Raspberry Pi hardware through a custom CSI interface. The sensor has a 5-megapixel native resolution in still capture mode. In video mode, it supports capture resolutions up to 1080p at 30 frames per second. The camera module is lightweight and small making it an ideal choice for mobile projects.

Figure 44: Raspberry Pi Camera

4.1.2 Real sense Camera:

Intel RealSense Technology is a product range of depth and tracking technologies designed to give machines and devices depth perception capabilities. The technologies, owned by Intel are used in autonomous drones, robots, AR/VR, and smart home devices amongst many other broad-market products.

Intel RealSense depth cameras give highly accurate dimensional data that can be used to measure objects of any size or shape with ease.

Figure 45: Intel Real sense Camera

4.1.3 Kinect camera:

Kinect is a line of motion-sensing input devices produced by Microsoft and first released in 2010. The devices generally contain RGB cameras, infrared projectors, and detectors that map depth through either structured light or time of flight calculations, which can in turn be used to perform real-time gesture recognition and body skeletal detection, among other capabilities. They also contain microphones that can be used for speech recognition and voice control. The depth and motion sensing technology at the core of the Kinect is enabled through its depth-sensing. The original Kinect for Xbox 360 used structured light for this: the unit used a near-infrared pattern projected across the space in front of the Kinect, while an infrared sensor captured the reflected light pattern. The light pattern is deformed by the relative depth of the objects in front of it, and mathematics can be used to estimate that depth based on several factors related to the hardware layout of the Kinect.

The software is what makes the Kinect a breakthrough device. Developers for the Kinect gathered an incredible amount of data regarding motion capture of actual moving things in real-life scenarios. Because the Kinect has an infrared projector, infrared camera, and color camera, it's a great imaging tool, even for robots. To enhance the range and autonomous nature of robots, they need to be able to see the environment around them. One way they do this is through simultaneous localization and mapping, or SLAM.

Figure 46: Kinect Camera

After reviewing the three types of cameras that are available to use in our project, the next table shows a comparison between them based on this comparison we used the **Kinect** for our project.

Table 10: Camera Comparisons

Pi Camera	Kinect Camera	Realsense Camera
Relatively, low-quality image	High-quality	High-quality
No depth sensor	Depth sensor (IR) available	Depth sensor available
Small and light. 2.5*2.4*0.9 cm	Large size 27.94*2.54*2.54 cm	Relatively small in size. 8.89*2.54*2.54 cm
Un-reliable	Reliable	Reliable
Low input power	High input power	Relatively low input power
Has many sources and libraries.	Very few sources and libraries.	Relatively has sources and libraries.
Relatively, very low cost. 400^{EGP}	Medium cost. 3200^{EGP}	Very expensive. 12000^{EGP}
Fixed	Can change FOV using a built-in motor	Fixed
-		-

4.2 LIBRARIES AND DRIVERS:

In order to set up the camera and use it by Ubuntu, we used the FREENECT library which contains all drivers for the camera like RGB, depth, and the microphone.

The main advantage of using this revision over the system package is:

- The ability to search for a Kinect through serial number is necessary for running multiple Kinects.
- USB 3.0 Support

libfreenect is a user space driver for Microsoft Kinect. It runs on Linux, OSX, and Windows and supports:

- RGB and Depth Images
- Motors
- Accelerometer
- LED
- Audio

4.2.1 Library setup:

This section is for the steps that we follow to setup the freenct library.

```
git clone https://github.com/OpenKinect/libfreenect
cd libfreenect
mkdir build.
cd build
cmake -L . # -L lists all the project options
make
cmake . -DBUILD_PYTHON=ON
make
cmake . -DCMAKE_BUILD_TYPE=debug
cmake .. -DBUILD_CPACK_DEB=ON -DBUILD_CPACK_RPM=ON -DBUILD_CPACK_TGZ=ON

cpack
```

Here's some output of the camera after installing the library:

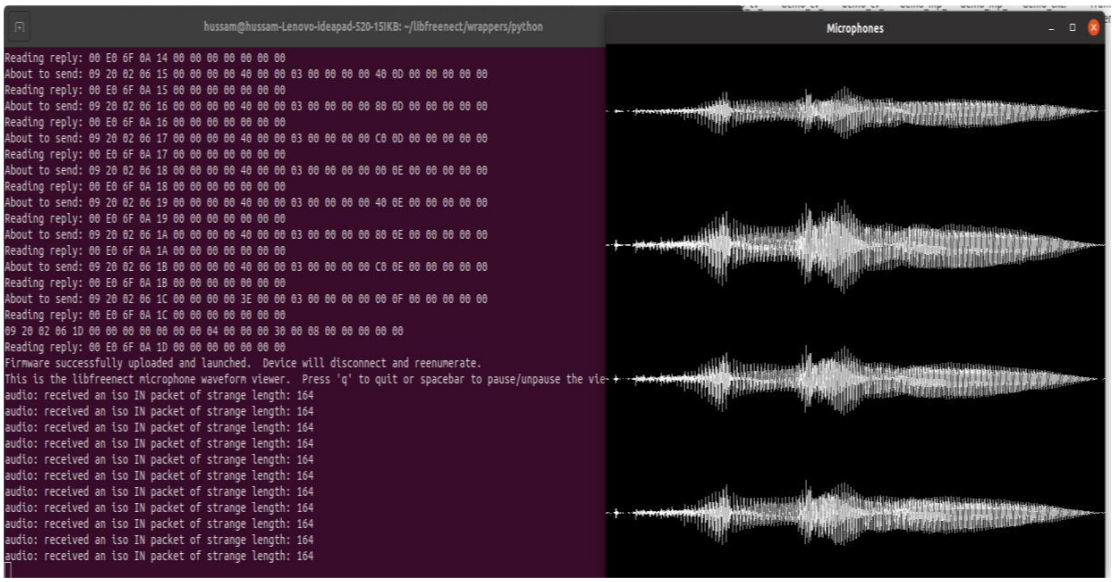


Figure 47: Mic View

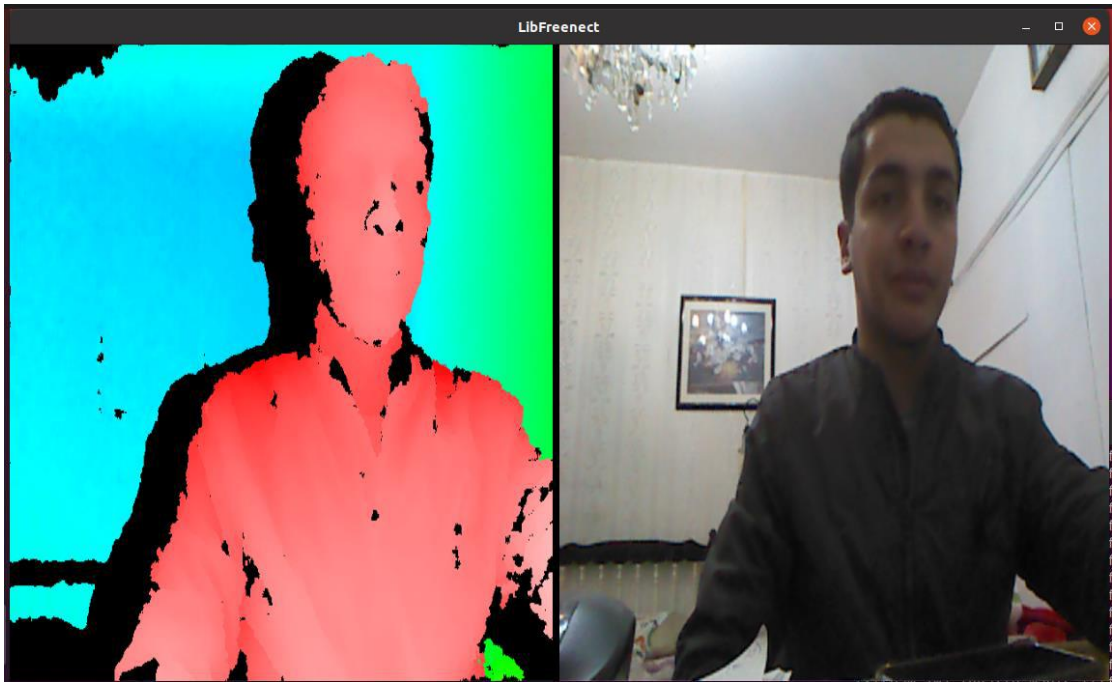


Figure 48: Depth and RGB output

In order to connect the output stream from the camera to the model we need to write a python code that we can merge it with the model.

```
import freenect
import cv2

if __name__=="__main__":
    while True:
        rgb,_=freenect.sync_get_video()
        rgb=cv2.cvtColor(rgb,cv2.COLOR_RGB2BGR)
        cv2.imshow("rgb_img",rgb)
        if cv2.waitKey(1)& 0xFF== ord('q'):
            break
    cv2.destroyAllwindows()
```

Figure 49: Python code for the Kinect

4.3 OBJECT DETECTION:

Object detection is the task of detecting objects of a certain class in an image. The methods can be categorized into two main types: one-stage methods and two stage-methods. One-stage methods prioritize inference speed, and example models include YOLO and SSD. Two-stage methods prioritize detection accuracy, and example models include Faster R-CNN, Mask R-CNN and Cascade R-CNN.

4.3.1 Yolo Algorithm:

we use yolo to detect our project because we notice that its accuracy is good enough reach to 98% in our model and faster than other techniques. YOLO is an algorithm that uses neural networks to provide real-time object detection. YOLO is an abbreviation for the term ‘You Only Look Once’. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images [8].

The YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

This means that prediction in the entire image is done in a single algorithm run. CNN is used to predict various class probabilities and bounding boxes simultaneously.

YOLO algorithm is important because of the following reasons:

- **Speed:** This algorithm improves the speed of detection because it can predict objects in real-time.
- **High accuracy:** YOLO is a predictive technique that provides accurate results with minimal background errors.
- **Learning capabilities:** The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

Yolo Format:

In YOLO labeling format, a .txt file with the same name is created for each image file in the same directory. Each .txt file contains the annotations for the corresponding image file, that is object class, object coordinates, height and width. This first number in the row means it belongs to class 0 , then numbers means x small, y small ,x large , y large.

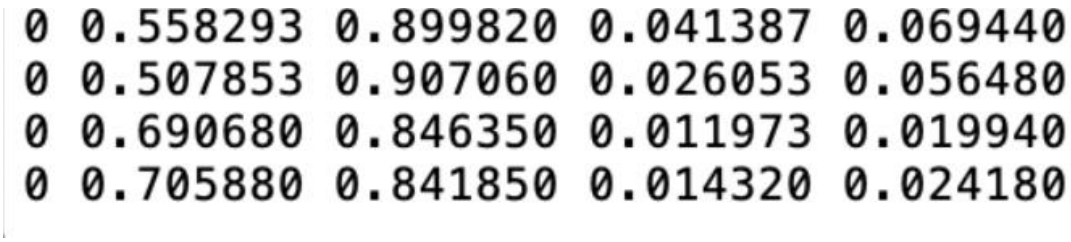


Figure 50: Yolo Format

4.3.2 CNN:

Most computer vision algorithms use something called a convolution neural network, or CNN. A CNN is a model used in machine learning to extract features, like texture and edges, from spatial data.

Like basic feedforward neural networks, CNNs learn from inputs, adjusting their parameters (weights and biases) to make an accurate prediction. However, what makes CNNs special is their ability to extract features from images [9].

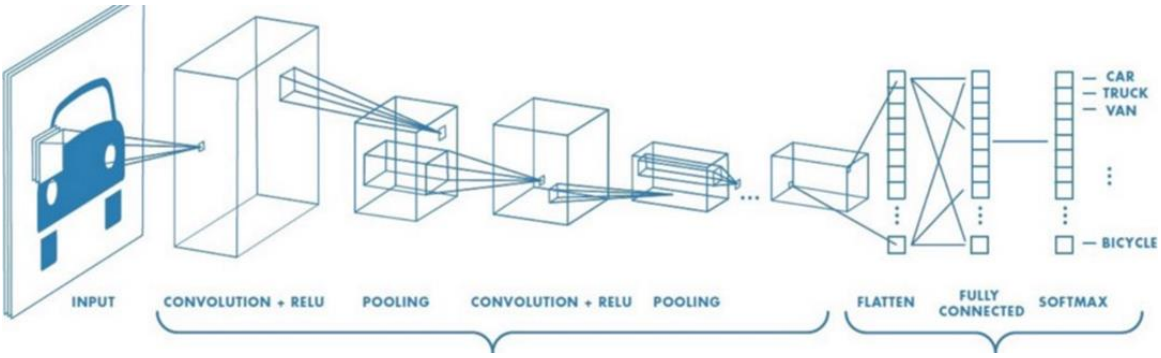


Figure 51: CNN Model

Take an image of a car, for example. In a normal feedforward neural network, the image would be flattened into a feature vector. However, CNNs can treat images like matrices as they exist and extract spatial features from them, like texture, edges and depth. They do this by using convolutional layers and pooling.

- Convolutional Layers

the convolutional layer applies a series of image filters to an input image represented as a matrix.

- Max Pooling

if we had many convolution kernels (meaning many feature maps), the data would have a lot of dimensions. we use max pooling to reduce the dimensionality.

4.4 CLASSIFICATION:

After detection, we go through classification mission. We apply Filters to our data set to learn features about empty and non-empty slot by converting our data images into matrices that have RGB values and go through the network in the final layer we have two nodes of our classification one for empty and one for non-empty.

The hardest problem I encountered in this chapter was to find ready-made dataset [10] to pass to the Convolutional Neural Network so we find this images that was labelled and have many images in sunny or rainy day and in the day and night it has many conditions of parking slot like parallel and perpendicular to the curb [11] ,then we convert this dataset to a yolo format so we can use them in our model. We made a python code specially for this training dataset to convert from its type to yolo format.

Here are examples from our dataset:

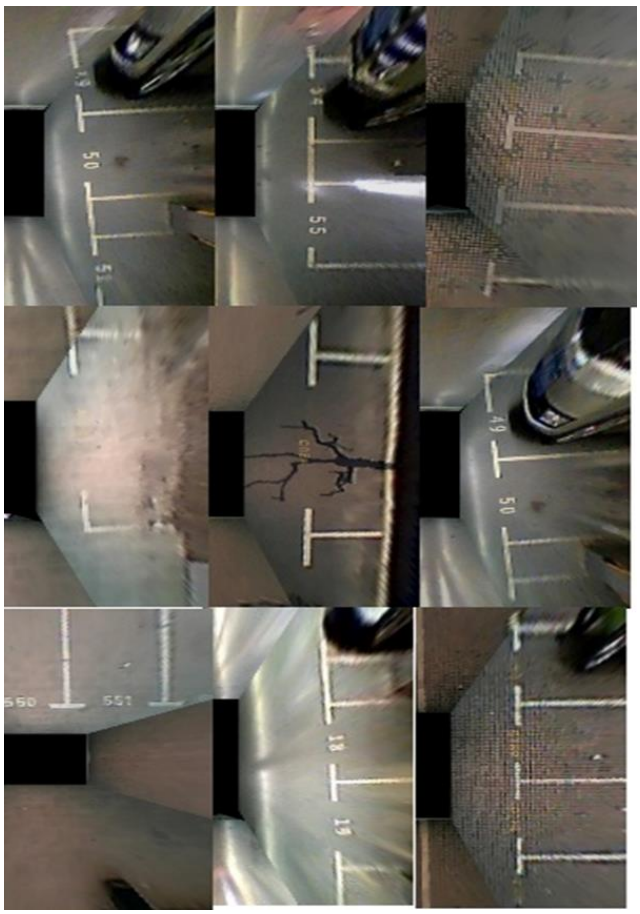


Figure 52: Example of Dataset

4.4.1 Model Architecture:

```
 params  module                                  arguments
   3520  models.common.Conv                      [3, 32, 6, 2, 2]
  18560  models.common.Conv                      [32, 64, 3, 2]
  18816  models.common.C3                        [64, 64, 1]
  73984  models.common.Conv                      [64, 128, 3, 2]
 115712  models.common.C3                        [128, 128, 2]
 295424  models.common.Conv                      [128, 256, 3, 2]
 625152  models.common.C3                        [256, 256, 3]
1180672  models.common.Conv                      [256, 512, 3, 2]
1182720  models.common.C3                        [512, 512, 1]
 656896  models.common.SPPF                      [512, 512, 5]
 131584  models.common.Conv                      [512, 256, 1, 1]
      0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
      0  models.common.Concat                    [1]
 361984  models.common.C3                        [512, 256, 1, False]
  33024  models.common.Conv                      [256, 128, 1, 1]
```

Figure 53: Model Architecture

```
Model summary: 214 layers, 7025023 parameters, 7025023 gradients
```

Figure 54: Model Summary

4.4.2 Model Train:

```
# Train YOLOv5s on data for 99 epochs
!python train.py --img 640 --batch 16 --epochs 99 --data custum_data.yml --weights yolov5s.pt --cache
```

Figure 55: Train The Model

- one epoch = one forward pass and one backward pass of all the training examples
- batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- Custum_data.yml = it is an yml file that contain our classes that need to be detected and the bath of the training data, validation data and test data

4.4.3 Results:

Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
98/98	4.63G	0.01065	0.007808	0.001702	18	640:

Class	Images	Instances	P	R	mAP50
all	107	126	0.976	0.868	0.988

Class	Images	Instances	P	R	mAP50
all	107	126	0.917	0.932	0.985
empty slot	107	110	0.979	0.864	0.978
not empty slot	107	16	0.854	1	0.991

Figure 56: Results

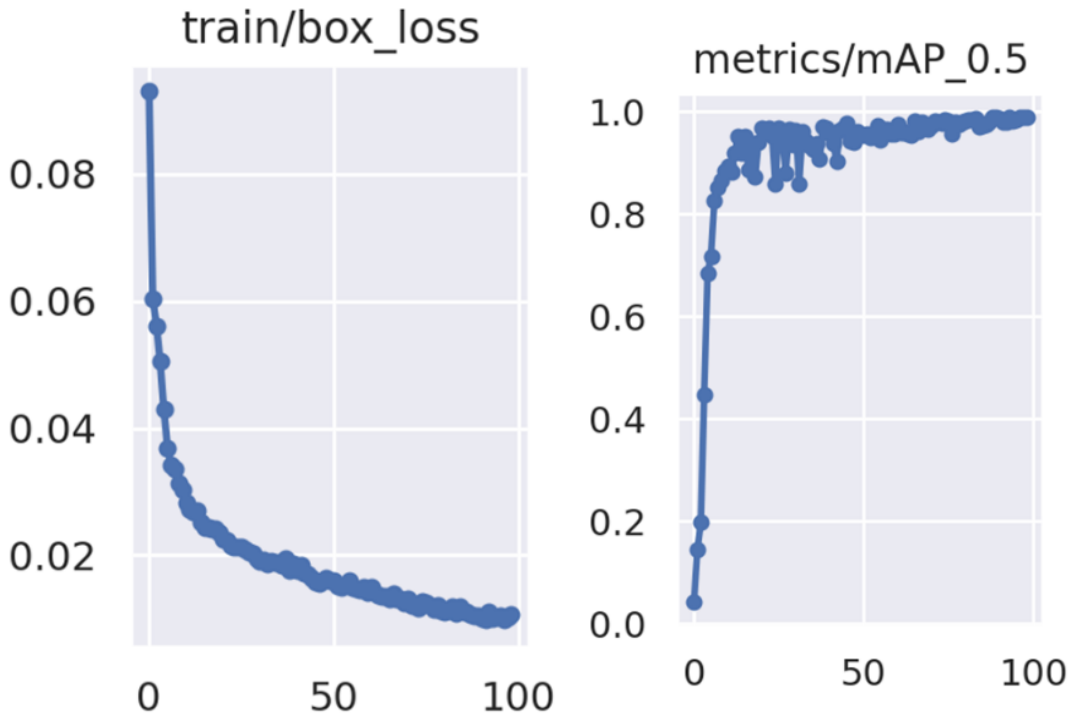


Figure 57: Loss and Accuracy with epochs

Using detection code, we apply our model to some images to see the behavior of the model we decide to show results that have more than 30% confidence level and here are some results:

```
!python detect.py --weights runs/train/exp/weights/best.pt --img 640 --conf 0.30
# display.Image(filename='runs/detect/exp/zidane.jpg', width=600)
```

Figure 58: Run the Model

Figure 59: Testing the Model

Then we take two images with our camera at faculty of engineering Ain-shams university and test it by the model

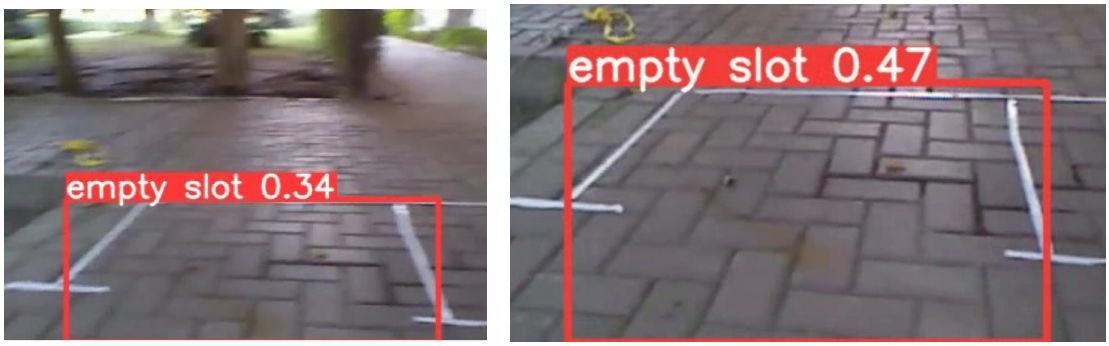


Figure 60: Result of our Images

Then we publish after every image or frame (in case of camera) if there is detection or no detection

```
384x640 1 empty slot, 11.9ms
384x640 1 empty slot, 11.8ms
384x640 1 empty slot, 11.5ms
```

Figure 61: Publishing the Result

4.5 CONCLUSION:

After determining the slot to park in we will now figure out a way to control our motors in the next chapter.

CHAPTER FIVE: CONTROL

5.1 VEICHLE SPESIFICATION:

Since the project entails the usage of pre-built vehicle, it was necessary to inspect the vehicle, its various parameters and on-board actuators and electronics and hence, determine its suitability for control. all of which were found in previous reference to the car.

Name	Dimensions in cm
Wheel Track	47.5
Tire Diameter	25
Wheel Base	66
Steering Linkages_1	28
Steering Linkages_2	7
Length	120
Width	70
Height	36
Steering Wheel	20

Figure 62: Vehicle Dimensions

5.1.1 PROPULSION SYSTEM:

The vehicle contains a DC motor driving each of the rear wheels through a speed reduction gearbox. The calculated reduction ratio from the gear train is 115.6.

Operating v	6v – 14.4v
Nominal v	12v
No Load RPM	11780
No Load A	0.8A
Stall Torque	46.9 oz-in / 331.2 mN-m
Stall Current	35A
Kt	1.34 oz-in/A / 9.5 mN-m/A
Kv	982 rpm/V
Efficiency	73%
RPM - Peak Eff	10275
Current - Peak Eff	5.1A

Figure 63: Dc motor specifications

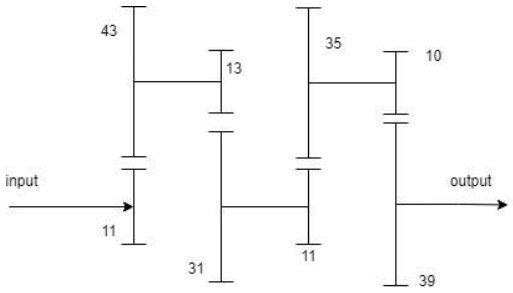


Figure 64: Propulsion gearbox schematic

The car had wheel incremental encoders to detect the speed of the wheels. The encoders acquired produces 360 pulses per revolution which allows for very fine control of the driven wheel

Figure 65: Incremental wheel encoder

The encoder is driven from the second shaft of the gearbox through a timing belt and pulley mechanism. The pulley ratio between the encoder and the gearbox shaft is 1:1 which then just leaves the speed reduction of the driven wheels to the second shaft which can be easily calculated and which further extends the resolution of the encoder for finer control adjustments. The timing belt setup allows for accurate performance since it doesn't allow for slipping between the pulley and belt but it leaves us with another problem. That which is the tensioning of the belt which was taken care of in the design of the encoder bracket. The bracket design consists of two parts which move relative to each other and allow for the tension adjustments of the belt with the help of a bolt and nut drive.

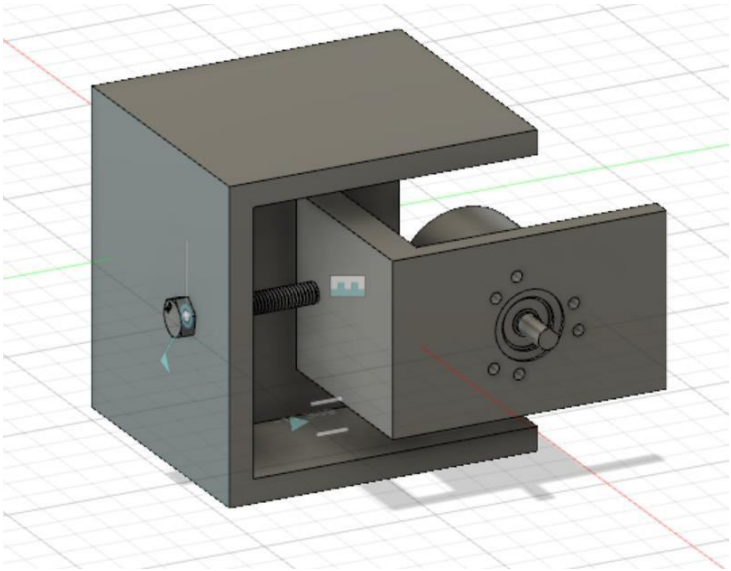


Figure 66: Encoder bracket cad design

5.1.2 STEERING ACTUATOR:

The steering mechanism in the vehicle consists of a DC motor which connects to a gear train which then transfers motion to the steering linkages which cause the rotation of the two front wheels. The Ackerman linkages offer 35 degrees of rotation in either direction which can be translated into a minimum turning radius of 765 cm. This is essential for the implementation of the path following algorithms during the traversal of the vehicle on the road determining the path and roads which can or cannot be traversed by the vehicle.

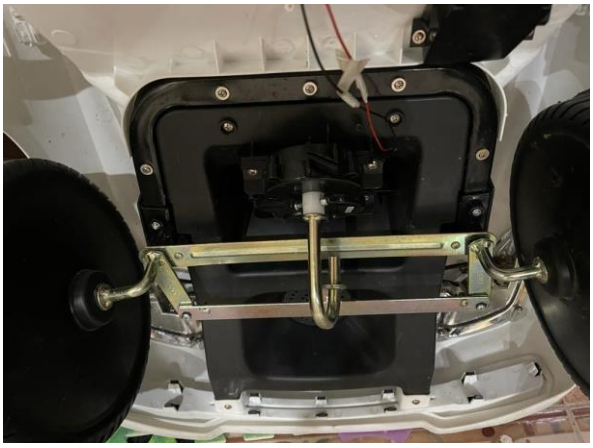


Figure 67:Steering linkages

For the purposes of control and autonomy however, an absolute encoder is required to determine the precise position of the steering wheel relative to a fixed position. However, due to availability and performance criteria none of the encoders found were suitable and thus a Servo motor was used instead of the DC motor. The servo motor offers precise steering angle input due to its closed loop feedback and thus negates the need for external encoder installation.

The Servo motor also necessitates a redesign of the gearbox which is not suitable for the high torque low speed of the servo motor compared to the DC motor. The adapted gearbox offers just 0.8 reduction and is placed inside the same housing of the previous gearbox and can be installed using the same mounting points.

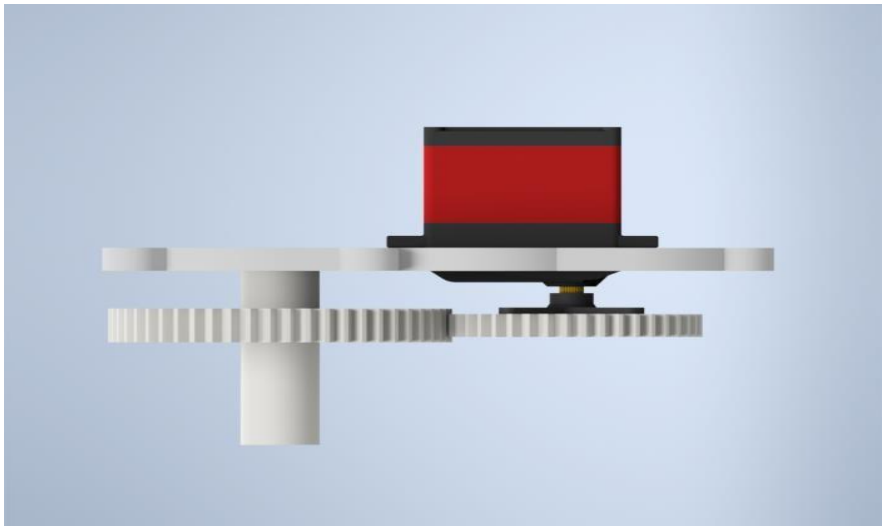


Figure 68:Steering gearbox cad design

Figure 69:Steering gearbox Assembly

5.2 MODELING:

We designed SIMULINK model for each dc motor to be able to apply different control methods. The challenge was to create accurate model to predict the system response and the design advantages that can be obtained from system modifications. The incorrect calculation of the motor parameters values leads to weakness in the control and instability, so the accuracy in extracting the parameters is a real problem.

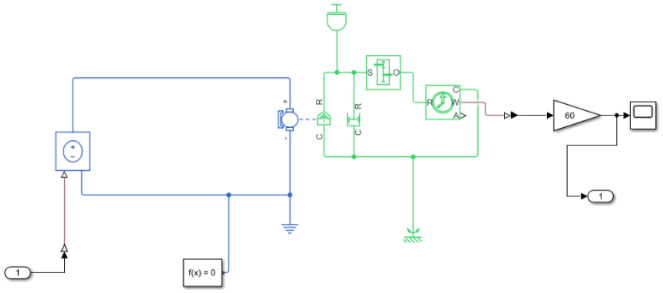


Figure 70: Dc motor model with gearbox, inertias, and damping

There were some **challenges** when we tried to find motor parameters:

1-Some parameters that affect the system such as inertia, friction or even damping, are very hard to be calculated or to measured accurately.

2- Motor efficiency decreases with time due to usage, and real motor efficiency cannot be easily detected.

3- Motor specifications(ex: no load rpm) in datasheet have error percentage, and the proof of this theory is that when you buy two identical motors and you power them with same voltage they give slightly different speeds(this is why we make single control loop for each wheel motor).

We used **Parameter estimator app** in **MATLAB** to overcome the above mentioned challenges.

5.2.1 Parameter Estimation:

Parameter estimation is the process of computing a model's parameter values from measured data. Parameter estimator uses optimization techniques to estimate model parameters. In each optimization iteration, it simulates the model with the current parameter values. It computes and minimizes the error between the simulated and measured output. The estimation is complete when the optimization method finds a local minimum. We have used parameter estimation to compute motor inertia(J_motor), motor damping (b_motor), armature resistance(R), Armature inductance(L),Back emf constant (Kv), External inertia(J) and External damping(B). Here is brief explanation for parameter estimations and steps that we have performed.

Parameter estimation steps:

1- Collect data:

We started by powering our motor with 1.4volts for 10 seconds and calculating the rpm using an incremental encoder. Then the direction was inverted(-1.4v) for 10 seconds. The pervious prosses was repeated again(1.4v for 10 seconds then -1.4v for 10 seconds) to have readings for 40 seconds. Our encoder was programmed to calculate rpm every 0.1 seconds, so now we have 400 readings over 40 seconds. Then we collected measured rpms in an excel sheet using **data streamer** tool in excel (search this part online).

It is preferred to collect data at maximum motor voltage (12v and -12v in case our motor) but unfortunately this was not possible in our test, because the encoder found in the car was not performing well at motor high speeds.

2- Enter voltage used in the experiment with respect to time as **input** to parameter estimator.

3- Select parameters required to be estimated and their initial value.

4- Estimate

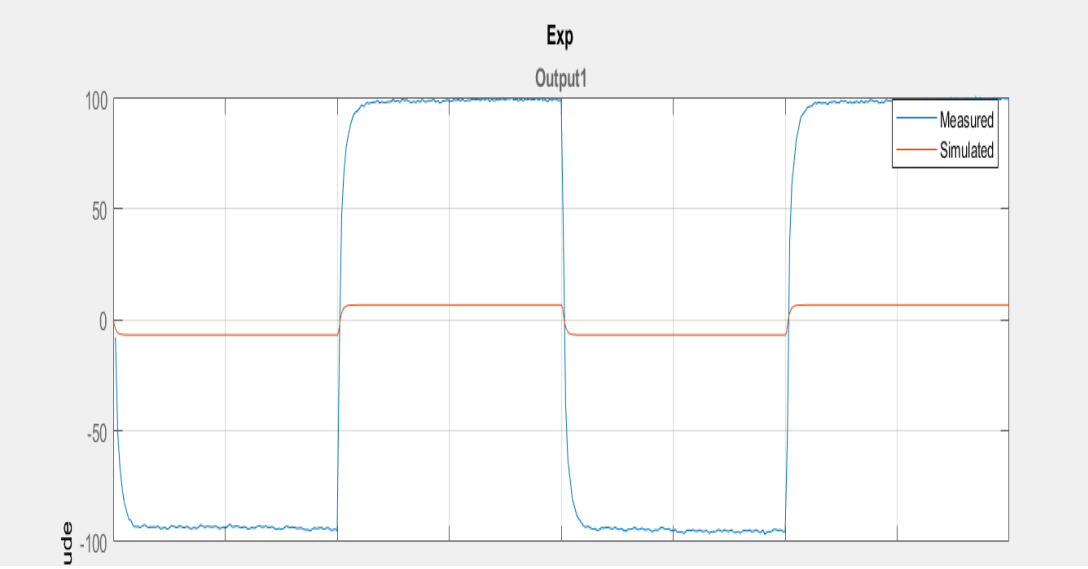


Figure 71: Simulation results before estimation

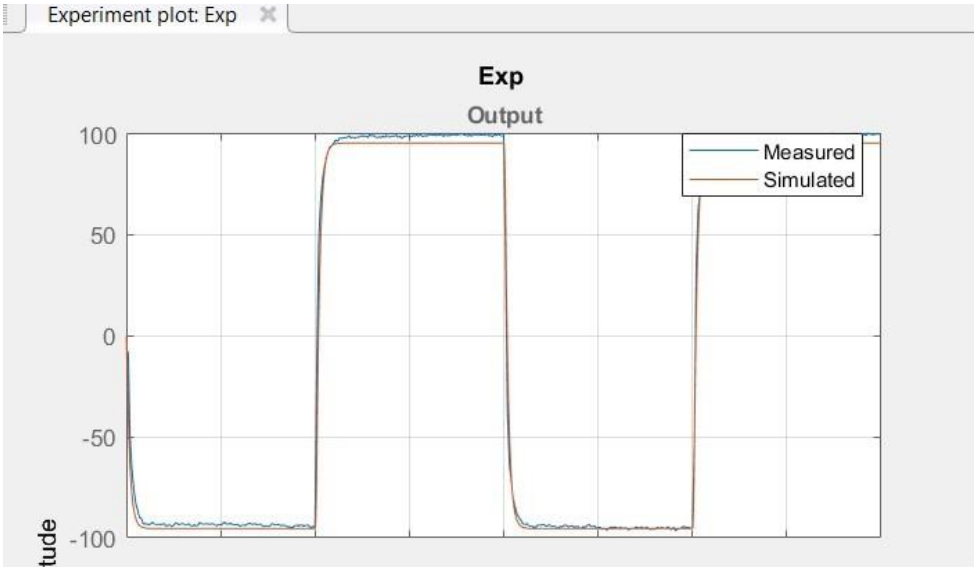


Figure 72: Simulation results after estimation

In First Figure's **parameter estimator** compares target output (measured rpm on reality) with simulation output. Input to simulation is the same input voltage used, and system parameters are initially assumed. In the second Figure's **parameter estimator** compares target output (measured rpm on reality) with simulation output after parameters updating. We can notice that simulation results after parameters updating are very close to real measurements.

5.3 LOW LEVEL CONTROL:

In brief, Proportional, Integral, Derivative PID controller is a feedback controller that helps to attain a set point irrespective of disturbances or any variation in characteristics of the plant of any form. It calculates its output based on the measured error and the three controller gains; proportional gain Kp, integral gain Ki, and derivative gain Kd. The proportional gain simply multiplies the error by a factor Kp. This reacts based on how big the error is. The integral term is a multiplication of the integral gain and the sum of the recent errors. The integral term helps in getting rid of the steady state error and causes the system to catch up with the desired set point. The derivative controller determines the reaction to the rate of which the error has been changing.

We have implemented pid controller and it gave good results in simulation and reality. After some tuning trials we have selected **kp = 0.06 , Ki =0.15 , kd=0**

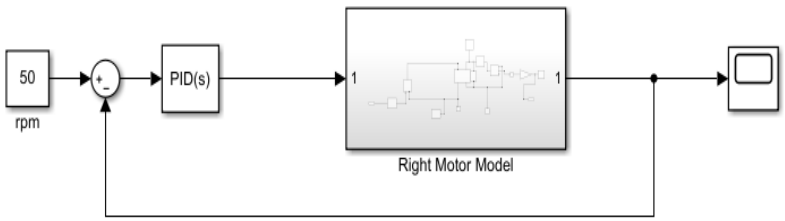


Figure 73:Pid Simulink model with 50 rpm input

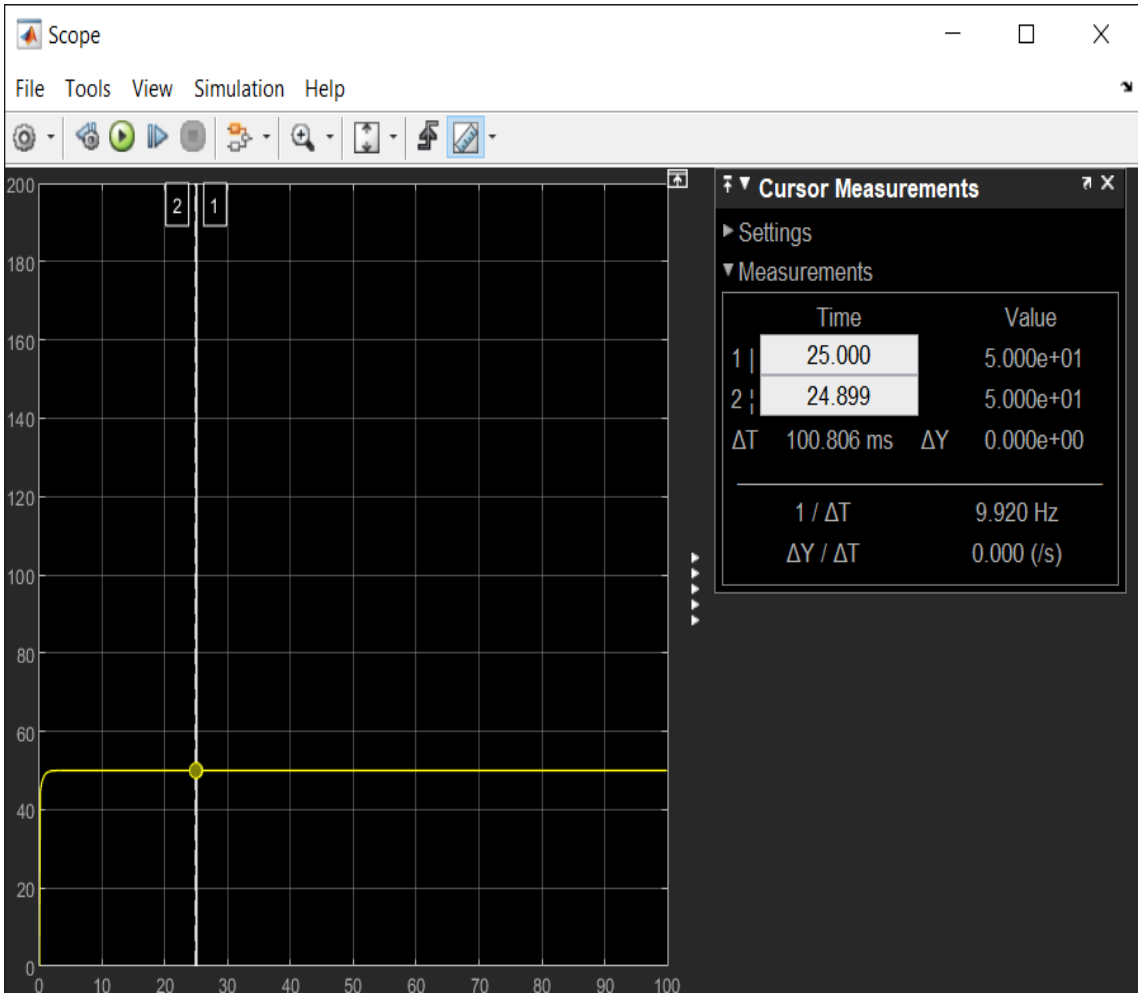


Figure 74:Simulation response with 50rpm exactly

When implementing this test on hardware, we have gotten approximately **50rpms** but with greater settling time.

CHAPTER SIX: SLOT ENTRANCE

This study proposes geometric path plans composed of two stages to automatically perform perpendicular and parallel parking with a reverse path in a narrow space. In perpendicular parking, the minimum width of the available parking spot is computed before the start of the parking operation, and the availability of parking and the optimal stage of the path plan are determined by considering the surrounding space together. In a similar concept, even in parallel parking, it is possible to determine whether to park by calculating the minimum length of the available parking spot and the number of repetitions of motion before starting the parking operation. The theoretical results for the parking spot and surrounding space related to the geometric path plans of perpendicular and parallel parking were confirmed through model car tests. Efficient automatic parking will be enabled by selecting and establishing an appropriate path plan along with the availability of parking in consideration of the parking spot and the surrounding space before starting the parking operation.

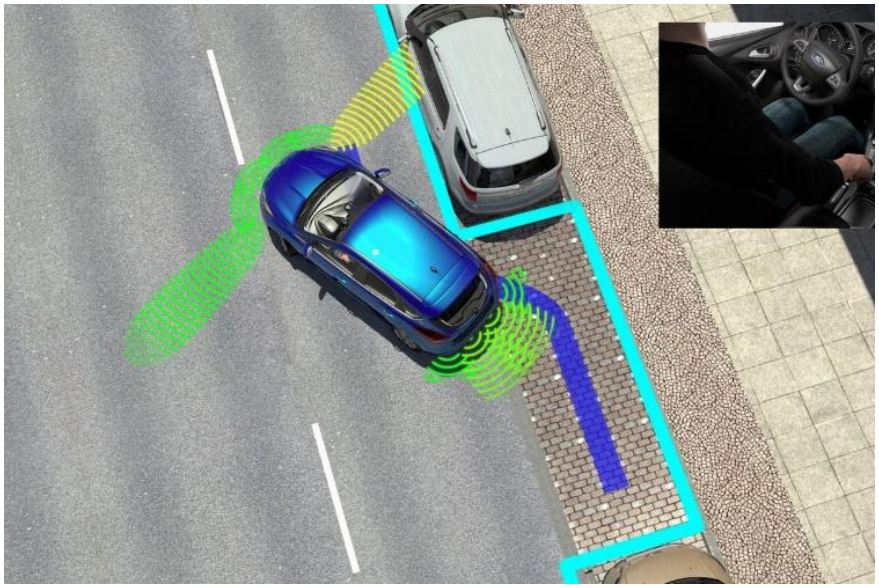


Figure 75: Movement of Parking

6.1 PARALLEL PARKING:

The vehicle is set to be positioned parallel to the parking spot on the right. If the length of the parking spot is long enough, it is possible to park comfortably forward, but it is effective to park backward in short parking spots for parallel parking located on the side of the road. In addition, parallel parking is usually attempted on the side of a road with traffic. The unit motion of the vehicle is divided into six characters (motions). In parallel parking, it is possible to determine whether to park by calculating the minimum length of the required parking spot and the number of repetitions of the parking operation.

$$(X_P, Y_P) = (\frac{\sqrt{3}}{2}R, -\frac{R}{2})$$

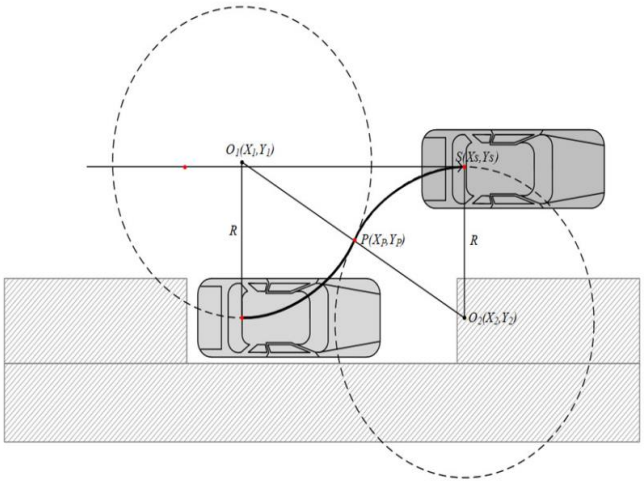


Figure 76: Parallel Parking

6.1.1 Plotting:

To be able to know how to park the car the following parameters needed to be measured:

- Radius = 80 cm
- 01 = (0,0)
- 02 = (138.564,-80)
- D = (0,-80)
- F = (69.282,-40)
- E = (138.564,0)

They are represented in the following figure

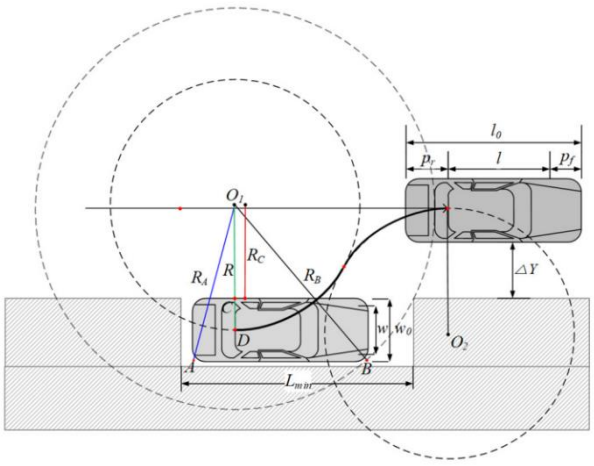


Figure 77: Parking Values

6.1.2 Dimensions:

For the dimensions of the car, half the width(35cm) plus half the curb(35cm) plus 10 cm between them equals 80cm.

Table 11: Car Parameters

Name	Dimensions in cm
Wheel Track	47.5
Tire Diameter	25
Wheel Base	66
Steering Linkages_1	28
Steering Linkages_2	7
Length	120
Width	70
Height	36
Steering Wheel	20

All these parameters were measured and checked.

6.1.3 MATLAB:

First Plotting the two tangent circles that the car will move on it with this code.

```
p = nsidedpoly(1000, 'Center', [0 0], 'Radius', 80);
plot(p, 'FaceColor', 'r')
axis equal

hold on.

p2 = nsidedpoly(1000, 'Center', [138.564 -80],
'Radius', 80);
plot(p2, 'FaceColor', 'r')
axis equal

hold off
```

the output will be,

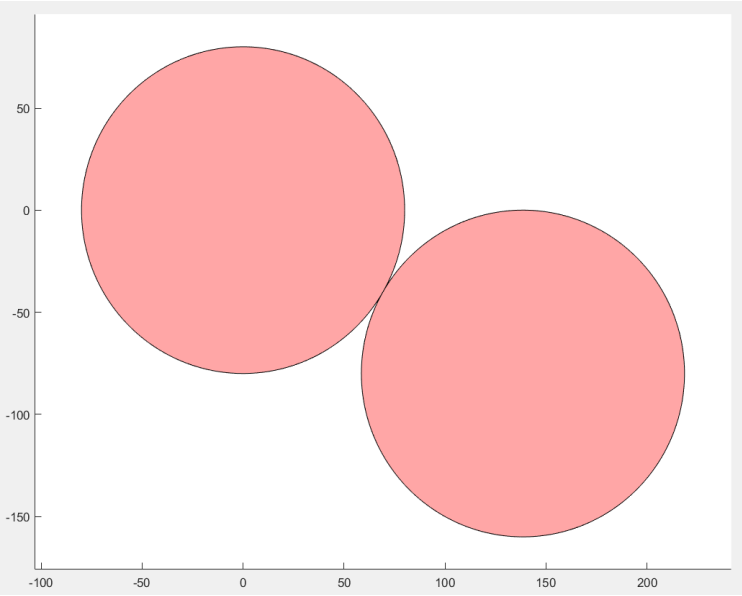


Figure 78: Drawing the circles

Then plotting the same two circles by determine the three points of the two arcs by this code.

```
p = nsidedpoly(1000, 'Center', [0 0], 'Radius', 80);
plot(p, 'FaceColor', 'r')
axis equal

hold on

p2 = nsidedpoly(1000, 'Center', [138.564 -80],
'Radius', 80);
plot(p2, 'FaceColor', 'r')
axis equal
 x = [138.564 69.282 0];
v = [0 -40 -80];
xi = [x(1):0.0000001:x(end)];
vid=interp1(x,v,xi,'spline');
plot(x,v,'b*')

plot(xi,vid,'r')
hold off
```

the output will be,

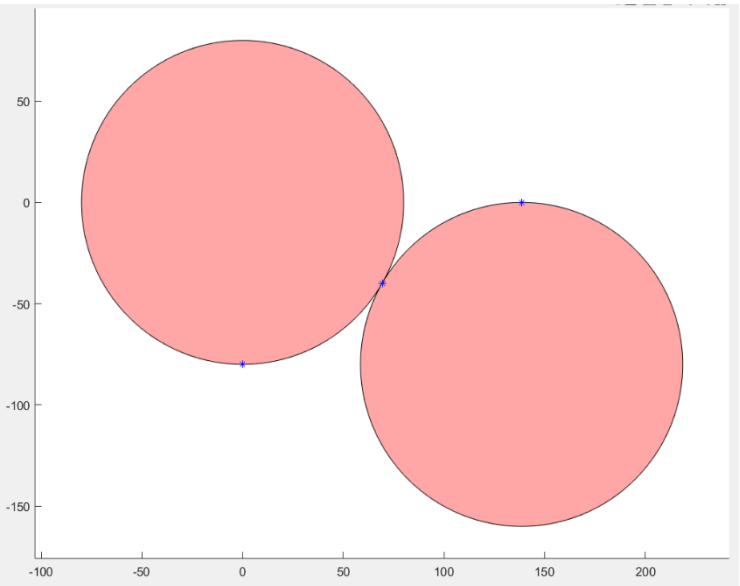


Figure 79: Three points on circle

Then plotting the two arcs that the car will move on it by this code

Then plotting the two arcs that the car will move on it by this code

```
a=[69.282 -40]; %P1
b=[0 -80]; %P2
r=80;      %radius
%next solution
syms x y
[x,y]=solve((x-a(1))^2+(y-a(2))^2==r^2,(x-b(1))^2+(y-
b(2))^2==r^2,x,y);
%plot arc
syms X Y
ezplot((X-x(1))^2+(Y-
y(1))^2==r^2,[min(a(1),b(1)),max(a(1),b(1)), ...
    min(a(2),b(2)),max(a(2),b(2))])
axis equal
figure
ezplot((X-x(2))^2+(Y-
y(2))^2==r^2,[min(a(1),b(1)),max(a(1),b(1)), ...
    min(a(2),b(2)),max(a(2),b(2))])
axis equal
```

the output will be,

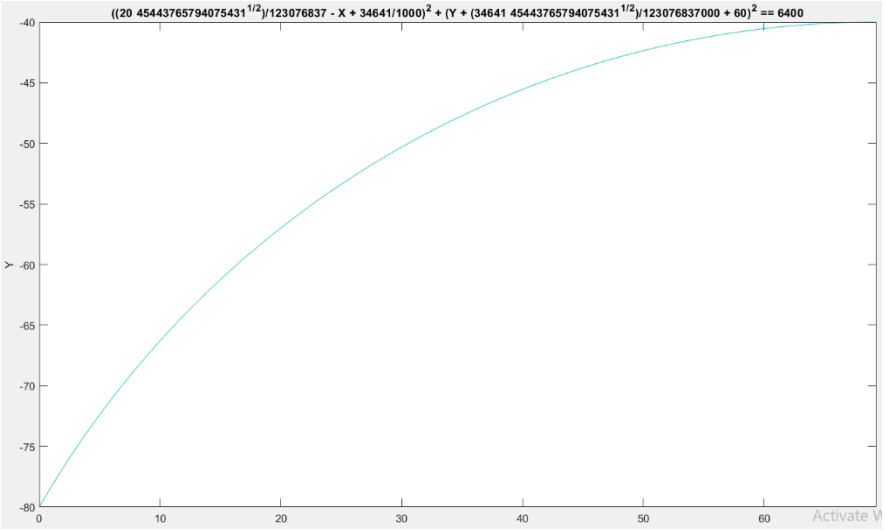


Figure 80: First Arc

The code of the second arc is,

```
a=[138.564 0]; %P1
b=[69.282 -40]; %P2
r=80;      %radius
%next solution
syms x y
[x,y]=solve((x-a(1))^2+(y-a(2))^2==r^2,(x-b(1))^2+(y-
b(2))^2==r^2,x,y);
%plot arc
syms X Y
ezplot((X-x(1))^2+(Y-
y(1))^2==r^2,[min(a(1),b(1)),max(a(1),b(1)), ...
    min(a(2),b(2)),max(a(2),b(2))])
axis equal
figure
ezplot((X-x(2))^2+(Y-
y(2))^2==r^2,[min(a(1),b(1)),max(a(1),b(1)), ...
    min(a(2),b(2)),max(a(2),b(2))])

axis equal
```

the output will be,

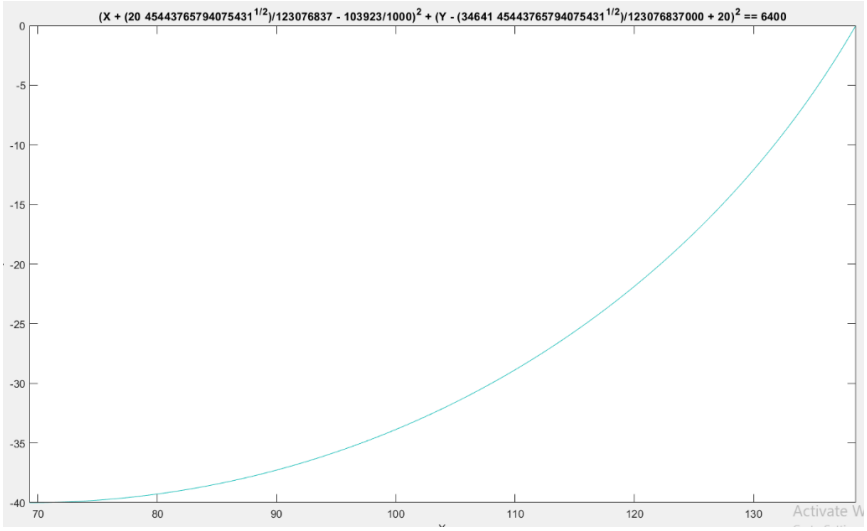


Figure 81: Second Arc

6.2 STEERING:

The car-like robot is subject to the limit of steering angle: $-\beta max < \beta < \beta max$. When the velocity v is constant and the steering angle β is fixed, the vehicle movement is almost a circle. The turning radius of that circle depends on the steering angle β. The steering angle therefore influences the minimum radius of the circle. This feature of the kinematics of the vehicle helps to plan a simple path with circular motions.

The turning radius is the radius of the circle created by a vehicle when it turns with a fixed steering angle. It is defined by a virtual wheel located in the middle of the front axle, using the steering angle β and the wheelbase.

Two other turning radii can be defined: the inner radius Ri which is the smallest radius formed by the inside rear wheel and the outer radius Re which is the largest radius formed by the outside front corner of the vehicle.

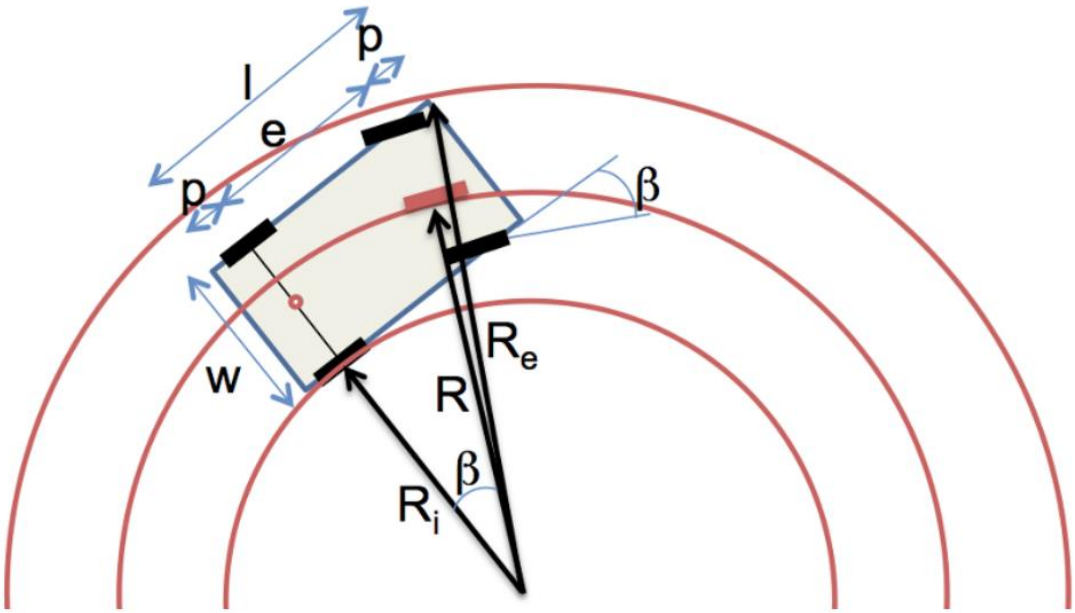


Figure 82: Steering

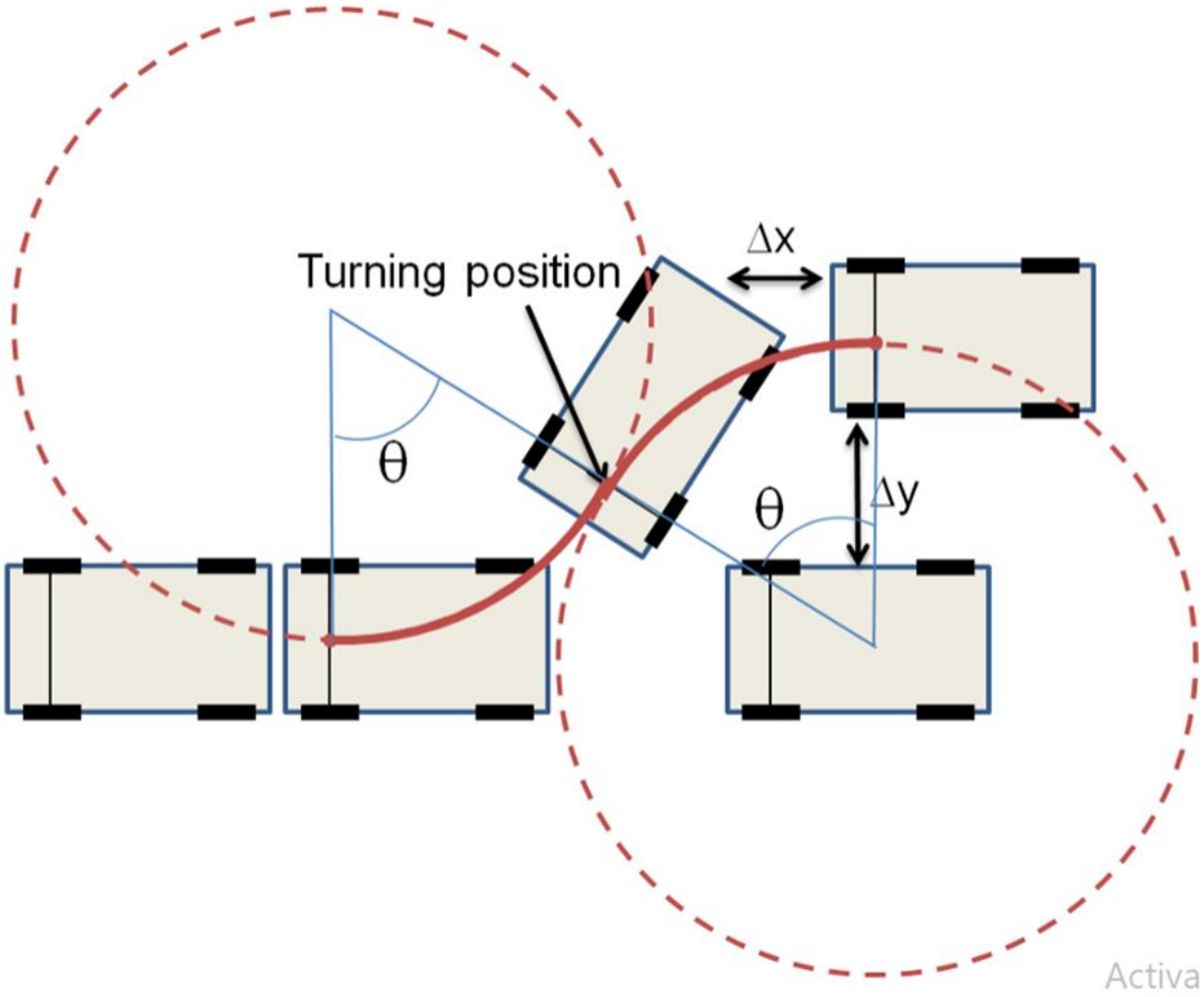


Figure 83: Initial Movements

The Equations:

$$R = e / \sin\beta$$

$$Ri = (R^2 - e^2)^0.5 - w/2 = (e^2/(\sin\beta)^2 - e^2)^0.5 - w/2 = e/\tan\beta - w/2$$

$$Re = ((Ri + w)^2 + (e + p)^2)^0.5 = (((R^2 - e^2)^0.5 + w/2)^2 + (e+p)^2)^0.5$$

6.3 PARKING SPACE:

In this section, the minimum length of the parking space to park a car in one trial will be precisely calculated. At the exit, the outer front corner will determine if the vehicle collides. The outer radius Remin should be therefore taken into account. The minimum length Lmin of the parking space can be calculated by using the Pythagorean theorem applied to the triangle C1BA.

Therefore if the length of the space is theoretically longer than Lmin, we can succeed parallel parking without collision.

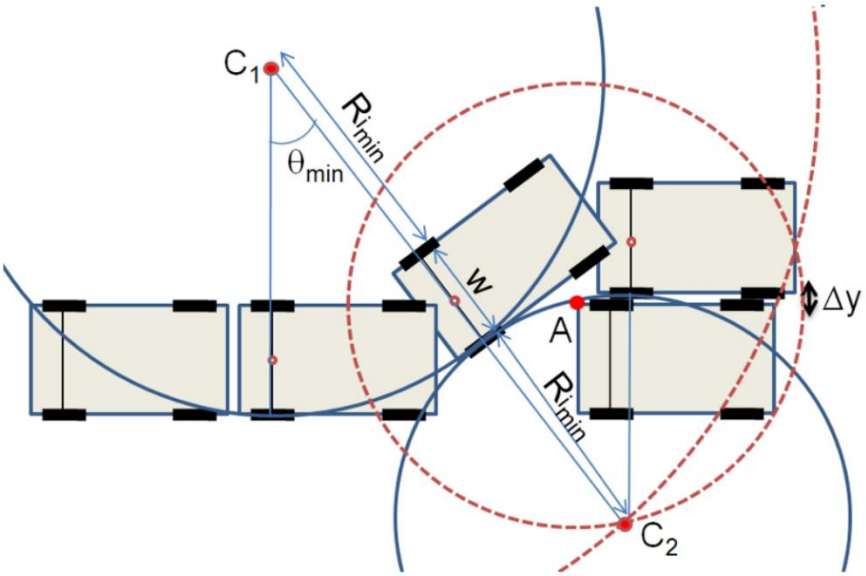


Figure 84: Movement Equations

The Equation:

$$Lmin = p + ((Remin)^2 – (Rimin)^2)^0.5$$

After knowing the movement we need to make the car track its movement on the path made.

CHAPTER SEVEN: TRACKING

7.1 INTRODUCTION:

In this chapter we represent the trajectory tracking, the general function of the trajectory tracking phase is to take group of way points as an input and its output is signals by the value of velocity and yaw angle to the low-level control to actuate actuators of steering and wheels.

This diagram showed in fig 1 shows a solution for the Waypoint Tracking Controller that receives as inputs a set of path waypoints and speed specifications along with the estimated vehicle state and generates as outputs the vehicle control signal

the selected behavior is translated into a path or trajectory that must be tracked by the low-level tracking controller

A convenient way to define this path is to generate a sequence of waypoints the vehicle has to go through. Finally, a feedback tracking controller is needed to select the appropriate actuator inputs to follow the planned waypoints also using the estimated vehicle pose and additional speed specifications from the behavioural layer.

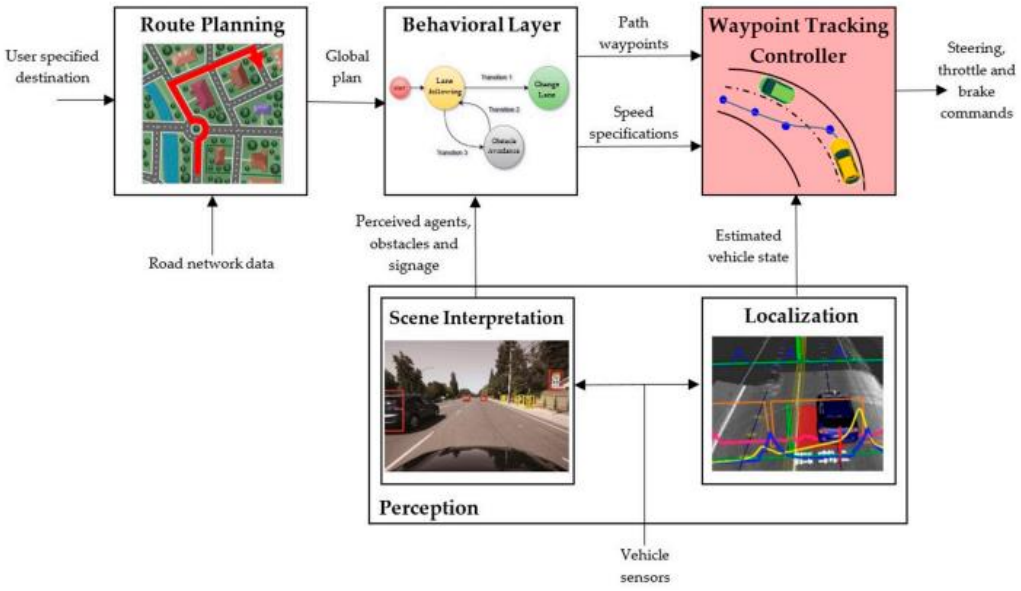


Figure 85: General navigation architecture of an autonomous vehicle

In Trajectory Tracking algorithm we used ROS packages to subscribe and publish data that help in integration between stages

Also Waypoint tracking controllers must consider feasibility, comfort and vehicle dynamics while following trajectory Interpolation techniques take the set of global waypoints and generate a smoother path so we studied these 3 ways:

- 1st is simple mathematical solution which is join waypoints with straight lines and circular shapes.
- 2nd polynomial curve to obtain smooth transitions between straight segments and curved ones
- 3rd Bézier curves low computational cost that rely on control points to define their shapes

There are different tracking controllers that can be implemented:

- Pure Pursuit
- LQR (linear quadratic regulator)
- Stanley controller
- MPC (Model Predictive Controller)

7.2 PURE PURSUIT CONTROLLER:

7.2.1 Introduction:

Pure pursuit is the geometric path tracking controller. A geometric path tracking controller is any controller that tracks a reference path using only the geometry of the vehicle kinematics and the reference path.

Pure Pursuit controller uses a look-ahead point which is a fixed distance on the reference path ahead of the vehicle as follows. The vehicle needs to proceed to that point using a steering angle which we need to compute.

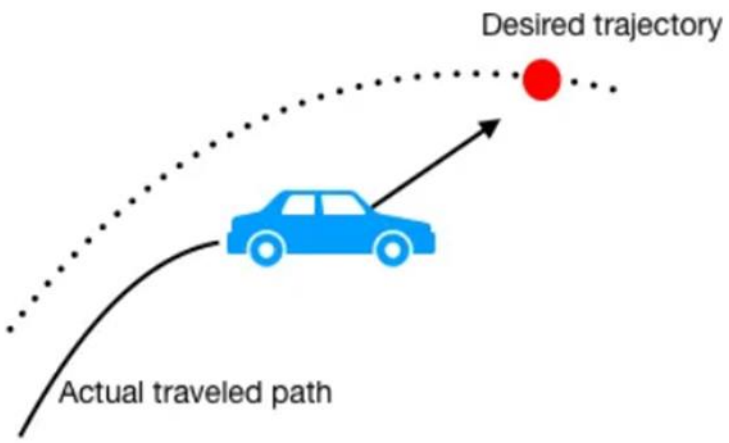


Figure 86: Geometric path tracking

In this method, the centre of the rear axle is used as the reference point on the vehicle and the target point is selected as the red point in the above

7.2.2 Pure Pursuit Formulation:

Sin rules are used:

$$\frac{l_d}{\sin(2\alpha)} = \frac{R}{\sin(\frac{\pi}{2} - \alpha)}$$

$$\frac{l_d}{2\sin(\alpha).\cos(\alpha)} = \frac{R}{\cos(\alpha)}$$

$$\frac{l_d}{\sin(\alpha)} = 2R$$

$$K = \frac{1}{R} = \frac{2\sin(\alpha)}{l_d}$$

Where distance between the rear axle and the target point is denoted as $\boldsymbol{ld}$, the instantaneous centre of rotation of this circle and the radius is denoted as **R**, the angle between the vehicle's body heading and the look-ahead line is referred as $\boldsymbol{\alpha}$ and **k** is the curvature.

Our target is to make the vehicle steer at a correct angle and then proceed to that point.

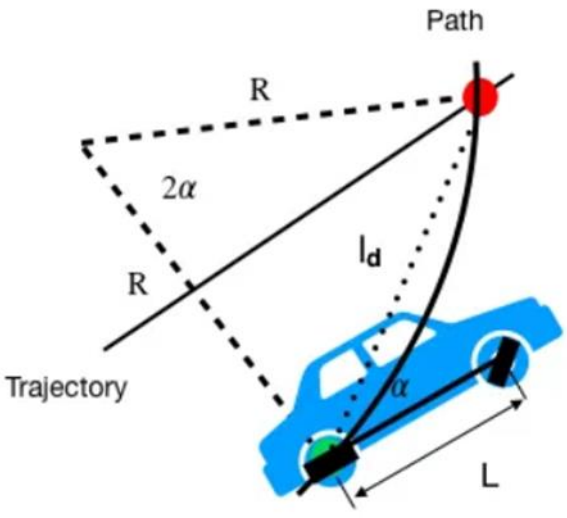


Figure 87: Pure pursuit geometric relationship

7.3 LQR (LINEAR QUADRATIC REGULATOR):

7.3.1 Introduction:

The Linear Quadratic Regulator (LQR) controller is a widely used control method in robotics for stabilizing systems with linear dynamics. It is a type of feedback controller that uses a combination of state feedback and a control input to stabilize a system and minimize a cost function.

waypoint tracking, the LQR controller is used to control the motion of a robotic system such that it follows a predefined set of waypoints. The waypoints can be represented by a desired state trajectory, x_d(t), and the control input is used to drive the system towards this desired trajectory.

7.3.2 LQR Formulation:

The cost function for waypoint tracking can be defined as:

$$J = \text{integral(from 0 to inf) } [||x(t) - x_d(t)||^2 + u^T(t) R u(t)] dt$$

Where **x(t)** is the current state of the system, **x_d(t)** is the desired state trajectory, **u(t)** is the control input, **R** is a positive definite matrix representing the control cost, and the first term represents the error between the current state and the desired state. The dynamics of the system can be represented in state-space form as:

$$dx(t)/dt = A x(t) + B u(t).$$

Where **A** and **B** are the state and input matrices, respectively.

The control input is then given by:

$$u(t) = -K(x(t) - x_d(t))$$

7.4 STANLEY CONTROLLER:

7.4.1 Introduction:

In this controller it is different it uses the front axle as its reference point, it looks at both the heading error and cross-track error, The cross-track error is defined as the distance between the closest point on the path with the front axle of the vehicle.

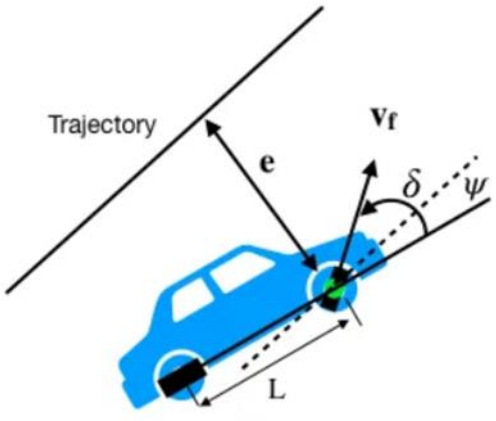


Figure 88: Stanley geometric relationship

7.4.2 Stanley Formulation:

Firstly, eliminating the heading error:

$$\delta(t) = \psi(t)$$

Secondly, eliminating the cross-track error

$$\delta(t) = tan^{-1}(\frac{Ke(t)}{V_f(t)})$$

Where e(t) is the closest point between the path and the vehicle

Last step, is to obey $[\delta_{max}, \delta_{min}]$

$$\delta(t) = \psi(t) + tan^{-1}\left(\frac{ke(t)}{v_f(t)}\right), \qquad \delta(t) \in [\delta_{min}, \delta_{max}]$$

7.5 MPC (MODEL PREDICTIVE CONTROLLER):

7.5.1 Introduction:

We should first know the cost function. For example, in this project, we want to control the vehicle to follow waypoints track. So, the cost function should contain the deviation from the reference path, smaller deviation better results.

This is like the optimization problem of optimal control theory and trades off control performance and input aggressiveness.

We want to seek the best input to optimize our cost function. We can summarize the whole MPC process as follows.

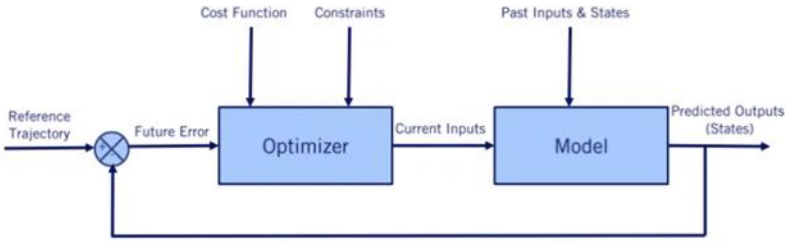


Figure 89: MPC structure

- Firstly, suppose our steering angle bounds are $\delta(t) \in [\delta min, \delta max]$. I use a simple method that discrete the input of the model, which is the steering angle δ into values with the same interval.

- Then we can get the predicted outputs which are [x, y, θ, δ] using the above model and the input δ.

- The last step is to select the smallest value of the cost function and its corresponding inputs δ. (In this case, we divided steering angle with 0.1 intervals from δmin -1.2 to δmax 1.2 radians. Then put it into the cost function and for loop to find the minimum value and its corresponding input δ.)

- Repeat the above process in each time step.

7.5.2 MPC Formulation:

$$J(x(t), u) = \sum_{j=t}^{t+T-1} \delta x_{jt}^T Q \delta x_{jt} + u_{jt}^T R u_{jt}$$

Where $\boldsymbol{\delta}\mathbf{x}$ is the distance between the predictive point and the reference point and **u** is steering input

$$\delta x_{jt} = x_{jt,des} - x_{jt}$$

7.5.3 Predictive Model:

In this controller we used simple bicycle model its formulas as follows:

$$x\_dot = v * cos(\delta + \theta)$$

$$y\_dot = v * sin(\delta + \theta)$$

$$\theta\_dot = v / R = v / (L/sin(\delta)) = v * sin(\delta)/L$$

$$\delta\_dot = \varphi$$

$$x\_(t+1) = x\_t + x\_dot * \Delta t$$

$$y\_(t+1) = y\_t + y\_dot * \Delta t$$

$$\theta\_(t+1) = \theta\_t + \theta\_dot * \Delta t$$

$$\delta\_(t+1) = \delta\_t + \delta\_dot * \Delta t$$

7.6 COMPARISON:

7.6.1 MPC VS LQR:

Table 12: MPC Vs LQR

MPC (Model Predictive Control)	**LQR (Linear Quadratic Regulator)**
can handle constraints on the control inputs and system states	While LQR con **not** do that
Requires system model to be known and prediction horizon to be chosen	Only requires linearization model of the system
Can handle system with **multiple** inputs and outputs	Can handle system with **single** input and output
Better for systems with **changing** operating points	Better for systems with **stable** operating points
Can handle system with time **varying** dynamics	Can handle system with **constant** dynamics
Provide feed forward control law	Provide feedback control law

7.6.2 Pure Pursuit VS Stanley:

Table 13: Pure Vs Stanley

Pure pursuit controller	Stanley controller
Uses look “ahead point” to determine the desired path	Uses “lateral deviation” from the desired path to determine control inputs
Can manage high speeds and tight turns	Can be more stable on sharp turns
Can be less stable on sharp turns	Can be less responses on sudden changes in path
Can be sensitive to noise in sensor data	Can be less effective at high speeds
Based on constantly pointing the vehicle’s front wheel towards a target point	Uses combination between pure pursuit and feedback control to vehicle move according to sensor data

This comparison provides a general overview of the main differences between pure pursuit and Stanley controllers. It highlights that pure pursuit is a simpler method for steering a vehicle to follow a path, while Stanley is a more advanced controller that uses additional sensor data and more complex algorithms for precise control.

The comparison also notes that pure pursuit is well-suited for real-time path-following applications, while Stanley is better suited for handling unknown and unpredictable environments.

7.7 VEICHLE MODEL AND LIMITATIONS:

The kinematic model of a vehicle assumes it has negligible inertia with limited speed (up to (50 Km/s)). To make the kinematic model simple using Ackerman steering we used the following criteria: vehicle pose (x, y, θ) is taken with respect to an external reference frame where (x, y) are the center point of front wheel axle and (θ) is the heading angle between its longitudinal axle and the external axle.

From these assumptions we can easily calculate the following:

$$\dot{x}(t) = V(t).\cos\big(\rho(t) + \theta(t)\big)$$

$$\dot{y}(t) = V(t).\sin\big(\rho(t) + \theta(t)\big)$$

$$\dot{\theta}(t) = \Omega(t) = \frac{V(t).\sin\left(\rho(t)\right)}{L}$$

Where V(t) is the speed, ρ(t) is the angle of a virtual central front wheel at the control point, $\Omega(t)$ is the angular velocity (first derivative of the heading angle) and L is the distance between front and wheel axle.

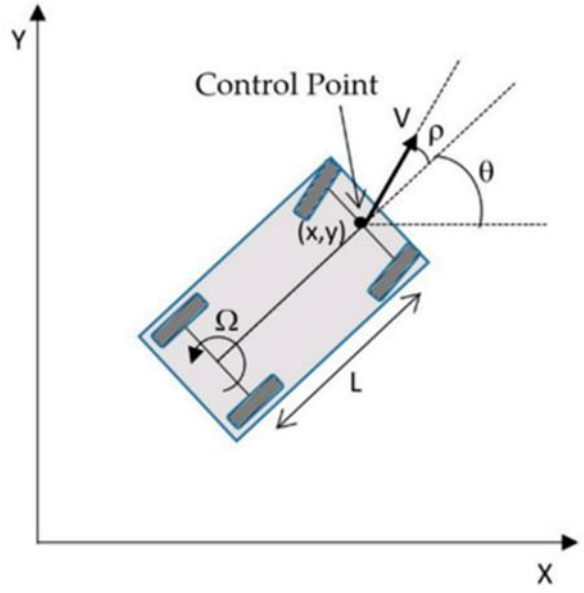


Figure 90: Kinematic model with Ackerman steering

7.8 CONTROLLER DESIGN:

In this figure bellow our controller design and how velocity and yaw angle signals is published

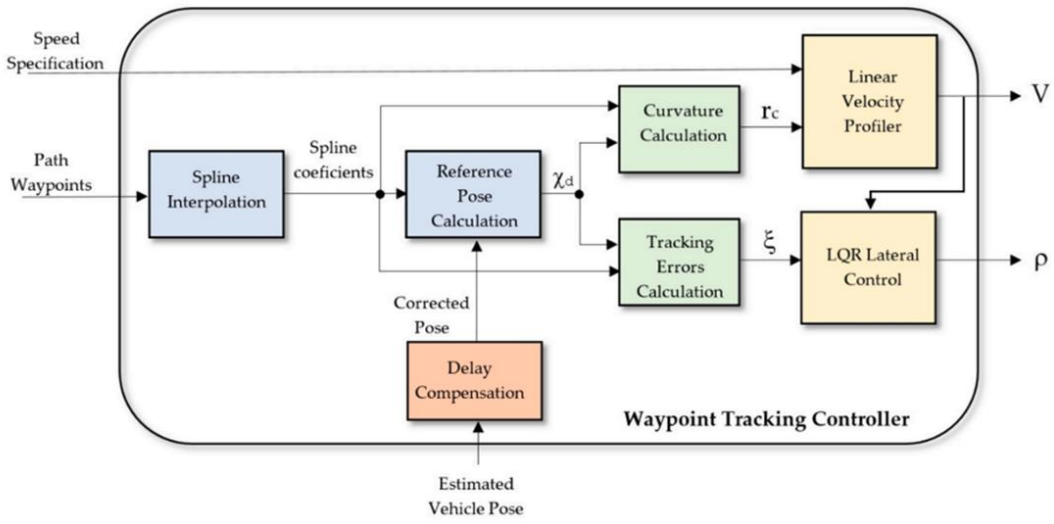


Figure 91: Waypoint Tracking

Inputs are a set of path waypoints; an external speed specification and the estimated vehicle pose.

Outputs are linear speed V and steering angle ρ. The main subsystems are a Spline Interpolator, a Linear Velocity Profiler, a Linear Quadratic Regulator (LQR) Lateral Controller and a Delay Compensation System.

7.8.1 Linear Velocity Profiler and Curvature Calculation:

The characteristics of the trajectory itself determine the optimal speed at which the vehicle must travel. , the most relevant is curvature C, that limits the maximum speed to assure that the vehicle follows the prescribed path minimizing sliding.

When dealing with two-dimensional spline curves, the radius of curvature r_c (defined as the inverse of the curvature C) in the point corresponding to a certain value of the parameter u is:

$$r_c(u) = \frac{(x'(u)^2 + y'(u)^2)^{3/2}}{x'(u).y''(u) - y'(u).x''(u)}$$

Then, an average speed vi is assigned to each segment of the spline, which is a function of its average radius of curvature:

$$\bar{v}_i = V_{MAX} * \frac{\min(\bar{rc}_i, RC_{MAX})}{RC_{MAX}}$$

where VMAX is the maximum allowed linear velocity for the vehicle and RCMAX is the maximum value of the radius of curvature that reduces the linear velocity below VMAX

7.8.2 Lateral Control:

We have implemented an LQR Optimal Controller for its stability characteristics and its easy tuning using cost functions.

The control law is reduced to the following expression:

$$\rho(k) = -K \cdot \xi(k) = -[K_1 \quad K_2] \cdot \begin{bmatrix} de(k) \\ \theta e(k) \end{bmatrix}$$

7.8.3 Delay Compensation:

The developed controller provides good results when the vehicle travels at slow speeds, so that the delays present in the robot's sensors and actuators can be neglected. However, autonomous vehicles in environments sometimes must travel at high speeds (up to 50 km/h) and, in this situation, delays significantly destabilize the control system. There are two main delay sources to take into account in the control loop: the delay in the localization system readings, and the one introduced by the actuators.

- It is assumed that at the time k in which the control signal is calculated, the available position reading X_r. If the vehicle travels at high speeds, this delay introduces a large error in the position used to calculate the control signal.
- It is assumed that the control signal calculated at time k will have effect on the vehicle after n_c sampling periods. Obviously, at this time the vehicle will also have modified its position considerably if the speed is high.

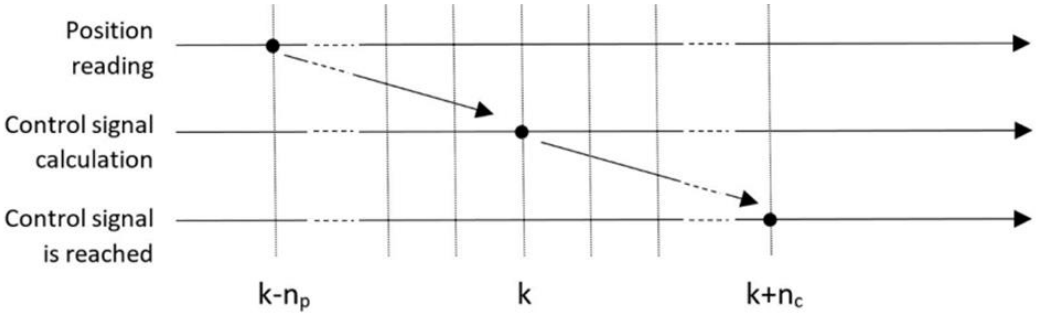


Figure 92: Delays involved in the control.

7.8.4 Simulation by MATLAB

Simulation of Stanley controller

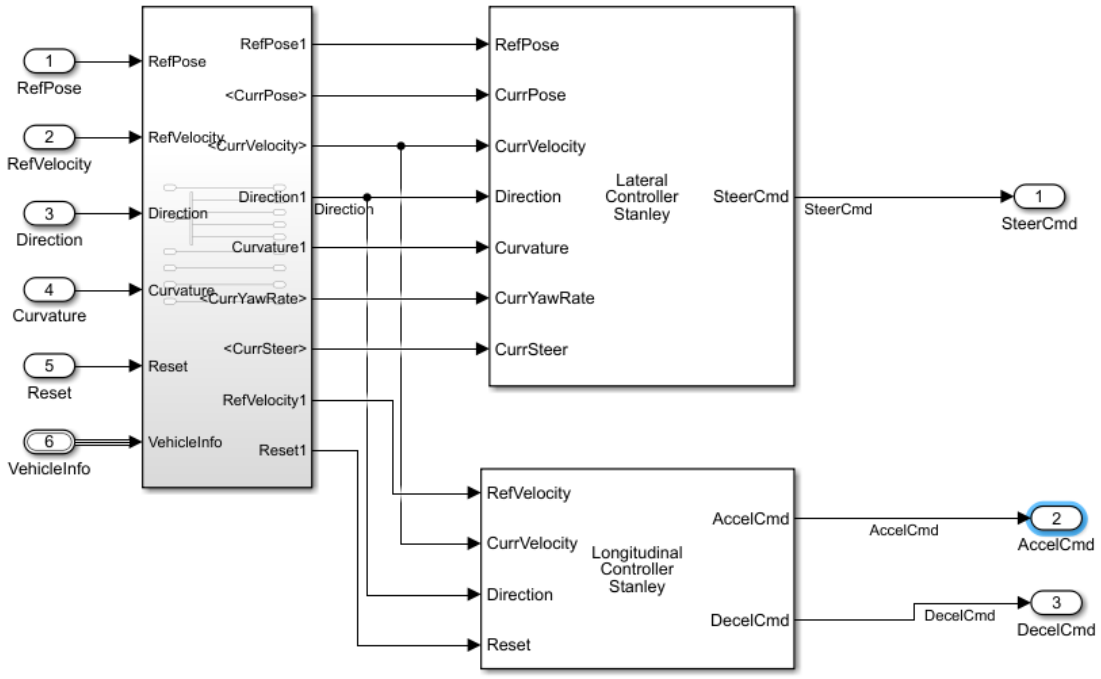


Figure 93: stanley simulation in matlab

This simulation is represented by two main blocks

1. Lateral controller Stanley : this block is responsible for the steering output and its input is (reference position which is the target point and the current position and current velocity and direction which is the difference between the current point and the next point and current yaw angle that is needed in the equation that calculates the steering angle)

2. Longitudinal controller Stanley: this block is responsible for velocity output control by acceleration and deceleration commands and its input is (reference velocity which is the limit of the velocity that cant exceed and current velocity that the car have now and direction which is difference between current and next point that is needed to get distance used in equation of velocity)

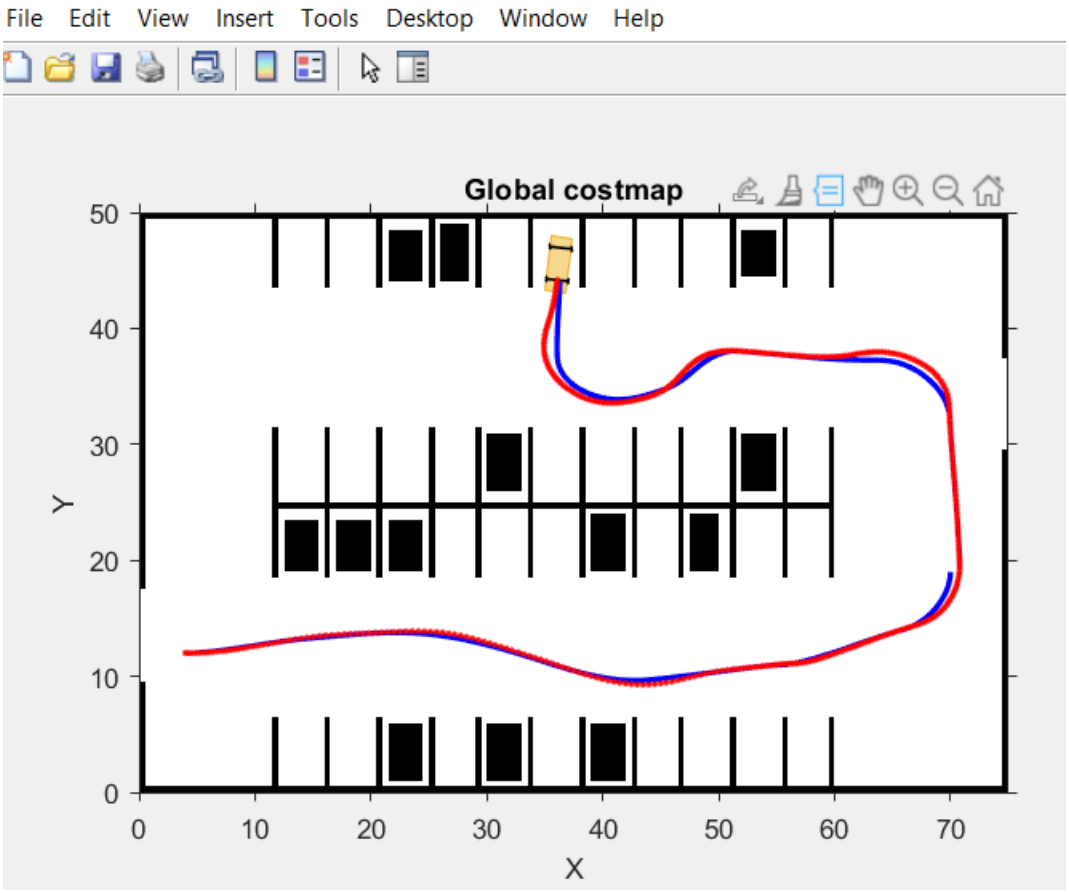


Figure 94: tracking simulation

CONCLUSION

After all implementations and research done, the electric vehicle was able to achieve some of the goals one of them is to park in its designated spot when positioned in the starting point calculated beforehand, in the simulation part every stage has completed its simulation to make the car park in its parking slot, the MATLAB model also outputs the right values when tested.

In the end we

- Acquired the map and location of the car using the lidar and google cartographer.
- Determined the path we will follow to reach the point which we will start parking from it.
- Located the parking slot using a kinetic camera.
- Controlled the motor using different algorithms.
- Determined the starting position in which we will start the parking movement.
- Using tracking algorithms to track the path.

CONTACT LIST

Ahmed Mamdoh	19P5326	19P5326@eng.asu.edu.eg	Vision and slot detection
Hussam Elsayed	18P6670	18P6670@eng.asu.edu.eg	Vision
Thomas Medhat	18P8912	18P8912@eng.asu.edu.eg	Path Planning
Mohamed Hatem	18P3449	18P3449@eng.asu.edu.eg	SLAM
Youssef Mohamed	18P5628	18P5628@eng.asu.edu.eg	Motor Control
Andrew Boulos	18P7917	18P7917@eng.asu.edu.eg	Path Tracking
Amr Anwar	18P4576	18P4576@eng.asu.edu.eg	Parking Movement

REFERENCES

[1] I. A.-N. X. C. a. P. M. Mustafa Al-Khawaldeh, Ubiquitous robotics for knowledge-based auto-configuration system within smart home environment, IEEE, 2016.

[2] J.-Q. Z. Y.-S. Z. a. B. K. Jie-Hua Zhou, Jie-Hua Zhou, Ji-Qiang Zhou, Yong-Sheng Zheng, and Bin Kong., IEEE, 2016.

[3] M. S. Z. M. H. I. a. N. H. Takayuki Kanda, An affective guide robot in a shopping mall, IEEE., 2009.

[4] R. U. a. A. Moran, "Mobile robot path planning in complex environments," IEEE, 2017.

[5] J. c. Latombe, "Robot motion planning," The Springer International Series in Engineering and Computer Science, 1991.

[6] M. Šeda, " Roadmap methods versus cell decomposition in robot motion planning," World Scientific and Engineering Academy and Society (WSEAS), 2007.

[7] N. a. M. B. K. G. Choubey, "Analysis of working of dijkstra and a* toobtain optimal path," *International Journal of Computer Science and Management Research,* vol. 2, p. 1898–1904, 2013.

[8] S. D. Joseph Redmon, "You Only Look Once:.," *University of Washington,Allen Institute for AI.,* 2015.

[9] K. H. Girshick, "R-CNN.," *Facebook AI Research (FAIR),* 2018.

[10] H. ,. Do, "Context-Based Parking Slot Detection With a Realistic Dataset," *IEEE,* 2020.

[11] J. H. Lin Zhang1, "Vision-based Parking-slot Detection: A DCNN-based Approach and A Large-scale Benchmark Dataset.," *School of Software Engineering, Tongji University, Shanghai, China,* 2016.

[12] F. S. O. a. D. F. W. Alberto Y. Hata, "Robust Curb Detection and Vehicle Localization in Urban Environments," *2014 IEEE Intelligent Vehicles Symposium (IV),* p. 6, 2014.

[13] I. J. Cox, ""Blanche–an experiment in guidance and navigation of an autonomous robot vehicle" in IEEE Transactions on Robotics and Automation," 1991. [Online].

[14] A. K. P. R. P. S. S. P. &. Y. H. C. G. A. Kumar, ""A LiDAR and IMU Integrated Indoor Navigation System for UAVs and Its Application in Real-Time Pipeline Classification"," 2017. [Online].

[15] I. B. C. D. A. V. Ioan Lita, "A New Approach of Automobile Localization System Using GPS and GSM/GPRS Transmission," *Electronics, Communications and Computers Department, University of Pitesti,* p. 5, 2006.

[16] R. Z. Q. H. &. M. A. J. Li, ""Feature-Based Laser Scan Matching and Its Application for Indoor Mapping" in Sensors," 2016. [Online].

[17] Y. Jia, "Quaternions and Rotations.," 2013. [Online]. Available: http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/quaternion.pdf.

[18] M. I. Y. J. J. K. S. M. I. H. G. J. S. M. I. Kichun Jo, "Precise Localization of an Autonomous Car Based on Probabilistic Noise Models of Road Surface Marker Features Using Multiple Cameras," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* p. 16.

[19] A. H. a. B. Soheilian, "Road Side Detection and Reconstruction Using LIDAR Sensor," *2013 IEEE Intelligent Vehicles Symposium (IV),* p. 6, 2013.

[20] A. T. S. &. D. M. B. W. Travis, ""Corridor Navigation with a LiDAR/INS Kalman Filter Solution" in IEEE Proceedings," 2005. [Online].

[21] A. H. a. D. Wolf, "Road Marking Detection Using LIDAR Reflective Intensity Data and its," *2014 IEEE 17th International Conference on,* p. 6, 2014.

[22] R. Gutierrez, "A Waypoint Tracking Controller for Autonomous Road Vehicles Using ROS Framework," *MDPI,* 2020.

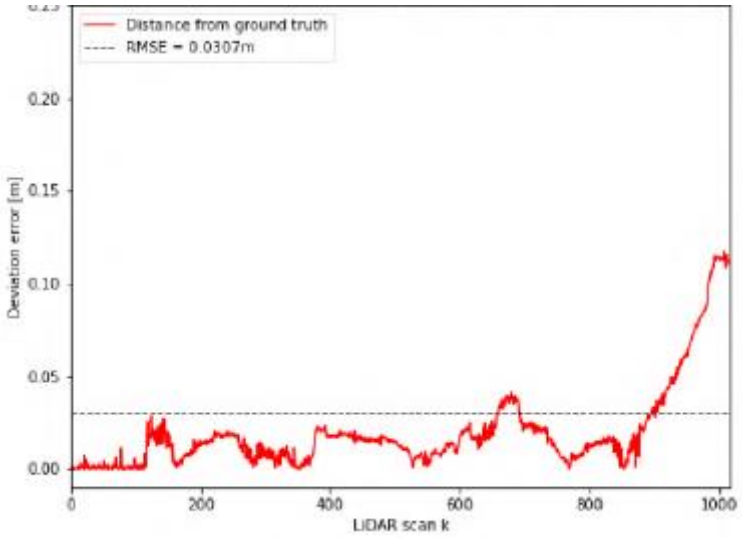
[23] F. Rego, “Cooperative path-following control with logic-based communications: theory and practice,” *Research Gate,* 2019.

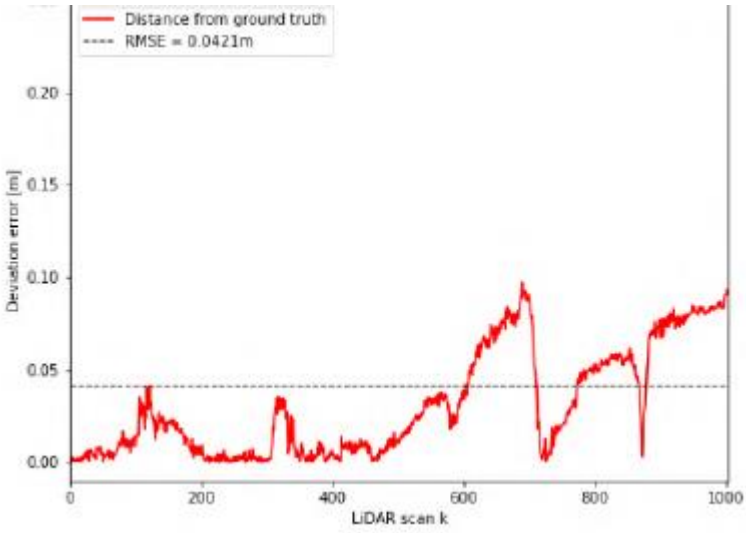
[24] S. V. Rakovi´c, “Handbook of Model Predictive Control,” *Springer,* 2010.

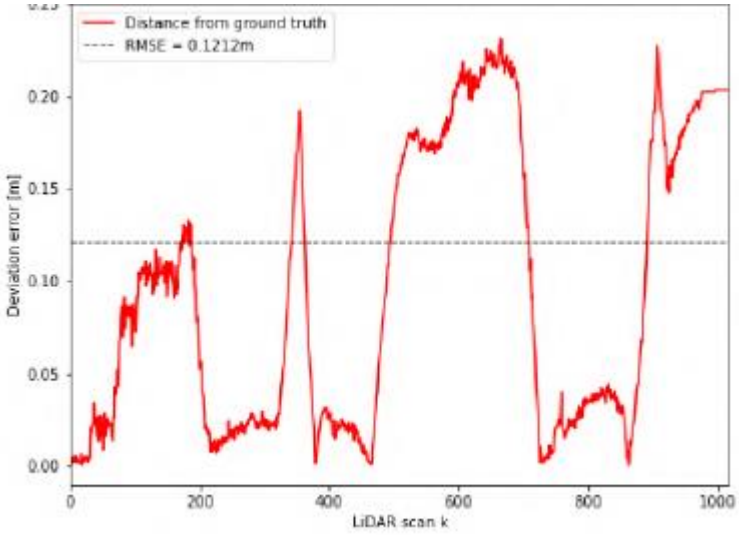
[25] D. Dolgov, “Practical Search Techniques in Path Planning for Autonomous Driving”.

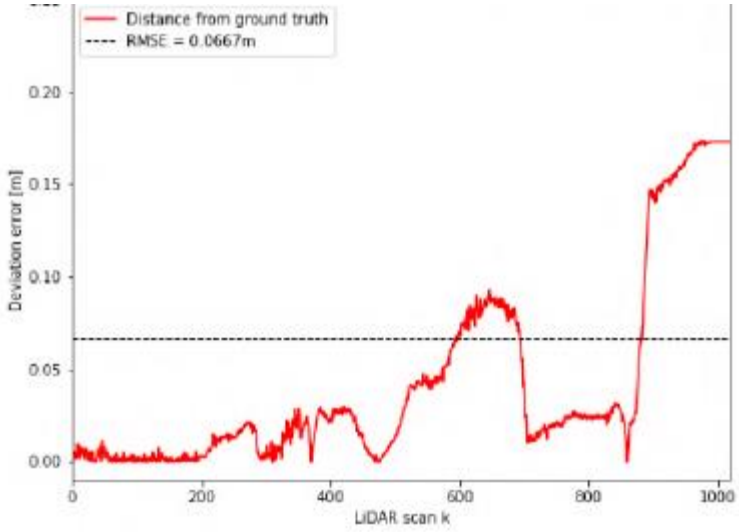
APPENDIX

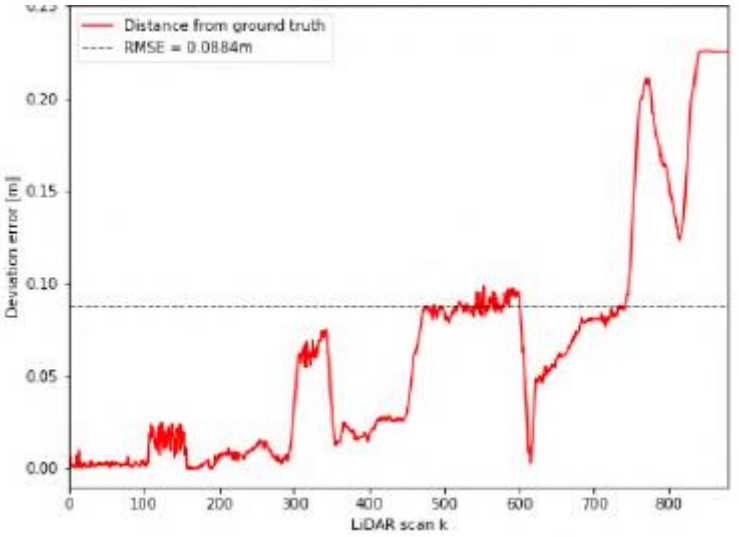
10.1 FEATURE BASED ERROR



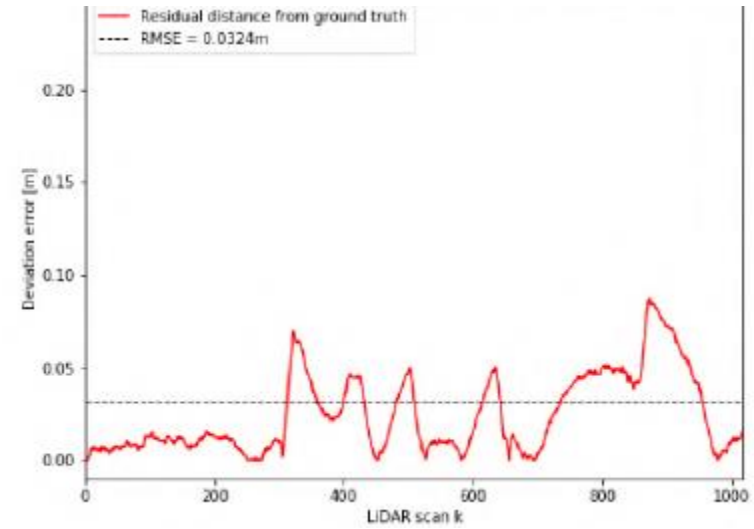


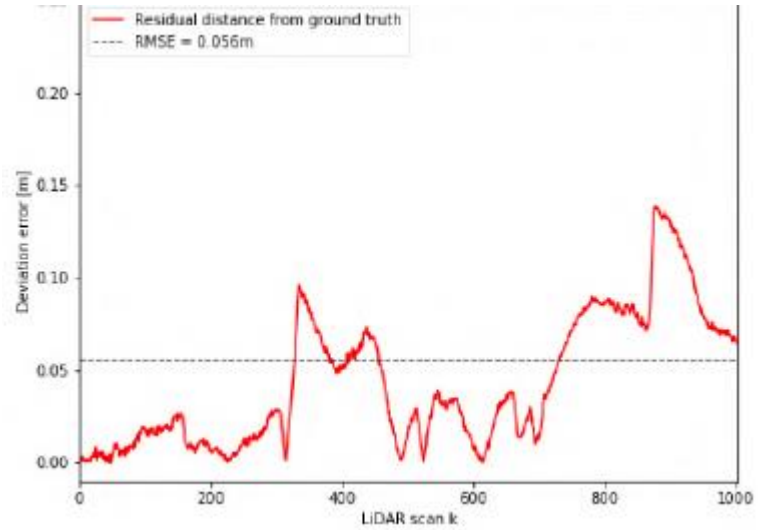

Distance from ground truth
RMSE = 0.1212m
Deviation error [m]
0.00
0.05
0.10
0.15
0.20
0
200
400
600
800
1000
LiDAR scan k

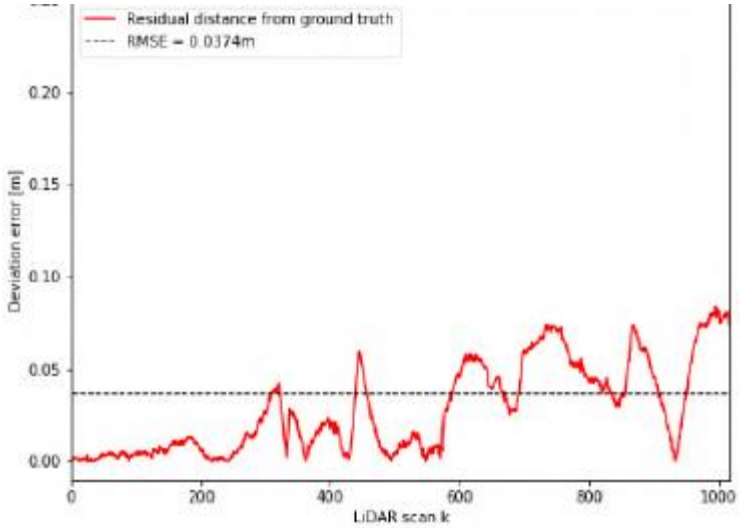

Distance from ground truth
RMSE = 0.0667m
Deviation error [m]
0.00
0.05
0.10
0.15
0.20
0
200
400
600
800
1000
LiDAR scan k

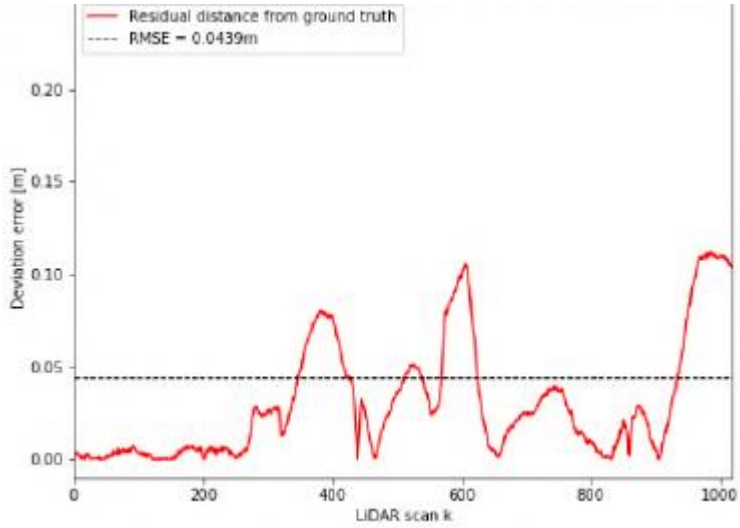

Distance from ground truth
RMSE = 0.0884m
Deviation error [m]
0.00
0.05
0.10
0.15
0.20
0
100
200
300
400
500
600
700
800
LiDAR scan k

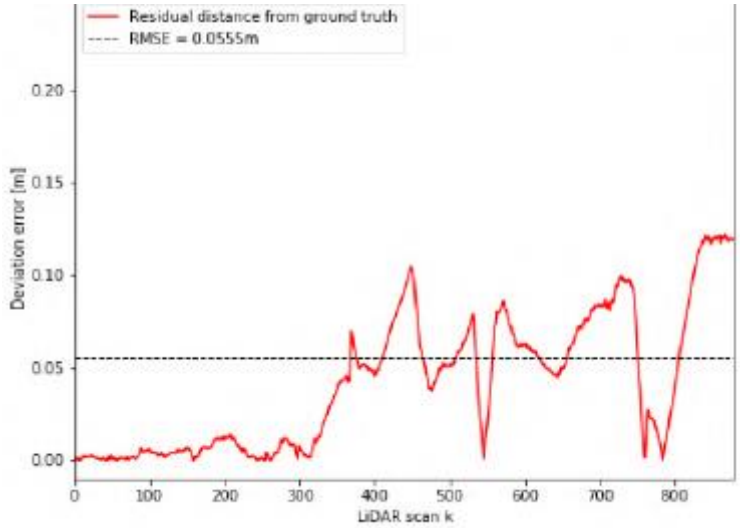
10.2 ICP ERROR








Residual distance from ground truth
RMSE = 0.0439m
Deviation error [m]
0.00
0.05
0.10
0.15
0.20
0
200
400
600
800
1000
LiDAR scan k


Residual distance from ground truth
RMSE = 0.0555m
Deviation error [m]
0.00
0.05
0.10
0.15
0.20
0
100
200
300
400
500
600
700
800
LiDAR scan k

10.3 ASTAR

Step by step A star path planning

Blue represent Start, Red Obstacles and Green Goal.

Every cell represents 10 distance measure number written in each will be the cost.

		Obstacle	Goal	
		Obstacle		
Start				

		Obstacle	Goal	
50	44	Obstacle		
Start 50 closed	50			

50	44	Obstacle	Goal	
50	Closed 44	Obstacle		
Start 50	50	50		

64	58	52		
50	Closed 44	Obstacle	Goal	
50	Closed 44	Obstacle		
Start 50	50	50		

64	58	52		
Closed 50	Closed 44	Obstacle	Goal	
Closed 50	Closed 44	Obstacle		
Start 50	50	50		

64	58	52		
Closed 50	Closed 44	Obstacle	Goal	
Closed 50	Closed 44	Obstacle		
Start 50	Closed 50	50		

64	58	52		
Closed 50	Closed 44	Obstacle	Goal	
Closed 50	Closed 44	Obstacle	44	
Start 50	Closed 50	Closed 50	50	

64	58	52		
Closed 50	Closed 44	Obstacle	Goal	
Closed 50	Closed 44	Obstacle	Closed 44	
Start 50	Closed 50	Closed 50	50	

64	58	52		
Closed 50	Closed 44	Obstacle	Goal 44	58
Closed 50	Closed 44	Obstacle	Closed 44	58
Start 50	Closed 50	Closed 50	50	64

64	58	52		
Closed 50	Closed 44	Obstacle	Goal 44 closed	58
Closed 50	Closed 44	Obstacle	Closed 44	58
Start 50	Closed 50	Closed 50	50	64

Path

		Obstacle	Goal	
		Obstacle		
Start	→	→	↗	

10.4 BFS

Step by step BFS path planning

Blue represent Start, Red Obstacles and Green Goal.

Every number represent the cell number.

1	2	3	4	5
6	7	Obstacle 8	Goal 9	10
11	12	Obstacle 13	14	15
Start 16	17	18	19	20

We will start from south counter clockwise. V will be mark of visited

1	2	3	4	5
6	7	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14	15
Start 16 V	17 V	18	19	20

Queue: 16 11 12 17

1	2	3	4	5
6 V	7 V	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14	15
Start 16 V	17 V	18	19	20

Queue: 11 12 17 6 7

1	2	3	4	5
6 V	7 V	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14	15
Start 16 V	17 V	18 V	19	20

Queue: 12 17 6 7 18

1	2	3	4	5
6 V	7 V	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14	15
Start 16 V	17 V	18 V	19	20

Queue: 17 6 7 18

1 V	2 V	3	4	5
6 V	7 V	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14	15
Start 16 V	17 V	18 V	19	20

Queue: 6 7 18 1 2

1 V	2 V	3 V	4	5
6 V	7 V	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14	15
Start 16 V	17 V	18 V	19	20

Queue: 7 18 1 2 3

1 V	2 V	3 V	4	5
6 V	7 V	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14 V	15
Start 16 V	17 V	18 V	19 V	20

Queue: 18 1 2 3 14 19

1 V	2 V	3 V	4	5
6 V	7 V	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14 V	15
Start 16 V	17 V	18 V	19 V	20

Queue: 1 2 3 14 19

1 V	2 V	3 V	4	5
6 V	7 V	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14 V	15
Start 16 V	17 V	18 V	19 V	20

Queue: 2 3 14 19

1 V	2 V	3 V	4 V	5
6 V	7 V	Obstacle 8	Goal 9 V	10
11 V	12 V	Obstacle 13	14 V	15
Start 16 V	17 V	18 V	19 V	20

Queue: 3 14 19 4 9

1 V	2 V	3 V	4 V	5
6 V	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Queue: 14 19 4 9 10 15 20

1 V	2 V	3 V	4 V	5
6 V	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Queue: 19 4 9 10 15 20

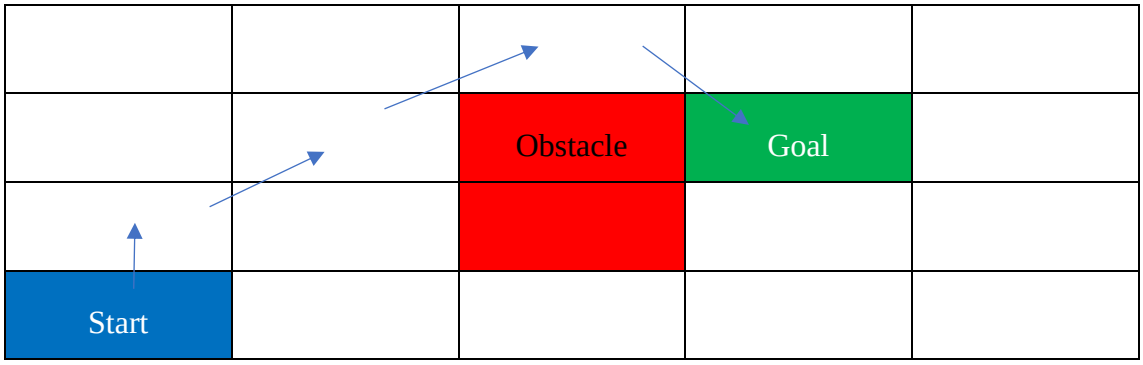
1 V	2 V	3 V	4 V	5 V
6 V	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Queue: 4 9 10 15 20 5

1 V	2 V	3 V	4 V	5 V
6 V	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Queue: 9 10 15 20 5 9 is goal so we stop

Path:



10.5 DFS

Step by step DFS path planning

Blue represent Start, Red Obstacles and Green Goal.

Every number represent the cell number.

1	2	3	4	5
6	7	Obstacle 8	Goal 9	10
11	12	Obstacle 13	14	15
Start 16	17	18	19	20

We will start from south counter clockwise. V will be mark of visited

1	2	3	4	5
6	7	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14	15
Start 16 V	17 V	18	19	20

Stack: 11 12 17

1	2	3	4	5
6	7	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14	15
Start 16 V	17 V	18 V	19	20

Stack: 11 12 18

1	2	3	4	5
6	7	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14 V	15
Start 16 V	17 V	18 V	19 V	20

Stack: 11 12 14 19

1	2	3	4	5
6	7	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 15 20

1	2	3	4	5
6	7	Obstacle 8	Goal 9	10
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 15

1	2	3	4	5
6	7	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 10

1	2	3	4 V	5 V
6	7	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 4 5

1	2	3	4 V	5 V
6	7	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 4

1	2	3 V	4 V	5 V
6	7	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 3

1	2 V	3 V	4 V	5 V
6	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 7 2

1 V	2 V	3 V	4 V	5 V
6 V	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 7 6 1

1 V	2 V	3 V	4 V	5 V
6 V	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 7 6

1 V	2 V	3 V	4 V	5 V
6 V	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 7

1 V	2 V	3 V	4 V	5 V
6 V	7 V	Obstacle 8	Goal 9 V	10 V
11 V	12 V	Obstacle 13	14 V	15 V
Start 16 V	17 V	18 V	19 V	20 V

Stack: 11 12 14 9 Reach goal stop

Path

		Obstacle	Goal	
		Obstacle		
Start				

10.6 DIJKSTRA

Step by step Dijkstra Path Planning

Blue represent Start, Red Obstacles and Green Goal.

Every cell represents 10 distance measure number written in each will be the cost and parent.

Inf	Inf	Inf	Inf	Inf
Inf	Inf	Obstacle	Goal Inf	Inf
Inf	Inf	Obstacle	Inf	Inf
Start 0	Inf	Inf	Inf	Inf

	A	B	C	D	E
1	Inf	Inf	Inf	Inf	Inf
2	Inf	Inf	Obstacle	Goal Inf	Inf
3	10 A4	14 A4	Obstacle	Inf	Inf
4	Start 0	10 A4	Inf	Inf	Inf

	A	B	C	D	E
1	Inf	Inf	Inf	Inf	Inf
2	20 A3	24 B3	Obstacle	Goal Inf	Inf
3	10 A4	14 A4	Obstacle	Inf	Inf
4	Start 0	10 A4	20 B4	Inf	Inf

	A	B	C	D	E
1	30 A2	34 B2	Inf	Inf	Inf
2	20 A3	24 B3	Obstacle	Goal Inf	Inf
3	10 A4	14 A4	Obstacle	34 C4	Inf
4	Start 0	10 A4	20 B4	30 C4	Inf

	A	B	C	D	E
1	30 A2	34 B2	38 B2	Inf	Inf
2	20 A3	24 B3	Obstacle	Goal Inf	Inf
3	10 A4	14 A4	Obstacle	34 C4	Inf
4	Start 0	10 A4	20 B4	30 C4	Inf

	A	B	C	D	E
1	30 A2	34 B2	38 B2	Inf	Inf
2	20 A3	24 B3	Obstacle	Goal Inf	Inf
3	10 A4	14 A4	Obstacle	34 C4	44 D4
4	Start 0	10 A4	20 B4	30 C4	40 D4

	A	B	C	D	E
1	30 A2	34 B2	38 B2	Inf	Inf
2	20 A3	24 B3	Obstacle	Goal 44 D3	48 D3
3					
4					

10 A4	14 A4	Obstacle	34 C4	44 D4
Start 0	10 A4	20 B4	30 C4	40 D4

	A	B	C	D	E
1	30 A2	34 B2	38 B2	48 C1	Inf
2	20 A3	24 B3	Obstacle	Goal 44 D3	48 D3
3	10 A4	14 A4	Obstacle	34 C4	44 D4
4	Start 0	10 A4	20 B4	30 C4	40 D4

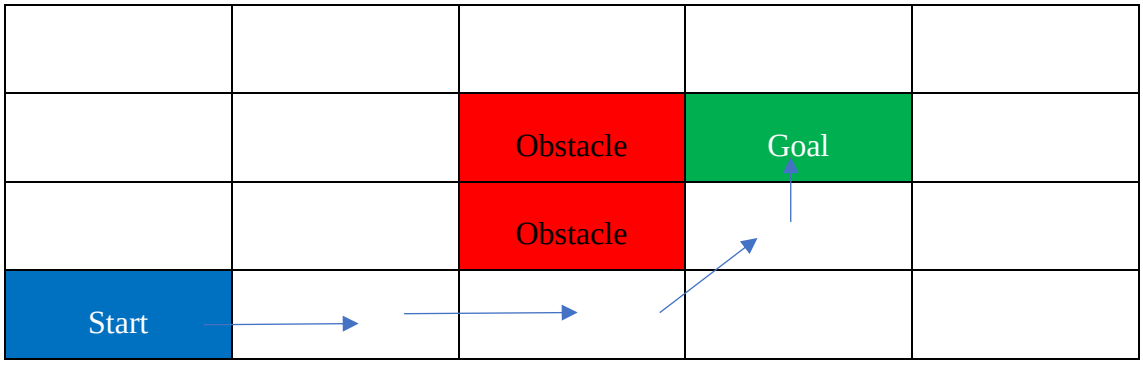
	A	B	C	D	E
1	30 A2	34 B2	38 B2	48 C1	Inf
2	20 A3	24 B3	Obstacle	Goal 44 D3	48 D3
3	10 A4	14 A4	Obstacle	34 C4	44 D4
4	Start 0	10 A4	20 B4	30 C4	40 D4

At this moment the goal will be in the closed list making us able to extract the path

Which will be D2-D3-C4-B4-A4 (notice D2 is the goal and A4 is the start)

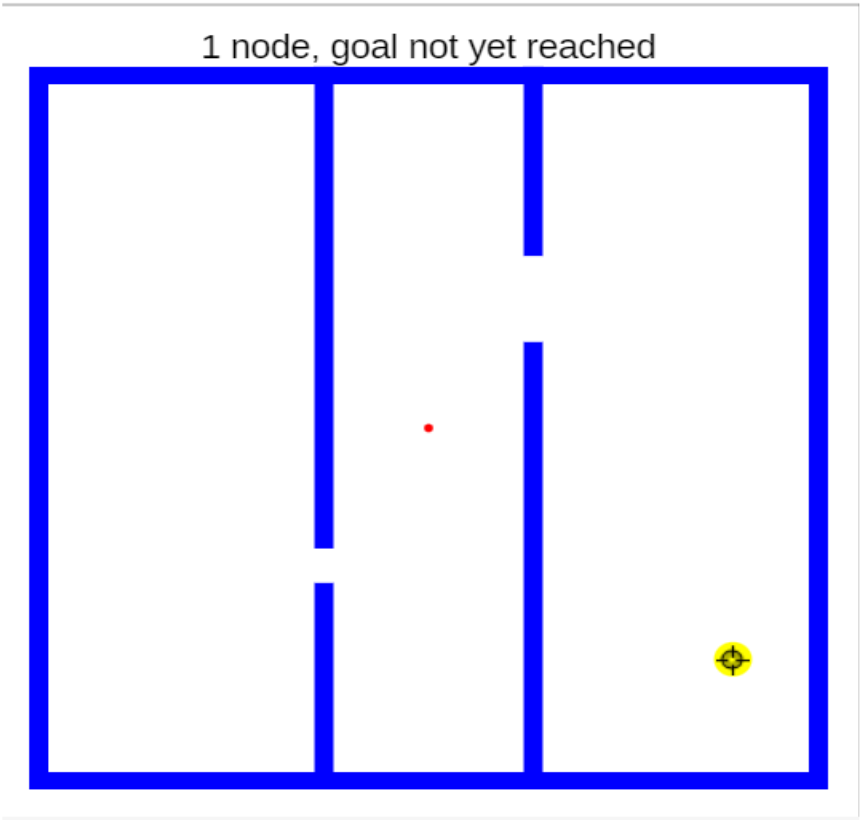
Path:

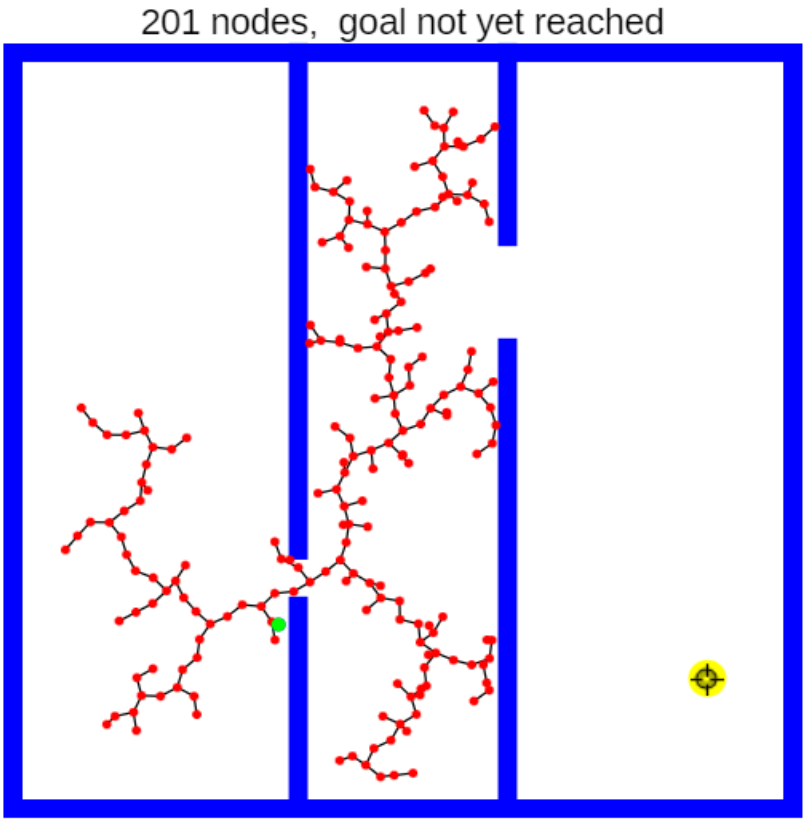
	A	B	C	D	E
1					
2					
3					

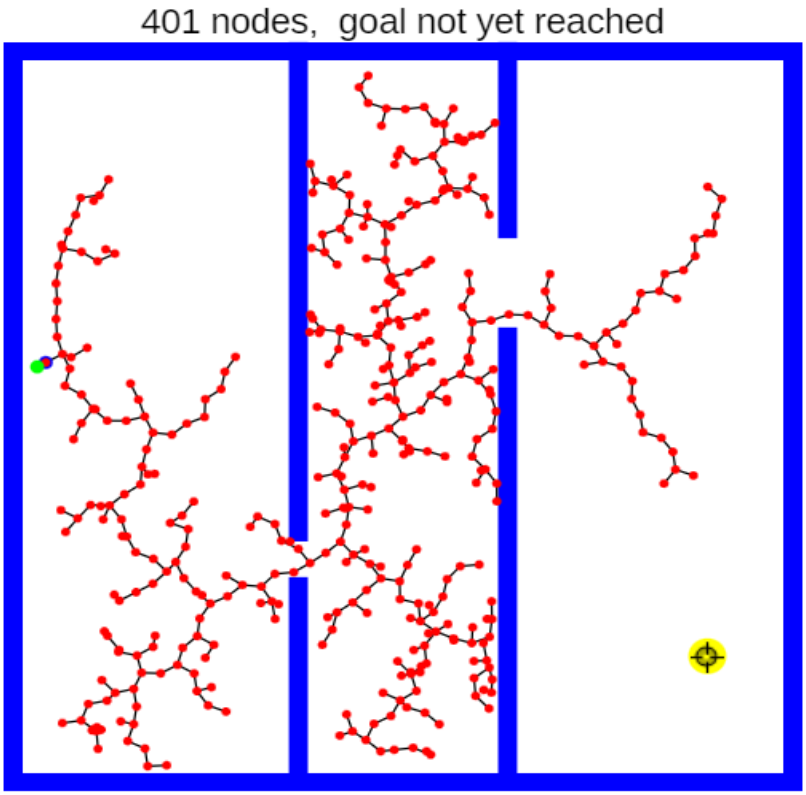

Obstacle
Goal
Obstacle
Start

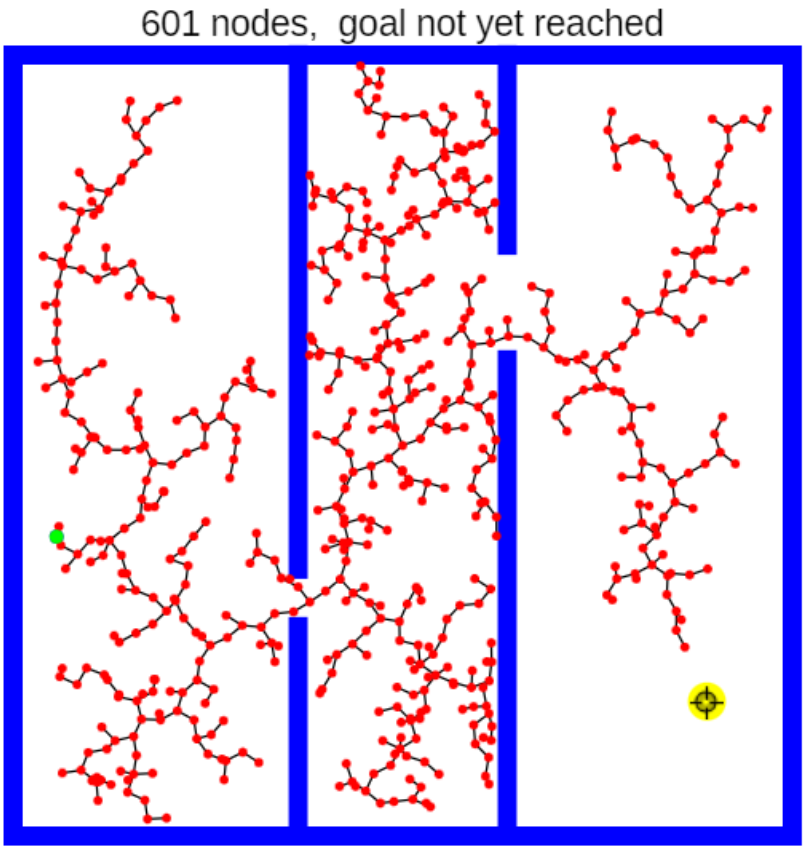
10.7 RRT

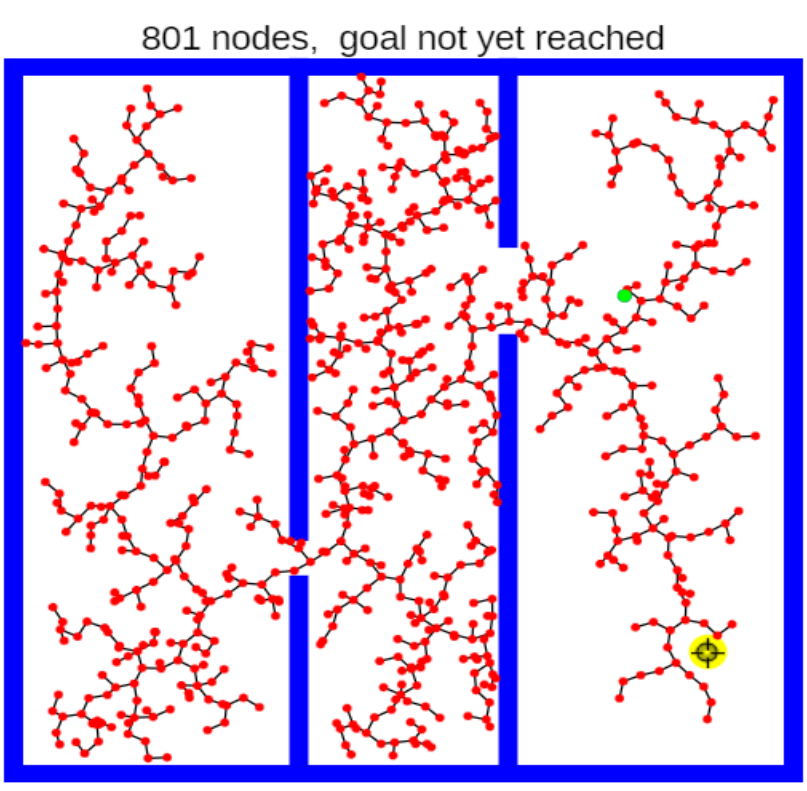
Step by step RRT Path Planning

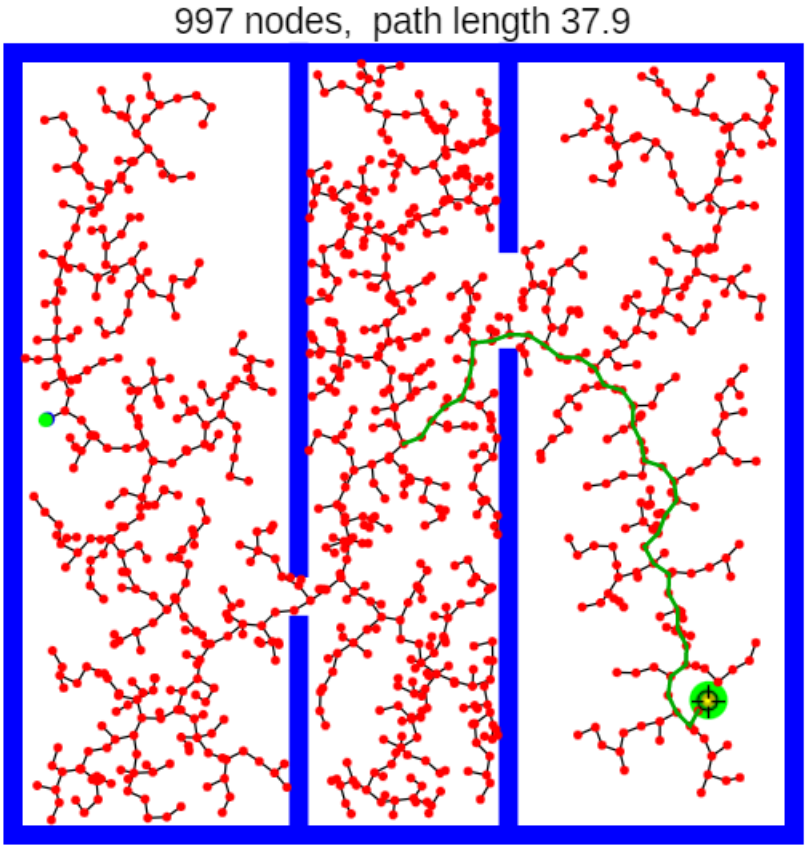



201 nodes, goal not yet reached


401 nodes, goal not yet reached

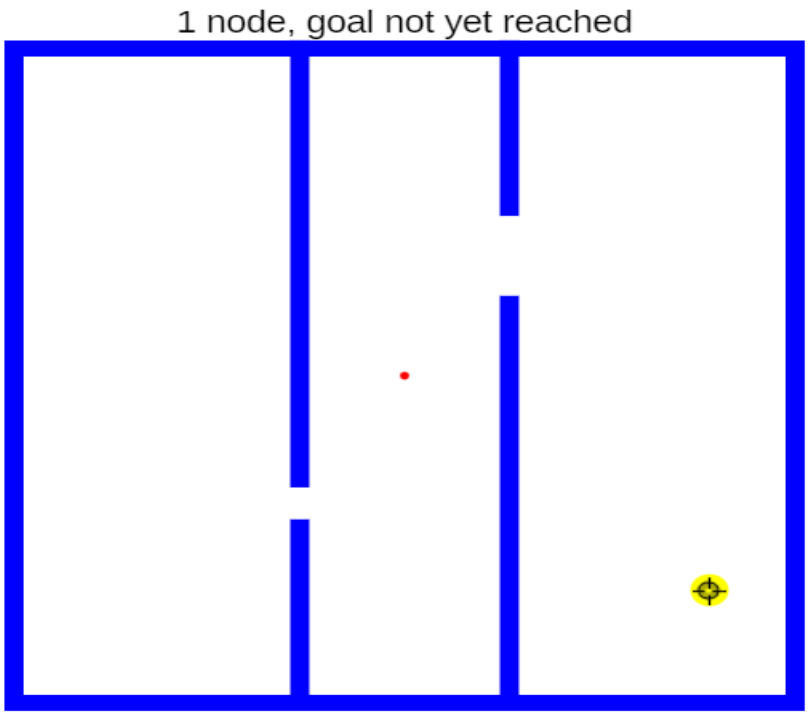

601 nodes, goal not yet reached


801 nodes, goal not yet reached

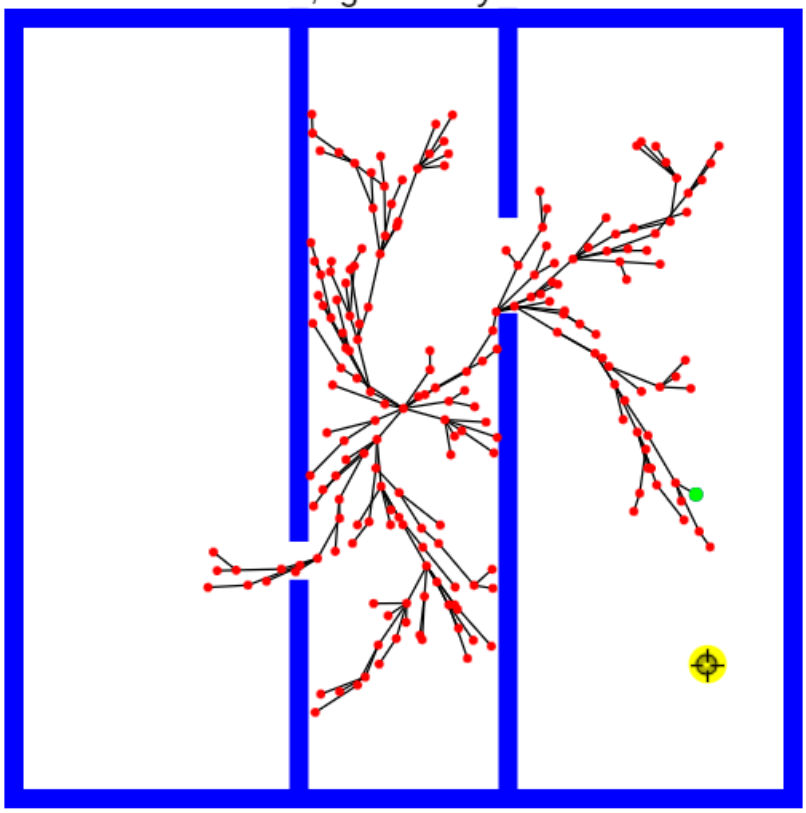

997 nodes, path length 37.9

10.8 RRT STAR

Step by step RRT* Path Planning



201 nodes, goal not yet reached



397 nodes, path length 30.53