Autonomous Electric Vehicle

Submitted By:
Abdelrahman Atef Mansour
Doaa Amir Zeidan
Mahmoud Emad Sayeh
Omar Atef Thabet
Youssef Ragaa Zakaria
Zeyad Yasser Sayed

Mechatronics Engineering and Automation

Assistant Prof. Mohamed Ahmed Mahmoud Abdelwahab

Date: 09/2/2022

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 partial fulfilment of BSc in ***Mechatronics Engineering and Automation*** is entirely our own work, that we have exercised reasonable care to ensure its originality, and does not to the best of our 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 work.

Signed: by all students

Omar Atef Thabet	17P8177
Mahmoud Emad Sayeh	17P87205
Youssef Ragaa	17P8154
Doaa Amir Zeidan	17P8146
Abdulrahman Atef Mansour	17P8185
Zeyad Yasser Sayed	16P8149

Date: 09/2/2022

ACKNOWLEDGMENT

We would like to express our appreciation to Assistant Prof. Mohamed Abdelwahab for all the effort and support that he provided throughout the project. He taught us the correct methodology to carry out 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 attempts of our team to create a prototype for an autonomous vehicle with a miniature car. Firstly, we discuss autonomy in general in regards to current advancements in the field, challenges and importance. There after we discuss the key steps involved in converting our vehicle to a suitable platform for implementing the various lower and higher level control algorithms needed for autonomy. Namely; the algorithms required for control of the various actuators present on the vehicle, as well as the algorithms responsible for generating the vehicle path and interacting with the environment as needed.

We will discuss the experimental stage where multiple algorithms are tested for each specific task and showcase our results and hence in light of these finding the algorithm that was implemented on board the vehicle. Lastly we discuss how the vehicle can be further developed to reach higher functionality and our conclusions on the topic and the limitations of current technology for the implementation of autonomy.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE: *INTRODUCTION*

1.1. CURRENT STATE AND MOTIVES FOR AUTOMATION

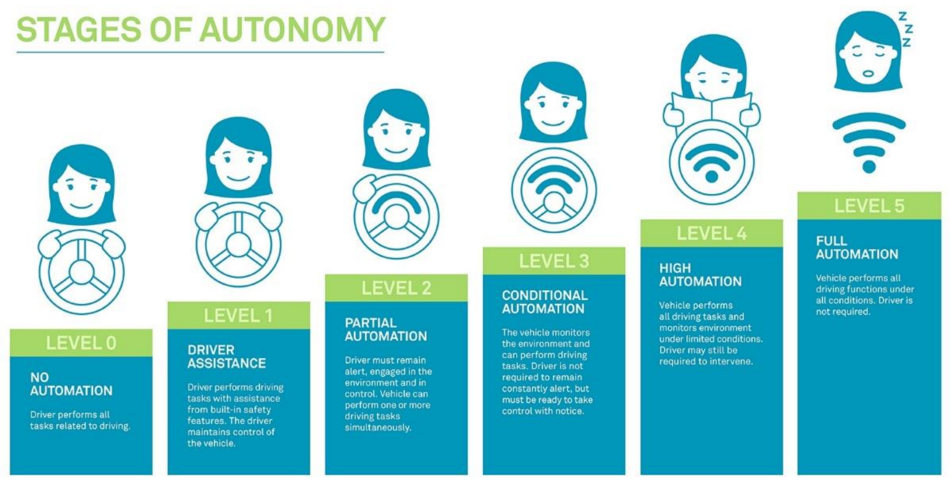
With strict government regulations coming out on emissions from internal combustion engines, the automotive industry is moving towards electrification. This further leads to the ease of development of autonomous navigation systems implemented in modern vehicles, since the electric platform facilitates the control required with autonomy. In addition, the advancements in technology and intelligent software design have enabled the progression of autonomous technology in vehicles in a dramatic pace. And as said previously having the vehicle be electric makes the control of the vehicle much simpler which facilitates the automation process. One of the key points in favour of automation is road safety and it is a much contested point with varying views from the public so it will be discussed separately.

1.2. DEGREES OF AUTOMATION

Any self-driving vehicle can be categorized based on the degree of automation implemented. The stages of automation range from zero to five, with zero being a vehicle without any autonomy where the driver must perform all the driving tasks, and five being complete autonomy where the driver needn't participate in any driving task and all of the vehicle functions and road navigation is carried out automatically by the vehicle. There lie some stages in the middle with varying degrees of automation and that is where we find ourselves today.

The first stage is driver assistance, where the driver is still responsible for driving the vehicle but with some on-board systems that make this task easier and or intervene in cases of emergency. Examples of this are ESP and ABS safety systems which can intervene during vehicle operation for short periods, but the driver will take over control shortly afterward. The second stage is partial automation where the driver is still responsible for observing and reacting to road conditions, but the vehicle can perform some driving tasks for extended periods, an example of this would be adaptive cruise control systems.

The third stage is conditional autonomy, this is when the driver can partake in other activities during their presence inside the vehicle but must be ready to take control of the vehicle if needed. Meanwhile, the on-board systems can carry out the vehicle operation and can monitor the environment and react accordingly. The fourth stage is high autonomy, where the vehicle navigate independently of the driver in specific conditions and the driver may be required to intervene here as well.


STAGES OF AUTONOMY

LEVEL 0
NO AUTOMATION
Driver performs all tasks related to driving.

LEVEL 1
DRIVER ASSISTANCE
Driver performs driving tasks with assistance from built-in safety features. The driver maintains control of the vehicle.

LEVEL 2
PARTIAL AUTOMATION
Driver must remain alert, engaged in the environment and in control. Vehicle can perform one or more driving tasks simultaneously.

LEVEL 3
CONDITIONAL AUTOMATION
The vehicle monitors the environment and can perform driving tasks. Driver is not required to remain constantly alert, but must be ready to take control with notice.

LEVEL 4
HIGH AUTOMATION
Vehicle performs all driving tasks and monitors environment under limited conditions. Driver may still be required to intervene.

LEVEL 5
FULL AUTOMATION
Vehicle performs all driving functions under all conditions. Driver is not required.

Table 1 Stages of Autonomy

1.3. CHALLENGES FOR AUTOMATION

The industry at the moment is at stage four of autonomy with increased effort to achieve complete autonomy. This, however, is easier said than done and as will be discussed in detail later, there arises many difficulties when trying to get a vehicle to traverse a human made road network without supervision. This is due in part to the fact that human roads are simply designed for humans, they require human perception to navigate and make decisions in real time about the best course of action during emergencies or even during normal operation, regarding road signs and traffic laws and such. Another major factor is other human drivers sharing the road with the autonomous vehicle. Since human driving behaviour is unpredictable and sometimes almost chaotic, it becomes increasingly difficult to have a vehicle interact with fellow human drivers.

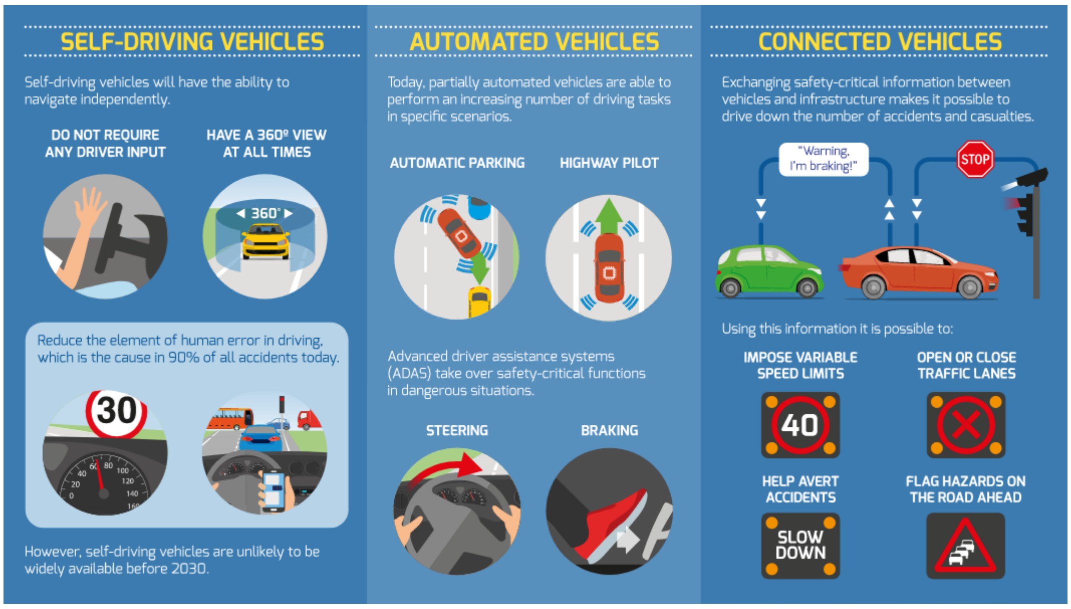
1.4. HUMAN ERROR

There is no denial that human error has been the largest contributor for vehicle accidents as mentioned by a 2016 study by the National Highway Transportation Safety

Administration (NHTSA) [A14] which found that human error accounts for anywhere between 94% to 96% of all road accidents. Furthermore, other studies have supported these results proving that the percentage is at least 90%. And in the first nine months of 2021, the (NHTSA) reported an estimate of 31,720 people have been killed in vehicle crashes. While, The World Health Organization (WHO) estimated that more than 1.3 million people die each year as a result of road traffic collisions.

Hence, engineers have been working on two approaches to minimize those errors, one of which is adding active safety systems and Advanced driver assistance systems (ADAS) to the vehicles which can intervene to correct a potential error that a driver is about to make such as ABS system which prevents panic braking from locking the wheels.

However, another approach that has seen major development in the recent years is "Autonomous Vehicles" also known as Self-Driving vehicles. This approach is believed by self-driving advocates that it can eliminate the human element error from the roads. Autonomous vehicles, which as mentioned by the www.RoadSafetyFacts.eu which is powered by The European Automobile Manufacturers' Association (ACEA), hold great potential to further improve road safety. Indeed, many of today's active safety technologies are already helping to prepare road users for a future when vehicles take over control from drivers.



Self-driving vehicles will have the ability to navigate independently.

Reduce the element of human error in driving, which is the cause in 90% of all accidents today.

However, self-driving vehicles are unlikely to be widely available before 2030.

Today, partially automated vehicles are able to perform an increasing number of driving tasks in specific scenarios.

Advanced driver assistance systems (ADAS) take over safety-critical functions in dangerous situations.

Exchanging safety-critical information between vehicles and infrastructure makes it possible to drive down the number of accidents and casualties.

Using this information it is possible to:

Table 2 Requirements of Autonomy

1.5. REQUIREMENTS FOR AUTONOMOUS VEHICLES

Autonomous vehicles rely on path planning, perception, control and localization, which requires precise knowledge of the vehicle position and orientation, to fit within the autonomy principles which include safety integrity level, defining the allowable probability of failure per hour of operation based on desired improvements on road safety today and to maintain knowledge that the vehicle is within its lane and to determine what road level it is on.

CHAPTER *TWO: GLOBAL PATH PLANNING*

NAVIGATION

Global path-planning is the first step of navigation. Different algorithms are tested to choose which path planning technique is most suitable for the level 5 autonomous car project. All algorithms are given a start position and a goal position. Some algorithms take the car dimensions and dynamic model, which might be given if needed.

2.1. A-STAR ALGORITHM

A star also known as “best-first algorithm” is one of the most popular well know path planning algorithms that first started around the 1970[A6] and since then it’s still a necessary algorithm that must be considered in the subject of autonomous motion planning. A-star is an informed search algorithm that depends on the algorithm to know the position of the goal point. The main idea of the A-star algorithm is that the path is decided using the Euclidean-heuristic function to get the shortest, best way possible. Euclidean heuristic is the function that calculates the smallest straight line between two points as shown in [Figure 1].

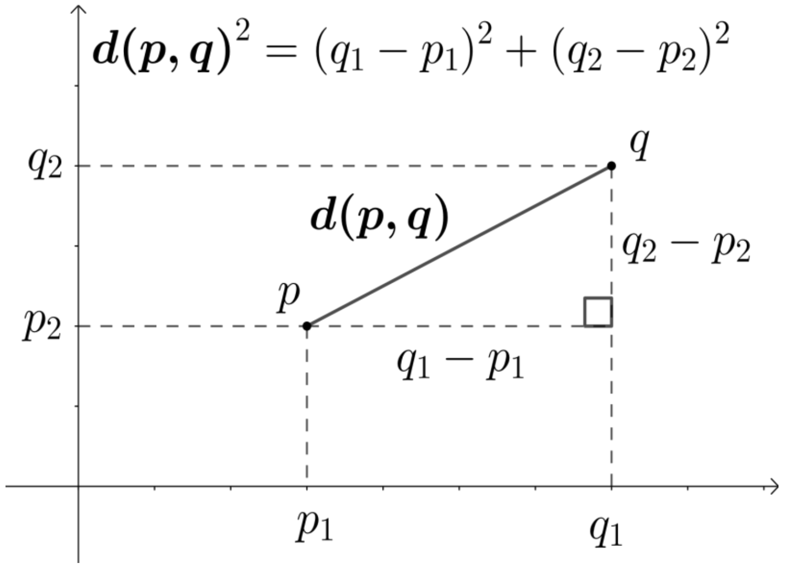


Figure 1 Euclidean distance calculations

The A-star is implemented as follows. First the map is divided into grids. The greater the number of grids the smoother and more efficient the path is going to be, but the complexity and computational timing will be compromised. Then each grid is given a distance number as shown in [Figure 3]. This value is the Euclidean distance calculations from the point to the goal point. Therefore, the algorithm searches for the shortest distance from the next point to the goal point neglecting the points where obstacles are included and the point its already on. As in [Figure 2]

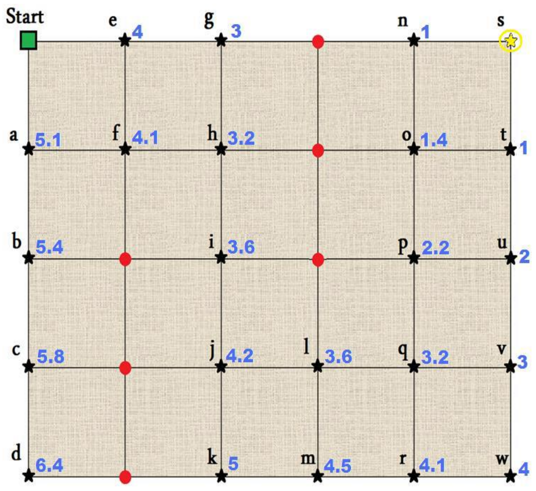


Figure 3 Euclidean dist. grid

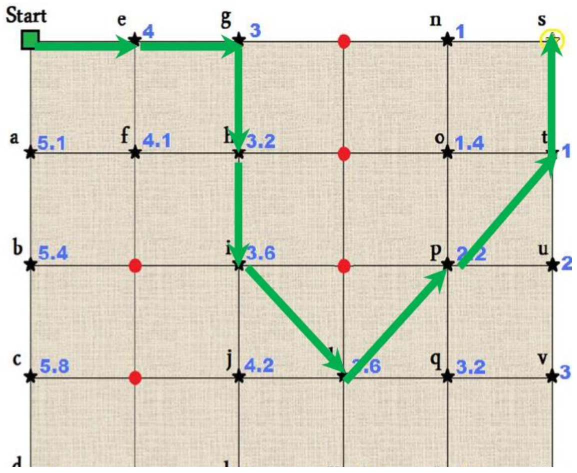


Figure 2 : path created using A-star

This graph-based search algorithm is great when taking in consideration the cost function but has its constraints[A8]. A-star can get trapped if an obstacle is too close to the goal and the planner would prefer to go through the obstacles than change its path to a point with larger distance to the goal. That's why A-star is also known the greedy algorithm. It also doesn't deal well with manoeuvres as it doesn't take in consideration the dynamics of the vehicle. That's where Hybrid A-star seps in.

2.2. HYBRID A-STAR

The main problem with A-star is that is deals with each point discretely with zero consideration of the next point. Which creates many complications when tested in the real world However, hybrid A-star solves this problem by solving the navigation path planning as a continuous function rather than discrete. Hybrid A-star is a derivation of the classic A-star algorithm, but it takes in consideration the minimum turning angle and dimensions of vehicle. A-star takes only sharp angles that is nearly impossible for a four-wheel Ackerman steering vehicle. But the hybrid reshapes the sharp A-star path for it to be possible to for the vehicle to manoeuvre. This result is obtained by optimizing the A-star algorithm. After calculating the Euclidean distance of each point, a cost function is calculated that considers the turning angle to the path and the next point. And so on, the algorithm keeps updating the next point to fit the given cost function till the goal is reached.

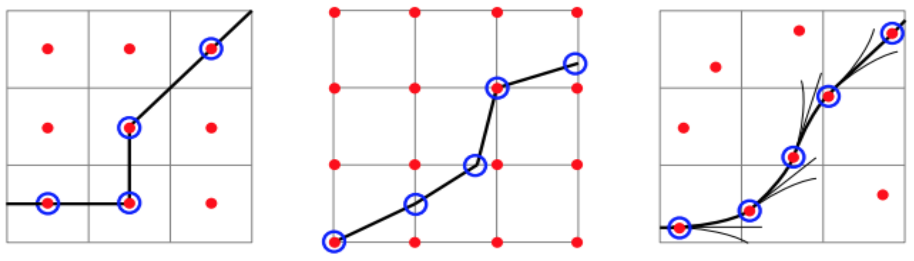
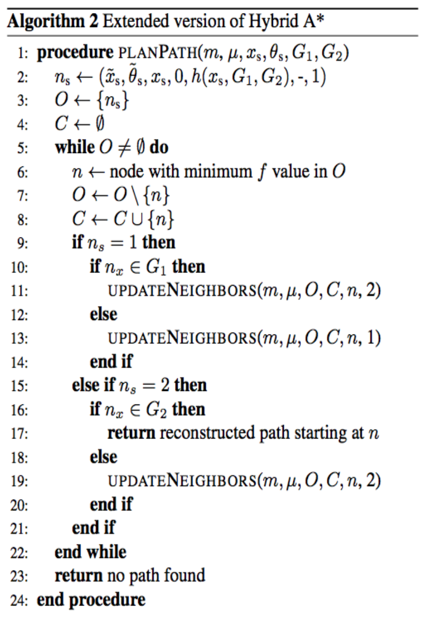


Figure 4 first graph represents A-star algorithm , second and third represents Hybrid A-star algorithm

Figure 4 compares between the A-star and hybrid A-star . its obvious that a-star creates sharp turns that can’t be handled by any vehicle, even after smoothing it a bit in the second example its not smooth enough for it to be acceptable.

The algorithm of hybrid A-star can go as follows



Algorithm 2 Extended version of Hybrid A*

```
procedure PLANPATH($m, \mu, x_s, \theta_s, G_1, G_2$)
    $n_s \leftarrow (\tilde{x}_s, \tilde{\theta}_s, x_s, 0, h(x_s, G_1, G_2), \text{-}, 1)$
    $O \leftarrow \{n_s\}$
    $C \leftarrow \emptyset$
    while $O \neq \emptyset$ do
        $n \leftarrow$ node with minimum $f$ value in $O$
        $O \leftarrow O \setminus \{n\}$
        $C \leftarrow C \cup \{n\}$
        if $n_s = 1$ then
            if $n_x \in G_1$ then
                UPDATENEIGHBORS($m, \mu, O, C, n, 2$)
            else
                UPDATENEIGHBORS($m, \mu, O, C, n, 1$)
            end if
        else if $n_s = 2$ then
            if $n_x \in G_2$ then
                return reconstructed path starting at $n$
            else
                UPDATENEIGHBORS($m, \mu, O, C, n, 2$)
            end if
        end if
    end while
    return no path found
end procedure
```

Figure 5 Hybrid A-star algorithm

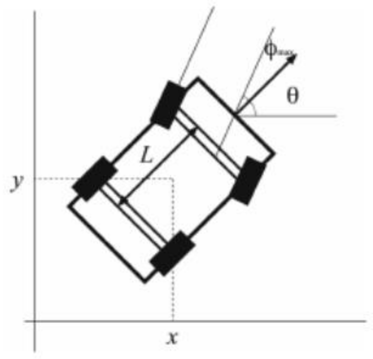


Figure 6 simple car model

2.3. RRT (RAPID-EXPLORING RANDOM TREE)

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 wouldn’t 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.

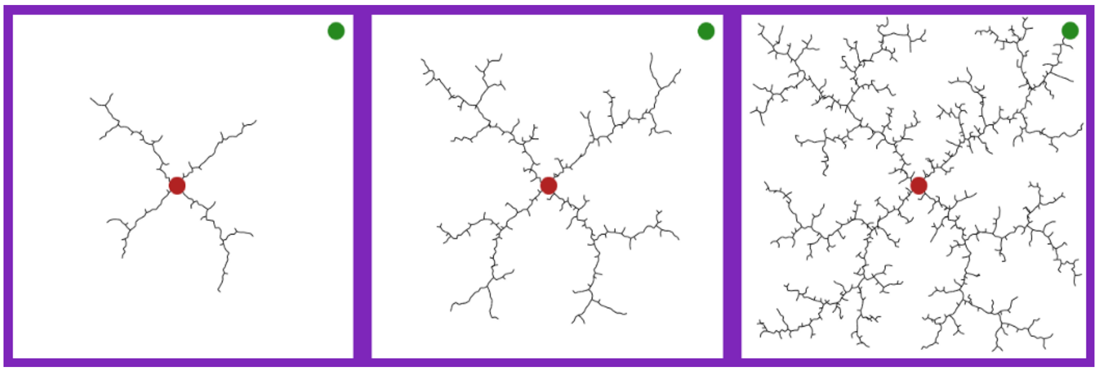


Figure 7 RRT tree generation

2.4. RRT STAR

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 isn’t 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, as introduced by Karaman and Frazzoli [A9].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

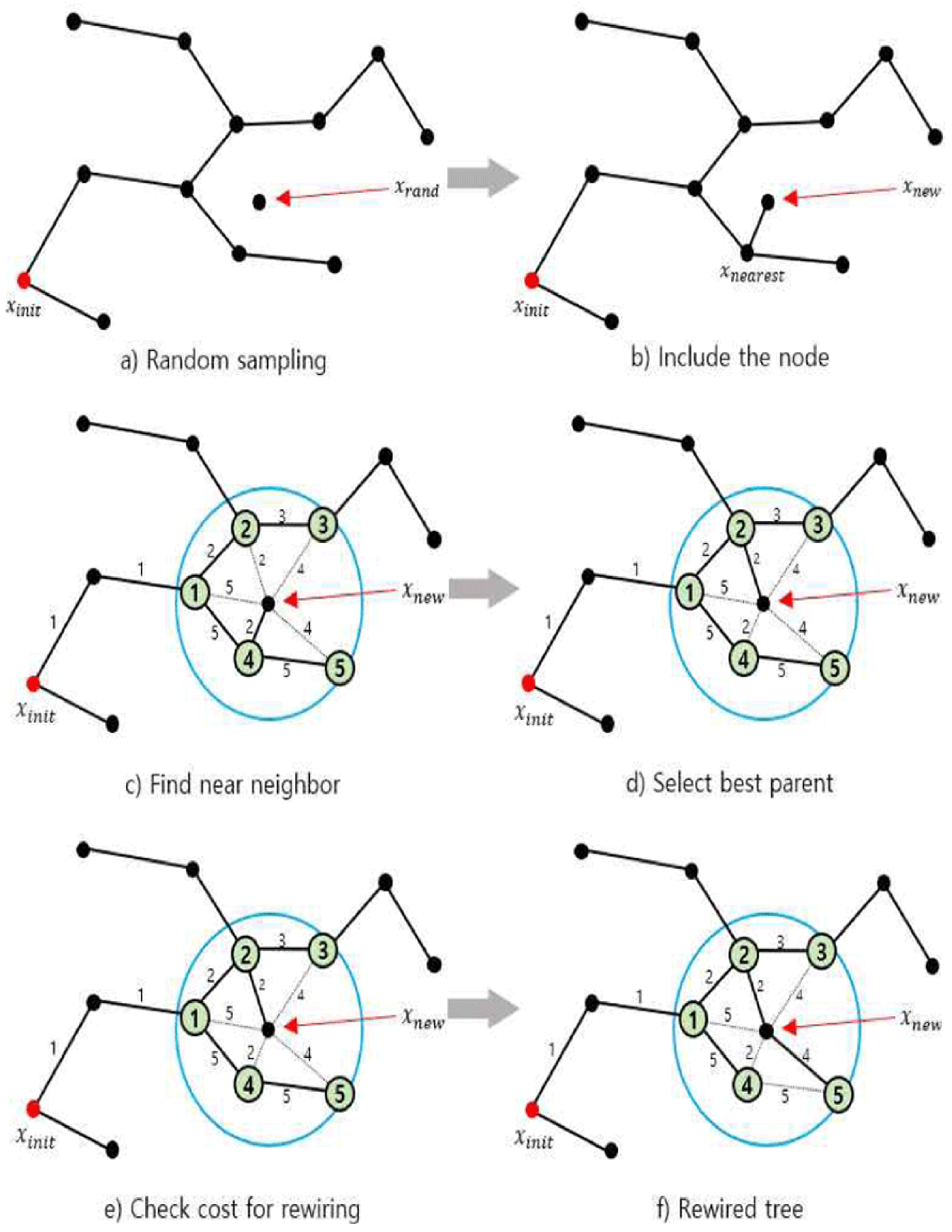


Figure 8 RTT Star nearest neighbor rewriting operation

2.5. BI-RRT

Another RRT optimized algorithm is the bi-RRT or the bidirectional-RRT algorithm. As discussed before, RRT starts with random point generated and checked till goal is reached. But this method is time consuming and takes memory spaces. Around 2010, developers thought that it might be easier if RTT was an informed search technique rather than a blind search[A9]. Informed search algorithm operates with the knowledge of where exactly the goal is placed.

The bi-RRT works by generating random nodes not only from the start point but also from the goal. And finally, when both trees collide the path between them is the path selected. Bi-RRT has proven to be very much time saving compared to the normal RRT searching algorithm.

Although bi-RRT saves computational time relative to RRT its not the best optimal path planning algorithm. Because like the family of RRT it doesn’t take in consideration the vehicle dimensions and creates a sharp path that’s not suitable for every vehicle.

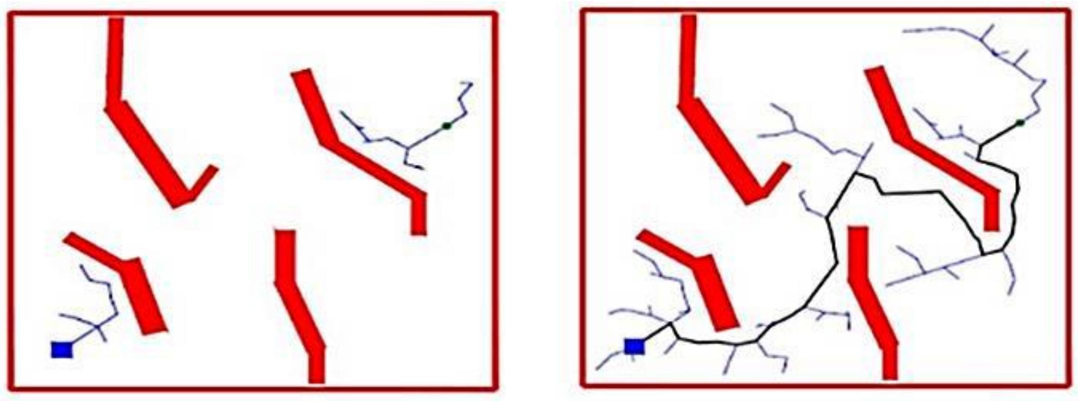


Figure 9 Bi-RTT tree generation

IMPLEMENTATION

We have implemented the path planning algorithms using MATLAB and utilizing its Navigation Toolbox ^{TM} motion planning tools [A13].

2.6. PRE-PROCESSING

Initially, we acquired our faculty's map by scanning the map provided during exams and scaling it using google Earth. Furthermore, this map required some pre-processing before being used by the motion planning API's. Hence, the image of the map obtained was imported to the MATLAB in PNG format [Figure 10 Image of Faculty of Engineering Ain Shams Map] then was converted to grayscale then converted to a binary occupancy map which is basically a digital representation of a map that the motion planning algorithms can interpret [Figure 11 Binary Occupancy Map of faculty of engineering Ain Shams University]. It consists of a matrix of zeros and ones, which represents empty and occupied blocks respectively. Occupancy maps can be obtained from either a handwritten matrix, an image, or data acquired from sensors in PMG format.

Finally, the map's occupied cells were inflated by 50 cm as a safety to account for the wheel track and prevent the vehicle from being too close to the obstacles.

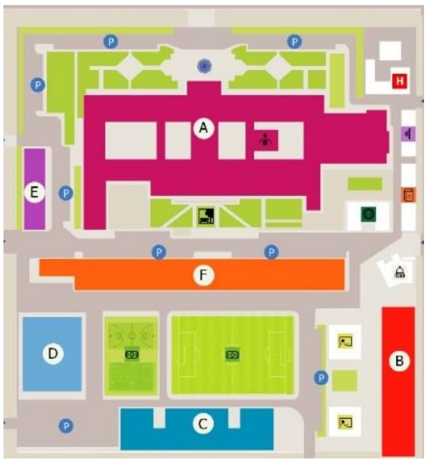


Figure 10 Image of Faculty of Engineering Ain Shams Map

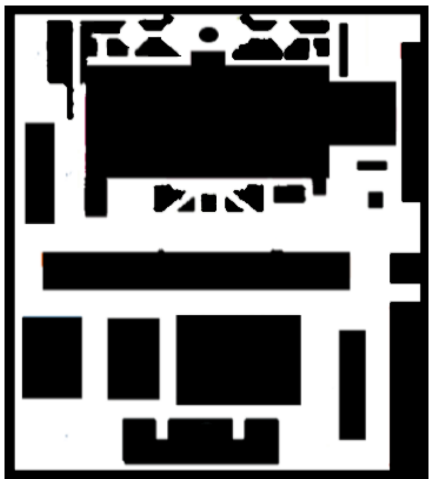


Figure 11 Binary Occupancy Map of faculty of engineering Ain Shams University

2.7. ALGORITHMS

The motion planning algorithms used were Hybrid A Star, RRT, RRT* and Bi-RRT. A `stateSpaceSE2` object had to be created from the map, which is object stores parameters and states in the SE(2) state space, which is composed of state vectors represented by [x, y, θ]. x and y are Cartesian coordinates, and θ is the orientation angle. The object uses Euclidean distance to calculate distance and uses linear interpolation to calculate translation and rotation of the state, as well as a `StateValidator` object that validates states and discretized motions based on the value in a 2-D occupancy map. An occupied map location is interpreted as an invalid state. Finally, the [x, y, θ] start and end goal vectors are initialized before implementing the path planning algorithms.

For Hybrid A Star, a planner object is created that takes `StateValidator`, `stateSpaceSE2` and minimum turning radius of the vehicle. Then the planning algorithm is initiated to plan a path from the start goal to the end goal. Added to that, max connection distance and max number of iterations are other parameters that were initialized for the other algorithms.

2.8. RESULTS

The algorithms were tested on two different input passes and their output was visualized and compared. The first path represented an S-shape starting from the bottom right corner as shown in Figure 13. While the second path was a line starting from the bottom left as shown in Figure 17.

2.8.1 First Path

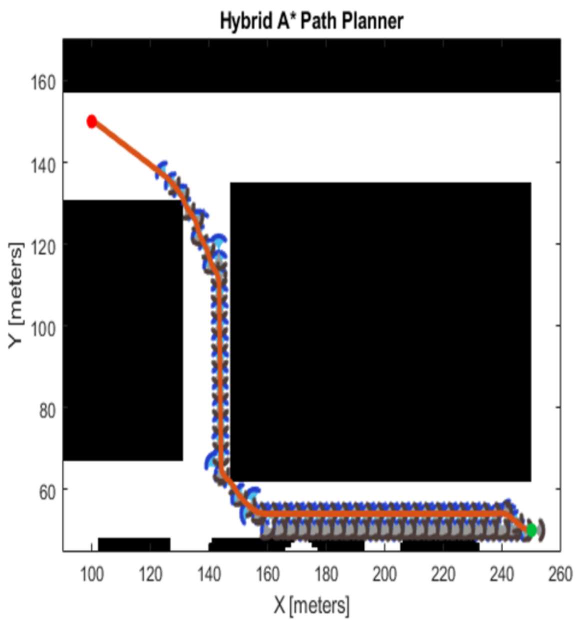


Figure 13 Hybrid A* First Path

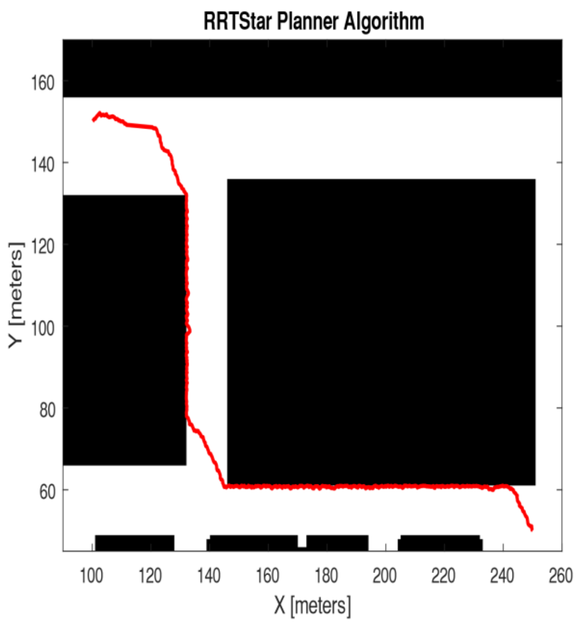


Figure 12 RRT Star First Path

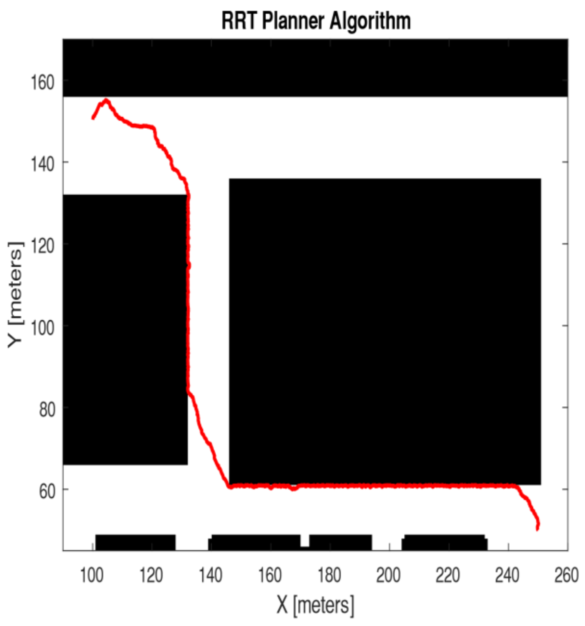


Figure 15 RRT First Path

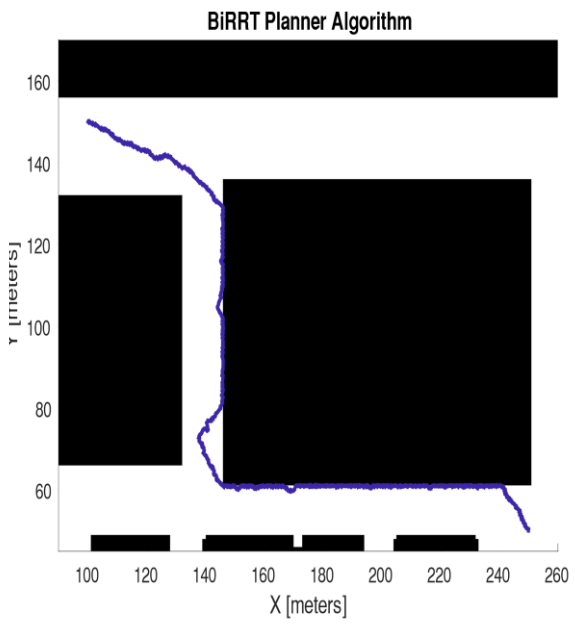


Figure 14 Bi-RRT Star First Path

2.8.2 Second Path

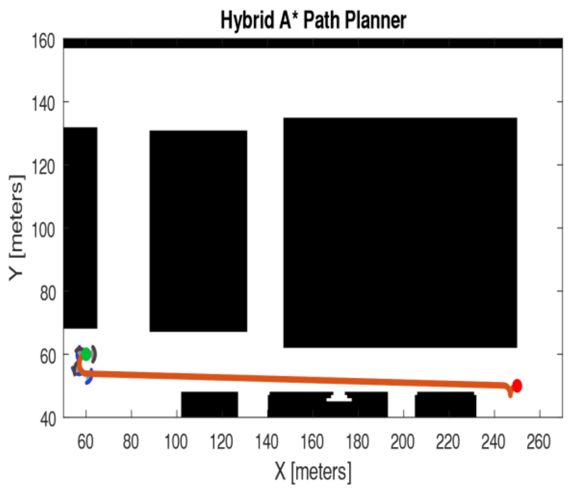


Figure 17 Hybrid A* Second Path

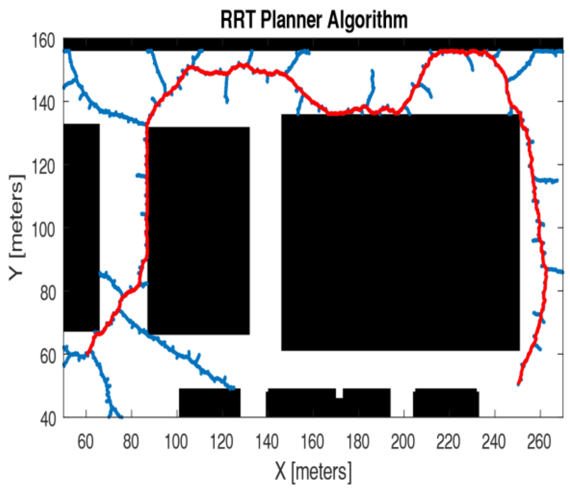


Figure 18 RRT Second Path

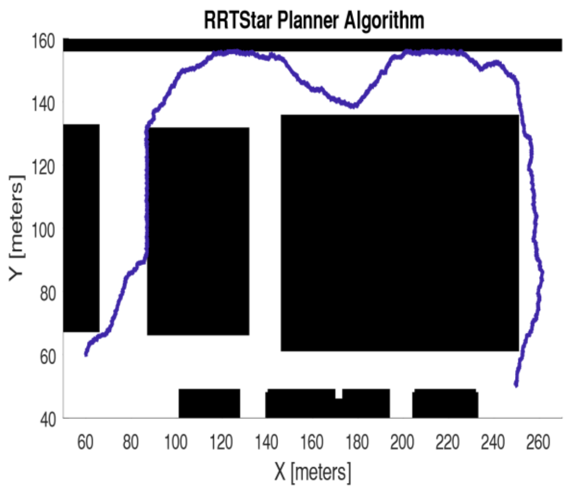


Figure 16 RRT* Second Path

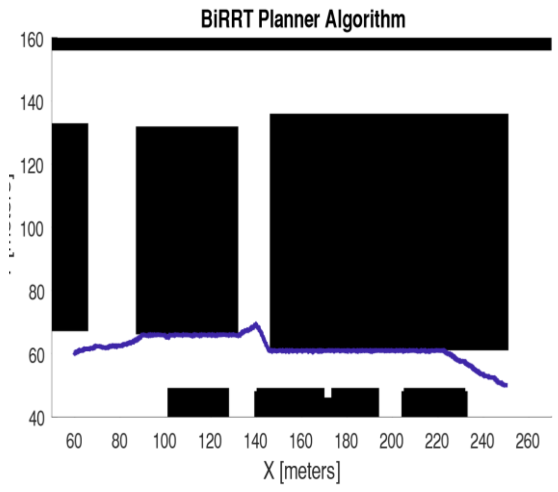


Figure 20 Bi-RRT Second Path

2.8.3 Spider Plot

The algorithms were compared based on 4 factors: computational time, path length, smoothness and minimum clearance distance. These factors were represented using a spider plot Figure 21.for easier comparison. While the generated paths and minimum clearance distances were visualized on the map.

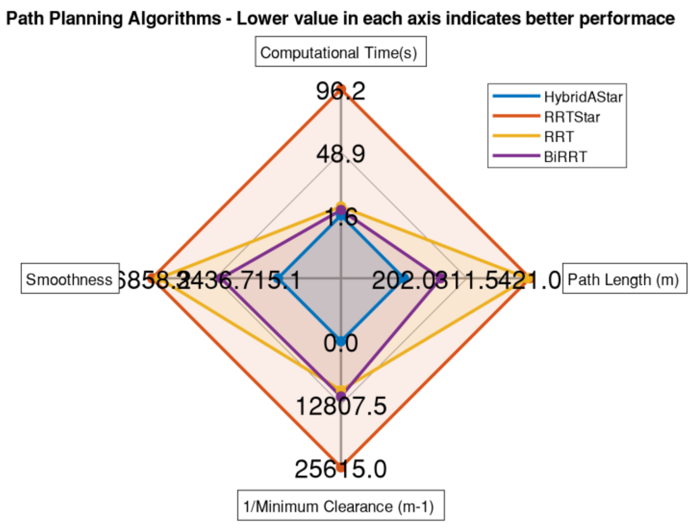


Figure 21 First Path Results

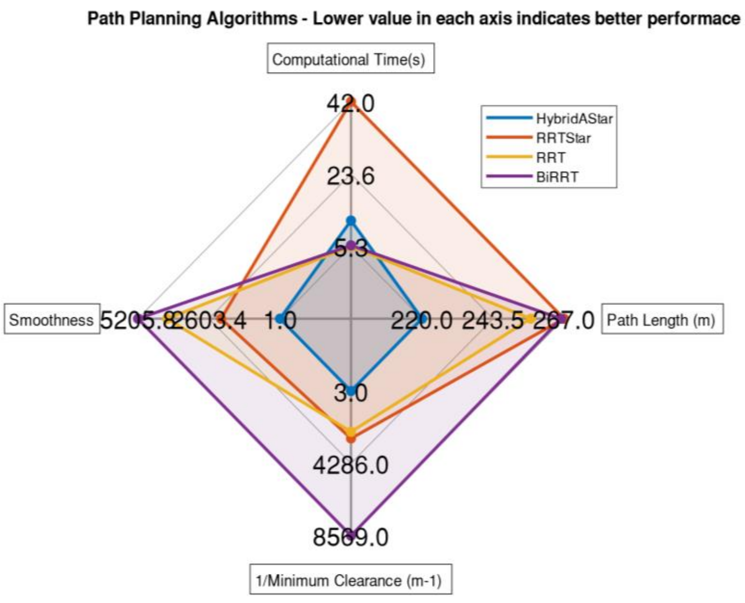


Figure 22 Second Path Results

2.9. DECISION MATRIX

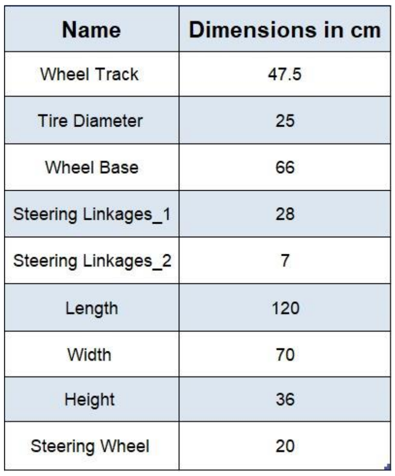
		Motion Planning Algorithms							
Section & Requirements	Level of Importance	Hybrid A*Star		RRT		RRT*		Bi-RRT	
		Score	Total	Score	Total	Score	Total	Score	Total
Path1									
Computational Time	2	5	10	4	8	1	2	4	8
Length	3	5	15	1	3	1	3	3	9
Smoothness	2	5	10	1	2	1	2	3	6
1/Minimum Clearance	1	5	5	3	3	1	1	3	3
Total			45		16		8		26
Path 2									
Computational Time	2	4	8	5	10	1	2	5	10
Length	3	5	15	2	6	2	6	4	12
Smoothness	2	5	10	2	4	1	2	1	2
1/Minimum Clearance	1	5	5	3	3	3	3	1	1
Total			**38**		**23**		**13**		**25**

Table 3 Path Planning Algorithms Decision Matrix

CHAPTER *THREE: LOCAL PATH PLANNING*

VEHICLE MODIFICATION

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. The parameters of the vehicle can be found in **Error! Reference source not found.** all of which were measured by hand.

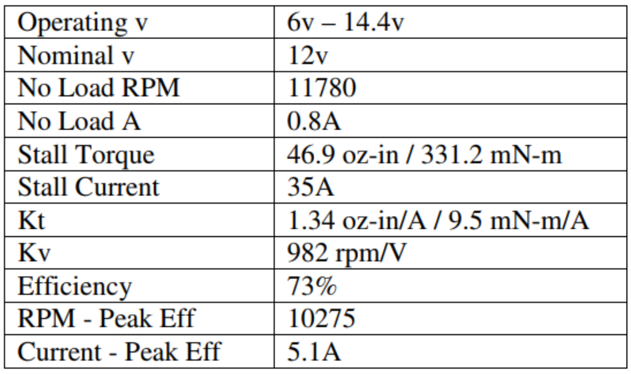


Name	Dimensions in cm
Wheel Track	47.5
Tire Diameter	25
Wheel Base	66
Steering Linkages_1	28
Steering Linkages_2	7
Length	120
Width	70
Height	36
Steering Wheel	20

Figure 23 Vehicle Dimensions

3.1 PROPULSION SYSTEM

The vehicle contains a DC motor driving each of the rear wheels through a speed reduction gearbox. The motor specifications and the gearbox schematic can be seen in **Error! Reference source not found.** and **Error! Reference source not found.** respectively. 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 24 DC motor Specifications

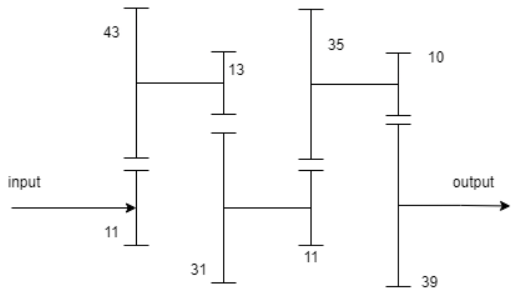


Figure 25 Propulsion gearbox schematic

Upon observing the performance of the vehicle it was concluded that the motors were powerful enough for all intents and purposes. And so, the DC motors were unchanged but an essential modification must have been made which is the installation of a wheel encoder in order to be able to apply the chosen control algorithm on the drive wheels. The encoder required for this purpose is an incremental encoder to detect the speed of the wheels. The encoder acquired produces 360 pulses per revolution which allows for very fine control of the driven wheel.

Figure 26 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.

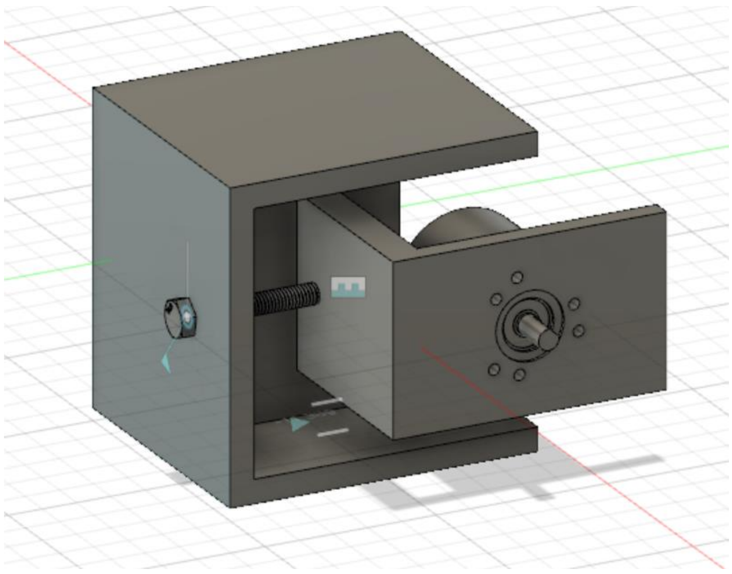


Figure 27 Encoder bracket CAD design

3.2 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.

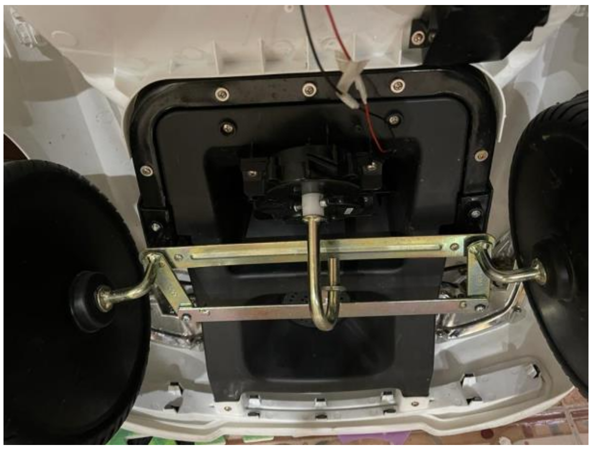


Figure 28 Steering linkages

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.

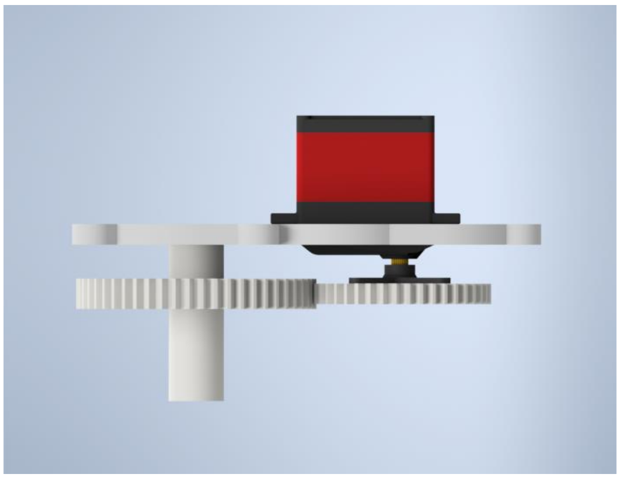


Figure 29 Steering gearbox CAD design

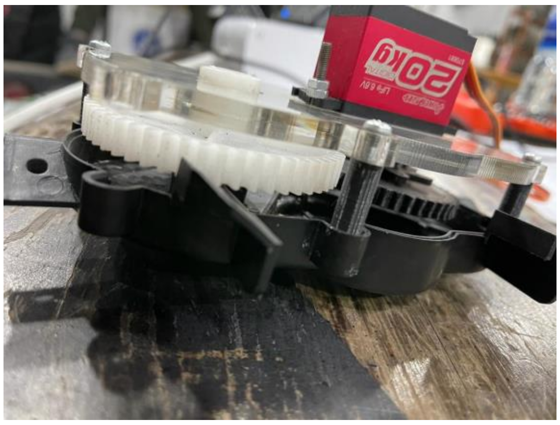


Figure 30 Steering gearbox Assembly

CONTROL

Three controllers were places on board the vehicle, each of which is responsible for a separate module. The first controller and the master controller is a raspberry pi which is responsible for the higher level control. It imports the generated path and implements the path following algorithm to find the desired speed and steering angle which are sent to the lower level controllers to actuate the steering and propulsion mechanisms. The lower level control consists of two Arduinos; one controls the steering module and the other controls the two motors of the propulsion system.

The controllers are able to communicate with each other to convey important inputs such as vehicle speed and odometry data from one module to the other. The layout of control architecture and communication protocols used can be seen in **Error! Reference source not found.**. The diagram abstractly illustrates the connections between the controller and various actuators and sensors used in the vehicle. The block diagram seen in Fig 32: illustrates the flow of control between these electronics and how data is passed around to generate the desired movement of the vehicle.

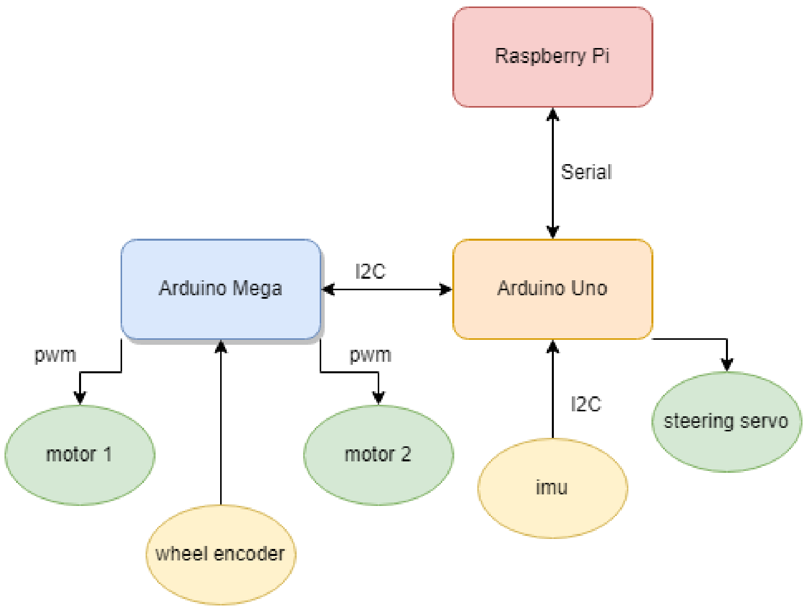


Figure 31 Control hierarchy

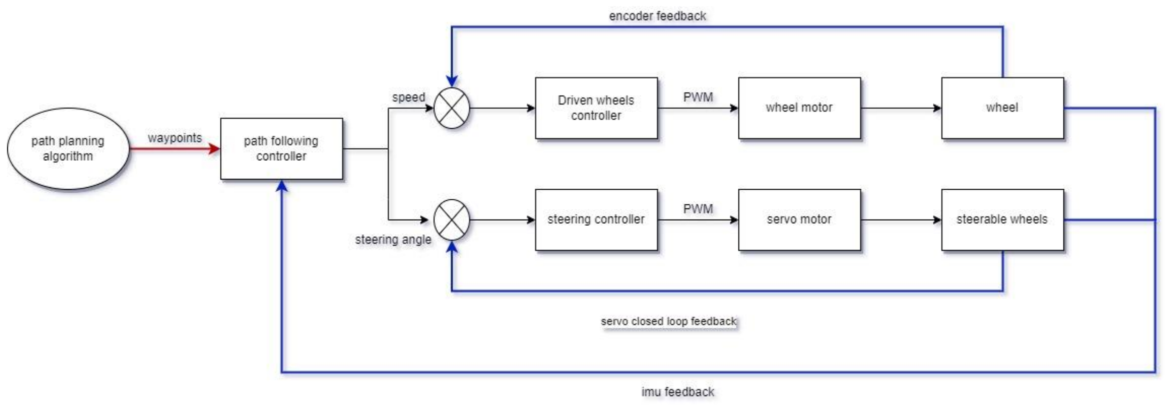


Figure 32 system block diagram

ODOMETRY

The localisation of the vehicle is done in two parts the first part is determining the distance travelled by the vehicle from its initial starting point. The second part is determining the orientation of the vehicle at any given time. Hence, the global coordinates of the vehicle in two dimensions can be calculated in reference to the original starting frame. The distance calculation can be done using the readings of the wheel encoder and assuming negligible slipping of the wheel on the ground the rotational motion of the wheel can be translated into linear motion according to the following equation.

$$\text{Distance} = r_w * \theta$$

Where:

$r_{w:}$ is the radius of the car wheel
θ: is the rotations of the car wheel in radians

The next stage is the pose estimation which can be achieved using the inertial measurement unit (IMU) placed on board the vehicle. The IMU produces a reading corresponding to the yaw rate of the vehicle, this reading can be integrated to produce the yaw angle of the vehicle at any time. The pose data can be used to resolve the distance data into it two components to find the global X and Y positons of the vehicle. This is illustrated in Fig 33.

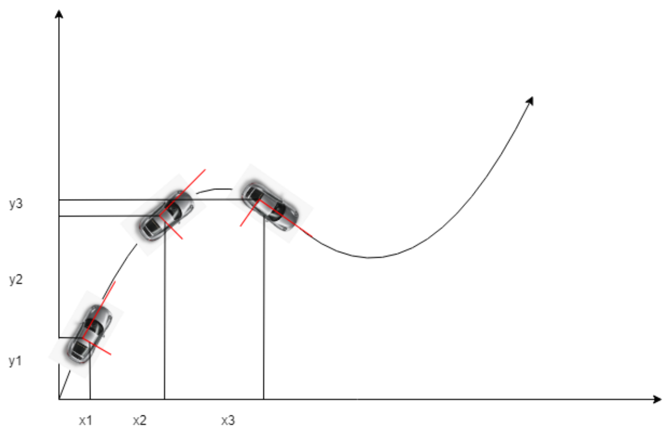


Figure 33 localization

And the global positions can be calculated using the following equations:

$$dx = dh\cos(yaw)$$
$$dy = dh\sin(yaw)$$

Where dx and dy are finite elements corresponding to the resolution of the dh element taken from the path of the vehicle travel. These can then be summed to add to the global position of the vehicle.

ELECTRICAL

The main power unit of the vehicle is a 60Ah, 12 volt car battery. The battery is required to power all the motors as well as the electronics required to operate the vehicle. The usage of the car battery required the implementation of some safety measures to ensure that none of the components would be damaged from over current or short circuits. A protection circuit was designed to protect the system from over current and reverse polarity in case the connection terminals to the system were switched. The short circuit protection circuit can be found in **Error! Reference source not found.** and its main principle of operation is a relay which is normally closed but when a short happens the coil energizes and cuts off power to the remainder of the circuit. The reverse polarity protection circuit can be seen in **Error! Reference source not found.** and its main idea is a MOSFET which acts as a switch turning off power to the electronics downstream of it in case of reverse polarity.

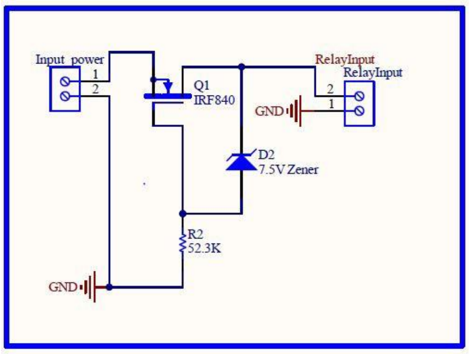


Figure 34 Reverse polarity protection

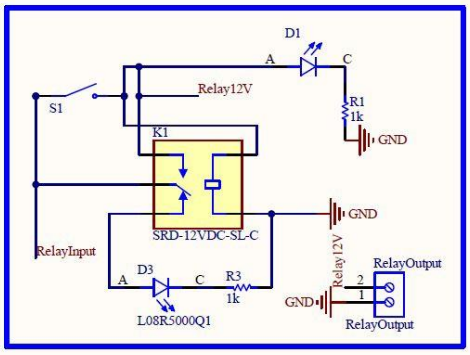


Figure 35 Short circuit protection

All these circuits were housed in one printed circuit board (PCB) which allows for the ease of integration of all the necessary circuits and connections between the various electronics and controllers. Both the Arduino microcontrollers are placed directly on the board and there are terminals going out to connect to the various actuators and power supply.

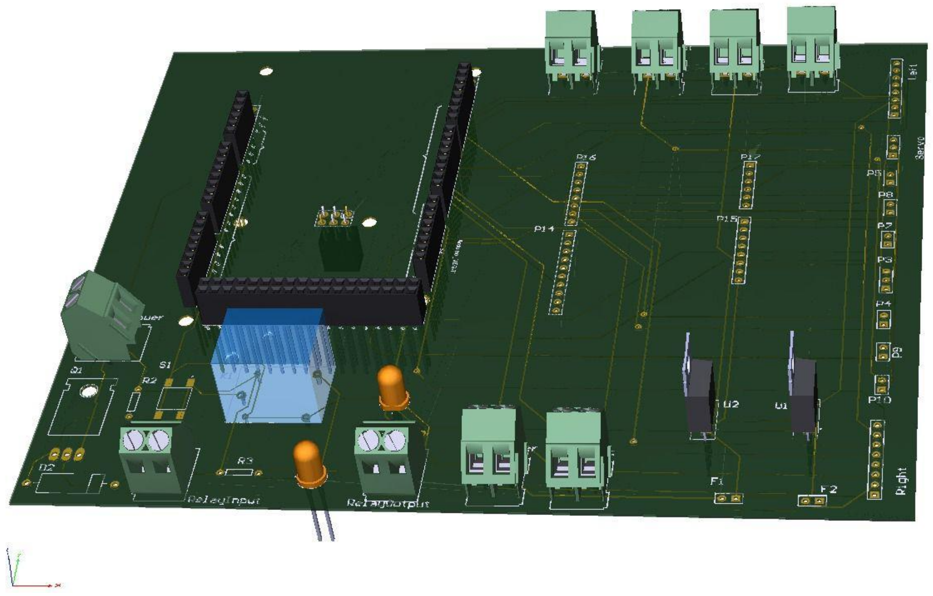


Figure 36 PCB 3D view

The electronics which are not placed on the PCB such as the motor drivers and the raspberry pi were all placed inside a fabricated box which houses all of the electronics together and is fastened to the vehicle body. The electronic box also houses fans which allows for the dissipation of the heat generated by the drivers or the controllers.

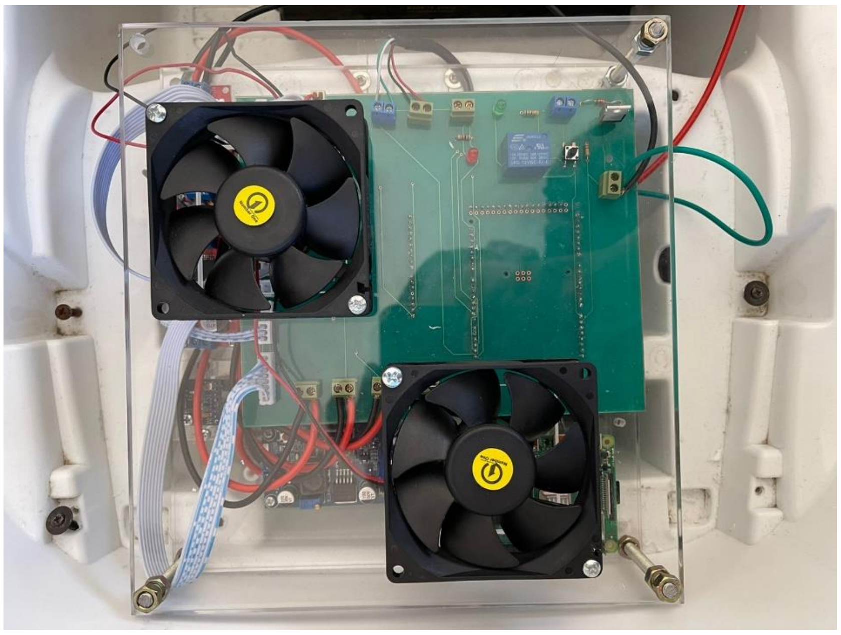


Figure 37 Electronics box

CHAPTER FOUR: GLOBAL PATH FOLLOWING

This section involves the implementation of path following techniques to move the vehicle across the path that was generated from the map by the Global Path Planning phase. The planned path is a set of waypoints with coordinated x and y.

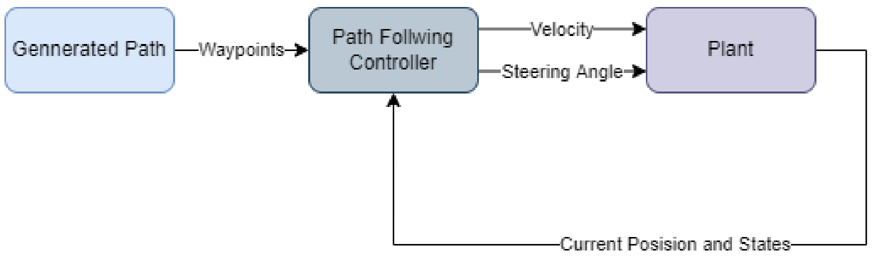


Figure 38 Basic Flow of Path Following

The general flow of the algorithm includes the measurement of the current position and states of the vehicle and returning it as feedback to the controller. The controller then finds a way to move the vehicle towards the path usually by changing the steering angle to one that sends it towards the trajectory.

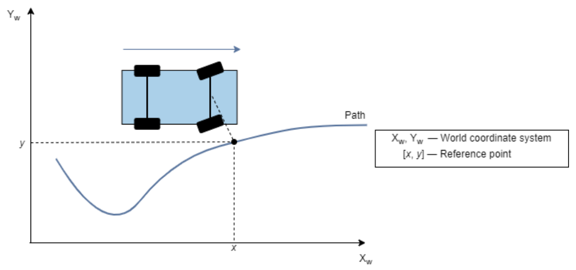

Y_w
Path
y
X_w, Y_w — World coordinate system
$[x, y]$ — Reference point
x
X_w

Figure 39 Path Following Diagram

The path following algorithm was implemented with Simulink that used ROS blocks to help publish data for a node to subscribe and change the steering angles of the vehicle for the low-level control.

There are several examples of Trajectory tracking controllers that were studied in the project.

1. Pure Pursuit
2. Stanley Controller
3. Model Predictive Controller

4.1 THEORY BEHIND EACH CONTROLLER

This section involves studding the theory behind some of the controllers used.

4.1.1 Kinematic Bicycle Model of the Vehicle

Before explaining how the controller works, it’s important to derive a model for our system. In our case, the vehicle is moving with a low speed, so it was acceptable to model it as a kinematic bicycle model. This model involves simplifying the vehicle to have only two wheels, one rear wheel without steering, and one front wheel that steers to change the direction of the vehicle trajectory [A1]

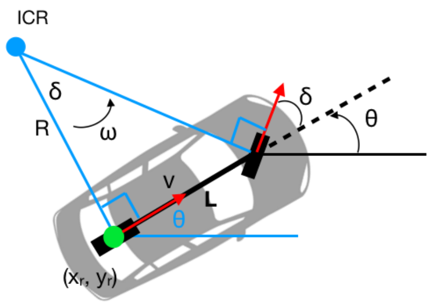


Figure 40 Kinematic Bicycle Model

In our model, the states that were analysed are the following [A1]:

X: The x coordinate of the vehicle.

y: The y coordinate of the vehicle.

θ: The Heading angle.

δ: The Steering angle.

The inputs are:

v: The Longitudinal Velocity

φ: The Steering Rate

Using resolution, we get that the following derivatives of the states:

$$\dot{x} = v \cos \theta$$

$$\dot{y} = v \sin \theta$$

The change of the heading angle is equivalent to the rate of rotation about the instantaneous center of rotation. Since $\omega = V/R$, and from the triangle

$R = L/\tan(\delta)$ we can determine that: $\dot{\theta} = \omega = v\tan(\delta)/L$

The derivative of the steering angle is equal to the steering rate therefore:

$$\dot{\delta} = \phi$$

The system can be represented with the following equations:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} cos(\theta) \\ sin(\theta) \\ tan(\delta)/L \\ 0 \end{bmatrix} V + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \varphi$$

4.1.2 Equation of motion of a two degree of freedom bicycle model:

<u>Assuming:</u>

- Steering angles is small so, $\cos(\delta) = 1$
- Linear Tire model
- Longitudinal velocity is constant

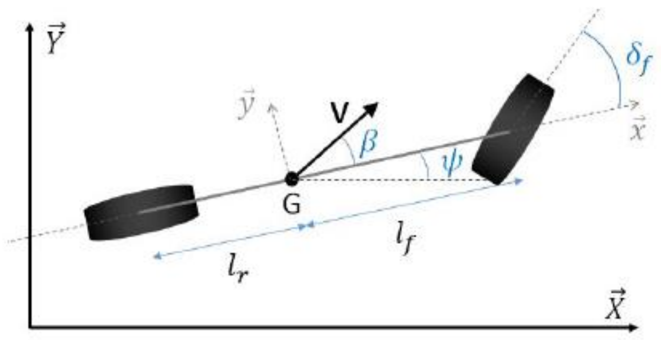


Figure 41 Bicycle Model Diagram

Using Newton’s second law to the equation of motion (Rajamani)of the two degrees of freedom:

$$m\left(\ddot{y} + \dot{\Psi}\,\dot{x}\right) = F_{yf} + F_{yr} \text{ (Lateral dynamics)}$$

$$I\ddot{\Psi} \;= L_f F_{yf} - L_r F_{yr} \text{ (Rotational dynamics)}$$

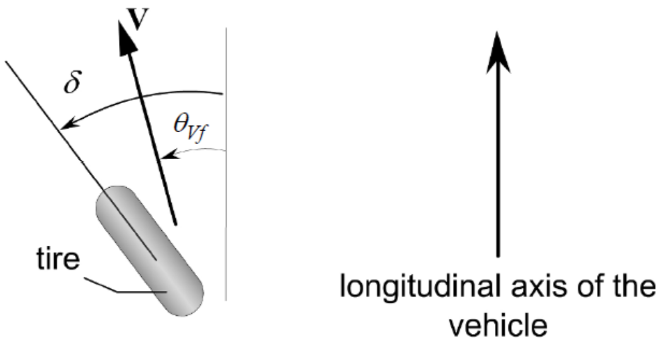


Figure 42 Tire model

And to get the forces on the front and back tire (Jason Kong, 2015):

$F_{yf} = -c_{\alpha,f}\alpha_f$ were, $\alpha_f = \delta - \theta_{v,f}$

$F_{yr} = -c_{\alpha,r}\alpha_r$ were, $\alpha_r = -\theta_{v,r}$

$\theta_{v,f} = \frac{\dot{y} - l_f\dot{\Psi}}{\dot{x}}$ were, $\tan(\theta_{v,f}) = \theta_{v,f}$

$\theta_{v,r} = \frac{\dot{y} - l_r\dot{\Psi}}{\dot{x}}$ were, $\tan(\theta_{v,r}) = \theta_{v,r}$

And by substituting in the two equation of motion and making the $\ddot{y}$ and $\ddot{\psi}$ in one side of the equation we can drive the state space model (Rajamani):

State Space model:

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2c_{\alpha,f} + 2c_{\alpha,r}}{m\,\dot{x}} & 0 & -\dot{x} - \frac{2c_{\alpha,f}l_f + 2c_{\alpha,r}2l_r}{m\,\dot{x}} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{2l_f c_{\alpha,f} - 2l_r c_{\alpha,r}}{I\dot{x}} & 0 & -\frac{2l_f^2 c_{\alpha,f} + 2l_r^2 c_{\alpha,r}}{I\,\dot{x}} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{c_{\alpha,f}}{m} \\ 0 \\ \frac{l_f c_{\alpha,f}}{I} \end{bmatrix} \delta$$

4.2 PURE PURSUIT CONTROLLER

The Pure Pursuit path tracking algorithm can determine the steering angle of the vehicle by having it find a point across the path that is a certain lookahead distance from the robot and calculate the steering angle required to reach this [A2]

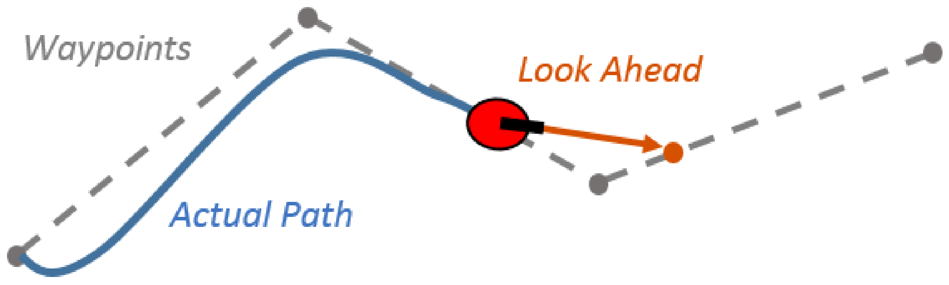


Figure 43 Pure Pursuit Description

Steps of the Pure pursuit algorithm include [A2]:

Measure the current position of the vehicle to get the coordinates initial coordinates

1. Find the closes point of the robot to the path to select a point ahead of it.

2. Select a point ahead of the path that is equal to or greater than the lookahead distance.

3. Measure the actual distance of the point from the vehicle and the angle of the line with the vehicle.

4. Calculate the desired Steering angle using the equations.

This controller is one of the most basic and easy to implement methods of path following; though it has some limitations which include not stopping at the desired end goal so the application must take into consideration stopping the vehicle and the robot doesn't follow the exact path that is required. After selecting the target position that is equal to the lookahead distance parameter and connecting the rear wheel with the arc, we can derive the equation that represents the controller [A3]: Connecting the two points with the instantaneous center of rotation will display a triangle. Using the Alternate Segment Theorem, we get that the arc angle is equal to 2α.

Using sin rule:

$$\frac{ld}{sin2\alpha} = \frac{R}{sin(\frac{\pi}{2} - \alpha)}$$
$$\frac{ld}{2sin\alpha cos\alpha} = \frac{R}{\cos{(\alpha)}}$$
$$\frac{ld}{sin2\alpha} = 2R$$

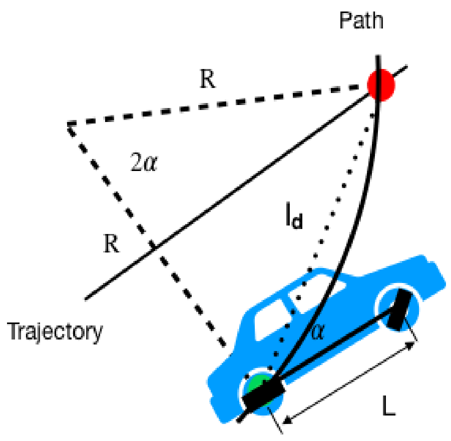


Figure 44 pure pursuit diagram

The curvature of the arc expressed as follows:

$$k = \frac{1}{R} = \frac{2sin\alpha}{ld}$$

We can use the bicycle kinematic model to get the desired steering angle:

$$R = \frac{L}{\tan{(\delta)}}$$
$$\delta = \arctan{(\frac{2Lsin\alpha}{ld})}$$

The lookahead distance parameter is the only parameter that must be tuned. Depending on it, the response of the system may vary. On one hand, if the distance it too small, then the vehicle will oscillate a lot while staying close to the path; on the other hand, if its large, there will be less oscillations, but the vehicle will reach the path at a more gradual pace. The lookahead distance must be tuned based on the desired response [A2]:

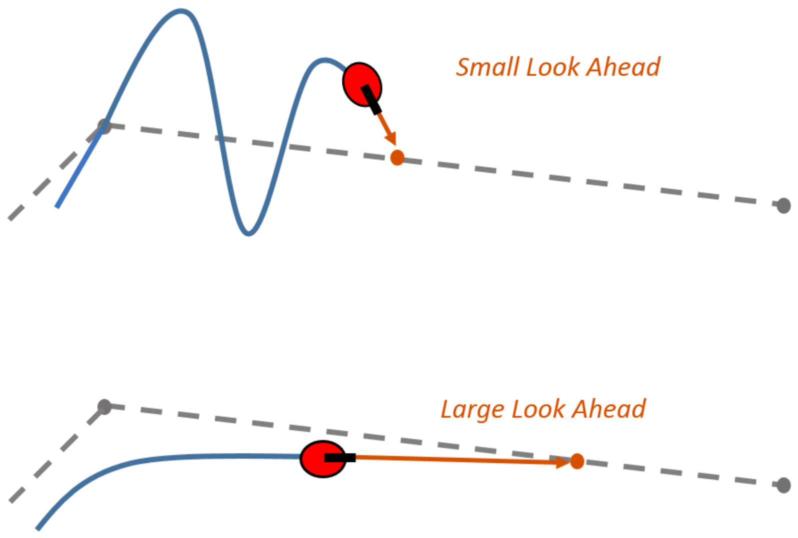


Figure 45 Pure Pursuit with Different Lookaheads

4.3 STANLEY CONTROLLER

This controller was first used by Stanford University's DARPA challenge. It takes the front axle as the reference on the vehicle. The Stanley controller aims to do the following [A3]

1. Fix the heading error ψ(t)
2. Fix the cross-track error e(t)
3. Stay within a predetermined steering bound $[\delta_{min},\delta_{max}]$.

We get the following equation:

$$\delta(t) = \psi(t) + \arctan\left(\frac{ke(t)}{v_f(t)}\right)$$

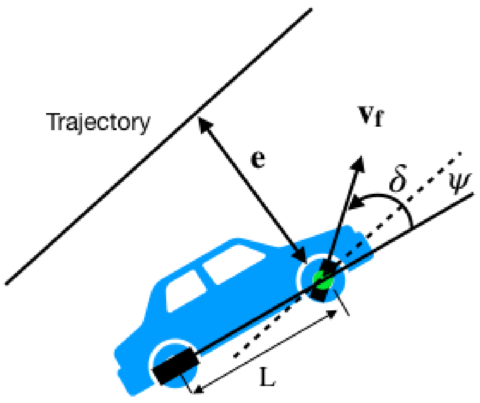


Figure 46 Stanley Controller Diagram

The response of the Stanley controller is much more stable than the pure pursuit as it doesn't oscillate around the path. The reason for that is that the since the correction of both heading and cross track errors are considered, they are not independent. For instance, if the heading error is zero and there is a cross track error, then the vehicle will steer towards the path which by reducing the e(t) it will increase the cross-track error toward the posit direction which will smoothen the steering angle [A3].

4.4 MODEL PREDICTIVE CONTROLLER

Model predictive controllers use plant, disturbance, and noise models for prediction and state estimation, MPC is based on iterative, finite-horizon optimization of a plant model. At time t the current plant state is sampled and a cost minimizing control strategy is computed (via a numerical minimization algorithm) for a relatively short time horizon in the future: [t,t+T]. Specifically, an online optimization is used to explore state trajectories that emanate from the current state and find a cost-minimizing control strategy until time t+T. Only the first step of the control strategy is implemented, then the plant state is sampled again, and the calculations are repeated starting from the new current state, yielding a new control and new predicted state path (Alberto Bemporad, N. Lawrence Ricker, Manfred Morari)

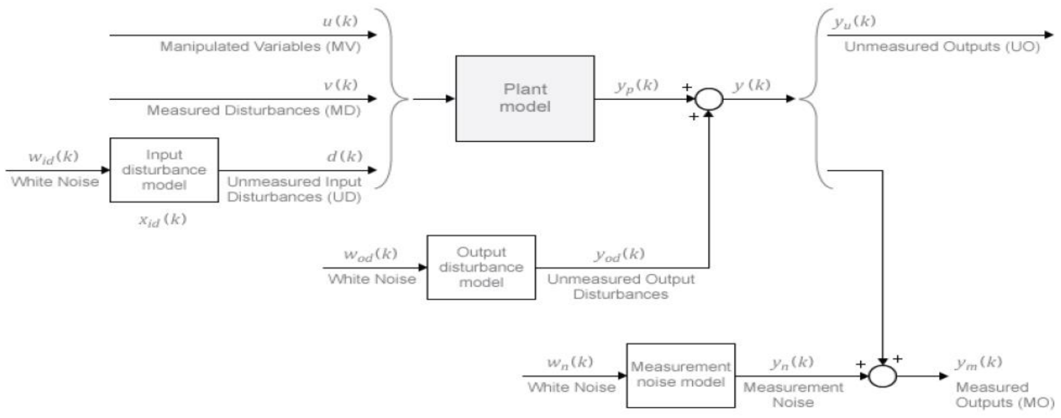


Figure 47 MPU block diagram

Plant model is the state space equation developed, we used MATLAB MPC block so, we load the LTI state space equation so, we don't need to linearize the system as it is already linearized, we specified the manipulated variable of the system which in our case is the steering angle δ and from the state space equation we took only the lateral position y and yaw angle ψ as our output from the system so, these output can be measured using our sensors so, they are measured output 'MO' and we don't include any disturbance in the system.

4.4.1 Optimization function

The optimization function used quadratic cost function:

$$J = \sum_{i=1}^{N} w_{xi}(r_i - x_i)^2 + \sum_{i=1}^{N} w_{ui}\Delta {u_i}^2$$

Were,

- x_i is the controlled variable
- r_i is the reference variable
- u_i is the manipulated variable
- w_{xi} is the weight coefficient
- $w_{ui}$is the weight coefficient that penalize the change in manipulated variable

4.4.2 Horizon period:

Is the amount of states the controller predicts in the future and put in the cost function to optimize the error?

System response to step reference with horizon of 10:

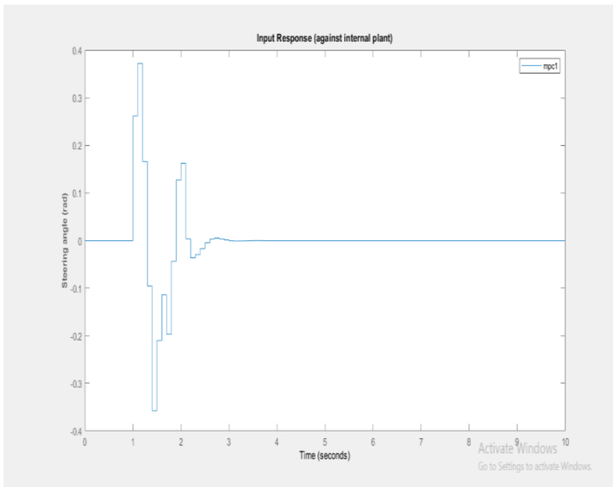


Figure 49 Horizon of 10 MPU Response Input

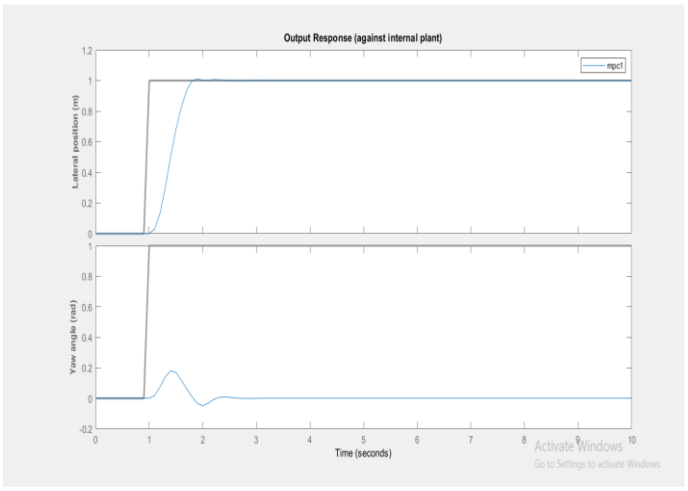


Figure 48 Horizon of 10 MPU Response Output

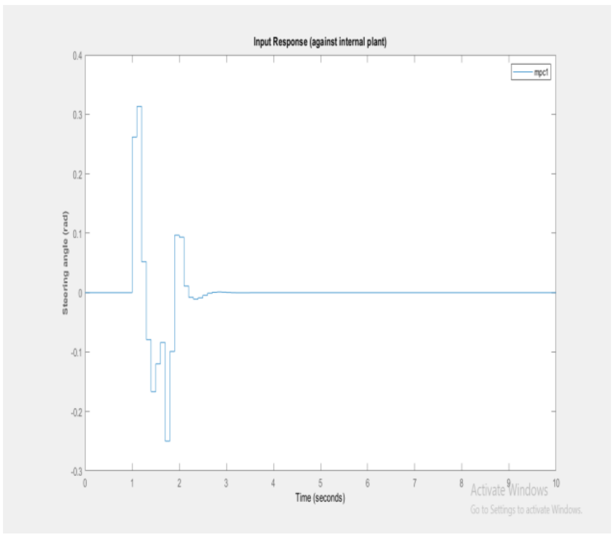


Figure 51 Horizon of 10 MPU Response Input

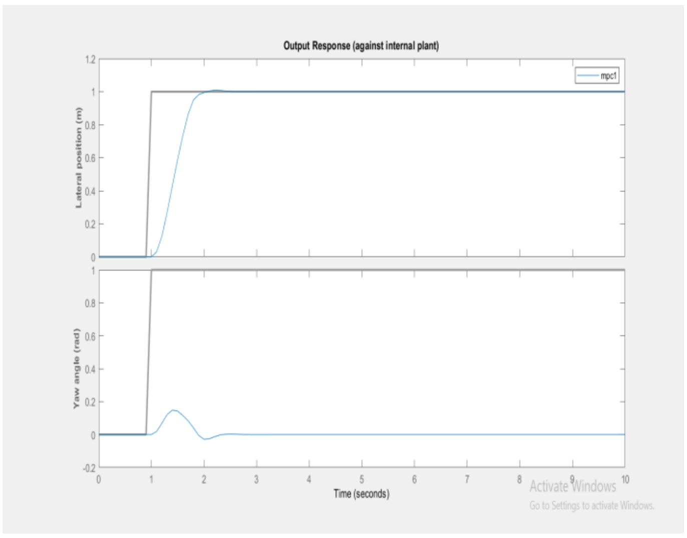


Figure 50 Horizon of 15 MPU Response Output

4.5 SIMULATION OF DIFFERENT PATH FOLLOWING TECHNIQUES

The simulation of the system was performed on Simulink as it contains ready block for each controller.

4.5.1 Pure Pursuit Simulink model

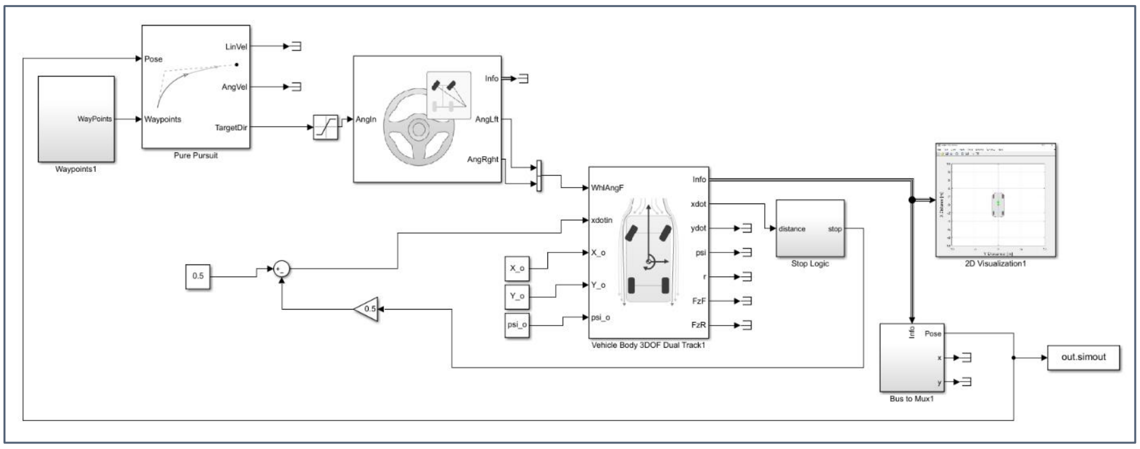


Figure 52 Pure Pursuit Simulink Model

4.5.2 Stanley Lateral Simulink model

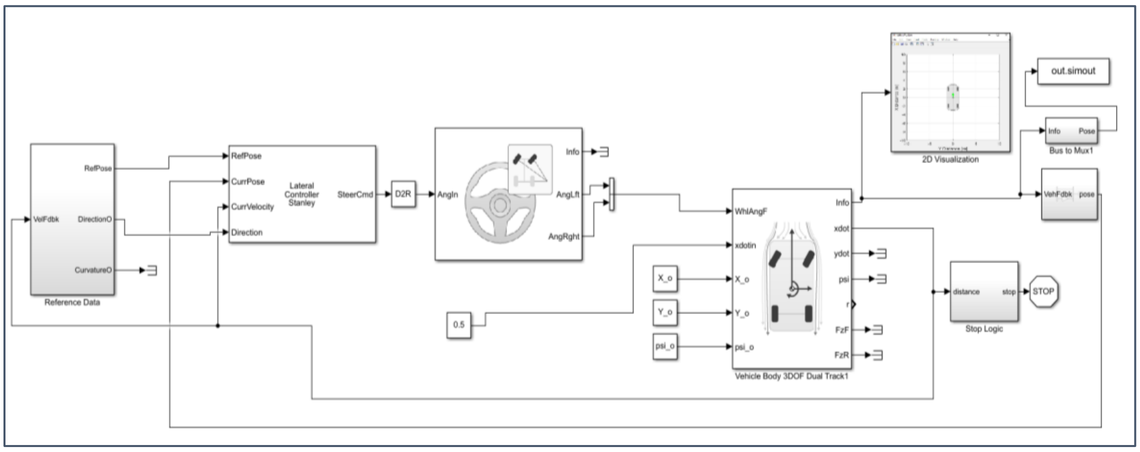


Figure 53 Stanley Controller Simulink Model

Some common blocks for the two controllers include:

1. **Vehicle Body 3DOF dual Track:** This block represents a 3 DOF rigid two-axle vehicle body model to simulate the position of the vehicle. It takes the desired velocity and the wheel angles [Right_Wheel, Left_Wheel] for the car. The output of the info contains some values used for the visualization.

2. **Akerman Steering**: This block calculates the steering angles of both the wheels.

3. **2D Visualization**: This block takes the information from the vehicle model to represent the data.

The two blocks for the Pure Pursuit and the Stanley Controller are described as follows:

4.5.3 Pure Pursuit

This block takes the waypoints of the path and the current position of the vehicle to calculate the steering angles required to follow the path.

4.5.4 Lateral Controller Stanley

It takes the waypoints and the current poison as well as the velocity of the vehicle to determine the steering angle.

4.6 SIMULATION OF PRE-SET PATH AND GENERATED PATH

Some paths were created to test the response of the system. This included a circle path and a lemniscate.

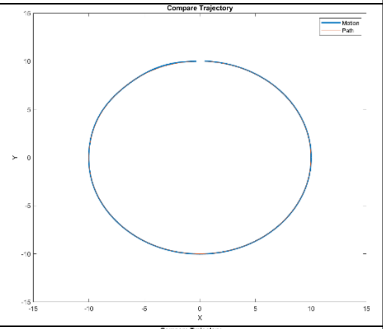
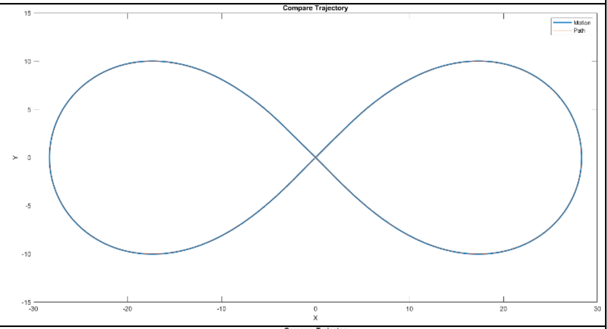
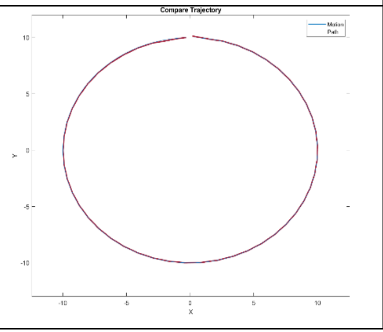
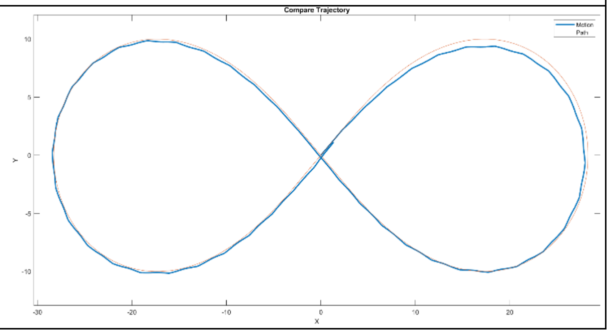
	Circle Path	Lemniscate
Pure Pursuit		
Stanley Lateral Controller		

Figure 54 Path Following of Circle and Lemniscate

Also, a path that was generated from the Hybrid A* algorithm was tested.

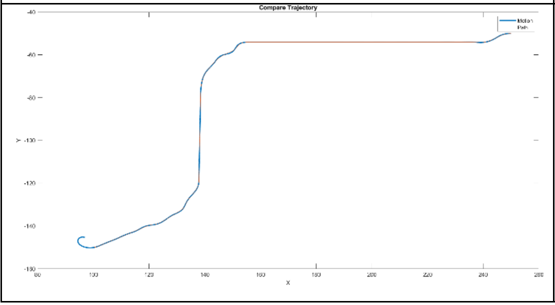
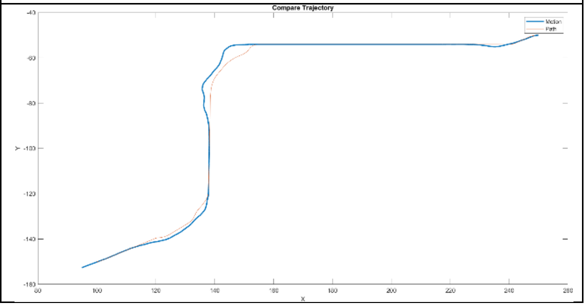
Pure Pursuit	Stanley
	

Figure 55 Path Following of Hybrid A* Generated Path

It appears that the pure pursuit had a better performance than the Stanley controller

The error of the system was tested:

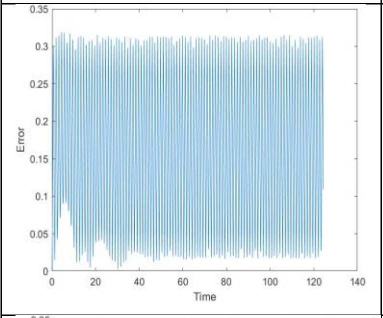
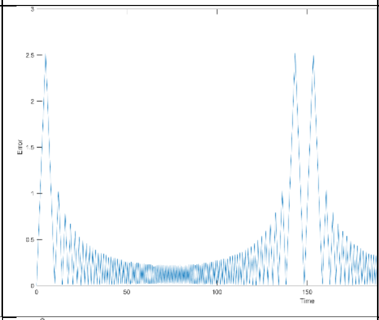
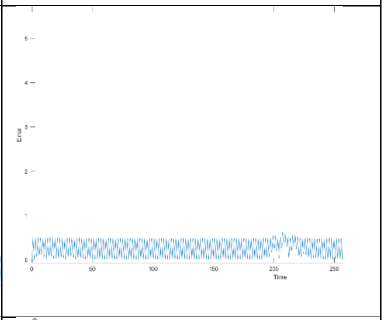
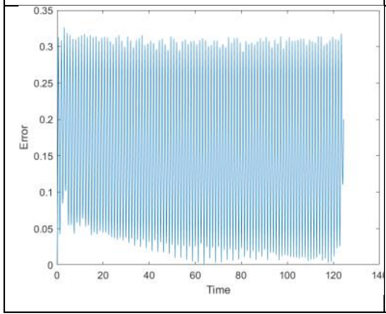
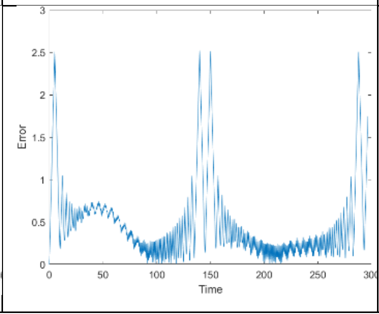
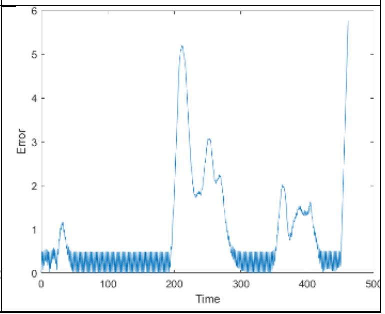
Circle Error	Lemniscate	Hybrid A* Path
		
		

Figure 56 Error curves of simulation

It appears that the pure pursuit had an overall better performance than the Stanley controller where in the Hybrid A* path the error once surpassed 5m.

4.7 MODEL PREDICTIVE CONTROLLER SIMULATION

Random path:

- Steering angle:

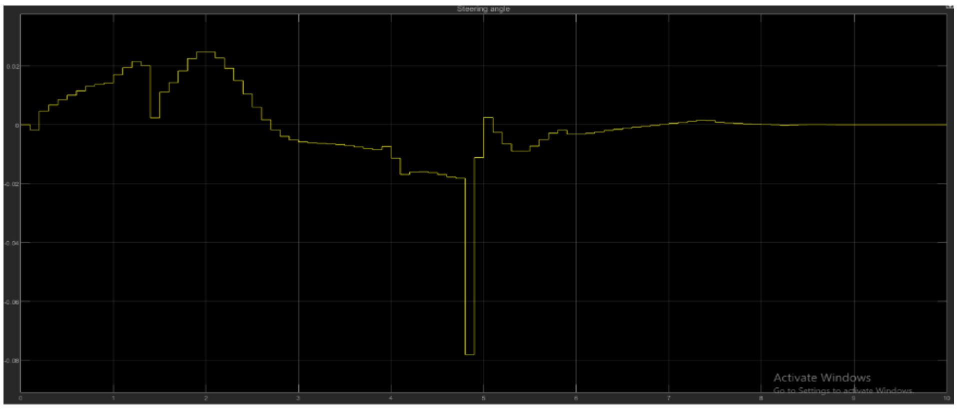


Figure 57 MPU Random Path Steering Angle

- Lateral position:

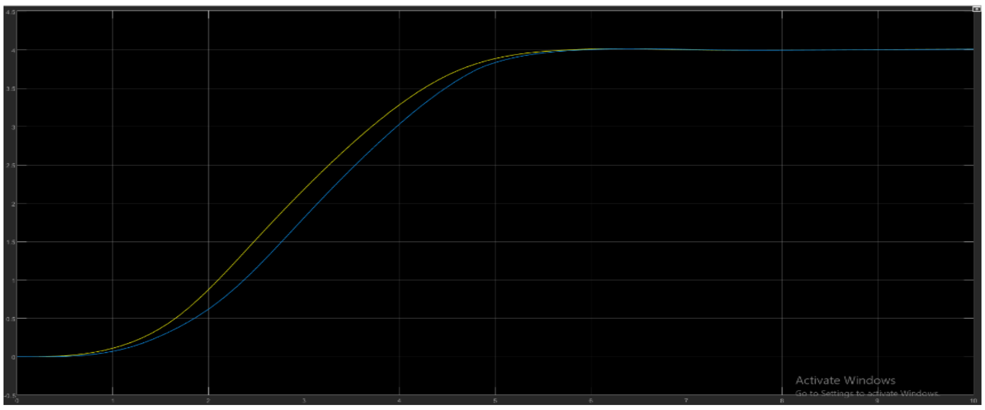


Figure 58 MPU Random Path Lateral Position

- Yaw position:

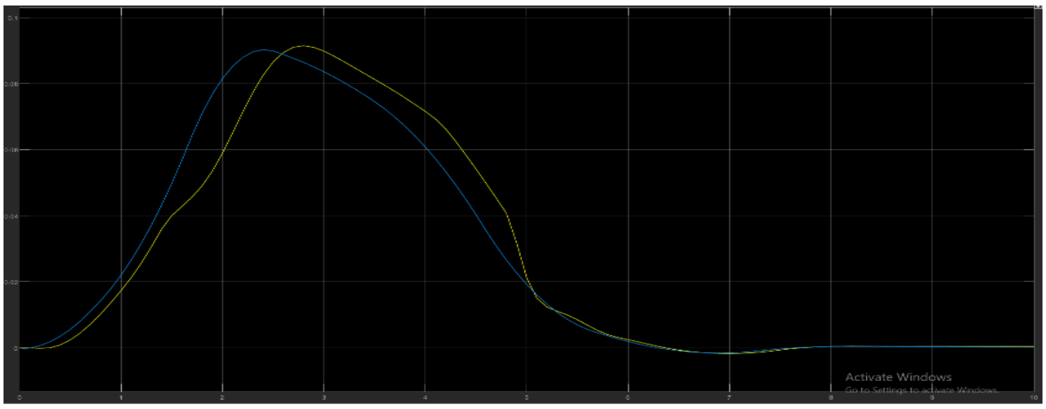


Figure 59 MPU Random Path Yaw Position

Circular path:

- Lateral position:

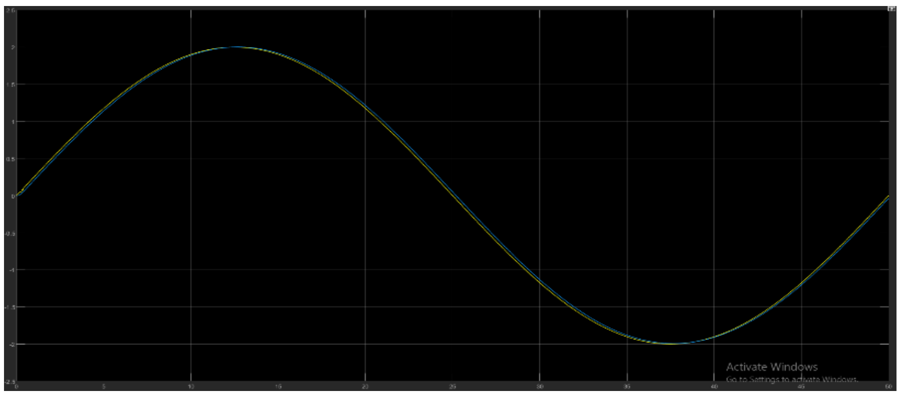


Figure 60 MPU Circular Path Lateral Position

- Yaw angle:

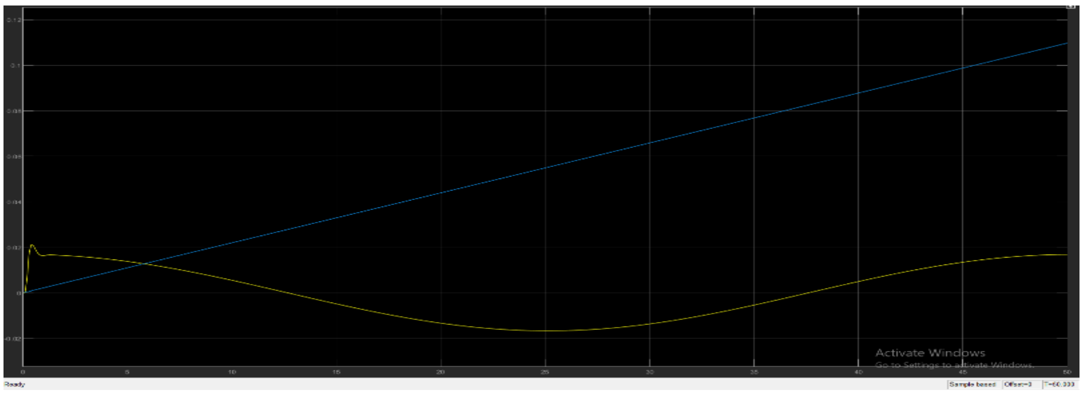


Figure 61 MPU Circular Path Yaw Angle

- Steering angle:

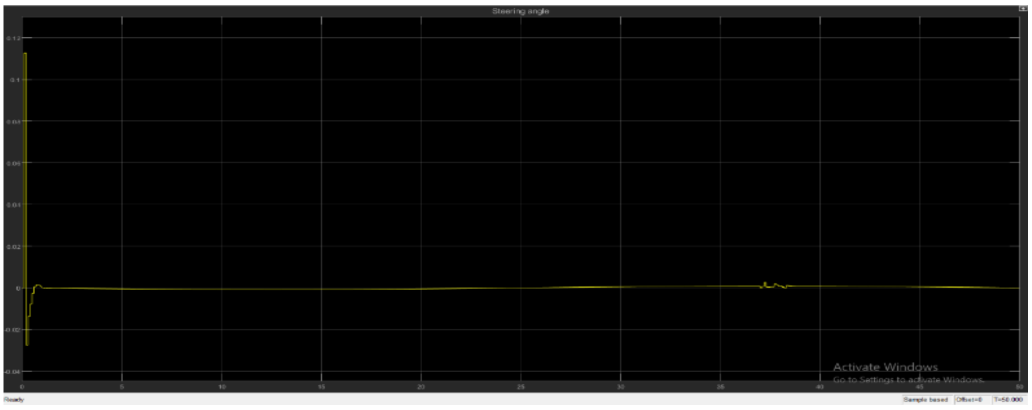


Figure 62 MPU Circular Path Steering angle

- Lateral position error at T =26 is -0.25 - (-0.204) = 0.046

- Yaw angle error at T = 25.5 is 0.047 - (-1.512) = 1.559

Hybrid A* path:

- Lateral position:

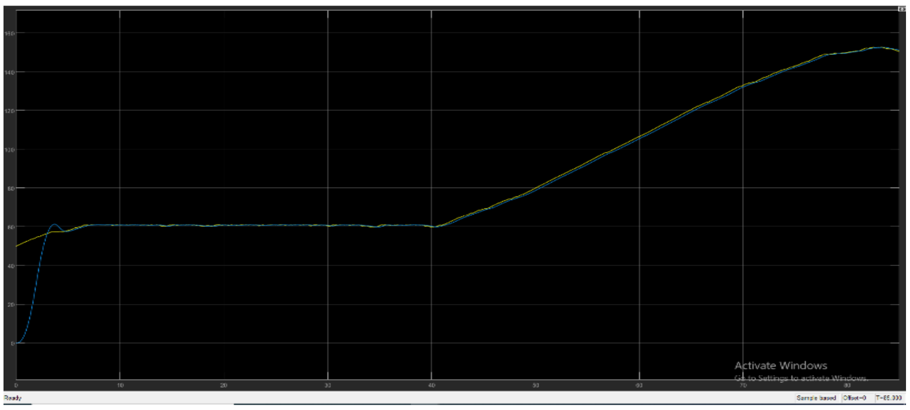


Figure 63 MPU Hybrid A* Path Lateral Position

- Yaw angle:

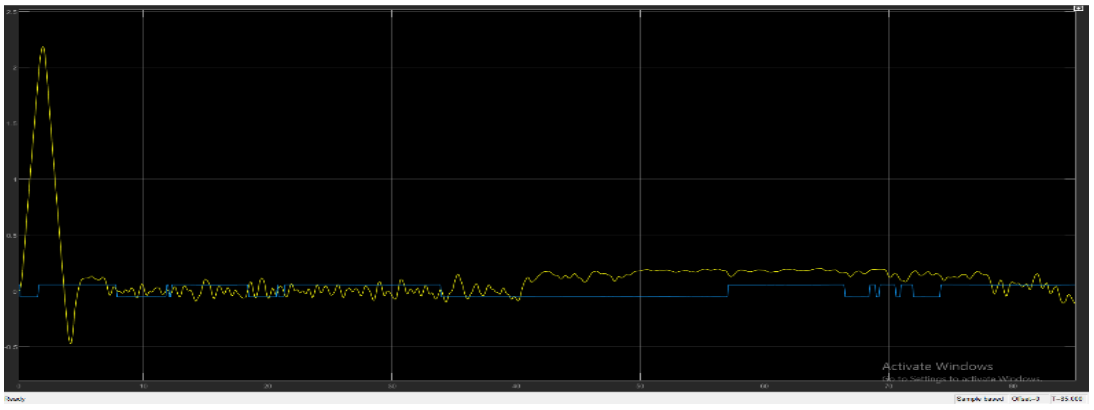


Figure 64 MPU Hybrid A* Path Yaw Position

- Steering angle:

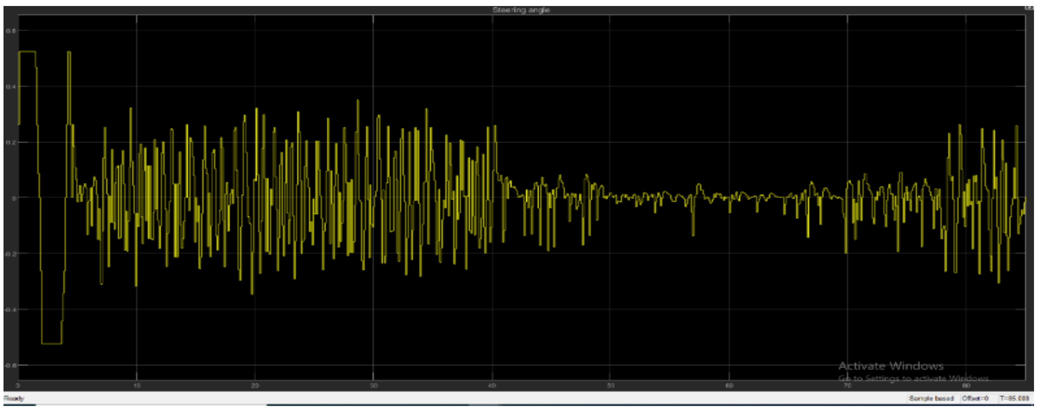


Figure 65 MPU Hybrid A* Path Steering Angle

- Lateral position error at t =52 is 117 – 115.6 = 1.4

- Yaw angle error at t =52 is -0.052-0.183= 0.235

4.8. HARDWARE IMPLEMENTATION

The hardware implementation of the models included hardware code generation on a Raspberry Pi with ROS. The following figure is of the system which is like the simulation only with a few added ROS blocks. The blocks contain are:

1. **Blank Message:** This creates a black message of a certain message type which will be set with the
2. **Bus Assignment:** This allows the modification of any element in the ROS message.
3. **Publish:** This creates a publisher for the node with a certain topic.

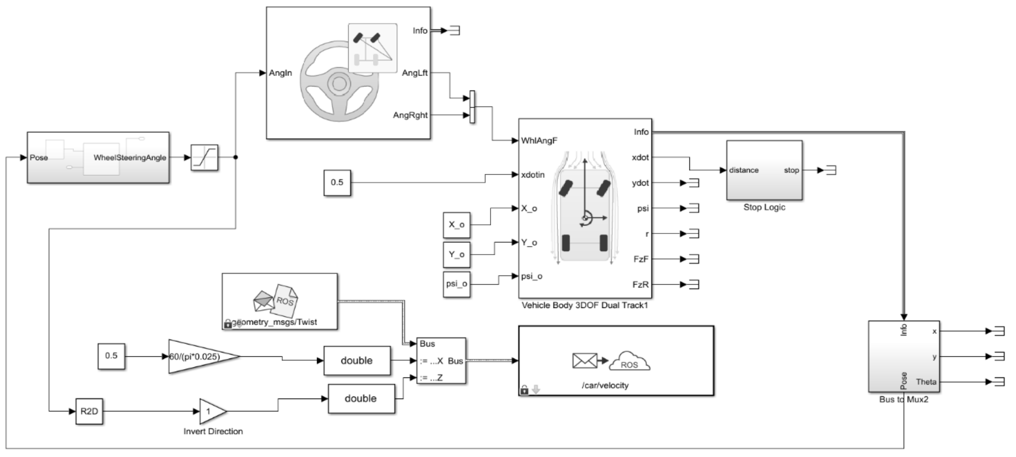


Figure 66 Simulink Hardware Model

The topic "/car/velocity" is published by the path following node and subscribed by a Ros serial Arduino node that modifies the steering angle of the car based on the topic values and sends the desired speed to the speed controller using I2c.

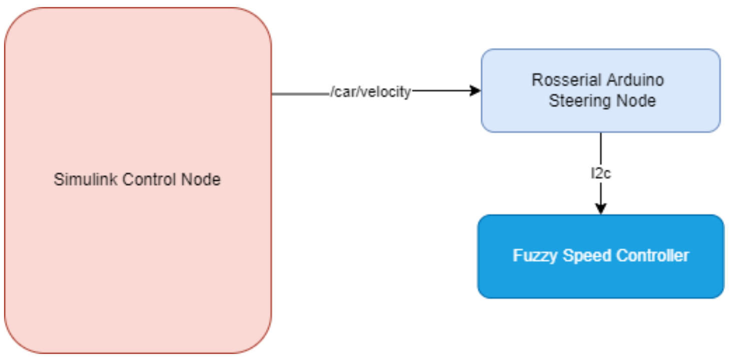


Figure 67 Simulink-ROS Low Level Flow

CHAPTER FIVE: RESULTS AND CONCLUSION

After all implementations were integrated, the electric vehicle's prototype was able to accomplish a load of its intended goals. One of which is to move from a point to point in a straight line. It also accomplished complicated path following techniques where the vehicle was able to move in a U-shape, S-shape, and in a full circle autonomously.

OUTPUT VIDEOS

Videos for the result of the vehicle movement can be found on Google Drive [Here] or on YOUTUBE where it follows three different paths and simultaneously the paths are plotted Realtime on MATLAB for visualization.

5.1 BILL OF MATERIALS

Product	Quantity	price	Total Price
Encoder	2	300	600
Raspberry pi 4	1	1300	1300
Motot Driver	2	130	260
Car Battery	1	700	700
Arduino Mega	1	250	250
Ultrasonic	3	100	300
IMU	1	180	180
Absolute Encoder	1	100	100
Connectors	2	20	40
electric vehicle	1	3000	3000
Electric Components	-	200	200
Wires	-	150	150
Raspberry pi battery	1	350	350
Fan	2	30	60
		Total	7490

Table 4 Bill of Materials

5.2 MANUFACTURING COST

Part	Manufacturing	Price
PCB	-	600
Gears	**3D Printing**	**100**
Encoder Mount\	3D Printing	100
ECU Box	**Laser Cutting**	**150**
Encoder Tensioner	3D Printing	500
	Total	1450

Table 5 Manufacturing Cost

APPENDIX

GLOBAL PATH PLANNING ALGORITHMS

Main.m

CONTENTS

HYBRIDASTAR IMPLEMENTATION

```
tic; %Computational Time
[HybridPath,smooth(1),minclearance(1)] = HybridAStar(); %Returns
path/smoothness/minimum clearance
len(1) = round(pathLength(HybridPath)); %Path Length (m)
t(1) = toc;
xlswrite('Hybridpath.xls',HybridPath.States)
```

RRT* IMPLEMENTATION

```
tic; %Computational Time
[RRTStarPath,smooth(2),minclearance(2)] = RTTstar(); %Returns
path/smoothness/minimum clearance
len(2) = round(pathLength(RRTStarPath)); %Path Length (m)
t(2)=toc;
xlswrite('RRTStarpath.xls',RRTStarPath.States)
```

RRT IMPLEMENTATION

```
tic; %Computational Time
[RRTPlannerPath,smooth(3),minclearance(3)] = RTTplanner();
len(3) = round(pathLength(RRTPlannerPath));
t(3)=toc;
xlswrite('RRTpath.xls',RRTStarPath.States)
```

BI-RRT IMPLEMENTATION

```
%The algorithm is implemented on mathworks online matlab and the generated
path is imported
[smooth(4),minclearance(4)]=BiRRT(); %Returns smoothness/minimum clearance
t(4) = 5.5; %Computational Time
len(4) = 266; % Path Length (m)
openfig('BiRRT.fig')
```

SPIDER PLOT

```
figure; %plotting computational time, length, 1/min clearance and smoothness
p= [t(1), len(1), round(1/minclearance(1)) ,smooth(1);t(2), len(2),...
    round(1/minclearance(2)) ,...
    smooth(2); t(3), len(3) ,round(1/minclearance(3)), smooth(3);...
    t(4), len(4),round(1/minclearance(4)),smooth(4)];

spider_plot(p, 'AxesLabels',{'Computational Time(s)'...
    ,'Path Length (m)', '1/Minimum Clearance (m-1)', 'Smoothness'}...
    ,'AxesInterval', 2,...
    'FillOption', {'on', 'on', 'on','on'},...
    'FillTransparency', [0.2, 0.1, 0.1,0.1]...
    ,'AxesFontSize', [16] )
legend('HybridAStar','RRTStar','RRT','BiRRT');
title('Path Planning Algorithms - Lower value in each axis indicates better
performace')
```

HybridAStar.m

HYBRID A* ALGORITHM

```
function [output,smooth,minclearance] = HybridAStar()
```

MAP PREPROCESSING

```
image = imread('map2.1.png');
grayimage = rgb2gray(image);
bwimage = grayimage < 0.5;
map = binaryOccupancyMap(bwimage);
```

PARAMS

```
stateSpace = stateSpaceSE2;

stateValidator = validatorOccupancyMap(stateSpace);

stateValidator.Map = binaryOccupancyMap(map);

stateValidator.ValidationDistance = 0.005;

planner = plannerHybridAStar(stateValidator
,'MinTurningRadius',3.8,'MotionPrimitiveLength',4,'NumMotionPrimitives',49)
```

PLANNER

```
%startPose = [250 50 pi];
%goalPose = [100 150 pi];
startPose = [60 60 pi];
goalPose = [250 50 pi];
refpath = plan(planner,startPose,goalPose);
pathMetricsObj=pathmetrics(refpath,stateValidator);
```

PLOT

```
figure; %minimum clearance visualized
show(pathMetricsObj)
title('HybridAStar Minimum Clearance')
legend('Planned Path','Minimum Clearance')

figure; %path visualized
show(planner);
hold on;
```

RETURN

```
output = refpath

smooth = smoothness(pathMetricsObj)

minclearance = clearance(pathMetricsObj);

end
```

RRT.m

RRT ALGORITHM

```
function [output,smooth1,minclearance1] = RTTplanner()
```

LOADING PREPROCESSED MAP

```
load('map.mat')
```

PARAMS

object stores parameters and states in the SE(2) state space, which is composed of state vectors represented by [x, y, θ].

```
stateSpace = stateSpaceSE2;

% object validates states and discretized motions based on the value in a 2-D
occupancy map.
stateValidator = validatorOccupancyMap(stateSpace);
stateValidator.Map = map;
stateSpace.StateBounds = [map.XWorldLimits; map.YWorldLimits; [-pi pi]];
stateValidator.ValidationDistance = 0.005;
```

PLANNER

```
startPose = [60 60 pi];
goalPose = [250 50 pi];
planner = plannerRRT( stateSpace,stateValidator);
planner.MaxConnectionDistance = 0.3;
refpath = plan(planner,startPose,goalPose);
pathMetricsObj1=pathmetrics(refpath,stateValidator);
rng(100,'twister'); % for repeatable result
[pthObj,solnInfo] = plan(planner,startPose,goalPose);
```

PLOT

```
figure; %path visualized
map.show;
title('RRT Planner Algorithm')
hold on
plot(solnInfo.TreeData(:,1),solnInfo.TreeData(:,2),'.-'); % tree expansion
plot(pthObj.States(:,1),pthObj.States(:,2),'r-','LineWidth',2) % draw
```

RETURN

```
output= refpath;

smooth1 = smoothness(pathMetricsObj1);

minclearance1 = clearance(pathMetricsObj1);

end
```

RRTStar.m

PLANNER

```
function [output,smooth,minclearance]=RTTstar()
```

LOADING PREPROCESSED MAP

```
load('map.mat')
```

PARAMS

object stores parameters and states in the SE(2) state space, which is composed of state vectors represented by [x, y, θ].

```
stateSpace = stateSpaceSE2;

% object validates states and discretized motions based on the value in a 2-D
occupancy map.
stateValidator = validatorOccupancyMap(stateSpace);
stateValidator.Map = map;
stateSpace.StateBounds = [map.XWorldLimits; map.YWorldLimits; [-pi pi]];
stateValidator.ValidationDistance = 0.005;
planner = plannerRRTStar( stateSpace,stateValidator);
planner.ContinueAfterGoalReached = true;
planner.MaxIterations = 25000;
planner.MaxConnectionDistance = 0.3;
```

PLANNER

```
startPose = [60 60 pi];

goalPose = [250 50 pi];

refpath = plan(planner,startPose,goalPose);
```

```
rng(100, 'twister') % repeatable result
[pthObj, solnInfo] = plan(planner,startPose,goalPose);
pathMetricsObj=pathmetrics(refpath,stateValidator)
```

PLOT

```
figure; %Minimum Clearance
show(pathMetricsObj)
title('RRTStar Planner Algorithm')
legend('Planned Path','Minimum Clearance')

figure; %path visualized
map.show;
title('RRTStar Planner Algorithm')
hold on;
%plot(solnInfo.TreeData(:,1),solnInfo.TreeData(:,2), '.-'); % tree expansion
plot(pthObj.States(:,1),pthObj.States(:,2),'r-','LineWidth',2); % draw path
```

RETURN

```
output = refpath;

smooth = smoothness(pathMetricsObj);

minclearance = clearance(pathMetricsObj);

end
```

BiRRT.m

BI-RRT

```
function [smooth,minclearance]=BiRRT()
```

MAP

```
load('birrtpath.mat')
```

RETURN

```
smooth = smoothness(pathMetricsObj)

minclearance = clearance(pathMetricsObj)
```

PLOT

```
figure; %path visualized
show(pathMetricsObj)
title('BiRRT Planner Algorithm')
%legend('Planned Path','Minimum Clearance')
```

Steering.ino

```
#define MIN(a,b) ((a) < (b) ? (a) : (b))
#define MAX(a,b) ((a) < (b) ? (b) : (a))
/////////////////Library Inclutions/////////////////////
#include <Wire.h>
#include <Servo.h>
Servo myservo;
#include <MPU6050.h>
#include "math.h"
////////////////////////////////////////////////////

//////////////////ROS Setup////////////////////////
#include <ros.h>
#include <geometry_msgs/Twist.h>
ros::NodeHandle nh;
geometry_msgs::Twist msg;

void roverCallBack(const geometry_msgs::Twist& cmd_vel) //
Callback when node recives data from topic /car/velocity
{
  velocity = cmd_vel.linear.x;
  fin_angle = cmd_vel.angular.z;
}

ros::Subscriber <geometry_msgs::Twist> sub("/car/velocity",
roverCallBack);//Setup a subscriber to the topic /car/velocity

           //with callback roverCallBack anf message type Twist
////////////////////////////////////////////////////

/////////////////IMU Setup//////////////////
MPU6050 mpu;
unsigned long timer = 0;
float deltaT = 0;
float yaw = 0;
float h = 0;
float r_wheel = 0.25; // in meters
double X = 0;
double Y = 0;
float RPM = 0;
float oldRPM=0;
/////////////////////////////////////////////////


///////////////Steering Setup///////////////////
int angle = 0;
int pos = 0;
int dir = 0;
int fin_angle =0;
int velocity = 0;
int prev_angle = 0 ;
/////////////////////////////////////////////////
```

```
void setup() {

  mpu.calibrateGyro();// Calibrate gyroscope. The calibration
must be at rest.
  mpu.setThreshold(3);// Set threshold sensivty. Default 3.
  nh.initNode();//Initalize the ROS node
  nh.subscribe(sub);//Initalize Subscribe
  myservo.attach(11); //Connect Servo
  Homing();//Return wheel to steering angle 0
  Wire.begin();Start I2c
}
void get_pose()//Function to get the position from the IMU
{
  Vector norm = mpu.readNormalizeGyro();
  deltaT_step = millis()-timer;
  yaw = yaw + norm.ZAxis * deltaT_step;
  deltaRPM = oldRPM - RPM;
  h = deltaRPM*2*M_PI*r_wheel;
  X += h*cos(yaw);
  Y += h*sin(yaw);
  timer = millis()
}

void Homing()//function that return the steering angle to zero
position
{
    if (dir <= 30)
    {
      for (pos = dir; pos <= 30; pos += 1)
      {
        myservo.write(pos);
        delay(15);
      }
    }

    else if (dir >= 31)
    {
    for (pos = dir; pos >= 30; pos -= 1)
    {
    myservo.write(pos);
    delay(15);
    }
  }
     dir = pos ;
}

int Turn_Left(int angle)
{
      myservo.write(angle);

}

int Turn_Right (int angle)
{
  myservo.write(angle);
}
void steer()//Function that steers the angle
{

  if (fin_angle>=30)
  {
    fin_angle =30;
  }
  else if (fin_angle<=-30)
  {
    fin_angle = -30;
  }
if(prev_angle != fin_angle)
{
    if (fin_angle >= 1)
    {
      angle = int(30 - fin_angle);
      Turn_Right(MIN(angle,30));
    }

    else if (fin_angle <= -1)
    {
      angle = int(30 - fin_angle);
      Turn_Left(MAX(angle,30));
    }

    else if (fin_angle == 0)
    {
     angle = int(fin_angle);
     Homing();
    }
    prev_angle = fin_angle;
}
}

void loop()
{

  nh.spinOnce();//Recive the Velocity and Steering from the ros
topic
  steer(); //Modify the Steering Angle
  Wire.beginTransmission(8);//Start Transmitting the Velocity to
the Velocity controller
  Wire.write(velocity);
  Wire.endTransmission();
  delay(100);

}
```

Motor.h

```
#include "Arduino.h"


#define  in1_mo1  8
#define  in2_mo1  7
#define  in1_mo2  9
#define  in2_mo2  10
#define  EN1  6
#define  EN2  5

//////////////////////// Initialization ///////////////////
double error =0.0 ;
double Motor_pwmOutput = 0.0 ;
double Motor_RPM = 0.0;
double Motor_RPM_output =0;
double Setpoint_rpm = 2000;
double error_precenrage = 0.0;
volatile unsigned long counter = 360;        // number of pulses
const float EncoderPPR = 360.0 /10;          // number of pulses per
rev
volatile unsigned int counter1, coun = 0;    //This variable will
increase or decrease depending on the rotation of encoder
double last_error = 0.0 ;
double Change_of_Error =0.0;
////////////////////////////////////////////////////////////////

///////////////// Encoder ////////////////
void ENCODER_B()
{
  if(digitalRead(3)==HIGH)
  {
  coun++;
  }
}

int Reset_counter(int counter)
{
  if (coun == counter)
  {
    coun=0;
  }
}
/////////////////////////////////////////


int CALCULATE ()
{
   if(Setpoint_rpm==0)
      {
      Motor_pwmOutput=0;
      }
   else
     {
     Motor_RPM = (counter1/EncoderPPR)*60.0;
     Motor_RPM_output = Motor_RPM * 0.03379 ;
     error = Setpoint_rpm - Motor_RPM ;
     error_precenrage = error / Setpoint_rpm ;
     Change_of_Error = last_error - error_precenrage ;
     last_error =  error_precenrage ;
     }
}


//////////////// Motors /////////////////
void Motor_right ()
{
          analogWrite (EN1 , Motor_pwmOutput );
          digitalWrite(in1_mo1,LOW);
          digitalWrite(in2_mo1,HIGH);
}

void Motor_left()
{
          analogWrite (EN2 , Motor_pwmOutput );
          digitalWrite(in1_mo2,HIGH);
          digitalWrite(in2_mo2,LOW);
}

////////////////////////////////////////////
```

Fis_header.h

```
#define FIS_TYPE float
#define FIS_RESOLUSION 101
#define FIS_MIN -3.4028235E+38
#define FIS_MAX 3.4028235E+38
typedef FIS_TYPE(*_FIS_MF)(FIS_TYPE, FIS_TYPE*);
typedef FIS_TYPE(*_FIS_ARR_OP)(FIS_TYPE, FIS_TYPE);
typedef FIS_TYPE(*_FIS_ARR)(FIS_TYPE*, int, _FIS_ARR_OP);
```

Fuzzy.ino

```
#include <TimerOne.h>
#include "fis_header.h"
#include "motor.h"


const int fis_gcI = 2;              // Number of inputs to the fuzzy
inference system
const int fis_gcO = 1;              // Number of outputs to the fuzzy
inference system
const int fis_gcR = 49;             // Number of rules to the fuzzy
inference system


FIS_TYPE g_fisInput[fis_gcI];
FIS_TYPE g_fisOutput[fis_gcO];

void setup()
{
  Serial.begin(115200);
  pinMode(in1_mo1, OUTPUT);
  pinMode(in2_mo1, OUTPUT);
  pinMode(in1_mo2, OUTPUT);
  pinMode(in2_mo2, OUTPUT);
  pinMode(2, INPUT_PULLUP); // internal pullup input pin 2
  pinMode(3, INPUT_PULLUP);                // internal pullup input
pin 3
  pinMode(0 , INPUT);                      // Pin mode for Input:
Error
```

```
  pinMode(1 , INPUT);                         // Pin mode for Input:
Change_of_Error
  pinMode(4 , OUTPUT); // Pin mode for Output: output1
  pinMode(9 , OUTPUT);
  attachInterrupt(1, ENCODER_B, RISING);
  Timer1.initialize(100000);
  Timer1.attachInterrupt(Timer_Handler);
}


// Loop routine runs over and over again forever:
void loop()
{

    g_fisInput[0] = error_precenrage ;                // Read Input:
Error
    g_fisInput[1] = Change_of_Error ;                 // Read Input:
Change_of_Error
    g_fisOutput[0] = 0;
    fis_evaluate();


    // Set output vlaue: output1
    analogWrite(2 , g_fisOutput[0]);
    Motor_pwmOutput = g_fisOutput[0] *255;


}


////////////////// Timer Interrupt ///////////////
void Timer_Handler()
{
  Timer1.detachInterrupt();

  if( coun != counter1 )
  {
  counter1 = coun;
  CALCULATE();
  Motor_right();
  Motor_left();
  Reset_counter(coun);
  }
  Timer1.attachInterrupt(Timer_Handler);
}
///////////////////////////////////////////////////


///////////////////////// fuzzy //////////////////
//////////// code generation from matlab /////////


//**********************************************************************
***
// Support functions for Fuzzy Inference
System
//**********************************************************************
***
// Triangular Member Function
FIS_TYPE fis_trimf(FIS_TYPE x, FIS_TYPE* p)
{
    FIS_TYPE a = p[0], b = p[1], c = p[2];
    FIS_TYPE t1 = (x - a) / (b - a);
    FIS_TYPE t2 = (c - x) / (c - b);
    if ((a == b) && (b == c)) return (FIS_TYPE) (x == a);
    if (a == b) return (FIS_TYPE) (t2*(b <= x)*(x <= c));
    if (b == c) return (FIS_TYPE) (t1*(a <= x)*(x <= b));
    t1 = min(t1, t2);
    return (FIS_TYPE) max(t1, 0);
}

FIS_TYPE fis_min(FIS_TYPE a, FIS_TYPE b)
{
    return min(a, b);
}

FIS_TYPE fis_max(FIS_TYPE a, FIS_TYPE b)
{
    return max(a, b);
}

FIS_TYPE fis_array_operation(FIS_TYPE *array, int size, _FIS_ARR_OP
pfnOp)
{
    int i;
    FIS_TYPE ret = 0;

    if (size == 0) return ret;
    if (size == 1) return array[0];

    ret = array[0];
    for (i = 1; i < size; i++)
    {
        ret = (*pfnOp)(ret, array[i]);
    }

    return ret;
}


//*********************************************************************
***
// Data for Fuzzy Inference
System
//*********************************************************************
***
// Pointers to the implementations of member functions
_FIS_MF fis_gMF[] =
{
    fis_trimf
};

// Count of member function for each Input
int fis_gIMFCount[] = { 7, 7 };

// Count of member function for each Output
int fis_gOMFCount[] = { 7 };

// Coefficients for the Input Member Functions
FIS_TYPE fis_gMFI0Coeff1[] = { -1, -1, -0.6667 };
FIS_TYPE fis_gMFI0Coeff2[] = { -1, -0.6625, -0.3291 };
FIS_TYPE fis_gMFI0Coeff3[] = { -0.6667, -0.3333, 0 };
FIS_TYPE fis_gMFI0Coeff4[] = { -0.3333, 0, 0.3333 };
FIS_TYPE fis_gMFI0Coeff5[] = { 0, 0.3333, 0.6667 };
FIS_TYPE fis_gMFI0Coeff6[] = { 0.3333, 0.6667, 1 };
FIS_TYPE fis_gMFI0Coeff7[] = { 0.658, 0.996, 1 };
FIS_TYPE* fis_gMFI0Coeff[] = { fis_gMFI0Coeff1, fis_gMFI0Coeff2,
fis_gMFI0Coeff3, fis_gMFI0Coeff4, fis_gMFI0Coeff5, fis_gMFI0Coeff6,
fis_gMFI0Coeff7 };
FIS_TYPE fis_gMFI1Coeff1[] = { -1, -1, -0.6667 };
FIS_TYPE fis_gMFI1Coeff2[] = { -1, -0.6667, -0.3333 };
FIS_TYPE fis_gMFI1Coeff3[] = { -0.6667, -0.3333, 0 };
FIS_TYPE fis_gMFI1Coeff4[] = { -0.3333, 0, 0.3333 };
FIS_TYPE fis_gMFI1Coeff5[] = { 0, 0.3333, 0.6667 };
FIS_TYPE fis_gMFI1Coeff6[] = { 0.337528329809725, 0.670928329809725,
1.00422832980973 };
FIS_TYPE fis_gMFI1Coeff7[] = { 0.6667, 1, 1.333 };
FIS_TYPE* fis_gMFI1Coeff[] = { fis_gMFI1Coeff1, fis_gMFI1Coeff2,
fis_gMFI1Coeff3, fis_gMFI1Coeff4, fis_gMFI1Coeff5, fis_gMFI1Coeff6,
fis_gMFI1Coeff7 };
FIS_TYPE** fis_gMFICoeff[] = { fis_gMFI0Coeff, fis_gMFI1Coeff };

// Coefficients for the Output Member Functions
FIS_TYPE fis_gMFO0Coeff1[] = { 0, 0, 0.1688 };
FIS_TYPE fis_gMFO0Coeff2[] = { 0, 0.1667, 0.3333 };
FIS_TYPE fis_gMFO0Coeff3[] = { 0.1667, 0.3333, 0.5 };
FIS_TYPE fis_gMFO0Coeff4[] = { 0.3333, 0.5, 0.6667 };
FIS_TYPE fis_gMFO0Coeff5[] = { 0.5, 0.6667, 0.8333 };
FIS_TYPE fis_gMFO0Coeff6[] = { 0.6646, 0.8312, 1 };
FIS_TYPE fis_gMFO0Coeff7[] = { 0.829071670190275, 0.995771670190275,
0.995771670190275 };
FIS_TYPE* fis_gMFO0Coeff[] = { fis_gMFO0Coeff1, fis_gMFO0Coeff2,
fis_gMFO0Coeff3, fis_gMFO0Coeff4, fis_gMFO0Coeff5, fis_gMFO0Coeff6,
fis_gMFO0Coeff7 };
FIS_TYPE** fis_gMFOCoeff[] = { fis_gMFO0Coeff };

// Input membership function set
int fis_gMFI0[] = { 0, 0, 0, 0, 0, 0, 0 };
int fis_gMFI1[] = { 0, 0, 0, 0, 0, 0, 0 };
int* fis_gMFI[] = { fis_gMFI0, fis_gMFI1};

// Output membership function set
int fis_gMFO0[] = { 0, 0, 0, 0, 0, 0, 0 };
int* fis_gMFO[] = { fis_gMFO0};

// Rule Weights
FIS_TYPE fis_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Type
int fis_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Inputs
int fis_gRI0[] = { 1, 1 };
int fis_gRI1[] = { 1, 2 };
int fis_gRI2[] = { 1, 3 };
int fis_gRI3[] = { 1, 4 };
int fis_gRI4[] = { 1, 5 };
int fis_gRI5[] = { 1, 6 };
int fis_gRI6[] = { 1, 7 };
int fis_gRI7[] = { 2, 1 };
int fis_gRI8[] = { 2, 2 };
int fis_gRI9[] = { 2, 3 };
int fis_gRI10[] = { 2, 4 };
int fis_gRI11[] = { 2, 5 };
int fis_gRI12[] = { 2, 6 };
int fis_gRI13[] = { 2, 7 };
int fis_gRI14[] = { 3, 1 };
int fis_gRI15[] = { 3, 2 };
int fis_gRI16[] = { 3, 3 };
int fis_gRI17[] = { 3, 4 };
int fis_gRI18[] = { 3, 5 };
int fis_gRI19[] = { 3, 6 };
int fis_gRI20[] = { 3, 7 };
int fis_gRI21[] = { 4, 1 };
int fis_gRI22[] = { 4, 2 };
int fis_gRI23[] = { 4, 3 };
int fis_gRI24[] = { 4, 4 };
int fis_gRI25[] = { 4, 5 };
int fis_gRI26[] = { 4, 6 };
int fis_gRI27[] = { 4, 7 };
int fis_gRI28[] = { 5, 1 };
int fis_gRI29[] = { 5, 2 };
int fis_gRI30[] = { 5, 3 };
int fis_gRI31[] = { 5, 4 };
int fis_gRI32[] = { 5, 5 };
int fis_gRI33[] = { 5, 6 };
int fis_gRI34[] = { 5, 7 };
int fis_gRI35[] = { 6, 1 };
int fis_gRI36[] = { 6, 2 };
int fis_gRI37[] = { 6, 3 };
int fis_gRI38[] = { 6, 4 };
int fis_gRI39[] = { 6, 5 };
int fis_gRI40[] = { 6, 6 };
int fis_gRI41[] = { 6, 7 };
int fis_gRI42[] = { 7, 1 };
int fis_gRI43[] = { 7, 2 };
int fis_gRI44[] = { 7, 3 };
int fis_gRI45[] = { 7, 4 };
int fis_gRI46[] = { 7, 5 };
int fis_gRI47[] = { 7, 6 };
int fis_gRI48[] = { 7, 7 };
int* fis_gRI[] = { fis_gRI0, fis_gRI1, fis_gRI2, fis_gRI3, fis_gRI4,
fis_gRI5, fis_gRI6, fis_gRI7, fis_gRI8, fis_gRI9, fis_gRI10,
fis_gRI11, fis_gRI12, fis_gRI13, fis_gRI14, fis_gRI15, fis_gRI16,
fis_gRI17, fis_gRI18, fis_gRI19, fis_gRI20, fis_gRI21, fis_gRI22,
fis_gRI23, fis_gRI24, fis_gRI25, fis_gRI26, fis_gRI27, fis_gRI28,
fis_gRI29, fis_gRI30, fis_gRI31, fis_gRI32, fis_gRI33, fis_gRI34,
fis_gRI35, fis_gRI36, fis_gRI37, fis_gRI38, fis_gRI39, fis_gRI40,
fis_gRI41, fis_gRI42, fis_gRI43, fis_gRI44, fis_gRI45, fis_gRI46,
fis_gRI47, fis_gRI48 };

// Rule Outputs
int fis_gRO0[] = { 1 };
int fis_gRO1[] = { 1 };
int fis_gRO2[] = { 1 };
int fis_gRO3[] = { 1 };
int fis_gRO4[] = { 2 };
int fis_gRO5[] = { 3 };
int fis_gRO6[] = { 4 };
int fis_gRO7[] = { 1 };
int fis_gRO8[] = { 1 };
int fis_gRO9[] = { 1 };
int fis_gRO10[] = { 2 };
int fis_gRO11[] = { 2 };
int fis_gRO12[] = { 3 };
int fis_gRO13[] = { 4 };
int fis_gRO14[] = { 1 };
int fis_gRO15[] = { 1 };
int fis_gRO16[] = { 2 };
int fis_gRO17[] = { 2 };
int fis_gRO18[] = { 4 };
int fis_gRO19[] = { 5 };
int fis_gRO20[] = { 6 };
int fis_gRO21[] = { 1 };
int fis_gRO22[] = { 2 };
int fis_gRO23[] = { 2 };
int fis_gRO24[] = { 4 };
int fis_gRO25[] = { 6 };
int fis_gRO26[] = { 6 };
int fis_gRO27[] = { 7 };
int fis_gRO28[] = { 2 };
int fis_gRO29[] = { 3 };
int fis_gRO30[] = { 4 };
int fis_gRO31[] = { 5 };
int fis_gRO32[] = { 6 };
int fis_gRO33[] = { 7 };
int fis_gRO34[] = { 7 };
int fis_gRO35[] = { 3 };
int fis_gRO36[] = { 5 };
int fis_gRO37[] = { 5 };
int fis_gRO38[] = { 6 };
int fis_gRO39[] = { 7 };
int fis_gRO40[] = { 7 };
int fis_gRO41[] = { 7 };
int fis_gRO42[] = { 4 };
int fis_gRO43[] = { 5 };
int fis_gRO44[] = { 6 };
int fis_gRO45[] = { 7 };
int fis_gRO46[] = { 7 };
int fis_gRO47[] = { 7 };
int fis_gRO48[] = { 7 };
int* fis_gRO[] = { fis_gRO0, fis_gRO1, fis_gRO2, fis_gRO3, fis_gRO4,
fis_gRO5, fis_gRO6, fis_gRO7, fis_gRO8, fis_gRO9, fis_gRO10,
fis_gRO11, fis_gRO12, fis_gRO13, fis_gRO14, fis_gRO15, fis_gRO16,
fis_gRO17, fis_gRO18, fis_gRO19, fis_gRO20, fis_gRO21, fis_gRO22,
fis_gRO23, fis_gRO24, fis_gRO25, fis_gRO26, fis_gRO27, fis_gRO28,
fis_gRO29, fis_gRO30, fis_gRO31, fis_gRO32, fis_gRO33, fis_gRO34,
fis_gRO35, fis_gRO36, fis_gRO37, fis_gRO38, fis_gRO39, fis_gRO40,
fis_gRO41, fis_gRO42, fis_gRO43, fis_gRO44, fis_gRO45, fis_gRO46,
fis_gRO47, fis_gRO48 };

// Input range Min
FIS_TYPE fis_gIMin[] = { -1, -1 };

// Input range Max
FIS_TYPE fis_gIMax[] = { 1, 1 };

// Output range Min
FIS_TYPE fis_gOMin[] = { 0 };

// Output range Max
```

```
FIS_TYPE fis_gOMax[] = { 1 };

//***********************************************************************
***
// Data dependent support functions for Fuzzy Inference
System
//***********************************************************************
***
FIS_TYPE fis_MF_out(FIS_TYPE** fuzzyRuleSet, FIS_TYPE x, int o)
{
    FIS_TYPE mfOut;
    int r;

    for (r = 0; r < fis_gcR; ++r)
    {
        int index = fis_gRO[r][o];
        if (index > 0)
        {
            index = index - 1;
            mfOut = (fis_gMF[fis_gMFO[o][index]])(x,
fis_gMFOCoeff[o][index]);
        }
        else if (index < 0)
        {
            index = -index - 1;
            mfOut = 1 - (fis_gMF[fis_gMFO[o][index]])(x,
fis_gMFOCoeff[o][index]);
        }
        else
        {
            mfOut = 0;
        }

        fuzzyRuleSet[0][r] = fis_min(mfOut, fuzzyRuleSet[1][r]);
    }
    return fis_array_operation(fuzzyRuleSet[0], fis_gcR, fis_max);
}

FIS_TYPE fis_defuzz_centroid(FIS_TYPE** fuzzyRuleSet, int o)
{
    FIS_TYPE step = (fis_gOMax[o] - fis_gOMin[o]) / (FIS_RESOLUSION -
1);
    FIS_TYPE area = 0;
    FIS_TYPE momentum = 0;
    FIS_TYPE dist, slice;
    int i;

    // calculate the area under the curve formed by the MF outputs
    for (i = 0; i < FIS_RESOLUSION; ++i){
        dist = fis_gOMin[o] + (step * i);
        slice = step * fis_MF_out(fuzzyRuleSet, dist, o);
        area += slice;
        momentum += slice*dist;
    }

    return ((area == 0) ? ((fis_gOMax[o] + fis_gOMin[o]) / 2) :
(momentum / area));
}

//***********************************************************************
***
```

```
// Fuzzy Inference
System
//*********************************************************************
***
void fis_evaluate()
{
    FIS_TYPE fuzzyInput0[] = { 0, 0, 0, 0, 0, 0, 0 };
    FIS_TYPE fuzzyInput1[] = { 0, 0, 0, 0, 0, 0, 0 };
    FIS_TYPE* fuzzyInput[fis_gcI] = { fuzzyInput0, fuzzyInput1, };
    FIS_TYPE fuzzyOutput0[] = { 0, 0, 0, 0, 0, 0, 0 };
    FIS_TYPE* fuzzyOutput[fis_gcO] = { fuzzyOutput0, };
    FIS_TYPE fuzzyRules[fis_gcR] = { 0 };
    FIS_TYPE fuzzyFires[fis_gcR] = { 0 };
    FIS_TYPE* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };
    FIS_TYPE sW = 0;

    // Transforming input to fuzzy Input
    int i, j, r, o;
    for (i = 0; i < fis_gcI; ++i)
    {
        for (j = 0; j < fis_gIMFCount[i]; ++j)
        {
            fuzzyInput[i][j] =
                (fis_gMF[fis_gMFI[i][j]])(g_fisInput[i],
fis_gMFICoeff[i][j]);
        }
    }

    int index = 0;
    for (r = 0; r < fis_gcR; ++r)
    {
        if (fis_gRType[r] == 1)
        {
            fuzzyFires[r] = FIS_MAX;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    fuzzyFires[r] = fis_min(fuzzyFires[r],
fuzzyInput[i][index - 1]);
                else if (index < 0)
                    fuzzyFires[r] = fis_min(fuzzyFires[r], 1 -
fuzzyInput[i][-index - 1]);
                else
                    fuzzyFires[r] = fis_min(fuzzyFires[r], 1);
            }
        }
        else
        {
            fuzzyFires[r] = FIS_MIN;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    fuzzyFires[r] = fis_max(fuzzyFires[r],
fuzzyInput[i][index - 1]);
                else if (index < 0)
                    fuzzyFires[r] = fis_max(fuzzyFires[r], 1 -
fuzzyInput[i][-index - 1]);
                else
                    fuzzyFires[r] = fis_max(fuzzyFires[r], 0);
            }
        }

        fuzzyFires[r] = fis_gRWeight[r] * fuzzyFires[r];
        sW += fuzzyFires[r];
    }

    if (sW == 0)
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = ((fis_gOMax[o] + fis_gOMin[o]) / 2);
        }
    }
    else
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = fis_defuzz_centroid(fuzzyRuleSet, o);
        }
    }
}
```

REFERENCES

[A1] Ding, Y. (2020). Simple Understanding of Kinematic Bicycle Model. [Blog] *Shuffleai*. Available at: https://www.shuffleai.blog/blog/Simple_Understanding_of_Kinematic_Bicycle_Model.html

[A2] Coulter, R. *Implementation of the Pure Pursuit Path Tracking Algorithm*. Carnegie Mellon University, Pittsburgh, Pennsylvania, Jan 1990.

[A3] Steven Waslander, Jonathan Kelly, *"Introduction to Self-Driving Cars"*, Coursera.

[A4] Hoffmann, Gabriel M., Claire J. Tomlin, Michael Montemerlo, and Sebastian Thrun. "Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing." *American Control Conference*. 2007, pp. 2296–2301. doi:10.1109/ACC.2007.4282788

[A5] Reid, Tyler & Houts, Sarah & Cammarata, Robert & Mills, Graham & Agarwal, Siddharth & Vora, Ankit & Pandey, Gaurav. (2019). Localization Requirements for Autonomous Vehicles. 2. 173-190. 10.4271/12-02-03-0012.

[A6] http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[A7] Alberto Bemporad, N. Lawrence Ricker, Manfred Morari. (n.d.). Model predective control toolbox. In Getting Started Guide.

[A8] David González, J. P. (2015). A Review of Motion Planning Techniques. IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS.

[A9] Iram Noreen, A. K. (2016). Optimal Path Planning using RRT* . (IJACSA) International Journal of Advanced Computer Science and Applications,.

[A10] Jason Kong, M. P. (2015). Kinematic and dynamic vehicle models for autonomous driving control design. IEEE Intelligent Vehicles Symposium (IV).

[A11] Michael Montemerlo1, J. B. (2017). Junior: The Stanford Entry in the Urban Challenge. Stanford Artificial Intelligence Lab.

[A12] Rajamani, R. (n.d.). In R. Rajamani, Vehicle Dynamics and Control (pp. XXVI, 498). Springer, Boston, MA.

[A13] https://www.mathworks.com/help/pdf_doc/nav/nav_ref.pdf

[A14] https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812456