DE-40(DEE)

YEAR 2022

NUST COLLEGE OF

ELECTRICAL AND MECHANICAL ENGINEERING

Path Planning of Autonomous Electric Vehicle

A PROJECT REPORT

DE-40 (DEE)

Submitted by

NC Muhammad Arslan Amjad Qureshi

NC Muhammad Hassan Ghafoor

NC Muhammad Talha Saleem

NC Muhammad Hashim

BACHELORS

IN

ELECTRICAL ENGINEERING

YEAR 2022

PROJECT SUPERVISOR

Dr. Fahad Mumtaz Malik

NUST COLLEGE OF

ELECTRICAL AND MECHANICAL ENGINEERING

PESHAWAR ROAD, RAWALPINDI

DEDICATION

This project is dedicated to our parents, their efforts and all the prayers were with us during the project. It is also dedicated to our teachers and all the technical staff who really helped us for the completion of this project.

CERTIFICATE OF APPROVAL

It is to certify that the project “**PATH PLANNING OF AUTONOMOUS ELECTRIC VEHICLE”** was done by **NC Muhammad Arslan Amjad, NC Muhammad Hassan Ghafoor, NC Muhammad Talha Saleem, and NC Muhammad Hashim** under the supervision of **Dr. Fahad Mumtaz Malik.**

This project is submitted to the **Department of Electrical Engineering**, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan in partial fulfillment of requirements for the degree of Bachelor of Engineering in Electrical engineering.

Students:

1- Muhammad Arslan Amjad

NUST ID: ______________________ Signature: _____________________

2- Muhammad Hassan Ghafoor

NUST ID: ______________________ Signature: _____________________

3- Muhammad Talha Saleem

NUST ID: ______________________ Signature: _____________________

4- Muhammad Hashim

NUST ID: ______________________ Signature: _____________________

APPROVED BY:

Project Supervisor: __________________ Date: ___________________
Dr. Fahad Mumtaz Malik

DECLARATION

We hereby declare that no portion of the work referred to in this Project Thesis has been submitted in support of an application for another degree or qualification of this of any other university or other institute of learning. If any act of plagiarism found, we are fully responsible for every disciplinary action taken against us depending upon the seriousness of the proven offence, even the cancellation of our degree.

1- Muhammad Arslan Amjad

NUST ID: ______________________ Signature: ______________________

2- Muhammad Hassan Ghafoor

NUST ID: ______________________ Signature: ______________________

3- Muhammad Talha Saleem

NUST ID: ______________________ Signature: ______________________

4- Muhammad Hashim

NUST ID: ______________________ Signature: ______________________

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

We are thankful to Allah Almighty, the most merciful and Al Fattah, for the blessings, courage,and the intellect He has bestowed us with, because of which it became possible for us to accomplish such a remarkable goal.

We are thankful to our parents and the family for their constant support and encouragement ateach thick and thin. We are much obliged to our supervisor Dr. Fahad Mumtaz for his continuous guidance and the valuable time he spent on providing us with the support and the encouragement through each stage. We are honored to have received his supervision in the project, enabling us to achieve the milestone. Moreover, we are grateful to all our fellows, friends who supported us intellectually and emotionally. Many thanks for all those who contributed in it, without all their support and appreciation we would be lacking a major sourceof inspiration and motivation. We appreciate the support from the Mechanical and Mechatronics members of our team who have provided us with immense assistance thought out the project playing a crucial role in the accomplishment. We are hopeful that this project will add value to the peoples' life and would be developed further to adopt this technology onthe massive level acting as a benchmark for future technologies of Pakistan.

ABSTRACT

The Path Planning Stack of the Autonomous Electric Vehicle mainly consists of three main modules which includes Global Path Planning (Planning through GPS, Maps and making an interface as well), Behavioral Path Planning (Creating all the drivable scenario and Vehicle Maneuvers according to the road rules and situations). Local path Planning (it is responsible for the execution of the Maneuvers at the road). Firstly, for our Autonomous Vehicle, Global Path Planning is executed which is implemented through Graphical user interface (GUI) which is synchronized with jetson Xavier AGX via ROS nodes and Global path planning also finds accurate starting location using the global positioning unit (GPU), so the vehicle can easily localize its position and create waypoints till destination. Behavioral path planning module utilizes waypoints and implements all the possible scenarios which a vehicle can maneuver while following its path for e.g. taking a U-turn, maintaining safe distance from the lead vehicle etc. In last execution of the maneuvers is efficiently implemented via Local Path Planning module.

Contents

LIST OF FIGURES

CHAPTER - 1. INTRODUCTION

1.1 Background

Every year, around 1.35 million people are killed in automobile accidents (WHO, 2020). Every year, among the persons (WHO, 2020). Speeding, distracted driving, and driving under the influence are all preventable human errors that contribute to road traffic deaths. Autonomous driving features can warn the driver of potential dangers.

There could be $450 billion in savings for automakers, insurers, and governments if 90% of American roads were covered by self-driving cars, according to a University of Texas study. As a result, legislators would have a strong incentive to clear the way for self-driving cars in the future. Autonomous vehicles are still decades away from widespread use, but the benefits of increased security, efficiency, and comfort will hasten the process.

Fully autonomous driving systems must have a high level of awareness of the vehicle's surroundings and be able to plan the vehicle's motion in a way that does not endanger nearby pedestrians or traffic, while maintaining some level of driving comfort inside the vehicle by minimizing sudden acceleration or changes in direction.

Autonomous vehicles use path planning to map out their routes and find their way around. Because of dynamic environments, safety, smoothness and real-time requirements and vehicle non-holonomic constraints, the study of path planning methods has always been a difficult problem, especially in complex environments. A variety of approaches have been proposed in this study to address the challenge of navigating through densely populated areas with numerous hazards.

The thesis explains different methods of Path Planning for Autonomous vehicle and the approach which we have used for the Path Planning. As Path Planning is computationally expensive process so the goal is to use such an approach which in not only computationally feasible but also generates the best results for us. Although Path Planning is one module of a self-driving car but it is further divided into different modules which are further sub-divided. There are basically three modules of Path Planning of Autonomous Vehicle which are:

Global Path Planning

The highest level of Path Planning is Global Path Planning. This module requires GPS, Google maps API and different graph-based search methods to function. The initial position of the vehicle is obtained from GPS and the destination is entered by the user through Google maps. Different Search Algorithms are used to obtain the most optimal path. This module is abstract from low level details like traffic signs, obstacles etc.

Behavioral Planning

An autonomous vehicle's "brain" is the behavior planner. If no obstacles are present, then the global planning layer generates kinematically and dynamically feasible paths. This is followed by a behavioral planning layer that considers both static and dynamic obstacles. A Finite State Machine (FSM) is the foundation of our behavioral planner layer. Three states are available to it, which are follow lane, decelerate, and stop.

Local Planning

The motion planners of autonomous vehicles are responsible for determining the vehicle's path through its environment while avoiding any obstacles. Once the local path planner generates a trajectory and velocity profile, it passes this information along via wirelessly to the vehicle's control systems. Using the Attractor Dynamic Approach (ADA), this report proposes a new local path planner for autonomous vehicles that is inspired by the movement of living things.

1.2 Project Objectives

The specific scope and objectives of the Final year project are as follows:

- Development of a prototype Self-Driving Car through conversion of an electric vehicle (done by superposing the vehicle manual control with electric actuation for throttle, steering, and braking).
- Achieving high level of autonomy wherein the vehicle itself is capable of driving from a start point to a destination on a well-structured road with clear lane markings and minimal rush irregularities. The human driver inside the vehicle will have the

supervisory control in case of emergencies.

- Integrating maps, perception sensors and motion sensors for the car to be able to perceive. While reducing sensor dependencies, to only camera-based perception.
- Develop a benchmark system which will facilitate subsequent research in this area for both academia and automobile industry.

1.3 Design Workflow

The implementation methodology for the project would be two dimensional. One of theaspects would be development of the Self Driving Car prototype, embedding the existing technologies and algorithms. The other one would be extension of state-of-the-art research in this domain. Eventually both the research and development algorithms will be embedded so that a prototype of Self Driving Car with efficient algorithms is produced as an outcome.

The low-level control system's trajectories are determined by the Path planning system. The user specifies a destination, and the computer determines how to complete the journey. The path planning task involves generating trajectories that ensure the vehicle arrives at its destination safely, using information from maps, vehicle dynamics, and an assessment of the current condition.

After a user enters the desired location, the vehicle consults maps to determine the best route to go. After that, a complex software processes all of the sensory inputs to determine the trajectory, and the instructions are delivered to the vehicle actuators, which control the vehicle's steering, braking, and accelerating. The self-driving car's is also dependent on Perception stack and Control Systems. It is the integration of all the three modules (Perception, Planning and Controls) that ensures the smooth working of the autonomous vehicle through the environment. In the below figure 1 it is shown that how the Path Planning module is related to Perception and Control Systems module and gives an idea how self-driving car works:

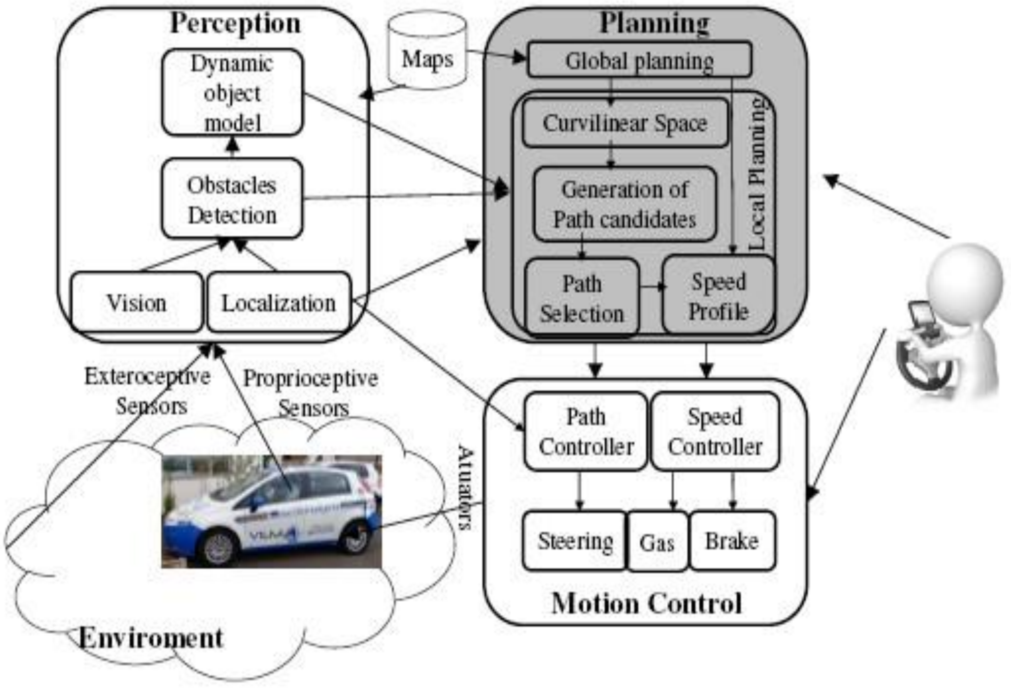


Fig 1. Different Modules of Autonomous vehicle and their link

CHAPTER - 2. LITERATURE REVIEW

2.1 Path Planning

2.1.1 Global Path Planning:

There is a lot of discussion about path planning in the autonomous vehicle navigation field. Our self-driving automobile can determine an optimal or nearly optimal path from the beginning state to the goal state based on one or more motion space performance metrics (such as the lowest working cost, shortest walking route, shortest walking time, and so on). [2-4] The shortest distance between two points is generally utilised as an indicator of performance when it comes to path design.
Classification of planning approaches can be based on two forms of environmental knowledge. Path planning can make use of both global and local map data [5]. The global path planning approach can be used to generate the path in a well-known setting (the position and shape of the obstacle are predetermined). Searches are based on maps of the area where mobile robots are stationed. Even the most sophisticated algorithm can uncover the best route. Global strategy development begins with the creation of an environmental modelling framework, which is then used as a basis for developing a path. [6]
Creating an environmental map is a prerequisite for path planning [7]. Environmental map building is essential for creating a spatial model or map that accurately depicts multiple elements in an area, such as obstacles, road signs, and so on. To help the vehicle choose the optimal path across an obstacle-filled environment model, an environmental map must be created.

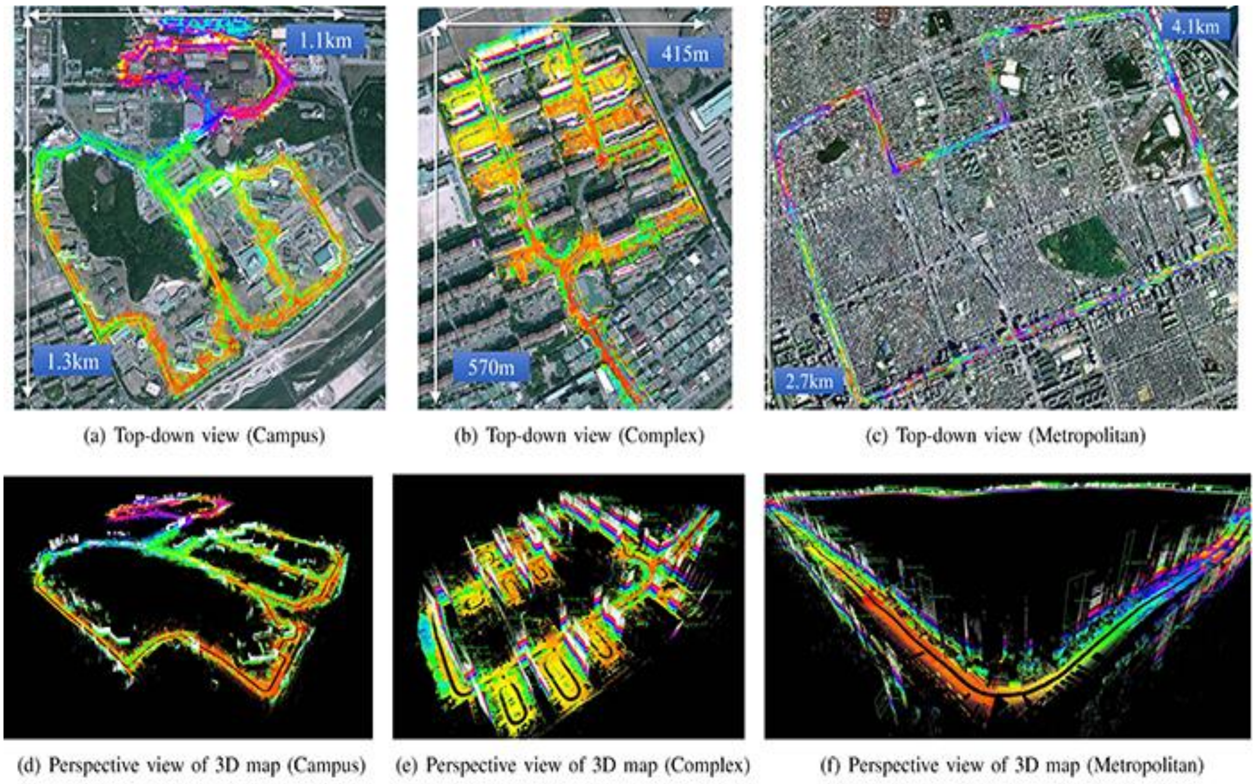

1.1km
1.3km
415m
570m
4.1km
2.7km
(a) Top-down view (Campus)
(b) Top-down view (Complex)
(c) Top-down view (Metropolitan)
(d) Perspective view of 3D map (Campus)
(e) Perspective view of 3D map (Complex)
(f) Perspective view of 3D map (Metropolitan)

Fig 2. Simultaneous map building

For path planning, there are a variety of established approaches for constructing an environment model. A grid decomposition map, quad split graph, visibility graph, and Voronoi diagram are some of the basic environment models that are now available

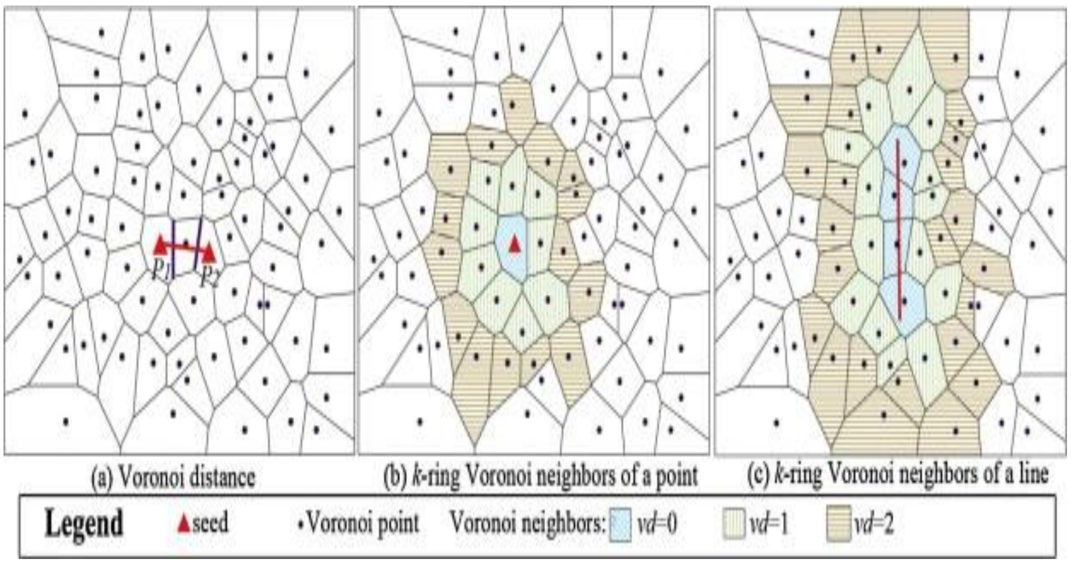

p_1
p_2
(a) Voronoi distance
(b) *k*-ring Voronoi neighbors of a point
(c) *k*-ring Voronoi neighbors of a line
Legend
▲seed
•Voronoi point
Voronoi neighbors: ▨ *vd*=0 ▨ *vd*=1 ▨ *vd*=2

Fig 3. Mapping by Voronoi Diagram

Following the compilation of an environmental map, global path planning is carried out. Heuristic search methods and intelligent algorithms are two main groups of global path planning algorithms. When using a heuristic search, the Dijkstra algorithm generates the A* algorithm as the starting point. Most commonly used in state-space heuristic graph search is the A* algorithm. In addition to dealing with state space issues, robot path planning is a common use for it. Robot route planning is another common usage for state space analysis. Several academics have refined the A* algorithm and created new heuristic search methods.

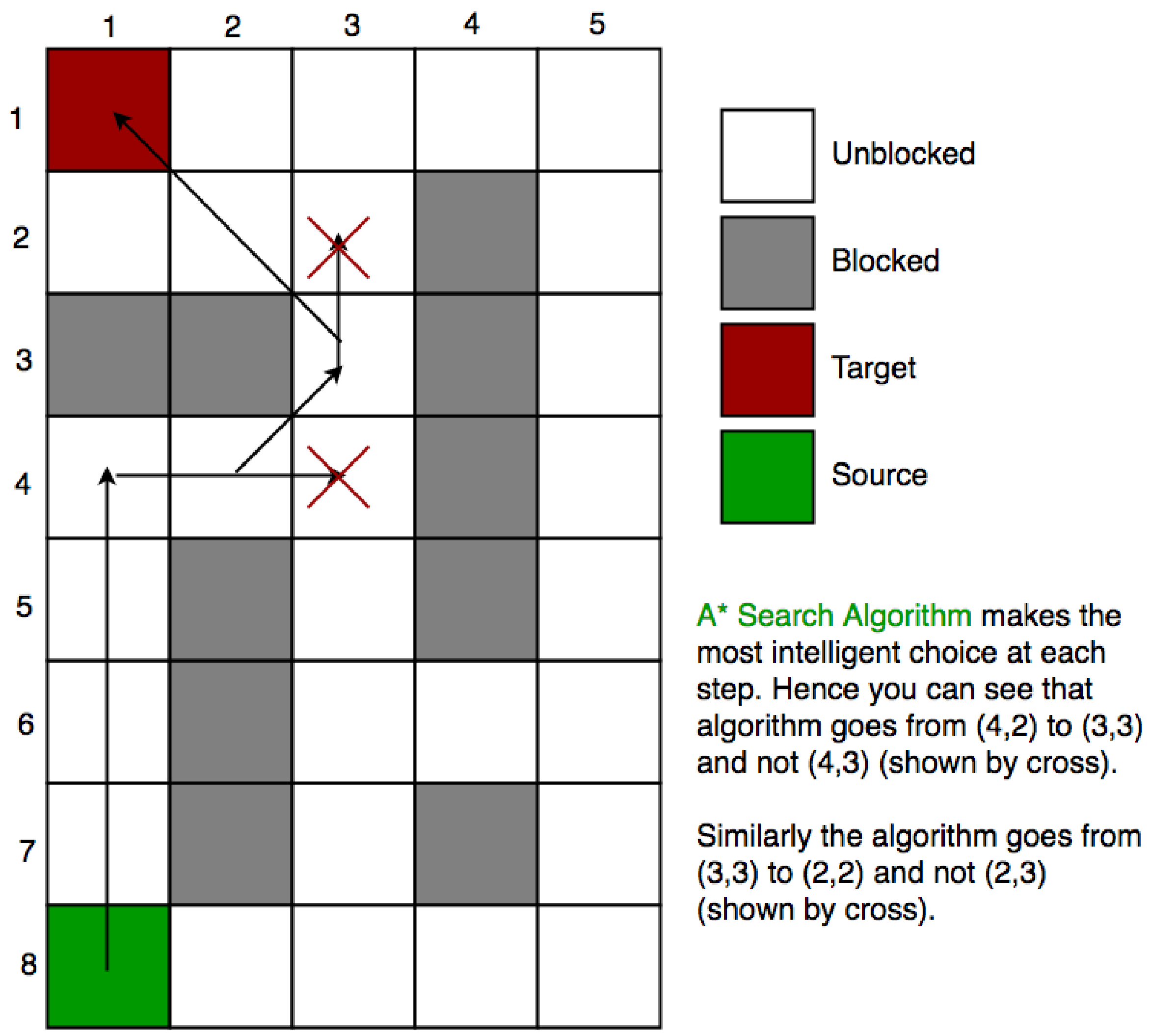
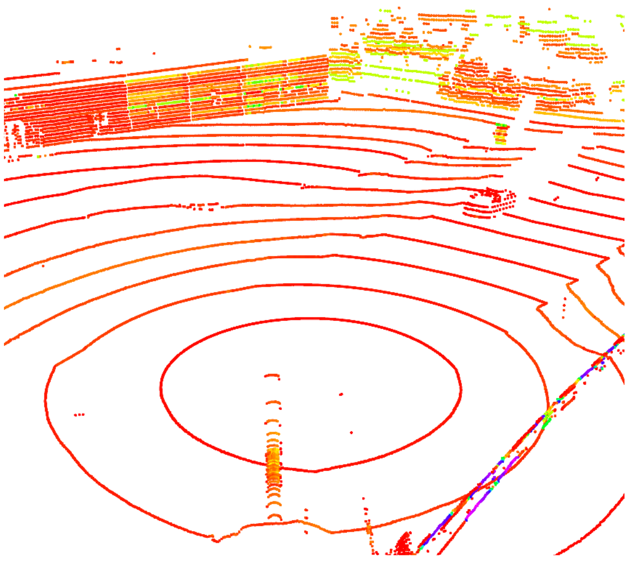

1
2
3
4
5
1
2
3
4
5
6
7
8
Unblocked
Blocked
Target
Source
A* Search Algorithm makes the most intelligent choice at each step. Hence you can see that algorithm goes from (4,2) to (3,3) and not (4,3) (shown by cross).
Similarly the algorithm goes from (3,3) to (2,2) and not (2,3) (shown by cross).

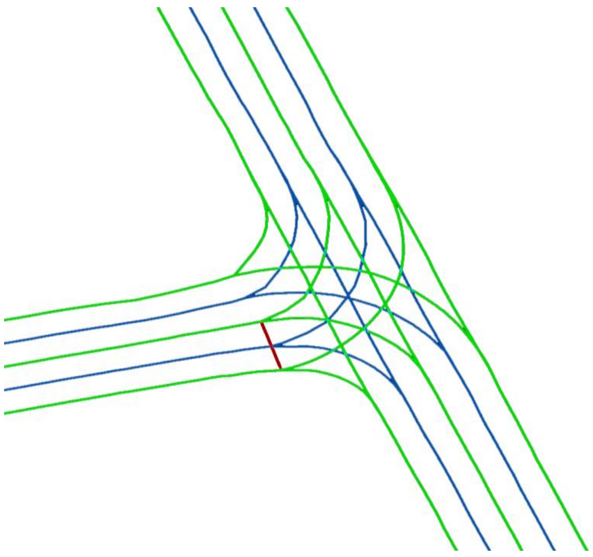
Fig 4. A Algorithm to find optimal path*

2.1.2 Behavioral Planning

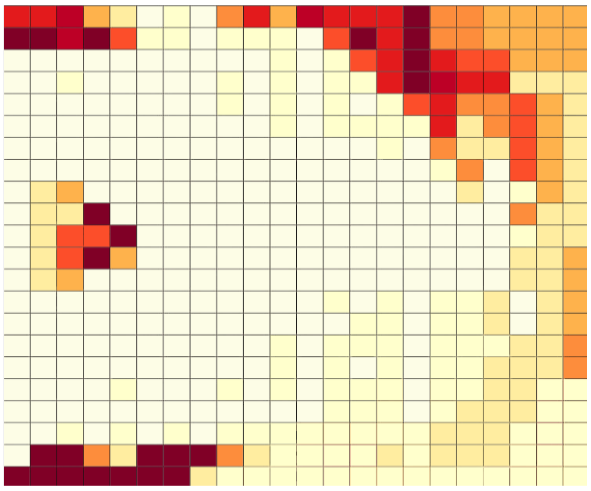
In terms of autonomous driving behavior planning, we can examine the design space [8]. (high-level decision making). On the basis of my own experience [9] We simplify the design process by dividing the design space into three axes based on the current state of the art. We examine the relevant research to determine the unavoidable qualitative trade-offs that exist within each axis. Our decomposition is depicted using feature diagrams (Fig. 10). Because of this, we've come up with a list of possible research avenues in the field of behavior planning design. It's because of the following that we've had to call it quits:

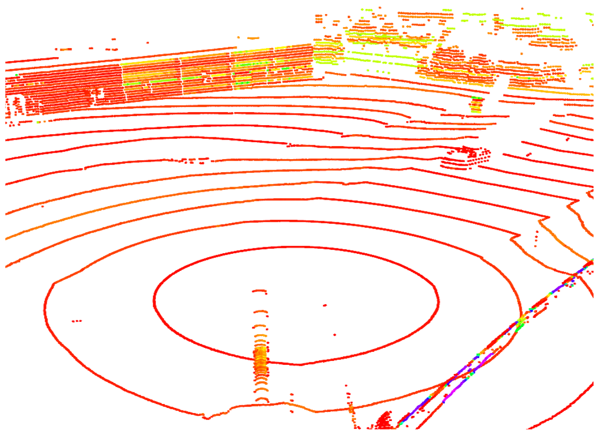
Road connections, signs and signals as well as interactions between drivers and other road users all contribute to discrete driving episodes. In any case, the vehicle must maintain a smooth, uninterrupted path on the road. The spectrum of conceivable discrete and continuous abstractions is one of the first major dimensions of design space that we investigate. As you can see in Figure 5, we used our autonomous driving stack to generate a variety of different visual representations.

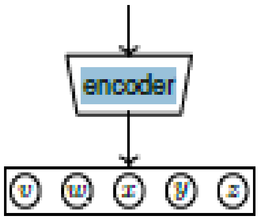




(a). Raw data: Lidar point cloud *(b) Feature based Road Map*







(c) Grid-based: 2D occupancy grid

(d) Latent representation

Fig 5. Four environment representations used in motion planning for autonomous driving, ordered from least to most abstract

A behavior planner (BP) deals with high-level decisions (such as whether to drive straight, turn left, or stop), whereas a local planner deals with low-level continuous actions (such as turning and braking) (LP). Optimization is better adapted to finding continuous activities than procedural systems are. On the road, things can't always be predicted, therefore the vehicle's behavior and that of other road users must be taken into account when making decisions on how to regulate it. As we move forward, we'll focus on the motion planner's overall design. The degree of integration and communication between the systems is taken into account on the second axis of decomposition. The most significant influence on the design of BP comes from the interaction between behavior and the local planner. Design axis number three is the depiction of decision making. Because of recent advances in hardware and software, learned logic is a viable and appealing alternative to traditional programs and expert systems. Our three primary axes of design choices are shown in Fig as a feature diagram. Additionally, the graphic includes a reference to other visual syntaxes.

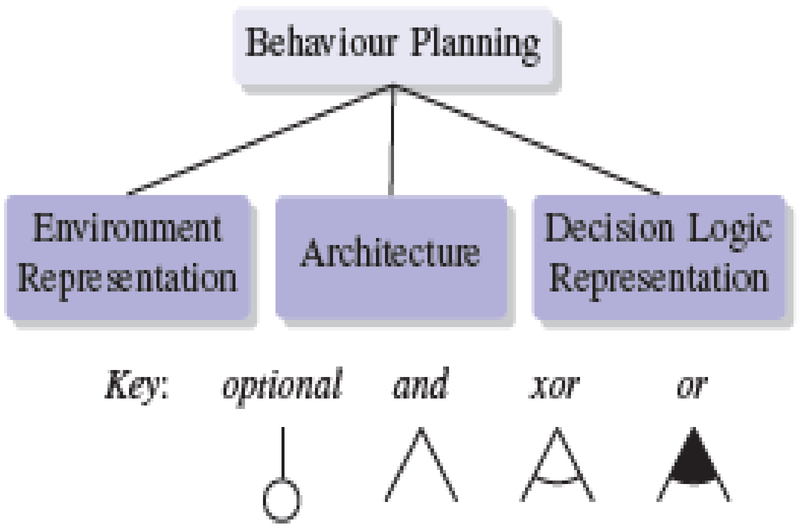


Fig 6. Feature diagram of high-level design choices

2.1.2.1 Architecture of the Motion Planner:

Integrating the BP into the motion planner is essential because it constrains the other components and the overall architecture [11-13]. On the one hand, it is architecturally possible to keep the BP module separate and separate from other parts of the motion planner, while on another, it is possible to integrate it partially or completely, as shown in Figure 5. The placement of motion planner elements in Fig. 7 shows that many early motion planning techniques had a top-down architecture design, as described in the issue specification.

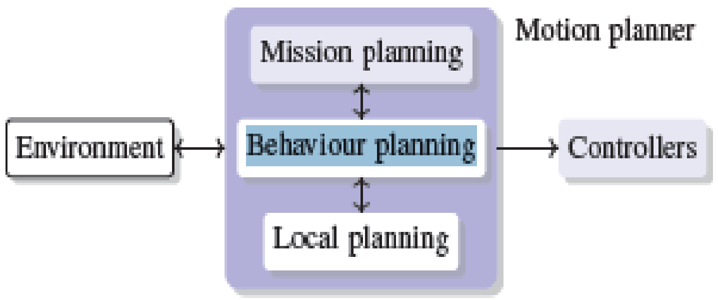


Fig 7. General architecture of motion planning. White boxes denote the scope of the current work.

2.1.3 Local Path Planning:

Planned routes that consider both the first and second layers of an MP are referred to as local paths. Vehicle dynamics are considered to generate practical paths, and not just practical ones but paths that are comfortable for humans to use. Numerous strategies have been proposed so far to generate a local path [14]. It's possible to categorize them as follows [15].

- **Space configuration**:

Candidate solutions, such as lattices, Voronoi diagrams, or sampling points, are applied to each cell that is free from collisions. To avoid accidents, they search for the best arrangement of linked cells to reach a specific local waypoint. This family of algorithms is fast, however the supplied solutions may not always be dynamically possible for the vehicle. This is a disadvantage. In addition, the decomposition of the space is affected by impediments and the ego-vehicle condition.

- **Path finders**:

A graph's nodes are searched for a path between them. At least one cost function, such as distance travelled or time spent, must be considered when creating a graph and determining the best path. A*, Dijkstra, and Rapidly Random Tree (RRT) algorithms are the state-of-the-art algorithms. The RRT-based algorithms are used in situations where the environment is unknown, whereas the first two are used in situations where the environment is known. One of the main drawbacks is that the given solutions may not be feasible for the vehicle [19-21].

- **Attractive and repulsive forces**:

Instead of using natural forces to guide the vehicle, these methods use artificial forces to guide the vehicle to its intended destination. Forces combine to change the vehicle's trajectory. It is possible for these algorithms to get bogged down in local minima or generate a solution that is inherently unstable for the vehicle while searching for a solution [22].

- **Artificial intelligence schemes:**

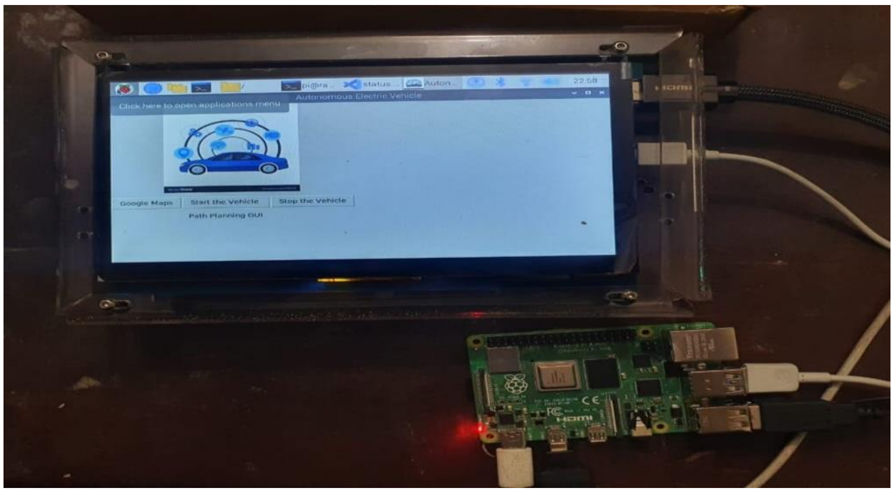
Solution algorithms utilize human-level reasoning in order to solve the problem. Local path planning can be tackled using fuzzy logic, artificial neural networks, swarm intelligence, and evolutionary algorithms [23,24]. The main advantage of this method is that it doesn't require the building of exact mathematical models. ' Fuzzification, interpretation, and defuzzification are all steps in FL that the designer should be familiar with. For example, FL has many more rules, a more complex defuzzification process and a wider range of member functions than this simple approach. It all depends on the designer's skill level, of course See [25,26] and the references therein for more information on artificially intelligent systems utilized in local autonomous vehicle path creation. However, due of their complexity, real-time computers have yet to successfully incorporate these artificial intelligence algorithms. When prior local path techniques were too restrictive, several of these approaches were employed in conjunction. The Probabilistic Road-map Algorithm [27] is one example of an algorithm that makes use of numerical optimization. Many curves' algorithms are implemented along an optimizer that considers objective functions such as the minimum distance or arrival time or energy that are relevant to your vehicle's dynamics when designing a feasible path. In order to improve the depiction of a driven corridor, the APF algorithm has been added into the attractive and repulsive forces algorithm.

CHAPTER - 3. GLOBAL PATH PLANNING

3.1 Communication with GUI

The odometry information such as speed and location will be communicated on a local network between NVidia Jetson Xavier and Raspberry pi through Ethernet, which will communicate the Track information along with the A* map to be plotted and updated on the map. Raspberry pi will also receive the Battery status from the Ethernet connection, which will be calculated by the State of Charge of the Battery. The CAN-BUS will be connected to the sensor to measure the battery voltage, which will determine the SOC. The A* algorithm can be implemented entirely on Raspberry pi and the waypoints can be communicated to Xavier, to be used in Local Planning, since the waypoints will be constant throughout the journey, so only one-time communication is required. The display for the vehicle environment can be generated using the forward camera feed, which will be preprocessed by Xavier, and the real-time output of the road plane equation, road boundaries, and the obstacle bounding box coordinates will be published on a local IP that Raspberry Pi can access in real-time. The Raspberry pi will plot these lines using the Python GUI library, and replace bounding boxes with the vehicle logos, with their relative size equal to the bounding box, and the logo respective to the label of the bounding box. The equipment required for User Interface is listed below:

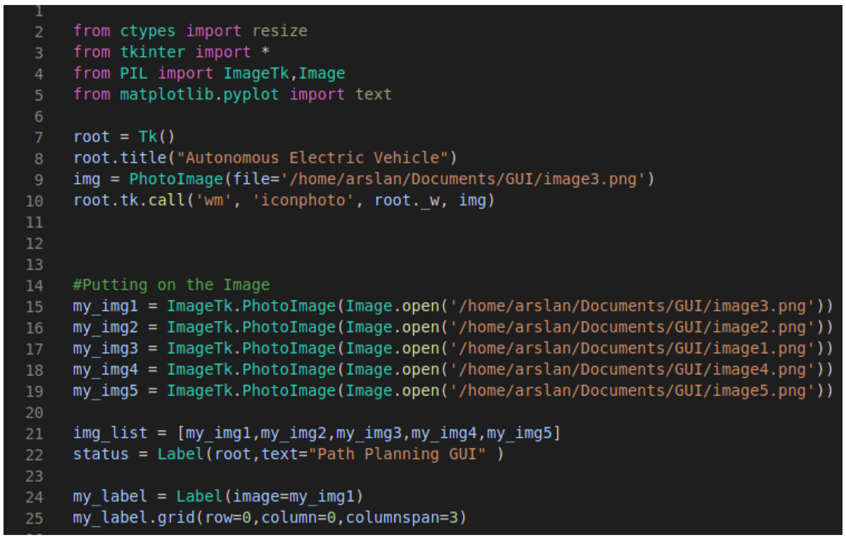
Raspberry pi 4	Value
Model	Raspberry Pi 4 Model B
RAM	4 GB
Network	Wi-Fi 802.11b / g / n
USB/s	2 USB 3.0 ports 2 USB 2.0 ports
Architecture	ARM 64
Wireless	2.4 GHz and 5.0 GHz IEEE 802.11
HDMI ports	2 micro-HDMI ports up to 4kp60
Display	


Autonomous Electric Vehicle
Click here to open applications menu
Google Maps
Start the Vehicle
Stop the Vehicle
Path Planning GUI
Fig 8. Raspberry Pi along with LCD

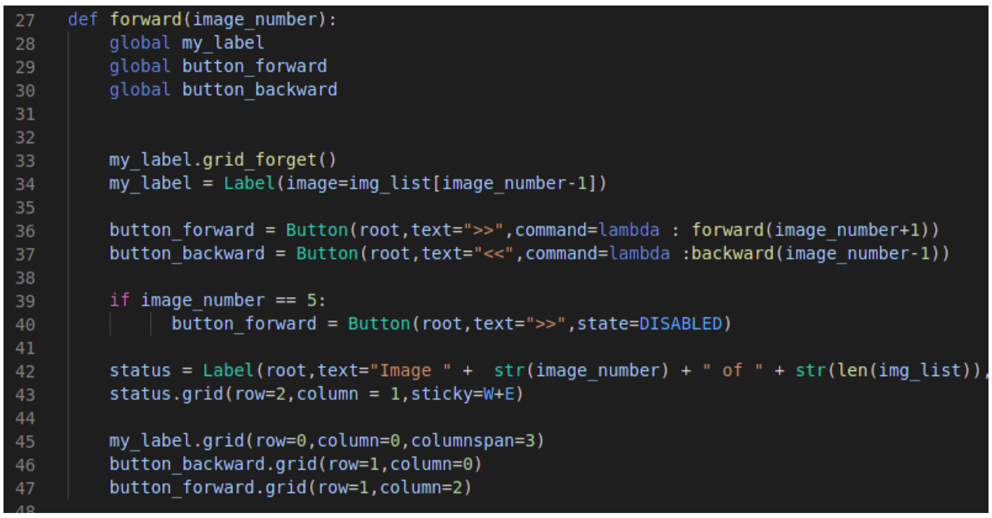
Graphical User Interface:

Tkinter and PyQT5 are two Python libraries that are used to implement the graphical user interface. The simplicity of the Tkinter library's syntax is one of its primary draws. The text widget packs a powerful punch while remaining incredibly user-friendly. Easy-to-use and powerful is also the case with the canvas widget. Those two widgets are the only ones in any other toolkit that offer the same balance of simplicity and power. PyQt5 GUI programming, on the other hand, is based on the concept of signals and slots for establishing communication between objects. That allows for greater flexibility in dealing with GUI events and a cleaner codebase as a result. Both libraries have been used in our GUI.

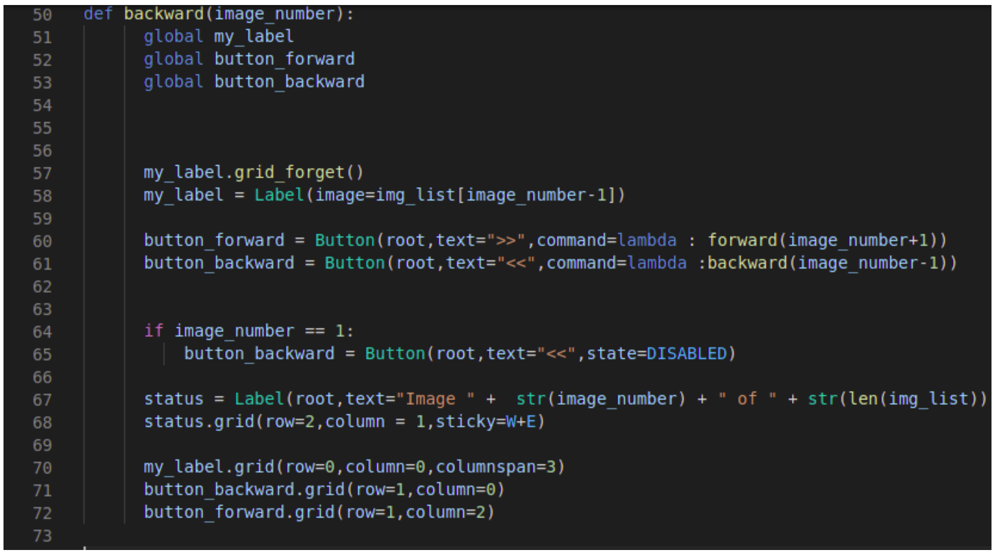
3.2.1 Tkinter:



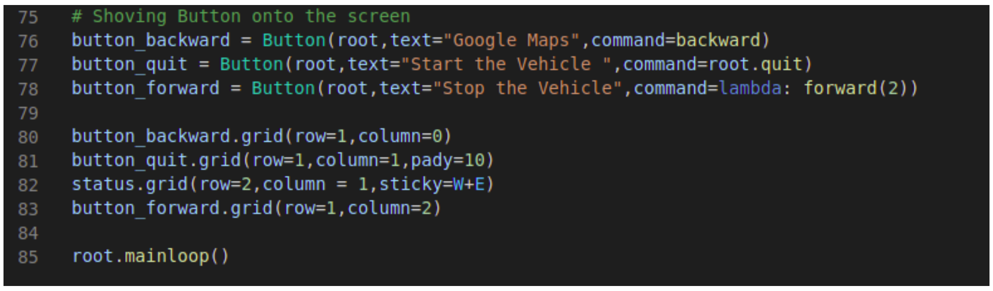
```
from ctypes import resize
from tkinter import *
from PIL import ImageTk,Image
from matplotlib.pyplot import text

root = Tk()
root.title("Autonomous Electric Vehicle")
img = PhotoImage(file='/home/arslan/Documents/GUI/image3.png')
root.tk.call('wm', 'iconphoto', root._w, img)



#Putting on the Image
my_img1 = ImageTk.PhotoImage(Image.open('/home/arslan/Documents/GUI/image3.png'))
my_img2 = ImageTk.PhotoImage(Image.open('/home/arslan/Documents/GUI/image2.png'))
my_img3 = ImageTk.PhotoImage(Image.open('/home/arslan/Documents/GUI/image1.png'))
my_img4 = ImageTk.PhotoImage(Image.open('/home/arslan/Documents/GUI/image4.png'))
my_img5 = ImageTk.PhotoImage(Image.open('/home/arslan/Documents/GUI/image5.png'))

img_list = [my_img1,my_img2,my_img3,my_img4,my_img5]
status = Label(root,text="Path Planning GUI" )

my_label = Label(image=my_img1)
my_label.grid(row=0,column=0,columnspan=3)
```



```python
def forward(image_number):
    global my_label
    global button_forward
    global button_backward


    my_label.grid_forget()
    my_label = Label(image=img_list[image_number-1])

    button_forward = Button(root,text=">>",command=lambda : forward(image_number+1))
    button_backward = Button(root,text="<<",command=lambda :backward(image_number-1))

    if image_number == 5:
        button_forward = Button(root,text=">>",state=DISABLED)

    status = Label(root,text="Image " +  str(image_number) + " of " + str(len(img_list)),
    status.grid(row=2,column = 1,sticky=W+E)

    my_label.grid(row=0,column=0,columnspan=3)
    button_backward.grid(row=1,column=0)
    button_forward.grid(row=1,column=2)
```



```python
def backward(image_number):
    global my_label
    global button_forward
    global button_backward



    my_label.grid_forget()
    my_label = Label(image=img_list[image_number-1])

    button_forward = Button(root,text=">>",command=lambda : forward(image_number+1))
    button_backward = Button(root,text="<<",command=lambda :backward(image_number-1))


    if image_number == 1:
        button_backward = Button(root,text="<<",state=DISABLED)

    status = Label(root,text="Image " +  str(image_number) + " of " + str(len(img_list))
    status.grid(row=2,column = 1,sticky=W+E)

    my_label.grid(row=0,column=0,columnspan=3)
    button_backward.grid(row=1,column=0)
    button_forward.grid(row=1,column=2)

```



```python
# Shoving Button onto the screen
button_backward = Button(root,text="Google Maps",command=backward)
button_quit = Button(root,text="Start the Vehicle ",command=root.quit)
button_forward = Button(root,text="Stop the Vehicle",command=lambda: forward(2))

button_backward.grid(row=1,column=0)
button_quit.grid(row=1,column=1,pady=10)
status.grid(row=2,column = 1,sticky=W+E)
button_forward.grid(row=1,column=2)

root.mainloop()
```

Output:

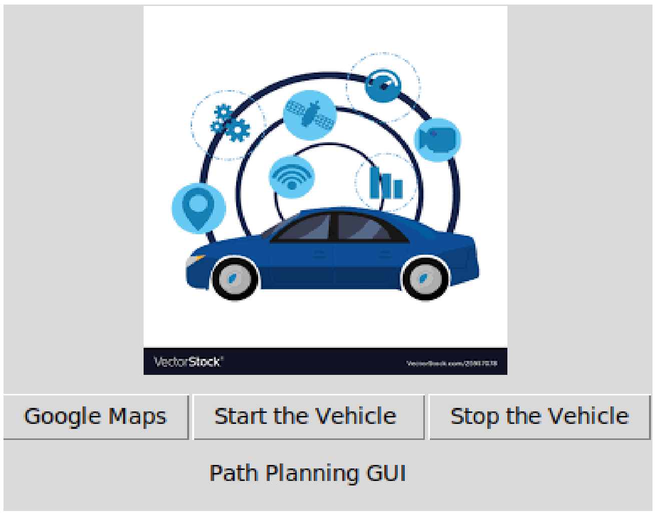
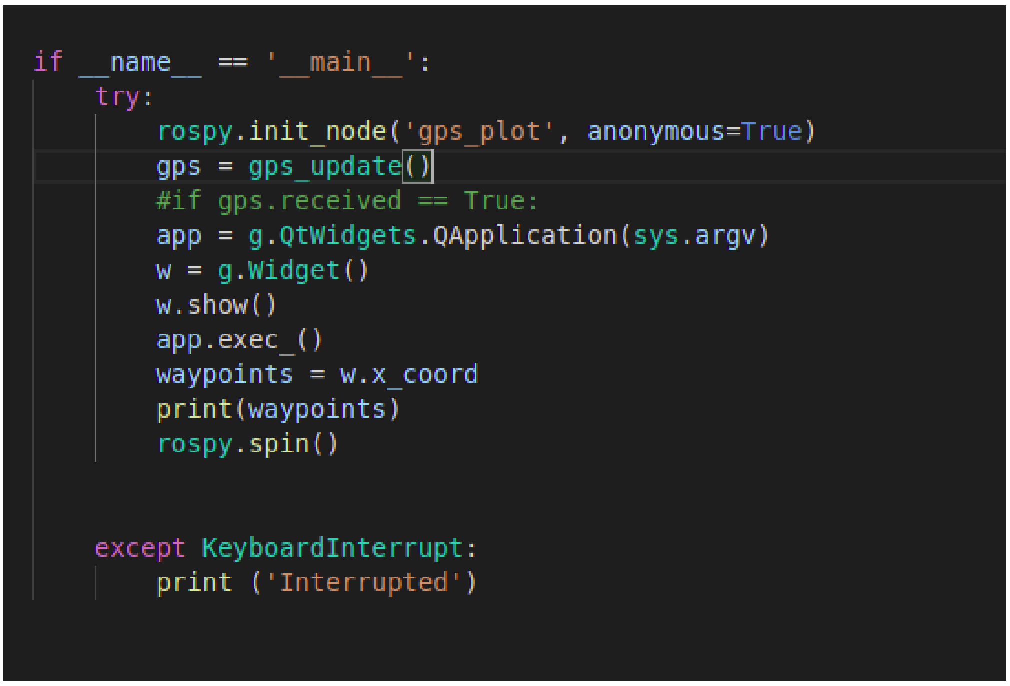


Fig 9. Layout of GUI

The three main functionalities are shown above. From Google Maps the user can enter the destination (end position). Starting and Stopping of the vehicle is also given in the hands of the user.

3.2.2 PyQt5:



```python
if __name__ == '__main__':
    try:
        rospy.init_node('gps_plot', anonymous=True)
        gps = gps_update()
        #if gps.received == True:
        app = g.QtWidgets.QApplication(sys.argv)
        w = g.Widget()
        w.show()
        app.exec_()
        waypoints = w.x_coord
        print(waypoints)
        rospy.spin()


    except KeyboardInterrupt:
        print ('Interrupted')
```

Output:

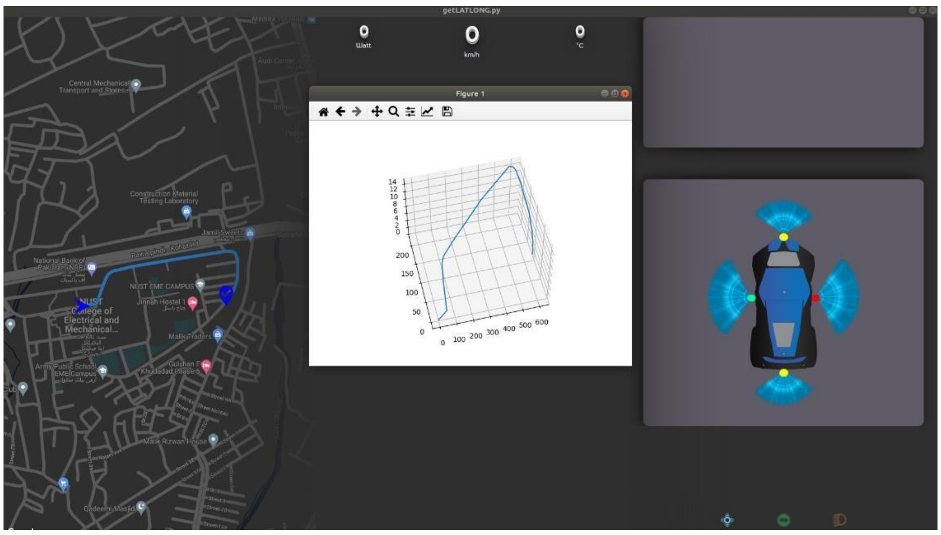


Fig. 10 GUI Code Output

3.3 GPS & Waypoint Generation:

Waypoint Generation is very essential for Global Path Planning. The Output of the Waypoint generation code will give us optimized waypoints from starting location to the end location.

The starting and ending location are the two inputs that are given to the waypoints code. The starting location can be found by the GPS module which is ublox NEO-M8N and the ending location will be given by the user through the GUI. These locations will be in the form of latitude and longitude. The Waypoint generation code will take these latitudes and longitudes of starting and ending point and it will generate waypoints throughout the path. This path will be given further to the Behavioral Planner.

3.3.1 GPS Module (ublox NEO-M8N):

ublox NEO-M8 Series GNSS Modules provide high sensitivity and minimal acquisition times while maintaining low system power. The NEO-M8 series of standalone concurrent GNSS modules is built on the exceptional performance of the ublox engine in the industry-proven NEO form factor.

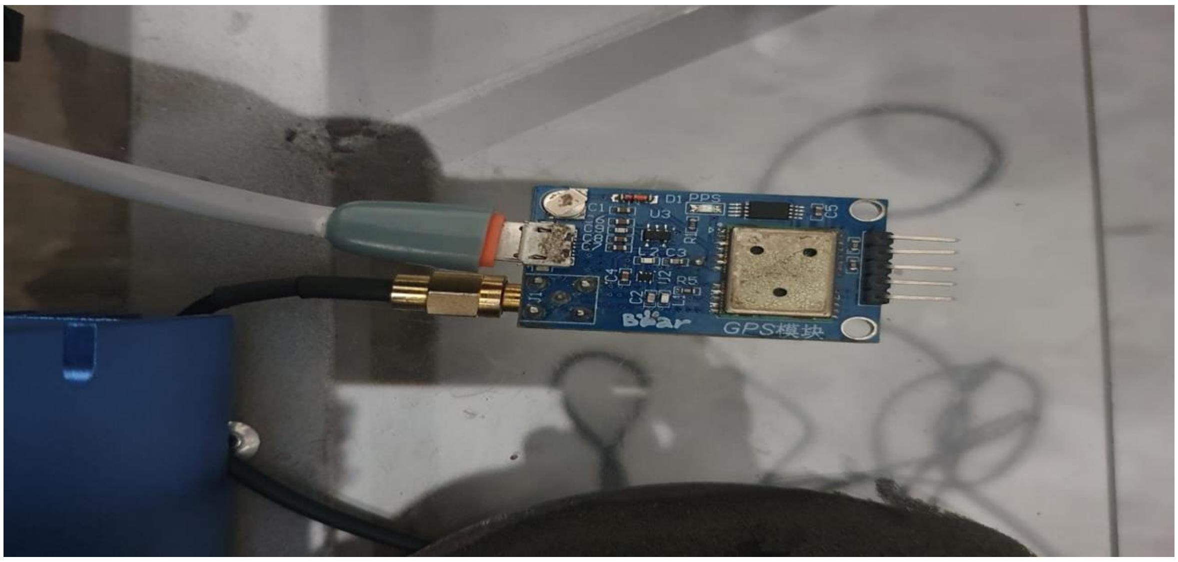


Fig. 11 GPS Module

GPS ublox Antenna:

These signals are then converted to an electronic signal that a GNSS or GPS receiver can use by using an antenna for receiving and amplifying the radio signals sent out by GNSS satellites at specific frequencies. In order to determine one's exact location, one uses a GNSS or GPS antenna's output to feed into a receiver.

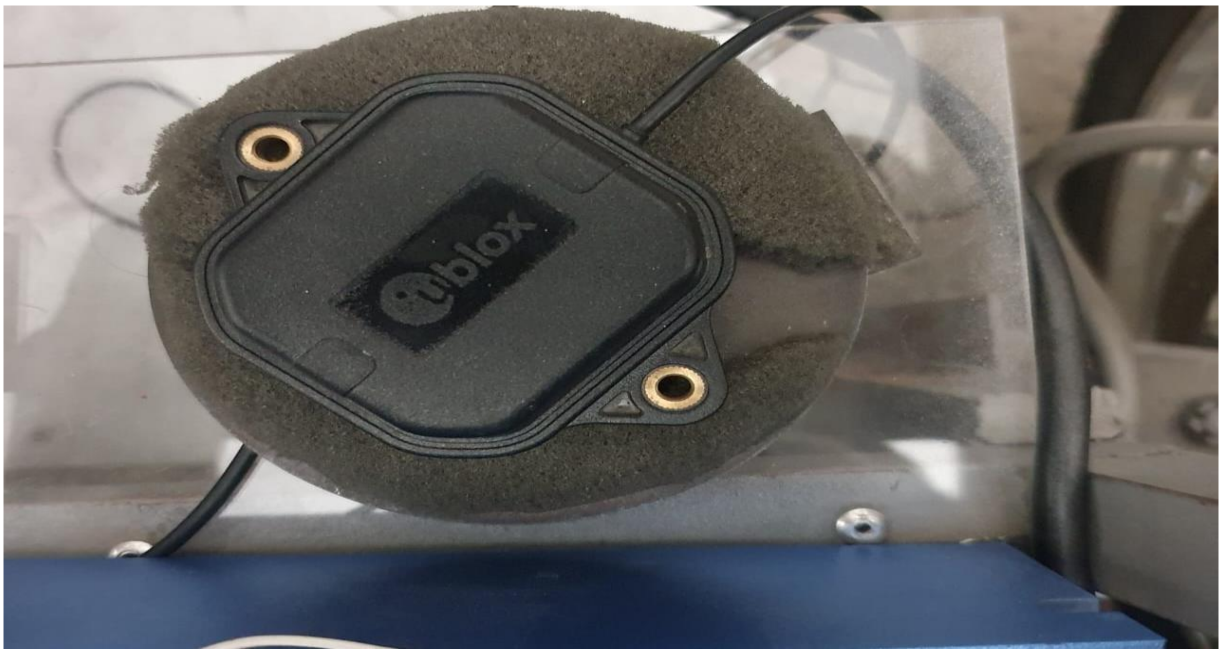
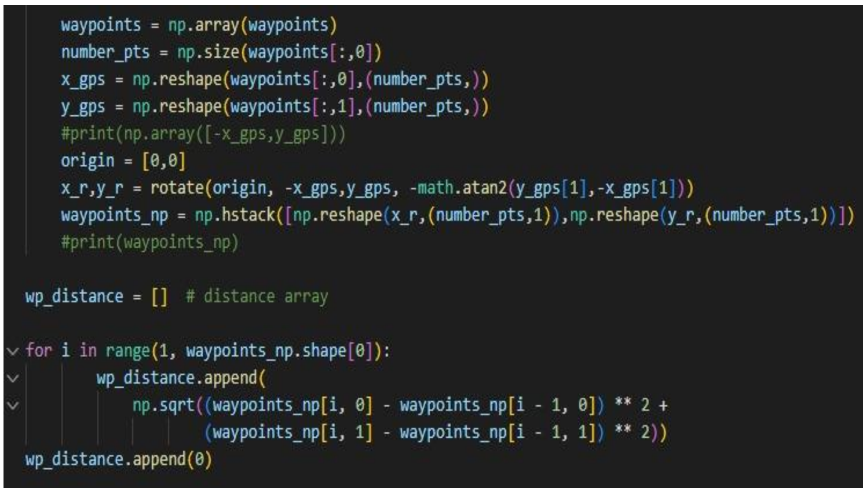


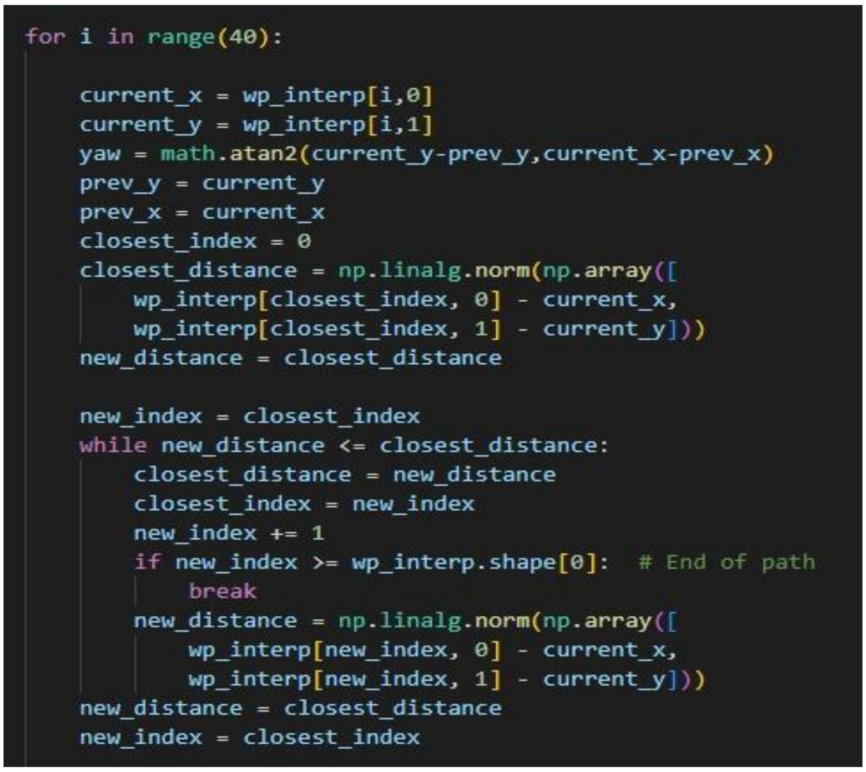
Fig. 12 U-blox Antenna

Waypoint Generation Code:

3.3.2.2 Taking Initial Waypoint:


```
        waypoints = np.array(waypoints)
        number_pts = np.size(waypoints[:,0])
        x_gps = np.reshape(waypoints[:,0],(number_pts,))
        y_gps = np.reshape(waypoints[:,1],(number_pts,))
        #print(np.array([-x_gps,y_gps]))
        origin = [0,0]
        x_r,y_r = rotate(origin, -x_gps,y_gps, -math.atan2(y_gps[1],-x_gps[1]))
        waypoints_np = np.hstack([np.reshape(x_r,(number_pts,1)),np.reshape(y_r,(number_pts,1))])
        #print(waypoints_np)

    wp_distance = []  # distance array

    for i in range(1, waypoints_np.shape[0]):
            wp_distance.append(
                np.sqrt((waypoints_np[i, 0] - waypoints_np[i - 1, 0]) ** 2 +
                        (waypoints_np[i, 1] - waypoints_np[i - 1, 1]) ** 2))
    wp_distance.append(0)
```

3.3.2.2 Locating next waypoint:



```
for i in range(40):

    current_x = wp_interp[i,0]
    current_y = wp_interp[i,1]
    yaw = math.atan2(current_y-prev_y,current_x-prev_x)
    prev_y = current_y
    prev_x = current_x
    closest_index = 0
    closest_distance = np.linalg.norm(np.array([
        wp_interp[closest_index, 0] - current_x,
        wp_interp[closest_index, 1] - current_y]))
    new_distance = closest_distance

    new_index = closest_index
    while new_distance <= closest_distance:
        closest_distance = new_distance
        closest_index = new_index
        new_index += 1
        if new_index >= wp_interp.shape[0]:  # End of path
            break
        new_distance = np.linalg.norm(np.array([
            wp_interp[new_index, 0] - current_x,
            wp_interp[new_index, 1] - current_y]))
    new_distance = closest_distance
    new_index = closest_index
```

CHAPTER - 4. BEHAVIORAL PLANNING

4.1 Behavior Planner:

The behavioral planner for our ego vehicle is based on a Finite State Machine. In our operational design domain, the stop signs and traffic lights are the regulatory elements. The behavioral planner consists of three finite states, a follow lane state, decelerate state, and stop state. During the normal operation of the autonomous vehicle, the follow lane state is active until our path planning algorithm encounters a stop sign or a traffic light. The finite state machine then switches the active state to decelerate and eventually stop at the destination planned, using sufficient time to safely maneuver and stop. Once stopped we gauge the status of the traffic light and as soon as it turns green, the vehicle starts to move, and a fixed duration of 30 seconds is configured at the stop signs. Once that is done, we then transition back to the follow lane state and continue with nominal driving. This completes the state machine transition cycle and allows us to handle the presence of a stop sign along our path. In Additional, we will also be embedding another state that will depict the Alternate Lane finder Execution and it could be to find alternate paths in order when our vehicle comes to stop when it detects the obstacle/object in front of it. The Finite State Model is shown in the below figure.

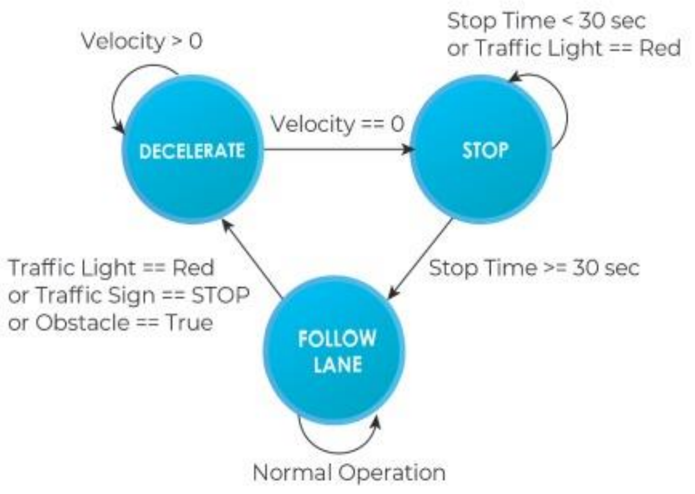


Fig. 13 Finite State Model

4.2 Traffic-free Reference Planning:

Reference planning layer takes lane-level submissions and produces a driving path and speed profile for the autonomous vehicle. The planner assumes that there will be no traffic on the road in this layer. Considering the kinematics and dynamics of the autonomous vehicle and the geometry of the road, it employs non-linear optimization to find a smooth and human-like path and speed.

4.3 Behavioral Executive System

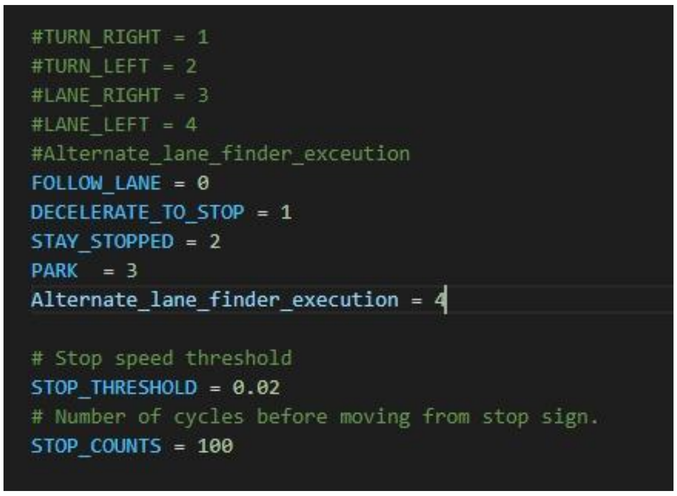
The Behavioral Executive was tasked with two primary responsibilities in terms of its job duties. Following Urban Challenge rules on logical traffic interaction, it had to ensure that the robot was adhering to:

- The maintenance of a safe following distance to a leading vehicle when traveling on a road.
- The determination of and adherence to the arrival-based precedence ordering at intersections.
- The safe merging into or across moving traffic at intersections such as at a typical T-intersection.
- Passing maneuvers on multi-lane roads, either at speed into another lane in the same direction or from a stop and into an oncoming travel lane if the system encounters a disabled vehicle.

First and foremost, the Behavioral Executive had to ensure that the system's mission was successfully completed, even if this meant temporarily deviating from traffic flow and social norms. When faced with a deadlock at an intersection or a roadblock, the Urban Challenge rules allowed for a few possible outcomes, such as turning around and rerouting, but there were plenty more that could happen. As a result, the only guidance given to vehicles in these situations was to perform qualitatively "safe" actions, with judges having sole discretion to penalize those that performed "unsafely."

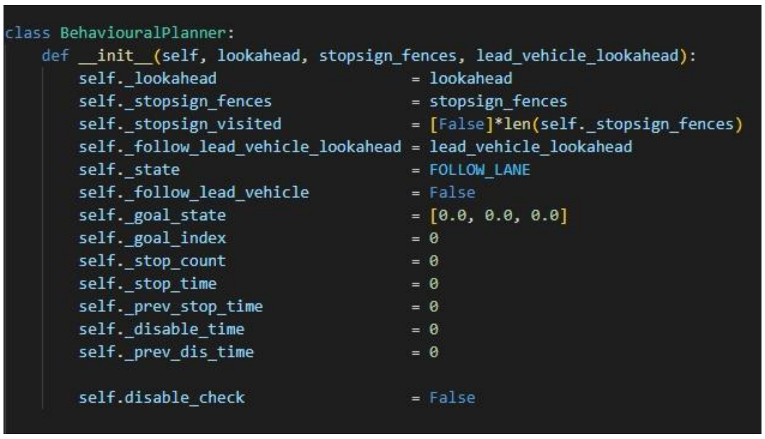
4.4 Code Snippets for the Behavioral Planning:

4.4.1 Initializing states for the Finite State Machine:



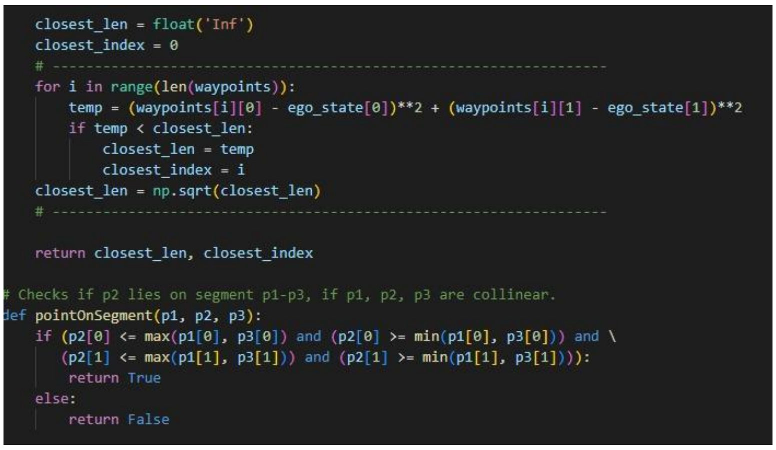
```
#TURN_RIGHT = 1
#TURN_LEFT = 2
#LANE_RIGHT = 3
#LANE_LEFT = 4
#Alternate_lane_finder_exceution
FOLLOW_LANE = 0
DECELERATE_TO_STOP = 1
STAY_STOPPED = 2
PARK   = 3
Alternate_lane_finder_execution = 4

# Stop speed threshold
STOP_THRESHOLD = 0.02
# Number of cycles before moving from stop sign.
STOP_COUNTS = 100
```

4.4.2 Initializing states for the Finite State Machine:



```
class BehaviouralPlanner:
    def __init__(self, lookahead, stopsign_fences, lead_vehicle_lookahead):
        self._lookahead                     = lookahead
        self._stopsign_fences               = stopsign_fences
        self._stopsign_visited              = [False]*len(self._stopsign_fences)
        self._follow_lead_vehicle_lookahead = lead_vehicle_lookahead
        self._state                         = FOLLOW_LANE
        self._follow_lead_vehicle           = False
        self._goal_state                    = [0.0, 0.0, 0.0]
        self._goal_index                    = 0
        self._stop_count                    = 0
        self._stop_time                     = 0
        self._prev_stop_time                = 0
        self._disable_time                  = 0
        self._prev_dis_time                 = 0

        self.disable_check                  = False
```

4.4.3 Ego State and Next point of interest(waypoint) examination:



```python
    closest_len = float('Inf')
    closest_index = 0
    # -----------------------------------------------------------------
    for i in range(len(waypoints)):
        temp = (waypoints[i][0] - ego_state[0])**2 + (waypoints[i][1] - ego_state[1])**2
        if temp < closest_len:
            closest_len = temp
            closest_index = i
    closest_len = np.sqrt(closest_len)
    # -----------------------------------------------------------------

    return closest_len, closest_index

# Checks if p2 lies on segment p1-p3, if p1, p2, p3 are collinear.
def pointOnSegment(p1, p2, p3):
    if (p2[0] <= max(p1[0], p3[0]) and (p2[0] >= min(p1[0], p3[0])) and \
       (p2[1] <= max(p1[1], p3[1])) and (p2[1] >= min(p1[1], p3[1]))):
        return True
    else:
        return False
```

CHAPTER - 5. LOCAL PATH PLANNING

5.1 Local Planning:

If impediments and changes in the environment are dynamic or unknown, a planning method called Local Path Planning can adapt to them. Dynamic path planning adapts to the continuously shifting environment whereas unknown path planning anticipates and avoids potential hazards. By "local path planning," we mean using sensory-based information to avoid potential hazards while yet maintaining a vehicle's safety while driving. In local path planning, a robot usually takes the shortest route possible from its starting point to its destination. By deviating from its intended path, the robot may better gauge the distance between its current location and a target point, as well as the exact placement of any impediments in its path.
It is also known that short paths can be calculated in real time by local path planners, who update the vehicle's heading if necessary. It's possible to apply the local path's essence to a larger group of path planners who keep tabs on the scenario's changes and request an updated planned path when the current one is no longer feasible. In addition, by storing the changes that the vehicle detects, this technique aids in the collection of additional environmental data.

The motion planners of autonomous vehicles are responsible for determining the vehicle's path through its environment while avoiding any obstacles, and this task is critical. The vehicle's low-level controller receives a trajectory and a velocity profile from the local path planner to perform this task.

5.1.1 Trajectory Generation:

For the vehicle to follow the path, a set of ordered loci (in the joint space or in the operational space) must be identified. The path serves as a straightforward representation of motion in terms of geometry. In most cases, the path is designed to avoid obstacles, traverse a complex maze, and so on. An object's trajectory is the sum of its individual velocities and accelerations. As a result, the task of designing a trajectory is significantly simplified. Local designers and planners determine and create the course (divide and conquer methodology). Trajectories cover different parts of the path. There are times when a single trajectory design must consider only neighboring trajectories on the path, as it is often necessary for pieces of trajectories to join smoothly. The trajectory generator takes as its input the path provided by path planning. A class of polynomial functions is used by the trajectory generator to "approximate" the desired path waypoints, and a time-based control sequence is generated to move the vehicle platform from its starting point to the destination. Motion control system inputs are generated by trajectory generation, which ensures that the planned trajectory is executed smoothly. Waypoints are commonly used to represent the sequence of points along a path. The process of creating a path between two or more points is known as "trajectory generation."

The figure 1 shows the scenario in which our vehicle must move from point A (starting) to point B (final). There are four obstacles in between starting and final position. At start our waypoint generation algorithm will generate waypoints which are P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8. The trajectory generator will create trajectory between two consecutive waypoints. The first trajectory will be created between points P_1 and P_2 and the vehicle will follow this trajectory and then its neighboring trajectory and this will continue as the vehicle reaches its final position. meanwhile while creating the trajectory our vehicle makes notice of the obstacle Infront of, it simultaneously making a collision avoidance system for itself. Hence the waypoints are generated while keeping in view the collision avoidance property. for instance, if our vehicle is at point P_1 and it must go to point P_8 it will be following proper route like first it will be going to point P_2 and then P_3 and at last to point P_8.

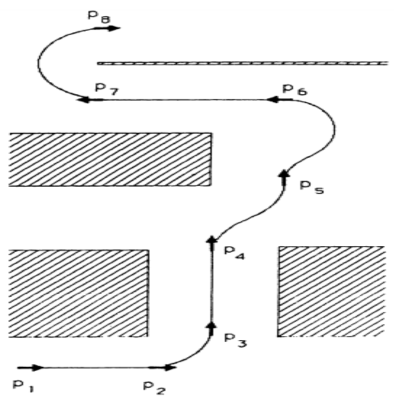


Fig. 14 Optimal Path Finding

5.1.2 Trajectory Generation Illustration:

Real-time trajectory planning is required. At the same time, the vehicle must ensure that no accidents occur. As a result, the trajectory model must allow for quick computation and simple communication across the various components, such as the path planner, trajectory generator, collision checker, and controller. The model must allow for the local deformation of a path or a trajectory in order to avoid having to re-plan the entire trajectory. The controller must also be able to switch from a predetermined path to a new one at any time. As a starting point, the trajectory generator uses the global planner's route as an input. Our surroundings are distinct from those of the entire world. Objects in motion, such as cars and people, make up the local environment. This means that the global path cannot be used for the vehicle's trajectory. Collision-free trajectory with velocity generation can be achieved by detecting and responding to dynamic objects. Sensors and real-time data from the surroundings can be used to generate a new trajectory for the low-level controller. It's all laid out in the diagram.

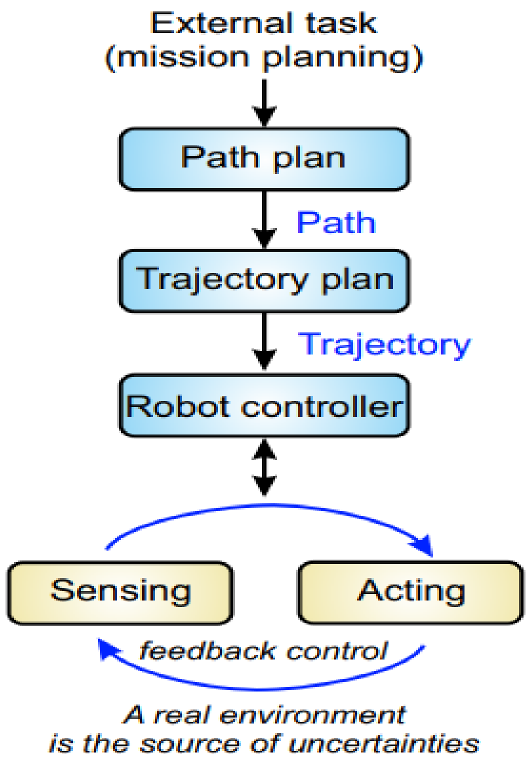


Fig. 15 Role of Trajectory Generation in Path Planning

5.1.2 Path Optimization:

Real-time performance of most lattice planners depends on proper discretization. When it comes to planning, this has a negative impact on the final outcome. In order to make autonomous vehicles perform as well as human drivers, it is critical that the trajectory be as good as possible, which is why we are working to improve performance by post-optimizing the trajectory.

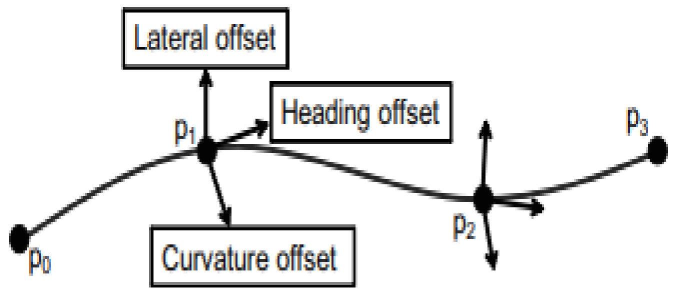


Fig. 16 Path optimization process. Relaxing lateral, heading and curvature offset

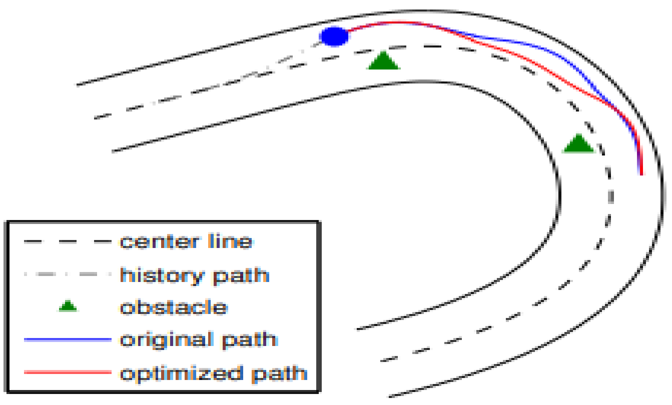


Fig. 17 Result for path optimization

The simplest strategy is to simultaneously improve the trajectory's path and speed. However, in a real-time application, this would be prohibitively slow. Obsolete optimization algorithms, which are usually O in computational complexity, have a computation cost of O (opt (M +N)). M, N, and opt () are parameters of the optimization algorithm (N2). Because of this, finding the global optimal solution becomes increasingly difficult for the optimizer as the dimensions grow. Consequently, we propose an iterative route optimization mechanism. It has an O(opt(M)) + O(opt(N)) computational complexity. We can get to the real optimum in a few iterations using this mechanism if the trajectory found in the planning phase is close enough.

1) **Path optimization:** There is a limit to how good a route can get using path discretization. Sampled endpoints have fixed lateral offsets, headings and curvatures that are related to the center line. As a result, loosening the restrictions (as shown in Fig. 5) and creating new paths between the new endpoints improves the trajectory's quality. A non-derivative optimization algorithm, the Simplex algorithm [17], is used for path optimization because the gradient for cost versus lateral offset and heading is extremely difficult to compute. As shown in Fig. 6, the red line shows a smoother path following path optimization as shown in figure 17
2) **Speed optimization:** The optimality of the speed profile is also restricted by the discretization of speed and the restriction of acceleration at endpoints. The parameters of the nodes in the speed profile are optimized in a manner similar to that used in path optimization. We optimize the speed and acceleration values of the nodes that generate the current lowest-cost speed profile. Connecting new nodes ensures that the speed profile does not change. As a result, the gradient method cannot be used in this optimization because the time of each point on the trajectory will change as a result of speed changes at nodes. That's why speed optimizations employ the non-derivative Simplex algorithm. Fig. 7 depicts the improved performance because of the speed optimization. Speed changes are more gradual and the amplitude of the speed at the endpoints is no longer discretized because of the relaxed acceleration constraints.

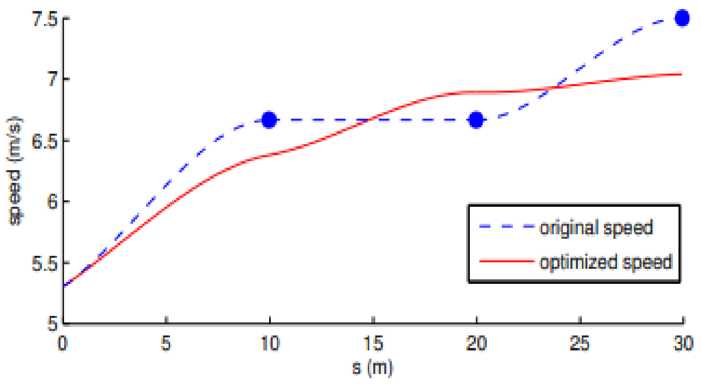


Fig. 18 Result for speed optimization

5.2 Local Path Planning Algorithm:

An accurate velocity profile and free collision trajectory must be obtained for the local path planning. Safety, comfort, and vehicle dynamics were all considered in both cases. For starters, there is a limit on how fast the vehicle can go at any given time. The following is a breakdown of Local Path Planning:

- Maximum curvature velocity |Vlat|: Driving around a curve, one's own vehicle can reach its maximum speed at this point. Maximum lateral acceleration aymax and road curvature are the parameters that define this area.

$$|\vec{v}_{lat}| = \frac{|a_{y_{max}}|}{\kappa}$$

- Maximum regulatory velocity |Vreg|: The road's regulatory elements, such as stop and yield signs, traffic lights, and speed limit signals, all play a role in keeping it safe. Information from the MP's second level was used to generate this report.
- Vehicle leader’s maximum velocity |Vlead|: An ego-top vehicle's speed is determined by how far ahead of it in time it is in relation to the preceding vehicle. This speed is calculated by integrating a desired longitudinal acceleration.
- As described in Equation 2, the maximum allowed velocity is defined as the minimum of a set that includes the prior calculated velocities.

$$|\vec{v}_{max}| = \min(|\vec{v}_{lat}|, |\vec{v}_{reg}|, |\vec{v}_{lead}|)$$

Thus, the first constraint is given by Equation 3:

$$|\vec{v}| \leq |\vec{v}_{max}|$$

Constraint No. 3 deals with the maximum rate at which a steering angle can change. Equations (2) and show how it can be divided into two types of relationships (3). Both of these are based on how fast the steering angle can change in relation to the given velocity |V| and how much yaw acceleration can be applied to it.

$$\begin{aligned} |\dot{\delta}| &\leq |\dot{\delta}_{max}| \\ |\tan(\delta(t)) - \tan(\delta(t-dt))| &\leq \frac{|\ddot{\psi}_{max}|(l_f + l_r)dt}{|\vec{v}|} \end{aligned}$$

Equations (4) and (5) provide the relationships that limit the lateral and longitudinal acceleration to a maximum value that also limits the wheels into the linear operating zone (5).

$$\begin{aligned} \sqrt{a_x^2 + a_y^2} &\leq |\vec{a}_{max}| \\ a_y &= |\vec{v}|\dot{\psi} \end{aligned}$$

Local path planners are expected to generate collision-free paths. Collision circles are used to accomplish this task, as shown in Figure 5, by setting at least three circles whose centers are located along the x-axis of the vehicle. In the event of a collision, the vehicle's position within the circle is considered to have been affected by the obstacle. As a result, the final constraint is about how far the trajectory can travel, which is given by Equation (6), where j is the index that corresponds to the given center circle, Rj is its radius, I is the index in m of given obstacles, and Djobs I is the distance of the given i-th obstacle from the given j-th circle's center. An obstacle is represented as a cuboid with its dimensions determined by its width, while the driving zone is also considered an obstruction and is represented as points on its border.

$$\sum_{i=1}^{m}(R_j - D^{j}_{obs,i}) \leq 0$$
$$j \epsilon \{r, f, c\}$$

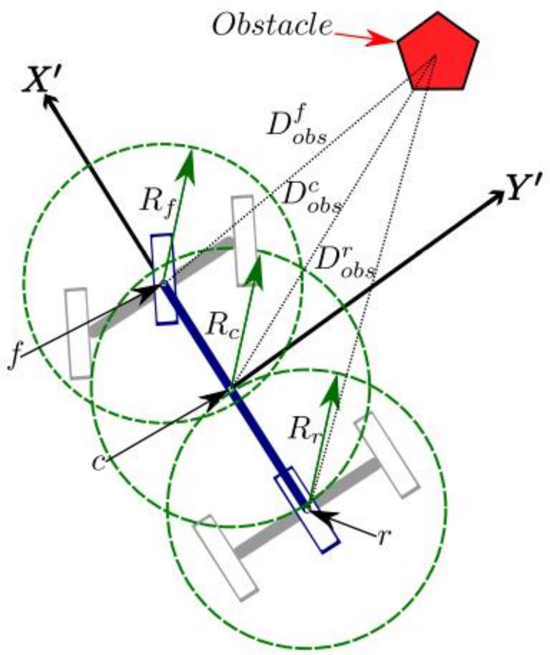


Fig. 19 Set of secure radius collision-free checker. The center of circles is placed along the X axis

It is imperative that a trajectory and velocity profile are provided to the low-level controller by the local route planner. This algorithm calculates a set of forward points, together with their respective steering angles and restrictions, based on the vehicle's present condition, impediments, and drivable limits. As a result, each point in the velocity profile is represented by a set of velocity values. It is necessary to solve the vehicle's dynamic model from the present to a time gap tw in order to calculate both.

The Local Path Planning Algorithm is explained in the figure below:

Algorithm 1: Local path planning: trajectory and velocity profile generation.

```
repeat
    Get from sensors: Vehicle state $\vec{x}$, surrounders actors, and lane limits at current
      time $t_o$
    Get $|\vec{v}_{max}|$
    $acceleration\_phase = 0$
    Calculate $\delta_{att}(t + \Delta t)$
    Calculate $\dot{\delta}$ with $\delta_{att}(t + \Delta t)$ and $x_3(t_o)$
    if $\dot{\delta}$ is inside constraints then
        Set $u_1$ as $\dot{\delta}$
    else
        Set $u_1$ as the maximun values of $\dot{\delta}$
    if $acceleration\_phase == 0$ then
        Calculate $a_x(t + \Delta t)$ according to $(|\vec{v}_{max}|^2 - x_4(t_o)^2)/(2s)$
        if $a_x(t + \Delta t)$ is inside constraints then
            Set $u_2$ as $a_x(t + \Delta t)$
        else
            if $a_x(t + \Delta t) < 0$ then
                Set $u_2$ as the maximun values of $a_x$ using $-\sqrt{|\vec{a}_{max}|^2 - a_y^2}$
                $acceleration\_phase = 1$
            else
                Set $u_2$ as the maximun values of $a_x$ using $\sqrt{|\vec{a}_{max}|^2 - a_y^2}$
    else
        if $acceleration\_phase == 1$ then
            Set $u_2$ as the maximum values of $a_x$ using $-\sqrt{|\vec{a}_{max}|^2 - a_y^2}$
    $t_s = t_o + t_w; t = t_o$
    repeat
        Calculate new vehicle state $\vec{x}(t + \Delta t)$ with $u_1$ and $u_2$ and save it
        $t = t + \Delta t$
        Calculate: $\delta_{att}(t + \Delta t)$; $\dot{\delta}$ with $\delta_{att}(t + \Delta t)$ and $x_3(t)$
        if $\dot{\delta}$ is inside contraints then
            Set $u_1$ as $\dot{\delta}$
        else
            Set $u_1$ as the maximun values of $\dot{\delta}$
    until $t_s \leq t$;
    For each state analyze if there is a Collision
    if Not Collision then
        Return to low controller the velocity profile and trajectory
    else
```

```
38          if $aceleration\_phase == 0$ then
39              $aceleration\_phase = 1$
40              Go to line (11)
41          else
42              $aceleration\_phase = 2$
43              Execute Algorithm 2
44              Return to low controller the velocity profile and trajectory
45  until Reach the last waypoint;
```

5.2.1 Local Path Planner Flowchart:

The flowchart of the overall implementation of local planner is given in below figure:

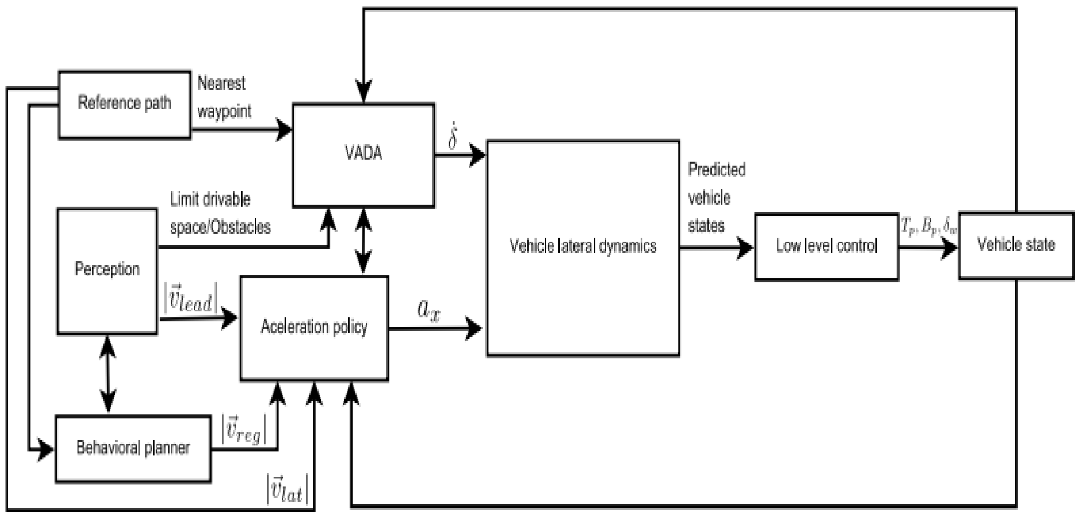


Fig. 20 Basic Flowchart of Local Path Planner

5.3 Local Path Planning Simulation on UNREAL Engine AirSim:

The ego-vehicle has to adjust its speed and trajectory to take into account the reference global path, the preceding vehicle's velocity, and the maximum curvature velocity in order to avoid collisions. As shown in Figures 7 and 8, the current scenario is shown at various points in time while the velocity profile and steering angle generated over the course of the scenario are shown in Figure 8.

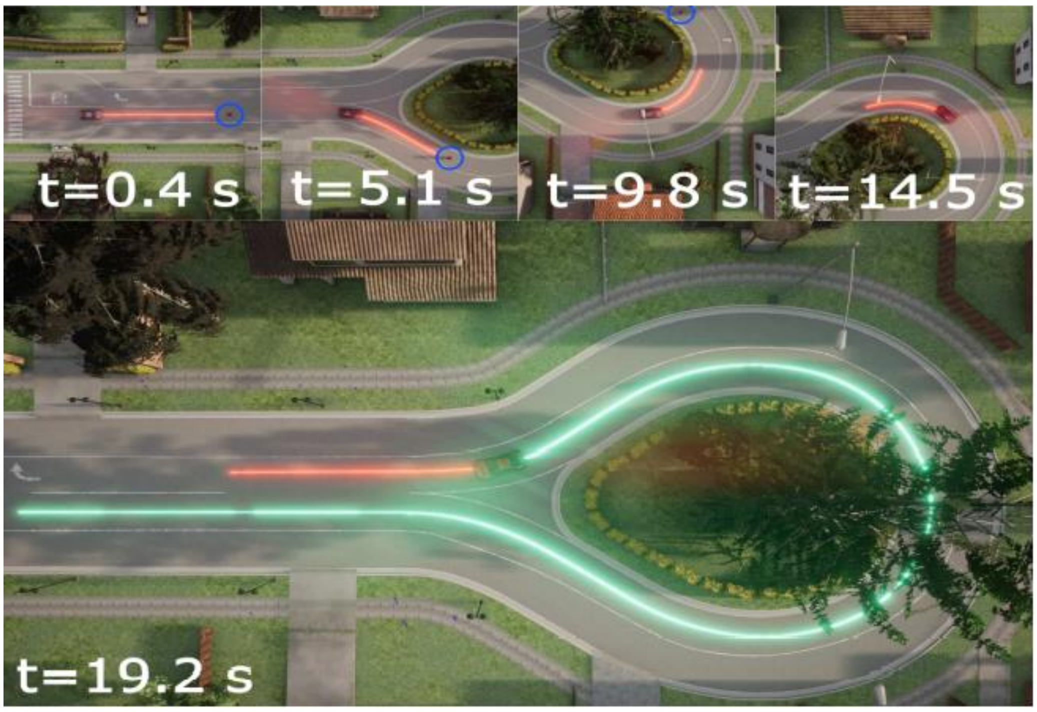


Fig. 21 At various points in time, the trajectory was generated. The effective path taken by the ego-vehicle is shown in green, while the local path planning is shown in red. A blue circle indicates the location of the preceding vehicle (the bicyclist).

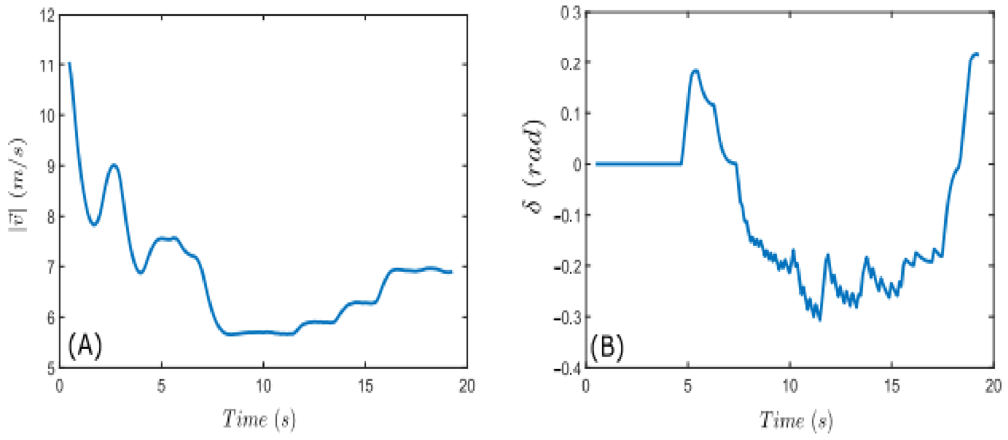


Fig. 22 (A) Velocity profile along the path. (B) Steering angle calculated by attractor

CHAPTER - 6. APPROACHES OF TRAJECTORY GENERATION

6.1 Attractive Dynamic Approach:

This approach is best suited for alternate lane finder execution, as it can be implemented with minimal control parameters and minimal planning/perception modules. This can also be called Dynamic Path Planning. The Main Focus of this approach is as follows:

- Multiple lane road with obstacles in the current lane.
- Moment at which vehicle avoids larger obstacle.
- Trajectory of the vehicle passing through the road.
- Speed Curve
- Acceleration Curve

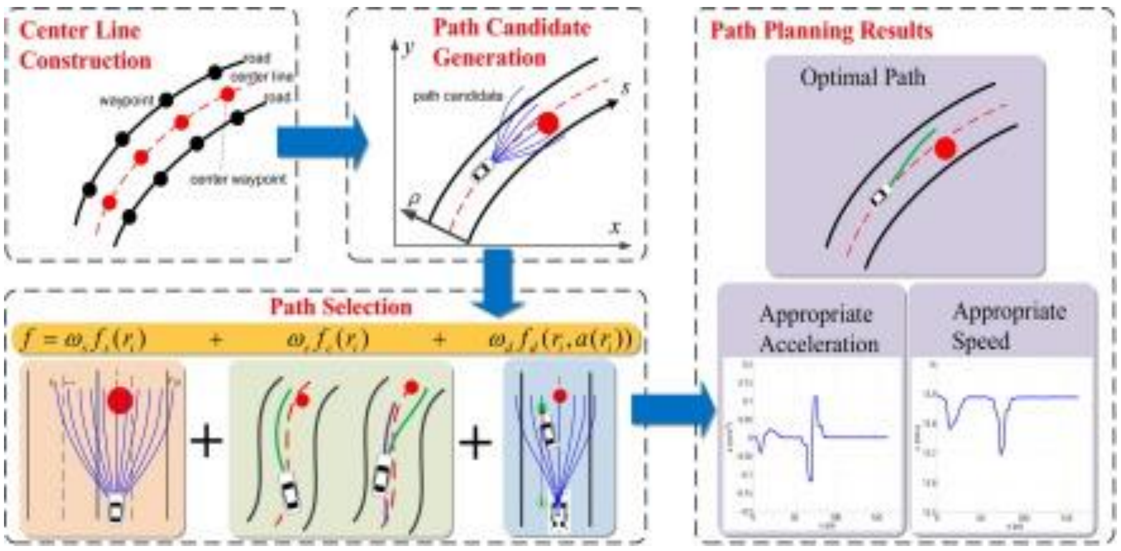


Fig. 23 Overview of ADA

6.1.1 Evading Static Obstacles:

The ego-vehicle had to perform an evasive maneuver because it was driving at a dangerous speed. Path planner algorithm detected a collision in the red trajectory in the first phase, as shown in Figure 9A. The algorithm determines that the ego-vehicle was driving at a speed that leads to a crash regardless of whether the speed is adjusted with the maximum comfort deceleration, so the algorithm moves to a state where the Repeller is taken into consideration (s). A free-collision path with a change in velocity within comfort parameters was generated as shown in Figure 9E and a profile of steering angle is shown in Figure 9D. Due to the dynamic nature of the situation, the values of the Repeller and the attractor(s) will naturally change over time. New trajectory paths are shown in Figure 9B after adjusting the vehicle's velocity and steering angle. In Figure 9C, the final trajectory generated is marked in green so that it can be compared to the initial trajectories.

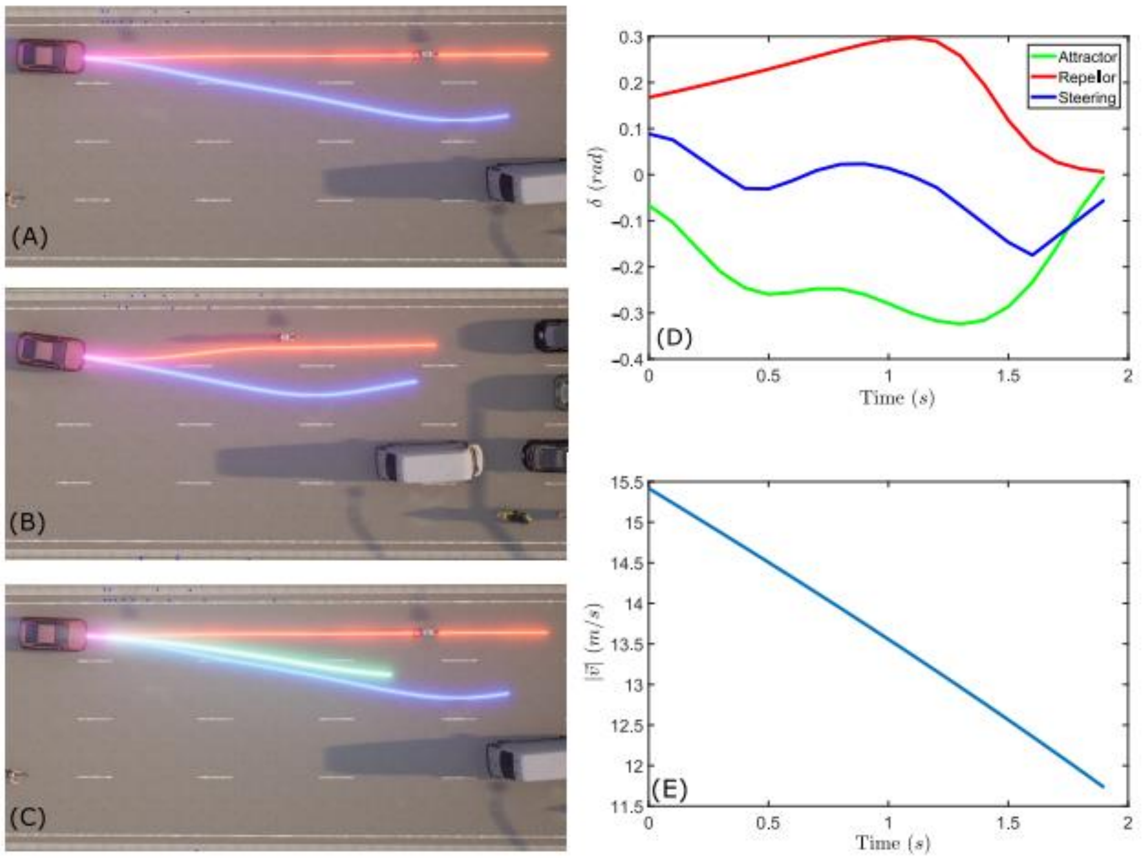


Fig. 24 Performing Different Maneuvers and Generating Graphs

When the angle obs converges to max, the generated trajectories converge toward the reference path because the Repeller becomes weaker. To avoid a collision, the current method was found to be able to design the trajectory and speed profile of the vehicle. It was possible to avoid a collision in this situation using the first trajectory calculated using the repellents, but it is true that this is not the case when the obstacle is directly in front of the ego vehicle (obs 0), as the repellent equation states. This is why collision analysis and defining the drivable zone are critical components of the strategy. In the following scenario, the proposed solution is shown to get around this problem.

6.1.2 Evading the Dynamic Obstacles:

A neighboring car merges into the ego-current vehicle's lane while there are other vehicles nearby that could cause a collision, so it must plan a safe route around them. It was determined that two crashes were likely to occur along this red path, one with the preceding car and the other that would merge into the lane of the ego-vehicle.

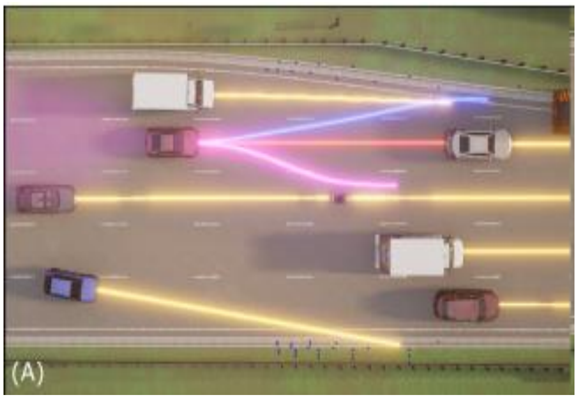


Fig. 25 Merging in alternate lane

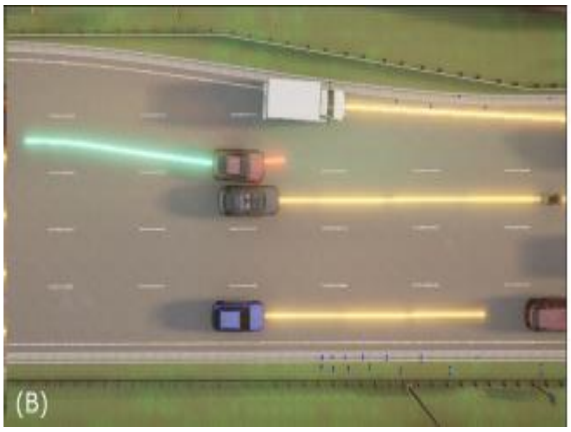


Fig. 26 Final Trajectory

Blue points that delimit the driving area have been added to resellers by the local planner algorithm because it determines that a collision will be formed with the previously specified cars. To avoid a collision, the system then generates a new path that includes all resellers. This new path stays within the drivable zone because the algorithm reduces the speed. Although a vehicle comes within a few feet of the ego-vehicle in Figure 10B, the local route planner manages to come up with a free-collision plan.

6.2 Implementation of ADA:

Accumulated Graphs via our Implemented ADA:

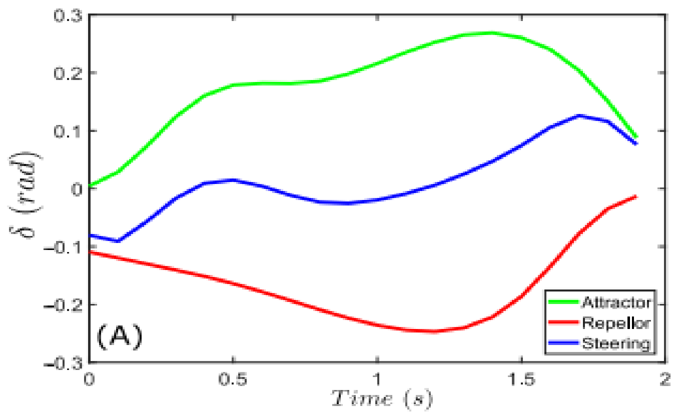


Fig. 27 (A) Steering angle calculated for blue trajectory

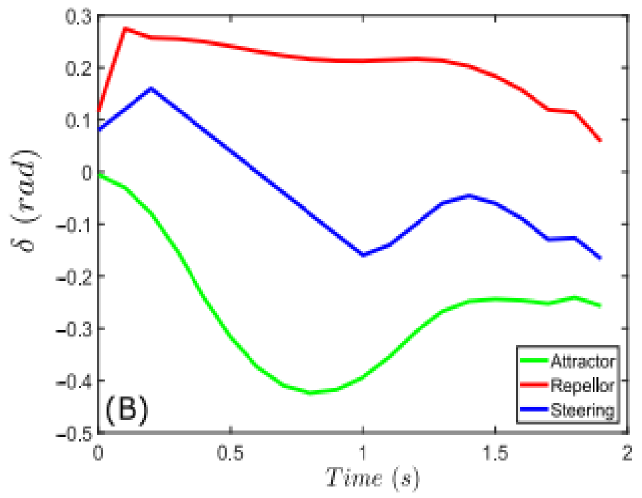


Fig. 27 (B) Steering angle calculated for magenta trajectory

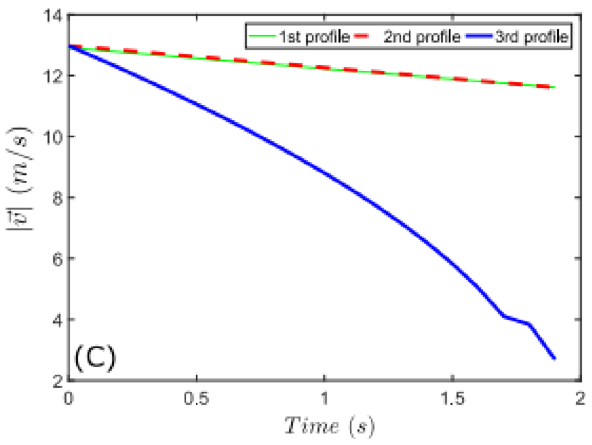


Fig. 27 (C) Velocity profile for each generated trajectory

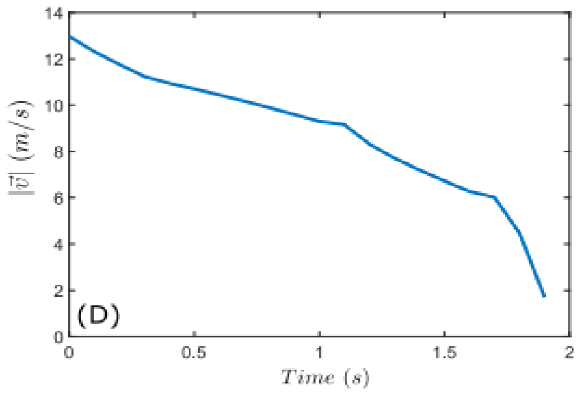


Fig. 27 (D) Resulting velocity profile for the first to final frame

6.3 FRENET OPTIMAL TRAJECTORY:

A robot's trajectory can be planned in a variety of ways. A trajectory can be thought of as a collection of state vectors x that are arranged in time.

6.3.1 Algorithm Implementation:

1. **Determine the trajectory start state:**

[x1, x2, θ, κ, v, a] (0)[x1,x2,θ,κ,v,a](0) Evaluation of the preceding trajectory at the anticipated start state yields the trajectory start state (low-level-stabilization). At startup and after startup, the current car position is used instead of the vehicle's current location (high-level-stabilization).

2. **Selection of the lateral mode:**

Latitudinally planned motion is either time-based (d(t)d(t)) or length-based (d(s)d(s)) on the basis of velocity v. The longitudinal start point s(0) can be calculated by projecting the initial state onto the reference curve. The frenet transformation can be used to determine the frenet state vector [s,s,s,d,d′,d″](0). The values of [d,d](0)[d,d](0) must be determined for use in the time-based lateral planning mode.

3. **Generating the lateral and longitudinal trajectories:**

In the frenet space, trajectories are generated for both lateral (mode dependent) and longitudinal motion (velocity keeping, vehicle following/distance keeping). For now, it is possible to ignore paths with large lateral accelerations compared to the reference path to speed up computation.

4. **Combining lateral and longitudinal trajectories:**

With J(d(t)s(t))=Jd[(t)]+ksJ(s(t))J(d(t),s)), each longitudinal trajectory is merged with each lateral trajectory and then translated back to world coordinates using the reference path for all active longitudinal modes. The curvature and acceleration of the trajectories are evaluated point-by-point to see if they comply with physical driving constraints. This results in a collection of world coordinates-based movements that may be drivable in a given mode.

5. **Static and dynamic collision check:**

If static and dynamic collisions can be avoided, each trajectory set is evaluated with rising overall costs. The most cost-effective path is then chosen.

6. **Longitudinal mode alternation:**

The trajectory with the greatest deceleration or the trajectory with the least acceleration is picked and passed to the controller using the sign-based jerk a (0)a (0) (in the beginning).

As opposed to the usual (x,y) Cartesian Coordinates, "Frenet Coordinates" provide a more natural representation of a point on a route.

We utilise the variables s and d to indicate a vehicle's position on the road or a reference path when using Frenet coordinates. There are two different types of road positions that can be measured using these two coordinates: longitudinal (sometimes called "longitudinal" displacement) and lateral (also known as "lateral" displacement). As may be seen in the following graphic:

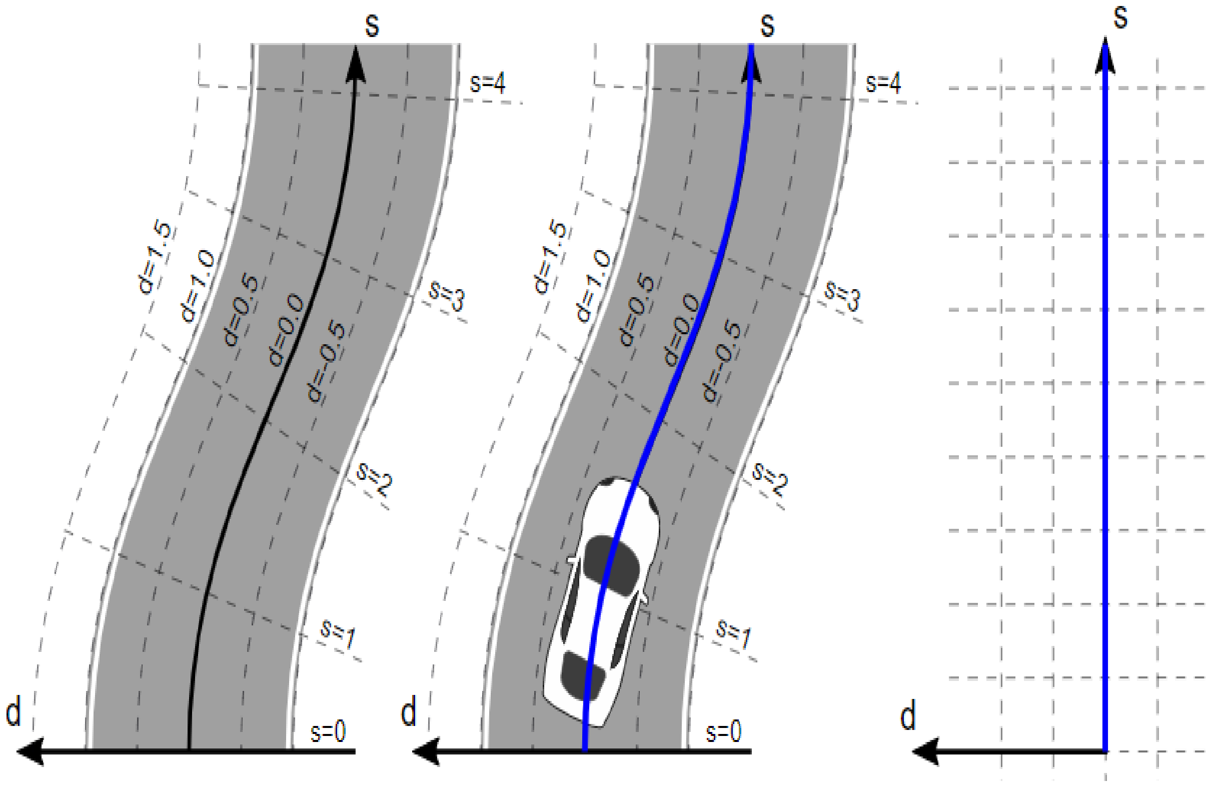


Fig. 28 Representation of Vehicle on road by Frenet coordinate systems

The cartesian coordinates can be simply converted to frenet coordinates in this scenario. It is for this reason that our coordinate frame can plan out a course of action along each of these paths: With the frenet optimal trajectory, you can get the best possible path and the greatest possible adjustment of coordinates all in one shot.

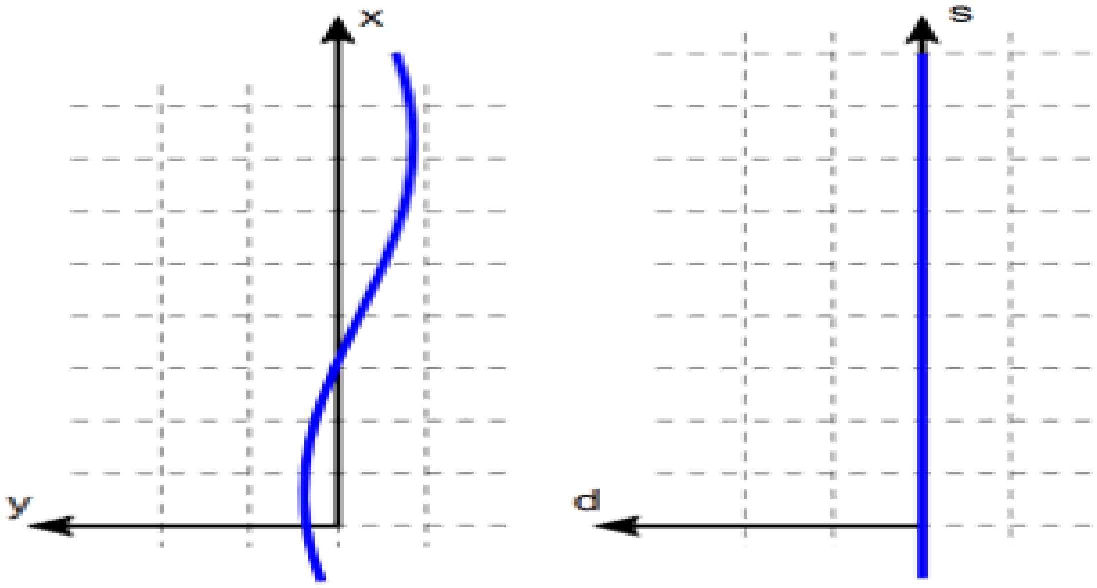
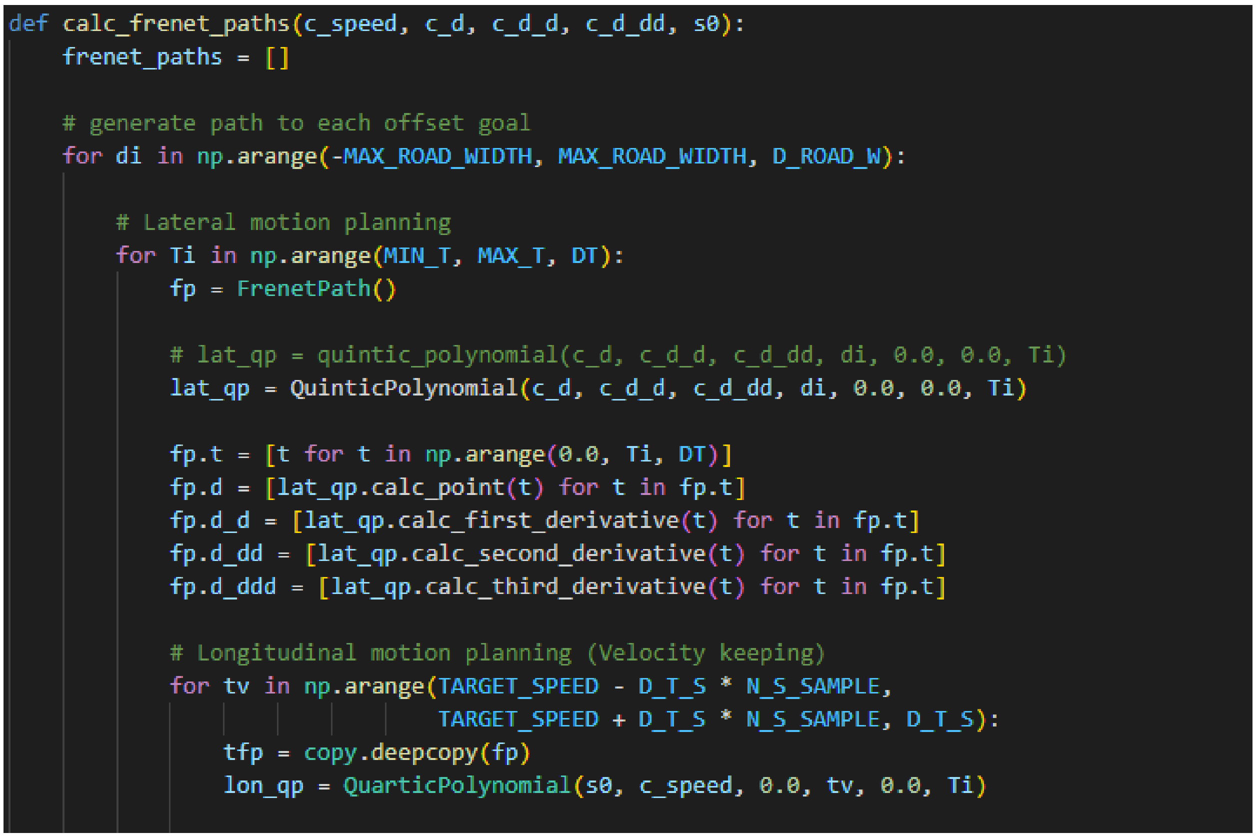
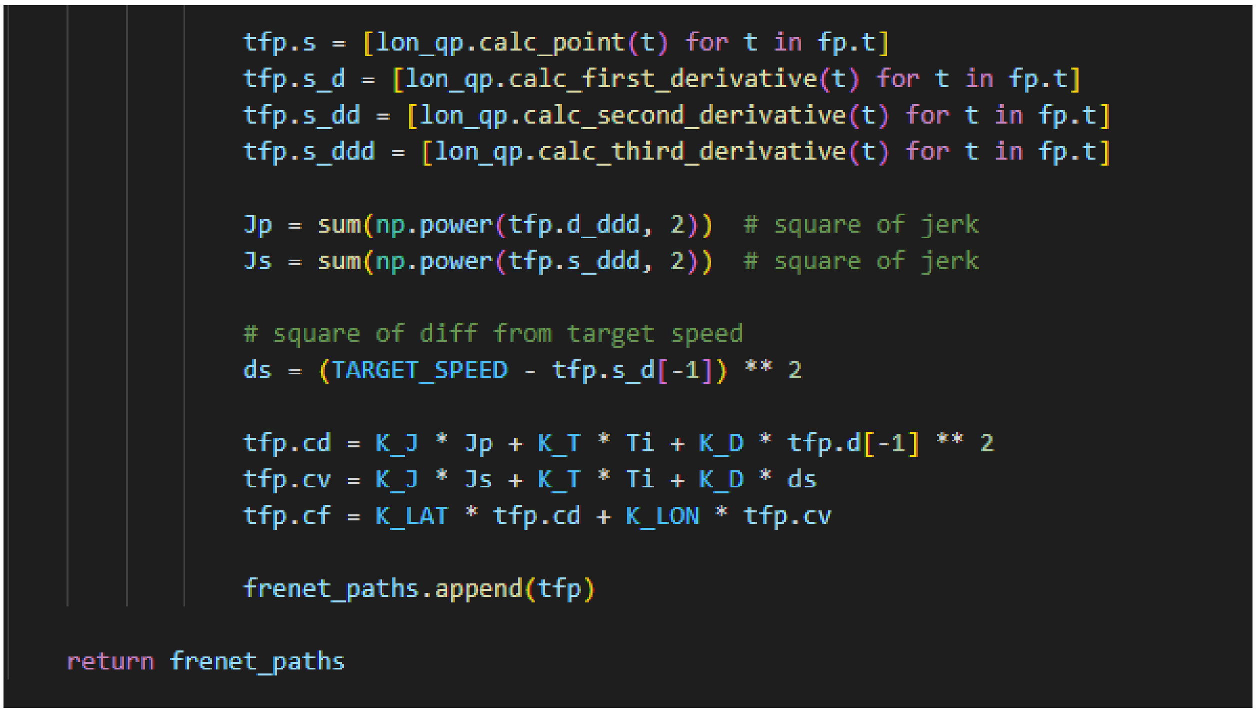


Fig. 29 Comparison of Cartesian coordinate and Frenet coordinate system

6.3.2 Code Snippets for the Frenet Optimal Trajectory:

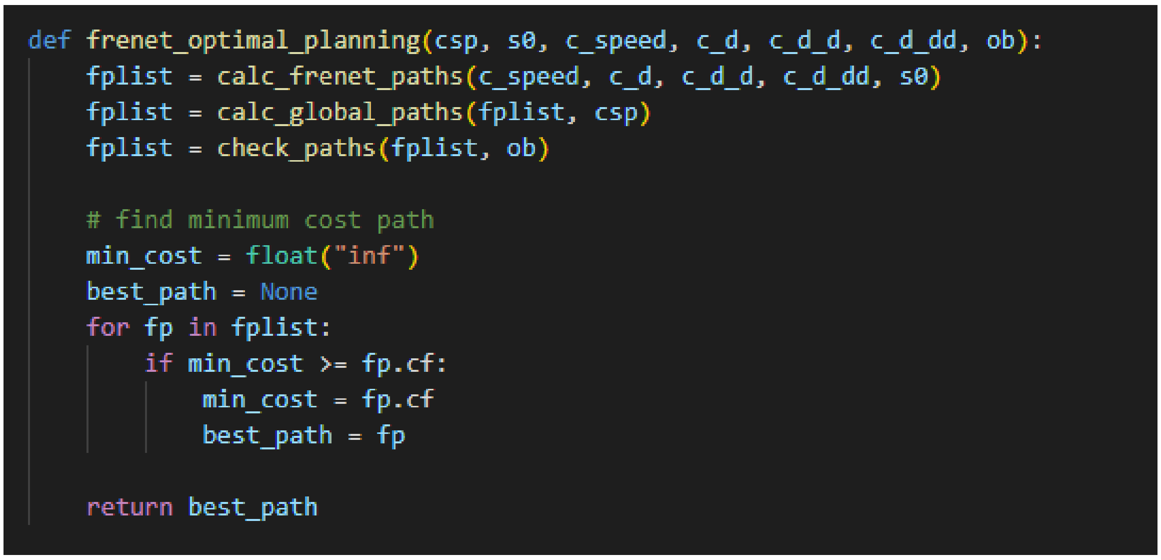


```
def calc_frenet_paths(c_speed, c_d, c_d_d, c_d_dd, s0):
    frenet_paths = []

    # generate path to each offset goal
    for di in np.arange(-MAX_ROAD_WIDTH, MAX_ROAD_WIDTH, D_ROAD_W):

        # Lateral motion planning
        for Ti in np.arange(MIN_T, MAX_T, DT):
            fp = FrenetPath()

            # lat_qp = quintic_polynomial(c_d, c_d_d, c_d_dd, di, 0.0, 0.0, Ti)
            lat_qp = QuinticPolynomial(c_d, c_d_d, c_d_dd, di, 0.0, 0.0, Ti)

            fp.t = [t for t in np.arange(0.0, Ti, DT)]
            fp.d = [lat_qp.calc_point(t) for t in fp.t]
            fp.d_d = [lat_qp.calc_first_derivative(t) for t in fp.t]
            fp.d_dd = [lat_qp.calc_second_derivative(t) for t in fp.t]
            fp.d_ddd = [lat_qp.calc_third_derivative(t) for t in fp.t]

            # Longitudinal motion planning (Velocity keeping)
            for tv in np.arange(TARGET_SPEED - D_T_S * N_S_SAMPLE,
                                TARGET_SPEED + D_T_S * N_S_SAMPLE, D_T_S):
                tfp = copy.deepcopy(fp)
                lon_qp = QuarticPolynomial(s0, c_speed, 0.0, tv, 0.0, Ti)
```



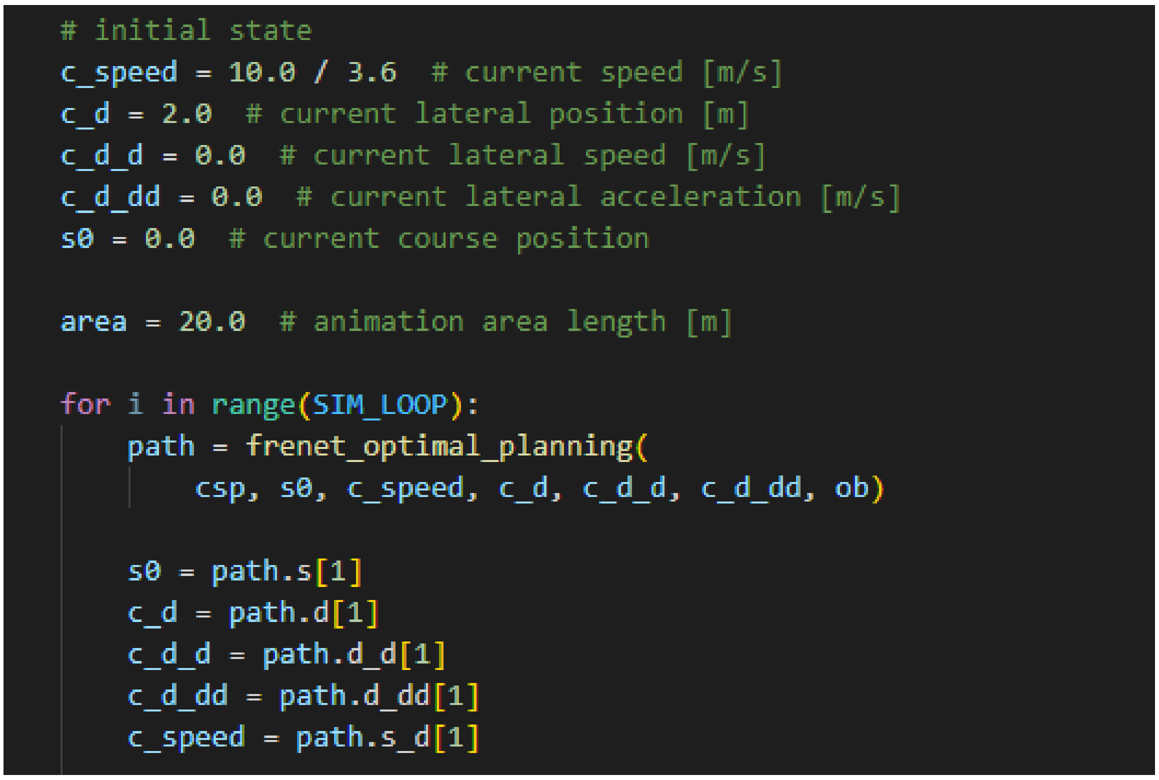
```
                tfp.s = [lon_qp.calc_point(t) for t in fp.t]
                tfp.s_d = [lon_qp.calc_first_derivative(t) for t in fp.t]
                tfp.s_dd = [lon_qp.calc_second_derivative(t) for t in fp.t]
                tfp.s_ddd = [lon_qp.calc_third_derivative(t) for t in fp.t]

                Jp = sum(np.power(tfp.d_ddd, 2))  # square of jerk
                Js = sum(np.power(tfp.s_ddd, 2))  # square of jerk

                # square of diff from target speed
                ds = (TARGET_SPEED - tfp.s_d[-1]) ** 2

                tfp.cd = K_J * Jp + K_T * Ti + K_D * tfp.d[-1] ** 2
                tfp.cv = K_J * Js + K_T * Ti + K_D * ds
                tfp.cf = K_LAT * tfp.cd + K_LON * tfp.cv

                frenet_paths.append(tfp)

    return frenet_paths
```

6.3.3 Function for FRENET Optimal Planning:



```
def frenet_optimal_planning(csp, s0, c_speed, c_d, c_d_d, c_d_dd, ob):
    fplist = calc_frenet_paths(c_speed, c_d, c_d_d, c_d_dd, s0)
    fplist = calc_global_paths(fplist, csp)
    fplist = check_paths(fplist, ob)

    # find minimum cost path
    min_cost = float("inf")
    best_path = None
    for fp in fplist:
        if min_cost >= fp.cf:
            min_cost = fp.cf
            best_path = fp

    return best_path
```

6.3.4 Estimation the next possible state in Maneuver:



```
# initial state
c_speed = 10.0 / 3.6  # current speed [m/s]
c_d = 2.0  # current lateral position [m]
c_d_d = 0.0  # current lateral speed [m/s]
c_d_dd = 0.0  # current lateral acceleration [m/s]
s0 = 0.0  # current course position

area = 20.0  # animation area length [m]

for i in range(SIM_LOOP):
    path = frenet_optimal_planning(
        csp, s0, c_speed, c_d, c_d_d, c_d_dd, ob)

    s0 = path.s[1]
    c_d = path.d[1]
    c_d_d = path.d_d[1]
    c_d_dd = path.d_dd[1]
    c_speed = path.s_d[1]
```

6.3.5 Result:

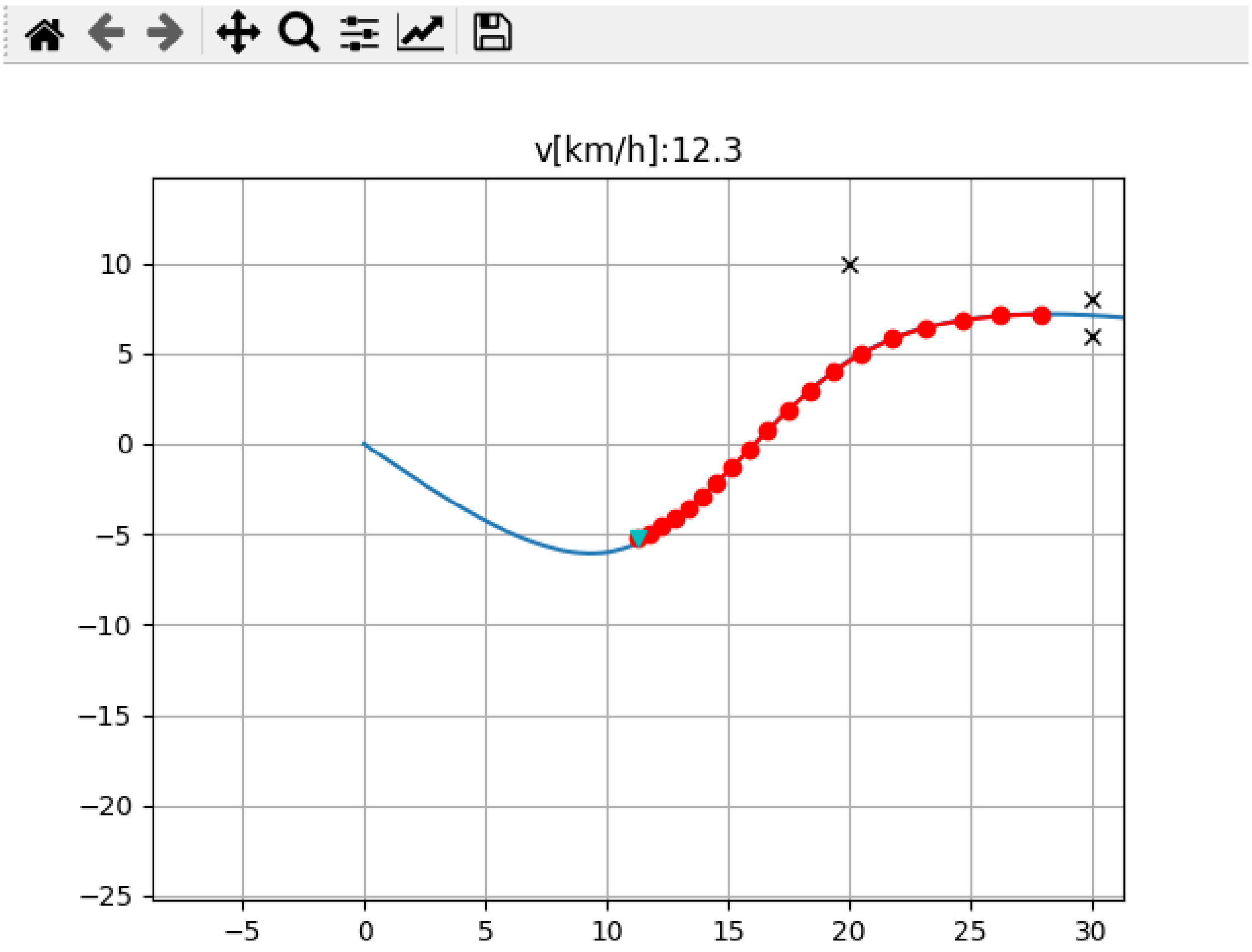


Fig. 30 The Result will be the Sequence of this trajectory and red waypoints on the s frame of the path indicates the optimal reference path planning.

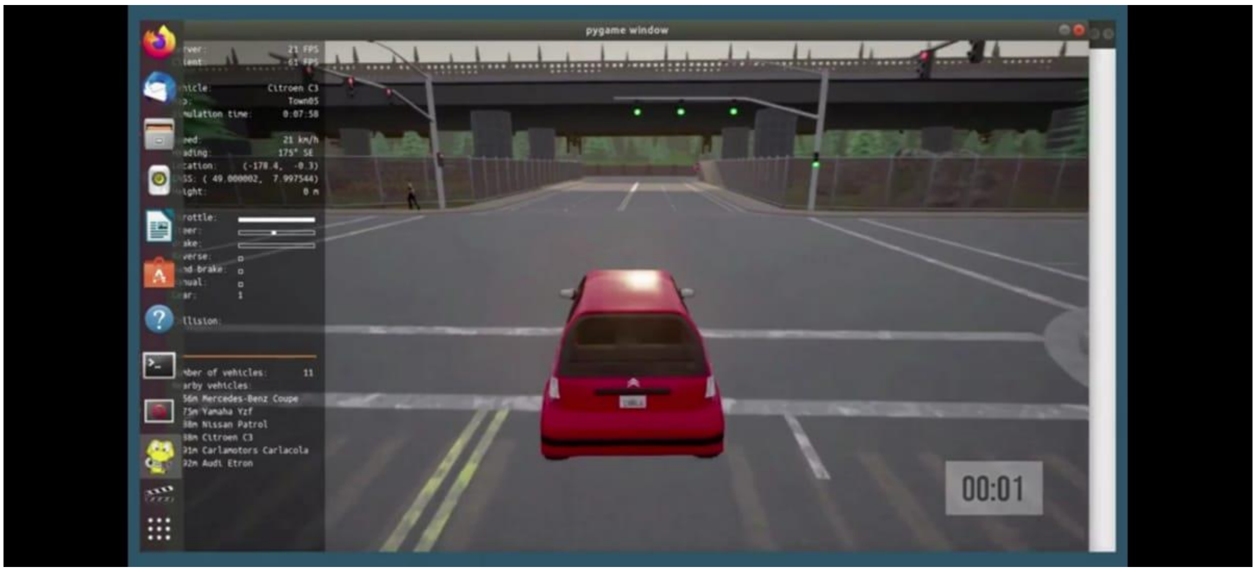
CHAPTER - 7. SIMULATIONS AND ROS

7.1 Simulation Software:

We have used various software for the implementation of our Path Planning Algorithms. The simulations of different software are given below:

7.1.1 Carla Simulator (0.9.6):

The CARLA environment has been used for the simulations. Photorealistic CARLA is a free, open-source training, validation, and testing simulator for autonomous vehicles. There is a wide range of digital assets and total control over the actors on the map, as well as control over ambient conditions, a sensor suite, maps production, a flexible API and server-client connection. CARLA includes a Python API module in addition to the CARLA Simulator, which incorporates all of the control logic, graphics, physics, and actor characteristics. As a result, the Simulator serves as the server, and the Python API governs communication between the client and the server.



```
rver:                 21 FPS
ient:                -61 FPS
hicle:            Citroen C3
p:                    Town05
mulation time:       0:07:58
eed:                 21 km/h
ading:               175° SE
cation:       (-178.4,  -0.3)
SS: ( 49.000002,   7.997544)
ight:                    0 m
rottle:
eer:
ake:
verse:        □
nd brake:     □
nual:         □
ar:           1
llision:
mber of vehicles:         11
arby vehicles:
56m Mercedes-Benz Coupe
75m Yamaha Yzf
88m Nissan Patrol
88m Citroen C3
91m Carlamotors Carlacola
92m Audi Etron
```

Fig. 31 CARLA simulation environment

We used the Carla Simulator environment to test the path planning codes. At first, to create the map and its structure, we used the **UNREAL Engine** software. For this, we used the

]

Linux platform in which we installed relative libraries which were useful for both the UNREAL Engine and the Carla Simulator.

7.1.1.1 ROS-BRIDGE:

When we want to run the ROS bridge, we set our ROS environment according to our ROS version (which is melodic in our case) in every terminal that we used and then we give the feedback to the Carla 0.9.6 Environment via the commands in the Ubuntu terminal as we did below:

```
source /opt/carla-ros-bridge/melodic/setup.bash
```

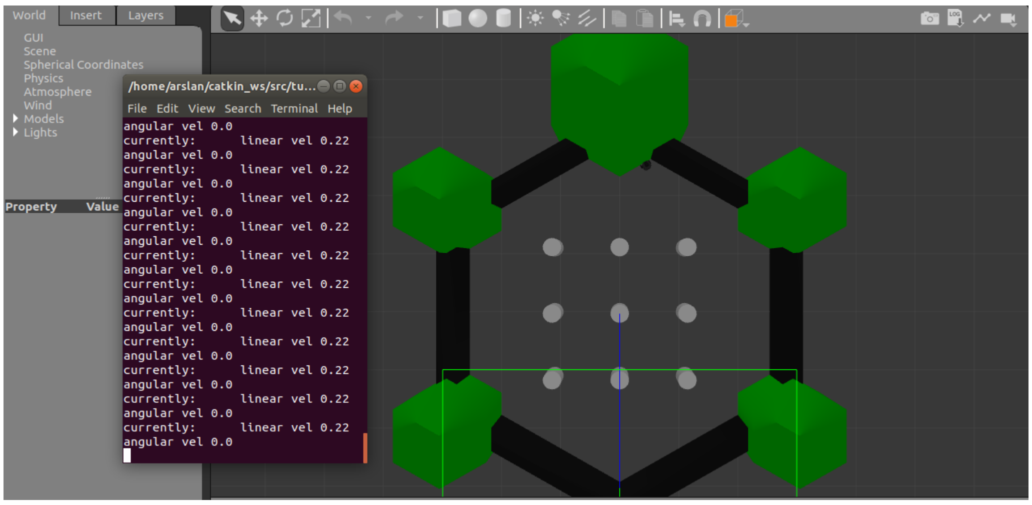
Then to launch this ROS_BRIDGE to communicate with Carla we used the command below to launch it:

```
roslaunch carla_ros_bridge carla_ros_bridge.launch
```

7.1.2 Gazebo Simulator:

It's a powerful robot simulator used in industry and academia that calculates physics, generates sensor data, and provides convenient interfaces. Robotics is benefiting from open-source software because it lowers the entrance barrier and speeds up progress.

In this software, we implemented our vehicle as a black robot operator and we checked the functionality of simple path planning maneuvers like going straight, going backward, taking the right and left turn, etc. The Pictorial Representation below depicts the Environment of the Gazebo Simulator:



```
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
currently:      linear vel 0.22
angular vel 0.0
```

Fig. 32 Environment of Gazebo Simulator

We basically used this Gazebo Simulator for our testing because in Gazebo we can create 3D scenarios with the many varieties like adding obstacles to check for the obstacle detection and checking whether the robot who is acting as an autonomous vehicle deviates from its path or stops whenever it faces an obstacle.

For Interfacing the Gazebo with our codes and algorithms, we used the ROS Environment as we did for the Carla 0.9.6 Simulator but in this case, there was no need for the Ros Bridge as it is only designated for the simulators like Carla Simulator, etc.

We also controlled the velocity, heading, and steering of the robot to test our robot/vehicle to learn and then act autonomously to it.

7.1.3 RVIZ:

RVIZ is a ROS graphical interface that allows you to visualize a lot of information, using plugins for many kinds of available topics. We Used the RVIZ for mainly state-estimation and localization of our vehicle, that how our vehicle will display its maneuver in a certain environment. But Rviz could also be used in displaying the environment prior to proper implementation of the path planning maneuvers.

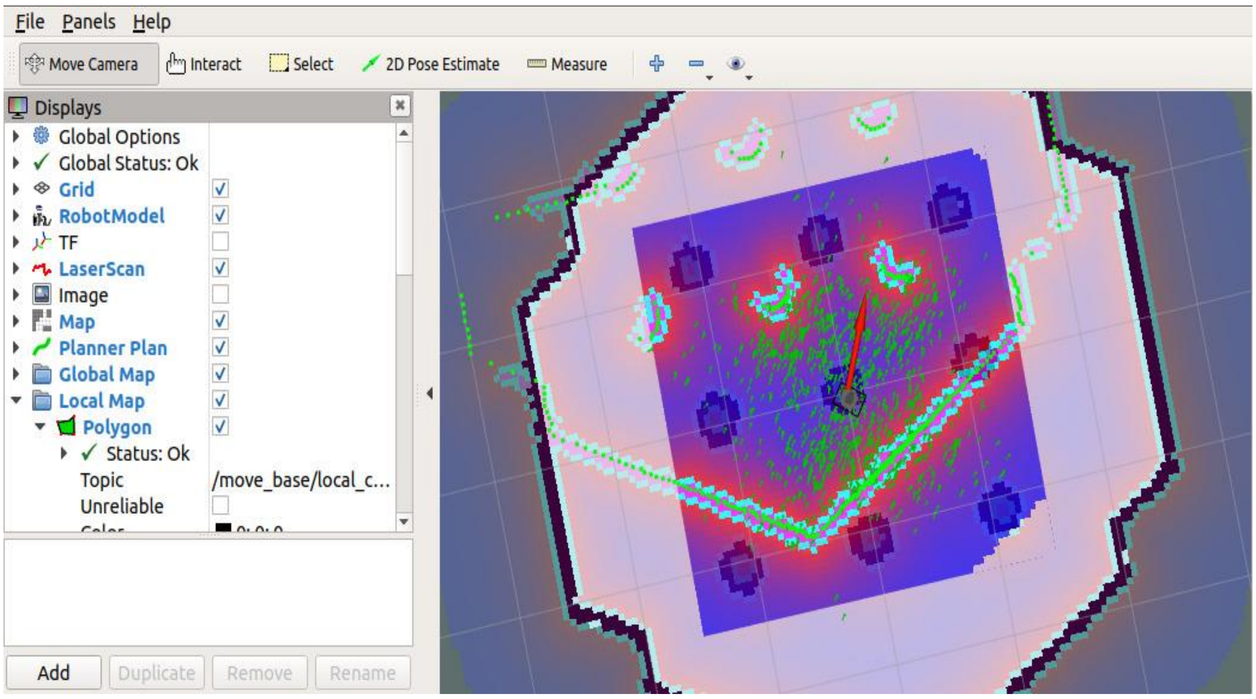


Fig. 33 Depicting the environment for the proper mapping and structure of its surrounding area

Mapping System/Environment that it shows:

Global Planning Map	Shows the Global Map that our Vehicle/Robot showcases.
Local Planning Map	Shows the Local Plan which our Vehicle/Robot will showcase and according to which our vehicle will plan its trajectory
Planner Map	Overall Planning Map for the whole venture/retrieval.

We used this RVIZ software and for its interfacing with the algorithms/codes in the Jetson AGX Xavier Developer Kit, we used ROS (Robot Operating System). For example, to simply open the RVIZ software we use the command:

rosrun rviz rviz

7.1.4 UNREAL Engine Editor:

Game development, architectural and automotive visualization, linear film and television production, broadcast and event production, simulation and training are all possible with UNREAL Engine's full-featured development platform. In our case, we have used the UNREAL ENGINE to create the maps on which our vehicle will plan the path in the simulation environment. The simulation environment can be any from CARLA SIMULATOR to UNREAL ENGINE AIRSIM. But, for all the detailed structuring and featuring of the map, we have used the UNREAL ENGINE platform.

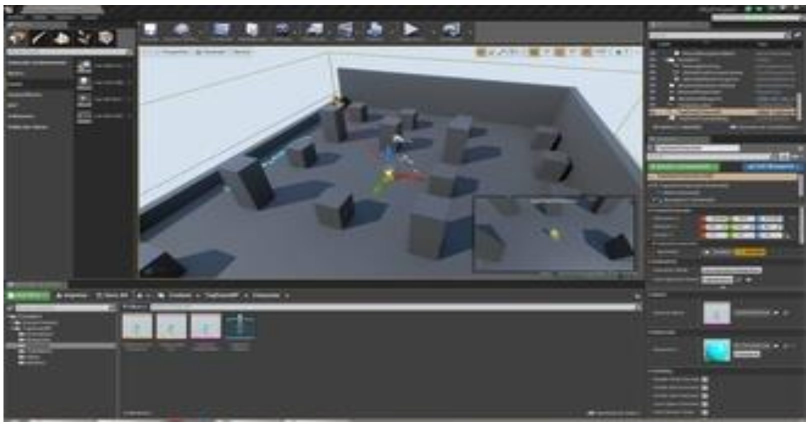


Fig. 34 Basic Layout of UNREAL Engine Editor

Fig. 35 Environment of UNREAL Engine

7.2 Communication with ROS:

7.2.1 Definition:

ROS is basically an open-source platform which is used to develop, integrate and communicate with the Self-Driving Car, and ROS includes plenty of different structure and functionality which could be used develop codes in a format which can communicate with our self-driving car effectively and efficiently.

7.2.2 Functionalities of ROS:

7.2.2.1 ROScore:

A collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a ROScore running for ROS nodes to communicate. ROScore is currently defined as:

- master
- parameter server
- ROSsout

7.2.2.2 ROSrun:

ROSrun allows you to run an executable in an arbitrary package without having to cd (or roscd) there first.

7.2.2.3 ROSnode:

ROSnode displays debugging information about ROS nodes, including publications, subscriptions, and connections.

7.2.2.4 ROSlaunch:

It Starts ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server.

7.2.2.5 ROStopic:

It is a tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.
The overall functionalities of ROS can be depicted from the figure below:

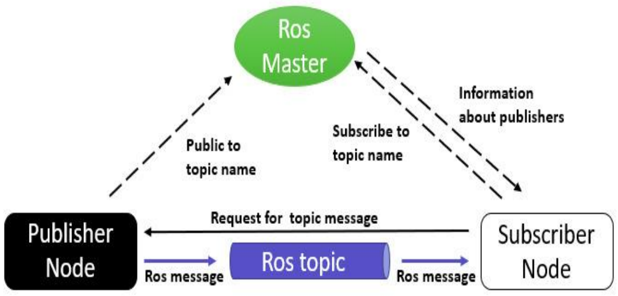


Fig. 36 Connection of publisher, subscriber and master node in ROS

7.2.3 Interfacing of ROS

All the codes of Path Planning, Perception stack and Control systems are connected through ROS. Below figure shows how ROS is interfaced with different modules of Autonomous vehicle.

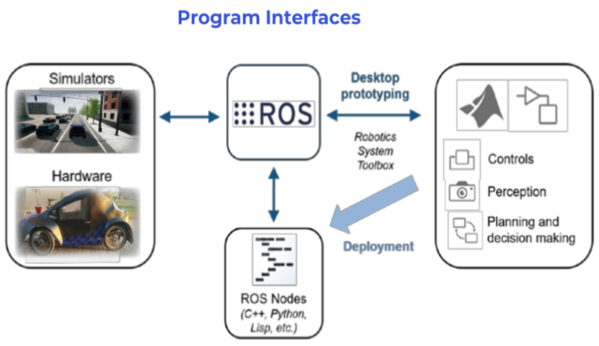


Fig. 37 ROS Interface

7.2.4 Pre-Requisites for ROS:

- Operating System Ubuntu 18.04
- ROS melodic version

For Implementing our codes on simulation, we will be using ROSbridge. Basically, ROSbridge is a WebSockets server with a JSON API exposing ROS service and pub/sub functionality. Mainly, simulators like Carla Simulators use ROSbridge for integrating

commands and codes, so the simulator can execute exactly on the basis of what the codes tell it to do.
For our hardware, we will be using ROSserial. ROSserial is a protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port or network socket. Mainly, hardware components like Arduino and other embedded components uses ROSserial as the way to communicate and integrate.

CHAPTER – 8. REFERENCES

[1] H. Gao, B. Cheng, J. Wang et al., "Object classification using CNN-based fusion of vision and LIDAR in autonomous vehicle environment," *IEEE Transactions on Industrial Informatics,* vol. 14, no. 9, pp. 4224–4231, 2018.

[2] C. Urmson, J. Anhalt, D. Bagnell et al., "Autonomous driving in urban environments: boss and the urban challenge," *Journal of Field Robotics,* vol. 25, no. 8, pp. 425–466, 200

[3] J. Leonard, J. How, S. Teller et al., "A perception-driven autonomous urban vehicle," *Springer Tracts in Advanced Robotics,* vol. 65, pp. 163–230, 2009.

[4] G. V. Raffo, G. K. Gomes, J. E. Normey-Rico, C. R. Kelber, and L. B. Becker, "A predictive controller for autonomous vehicle path tracking," *IEEE Transactions on Intelligent Transportation Systems,* vol. 10, no. 1, pp. 92–102, 2009.

[5] M. Kulich, V. Kozák, and L. Přeučil, "Comparison of local planning algorithms for mobile robots," *Modelling and Simulation for Autonomous Systems,* Springer International Publishing, New York, NY, USA, 2015.

[6] N. Ganganath and C. T. Cheng, "A 2-dimensional ACO-based path planner for off-line robot path planning," in *Proceedings of the International Conference on Cyber-Enabled Distributed Computing & Knowledge Discovery,* IEEE, Beijing, China, October 2013.

[7] Gao, X. Fourteen Lectures on Visual SLAM: From Theory to Practice; Publishing House of Electronics Industry: Beijing, China, 2017.

[8] A. MacLean et al., "Questions, options, and criteria: Elements of design space analysis," Human Computer Interaction, vol. 6, no. 3-4, pp. 201–250, 1991.

[9] T. Pender, "Waterloo's 'Autonomoose' hits 100- kilometer milestone," Waterloo Region Record, August
2018. [Online].

[10] K. C. Kang et al., "Feature-oriented domain analysis (FODA): feasibility study," Carnegie Mellon Univ., Software Eng. Inst., Tech. Rep.,1990.

[11] C. Urmson et al., "Autonomous driving in urban environments: Boss and the Urban Challenge," Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I, vol. 25, no. 8, pp. 425–466, June 2008.

[12] A. Bacha et al., "Odin: Team VictorTango's entry in the DARPA urban challenge," Journal of field Robotics, vol. 25, no. 8, pp. 467–492, 2008.

[13] C. Urmson et al., "Tartan Racing: A multi-modal approach to the DARPA urban challenge," Carnegie Mellon Univ., Tech. Rep., 2007.

[14] González, D.; Pérez, J.; Milanés, V.; Nashashibi, F. A Review of Motion Planning Techniques for Automated Vehicles. IEEE Trans.Intell. Transp. Syst. 2016, 17, 1135–1145.

[15]. Claussmann, L.; Revilloud, M.; Gruyer, D.; Glaser, S. A review of motion planning for highway autonomous driving. IEEE Trans. Intell. Transp. Syst. 2019, 21, 1826–1848.

[16]. Mentasti, S.; Matteucci, M. Multi-layer occupancy grid mapping for autonomous vehicles navigation. In Proceedings of the 2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE), Turin, Italy, 2–4 July 2019; pp. 1–6.

[17]. De Lima, D.A.; Pereira, G.A.S. Navigation of an autonomous car using vector fields and the dynamic window approach. J. Control Autom. Electr. Syst. 2013, 24, 106–116.

[18]. Sedighi, S.; Nguyen, D.V.; Kapsalas, P.; Kuhnert, K.D. Implementing Voronoi-based Guided Hybrid A* in Global Path Planning for Autonomous Vehicles*. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 3845–3852.

[19]. Yijing, W.; Zhengxuan, L.; Zhiqiang, Z.; Zheng, L. Local path planning of autonomous vehicles based on A* algorithm with equal-step sampling. In Proceedings of the 2018 37th Chinese Control Conference (CCC), Wuhan, China, 25–27 July 2018; pp. 7828–7833.

[20]. Feraco, S.; Luciani, S.; Bonfitto, A.; Amati, N.; Tonoli, A. A local trajectory planning and control method for autonomous vehicles based on the RRT algorithm. In Proceedings of the 2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE), Turin, Italy, 18–20 November 2020; pp. 1–6.

[21]. Zong, C.; Han, X.; Zhang, D.; Liu, Y.; Zhao, W.; Sun, M. Research on local path planning based on improved RRT algorithm. Proc. Inst. Mech. Eng. Part D J. Automob. Eng. 2021, 235, 2086–2100.

[22]. Wang, P.; Gao, S.; Li, L.; Sun, B.; Cheng, S. Obstacle avoidance path planning design for autonomous driving vehicles based on an improved artificial potential field algorithm. Energies 2019, 12, 2342.

[23]. Song, Q.; Zhao, Q.; Wang, S.; Liu, Q.; Chen, X. Dynamic Path Planning for Unmanned Vehicles Based on Fuzzy Logic, and Improved Ant Colony Optimization. IEEE Access 2020, 8, 62107–62115.

[24]. Grigorescu, S.M.; Trasnea, B.; Marina, L.; Vasilcoi, A.; Cocias, T. NeuroTrajectory: A Neuroevolutionary Approach to Local State Trajectory Learning for Autonomous Vehicles. IEEE Robot. Autom. Lett. 2019, 4, 3441–3448.

[25]. Elsayed, H.; Abdullah, B.A.; Aly, G. Fuzzy logic-based collision avoidance system for autonomous navigation vehicle. In Proceedings of the 2018 13th International Conference

on Computer Engineering and Systems (ICCES), Cairo, Egypt, 18–19 December 2018; pp. 469–474.

[26]. Hamid, U.Z.A.; Saito, Y.; Zamzuri, H.; Rahman, M.A.A.; Raksincharoensak, P. A review on threat assessment, path planning and path tracking strategies for collision avoidance systems of autonomous vehicles. Int. J. Veh. Auton. Syst. 2018, 14, 134–169.

[27]. Feraco, S.; Bonfitto, A.; Khan, I.; Amati, N.; Tonoli, A. Optimal Trajectory Generation Using an Improved Probabilistic Road Map Algorithm for Autonomous Driving. Am. Soc. Mech. Eng. 2020, 83938, V004T04A006.