Development of Real Time Self Driving Software for Wheeled Robot with UI based Navigation

Karthik Balaji Keshavamurthi

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Saied Taheri, Chair
Corina Sandu
John B. Ferris

August 11, 2020
Blacksburg, Virginia

Keywords: SLAM, Localization, Motion Planning, Perception, Autonomous Vehicles

Development of Real Time Self Driving Software for Wheeled Robot with UI based Navigation

Karthik Balaji Keshavamurthi

(ABSTRACT)

Autonomous Vehicles are complex modular systems with various inter-dependent safety critical modules, the failure of which leads to failure of the overall system. The Localization system, which estimates the pose of the vehicle in the global coordinate frame with respect to a map, has a drift in error, when operated only based on data from proprioceptive sensors. Current solutions to the problem are computationally heavy SLAM algorithms. An alternate system is proposed in the thesis which eliminates the drift by resetting the global coordinate frame to the local frame at every motion planning update. The system replaces the mission planner with a user interface(UI) onto which the User provides local navigation inputs, thus eliminating the need for maintenance of a Global frame. The User Input is considered in the decision framework of the behavioral planner, which selects a safe and legal maneuver for the vehicle to follow. The path and trajectory planners generate a trajectory to accomplish the maneuver and the controller follows the trajectory until the next motion planning update. A prototype of the system has been built on a wheeled robot and tested for the feasibility of continuous operation in Autonomous Vehicles.

Development of Real Time Self Driving Software for Wheeled Robot with UI based Navigation

Karthik Balaji Keshavamurthi

(GENERAL AUDIENCE ABSTRACT)

Autonomous Vehicles are complex modular systems with various inter-dependent safety critical modules, the failure of which leads to failure of the overall system. One such module is the Localization system, that is responsible for estimating the pose of the vehicle in the global coordinate frame, with respect to a map. Based on the pose, the vehicle navigates to the goal waypoints, which are points in the global coordinate frame specified in the map by the route or mission planner of the planning module. The Localization system, however, consists of a drift in position error, due to poor GPS signals and high noise in the inertial sensors. This has been tackled by applying computationally heavy Simultaneous Localization and Mapping based methods, which identify landmarks in the environment at every time step and correct the vehicle position, based on the relative change in position of landmarks. An alternate solution is proposed in this thesis, which delegates navigation to the passenger. This system replaces the mission planner from the planning module with a User Interface onto which the passenger provides local Navigation input, which is followed by the vehicle. The system resets the global coordinate frame to the vehicle frame at every motion planning update, thus eliminating the error accumulated between the two updates. The system is also designed to perform default actions in the absence of user Navigation commands, to reduce the number of commands to be provided by the passenger in the journey towards the goal. A prototype of the system is built and tested for feasibility.

Dedication

Dedicated to my family, who have been a pillar of support through my entire life.

Acknowledgments

I would first like to thank my advisor, Dr. Saied Taheri, who has guided me through the past two years while I was trying to find my feet in graduate school. I am grateful for the flexibility he has given me in choosing and working on a project.

I would also like to thank Dr. Ferris and Dr. Sandu, for their participation in my thesis committee, and for being supportive and providing valuable feedback towards the project.

I would like to thank all my batch mates, with whom I was frequently involved in technical discussions. One such discussion led to this project and I am grateful for sharing time in graduate school with them. A special mention to all my lab members at Center for Tire Research (CenTiRe), past and present, who provided guidance and support towards my project and other aspects in the Lab.

It is impossible not to show gratitude to my roommates, who have been supporting me emotionally during my research. Last, but certainly not the least, I would like to thank my family, who have given me the freedom to travel thousands of miles, to a new country, so that I could pursue my dream. I am where I am because of everyone in this page. I thank you with all my heart.

Contents

List of Figures

List of Tables

List of Abbreviations

ABS Anti-lock Braking System

ACC Adaptive Cruise Control

ADAS Advanced Driver Assist System

AV Autonomous Vehicle

CAV Connected Autonomous Vehicles

DARPA Defense Advanced Research Projects Agency

ESC Electronic Stability Control

FOV Field of View

GPS Global Positioning System

IMU Inertial Measurement Unit

LIDAR Light Detection and Ranging

NLP Natural Language Processing

RADAR Radio Detection and Ranging

SAE Society of Automotive Engineers

V2I Vehicle to Infrastructure Communication

V2V Vehicle to Vehicle Communication

Chapter 1

Introduction

The field of automotive engineering was one of the most active fields of research in the late 20th and early 21st century. In spite of automobiles being around since 1885, there has been rampant growth in the domain towards the end of the 20th century. This growth is partially credited to the development of electronics and computation industry. The last few decades of the 20th century witnessed the inclusion of various electromechanical systems to improve vehicle safety, comfort and performance. Cruise Control, which is a driver assist system that maintains vehicle speed based on the driver's setting, started featuring in vehicles in the 1970s. Vehicles started including Antilock Braking System(ABS) since the 1980s. Since the 1990s, Electronic Stability Control systems, which assists the driver in maintaining stability of the vehicle, were common. The pattern which can be seen from this timeline is that safety critical innovations started featuring electronic components and safety was improved on vehicles by adding a layer of control to human driving by software based systems.

This layer of control started increasing as time passed. Automotive companies started focusing in automating aspects of driving gradually by investing on research on Advanced Driver Assist Systems(ADAS) [4]. An example of an ADAS system is the Adaptive Cruise Control [36], which is an upgrade of the basic cruise control system. It maintains the speed of the vehicle based on road conditions and also maintains a safe distance from a preceding vehicle based on the current speed of the preceding vehicle, which is detected by sensors like

RADAR and camera. Another example is the Lane Keep Assist system [18], which aims to keep a vehicle within lane boundaries. It consists of a camera that identifies lane bounds and controls the steering of the vehicle to put the vehicle back into the lane in the event that one strays away from the lane. Currently, the major focus of innovation in the automotive industry is to eliminate the influence of a human driver in the functioning of a vehicle. Such a vehicle is known as an Autonomous Vehicle(AV) or a Self-Driving Vehicle.

1.1 Autonomous Vehicles

Self-Driving vehicles, as the name suggests, are vehicles that are designed to drive by itself from point A to point B maneuvering through an environment with vehicles with manual drivers and regular traffic conditions. A Self-Driving Vehicle is one of the most prominent research fields of the 21st century. It has garnered the interest of scholars and engineers alike across the globe. The DARPA Grand Challenge [64] and DARPA Urban Challenge [66] events gave global recognition to the problem, with several universities working towards obtaining a feasible solution to the problem. Various automotive companies picked up the baton from these universities and are currently dedicating enormous funding for research related to releasing commercial Self Driving Vehicles.

1.1.1 Levels Of Autonomy

Before discussing the developments in the field of Autonomous Vehicles, a brief introduction of the levels of Autonomy of cars [11], defined by Society of Automotive Engineers must be addressed to provide an overview of how far we have come to solve the problem and the distance still left towards the solution. There are six levels of automation from level 0

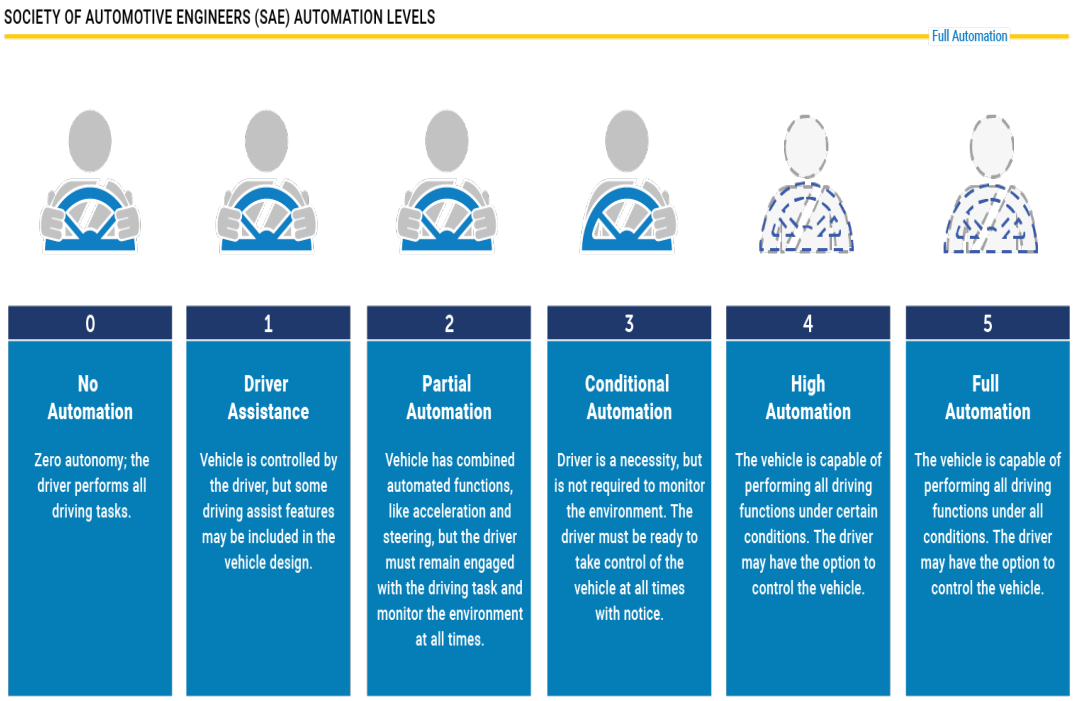

Figure 1.1: SAE levels of Autonomy, described by NHTSA [51]

to level 5, with each level introducing a higher degree of automation in vehicles, as shown in figure 1.1. Level 0 is a vehicle with no autonomy based features. Level 1 vehicles have control in either longitudinal or lateral direction. Cruise control and even Adaptive Cruise control based vehicles are level 1 vehicles. Level 2 is the hands off level, where the vehicle controls both the longitudinal and lateral control, however requires the driver to intervene if the automation fails. Level 3 vehicles allow the driver to take his ”eyes off” the road as the vehicle will contain control of all the essential tasks of driving. However, when the vehicle is uncertain of the environment, it informs the driver to take control of the vehicle. Level 4 vehicles offer full automation in limited operational domain. They could work only in a geofenced area or specific time of day. Level 5 vehicles offer full automation in any weather and road conditions. Level 5 vehicles are the target of the industry.

The Tesla Autopilot system [63], which is the self-driving package that is available at limited driveways in the US, is a level 2 system, which requires the presence of a human to supervise the functioning of the vehicle and must take corrective measures in a case where the vehicle performs a risky or unnecessary maneuver. Waymo, a subsidiary of Google, who are currently leading the research in autonomous vehicles, have released level 4 autonomous taxis in Phoenix, Arizona and are being road tested with human supervision.

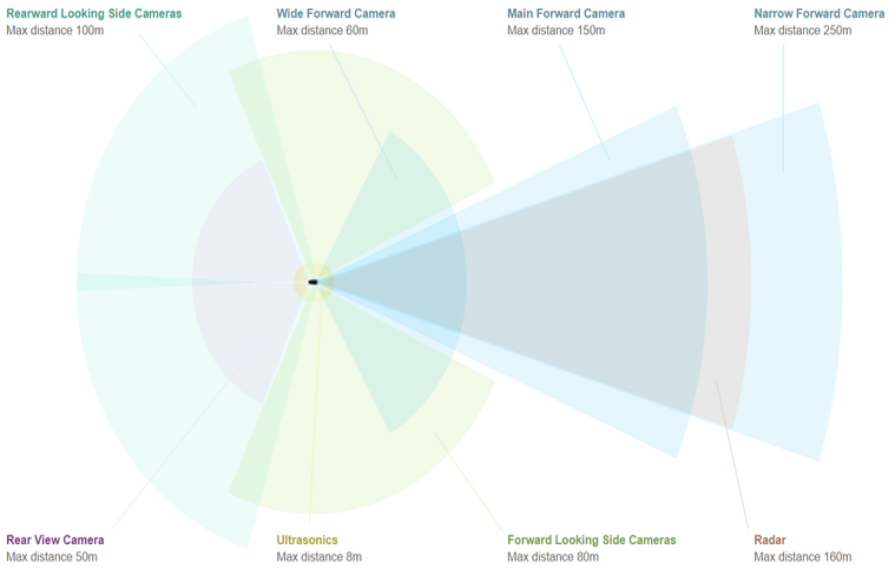

Figure 1.2: Tesla Model X sensor suite, from [63]

1.1.2 Approaches

Different companies have taken different approaches towards the development of autonomy. Tesla, an automotive company, has famously declared to work only with vision based sensors to perceive the environment [17]. They plan to use only Cameras and RADAsR to solve the autonomous perception problem and have made claims to remove the RADAR from the perception task eventually citing that humans identify obstacles using only their eyes

and hence autonomous cars should also depend only on vision. The Tesla Autopilot, a level 2 self-driving system works with a powerful suite of sensors including long range and mid range radars, cameras that cover the entire 360° field of view(FOV) and ultrasonic sensors for short range perception as shown in figure1.2. Other front runners in the industry like Waymo, Uber, and Cruise, use LIDARs, which are extremely expensive, apart from the other hardware in their perception stack, for the detection of the static and dynamic obstacles in the environment. The official image of the perception sensors of the 5th generation Robtaxi prototype of Waymo as shown in figure 1.3, contains multiple lidars across the periphery of the vehicle as well as multiple RADARs and cameras to perceive the environment across all 360°.

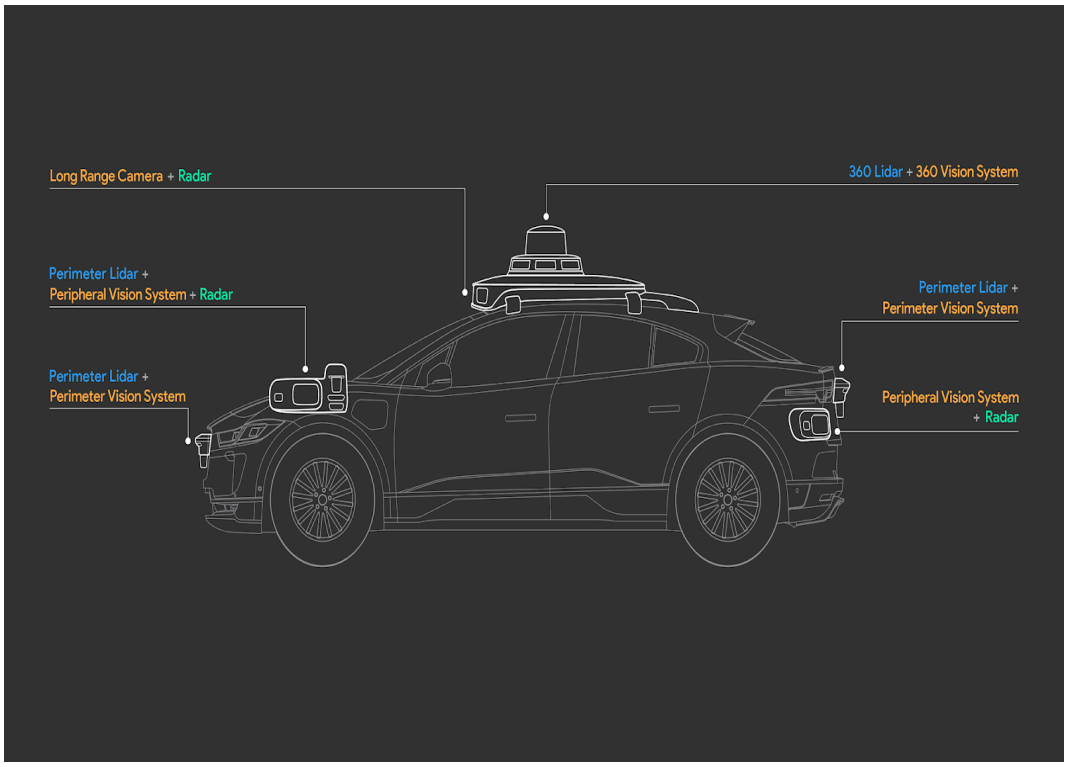

Figure 1.3: The Waymo sensor suite, described in [35]

1.1.3 Perks of Autonomous Vehicles

One might question the importance of AVs in everyday traffic. The prime purpose of the development of Autonomous vehicles is on the premise that it can shorten the amount of road accidents which cause loss to life and property. It has been estimated that majority of accidents occur due to driver's error of judgement [61]. Various accidents are caused due to Negligence of driver towards the environment, or to the road rules. Autonomous Vehicles tend to lower the risk of accidents of such type as they strictly adhere to the conditions programmed in the software and are only as effective as their decision framework. Apart from safety, there are various other factors boosting the economy. Autonomous Vehicles would make parking lots obsolete and that could free up huge chunks of infrastructure. Also, Autonomous Vehicles are said to reduce fuel emissions [46], which is a boon to the environment. One major perk of Autonomous Vehicles is the potential saving in time that is saved by communication among vehicles(V2V) and also communication between vehicles and infrastructure (V2I) like traffic signals. This is another active area of research in the AV domain known as Connected Autonomous Vehicles(CAV).

1.2 Motivation

From a safety standpoint, one can realize the importance of the software and hardware design of these vehicles, and the degree of robustness and meticulousness required for the effective functioning of such systems. Any errors in design could cause potential loss to property and even lives [12]. An example of this is the Uber Crash of 2017. The Company's software had not modeled the pedestrian to jaywalk and that cost the life of a human being [29]. This is the reason why extensive testing is required in both simulation and real world road tests under supervision to make sure all bases are covered.

In spite of many companies spending copious amounts of money and time towards solving the problem of autonomy, commercial autonomous vehicles have not yet been released. The reason for this is the high complexity of the complete self-driving problem. There are countless scenarios of traffic that need to be accounted for and not all have been completely addressed. For example, one of the major issues of autonomous vehicles is performance in inclement weather conditions. A camera-only perception stack would have issues in mapping the environment in case of heavy fog or rain. Hence, it is essential to design the system, with enough redundancy to account for the failure of certain subsystems. The example in this case, would be the presence of RADAR and LIDAR, to perceive the static and dynamic obstacles around the ego vehicle [45]. It is also essential to process the input from these sensors as the data might be noisy as well as be distorted by any environmental conditions like illumination, temperature, and humidity to name a few.

1.3 Objective

One such problem in AVs is the issue of propagation of localization errors. Localization is the process of locating the ego vehicle in a larger map as well as in an immediate environment of obstacles. Sensors used in localization include Global Positioning System (GPS), Inertial Measurement Unit(IMU), wheel encoders, range sensors like RADAR, LIDAR and also visual odometry from a camera. Usually information from all these sensors are integrated using the process of sensor fusion to obtain the location of ego vehicle with minimal error. One of the major sensors in this suite is the GPS, which provides information about the position of the ego vehicle in the global frame. However, there are issues with GPS output in the presence of high skyscrapers or tunnels and bridges, which are quite common in large urban cities. When GPS information is missing, the data from inertial sensors are

used to estimate position of the vehicle. However, these sensors have incremental build up in errors over time and hence the accuracy of such sensors degrade over time [9]. Another issue is the slow update rate of the GPS in high speed traffic, where the propagation of errors is more pronounced due to the dynamicity of the scenario. This has been tackled by companies by the construction of High Definition maps that contain landmarks at specific intervals that can be picked up by feature based visual odometry to correct for the position. However, construction of these maps is extremely time consuming, which require the vehicle to map the area several times and update created maps several times to account for the changes in the environment in a busy urban setting. Simultaneous Localization and Mapping(SLAM) based methods exist, which enable the vehicle to maneuver in environments by identifying landmarks and localizing vehicles in the global frame, based on the relative positioning of the ego vehicle from the identified obstacles in continuous time steps. These processes are computationally expensive and are prone to errors due to false detections and false loop closure scenarios.

The system proposed in this thesis aims at alleviating the issue of localization errors due to poor GPS signals by allowing the passenger to take over navigation of the vehicle. The passenger is tasked at providing a route for the car to follow in incremental steps, such as, instructing a vehicle to take a turn at the intersection, stop at a destination, or take an exit on a highway. This system would remove the necessity of maintaining a global frame of reference, on which a route is planned by a mission planner. The on-board software still performs the path planning and trajectory tracking of the local path planning problem by computing and executing safe and comfortable paths to the local goals provided by the passenger. At every motion planning update, however, the global frame is replaced with the ego vehicle frame, thus following the new trajectory in the new frame. Such a system could be activated when the vehicle detects poor GPS signals, or high localization errors,

due to sparse identification of landmarks, and until the car reaches a spot with a stable signal, or can even be employed as a standalone semi- autonomous ADAS feature that could take charge of the tedious task of driving the vehicle in traffic. A Behavioral Planner needs to be designed by formulating a decision framework to select actions based on the current environment, road rules and the navigation input from the passenger. A prototype of the real time software of such a system has been designed on a wheeled robot platform with a small subset of sensors that are used in autonomous vehicles. The prototype aims at solving a simplified problem as that stated above to check for the feasibility of a solution.

1.4 Outline

Chapter 2 discusses provides a background on the autonomous driving problem with more details on the issues of autonomous driving and the SLAM solution to Localization. Chapter 3 will provide an overview of the hardware used in the construction of a wheeled robot prototype, which includes documentation of the various sensors and computation platforms used and the reason for their selection. Chapter 4 will be a detailed overview of the complete software stack of the system, which includes perception from a depth camera, lane detection, object detection, path planning, trajectory planning and control system. The Results observed from the prototype are discussed in chapter 5, followed by which conclusions and prospective future additions to the project are discussed.

Chapter 2

Background

The previous chapter was a brief introduction of the problem statement and the proposed solution. This chapter will provide the required background on the various sub-systems of the autonomous vehicle and the previous work done on the same. Section 2.1 will give an overview of the typical hardware setup of an AV, which will be followed by a detailed description of the design of a typical AV software in 2.2.

2.1 Hardware

A self-driving vehicle is an integrated unit of several electro-mechanical systems [7], operating either independently or in tandem with one another. As discussed before, there have been varied approaches taken towards solving the problem. Different approaches demand type of hardware. A broad classification of the hardware of a self driving vehicle includes sensors, actuators, computation hardware, and communication peripheral.

2.1.1 Sensors

Autonomous vehicle software use data from a wide range of sensors for their functioning. High sensor redundancy is essential for AVs to function with robustness and reliability. The sensor suite is planned in such a way that failure of a specific sensor must always be com-

pensated by data from another. Another reason for sensor redundancy is the increase in the precision of the overall data. Sensor fusion is a technique by which noisy data from various sensors are used to obtain an estimate of the quantity being measured. In autonomous vehicles, sensor fusion is employed primarily in the localization system. Sensors are broadly classified into two categories [8]. They are exteroceptive sensors, which are used for perception and proprioceptive sensors, used for localization and other estimation tasks.

2.1.2 Exteroceptive Sensors

Exteroceptive sensors are used for perceiving the environment to identify and localize the obstacles around a vehicle. Obstacles are of two categories: static and dynamic. Static obstacles in traffic include dividers, road side units, safety cones, parked vehicles to name a few. Dynamic obstacles majorly include other motor vehicles, pedestrians, bikes, and animals. Exteroceptive sensors are tasked to detect and locate the obstacles as well as classify them into static or dynamic. This is due to the condition that dynamic obstacles are excluded in mapping task, but their information is required for motion planning. Hence, these sensors must be able to facilitate design of algorithms that can detect and track dynamic obstacles, also obtaining information about the state of the tracked vehicle of obstacle. Hence, data from exteroceptive sensors are useful for identifying obstacles on the environment as well as estimating the state variables of the dynamic obstacles which includes their position and even velocity.

There are a few metrics related to exteroceptive sensors that need to be defined before we explain the characteristics of different sensors. These metrics are tools to define the performance of each sensor. Certain applications require a higher degree of satisfaction towards a single metric, while certain applications demand versatility. These metrics, therefore, are the criteria on which appropriate sensors are selected for appropriate applications. The most

important metrics for autonomous vehicles include:

- **Range :**

  Range is the maximum distance up to which the sensor can provide reliable data, as shown in 2.1.

- **Field of View :**

  Field of view is the angle of coverage of the sensor. It is similar to a sector of a circle, where the sensor can only estimate information along this sector, as shown in 2.1.

- **Resolution :**

  - **Linear Resolution :** It is the smallest unit of distance a sensor can measure.
  - **Angular Resolution :** It is the smallest change in angle that a sensor can measure.

- **Accuracy :**

  It is a measure of amount of deviation from the true value of a reading.

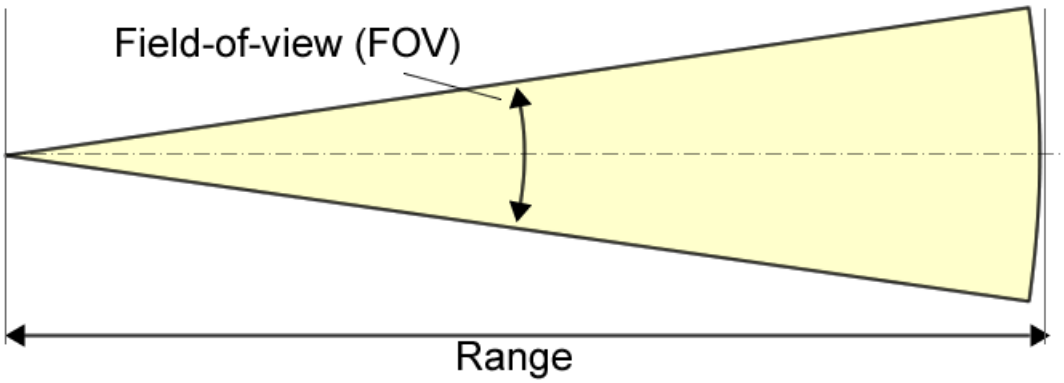

Figure 2.1: Picture depicting FOV and Range of a sensor, from [7]

2.1.3 Camera

The first class of sensors we will be discussing is the vision sensor. A camera is an optical sensor that captures images by restricting light through a small pin-hole, known as aperture. This light that enters the camera through an aperture is converted into electrical pulses, which is stored as pixels and this information can be recreated to form an image. Images can be classified as either monochrome or color images. Monochrome images are images taken in grayscale, which have only one channel, and color images have three channels namely R(Red), G(Green) and B(Blue). The intensities of each layer are superimposed to get the color image. Data from camera is extremely useful in identifying the surroundings. Computer Vision is an inter-disciplinary field of research that focuses on creating algorithms for a computer and hence, a machine, to understand its surroundings by the use of image or video streams.

There have been huge developments in the field of computer vision in the past few decades particularly in the application of autonomous vehicles [33]. Data from the camera is instrumental in identifying the dynamic and static obstacles in the environment of an ego vehicle by processing the various channels of data. Application of Deep Learning to computer vision algorithms are now ubiquitous in AVs. Lane Detection, Pedestrian and Vehicle Detection, traffic sign detection, traffic signal detection, and many more algorithms solely depend on camera and this makes it one of the most important sensors in an Autonomous vehicle [49]. Currently, autonomous vehicles have multiple cameras to obtain visual cues from all 360°, and use this data in various subsystems including Localization and mapping. Tesla vehicles that support autopilot have 8 cameras, whose information is fused and processed for the basic functioning of the autopilot. An image of how the Tesla autopilot views the road is shown in 2.2. It has been estimated that level 5 complete self-driving vehicles would have about 12 cameras to obtain all the visual cues for the planning and localization functions of

the self driving car.

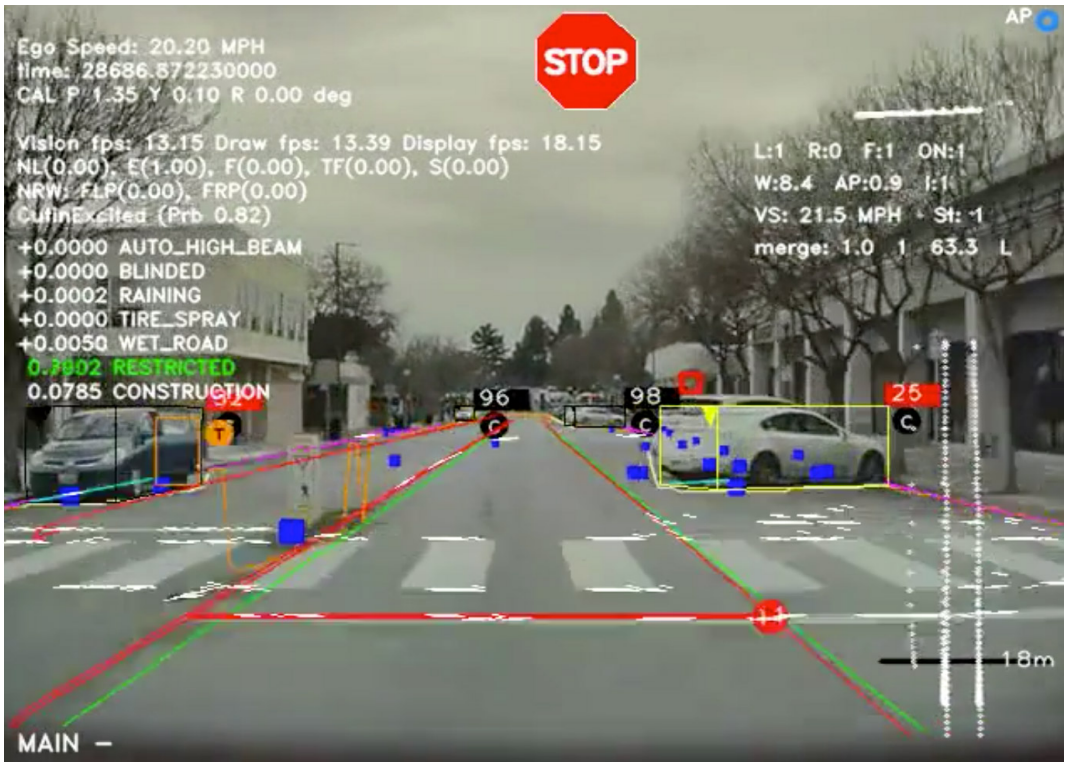

Figure 2.2: Image released by Tesla showing how it sees the road. We can see that it detects all the lanes, horizontal lines and cars. It also identifies cars and puts an ID number to track them.

Depth Camera

A depth camera is a special type of camera which provides the RGB data of an image as well as information about the depth associated with every pixel. The depth camera uses a stereo camera setting, which is, two cameras separated by a small distance, that are used to identify depth by analyzing key points in each individual camera, to calculate disparity and hence depth [48]. This is known as epipolar geometry. Modern depth cameras also contain infrared (IR) imagers, whose data is fused with stereo camera setting to get more accurate depth data. Depth cameras are used in a few AV prototypes. They are however,

more common in wheeled robots, which have spatial constraints and cannot accommodate a wide range of sensors. Application of computer vision on depth cameras can be used to identify all the visual cues from the environment as well as map them around the ego vehicle. Computer vision based tracking algorithms can be used to estimate the velocity of detected dynamic obstacles, which is used for motion estimation of these vehicles in the behavioral planner module. We will be using a depth camera in our prototype.

2.1.4 LIDAR

LIDAR stands for Light Detection and Ranging. It is a time of flight(TOF) based vision sensor. LIDAR sensors sense the environment in three steps. They emit light pulses at regular very small intervals. These light pulses are reflected off the environment and are received back to the sensor. A high resolution clock is used to find the time elapsed between transmission and reception of the light pulses, which is known as the time of flight. Using the TOF and the speed of light, adjusted to environmental conditions, we get the distance the light travelled, the half of which is the distance to the obstacle. There is a lot of post processing of this TOF data, which accounts for the vehicle motion within this time as well as handling corrupt data. LIDARs. LIDARs have been in use in the remote sensing application for decades and have only entered the realm of autonomous vehicles in the past few decades.

Autonomous vehicle perception system requires a sensor that could detect obstacles across 360° FOV with high accuracy and high speed. A LIDAR has all these characteristics [16], which makes it the first choice sensor for perception system. An example of a LIDAR, the Velodyne HDL- 64E has a 360° FOV with a range of 120 meter and angular resolution of 0.08°. Using LIDAR, a vehicle could see in the dark as well as in extreme rain, or fog, by processing the received data accordingly. The only disadvantage of using LIDARs is the

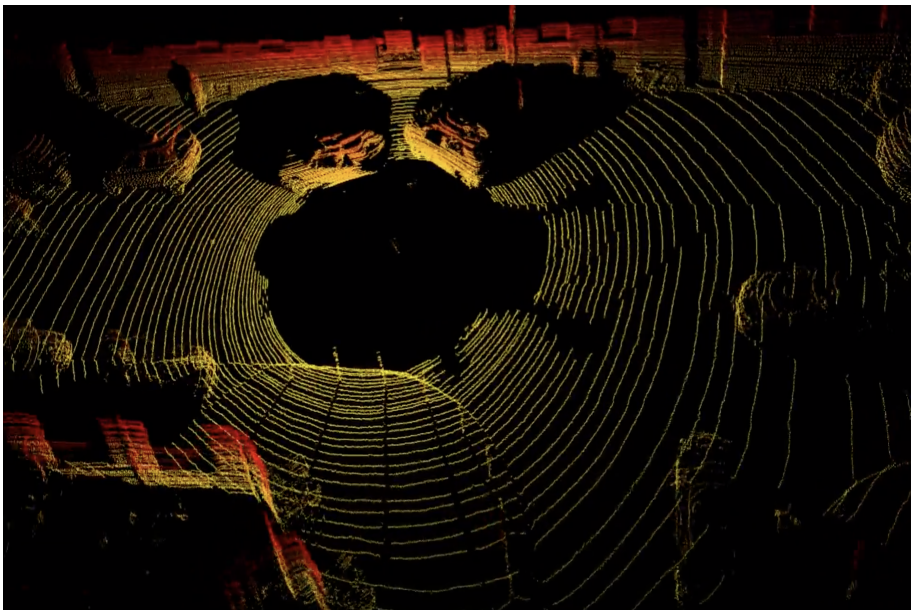

Figure 2.3: Visualization of a Point cloud from a LIDAR from the KITTI dataset [52]

extremely high cost associated with them. There is active research being pursued to create economical options to this technology. A visualization of data from LIDAR from the KITTI dataset is shown in 2.3.

2.1.5 RADAR

RADAR stands for Radio Detection and Ranging. It is similar to the working of the LIDAR. The only difference is that RADAR emits high -frequency radio waves instead of light pulses. RADARs are used in vehicles with Adaptive Cruise Control functionality. The major difference between LIDAR and RADAR is that, data from a RADAR can also provide information about the relative velocity of an object in the environment.They are used in Police vehicle speed scanners to measure the speed of vehicles on the road. Adaptive Cruise control system measures the velocity of the forward vehicle and adjusts the ego vehicle's speed to maintain a safe distance behind the leading vehicle. RADARs also work in darkness and

they also are contingencies for occlusion. Autonomous vehicles extract information about vehicles in front of a leading vehicle by Radio waves that are bounced off the road and onto vehicles that cannot be seen from a camera. A combination of high range low FOV and low range high FOV RADARs are used in AVs [57]. The advantages of RADAR is the high range accuracy and direct measurement of relative velocity. However, they do not have good angular resolution and are susceptible to interference from other radio sources. They also do not work well in close range.

2.1.6 Ultrasonic Sensors

Ultrasonic sensors are also TOF sensors and they emit high frequency audio signals for detection of obstacles. Ultrasonic sensors are low range simple sensors for low-frequency applications. They are used as proximity or parking sensors in vehicles. Blind-spot detection in vehicles also use the ultrasonic sensor. They are less expensive and are readily available in the market. The major advantage of ultrasonic sensors is that they are the most accurate sensors for very small range detection. However, one crucial disadvantage of ultrasonic sensors is due to the fact that the audio signal might be distorted if the air carrying the signal is not stationary. This means that they cannot be used at high velocities. They also have the interference problem similar to that of a RADAR.

2.1.7 Proprioceptive Sensors

So far, we have discussed sensors that can be used to obtain information about the location and type of obstacles in the environment. However, knowledge of the states of the current or ego vehicle is also essential for the working of an autonomous vehicle. The state of a vehicle is a set of parameters that define the current and future behavior of a vehicle. Knowledge

of the state of the current vehicle and the obstacles in the environment together need to be used by the software to guide the vehicle collision free in the environment. The sensors used to measure the state of the current vehicle are known as proprioceptive sensors.

There are a few metrics for these sensors that characterize their performance. These metrics need to be considered for the selection of these sensors as there is no one fits all sensor and we need to select these instruments according to the application.

- **Range :**

  The range of the sensor is the maximum and minimum measure of a quantity that can be identified by the sensor. Anything above the maximum will give a saturated response.

- **Resolution :**

  Resolution is the smallest measure of a quantity that can be detected by a sensor.

- **Bandwidth :**

  Bandwidth is a parameter that indicates the ability of a sensor to respond to changes in the measured quantity.

- **Accuracy :**

  Accuracy is the percentage of closeness to the measured and true value of a quantity.

2.1.8 Global Positioning System

The Global Positioning System(GPS) sensor is used to measure the position of an object relative to the Earth's coordinate system. United States based Global Navigation Satellite System(GNSS) called NAVSTAR, comprises of 24 satellites orbiting around the earth every 12 hours. These satellites transmit signals continuously comprising of a timestamp of the

start of signal. This is received by the GPS sensor on-board and the TOF is calculated to obtain the distance between the GPS and the corresponding satellite, after correcting for relativity and other errors. To obtain the 3 coordinates of a vehicle, the GPS needs to receive data from at least 4 satellites and the position is obtained by a process called Trilateration. GPS is the only sensor that can be used to obtain absolute position information in the world coordinates. This makes it a very important sensor for localization of a vehicle with respect to a map.

The coordinates received from GPS devices are not accurate. There are a lot of errors associated with clocks, the atmosphere, location, elevation, etc., that affect GPS data. Commercial GPS systems have an accuracy of 5 meters 95% of the time. High end Dual frequency GPS sensors, which are more suited for autonomous vehicle application have an accuracy of 5cm [2], which is tolerable for secure operation of AVs. However, signal from these GNSS satellites are weak and are hence may not be able to reach the receiver in areas with tall buildings, bridges, tunnels, etc.,. The accuracy in such cases drops down and the receiver also potentially might not receive data from 4 satellites, without which position triangulation is not possible. This is the major deficiency of the GPS sensor, the mitigation of which is a field of active research.

2.1.9 Inertial Measurement Unit

The other major proprioceptive sensor is the Inertial Measurement Unit(IMU), which is a combination of sensors which provide information about the dynamic states of a vehicle. The IMU contains accelerometers, gyroscope and optionally magnetometer to measure information such as acceleration and velocity of a vehicle. The accelerometers measure the linear acceleration in a specific direction. IMU comprises of three accelerometers to measure

linear acceleration in three axes. The gyroscope is used to measure the roll, pitch and yaw of the vehicle and magnetometer is a sensor that measures the orientation of vehicle with respect to the Earth's magnetic field, which can also be used to estimate the orientation of the vehicle. The technique of calculating the position of a vehicle from the data from IMU, which is speed and acceleration is known as dead reckoning. IMU cannot be used to directly measure the absolute position of a vehicle. However, IMUs can be operated at high sample rates, which makes them ideal for calculating position in between GPS measurements. The main disadvantage of IMU is that they have a build up of error with time, which makes it unsuitable for a stand-alone navigation sensor [71].

2.1.10 Wheel Encoders

Wheel encoders measure the rotational velocity of a wheel. This can be converted into velocity at the wheel by taking the loaded radius of a tire into account. However, this does not account for the slip on the tires and hence, estimates of velocity from wheel encoders alone would not be accurate. Wheel encoders are of two types: Absolute and Incremental. Absolute wheel encoders return the current angular position of the wheel, whereas incremental wheel encoders return information regarding the rate of rotation of the wheel. Incremental encoders are typically used in autonomous vehicles, where rotational velocity is used in the sensor fusion module to get a better estimate of the velocity. Wheel Encoders also have incremental build up in error with time and hence cannot be used as stand-alone navigation sensors.

2.2 Software

An overview of the major sensors used in Autonomous vehicles has been provided in the previous section. However, the design of software for autonomous vehicles has a greater effect on the performance of the AV. Having high end sensors in a vehicle without proper software design would do no good. The brain of the autonomous vehicle is in the software. Hence, in this section we will provide an overview of the software architecture of autonomous vehicles as well as a review of the state of the art algorithms currently employed in corresponding subsystems.

Autonomous Vehicles are comprised of software modules that perform certain functions by interacting with sensors and actuators as well as with other software modules on the vehicle. The software architecture of the Boss Autonomous vehicle by Carnegie Mellon University, the finalists of the DARPA Urban Challenge 2007 [66], is shown in figure 2.4 . The Autonomous Vehicle software is typically organized into 4 modules : Localization, Perception, Planning and Control. Localization is the module that estimates the state of the vehicle from the proprioceptive sensors of the vehicle. Currently, localization software also receives input from perception sensors to localize the ego vehicle in the map using visual cues and landmarks, which is discussed in 2.3. Perception is the module that takes care of identifying the environment of the vehicle. It is responsible for identifying the static and dynamic obstacles in the environment, which is mapped and utilized in the Planning module. The Planning module is the brain of the entire system and is responsible for any degree of autonomy in a vehicle. The planning module maps the environment based on perception outputs and plans safe and comfortable motion plans towards the motion goal, whilst avoiding the obstacles. The motion plan output from the planning module is tracked by the vehicle control system. There are multiple levels of control system in an autonomous

vehicle. We will discuss the high level tracking controller in Section 2.2.10.

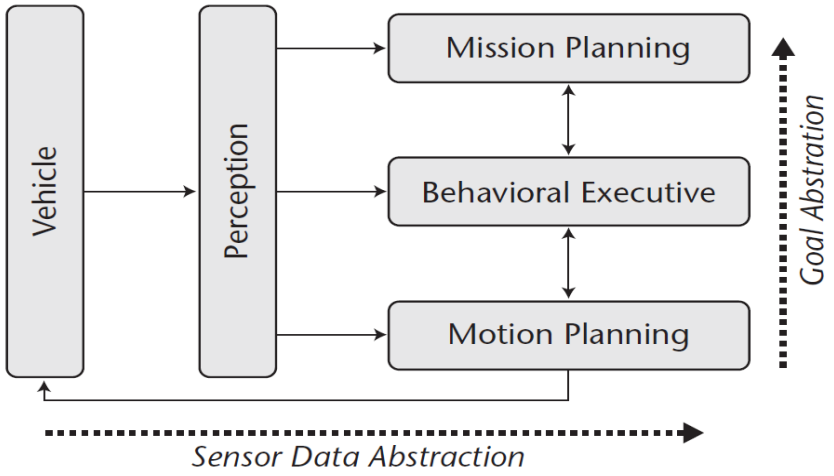

Figure 2.4: The Software Architecture of the CMU Boss vehicle of the 2007 DARPA Urban Challenge

2.2.1 Perception System

The Perception system of an Autonomous vehicle identifies the obstacles in the environment as well as localize them relative to the vehicles [3]. This system is analogous to the eyes and ears of a human body. There have been rapid developments in technology pertaining to the Perception system. The system employs sensors like Camera, LIDAR, RADAR, and ultrasonic sensors to accomplish this task. The various programs of the perception system of an AV includes Lane Detection, Horizontal line detection, vehicle and pedestrian detection, drivable surface estimation, to name a few. Perception system is also responsible for tracking the dynamic obstacles in an environment. The information about the dynamics of an obstacle is used in the planning module to estimate the future states of the obstacle based on certain models, which is taken as the basis for planning a trajectory to avoid said obstacles.

2.2.2 Object Detection

The object detection module is one of the most important safety critical application of the perception system. Object detection is the task of classifying what kind of obstacle an entity is, whilst also obtaining its location in the environment. The object detection problem has been solved in the literature using data from individual or combination of various exteroceptive sensors. A RADAR based object detection and tracking system, based on clustering is explained in [44]. Here, real-time Hierarchical clustering is utilized by the authors, which is a type of clustering without prior knowledge of number of clusters. They use the DBSCAN, which is a density based clustering algorithm, which takes RADAR noise into account to cluster nearby points as a vehicle. However, current state of the art methods in object detection rely on Deep Learning principles. There are real-time 3D object detection methods that run up to 25 frames per second(FPS) in [73], [60], [30], that use 3D convolution neural networks on pointclouds from LIDARs to detect the 3D pose of a vehicle. These are the state of the art methods on the KITTI 3d object detection challenge [21] currently. The result of [30] is shown in figure 2.5 The objects are detected in 3D and are bounding boxes are placed around them. However, these methods do not take into account the visual cues that are provided in RGB data.

Camera based object detection methods are quite common in AVs currently. The 3D pose of a vehicle is obtained by 2D object detection in an RGB image followed by clustering detection with LIDAR or RADAR data [**?**]. There are also stereo camera's RGBD data based object detection schemes available. The 2D object detection problem is a well recognized problem, that currently has open-source solutions available, such as YOLO [56], Single Shot Detectors (SSD) [43], RCNN [24], Fast RCNN [23], Faster RCNN [58], etc., that use convolution neural networks(CNN). They are used in identifying the road side units such as stop signs, regulatory signs, construction cones, and others. Knowledge of the presence of these elements

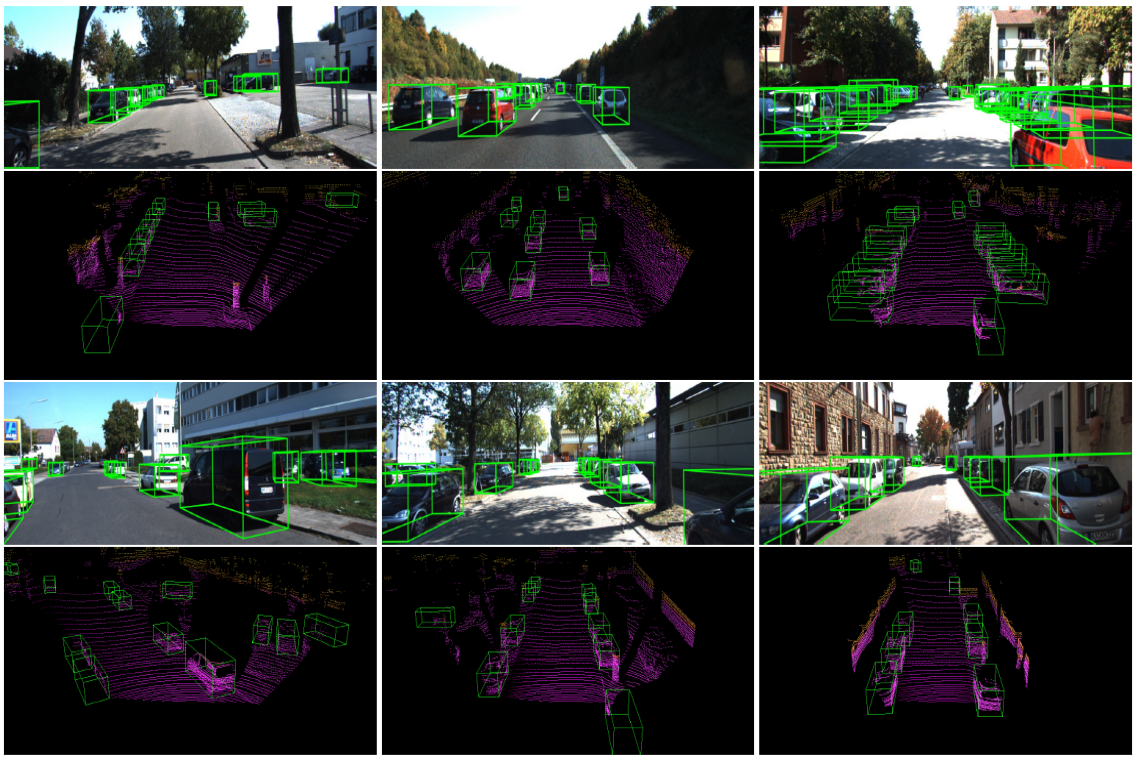

Figure 2.5: The detection result of [30] shown here. Bounding boxes are placed around these obstacles, whose coordinates are sent to the planning module1

is necessary for the Behavioral and Path planners.

2.2.3 Object Tracking

Tracking is the process of identifying a previously detected object in subsequent time steps. Tracking of vehicles and pedestrian, allows the behavioral planner to produce safe motion-aware paths for the ego vehicle to follow. Tracking of vehicles and pedestrians involves identifying a new object with an ID number and new detections of previously identified obstacles with their respective IDs. The process of identifying whether a detection is a previously detected object or a new one is known as Data Association. Data Association is also responsible for handling false detections and missed detections of obstacles. Once ID

numbers are added to the dynamic obstacles, their motion is predicted using appropriate models such as Coordinated Turn model, constant velocity model, or any other simple non-linear motion model, to get future estimates of a vehicle with respect to time, which is used to plan the trajectory of the ego vehicle. Computer Vision and deep learning based tracking algorithms are present for the Data Association task. Kalman Filtering based methods, however have better real-time performance in 3D object tracking, where data association is based on the Hungarian algorithm [55].

2.2.4 Localization

Various Autonomous Vehicles prototypes work with pre-built high-definition maps, using which paths are planned towards the goal location. The last section discussed the perception system in brief, which outputs the drivable surface and obstacles in the immediate environment. However, without ascertaining the location of the vehicle with respect to the HD map, it is not possible to plan path towards the goal. The Localization module is responsible for estimating this position with respect to ego vehicle as accurate as possible. It estimates the state of the vehicle by utilizing the data from the sensors equipped in the AV. We have already discussed about how a GPS can be used to obtain the absolute position of a vehicle. We have also discussed the use of inertial sensors and the process of dead reckoning to estimate the position of a vehicle by utilizing velocity and acceleration information. This process of obtaining the position of the vehicle with a combination of sensors is known as sensor fusion. GPS-IMU sensor fusion [34], which is explained above, is observed to not be accurate enough for application in AVs due to the drift caused by dead reckoning. Probabilistic filtering based methods were introduced, in conjunction with features and landmarks in the environment to obtain more accurate predictions of location of the ego vehicle. Systems like GPS-LIDAR fusion [32], camera-LIDAR fusion [70], among others are seen in literature,

that are being studied to eliminate the drift in localization estimates. Examples of such systems include a probabilistic Bayesian filter, known as the Extended Kalman Filter(EKF) to localize the vehicle, in the online- map that is constructed at the current time step [67], using data from a combination of sensors. These systems use motion models of vehicles, such as the non-linear bicycle model, in conjunction with the sensor inputs(z_k) at time k, to estimate the state(x_k) of the vehicle. Once a vehicle reaches a point that is already mapped using SLAM, the AVs use the loop closure techniques [41] to estimate their positions in the map. A more detailed review of the SLAM process is presented in Section 2.3.

2.2.5 Planning

The planning module is the brain of the system. It accepts the inputs of the perception system, uses the vehicle pose estimated by the localization system and then plans a safe path towards the motion goal. the Motion planning problem is broken into hierarchy of sub-problems [53] with changes in the degree of abstraction in each sub-problem. The levels of planning are shown in figure 2.6. The highest level is the Mission planner, which finds a high level route for the vehicle to take based on a map of nodes representing intersection and edges as streets to a goal location. This forms the reference path, which is the local goal for the sub-problems. The behavioral planner, with inputs from the perception and localization system computes a behavior for the vehicle to follow based on its decision framework. The local motion planner is composed of a path planner followed by a trajectory planner. A path is a sequence of way-points in space that are collision free, with respect to the static obstacles in the environment. Trajectory is a time-scaled function, which denotes the position a vehicle has to maintain with respect to time, for it to travel towards its goal safely and comfortably. It is in this phase, the dynamic obstacles are introduced, to get a velocity profile that assures no collision between the ego vehicle and the dynamic obstacles, which include other vehicles,

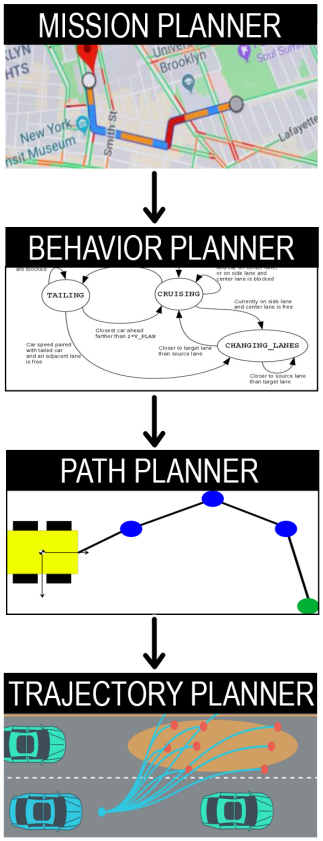

Figure 2.6: Hierarchical Structure of Motion Planning module of Autonomous Vehicles

bikes, pedestrian,etc.,.

2.2.6 Mission Planning

Mission Planning is the highest level sub-module in the motion planning system. It contains a road network with nodes representing intersections and edges representing streets towards intersections. The function of the Mission Planner is to find a high level path which contains a sequence of streets and intersections to visit for a vehicle to move from the start location to the goal location. The road network is maintained in the form of graphs. Path between

nodes, can be found by implementing the well-documented graph search algorithms.

Graph and Graph Search

A graph is a data structure that contains vertices and edges that connect said vertices. Each edge represents a connection between two vertices. Graphs can be directed or undirected. Directed graphs imply paths between two vertices exist only in a single direction. Undirected graphs imply two way paths. Weighted graphs are special graphs that contain a weight or cost associated with traversing an edge of a graph. A road network graph is an example of an undirected weighted graph, where the weight of edges or streets could be any function that relatively differentiates paths, such as path length, path traversal time, etc.,. Graphs can also be used in the local path planning problem, in which case, directed weighted graphs will be used. Graph Search algorithms can be used to find the shortest path in terms of distance or other cost functions between two points in the graph. Various graph search algorithms exist such as the Depth First Search(DFS) algorithm, Breadth First Search(BFS) algorithm, Dijkstra's algorithm [13], A* algorithm [28], D* algorithm, to name a few. The A* algorithm is used in the local motion planner of the prototype and is described in Section 4.5.4.

2.2.7 Behavioral Planner

The Behavioral Planner is responsible for the selection of maneuvers, the vehicle has to perform, to move towards the goal in a safe manner [38]. In the context of behavioral planner, a state is a maneuver a vehicle can execute, which could be maintain lane, change lane, stop at stop line, emergency stop, turn right or left, among others. There are countless scenarios of driving, which cannot be documented completely. Hence, a robust decision framework needs to be created such that a vehicle can estimate safe path for a vehicle to execute

under uncertain conditions. A computationally efficient and simple solution to the Behavior selection problem is a finite state machine, which selects the state of the vehicle based on a transition function, that depends on the previous state as well as the environment [54]. The Finite State Machine based Behavioral Planner will be used in the prototype designed in this thesis. However, for complex real world scenarios, there is high uncertainty in the behavior of the pedestrian or vehicle compared to their respective models and this needs to be accounted for. Therefore, reinforcement learning based Behavioral Planners that use optimization of a Markov Decision Process(MDP) to select the state and action of a vehicle are used in contemporary AVs, as in [72], [69]. Game Theory based Behavioral Planners [65] are an active area of research in the current age.

2.2.8 Path Planning

A path is a sequence of waypoints in space that ensures collision free movement to the goal configuration that is specified by the behavioral planner. The path planning module generates a feasible path to the goal, which is utilized by the trajectory planner, to create a dynamically feasible and optimal trajectory that a vehicle can track. The path planning module requires a map of the surroundings that covers the planning radius. This map is embedded with information regarding the location of the static obstacles in space, and is the configuration space of the vehicle for the path planning problem. The free space is the obstacle free points in the configuration space. Hence, all the waypoints selected by the path planner need to be in the free space, and the edge connecting the two waypoints must be collision free. Let 'X' be the configuration space, 'X_{free}' be the free space and 'x' be any configuration in the configuration space. Then the path planning problem can be formulated as:

Find $\sigma(\alpha)$ **for** $\alpha = [0, 1]$ such that,

- $\sigma(0) = x_{start}$,
- $\sigma(1) = x_{goal}$, and
- $\sigma(\alpha) \in X_{free}$

Various solutions exist for the path planning problem. The major categories of Path planning algorithms include: Graph Search Algorithms, Incremental Search Algorithms, Sampling Based Algorithms and Optimization based Algorithms. Selection of algorithms depend on the degree of robustness and computational capacity of the system. A review of the various path planning algorithms is shown in [53] and major algorithms are discussed in 4.5.

2.2.9 Trajectory Planning

The Trajectory planning module generates comfortable, dynamically feasible trajectory, that traces the path, is collision free, and minimizes a cost function and boundary constraints [26]. The trajectory planning module generally creates a function of a path variable α, which ranges between 0 and 1, by fitting a parametric curve based on the boundary conditions. Examples of parametric curves include cubic splines, quintic splines, bezier splines, cubic spirals, etc.,. The coefficients of the splines define the behavior of a trajectory variable, which could be the x, y or z coordinate as well as orientation [25]. The coefficients are calculated by solving, generally a non-linear programming(NLP) problem by minimizing a cost function subject to the dynamics of a vehicle(to assure dynamic feasibility), as well as the boundary conditions, which include the start and goal configuration, and constraints specific to the problem, which include curvature constraints, operating within the friction ellipse, Collision free, etc.,. The conversion of trajectory as a function of α to a function of

time is known as time scaling, which involves optimizing α as a function of time [27]. The trajectory as a function of time is sent as input to the control system for path tracking.

2.2.10 Control Module

The last module of software being discussed is the vehicle control system. This discussion only pertains to the high level trajectory tracking controller of the Autonomous Vehicle and not the various low level control systems such as traction control, stability control, etc.,. The control module generates control inputs required to track or follow the trajectory that is planned from the motion planner. In case of an autonomous vehicle, the control inputs are the steering and the throttle/braking commands. The DARPA Urban Challenge 2007 witnessed vehicles by teams equipped with individual controllers for longitudinal and lateral control [66]. The winning vehicle of the event, CMU's Boss, contained a PID based longitudinal controller which tracked the speed profile of the vehicle and a Pure Pursuit controller for lateral control [62]. Currently, Model Predictive Control, an optimal control strategy produces state of the art performance. The Model Predictive control [5] generates open loop control commands based on optimization of a cost function, based on a model of a vehicle for a time horizon. The MPC is computationally heavy. However, with modern developments in computation, Real time MPC solutions are available.

2.3 Simultaneous Localization and Mapping

Vehicles operating with HD maps, however, cannot be termed as completely autonomous, as they are unable to move in un-mapped regions in the environment. SLAM is the set of algorithms employed to localize a vehicle, using the landmarks in the environment, and map

the landmarks, based on the position of the ego vehicle in the global coordinate system, at the same time. Perfected SLAM-based systems would ideally enable the vehicle to move in uncertain environments, as mapping is completely online, and vehicles could plan paths in the global frame, towards the route planner sub-goal. SLAM algorithms use exteroceptive sensors, to identify coordinates of obstacles relative to the ego vehicle. The absolute position is obtained using GPS initially. As the vehicle moves, the landmarks are continuously extracted, and matched with previously identified obstacles. The location of the matched obstacles are used to correct for the position that is estimated by prediction and correction of the vehicle position using data from proprioceptive sensors [14]. A flow chart depicting the high level SLAM tasks is shown in 2.7. The SLAM module contains various sub-modules, such as Landmark and Feature Extraction, Data Association, Prediction and Correction [59].

Landmark or Feature extraction is the detection of features from exteroceptive sensor data. The method employed to detect Landmarks, depend on the type of sensors used. Currently, most SLAM based AVs in the literature employ camera based feature detection and extraction. Keypoints are detected and described in a specific manner in which they can be compared and matched in subsequent frames. There are various visual feature extraction systems available such as ORB [50], SURF [22], SIFT [19], to name a few. The process of classifying detections in the new frame as recurring detections or new detections is known as Data Association. Once new and recurring detections are classified, the following processes are executed in a loop:

- **Prediction :** The location of the ego vehicle and the landmarks, which constitute the SLAM state vector, are predicted using the vehicle motion model, with data of the current control input.

- **Observation :** The measurements of the landmarks are compared with that of the

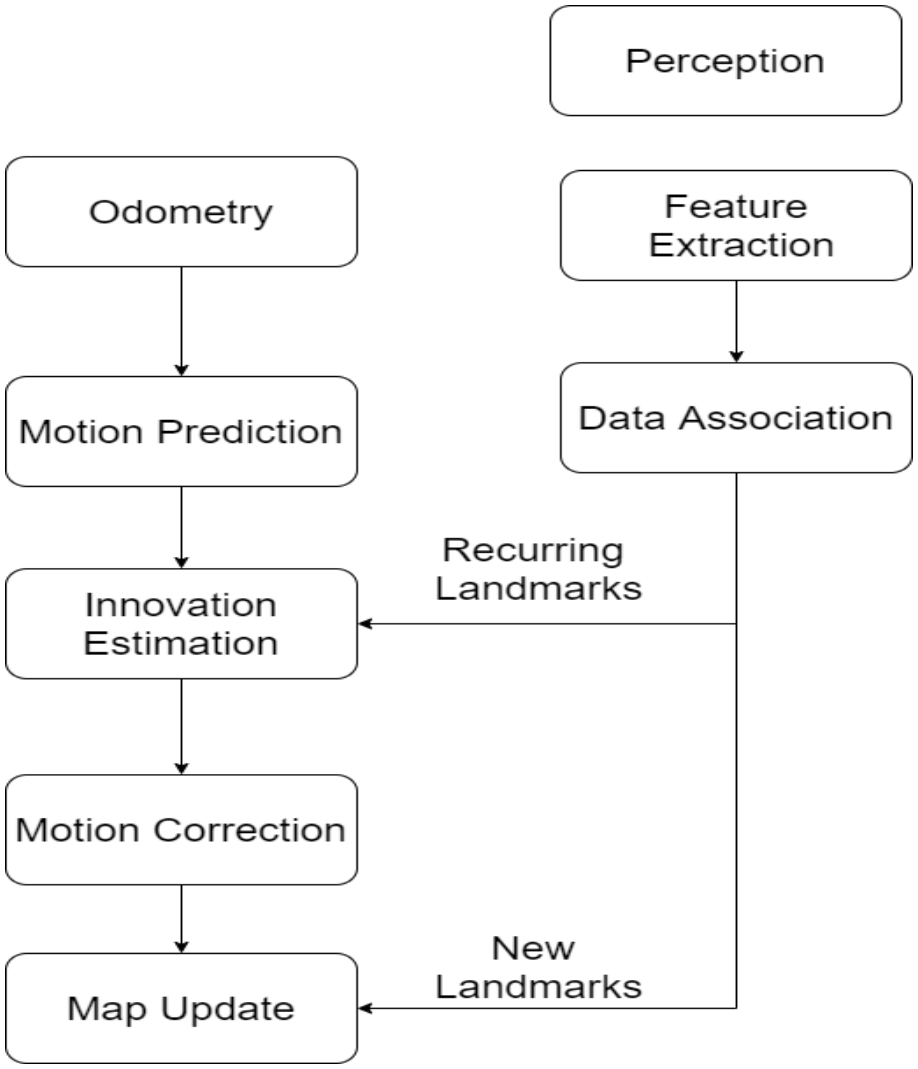

Figure 2.7: A typical SLAM algorithm Flowchart

predicted position, to obtain the innovation or correction gain.

- **Correction :** Using the innovation and a gain matrix, which is recursively computed, the complete state vector, which is the position of the vehicle and landmarks in the global coordinate system are corrected, and covariance is recalculated.

- **Mapping :** Following this step, the new obstacles are added to the state vector, with initial covariance, that is added to the covariance matrix.

Once a vehicle reaches a point, that is already mapped, a process known as loop closure is used to correct the complete map as well as accurately estimate the position of the vehicle. Loop closures are however, used for localization only after a particular point is visited more than a threshold number of times by a vehicle.

There are various advantages to SLAM algorithms, as SLAM based autonomous vehicles, need not be geofenced, and can operate in uncertain environments. However, the complete functioning of these methods depend on robust feature detection, extraction and matching, to obtain reliable localization estimates. The landmarks extracted from such algorithms must be static, reducing the complexity of motion model of the SLAM state vector. Such landmarks are easily estimated in urban environments, which contain large buildings and various static obstacles. However, in a fast paced highway, identification of static obstacles is a challenge, due to the sparseness of static obstacles in a highway setting. There are also errors that are caused due to false matching in the data association and module and missed detections. Another challenge in the SLAM problem is the high accumulation of the state vector, due to accumulation of landmarks in an environment, which creates a bottleneck for the system to perform real-time.

2.4 Summary

This chapter presented a brief review of the complete hardware and software suite of an Autonomous Vehicle. The typical sensors used in an AV were reviewed, with a description of the type of information a specific type of sensor can provide. The pros and cons of each sensor, and the selection metrics were briefed. This was followed by a review of the complete software architecture of a typical Autonomous Vehicle. Every individual subsystem was briefed, as well as the type of information that is processed using these software systems, so

as to get a complete picture from the point of extracting information from the sensors to the point of application of control inputs to the actuators. Current measures taken to reduce the increase of localization errors are described, with their merits and deficiencies, to elucidate how the system proposed in the thesis could be a possible measure in solving the problem.

Chapter 3

Wheeled Robot Hardware

Chapter 2 provided a background on the system composition of an AV, with a review of the modern state of the art software algorithms employed. This chapter will describe the construction of a wheeled robot, which will check for the feasibility of a human- based navigation system that is introduced in 1.3. The prototype of the system must have the following characteristics:

1. The robot must have an on-board computation board to provide a motion plan for a controller to follow.

2. The robot should be capable of locomotion along longitudinal axis x, lateral axis y and must be able to yaw r, with respect to the vehicle coordinate system, shown in figure 3.1.

3. The robot must be equipped with a perception sensor, which could provide RGB data and a point-cloud.

4. The robot must be equipped with an inertial sensor for localization using dead reckoning.

5. The robot must be equipped with an ADC to accept sensor data at 100 Hz sample rate.

6. The robot requires an embedded platform, which acts as the controller, that can accomplish serial communication with the on-board computation board.

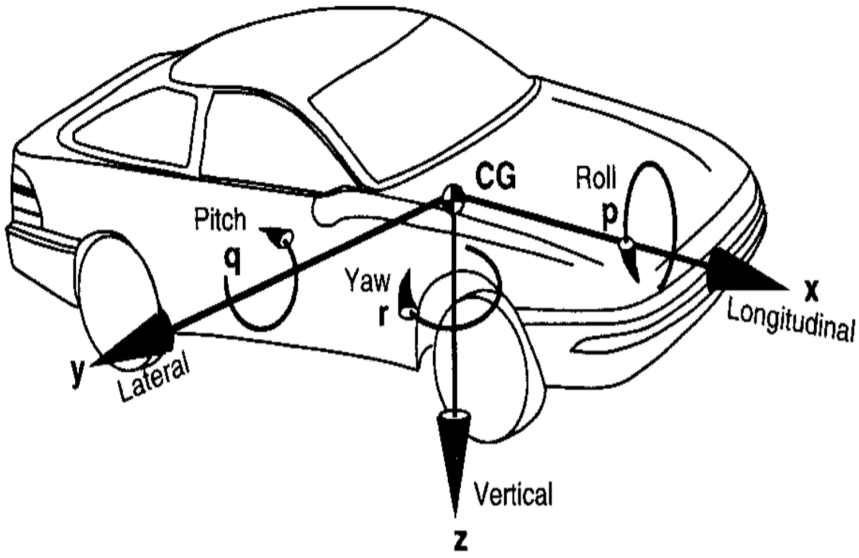

Figure 3.1: The vehicle coordinate system adapted from [**SAE insert**]

3.1 Robot Design

The wheeled robot must encompass all the hardware elements of the robot discussed above, along with the hardware required for its locomotion. The actuators, which provide force for locomotion need to be powered by DC power supply, which must also be accommodated in the chassis. The actuators in this robot are 9V DC motors, whose speed and direction control can be achieved by simple H-bridge circuitry. The speed is controlled by Pulse Width Modulation (PWM) signals, that is sent by the controller. The controller software is programmed on an Arduino Uno, an open source microcontroller with analog and digital I/O interface. The Arduino Uno samples acceleration signals from the inertial sensor and modulates the speed based on the control law. The Arduino Uno also is connected via serial

communication with the on-board computation board which is discussed in 3.4. The Arduino Uno is selected as they can be programmed with simple C++ language and uploading the program into the micro-controller does not require extra hardware. The Arduino IDE sets up a serial link to compile and upload the code to the Uno, thus simplifying the code update process. The maximum sampling rate possible with an Arduino Uno is 9615Hz, which is higher than the required setting of 100Hz and hence would be suitable for our application.

3.1.1 Steering Mechanism

The robot must be capable of longitudinal and lateral motion. Longitudinal motion can be achieved by in-hub motors on the rear axle of the robot, which generate torque that is converted into traction force by the tires. Lateral motion however, is achieved by steering the vehicle. Autonomous vehicles, employ the Ackermann Steering mechanism, which consists of an open four- bar linkage system. However, this would complicate the design of the robot by adding extra parts such as a rack and pinion, tie rods, to name a few. For the purpose of a compact design, a skid steering system [68] is selected for this application. Skid steering is the most common steering mechanism for wheeled robots, as they require lesser moving parts and are compact in terms of space. Skid steering is the mechanism by which the vehicle steers due to difference in velocity along the left and right wheels. The vehicle steers towards the direction with the lower velocity and the steering action occurs due to slip between the tire and the surface and hence the name. The mechanism is employed in the prototype by connecting the 4 wheels of the robot with individual in-hub motors, whose speed is varied by the controller. The left wheels(both front and back) rotate with angular velocity (ω_L), while the angular velocity of the right wheels is (ω_R), as shown in figure 3.2. Then :

- For Left turn, $\omega_L < \omega_R$

- For Right turn, $\omega_L > \omega_R$
- For no steering, $\omega_L = \omega_R$

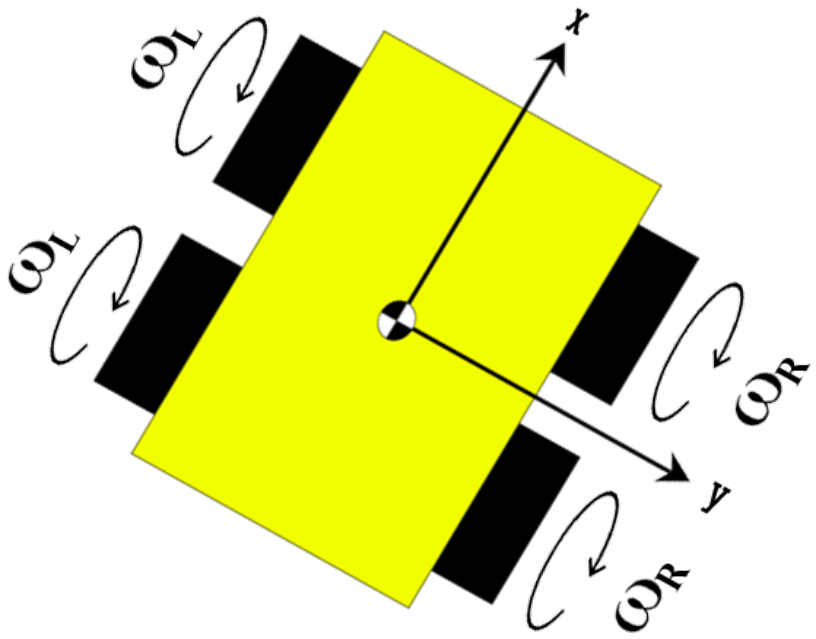

Figure 3.2: The layout of a skid steering vehicle

The disadvantage of such steering mechanism is the complexity of modeling the slip that causes steering action. A simple kinematic model is employed as the wheeled robot does not operate over large speeds. The model for the skid steering system is explained in the controller section in 4.6.1.

3.2 Inertial Sensor

The system proposed in this thesis aims at solving the issue of GPS errors by navigation assist from the driver, as an alternative to computationally heavy SLAM based or map

based methods.The robot requires a localization sensor that measures the acceleration of the vehicle which can be used to obtain position in future time steps using dead reckoning. As discussed in 2.1.9, this can be achieved using an Inertial Measurement Unit. We are using an MPU-9250, which is a 9 degree of freedom IMU, comprising of accelerometers that measure linear acceleration along the x,y and z axes, gyroscopes, that measure the pitch, yaw and roll of the sensor and magnetometer, which measures the orientation of the sensor, with respect to the Earth's magnetic field. The MPU-9250 is selected for this application, as drivers for the sensor are open-source and are readily available, which implies interfacing the sensor is only restricted to connecting the IMU to the microcontroller, which is the Arduino Uno. The MPU-9250 is a MEMS sensor, that is characterized with lower accuracy as compared to state of the art IMUs. However, the objective of the project is for the AV to safely move in its surroundings with uncertainty in position, which is one of the factors considered in the selection of this sensor.

3.3 Vision Sensor

The robot requires a source of information regarding its surroundings, using which dedicated algorithms can identify the obstacles and plan safe paths around them. For the autonomous functioning of the robot, lanes have to be identified to provide bounds on the robot's position. As discussed in Section 2.1.3, visual cues from the environment can only be extracted by a Camera, which provides RGB information. Algorithms can be defined, that extract a particular color or object from this RGB data, as discussed in section 2.2.1. The robot also requires data regarding the relative position of points about the robot, so as to obtain the coordinate of obstacles in the vehicle coordinate system. This can be achieved by a variety of exteroceptive sensors. However, for a compact design, a depth camera was selected as the

vision sensor as it provides both forms of data with minimal noise. The only disadvantage is that we require multiple depth cameras to cover 360°. However, the prototype of the system is designed without consideration of dynamic obstacles and maneuvers like lane changes. Hence, a depth camera with a minimum of 45° FOV would be sufficient for the robot.

There are various types of depth cameras available in the market that can be used for the particular application. Mynt, Microsoft, Intel, Sick, StereoLabs are a few companies that have stable OEM Depth cameras available in the market for IOT Robotic Applications. Depth cameras perceive depth using different approaches. StereoLabs ZED is an example of an embedded stereo camera setting, that uses only stereo camera data to obtain disparity and hence the depth of pixels in the FOV. This is done by epipolar geometry. However, the accuracy of such sensors is affected by the lighting conditions and reflectivity of surfaces. This issue is addressed in depth cameras that fuse depth obtained from the stereo camera with active infrared(IR) stereo sensors. This provides accurate depth information for pixels within the sensor's range. One of the famous commercial active IR based depth cameras are the Intel Realsense depth cameras.

3.3.1 Intel Realsense D435

The depth camera selected for this application is the Intel Realsense D435. The D435 can provide depth and RGB information at 60 frames per second (FPS) for a field of view of 70° in the XY plane. The D435, contains two IR cameras, and an IR projector to capture depth data and an RGB module to capture RGB data, as shown in 3.3. The D435 has an internal PCB that fuses the depth and RGB data to provide aligned depth images for image processing. The most important advantage of D435 is the Intel Realsense Software Development Kit, which is a software library that is open-source and free to use for robotic applications. The

software designed in this project utilizes the library to extract the RGB image as well as identify the coordinates of obstacles and lanes and transform them into coordinates in the Vehicle frame using the intrinsic and extrinsic parameters using projection geometry. The depth camera performs on-board calibration and tunes its intrinsic parameters across each frame and hence, the user is not required to perform calibration of the camera and can extract the calibration information by a simple call to a function in the Realsense library.

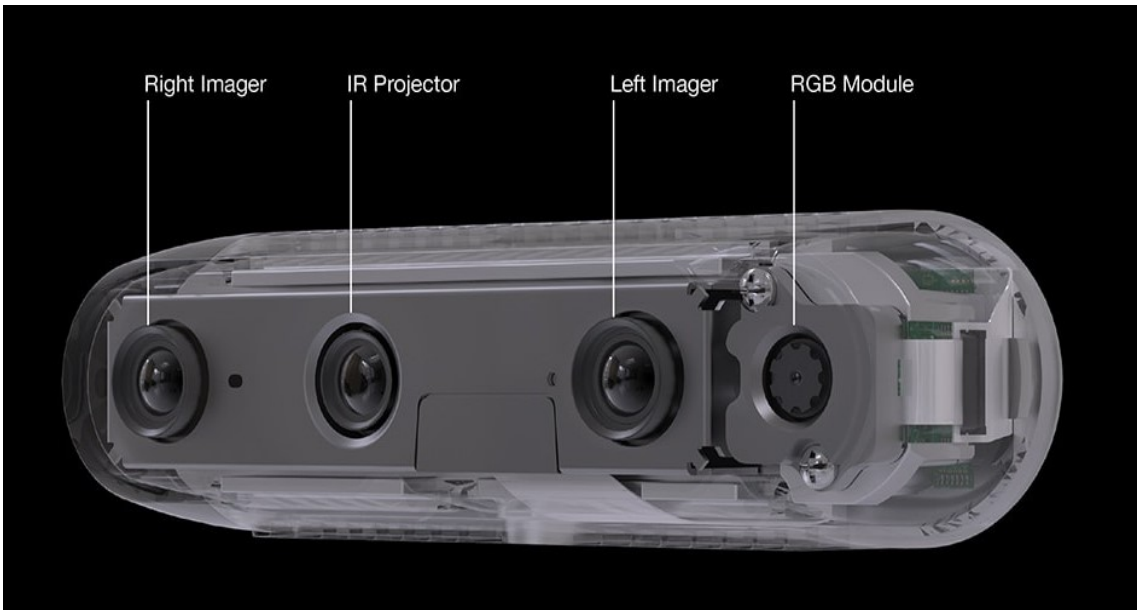

Figure 3.3: Sensors used in Intel Realsense Depth Camera, courtesy of Intel

3.4 Computation Board

We have discussed the controller board(Arduino Uno), which communicates with the inertial sensor and calculates control input which is sent to actuators. We have also discussed about the depth camera which provides RGB-D data. However, we require a computation platform that processes the environment from the RGB-D data and generates a motion plan, which can be tracked by the controller. A computation board suitable for the wheeled robot prototype

must have the following characteristics:

- The processor on the board must have sufficient computational capacity to run computer vision algorithms.
- The computation board must have large enough static memory to store the libraries required for the software.
- The computation board must have sufficient dynamic memory in the form of RAM, which can be utilized by the software.
- The computation board must be able to accept serial communication through USB 3.0 ports to connect to the depth camera.
- The computation board must be able to wirelessly connect to other servers.

There are many IOT ARM architecture single board computers available for robotic prototypes. The raspberry pi series of microcontrollers are prominently used in IOT applications with a 4GB ram and GPIO pins and USB slots for connection to camera and other applications. They are built over the Raspbian Operating system and can directly compile codes programmed in C++ and python. They also have Robot Operating System(ROS) support. However, they do not have a Graphical Processing Unit(GPU), and hence would not support parallel processing. One of the newer computation boards for robotic and AI projects are the NVIDIA Jetson series of computation boards. The computation board selected for this project is the NVIDIA Jetson Nano.

3.4.1 NVIDIA Jetson NANO

The NVIDIA Jetson NANO is the latest ARM microcontroller released as part of the Jetson series for lightweight robotic and AI projects, which is shown in figure 3.4. The Jetson NANO

Figure 3.4: An image of the Jetson Nano computation board, courtesy of NVIDIA

has a 4 core processor with a 4GB RAM. The multi-core processor enables multi-threading, the process of scheduling operations in multiple cores simultaneously to increase the speed of the software. The Jetson nano contains 4 USB 3.0 ports, two of which are utilized by the depth camera and the arduino microcontroller. One of the greatest advantages of the Jetson Nano is that they operate with a Linux Ubuntu operating system, which supports all commercial libraries and is hence desirable for software development. It can also be accessed using SSH protocol on any computer that is connected to the same network, which reduces complexity in uploading code to the Robot. Jetson NANO also provides CUDA support, which enables the software developers to use the GPU for parallel processing, which is desirable in projects that employ Machine Learning and computer vision based tasks.

3.4.2 Human Machine Interface

The hardware elements on-board the robot have been discussed thus far. However, the components that have been discussed cannot be used as an interface for a human to communicate. Therefore, a remote computer with a User interface is required to extract the

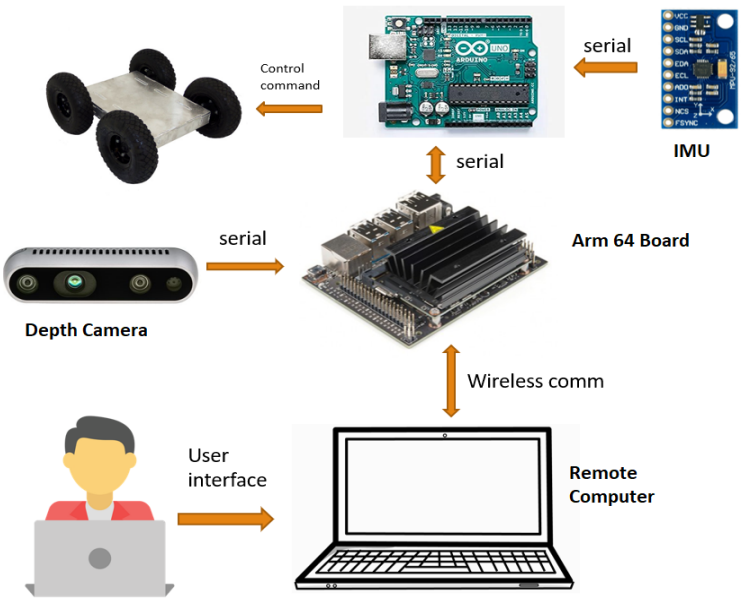

Figure 3.5: The complete Hardware stack and their connection interface.

navigation command and transmit the command to the robot. The UI could be a Graphical User Interface(GUI) or a terminal, where the user can type in commands for the robot to follow. This computer also performs a fragment of the processing of the software, which will be discussed in chapter 4. In our project, we use a laptop with an 8 core 2.8GHz processor, that wirelessly transmits the command received from the user to the robot. The complete Hardware architecture is shown in figure 3.5. The components in the robot are connected using serial communication and the robot is connected to the computer with UI, through wireless TCP protocol.

3.5 Robot Summary

The selection criteria of the hardware elements of the wheeled robot was described in this chapter. The wheeled robot uses a skid steering setup, whose motion is provided by 4 in-hub 9 volt DC motors, connected to H-bridge circuitry for speed control. The MPU - 9250 MEMS

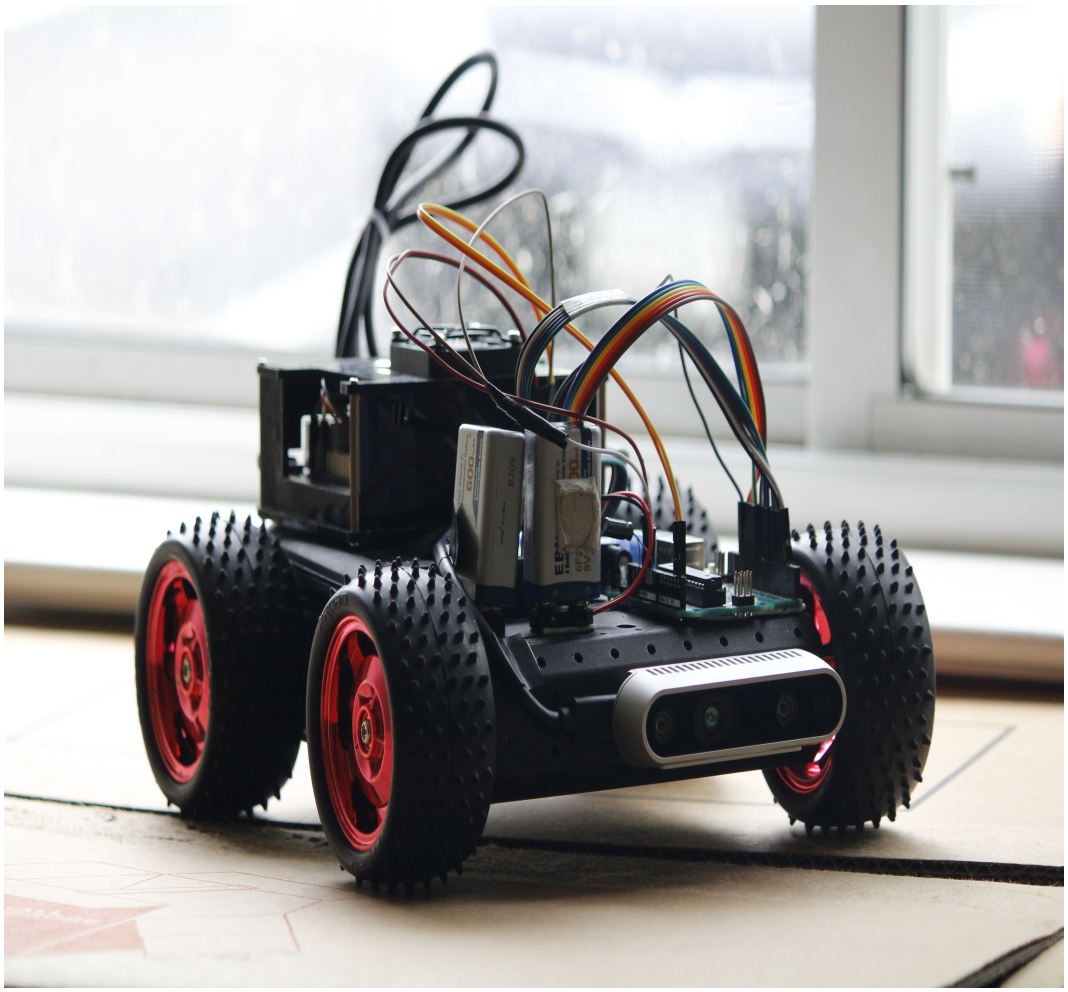

Figure 3.6: The Wheeled Robot prototype

IMU is the inertial sensor, whose noisy data will be used for the localization of the robot. The Arduino Uno is the microcontroller which controls the speed of the motors by PWM signals, based on the input from the IMU. The controller is connected by serial communication to the NVIDIA Jetson Nano ARM microcontroller, which generates motion plans that are tracked by the controller. Finally, the Intel Realsense D435 is the depth camera that functions as the lone perception sensor for the robot. The wheeled robot constructed is shown in figure 3.6.

Chapter 4

Software

The Software of any autonomous system is analogous to the nervous system of the human body. The software of an Autonomous Vehicle is designed with various constraints such as real-time constraint, security constraints, safety constraints, and others. The AV software must also ensure redundancy for safety critical functions and must be designed to account for any road scenario. For a wheeled robot to drive autonomously in the environment, a software has to be capable of extracting data from the sensors on-board and, based on a decision framework, generate a path to a suitable goal location with regards to safety and comfort. The hardware suite used in the wheeled robot prototype was discussed in chapter 3. In this section, the software design process will be illustrated. The chapter will start with a list of functions the software needs to perform, with their characteristics and design constraints and assumptions. This is followed by a description of the Software architecture and then a detailed description of the various sub-systems that constitute the project.

4.1 System Objective

The software of the wheeled robot must achieve the following operations:

1. A user interface, which can extract navigation command, must be available in the front end.

2. The software must be capable of communicating User input from a remote computer to the robot.
3. The software must be able to read the surroundings using the depth camera and identify the coordinates of the obstacles.
4. The software must identify the lane boundaries.
5. The software must identify horizontal lines, which are the stop lines.
6. The software must be capable of mapping its immediate surroundings in a format that is compatible with Path Planning.
7. The software must be able to decide a maneuver to perform, based on the environment and the presence or absence of user input.
8. The software must be capable of planning a collision-free path to achieve the maneuver.
9. The software must be capable of following the planned path by controlling the motors of the robot.

Constraints

These functions need to be performed considering a few constraints, which are the following:

1. The software must be **Real-Time**. For this purpose, the entire software is programmed in the C++ programming language.
2. Motion Planning updates must be at least **10Hz**.
3. The controller must operate with at least a **100Hz** update rate.
4. The mapping is performed upto a radius of **3 meters** about the ego vehicle.

Assumptions

The software has been designed for a specific set of scenarios to check for the validity of the solution proposed in the thesis. Hence, there are a few assumptions that have been considered to simplify the problem. These are as follows:

- The environment contains only one lane and hence lane change maneuvers are not included in the Finite state machine of the Behavioral Planner.
- The ego vehicle's motion plan is performed only for static obstacles. Hence, it is assumed that there are no pedestrians or other dynamic obstacles in the environment.
- The surface of the road remains constant for the wheeled robot. This implies, only a single set of control gains need to be tuned for the robot.

4.2 Software Architecture

The previous section has laid out the objectives and constraints of the robot's software design. As discussed in 3.5, there are two computation platforms within the robot. One embedded computer is responsible for the control system of the vehicle and the other the perception, mapping and path planning. It is also known that there needs to be a UI on a remote laptop, which accepts human input and communicates with the robot. However, it can be seen that the computation board on the robot, which is less powerful than the remote computer, performs the majority of processing for the software, which could reduce the speed of the complete system. Hence, to make the system real-time, a distributed computing architecture was decided to be applied to increase the processing speed of the complete software.

The Client-Server model has been selected as the distributed computing model. The Client

and Server are computer programs on either the same hardware, or connected by any type of network, that perform the required process by distributing the payload. The working of the Client Server model can be described as follows:

- The Client program performs certain functions and sends out a request message to another program, which is the Server.

- The Server, receives the request from the Client and processes the required data.

- The Server sends out the response to the request, which is received and processed further by the Client.

An example of Client-Server interaction is the process of a web browser loading a webpage. The browser is the client program, which requests the a specific page of a website, through the internet. The Server, receives this request and transmits binary data, which is converted into the required webpage by the browser. The advantage of the client server architecture is higher speed of processing despite lower computation specifications of the client. In our example, the client programs runs on the Jetson NANO, which sends out a request, through wireless communication, which contains data output from the perception system, such as lane and obstacle coordinates. The Server program runs in the remote computer that contains the UI for the robot. The Client-Server software architecture of the Wheeled robot is shown in figure 4.1. The server receives the perception data, creates a map of the surrounding and performs motion planning to generate a safe path, which can be traced by the robot. This path is the response message, which is sent back to the client. The client then transmits this data to the controller board, using serial communication.

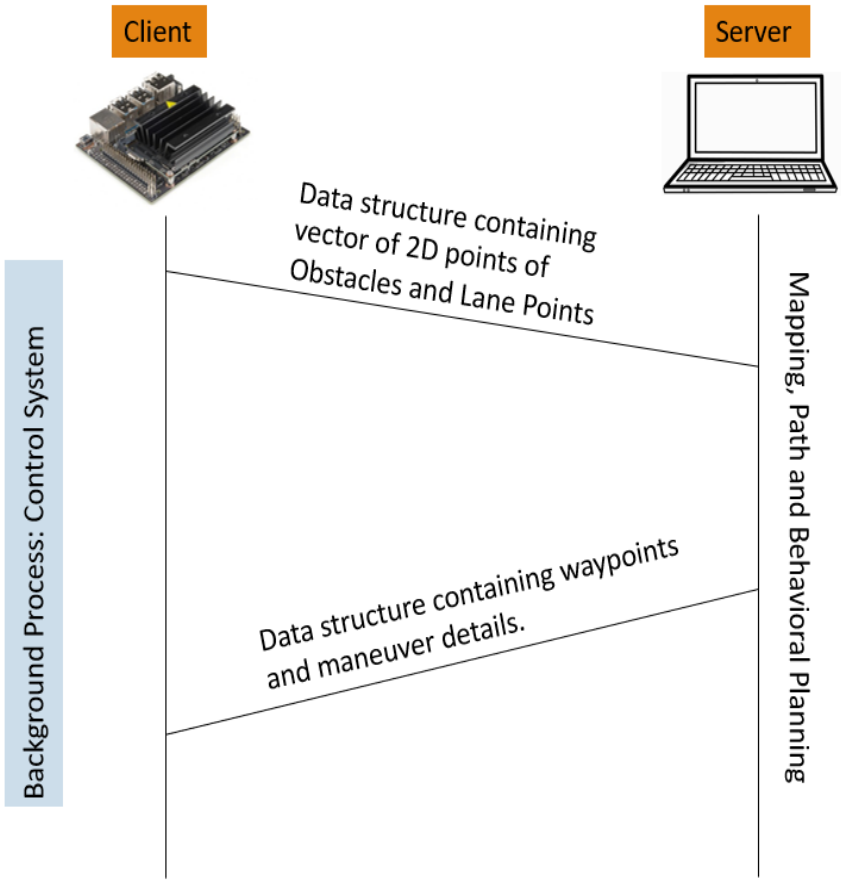

Figure 4.1: The Software Architecture of the wheeled robot describing the computation flow.

4.3 Perception System

The previous section provided an overview of the system's software architecture. The concept of distributed computing was introduced and the various processing centers of the robot were described. We will now discuss the first operation performed by the Jetson Nano(client), which is, perceiving the environment. The perception system of an AV was already introduced in 2.2.1. The major functions of the perception system of the prototype are:

- Identification of static obstacles in the environment.
- Identification of lane boundaries.

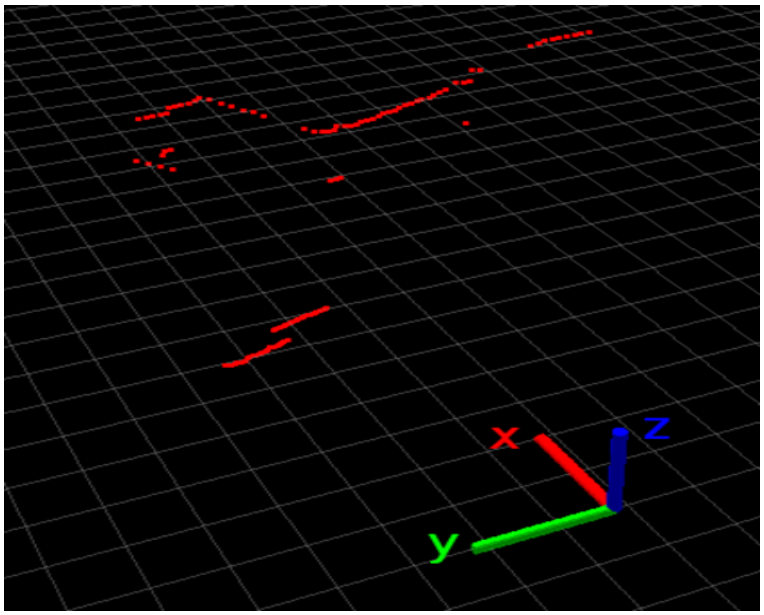

Figure 4.2: Visualization of obstacles(in red) post conversion into vehicle coordinate frame

- Identification of stop lines and other horizontal lane lines.
- Estimation of the position of the above points in the vehicle coordinate system.

All the above functions need to be performed using the RGB-D data from the Intel D435 depth camera, using various image processing techniques. The software utilizes Realsense SDK, to obtain the filtered RGB data and the OpenCV Library [6], which is an open-source computer vision library, to extract the visual features of the image. The Realsense library allows us to extract depth and color frames separately. The first step would be to align the depth and color images, such that, for every pixel in the color frame, we get a depth value.

4.3.1 Obstacle Detection

Once the color and depth frames are aligned, we can extract the depth of every pixel on the color frame. Obstacle identification is only limited to the 2D XY plane of the robot for simplicity. Using the aligned frame, we can extract the range or depth of a pixel. We

Figure 4.3: Lane Detection test image, courtesy of [reference]

can then use the intrinsic parameters to obtain the spatial coordinates of the pixel in the camera coordinate frame. These camera frame coordinates are transformed into 2D Spatial coordinates in the vehicle frame, using the extrinsic parameters. The intrinsic parameters can be obtained by a simple call of a function in the realsense library. Using this, we can de-project the pixel into points in space, whose output is visualized in figure 4.2.

4.3.2 Lane Detection

Once the static obstacles are recognized, algorithms can be created, that can maneuver the wheeled robot in a collision free manner. However, for an AV, the drivable space does not only depend on the static obstacles, but also the location of the lanes. Hence,it is essential for the software to identify the lanes, to safely guide the robot towards the destination. In the simplified prototype, there exists only one set of lanes. The working of the Lane detection algorithm is explained on the test image shown in figure 4.3. Lane Detection in

AVs has been accomplished using standard computer vision algorithms as well as application of deep learning. Supervised Learning methods exist [74] [10] [47], that identify lanes whilst accounting for occlusion and illumination changes. However, the prototype operates in a standard environment and does not experience scenarios of occlusion. Therefore, we have not employed the computationally heavy deep learning based lane detection algorithms for the wheeled robot.

The human brain identifies lanes, by filtering specific colors(white and yellow) in the road background. This process of isolating colors from any color space is known as color thresholding. In the prototype, we will perform Binary thresholding on an RGB image, to isolate white pixels from the image. Post thresholding, straight lines can be identified using hough transform. However, roadways generally have curved lanes. Hough transform based lane detection proves extremely inaccurate at high curvatures. Hence, in our approach, we will be using Sliding window search [40] to obtain lane coordinates, and fit these coordinates to obtain a 2nd degree polynomial denoting the position of the left and right lane.

Thresholding

Color thresholding can be performed in various color spaces. Color space of an image is a set of parameters, that define the color of a pixel of an image. Red Blue Green (RGB) is the default color space for images. Examples of other color spaces include HSV (Hue Saturation Value), HSL(Hue Saturaltion Lightness), LAB(Lightness, A-channel, B-channel), to name a few. The intensity of every channel in a color space is superimposed to generate the image recorded by the camera. Example of intensities of the RGB and HSV color spaces for the test image in figure 4.3, is shown in figure 4.4. Images can be converted from one color space to the other using the OpenCV library. To isolate a color in any color space, we select a range of values in each channel. If a pixel's intensities in all three channels are

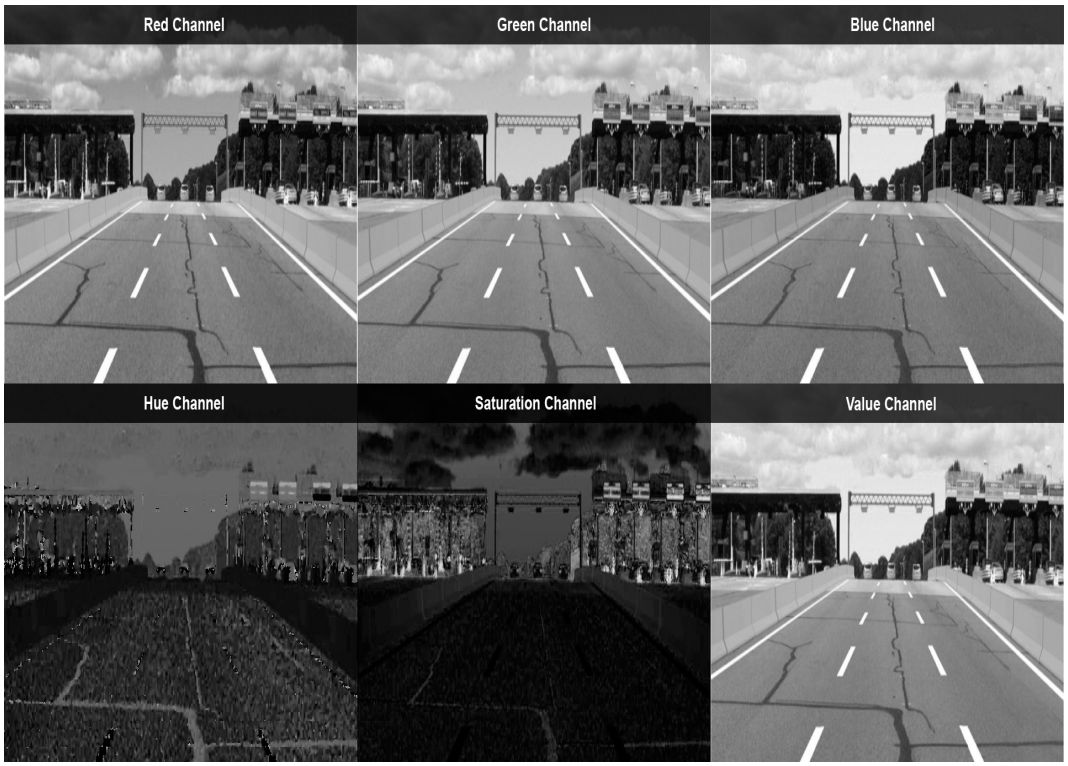

Figure 4.4: Visualization of intensities of individual channels of RGB and HSV color spaces for the test image

within their respective threshold range, the pixel gets a value 1, while the value is assigned 0 otherwise. This is known as Binary thresholding. As an example, to isolate the color white in RGB channel, that use 8bit encoding(pixel intensities range between 0 and 255), we issue a lower bound of pixel values in the R,G,B channels as (240,240,240) and higher bound as (255,255,255). Any pixel that satisfies the criteria is classified as a white pixel and is classified as black otherwise. Selection of color space for thresholding depends on the application and the lighting conditions. However, for experimental conditions, and due to the real-time constraint, we opt to threshold in the RGB space. This also saves computation potentially performed to convert the RGB image into other color spaces.

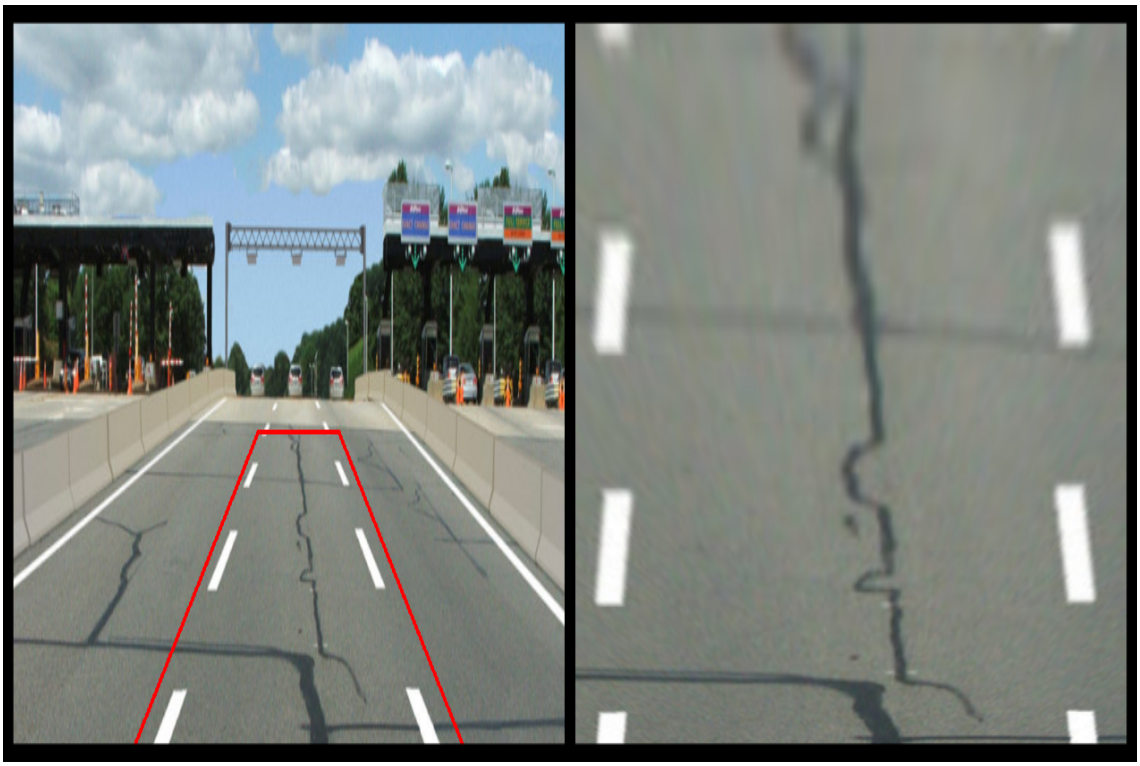

Figure 4.5: Images showing the Region of Interest and the Birds Eye View obtained after IPM

Birds Eye View

Based on literature review, it is noted that it is simpler to identify lanes on a top view of the road surface, rather than the normal view of the camera. This is because straight lines on the road appear as sloped lines in the default image, thus causing errors in terms of identifying certain pixels. Due to this reason, every road feature identification system in AVs, such as lanes, horizontal lines, left only turn, and others are identified using a warped top view of the road surface. This is known as the Birds eye view(BEV) and the process of warping the Region of Interest(ROI) of the road, into a different perspective is known as Inverse Perspective Mapping(IPM). OpenCV library contains a set of functions to perform the mapping and its inverse. Using these functions, we can transform the default view image to the Birds eye view and vice versa. Hence, lane points can be identified in the Birds Eye

View and can be converted into pixel coordinates in the original RGB frame. Once the pixel coordinates of the lane points are known, their coordinates in the vehicle frame can be calculated using homogeneous transformation. The left image of figure 4.5, is the Region of interest for lane detection for the test image, and the right image shows the mapped Birds Eye View of the Region of Interest.

Sliding Window Search

Once the ROI is converted into the BEV, using IPM and binary thresholding is achieved to filter white pixels, further de-noising functions are applied such as Gaussian smoothing, erosion and dilution. Histogram analysis is performed for a sub-image of the processed BEV image, where we can identify two peaks, which are the left and lane points respectively. We store the left and right lane points in separate arrays. Then we perform a sliding window search across the processed BEV image, by repeating the histogram analysis to obtain the two peaks, on every window and appending the points into the respective arrays. Thus at the end of the sliding window search, we have separate arrays of pixel coordinates of left and right lane points.

Conversion to Spatial Coordinates

The lane points array contains the pixel coordinates of lanes in the Birds eye view frame. We invert these pixels to the default view and then transform those pixels into spatial points in the vehicle coordinate frame. These points are fit into a second order polynomial, and N equally spaced points(12 in our case) are sampled and are used in the path planning problem. The thresholded Birds Eye View and the main image frame with the Lane Points sampled from the function obtained by curve fitting are shown in figure 4.6.

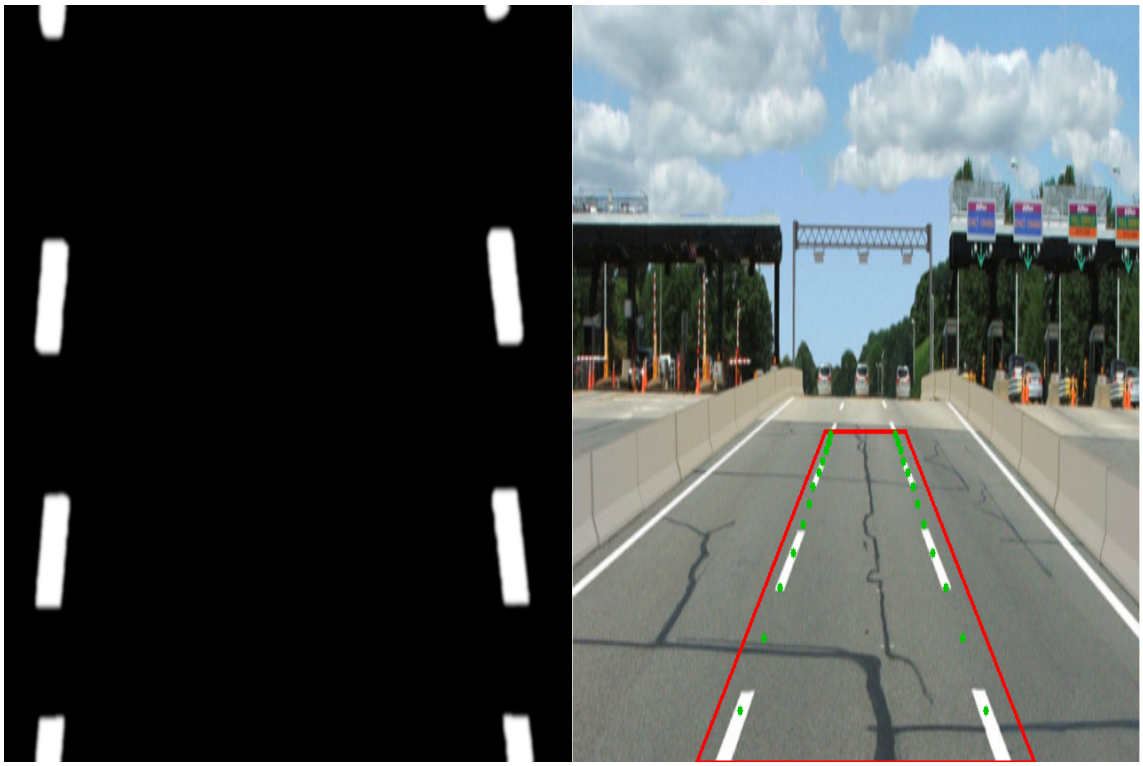

Figure 4.6: The thresholded BEV(left) and detected lane points(in green) in the original image frame(right)

4.4 Mapping

The process of identification of lanes and static obstacles in the environment have been discussed thus far. Knowledge of these coordinates, must be converted into a map to identify the free space of the vehicle. In 2.2.8, we defined the configuration space as the set of reachable states of a vehicle. In terms of a skid steering robot, the configuration space is $\mathbb{R}^3$, which implies, the robot can achieve any position and orientation in any coordinate space, using appropriate control algorithms. The free space is a subset of the configuration space, which are free of obstacles. Hence, mapping is the process of updating the free space for the planning problem, in a data structure, which is compatible with the path planning algorithm. Various mapping data structures exist for Autonomous Vehicles. HD maps are multilayered

maps, used in AVs that contain layers depicting lanes, road signs, and other static obstacles in the environment in 3D, which are constructed by 3D reconstruction techniques and border scanning algorithms. These are computationally heavy for the prototype.

4.4.1 Occupancy Grid

Wheeled Robots generally employ occupancy grids as maps for planning. Occupancy grid is a discretized grid of cells that contain information about a specific point in space. a 2D occupancy grid, as shown in figure, is a grid of 2D points in space. Each cell holds an x and y value, as well as information regarding whether the point is an obstacle, or free. There are two forms of occupancy grids:

- **Binary Occupancy grid** - A grid wherein cells hold either a 0, indicating free, or 1, indicating obstacle.

- **Probabilistic Occupancy grid** - A grid with each cell characterized by a value between 0 and 1, which indicates the probability that the corresponding cell is an obstacle.

Probabilistic Occupancy grids account for uncertainty in information regarding a cell, and updates values based on Bayes rule, known as the Bayesian Log-odds update. These systems identify common points in multiple frames, thus adjusting for the probability of the presence of an obstacle at a node in the grid. Our system works completely memory less, without account for the surroundings in the previous time step. Therefore, binary occupancy grid is constructed in real-time in the robot software, using coordinates of the obstacles and lanes obtained from the perception system.

Formation of Occupancy grid

A class object Node contains information regarding the coordinates of the centroid and end points of every discretized cell of a grid. Then we create a 2D matrix of Node objects. The number of rows and columns of the grid depends on the discretization resolution(width and height of every Node) and the planning range, which is the maximum distance in x and y axis for which the planning is performed. Once a 2D matrix of Node Objects is created and updated with appropriate coordinates for each Node, the obstacles are updated to the grid in O(n) time complexity. This form of 2D binary occupancy grids, can be employed in path planning using sampling based methods and lattice planners.

4.5 Motion Planning

Once an occupancy grid is created, the planning module estimates a safe maneuver, based on a decision framework, and calculates a safe and dynamically feasible path that can be traced by the vehicle. As discussed in 2.2.5, the planning module of an AV is composed of an hierarchical structure. Figure 4.7 shows the comparison of the system proposed in the thesis to that of a typical AV motion planning module. The mission planner is removed from the system, and a human input block is inserted, which enables the coordinate frame reset, and hence the removal of localization system from the planning framework. The working of the stipulated model is composed of the following steps:

- The Behavioral planner accepts user inputs(if present) and perception inputs and selects safe maneuver based on a decision framework programmed.
- The Path planner generates spatial waypoints that form a collision free path to achieve maneuver decided by the Behavioral Planner.

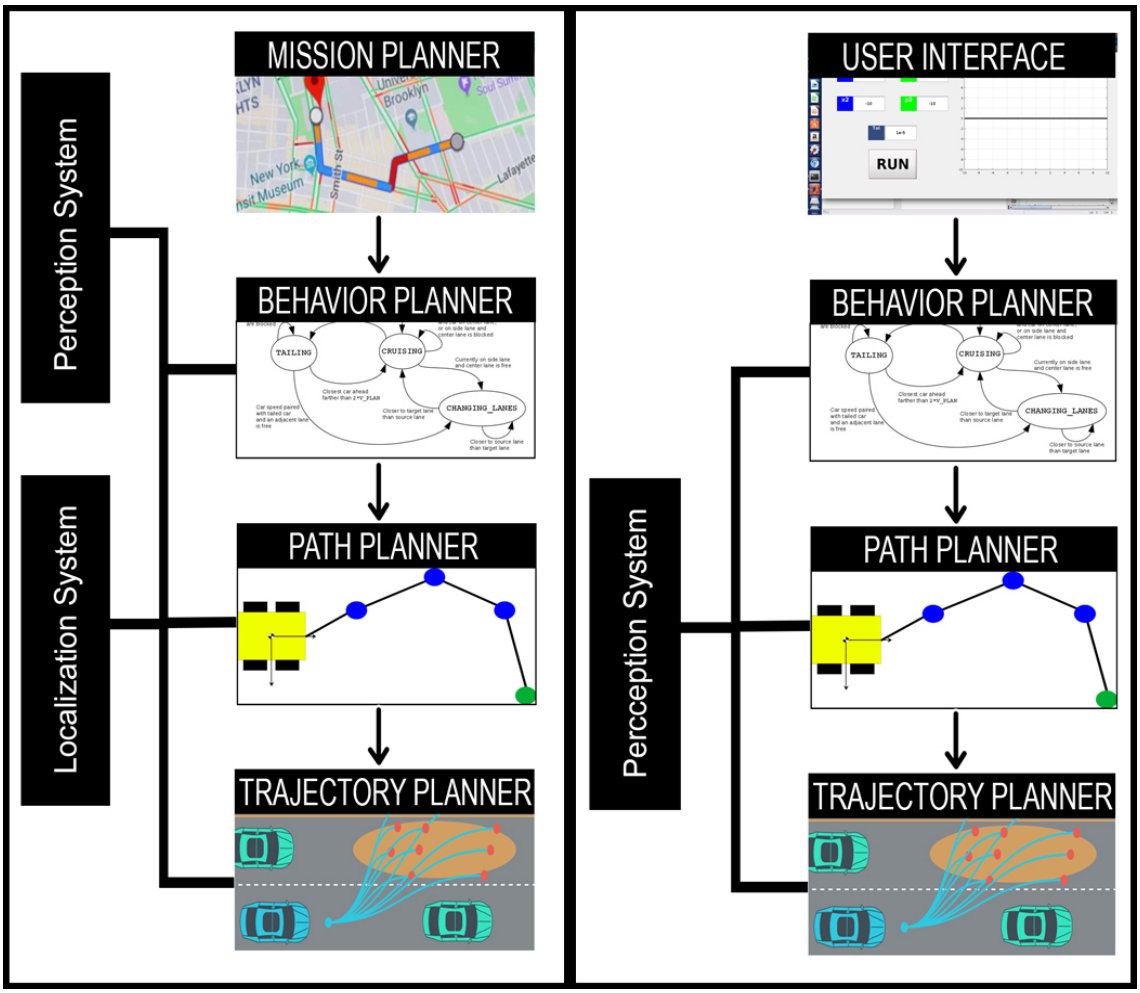

Figure 4.7: A typical Hierarchical motion planning(left) module of an AV, and the proposed Motion Planning Module(right), with User inputs inclusion into Planning framework

- The Trajectory planner solves an optimization problem to generate spline functions of the position and velocity of a vehicle, that trace the waypoints, satisfy the boundary conditions, satisfy auxiliary dynamic constraints, and avoids collision from obstacles.

The following subsections will describe each individual planning problems and proposed solutions.

4.5.1 Behavioral Planning

An autonomous vehicle must be capable of performing safe maneuvers, that comply with the rules of the road. It must abstain from taking a lane, which is intended for traffic on the other way, as well as be able to only perform a left turn at a left only lane. Hence, a framework has to be maintained, based on which, safe and legal maneuvers can be selected for every possible scenario. the framework is dependant on the environment around the ego vehicle, which includes lane boundaries, road sign units(RSU), such as speed limit boards and stop signs, static obstacles present in the road, as well as the dynamic obstacles in traffic, as well as the rules of the road, as mentioned above. This task is performed by the Behavioral planner in the motion planning hierarchy. In a typical Autonomous vehicle, goal waypoints are provided by the mission planner, towards which the vehicle is tasked to travel. In these systems, maps are either pre-built, or an incremental online map is created in real time by SLAM, upon which an optimization of an MDP is performed to generate an action, that satisfies the safety and legal constraints. In our system, however, an additional element is added to the decision framework, which is the navigation input by the user. The software must be able to select maneuvers that is intended by the passenger, but also must be capable of ignoring navigation inputs, that are either not safe, or not compliant to the road rules. An example of this would be asking the vehicle to take right in a left only lane.

For the prototype, the Behavioral Planner is constructed with a Finite State Machine(FSM). These planners are also known as rule based planners, and are simple to construct and execute. It is similar to a combination of ”if then else” conditions. The FSM consists of a set of states, which are maneuvers, and actions represent change from one state to another, which is calculated based on a transition law. Generally, FSMs use simplified models to simulate motion to a set of possible actions for a time horizon t_h, and selects the maneuver, which is safe as well as enables the vehicle to reach closer to the goal during the time

horizon. However, in our system, there is no motion goal in certain scenarios, due to absence of human input. In such cases, default actions need to be defined, that is dependant on the environment as well as the previous state. The states considered in the simplified prototype are:

- Follow a lane.
- Stop at Stop Line.
- Proceed straight at the intersection
- Turn Right at intersection
- Turn Left at intersection

The FSM is visualized in figure 4.8. The states are the ellipses and the connections between them are actions, which are selected based on Transitions. There is a navigation command buffer, which maintains the most recent navigation input from the user, which is also considered in the planning. The transitions [T1]-[T8], as shown in 4.8, are explained briefly below:

- **T1 - Start to Maintain Lane** : The start is the program that runs at the startup of the vehicle. Maintain Lane state can be activated by 4 actions. The state shifts to Maintain Lane, by T1, when the startup is complete and the current lane is identified. The FSM sends the centerpoint between the lanes at the planning distance as the destination.
- **T2 - Maintain Lane to Stop** : The Stop maneuver is activated only when a horizontal stop line is detected in a lane. In such a case, the centerpoint of the lane at the

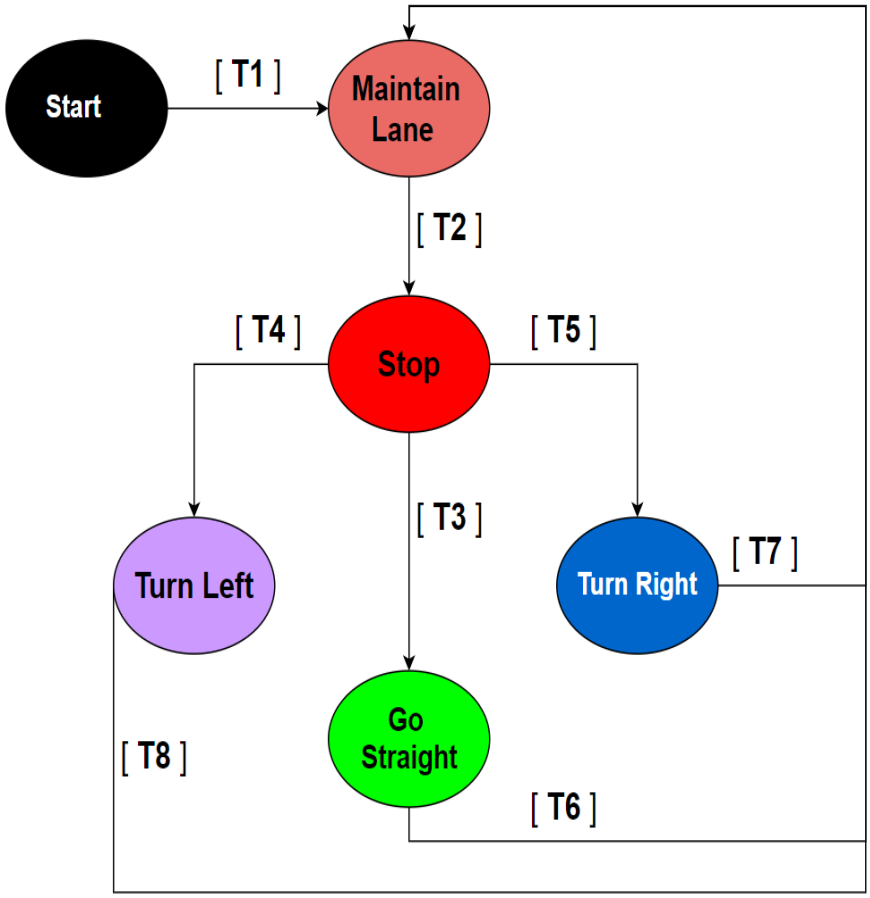

Figure 4.8: FSM used in the Prototype

stop line is sent as destination and a stop line flag is sent to the trajectory planner, to plan a trajectory with velocity at goal as 0 m/s.

- **T3 - Stop to Go Straight** : The Go Straight state can only be activated, when the previous state is a stop state, and if the most recent user command is ”go straight”, or no navigation command is present in buffer, and a straight road exists.

- **T4 - Stop to Turn Left** : The Turn Left state is activated only when the previous state is the stop state, and the horizontal lane detector, detects the intersection lanes, and the navigation command on the buffer is ”turn left”. T4 also switches to turn left,

when horizontal lanes are detected and there is no command on the buffer, and there are no lanes to the right.

- **T5 - Stop to Turn Right** : The Turn Right state is activated only when the previous state is the stop state, and the horizontal lane detector, detects the intersection lanes, and the navigation command on the buffer is "turn right". T5 also switches to turn left, when horizontal lanes are detected and there is no command on the buffer, and there is no road straight ahead.

- **T6 - Go Straight to Follow Lane** : Once the vehicle has exited the intersection, the Maintain Lane state is activated and hence the centerpoint between the lanes at the planning distance is sent as destination to the path planner.

- **T7 - Turn Left to Follow Lane** : Once the vehicle has executed the left turn, the Maintain Lane state is activated and hence the centerpoint between the lanes at the planning distance is sent as destination to the path planner.

- **T8 - Turn Right to Follow Lane** : Once the vehicle has executed the right turn, the Maintain Lane state is activated and hence the centerpoint between the lanes at the planning distance is sent as destination to the path planner.

These states are simplified to check if such a system with coordinate frame reset is feasible. For implementation with AVs, more complex scenarios are to be appended to the FSM, such as change to left lane, Emergency stop, Right turn at a Green Light, etc. However, the base framework is similar to that constructed in the prototype.

4.5.2 Path Planning

Once a maneuver is selected, the behavioral planner outputs the destination of the local planning problem. This destination vertex is the goal location of the path planning problem. The occupancy grid, which contains information regarding location of obstacles in the configuration space, is also an input to the path planner. As discussed in section 2.2.8, the path planner is tasked at finding collision free intermediate configurations(waypoints) that the robot must follow to reach the goal location. There are several approaches taken in the literature to solve the planning problem. Path planning solutions for AVs are broadly classified into three methods:

- Incremental search methods
- Graph Search Methods
- Optimization Based Planners
- Sampling Based Planners

Variational Methods

Variational methods involve simulating a template of control inputs on a motion model to obtain candidate paths for a planning time horizon. A path is selected of these candidate paths based on a cost function and collision constraints. Typical cost functions award close distance to the goal location at the end of the time horizon and penalize high curvature. The receding horizon path planner is a notable example of this type. These solutions are computationally efficient and provide dynamically feasible paths. However, in complex environments, Variational methods provide poor quality paths as they account for obstacles only until the time horizon.

Graph Search Based Methods

As discussed in 2.2.6, a graph is a set of vertices and edges that denote a mathematical relationship. Graph Search based planners directly operate on occupancy grids obtained from the mapping module. Adjacent free Nodes are assumed connected with an edge, whose weight is the euclidian distance between the two Nodes. Graph search methods like Dijkstra, A*, D*, Hybrid A* among others identify the shortest path in terms of weight between start and the goal Node of the Occupancy grid. Graph Search methods are computationally tractable. However, the paths obtained require post processing to generate a smooth path.

Optimization Based Planners

Optimization based planners solve a non-linear optimization problem at every time step. The NLP is the minimization of a certain cost function, with the control parameters, which are the variables whose values are altered to get desired results, being coefficients of parametric curves. Parametric curves used in motion planning include Bevier curves, cubic and quintic splines, cubic spirals, clothoids, etc.,. The optimizer changes control parameters to minimize the cost function, whilst also satisfying all equality and inequality constraints. These include boundary constraints, curvature constraints, collision check, and others, depending on the application. The Optimization based planners provide accurate paths and can be directly used by a controller, without post processing. However, they are computationally expensive and potentially could provide local optimum solutions, rather than global optimum.

Sampling Based Planners

Sampling based planners randomly sample points in the configuration space and check for the connectivity and add to paths. These planners are computationally efficient and can be

programmed to provide paths, that are smooth, less varying and collision free. Sampling based planners are more prominent in Autonomous Vehicles as they are more suitable for realtime performance, and provide better paths, as compared to deterministic methods like incremental search and graph search methods. The commonly practiced Path Planning algorithms for autonomous vehicles are listed in table 4.1. We will run two path planning algorithms in parallel in the proposed system, which are the PRM, coupled with an A* algorithm and the RRT* algorithm.

Table 4.1: Local Path Planning survey

Algorithm	**Ref**	**Pros**	**Cons**
D* Algorithm	[1]	Fast Re-planning for multiple Queries	Requires Post processing
Hybrid A*	[15]	Utilizes vehicle model for generation of feasible paths	Does not guarantee least cost solution
Lattice tree + Dijkstra	[31]	Computationally Efficient	Does not provide complete solution.
PRM + Dijkstra	[39]	Probabilistically complete solution	Complete Path may not be explored if Parameters are not selected properly.
RRT	[42]	Probabilistically Complete Solution	Provides Jerky Paths.
RRT*	[37]	Provides smooth paths and system is probabilistically complete	Algorithm does not provide information regarding unavailability of path.

4.5.3 Probabilistic Road Mapping

Graph Search methods that work on an occupancy grid tend to provide paths that are noisy. This is tackled by the PRM path planning algorithm. Probabilistic Road Mapping is the process of creating a sparse graph from an occupancy grid, using which a graph search

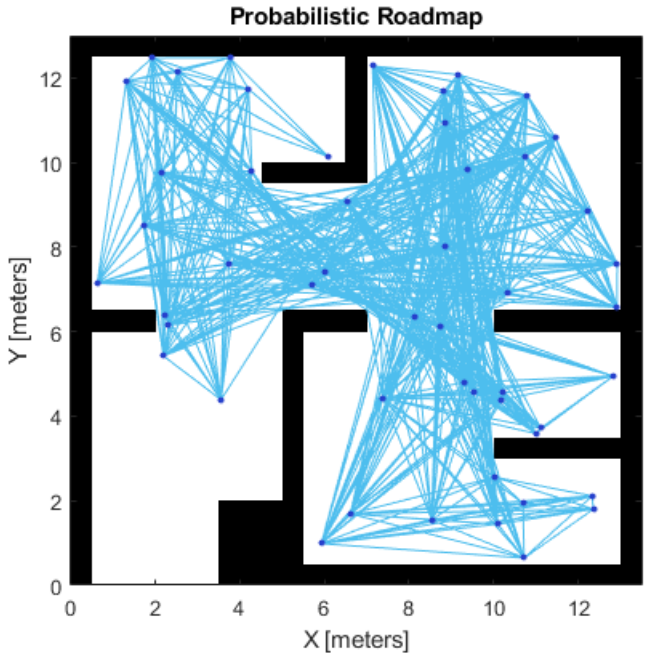

Figure 4.9: A sample graph constructed using PRM by sampling 50 points

algorithm can find the shortest cost map from the start vertex to the goal vertex.A sample graph created using PRM algorithm, is shown in figure 4.9. Given an occupancy grid, an empty graph, in the form of an adjacency list is constructed. The adjacency list is a vector, that describes the connectivity of every vertex in a graph. The PRM algorithm employed in the prototype software follows the following steps:

- Create an empty graph G.
- Iterate loop 'N' times.
  - Sample a random Node 'V
  - If 'V' is located at an obstacle, end loop.
  - If 'V' is not an obstacle, add 'V' to the Graph.
  - Identify the 'k' nearest neighbours in the graph, within a radius 'r', based on L2 distance.

  - Check for collision for potential edges between V and 'k' that are within edge radius.
  - Add eligible edges to the graph G.

- Once the graph is constructed, add the source and destination to the Graph.
- Perform Graph Search to obtain path between source and destination using the graph.

The PRM algorithm provides a probabilistic complete solution. This implies that given enough samples, if a path exists within two vertices, it will be detected by the PRM planner. The performance of the PRM planner depends on the following parameters:

- The total number of samples - **N**.
- The edge radius - **R**.
- Number of nearest neighbours considered - **k**.
- The collision check algorithm.
- The distance check metric (L2 in our case).

These parameters were tuned based on the nature of the paths and computational time for generating paths. Higher the number of samples 'N', higher is the configuration space that is explored by the planner. Larger values of k and N increase the performance of the planner. However, they also increase computational time. Selecting a low edge radius 'r' could cause the algorithm to skip a large number of edges. However, too high of an 'r' value could cause graphs to lose edges more often as they fail the collision check function.

Collision Check Algorithm

The collision check algorithm of the PRM planner, checks for the presence of obstacles between a sampled vertex, and its neighbour vertex. The collision check algorithm designed for this application, generates the equation of a straight line between the two vertices. It then samples 'M' equally spaced points(5 in the tuned application) in the line segment between the two vertices and checks if the distance between the vertex and any obstacle is twice the radius of the circumcircle of the robot. Let d be the distance between an obstacle and the interim point and r_1 be the radius of the circumcircle of the robot. This implies if, $d \leq (2{*}r_1)$, then a collision is possible between the two vertices, and does not exist otherwise.

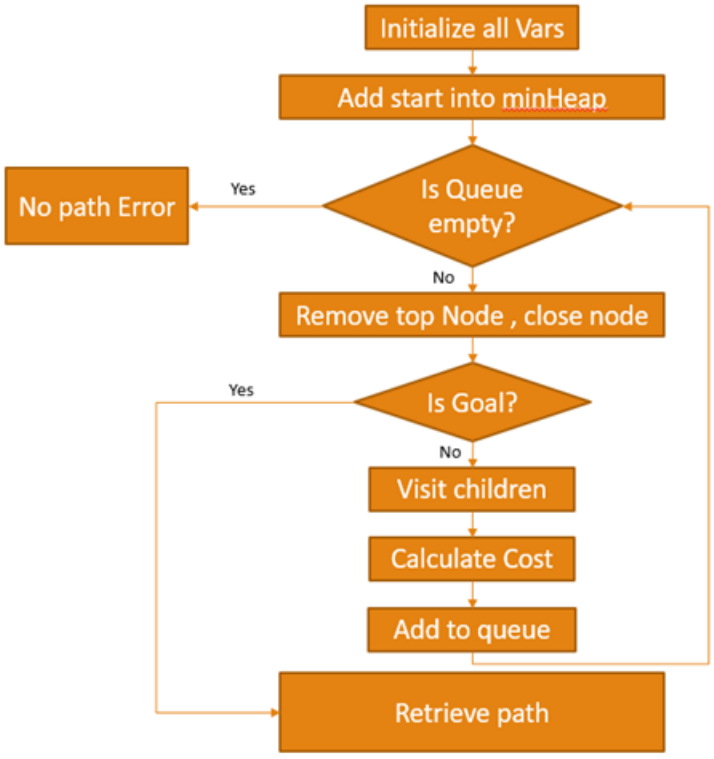

Figure 4.10: Flowchart of A* Algorithm used in the prototype

4.5.4 A* Search

Once a graph is constructed by the PRM algorithm, any graph search algorithm can be used to detect a path between the start and goal vertex, if a path exists. The A* search is employed in this project, as it detects the path based on a heuristic, which can be set by the User. The A* algorithm employed in the software for the prototype is shown in figure 4.10. The heuristic cost function selected for this function is the distance between the visited vertex and the goal vertex. Due to this heuristic, every new node visited by the A* algorithm, within its loop, will be towards the goal location. This implies in a sparse environment, a path between the start and goal vertex is found quicker than the traditional Dijkstra graph search algorithm.

4.5.5 RRT*

The PRM based planners search the sample space for N iterations before searching for a path. This is useful in conditions where the same map can be used to solve multiple queries. However, faster sampling based solution can be achieved, if we create the graph, whilst checking if a path is complete between the start and goal path. The Rapidly Exploring Random Trees (RRT) algorithm generates a tree instead of a graph, between the source and the destination. A tree is a special form of a graph, where any node can have a maximum of one parent. Thus, if the RRT algorithm, which starts exploring from the start vertex, reaches a goal vertex, a path can be traced by checking the parent of every node from the goal vertex, upto the start. Since RRT is a sampling based planner, it also provides a probabilistic complete solution. The RRT algorithm is better suited for real-time applications, as paths are generated with lesser samples in an environment with a sparse density of obstacles.

There exist multiple variants of the RRT algorithm, that increase the speed and quality of

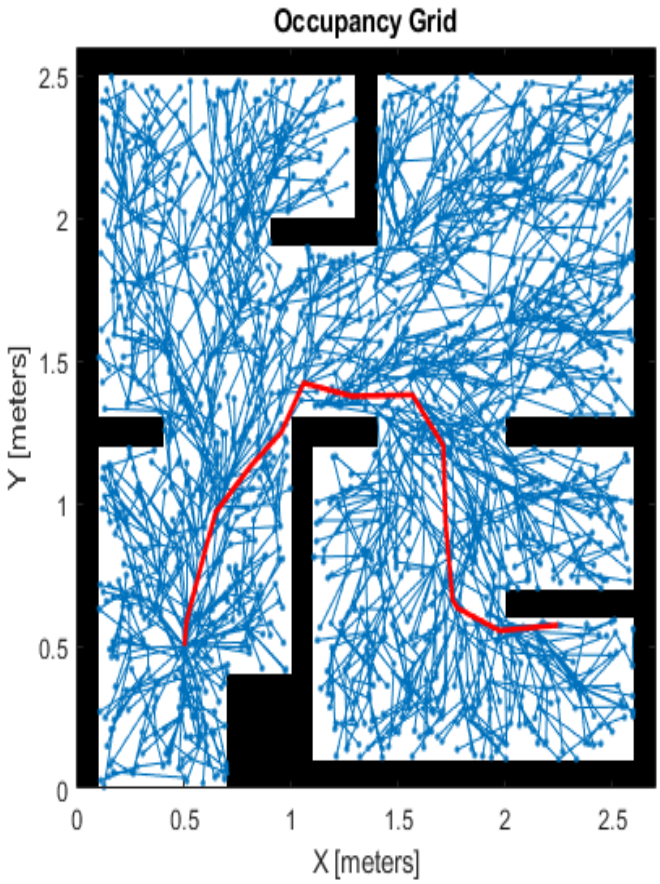

Figure 4.11: A path generated by the RRT* application on a sample map

the path. RRT algorithm tends to produce cubic paths, since nodes are connected only based on their distance and hence, paths with steep turns could be selected as they are technically the shortest path between the start and goal vertices. The RRT* algorithm is an example of such an addition. The RRT* algorithm follows the same sampling procedure of the basic RRT algorithm. However, at every time step, it performs corrections on the generated path to every vertex within the neighbourhood of a sampled point, thus generating the least cost path towards every node in the neighborhood of the sampled node. A sample path generated by RRT* is shown in figure 4.11. These paths generated by RRT* are smoother than conventional graph search algorithms, due to the correction process performed at every time step.

The RRT* algorithm depends on the following parameters:

- Maximum number of points to be sampled - N
- Cost Heuristic
- Distance threshold - δ
- Goal Radius - r
- Collision Check Algorithm

Parameters

The PRM algorithm samples N points, on completion of which a path is searched. The RRT* algorithm searches for a path at every sampling iteration and the algorithm stops when the goal region is reached. However, if no path exists, the algorithm must be able to return the no path flag to the trajectory planner. Therefore, the maximum number of sample points 'N' must be carefully selected. If the value selected is too low, the algorithm ends premature before a path is obtained, and too high a value implies waste of computation in a no solution condition. Proper selection of cost heuristic would ensure smooth path generation by the algorithm. The cost function must penalize larger angle of turn, rather than larger distance. The distance threshold is similar to the edge radius parameter of PRM planner. It is the maximum distance of connectivity between two nodes. The goal radius defines a region about the goal vertex. The algorithm ends when a point in the region is sampled. The collision check algorithm is the same as that of the PRM collision check algorithm.

The RRT* algorithm for the robot prototype is as follows:

- Create empty tree and add the source vertex into tree.

- Define the goal region and δ.

- Loop N times

  - Sample a point and check for its nearest neighbour.

  - If the sampled point is within δ distance and is collision free from the nearest node, add to tree.

  - If distance is greater than δ, then an intermediate point is selected in the line between the sampled point and the nearest neighbour. If the path between the nearest node and intermediate point is collision free, add point to the tree.

  - If no intermediate points are collision free, then disregard the sampled point.

  - If a point is added to the tree, a check is performed on every node in the neighbourhood of the new node to check if a collision free path, with lesser cost exists to a node already on the tree, from the newly added node.

  - If such a path exists, the tree's connectivity is edited.

- If the most recent node is in the goal region, generate the path by backtracking the parents of nodes from goal region to the start vertex.

The RRT* algorithm tends to find smooth paths towards to goal, while the PRM finds any path that is available. Thus running these sampling based systems in parallel threads with a stop condition can save time for most situations of the prototype, as there is no clutter of obstacles in the environment of the wheeled robot. Thus, through the application of PRM and RRT* in parallel, we obtain 2D waypoints that are sent to the trajectory planner.

4.5.6 Trajectory Planning

Optimization based path planning algorithms works only based on achieving a safe path while satisfying the constraints on the NLP. These problems take longer time to converge due to the high non-linearity of the problem. However, once a path is formed in terms of waypoints, the curve fitting could be handled as a Non-Linear Program, where the error is the mean square difference between the waypoints and the simulated position of the vehicle. The NLP problem is formulated below:

Let X_{des} be the vector of waypoints in space, which include both the x and y coordinates of the waypoints. Let α be the path variable, which ranges from 0 to 1, where $\alpha = 0$ is the start node and $\alpha = 1$, the destination node. Thus, the objective of the trajectory planning is to find $X(\alpha)$, a collision free trajectory that minimizes a cost function and satisfies the constraints. In the robot problem, we do not have any curvature constraints and hence we fit the waypoints to a cubic polynomial. Let

$$X(\alpha) = p_0 + p_1 * \alpha + p_2 * \alpha^2 + p_3 * \alpha^3 \tag{4.1}$$

Thus, the optimization problem is as follows:

$$\min_{p_0,p_1,p_2,p_3} \quad \int_0^1 (X - X_{des})^T (X - X_{des}) d\alpha \tag{4.2}$$

The minimization is subject to the following constraints:

- The kinematics of the system $\dot{X}$.
- The saturation limit of the input, the PWM signal limit, which is between 0 and 255.
- The boundary conditions, $X(0) = X_{start}$ and $X(1) = X_{goal}$.

Other constraints can be used to tighten the optimization problem such as curvature constraints, acceleration constraints, among others. After this step, the function obtained is time scaled to obtain a trajectory with respect to time, which is passed to the controller. The optimization is run on the IPOPT solver in C++.

4.6 Trajectory Tracking controller

Once the trajectory is created, based on the maneuver selected from the behavioral planner in the remote computer, it is sent to the robot computer for tracking the trajectory. The Jetson Nano communicates the updated waypoints to the controller through serial communication. The controller, thus needs to calculate the steering and throttle/brake inputs to follow the required trajectory. Various control strategies exist for tracking trajectories. LQR based linear controllers are used in systems with linear behavior for tracking controllers. Model Predictive Control(MPC) is an optimal control strategy for Multiple Input Multiple Output(MIMO) systems. These systems generate open loop control for a time horizon, based on the solution of an optimization problem. However, these systems require high fidelity models, and these systems are computationally expensive. Hence, for the robot, individual linear and lateral controllers are designed.

4.6.1 Skid Steering Model

A kinematic model of the robot is necessary for designing the controller of the vehicle. AVs use the kinematic bicycle model for high level controllers. However, controllers with high fidelity models are used in the low level controllers to guide the vehicle towards the reference. The wheeled robot prototype, uses a skid steering mechanism. A kinematic model for such

a system is required to generate the control law for the lateral controller. An abstract kinematic model for skid steering mechanism, which is employed in the robot is discussed in this section.

Consider a skid steering vehicle, similar to figure 3.2. The velocity of the left wheel is v_L and that of the right wheel is v_R. It can be assumed that $v_L = r_L * \omega_L$, and $v_R = r_R * \omega_R$, where r_L, r_R are the radii of the left and right wheels respectively. Let v be the velocity of the vehicle. The state of the robot consists of three parameters, namely (x,y,θ). There are two forms of interpretation of inputs to the system. One form is (v_L, v_R) and the other is $(v, \Delta v)$. The model for the system employs the latter. Let x and y be the displacements along the longitudinal and lateral axis, and θ be the orientation of the vehicle. $\dot{x}, \dot{y}$ and $\dot{\theta}$ are the time derivatives of the respective state variables. L is the wheelbase of the robot. The equations of motion are :

$$\dot{x} = vcos\theta \tag{4.3}$$

$$\dot{y} = vsin\theta \tag{4.4}$$

$$\dot{\theta} = \frac{\Delta v}{L} \tag{4.5}$$

4.6.2 Longitudinal Control

The equations of motion that approximate the kinematics of the skid steering robot have been discussed. We deconstruct the control system into two separate controllers to control longitudinal and lateral motion. Longitudinal control for the wheeled robot is a standard PID controller and the longitudinal error is calculated as follows:

$$error_{longitudinal} = v_{des} - (\dot{x}cos(\theta) + \dot{y}sin(\theta)) \tag{4.6}$$

, where $\dot{x}$ and $\dot{y}$ are the velocities along x and y axis of the robot frame respectively. $\dot{x}$ and $\dot{y}$ are obtained from the localization module, using which the error is calculated for a point at a lookahead distance, and appropriate control action is selected, based on the appropriately selected control gains, namely K_p, K_i, and K_d. This control action thus calculated is converted into PWM signals for the motors.

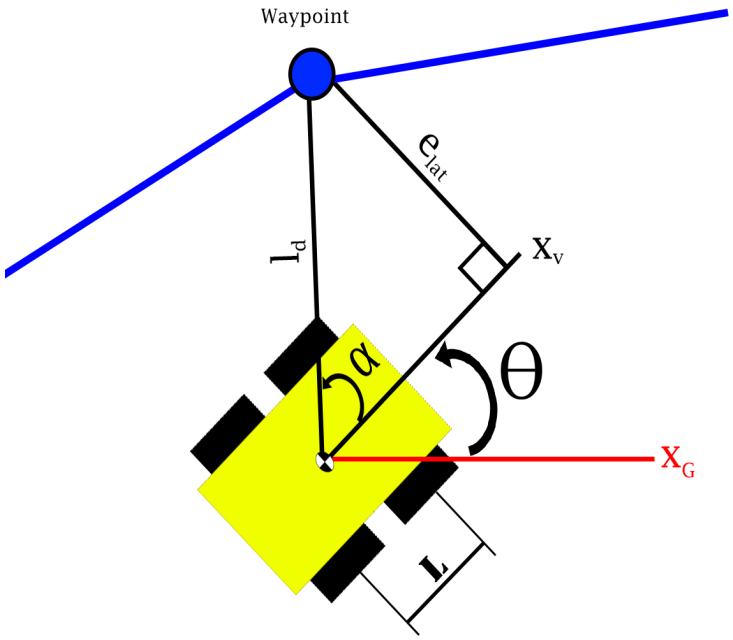

Figure 4.12: Pure Pursuit Description for Skid Steering Robot

4.6.3 Lateral Controller

The lateral controller selected for this application is the Pure pursuit controller. The pure pursuit controller is computationally efficient, and behaves similar to proportional control system. The system diagram , shown in figure 4.12, indicates the current position of the vehicle and the reference point at a lookahead distance l_d. X_v is the vehicle frame and

X_G is the Global Frame. The angular difference between reference point and the current orientation is α. The intended radius of curvature is R, wheelbase is L, and e_{lat} is the cross track error. Once we calculate α, we can generate the steering angle δ, using the relation,

$$\delta = \tan^{-1}(\frac{2L\sin\alpha}{l_d}) \tag{4.7}$$

However, we also know that $\dot{\theta} = \delta$. Thus adding a gain K_{lat}, we get:

$$\Delta v = K_{lat}.\delta \tag{4.8}$$

Thus, we obtain the control input Δv, that minimizes the cross track error between the waypoint and the current position. The gain value K_{lat} is tuned based on experiments, where the vehicle was tasked at following a lane.

4.7 InterProcess Communication

The previous section discussed the complete set of algorithms that have been employed in the software. However, data needs to be transferred between certain algorithms and this is known as interprocess communication. Robot Operating System(ROS) is a middleware, which is widely used in the robotic industry to achieve communication between different modules. However, ROS does not guarantee real-time performance. Due to the hard real-time constraint on the system, a data communication system must also be designed for the software to communicate. The communication required in the wheeled robot include:

1. A program on Jetson Nano must send perception output to the Remote Computer as request.

2. A program on the remote computer accepts the request message and must transmits a trajectory back to the Jetson Nano.

3. The Jetson Nano receives the response message and transmits the trajectory to the controller.

Problem 3 can be satisfied by a serial communication channel setup between the jetson Nano and the Arduino Uno. Problem 1 and 2 must be setup through wireless communication.

4.7.1 Transport Layer

Setting up data transmission through a network involves 7 layers, based on the OSI model. They are :

- Application Layer
- Presentation Layer
- Session Layer
- Transport Layer
- Network Layer
- Data Link Layer
- Physical Layer

The Network, Data link and Physical layers are related to the hardware elements of data transmission. The Application, Presentation and Session layer, deals with encryption and preparation of data to be sent into the network. The Transport layer of the OSI model

is responsible for the actual data transmission between the network nodes, and this must be defined by the software. There are two common standard protocols practiced in the transport layer, which are:

- User Datagram Protocol(UDP)
- Transmission Control Protocol(TCP)

There are pros and cons to both these protocols. UDP is connection-less protocol, where data is transmitted from the source node. Any node can intercept such messages, and can communicate back to the source node. This protocol enables fast data transfer. However, security issues exist with UCP. There is also an issue of loss or corruption of data, when received by the receiver. TCP is a connection based protocol, which performs a 3 way handshake before data packets can be transmitted between the two nodes. This system is secure, as data transmission only occurs when the handshake is complete. TCP also has no corruption of data. However, the three way handshake, introduces a latency, when data of large size is transmitted. However, for our application, data corruption is not tolerable, as the data is used by safety critical applications. Therefore, TCP protocol was the default choice for our application. It is also seen that the latency is negligible as the size of data packets sent is not too large.

4.7.2 TCP Client Server Setup

The IP address and port number of both the client and the server are already defined in the programs. A synchronous TCP client server setup is created using the Boost ASIO library, which abstracts the socket library. A synchronous setup implies the processing is paused until a read or write function is complete. Data must be prepared into a stream of bytes,

to be transmitted through TCP protocol. This process of conversion of custom data objects into a stream of bytes is known as serialization. Serialization of the request and response message is performed using Boost Serialization library. The client prepares the data buffer and initiates a connection to the server, whose location is identified in the network by the IP address and port number. The server actively listens for any connection requests from clients. The server accepts an eligible connection. Once the connection is accepted, data transmission can be performed between the two nodes. The server receives the request and performs the mapping, path and trajectory planning, and serializes the resulting trajectory. This is sent back to the client and the connection is closed. This process repeats for every time step.

4.8 Software Summary

The complete software architecture employed in the software of the prototype was discussed in this chapter. Distributed Computing using Client Server model is set up to share computation resources between the robot on-board computer and the remote computer, which holds the User Interface. The perception module of the system is executed in the on board computer, which is transmitted to the remote computer by TCP protocol. The remote computer maps the perception data into an occupancy grid. The Behavioral planner selects a maneuver based on the decision framework and plans path with two algorithms(RRT* and PRM-A*) running in parallel. Once a path is found, a trajectory is fit and sent back to the onboard computer. This trajectory is tracked by a controller, which performs PID based longitudinal control and a pure pursuit based lateral control, based on a kinematic skid steering model.

Chapter 5

Results and Discussion

In the previous chapter, a detailed description of the various modules of the software stack was provided. The software, interacts with the hardware, that was described in Chapter 3, to enable the wheeled robot to move across the environment. Each individual module is critical to the working of the complete system. The objective of the robot is to check for the feasibility of a integration of user input based navigation, in an uncertain environment, for an Autonomous Vehicle. Hence, simplified road scenario tests are performed on the robot, to check qualitatively if the vehicle adapts to the user input, or lack thereof, and locomote safely in the environment. The complete system architecture is shown in figure 5.1. In this chapter, we will discuss the results of implementation of these sub-systems, on the wheeled robot, as well as critique on the challenges observed. We will also discuss the potential additions to the software to be compatible with an AV.

5.1 Robot Assembly and Calibration

A wheeled robot prototype is designed and assembled with the hardware suite that was discussed in Chapter 3. The IMU is placed in the geometric center of the vehicle and depth camera is mounted facing towards the front, to capture RGB images. The actuators, are connected to a controller, and actuation or control signals are transmitted in the form of PWM signals. With the actuators, a computation board and a UI, a drive by wire setup is

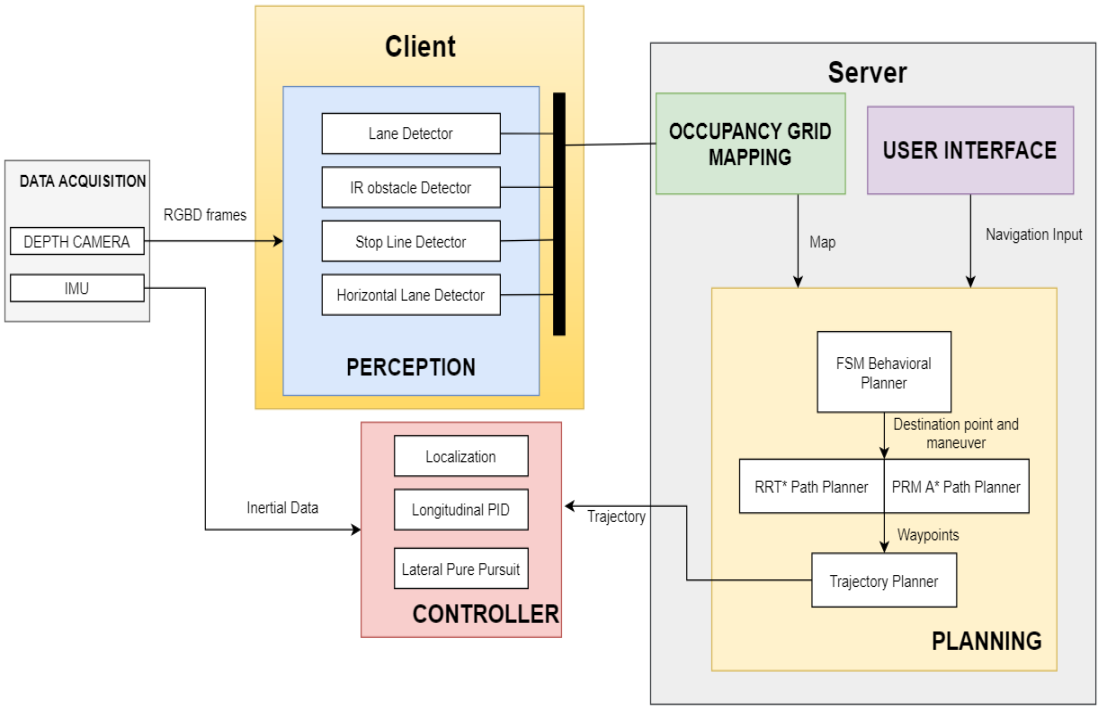

Figure 5.1: System Architecture for Wheeled Robot Prototype

made. However, the sensors enable the autonomy functions of the robot. These sensors of the robot, have to be calibrated, to accurately predict the environment and compute secure paths. The sensors in the system are calibrated to suit the current application, and testing conditions.

The IMU was observed to provide data with a high degree of noise. As the vehicle starts, a calibration program is run on the controller, which measures the data from IMU at zero control inputs and hence no motion. This data, over a period of 10 seconds is used to estimate the bias and variance of the sensor. Dead Reckoning is performed when the robot is in motion, correcting for the zero-input bias, measured at the startup. A threshold is also set, based on visualization of data, within which, the accelerations were set to 0. The threshold was set, as the velocity values, due to dead reckoning, were continuously accumulating, even under constant velocity conditions, due to noise in the sensor, which affected the working of the

longitudinal controller, due to buildup in the integration error.

The localization system of an Autonomous vehicle, contains multiple sensors, other than IMU, such as GPS and wheel encoders, as discussed in 2.1.7. State Estimation methods, such as a simple Extended Kalman Filter(EKF), not to be confused with SLAM EKF, can be used to fuse the data from these sensors. The calibration of such a system would be estimation of the process and measurement noise for the vehicle and the localization sensors used. Thus employing the proposed system can free up the computation, that is required in Feature based Localization in HD map based methods, and Baysian Filtering based SLAM algorithms, whose state vector is the size of the state of the vehicle, as well as the state of the landmarks in the current environment.

The second major sensor used in the robot prototype is the D435 depth camera. As mentioned in 3.3.1, the selection of the sensor was due to the in-board computation that filters the data to provide accurate aligned depth and RGB information. However, we must provide thresholds in the depth values provided by the depth camera. This is due to the fact that, the sensor does not provide reliable depth data, within 30 cm of the sensor and above 5 meters. There is also corruption of data, that occurs when reflective surfaces are present in the environment. To account for all of these, a post processing technique known as Spacial Edge-Preserving filter, as explained in [20] is employed, by using the Realsense SDK.

Vision sensors of Autonomous vehicles comprise of multiple cameras to generate RGB data of the surroundings and range sensors such as RADAR and LIDAR, to generate depth data. Calibration of the perception system for an AV would start with a process known as image stitching, which combines RGB frames from multiple cameras, accounting to the overlap in the FOV of cameras. This is followed by LIDAR-Camera Calibration and RADAR-Camera calibration, to associate depth values to appropriate pixels in the environment, in addition to the degree of uncertainty of the measure, which depends on the sensor. On completion of

these steps, an AV contains a 360° image of its surroundings, with information of associated depth.

5.2 Inter-Process Communication

So far, the calibration of the sensors used in the robot has been discussed. Sensor data, extracted from ADCs need to be transferred to different elements in the system software for processing. The wheeled robot prototype communication system is as follows:

- IMU data is sampled by the Arduino Uno's ADC.
- RGBD data is sampled by the on-board chip on the D435 depth camera and sent to On-board computer by serial communication.
- On-board computer executes the perception module of the software and transmits detected obstacles and lanes to the remote Computer using TCP wireless protocol.
- Remote Computer executes the planning module of the software and transmits the path to the on-board computer using TCP wireless protocol.
- On-board computer transmits the path to the Arduino Uno by serial Communication.

The TCP protocol based client server communication is established in real-time. The client identifies the coordinates of the obstacles and lanes and serializes the data and transmits it to the server, which is identified by an IP address and port number. The server program receives the serialized class which contains the coordinates of the lanes and obstacles, from the client. The server de-serializes the data to access the obstacle information, using which the planning module generates a safe trajectory which is serialized and sent back to the client. The client receives the serialized version of the trajectory, de-serializes and sends to

the arduino via serial port. An AV is large enough to hold large computation platforms and hence wireless communication is usually unnecessary. However, there are few cases when sensors are placed in moving parts and data is sent out wirelessly to receivers. In AVs, the sensor data is communicated system wide using CAN bus, reducing delays to a minimum between sensor data acquisition and processing.

5.3 Perception Software

The algorithms for Lane and Obstacle detection were discussed in Section 4.3. There is also a Hough transform based horizontal line detection, based on a rotated Birds eye view, similar to lane detection, to identify the location of the stop lines. When the vehicle is stopped, lane detection is also performed on the rotated BEV image to identify intersection lanes. The performance of these algorithms were analyzed, and a few parameters were tuned, such as the ROI for lane detection, sliding window resolution,etc. Every individual perception function is performed on the same RGB frame. However, running them sequentially in a pipeline creates a latency, due to which planning might fail as the environment is appreciably changed, as compared to the perception output, due to motion of the robot. Since, each program is independent of each other, multi-threading was decided to be tested. Multi-threading is the process of scheduling tasks on multiple cores simultaneously, to perform tasks with parallel processing. Multi-threading, however, contains an overhead in creation of threads and handling memory. Hence, an analysis of the performance of multiple scenarios of the perception task was performed, whose data is shown in Table 5.1, which clearly shows that the average time taken for the perception system to complete is lower in case of multiple threads. An object detection system was tested with the software, which detects vehicles and pedestrians. Despite having no pedestrians in the scene, there is a huge latency created

when systems are run sequentially. Hence, the final perception system performs different visual detections on parallel threads.

Table 5.1: Perception System Performance Analysis

Condition	**Mean Execution time without thread**	**Mean Multithreaded Execution time**
Lane and Obstacle Detection	24 ms	14 ms
Lane, Stop line and Obstacle Detection	30 ms	14 ms
Horizontal and Vertical Lane Detection with Stop lines	42 ms	16 ms
Horizontal and Vertical Lane Detection with Stop lines and Object Detection	134 ms	41 ms

Perception Stack of the AV is one of the bottlenecks of the software stack. Additional algorithms, from that of the prototype, such as object detection, tracking, feature detection for visual odometry, etc., also need to run in parallel. However, modern AVs contain Graphical Processing Units(GPU), which allows parallel processing, that speeds up deep learning based algorithms, that work with tensors and matrices. One drawback of the proposed system, is the scenario in which lane lines are not visible due to occlusion by vehicles, in high traffic urban scenarios. This can, however, be accounted for, by installing a platooning state, in the behavioral planner, that follows the preceding vehicle, until intersections.

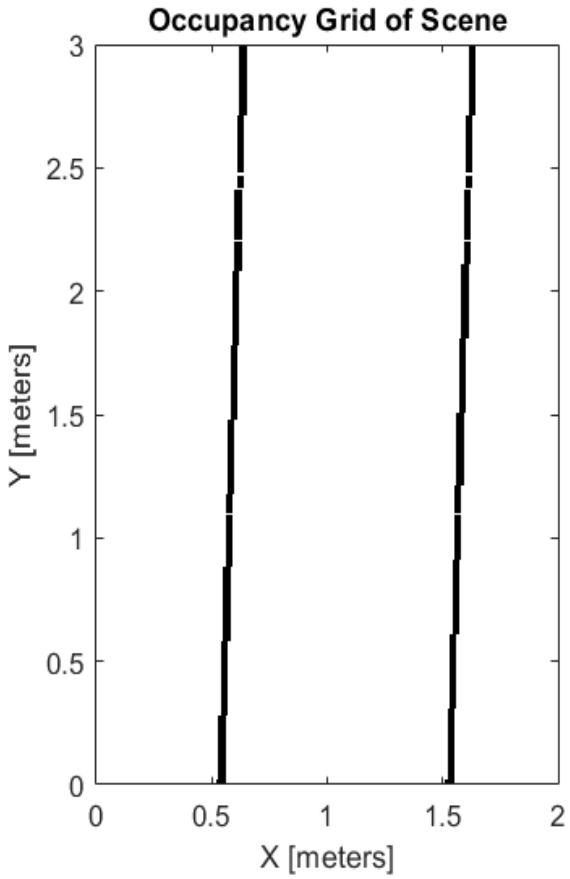

Figure 5.2: A visualization of an Occupancy Grid generated at a frame. All obstacles in the current frame are due to lanes.

5.4 Mapping

Real time on-line maps are created in the form of a 2D binary occupancy grid, based on the outputs from the perception system on the remote computer. The obstacles are added to the grid by employing multi-threading, to parallelize the task. The system is agnostic of the map of the previous time step and only depends on the current perception inputs. This saves memory in the form of an expanding map buffer, as when SLAM is applied. A visualization of the binary occupancy grid formed due to detection of obstacles and lanes is shown in Figure 5.2. The performance of the Mapping module depends on the number of obstacles in

the environment. This system does not identify any landmarks in the environment as done by SLAM based algorithms in contemporary AVs. The system proposed localizes the ego vehicle, based on the road infrastructure, such as lane markings and static obstacles, and plans paths based on the local goal set by the user. This has similarities to the mapping stage of a SLAM based AV, which maps the environment, extracts and associates landmarks, and builds incremental maps, while planning towards the goals set by a route planner. If a point on the map is revisited, the vehicle localizes by loop closure techniques, by identifying the landmarks and associating them with that in memory. The advantage of the proposed system over current system in the mapping perspective is the memory-less property. Memory is required in SLAM based systems as route is planned on an absolute coordinate system and the vehicle has to maintain a global coordinate system to traverse the route. Since the user provides local navigation goals, the vehicle is only tasked to controlling the vehicle towards the local goal and hence the coordinate system can be reset at every motion planning update. The large amount of computation performed in landmark extraction, data association and localization, can be avoided.

5.5 Behavioral Planning

The behavioral planner on the robot is based on a Finite State Machine, that works based on the proposed state and transition law, that accepts the user input from a UI as a factor in deciding action. The behavioral planner decided performs default maneuvers when user provides no input, allowing the system to be robust during absence of user inputs. The default rules are formed in a way to reduce the number of inputs a human has to provide. An example of this is that the user can abstain from informing go straight at every intersection, or stop sign. If the straight maneuver is added in default, the user only has to provide

navigation inputs at stop sign, if a turn is necessary. The wheeled robot works on a limited set of maneuvers and is tested for limited scenarios. The behavioral planner can be adapted to an AV by meticulously accounting for all local road scenarios such as handling of a roundabout, taking the exit at a highway, handling a five way intersection and others.

5.6 Path and Trajectory Planning

We have implemented two sampling based planners, RRT* and a combination of PRM and A*search, running in parallel threads, as the path planner for the current system. In release mode, we get an average time of execution as 20 ms for the PRM A* combination, and 12 ms for the RRT* algorithm. The reason why average time is considered is due to the fact that the performance depends on the environment. An example of the path generated by RRT* and PRM A*, is shown in Figure 5.3. The left image is the RRT* path, which has tighter constraints on the angle, for which two nodes can be connected as edges. The PRM-A* combination, contrary to the RRT*, is able to connect edges with sharp corners to the graph. This is ensure we do not lose a path in an environment which has high density of obstacles, because of the angular constraints that are installed to decrease the curvature of the calculated path for the robot to follow. Both algorithms run in parallel, and the execution stops when either one has found a path. It is seen from observations that RRT* always finds the path before the PRM-A* planner. This is due to the fact that path finding process is initiated only after the PRM algorithm has sampled all the N points. However, testing has been done only in sparse obstacle environment. It can be postulated that, in an environment with dense obstacles, the PRM-A* planner will perform a better job in finding a path, if available, as it connects any two edges within the radius. The paths are then converted to trajectories, by curve fitting and time-scaling.

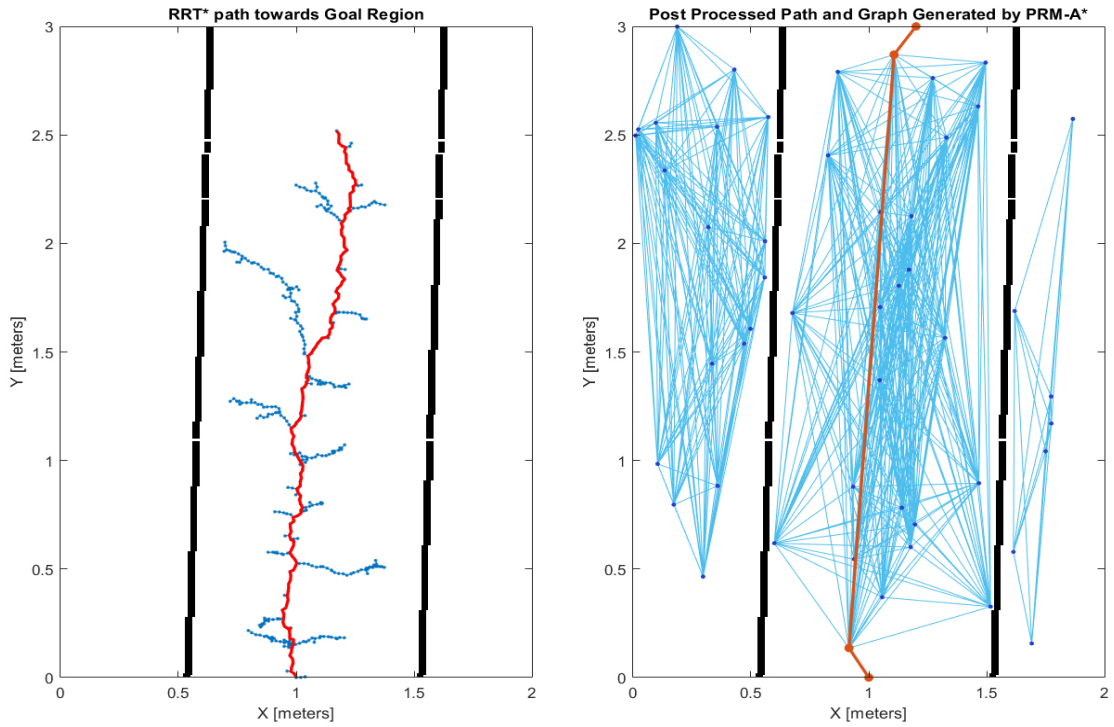

Figure 5.3: RRT* Path(left) and PRM-A* path(right) for the same occupancy grid.

A path planner with a similar configuration can be used in an Autonomous Vehicle as is. There are motion model aware path planning algorithms, such as the Hybrid A* path planner, and even Model Predictive Control(MPC), which can be applied to provide higher fidelity paths. The Trajectory planner of the Autonomous Vehicle, might however require, further addition of constraints. Constraints on the curvature of the path are essential, without which, we may obtain paths that are not dynamically feasible. Creating a soft constraint for the boundary, instead of a hard boundary constraint, also, based on the literature, provides better quality paths. The major addition required on the trajectory planner for AVs, that is not employed in the prototype is the handling of the dynamic obstacles. A data structure must be maintained that keeps record of the dynamic obstacles in the environment, with their motion models. Hence, the collision check can be a geometry check of overlap between the position of ego vehicle in time and the tracked vehicle in time. Based on this, the time

scaling of the ego vehicle trajectory can be adjusted to account for the trajectory of the other vehicles or pedestrians or any other form of dynamic obstacle.

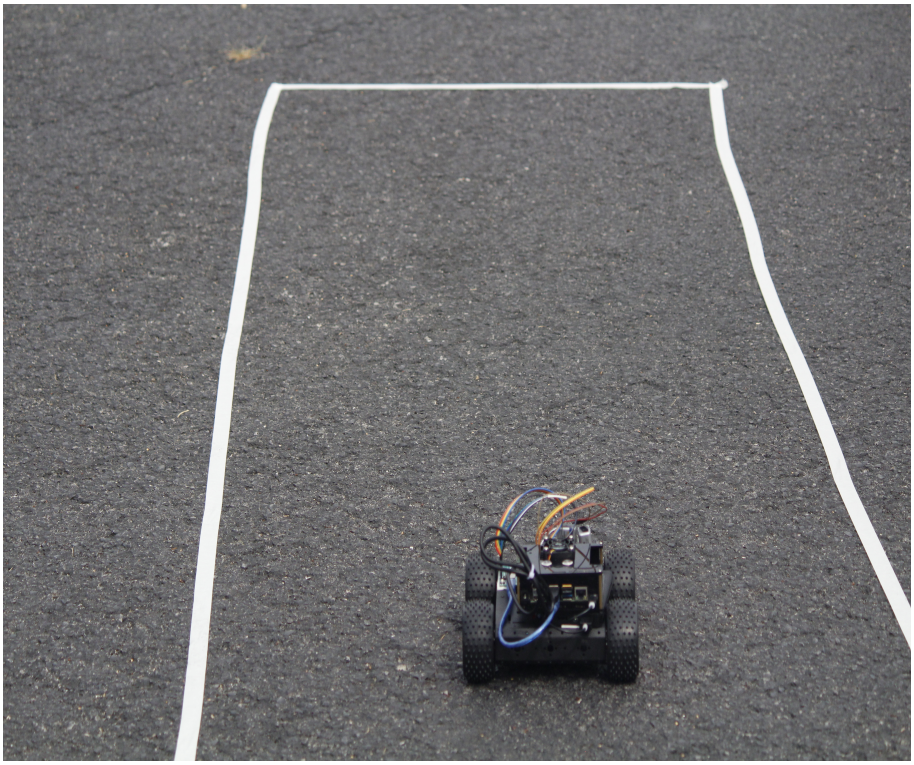

Figure 5.4: Robot tested on road surfaces for various conditions

5.7 Trajectory Tracking Controller

Disjoint longitudinal and lateral controllers, which are discussed in 4.6, are implemented in the Arduino Uno microcontroller. The controller is set to operate at 100Hz, the sampling rate configured for the IMU. The selection of 100Hz is due to the drawbacks of the system clock of the Arduino Uno, whose system clock saturates, when time is measured in microseconds. Higher sampling rate improves the performance of the localization of the robot, and hence would avoid the gitchy motion, that can be seen at times by the robot, due to the localization errors caused by the noise in the IMU. State estimation algorithms using sensor fusion, can

also improve the performance of the controller. The controller resets the global coordinate frame to the vehicle coordinate frame at every motion planning update, and follows a new trajectory towards the local motion goal.

The controller works based on a proportional control law for lateral control, and PID for longitudinal. Hence, the gains for the controller need to be tuned, based on the surface being tested, owing to the slip phenomenon, which causes the steering action of the prototype. An AV could use MPC and other optimal control strategies, to generate controllers agnostic of the surface. The AVs require further standard low level controllers, to control the ride and handling of the vehicle, whilst following the trajectory planned by the system. Another important addition required is the handling of any jerks that might be caused by reset of the coordinate frame and trajectory. This can be handled, by adding a jerk parameter in the cost function, to ensure gradual change of control inputs, immediately following the reset.

5.8 Tests and Performance

The robot is tested on surfaces, with white tapes indicating lanes, as shown in Figure 5.4. The mean software update time for every 100 updates vs number of updates, for a total of 20000 motion planning updates, is shown in figure 5.5. It is seen that the mean update rate of the system is 24Hz(41 ms update time), which is much higher than slam based systems, despite the overhead in the client server communication. Based on the tests, the robot follows the motion command that is entered by the user, in a terminal. In the absence of a command, the robot follows the action specified in the default transition law. The robot is able to follow the lanes in the outdoor conditions, stop at stop lines and turn at the intersection.

The robot was tested for all the 8 transitions, discussed in Figure 4.8. It executes the

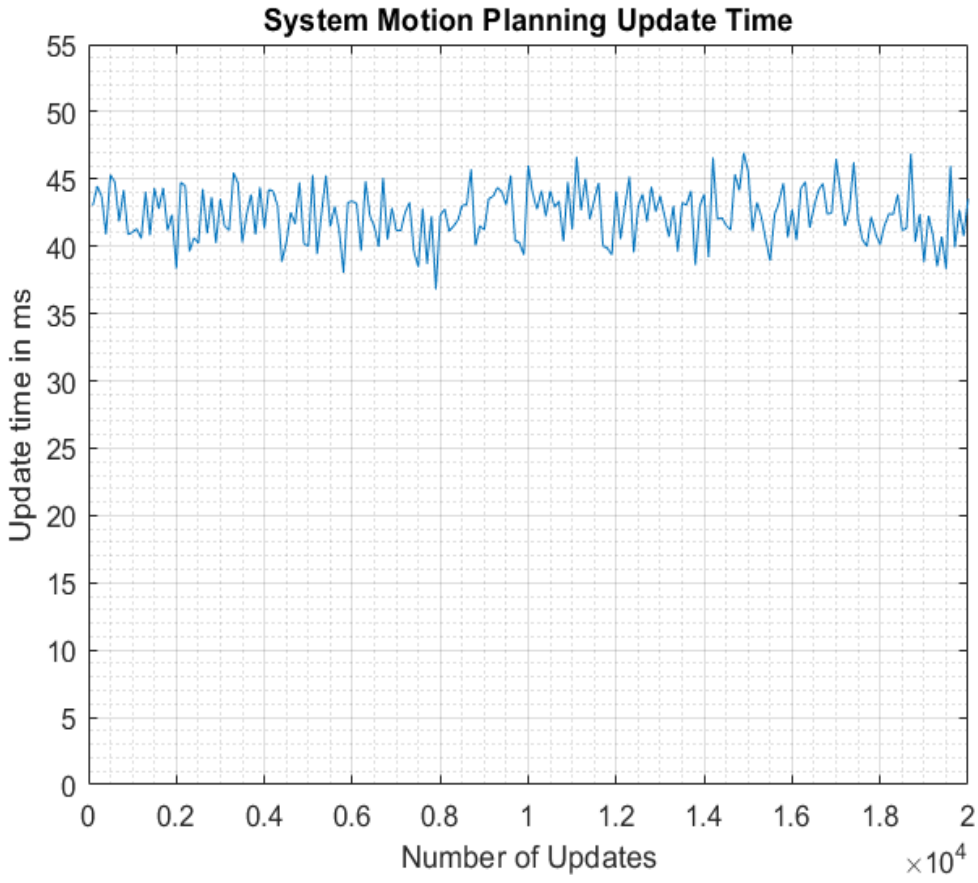

Figure 5.5: Mean Update time for 20000 Motion Planning updates

maneuvers in a continuous manner. Due to the Coordinate reset procedure, the absolute position of the vehicle, is not estimated. However, the occupancy grid and paths for the vehicle formed are shown in figures in the current chapter. There are certain issues observed in the serial communication interface of Arduino and Jetson Nano. Based on diagnosis, it is seen that a few path messages are corrupted due to the high speed of data transmission. The baudrate was increased, and the transmission errors reduced. These errors, however, are specific to the hardware setup and can be removed with a CAN bus network. From the above results, it can be seen that motion planning with user navigation inputs, by means of coordinate frame resets is feasible.

Chapter 6

Conclusions and Future Work

We have discussed in detail about a human-navigation based semi-autonomous vehicle software. In this chapter, a summary of the contribution of this work is presented. Also suggestions for future work are provided to add to the system to potentially commercialize the solution presented in the thesis.

6.1 Summary

We have proposed an alternate strategy towards accounting for localization errors faced by an Autonomous Vehicle in the environment. Contemporary AVs localize the ego vehicle based on visual features and landmarks stored in the memory. These processes are computationally expensive and the technology is still under active research for improvement. The system proposed in the research, delegates navigation to the user or passenger of the vehicle, whilst maintaining active control of the driving of the vehicle. The passenger provides local navigation goals, which are utilized by the behavioral planner to estimate a safe maneuver towards a valid local goal. A brief review of the system proposed is shown in the following:

- Perception system identifies the static obstacles and road infrastructures in the environment and localizes them with respect to the vehicle in the vehicle coordinate frame.

- An online map is created based on the perception output, in the vehicle coordinate frame.
- The system asynchronously accepts Navigation inputs from the user through a UI, which is sent to the Behavioral Planner.
- The Behavioral Planner decides a safe maneuver based on a decision framework depending on the environment, road rules and user navigation input.
- A path planner generates a safe path the ego vehicle needs to trace to execute estimated maneuver.
- The controller follows the new trajectory generated from the trajectory planner, until the next motion planning update.
- At every motion planning update, the global coordinate frame is reset to the vehicle coordinate frame, thus eliminating position errors accumulated from a generic localization system consisting of IMU,GPS and other sensors.
- In the interval between motion planning updates, the vehicle operates in the updated global coordinate frame and localizes itself using the sensor suite mentioned above, using which control inputs are provided to follow the path.

6.2 Inference

The main advantage of this system is that localization is performed based on the sensors for only up to a maximum of 1/10th of a second, after which error accumulation is reset due to the re-planning of the path, based on changes in the environment. Since no global coordinate system is maintained, the heavy computation performed in localizing vehicles,

and landmarks in the environment, as done in contemporary Simultaneous Localization and Mapping methods, or localizing the vehicle in a map, based on the visual features and landmarks that are stored in a map, can be removed. Such a system can be used in areas where the localization errors have accumulated to an unsafe degree, or as a standalone ADAS system, which relieves the passenger of the vehicle from driving the vehicle and is just tasked at providing the route for a vehicle to follow.

A prototype of the system is constructed in the form of a wheeled robot, and is allowed to operate in a controlled environment with only a single set of lanes and no dynamic obstacles.The prototype solves a simplified complete Autonomous Vehicle problem with the coordinate reset procedure and user inputs, including the interfacing of every sub-system of the AV, to indicate the feasibility of the proposed system in road environments. It is seen that the robot performs safe maneuvers to locomote from one point to another, by identifying the road lanes, and planning a path, based on the presence, or absence of a navigation input from the user. The same system can be adapted into an AV, with certain modifications, to navigate in uncertain environments, while circumventing the problem of localization in contemporary vehicles.

6.3 Future Work

The current prototype is a simplified setup working in controlled environments. Future work to polish the current system could be suggested in the following ways:

1. Testing of the Human-Based Navigation system on a dynamic environment, with more complicated maneuvers.

2. Real-time surface classification systems exist, which identify the surface of the road,

using the tire as a sensor. Hence, the constraints on the motion planning module could be modified based on the surface classified, to provide a robust Planning and control system, that accounts for the changing road conditions.

3. Platooning systems could be added to the software for the system to operate in a dense urban setting, wherein the lanes are occluded.

4. V2V communication systems can be integrated with the proposed system to obtain the planned trajectory of vehicles on the road, which can be utilized in the trajectory planner.

Bibliography

[1] Marija Đakulović, Mijo Čikeš, and Ivan Petrović. Efficient interpolated path planning of mobile robots based on occupancy grid maps. *IFAC Proceedings Volumes*, 45(22): 349–354, 2012.

[2] Steven Ashley. Centimeter-accurate gps for self-driving vehicles. sae.org/news/2016/10/centimeter-accurate-gps-for-self-driving-vehicles, November 2016.

[3] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Paixao, Filipe Mutz, et al. Self-driving cars: A survey. *arXiv preprint arXiv:1901.04407*, 2019.

[4] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner. Three decades of driver assistance systems: Review and future perspectives. *IEEE Intelligent Transportation Systems Magazine*, 6(4):6–22, 2014.

[5] Francesco Borrelli, Paolo Falcone, Tamas Keviczky, Jahan Asgari, and Davor Hrovat. Mpc-based approach to active steering for autonomous vehicle systems. *International journal of vehicle autonomous systems*, 3(2-4):265–291, 2005.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] KJ Bussemaker. Sensing requirements for an automated vehicle for highway and rural environments. 2014.

[8] Sean Campbell, Niall O'Mahony, Lenka Krpalcova, Daniel Riordan, Joseph Walsh, Aidan Murphy, and Conor Ryan. Sensor technology in autonomous vehicles: A re-

view. In *2018 29th Irish Signals and Systems Conference (ISSC)*, pages 1–4. IEEE, 2018.

[9] Christopher R Carlson, J Christian Gerdes, and J David Powell. Error sources when land vehicle dead reckoning with differential wheelspeeds. *Navigation*, 51(1):13–27, 2004.

[10] Zhe Chen and Zijing Chen. Rbnet: A deep neural network for unified road and road boundary detection. In *International Conference on Neural Information Processing*, pages 677–687. Springer, 2017.

[11] SAE On-Road Automated Vehicle Standards Committee et al. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. *SAE Standard J*, 3016:1–16, 2014.

[12] Alexis Davies. Google's self-driving car caused its first crash - wired. www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/, February 2016.

[13] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[14] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 17(3):229–241, 2001.

[15] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 1001(48105): 18–80, 2008.

[16] Raúl Dominguez, Enrique Onieva, Javier Alonso, Jorge Villagra, and Carlos Gonzalez. Lidar based perception solution for autonomous vehicles. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 790–795. IEEE, 2011.

[17] Yarrow Eady. Tesla's computer vision master plan - medium article. medium.com/protopiablog/teslas-computer-vision-master-plan-512b36d8acbf, April 2018.

[18] Christopher L Van Dan Elzen, Ka Chai Cheok, and Micho Radovnikovich. Lane keeping system and lane centering system, November 10 2015. US Patent 9,180,908.

[19] John Folkesson and Henrik Christensen. *SIFT Based Graphical SLAM on a Packbot*, pages 317–328. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-75404-6. doi: 10.1007/978-3-540-75404-6_30. URL https://doi.org/10.1007/978-3-540-75404-6_30.

[20] Eduardo SL Gastal and Manuel M Oliveira. Domain transform for edge-aware image and video processing. In *ACM SIGGRAPH 2011 papers*, pages 1–12. 2011.

[21] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[22] Arturo Gil, Miguel Juliá, and Óscar Reinoso. Occupancy grid based graph-slam using the distance transform, surf features and sgd. *Engineering Applications of Artificial Intelligence*, 40:1–10, 2015.

[23] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[24] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[25] Christian Götte, Martin Keller, Till Nattermann, Carsten Haß, Karl-Heinz Glander,

and Torsten Bertram. Spline-based motion planning for automated driving. *IFAC-PapersOnLine*, 50(1):9114–9119, 2017.

[26] Tianyu Gu and John M Dolan. On-road motion planning for autonomous vehicles. In *International Conference on Intelligent Robotics and Applications*, pages 588–597. Springer, 2012.

[27] Tianyu Gu, Jarrod Snider, John M Dolan, and Jin-woo Lee. Focused trajectory planning for autonomous on-road driving. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 547–552. IEEE, 2013.

[28] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[29] Andrew J Hawkins. Serious safety lapses led to uber's fatal self-driving crash, new documents suggest - the verge. www.theverge.com/2019/11/6/20951385/uber-self-driving-crash-death-reason-ntsb-dcouments, November 2019.

[30] Chenhang He, Hui Zeng, Jianqiang Huang, Xian-Sheng Hua, and Lei Zhang. Structure aware single-stage 3d object detection from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11873–11882, 2020.

[31] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2719–2726. IEEE, 1997.

[32] Veli Ilci and Charles Toth. High definition 3d map creation using gnss/imu/lidar sensor integration to support autonomous vehicle navigation. *Sensors*, 20(3):899, 2020.

[33] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Arxiv*, pages arXiv–1704, 2017.

[34] Daesung Jeon, Hoimyung Choi, and Jaehwan Kim. Ukf data fusion of odometry and magnetic sensor for a precise indoor localization system of an autonomous vehicle. In *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 47–52. IEEE, 2016.

[35] Satish Jeyachandran. Introducing the 5th-generation waymo driver: Informed by experience, designed for scale, engineered to tackle more environments. blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html, March 2020.

[36] Junmin Wang and R. Rajamani. Adaptive cruise control system design and its impact on highway traffic flow. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, volume 5, pages 3690–3695 vol.5, 2002.

[37] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[38] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60: 416–442, 2015.

[39] Lydia E Kavraki, Mihail N Kolountzakis, and J-C Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1): 166–171, 1998.

[40] CY Kuo, YR Lu, and SM Yang. On the image sensor processing for lane detection and control in vehicle lane keeping systems. *Sensors*, 19(7):1665, 2019.

[41] Mathieu Labbe and Francois Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3): 734–745, 2013.

[42] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[43] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[44] Ankith Manjunath, Ying Liu, Bernardo Henriques, and Armin Engstle. Radar based object detection and tracking for autonomous driving. In *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, pages 1–4. IEEE, 2018.

[45] Raffi Mardirosian. Lidar vs. camera: Driving in the rain - ouster. ouster.com/blog/lidar-vs-camera-comparison-in-the-rain, February 2020.

[46] Jeff McMahon. Big fuel savings from autonomous vehicles - forbes. www.forbes.com/sites/jeffmcmahon/2017/04/17/big-fuel-savings-from-autonomous-vehicles/#5086df1e4390, April 2017.

[47] Jesús Munoz-Bulnes, Carlos Fernandez, Ignacio Parra, David Fernández-Llorca, and Miguel A Sotelo. Deep fully convolutional networks with random data augmentation for enhanced generalization in road detection. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 366–371. IEEE, 2017.

[48] Paul Munro and Antony P Gerdelan. Stereo vision computer depth perception. *Country United States City University Park Country Code US Post code*, 16802, 2009.

[49] Gene Munster. Seeing the road ahead: The importance of cameras to self-driving vehicles - loup ventures. loupventures.com/seeing-the-road-ahead-the-importance-of-cameras-to-self-driving-vehicles-2/, April 2018.

[50] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[51] USA National Highway Transport Safety Administration. Automated vehicles for safety. https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

[52] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. The kitti vision benchmark suite. www.cvlibs.net/datasets/kitti/.

[53] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[54] Lucas C Possatti, Rânik Guidolini, Vinicius B Cardoso, Rodrigo F Berriel, Thiago M Paixão, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Traffic light recognition using deep learning and prior maps for autonomous cars. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[55] Akshay Rangesh and Mohan Manubhai Trivedi. No blind spots: Full-surround multi-object tracking for autonomous vehicles using cameras and lidars. *IEEE Transactions on Intelligent Vehicles*, 4(4):588–599, 2019.

[56] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[57] Giulio Reina, James Underwood, Graham Brooker, and Hugh Durrant-Whyte. Radar-based perception for autonomous outdoor vehicles. *Journal of Field Robotics*, 28(6): 894–913, 2011.

[58] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[59] Søren Riisgaard and Morten Rufus Blas. Slam for dummies. *A Tutorial Approach to Simultaneous Localization and Mapping*, 22(1-127):126, 2003.

[60] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020.

[61] Santokh Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Technical report, 2015.

[62] Jarrod M Snider et al. Automatic steering methods for autonomous automobile path tracking. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[63] Tesla. Tesla autopilot - official website. www.tesla.com/autopilot.

[64] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al.

Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23 (9):661–692, 2006.

[65] Annemarie Turnwald and Dirk Wollherr. Human-like motion planning based on game theoretic decision making. *International Journal of Social Robotics*, 11(1):151–170, 2019.

[66] Chris Urmson, Chris Baker, John Dolan, Paul Rybski, Bryan Salesky, William Whittaker, Dave Ferguson, and Michael Darms. Autonomous driving in traffic: Boss and the urban challenge. *AI magazine*, 30(2):17–17, 2009.

[67] Daobin Wang, Huawei Liang, Tao Mei, Hui Zhu, Jing Fu, and Xiang Tao. Lidar scan matching ekf-slam using the differential model of vehicle motion. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 908–912. IEEE, 2013.

[68] Tianmiao Wang, Yao Wu, Jianhong Liang, Chenhao Han, Jiao Chen, and Qiteng Zhao. Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor. *Sensors*, 15(5):9681–9702, 2015.

[69] Junqing Wei, John M Dolan, Jarrod M Snider, and Bakhtiar Litkouhi. A point-based mdp for robust single-lane autonomous driving behavior under uncertainties. In *2011 IEEE International Conference on Robotics and Automation*, pages 2586–2592. IEEE, 2011.

[70] Ryan W Wolcott and Ryan M Eustice. Visual localization within lidar maps for automated urban driving. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 176–183. IEEE, 2014.

[71] Ami Woo, Baris Fidan, and William W Melek. Localization for autonomous driving. *Handbook of Position Location: Theory, Practice, and Advances, Second Edition*, pages 1051–1087, 2018.

[72] Kyle Hollins Wray, Stefan J Witwicki, and Shlomo Zilberstein. Online decision-making for scalable autonomous systems. In *International Joint Conference on Artificial Intelligence*, 2017.

[73] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[74] Qin Zou, Hanwen Jiang, Qiyu Dai, Yuanhao Yue, Long Chen, and Qian Wang. Robust lane detection from continuous driving scenes using deep neural networks. *IEEE transactions on vehicular technology*, 69(1):41–54, 2019.