Control Architecture for Autonomous/Assisted Driving of Electric Vehicle

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT
FOR THE DEGREE OF

Master of Technology

IN THE
FACULTY OF ENGINEERING

BY

SACHIN THOMAS

GUIDED BY

PROF. L UMANAND
DR. T V PRABHAKAR

DEPARTMENT OF ELECTRONIC SYSTEMS ENGINEERING

INDIAN INSTITUTE OF SCIENCE, BANGALORE

JUNE 2024

Synopsis

Autonomous driving technology is rapidly gaining traction in the market due to its inherent benefits, including improved safety, enhanced driver comfort, and more efficient transportation. By merging the advantages of autonomous driving with electric vehicles (EVs), the performance and benefits of future vehicles are poised to escalate further. Given its rising popularity and advantages, intensive research is underway across various subfields of autonomous driving, encompassing sensor fusion, algorithm development, and architectural advancements. However, much of this research primarily targets enhancing system performance, often overlooking considerations of computational complexity and hardware resource requirements. This emphasis on performance optimization presents challenges for implementation in academic environments.

This project centers on the development of an autonomous driving vehicle tailored specifically for academic environments. Emphasizing modularity, minimal resource utilization, and scalability, the project aims to address the challenges of implementing complex autonomous systems within academic constraints. A detailed examination is undertaken to explore various approaches to achieve modularity while minimizing complexity and reducing sensor integration to attain Society of Automotive Engineers (SAE) level 3 autonomy. This discussion encompasses critical elements such as perception of the surrounding environment, localization, reference generation, and control strategies, each meticulously designed to optimize system functionality while conserving resources.

The project incorporates a custom-designed electronics module to replicate the functions of an autonomous driving vehicle with key features like obstacle handling and lane following, essential for effective navigation. The system was implemented on a tabby test vehicle and subjected to various scenarios to verify its performance. Notably, the system achieves speeds of up to 30 km/h while maintaining a throughput of 8 frames per second, affirming its suitability for academic environments and practical implementation.

Acronyms

ADAS Advanced Driver Assistance System

BMS Battery Management System

CNN Convolutional Neural Network

DDPG Deep Deterministic Policy Gradient

EV Electric Vehicle

FOV Field of View

GPS Global Positioning System

GPU Graphics Processing Unit

IMM Intelligent Mobility Module

LIDAR Light Detection and Ranging

MaaS Mobility as a Service

MDE Monocular Depth Estimation

ML Machine Learning

PID Proportional Integral Derivative

PWM Pulse Width Modulated

RL Reinforcement Learning

SAE Society of Automotive Engineering

SCM Speed Control Module

SPI Serial-Peripheral Interface

V2X Vehicle to Everything

YOLO You Only Look Once

Acknowledgements

Firstly, I would like to express my deepest gratitude and respect towards my research supervisors Prof. L. Umanand and Dr. T.V. Prabhakar. The discussions I had with them have been very fruitful in every aspect of developing this project. Their encouragement, guidance, and support from the initial to the final phase of the project helped me to successfully meet the goals.

Next, I would like to thank the members of the evaluation panel which included Prof. Kuruvila Varghese and Prof. Haresh Dagale. The suggestions and feedback they provided during the reviews were invaluable and helped me to improve the scope of the project. I am also grateful to all the professors at DESE for offering outstanding courses and invaluable instructions throughout my MTech program at IISc, which has greatly enriched my knowledge and helped me to improve my technical confidence.

My interaction with members of the Power Electronics lab in DESE has significantly contributed to the enhancement of the project. This includes technical help and suggestions from Dr. Ruman Kalyan Mahapatra, Joshua Mathew, and Arun K R Kumar, assistance and support from Naresh Meena, Vaidika, Syam Sunder Nair, and Vibhore Jain. I also thank my labmate Sagarika for bringing a cheerful energy to the lab. I will always cherish the time I have spent in the lab with my colleagues.

In addition to that, I want to thank members of the Mechanical Workshop, who had done significant work in assembling the tabby car, interfacing sensors, and designing an enclosure for my electronics module. I am also grateful to my friend and classmate Derick George for helping me capture different videos across the campus for the project. This was of considerable importance for testing and validating the design. I want to extend my gratitude to Aakanksha Vivek Gadakar, whose work on reference generation in autonomous vehicles was immensely valuable and served as a crucial starting point for my project. I also thank the DESE office staff for providing administrative support whenever needed.

Further, I am grateful for the numerous offline and online lectures I attended, which directly impacted my project's development. This includes Dr. Andrew Ng's machine learning courses in Coursera, Self Driving Cars Specialization course offered in Coursera by Dr. Steven Waslander and Dr. Jonathan Kelly of the University of Toronto, Nvidia training course offered by the Nvidia team and conducted by CDS, IISc.

Finally, I have also enjoyed a lot, spending time with my friends at IISc. I am very much thankful and grateful to all my friends in DESE and other departments with whom I have cherished my time in IISc.

Contents

List of Figures

Chapter 1

Introduction

Since the inception of fuel-driven cars, manual driving has been the norm, offering individuals a sense of control and independence on the roads. However, as our global commute times increase and traffic conditions worsen, issues such as driver fatigue, distraction, and lapses in judgment have become increasingly prevalent, posing risks to both drivers and pedestrians. The traditional reliance on human drivers, while providing employment opportunities and a sense of autonomy, also introduces significant safety concerns, particularly on long-distance journeys where driver fatigue can lead to catastrophic consequences. According to a study conducted by the National Crimes Records Bureau (NCRB) as shown in Fig. 1.1, approximately 4 lakh road accidents take place in the country each year, resulting in roughly 1.5 lakh casualties [1]. Moreover, as traffic congestion intensifies [2] as shown in Fig. 1.2, the margin for error decreases, highlighting the urgent need for innovative solutions to enhance road safety and efficiency.

Sl. No.	Year	Road Accidents (in thousand)	% Variation over Previous Year	Persons Injured (in thousand)	% Variation over Previous Year	Persons died (in Nos.)	% Variation Over Previous Year	No. of Vehicles (In Thousand)#	% Variation over previous Year	Rate of Deaths per thousand Vehicles (Col.7/Col.9)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
1	2017	445.7	-5.80%	456.2	-6.00%	1,50,093	-1.10%	2,53,311	10.1%	0.59
2	2018	445.5	0.00%	446.5	-2.10%	1,52,780	1.80%	2,72,587	7.6%	0.56
3	2019	437.4	-1.8	439.2	-1.6	1,54,732	1.3	2,95,772	8.5%	0.52
4	2020	354.8	-18.9	335	-23.7	1,33,201	-13.9	2,95,772*	-	0.45
5	2021	403.1	13.6	371.9	11.0	1,55,622	16.8	2,95,772*	-	0.53

Figure 1.1: Yearwise accidents and causalities in the Indian roads

The advent of autonomous driving electric vehicles (EVs) represents a transformative leap for-

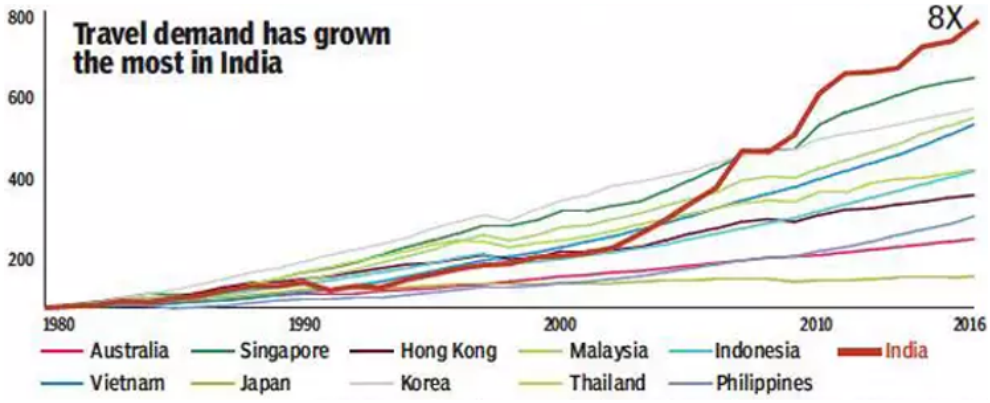


Figure 1.2: Travel demand across countries

ward in transportation technology, offering the promise of safer, more efficient, and environmentally sustainable mobility solutions. By leveraging cutting-edge artificial intelligence (AI) and sensor technologies, autonomous EVs have the potential to significantly reduce the incidence of accidents caused by human error, including fatigue, distraction, and lapses in judgment. Furthermore, with advancements in battery technology and the proliferation of renewable energy sources, autonomous EVs not only offer the prospect of cleaner air and reduced carbon emissions but also present a viable pathway towards achieving broader sustainability goals in the transportation sector. A study published [3] by researchers from the University of Michigan Transportation Research Institute (UMTRI), the Virginia Tech Transportation Institute (VTTI), and General Motors (GM) has revealed that human ride-hail drivers are more prone to accidents than self-driving cars. As shown in Fig. 1.3, it is found that there are nearly 65% reductions in the collisions in the automatic cruise mode than when operated in manual mode.

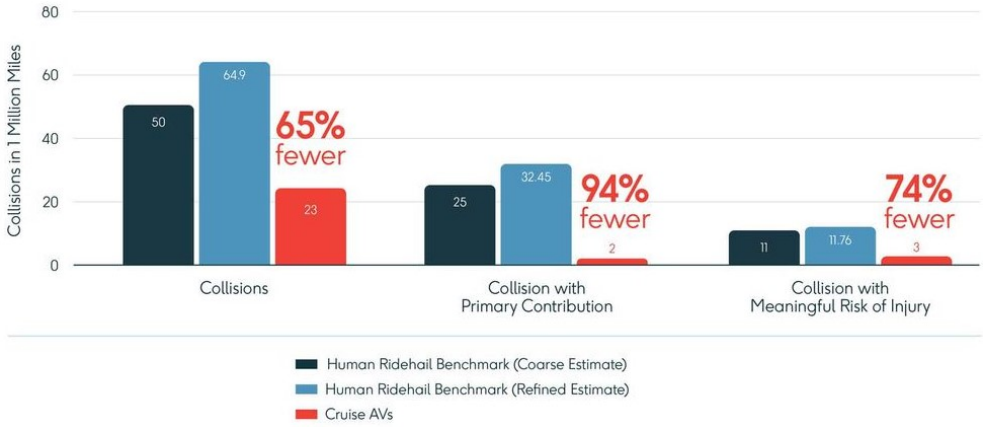


Figure 1.3: Accidents comparison between a human driver and autonomous driving car

Based on these studies, it can be inferred that autonomous driving vehicles are poised to become

the standard in the future, offering a substantial reduction in road accidents and alleviating driver fatigue. Moreover, combining the autonomous driving feature with an electric vehicle opens a plethora of advantages in terms of safety, comfort, reduced environmental impact, and efficiency.

1.1 History of Autonomous Driving Cars

The history of autonomous driving cars can be traced back to a century ago when different companies had demonstrated radio-controlled cars. In 1925, Houdina Radio Control demonstrated the radio-controlled American Wonder on New York City streets which was followed by Norman Bel Geddes's Futurama exhibit sponsored by General Motors at the 1939 World's Fair [4]. Later, after World War II, in 1953, RCA Labs successfully created a system with a miniature car guided and controlled by wires laid in a pattern on a laboratory floor. Further, they had extended the demonstration to on-road in 1957 [5]. The 1960s and 1970s decades also saw autonomous driving research based on the signal-obtained sensors mounted beneath the track. These activities were carried out by Ohio State University's Communication and Control Systems Laboratory, Citroen DS, Bendix Corporation, etc. However, these cars were nowhere near the autonomous driving car standards demanded in the present era.

The first self-driving car that did not rely upon rails or wires under the road was designed by the Japanese Tsukuba Mechanical Engineering Laboratory in 1977. The car was equipped with two cameras that used analog computer technology for signal processing. In the 1980s, a vision-guided Mercedes-Benz robotic van achieved a speed of 95.9 km/h on streets without traffic. In the same decade, the DARPA-funded Autonomous Land driven Vehicle (ALV) project achieved the first road-following demonstration that used Lidar, computer vision, and autonomous robotic control to direct a robotic vehicle at speeds of up to 31 km/h. Later by 1989, Carnegie Mellon University had pioneered the use of neural networks to steer and otherwise control autonomous vehicles forming the basis of contemporary control strategies [6]. The 1990s decade witnessed several academic and industry projects on various aspects of autonomous driving cars like platooning, lane changing, adaptive cruise control, etc. Some of the prominent projects include Demo '97, Asia Motors, twin robot vehicles VaMP and Vita-2 of Daimler-Benz and Ernst Dickmanns, Lucas Industries, Carnegie Mellon University's Navlab, ARGO Project, ParkShuttle, etc [7]. Though many of these projects had demonstrated several autonomous driving features, none of them had become commercially successful as the system was quite complex for the period.

The 2000s decade saw different challenges conducted by private and public agencies. Among them, the most successful was that of three challenges conducted by DARPA (the Defense Advanced Research Projects Agency). In the first Grand Challenge held in March 2004, DARPA offered a $1 million prize to any team that could create an autonomous car capable of finishing a 150-mile course in the Mojave Desert. However, no team was successful in completing the course [8]. In the second DARPA Grand Challenge of 2005, GPS points were placed and obstacle types were located in advance. This time five vehicles completed the course [9]. Further, in 2007, DARPA again sponsored Grand Challenge III. In this race, a 2007 Chevy Tahoe autonomous car from Carnegie Mellon University earned the 1st place. Also, Lidar quickly became an integral sensor for self-driving vehicles, and five out of six of the vehicles that finished used a Lidar [10].

The 2010s decade saw a significant increase in the research and development activities of autonomous driving cars. Many major automotive manufacturers, including General Motors, Ford, Mercedes-Benz, Volkswagen, Audi, Nissan, Toyota, BMW, and Volvo got involved in this area. In 2014, Google announced plans to unveil 100 autonomous car prototypes built from scratch inside Google's secret X lab, as manifestations of years of work that began by modifying existing vehicles [11]. In the same year, Tesla Motors announced its first version of Autopilot. Model S cars equipped with this system are capable of lane control with autonomous steering, braking, and speed limit adjustment based on signal image recognition [12]. The present decade is also going through some important research and commercial activities in the autonomous driving field, especially in sensor integration and fusion, novel algorithms for perception, and path planning. This trend is aided by the new GPUs and accelerator chips which offer low-latency computations with reduced power and area along with research activities in the artificial intelligence domain. Considering the strong demand and government funding, the future of autonomous driving cars is bright and competitive.

1.2 Technologies in Autonomous Driving

Autonomous driving combines advanced technologies and smart designs to allow vehicles to operate without human help. These technologies involve multiple sensors for understanding the scenario, navigation, and control, different architectures and algorithms, etc. A variety of sensors can be used by the vehicle to understand the surrounding scenario and create multiple layers of perception maps [13], [14]. These sensors include LIDAR (Light Detection and Ranging), Radar (Radio Detection and Ranging), cameras, ultrasonic sensors, etc. Lidar uses

laser beams to measure distances to surrounding objects to create a detailed 3D map of the environment. Radar systems use radio waves to detect objects and measure their speed and distance. Cameras provide visual information about the environment and are used for lane detection, traffic sign recognition, object detection, and classification. Finally, ultrasonic sensors are used for short-range detection and are commonly employed in parking assistance systems. A comparison of each of the sensors is tabulated in table 1.1.

Sensor	**Advantages**	**Disadvantages**
Lidar	High-resolution 3D maps	High cost
	Accurate distance measurement	Affected by adverse weather
Radar	Reliable in all weather conditions	Lower resolution
	Can measure the object's speed	Limited ability to distinguish objects
Camera	Provides rich information	Affected by lighting conditions
	Essential for recognizing traffic signs	Limited depth perception
Ultrasonic	Excellent for short-range detection	Limited range
	Inexpensive and easy to integrate	Low resolution

Table 1.1: Comparison of sensors for perception

Now, for localization of the vehicle on the outside world, different sensors can be used like Global Positioning System (GPS), Inertial Navigation Systems (INS), Visual Odometry (VO), Radio Frequency Identification (RFID), etc. In GPS, the satellite-based navigation system is used to provide location and time information anywhere on Earth. INS utilizes sensors such as accelerometers and gyroscopes to continuously track the vehicle's position, velocity, and orientation relative to a starting point. VO estimates the vehicle's motion by analyzing consecutive images captured by onboard cameras. Additionally, RFID tags embedded in the environment provide localization information to the vehicle as it passes by. A comparison of each of the sensors is tabulated in table 1.2.

Now, different architectures are used for autonomous driving, including modular rule-based methods, end-to-end learning, reinforcement learning (RL), etc. Modular rule-based methods divide the autonomous driving task into separate modules, each responsible for specific functions like perception, planning, and control, utilizing rule-based algorithms. On the other hand, end-to-end learning trains a neural network to directly map sensory inputs to driving actions without explicitly designing intermediate representations, allowing for a holistic approach to driving behavior. Finally, reinforcement learning teaches the vehicle to make decisions by trial

Sensor	Advantages	Disadvantages
GPS	Global coverage	Signal blockage in urban area
	Accurate over long distances	Reduced accuracy in dense vegetation
INS	Continuously available	Drift over time
	Accurate short-term positioning	Sensitive to external disturbances
VO	Accurate in feature-rich environments	Limited accuracy in poor environments
	Operate without external infrastructure	Susceptible to motion blur and lighting
RFID	Passive operation	Limited range
	Low cost per tag	Susceptible to interference

Table 1.2: Comparison of localization sensors

and error, receiving rewards or penalties based on its actions, enabling adaptive and dynamic behavior in uncertain environments. The selection of these architectures depends on the complexity of the design, modularity, and performance cost.

In addition to these fundamental technologies, autonomous driving vehicles also implement advanced features like Vehicle-to-Everything (V2X) communication [15] to interact with other vehicles, infrastructure, pedestrians, and the cloud for situational awareness and enable cooperative driving strategies, Cloud computing to increase the computational power needed for processing large amounts of data and running complex AI models, and Cybersecurity for ensuring the security of the vehicles.

1.3 Motivation

In the current decade, there is a significant shift underway to redefine the transportation sector. While advancements in vehicle and road technologies have been limited since the early days of motor transportation, there is now a pressing need to implement substantial improvements at the architectural level to enhance user comfort and infrastructure efficiency. One notable development in this transformation is the rise of Mobility as a Service (MaaS) [16], [17], which integrates various forms of transportation services into a unified platform accessible through digital interfaces. MaaS initiatives aim to optimize the use of existing infrastructure, reduce congestion, and provide seamless, user-centric mobility experiences.

MaaS is enabled by the concept of PACE - Personalized, Autonomous, Connected, and Electric.

Personalized - Tailoring transportation services to individual preferences and needs, offering customized routes, schedules, and modes of travel to enhance user experience and satisfaction.
Autonomous - Integrating autonomous vehicles into transportation networks to enhance safety, efficiency, and accessibility, reducing the need for human drivers.
Connected - Leveraging connectivity technologies to facilitate seamless communication between transportation modes, infrastructure, and users, optimizing routing and enhancing real-time information sharing.
Electric - Utilization of electric propulsion systems, contributing to reduced emissions and environmental sustainability in the transportation ecosystem.

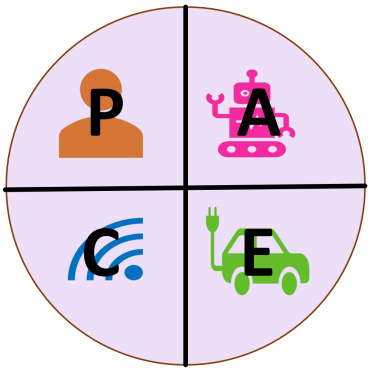


Figure 1.4: Pillars of MaaS

By building a strong foundation in each of the pillars of MaaS, the future transportation sector is expected to make a major revolution. The Department of Electronics Systems Engineering (DESE) has a long term vision to realize a working model of MaaS which can be deployed in the Indian scenario. Different teams across the department are focusing on each key pillar which will be finally integrated into a common platform. This MTech project titled ”Control Architecture for Autonomous/ Assisted Driving of Electric Vehicle” focuses on implementing the autonomous driving pillar of the PACE architecture. The primary motivation for pursuing autonomous technology in cars to to enhance the safety of the person inside the car and for the people on the road.

Studies related to autonomous driving vehicles are undertaken in several institutions ranging from academics to industry. History shows that there were multiple car Original Equipment Manufacturers (OEMs) and educational institutions that have pursued autonomous driving technology. Along with that, the improved scenarios in the artificial intelligence and semiconductor domains are accelerating the adoption of this technology from academia to a commercially viable option. Despite these advancements in autonomous driving technology, there

are several challenges associated with them. Firstly, the present autonomous driving cars rely on a fusion of sensors for perception and localization ranging from camera, Lidar, radar, ultrasonic, etc as mentioned earlier. For instance, the Jaguar I-PACE of Waymo is fitted with 4 Lidars, 8 cameras, and 4 radars to implement autonomous driving features [18]. These vast numbers of sensors and the fusion of their outputs make the system more complex, which will demand a dedicated hardware accelerator to process them in real-time. This will in turn limit the development of these products among a niche group of companies which can increase the product price due to supply shortage. To overcome this dependency of the onboard algorithms for autonomous driving on sensor fusion, the usage of monocular camera [19], [20] for perception and developing strong algorithms based on it can be pursued. This will make the overall system simple as well as modular for wider usage.

Along with simplifying the sensor usage, the complex nature of the complete autonomous driving system makes it demanding for the system engineer. The tasks involved in designing an autonomous driving car range from architecture and algorithm design, hardware design for meeting the needs, designing the steering motor and speed control, interaction with the other subsystems, etc. To fulfill these complex diverse tasks requires an extensive and thorough understanding of the system, proper planning and the necessary skills to implement them making it extremely challenging in an academic environment.

Moreover, the entire design has to also follow certain features like

- Simplistic algorithm
  - Run key algorithms with information extraction from image frames
  - Facilitate use of modular components
- High speed execution
  - Achieve industry-competent execution speed
  - Lower the computational power
  - Fractional memory requirement
- Implement the entire control loop on an electric car for proof of concept

In addition to the safety features of autonomous driving cars, they will also provide new mobility options for the disabled and reduce pollution due to increased autonomous public transport that will be implemented on electric drives. Thus, the motivation of this project is to combine

the virtues of driver-assisted cars with those of the electric vehicle, a powerful amalgamation. The project aims to develop a fast, accurate, and simplistic system for autonomous or assisted driving.

1.4 Scope

The Society of Automotive Engineers (SAE) has defined six levels of driving automation, outlining the degree of control a vehicle has versus the driver. Level 0 represents no automation, with the driver responsible for everything as tabulated in table 1.3. Levels 1 and 2 offer driver assistance features like adaptive cruise control or lane departure warnings, but the driver must stay engaged. Level 3 allows for conditional automation, where the car can handle some tasks in specific environments, but the driver needs to be ready to take over when prompted. Levels 4 and 5 signify high and full automation respectively, where the car can manage most or all driving situations without human intervention, though operational areas may be limited [21].

Level	Condition	Monitoring	Steering	Fall-back	Operations
0	No Automation	Human	Driver	Driver	Limited
1	Driver Assistance	Human	Both	Driver	Limited
2	Partial Automation	Human	Autonomous	Driver	Limited
3	Conditional Automation	Sensors	Autonomous	Driver	Limited
4	High Automation	Sensors	Autonomous	Autonomous	Limited
5	Full Automation	Sensors	Autonomous	Autonomous	Unlimited

Table 1.3: SAE Levels of Driving Automation

Some examples of cars in each of the SAE levels are given in table 1.4 [22].

Level	Examples
0	Maruti Alto, Tata Indica
1	Honda City, Tata Harrier
2	Tesla Model S, GM Cadillac
3	Audi A8L
4	Alphabet Waymo, Magna Max4
5	Swayatt Robotics (not certified)

Table 1.4: Examples of cars in each SAE level

The main aim of the project is to develop an autonomous driving electric vehicle that meets the requirements of SAE level 3, i.e. Conditional Automation. At this level, both the monitoring of the surroundings and control actions are taken by the autonomous system. But in case of any emergency, the driver has to take appropriate action for the safety of the passengers, vehicle, and the surrounding elements. However, there are several constraints put on the project to achieve tangible outcomes within the stipulated time period of the project.

The scope of this thesis will be to solve the problems related to autonomous driving using a monocular camera on left-hand side drive roads. We aim at developing an end-to-end proof of concept for the implementation of the camera perception pipeline with minimum latency and data overhead. The following will be the scope of work carried out in this thesis

- Use the monocular camera as the primary sensor to process visual of the road, extract 3D information and generate references for steering angle and speed of the host vehicle
- Design comprehensive algorithms and implement hardware for
  - Collision avoidance warning system
  - Automatic lane centering
- Develop a fast and simplistic solution for real-time applications supporting
  - Object detection
  - Depth estimation
  - Lane detection
  - Pothole detection
  - Generating surrounding perception
  - Path planning
  - Sensors interface and steering control
- Design a hardware platform to implement the algorithms
- Demonstrate the autonomous driving capability of the vehicle

1.5 Thesis Organization

Chapter 2 provides a comprehensive analysis of the system architecture for the autonomous driving electric vehicle, detailing the key component, the Intelligent Mobility Module (IMM) along with its design criteria and architectural overview. Additionally, it includes a study on selecting a suitable controller for the IMM.

Chapter 3 delves into various perception algorithms essential for autonomous driving. It examines object detection, depth estimation, lane detection, and pothole detection algorithms, elucidating their roles in enabling the vehicle to identify objects, estimate distances, maintain lane integrity, and detect road hazards.

Chapter 4 explores the components of route planning and control in depth. It covers the localization module for determining the vehicle's precise position, the path planning module for generating routes, the reference generation module for translating plans into actionable commands, and the control law for ensuring precise and responsive vehicle steering.

Chapter 5 delves into machine learning-based data-driven approaches for reference generation to improve the performance in autonomous driving vehicles, focusing on end-to-end learning and reinforcement learning methods.

In Chapter 6, hardware implementation aspects and software design of algorithms are discussed. The hardware design covers circuit design and sensor interface to the IMM, while the software section explores algorithm implementation on the controller.

Chapter 7 presents the results obtained from the project. It evaluates hardware performance, particularly the IMM, followed by software performance analysis. Additionally, system performance in an emulated environment is discussed.

Finally, Chapter 8 concludes the thesis, summarizing key findings and suggesting future research directions in autonomous driving technology.

Throughout the document, the vehicle that is being controlled and monitored by the autonomous driving system is addressed as 'ego vehicle' to establish a clear reference point from which all observations, measurements, and decisions are made.

Chapter 2

System Architecture

This chapter describes a detailed overview of the system architecture of the autonomous driving electric vehicle. This project is implemented in an in-house designed Tabby car. The design is based on an open-source framework by *Open Motors* [23]. The system details of the vehicle are provided in table 2.1 and the vehicle is shown in Fig. 2.1.

Figure 2.1: Open motors hardware electric vehicle platform

Parameter	Unit	Value
Dimension	mm	2330 x 1488 x 1380
Weight without batteries	kg	380
Weight with batteries	kg	520
Wheel circumference	mm	1250

Table 2.1: System details of the vehicle

The vehicle has a steel S235JR chassis and four numbers of Li-ion batteries each of 48V in series, which makes it 184V for vehicle propulsion.

The system interface of different modules in the vehicle is shown in Fig. 2.2.

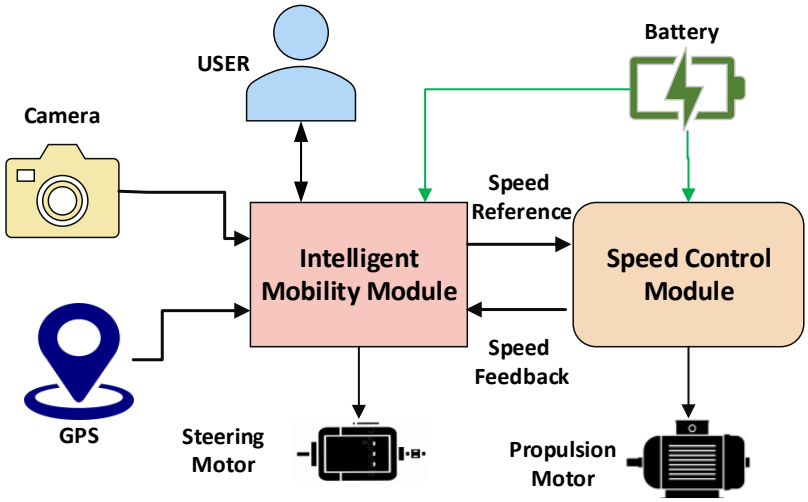


Figure 2.2: Overall system interface of the vehicle

The vehicle consists of two electronics modules namely the Intelligent Mobility Module (IMM) and Speed Control Module (SCM), two actuating elements - a DC motor for steering control and an induction motor for propulsion control, and two sensors for perception and localization - camera and GPS. Here, the Intelligent Mobility Module forms the Central Platform Computer of the vehicle and the Speed Control Module forms the peripheral unit for controlling the vehicle's speed.

The IMM acts as the brain of the autonomous car. It's a powerful computer responsible for high-level decision-making, processing sensor data, and controlling the vehicle's movement. The key functionalities of IMM are:

- **Sensor Fusion:** The IMM receives data from various sensors like the camera and GPS. It combines this information to create a comprehensive understanding of the car's surroundings.
- **Localization and Mapping:** Using the fused sensor data, the IMM determines the car's precise location within its environment.
- **Path Planning and Decision Making:** Based on the perceived environment and a predetermined destination, the IMM creates a safe and efficient path for the car to follow. This path is predetermined beforehand and stored on board (not computed in real-time). The decision-making involves factors like lanes, potholes, obstacles, and pedestrian crossings.

- **Reference Generation:** The IMM translates its path-planning decisions into actionable commands for the actuators. It sends reference signals to the SCM for speed adjustments and to the DC motor for steering control.
- **User Interface:** The IMM also interfaces with the user for receiving inputs and displaying outputs. The inputs involve Start and Destination locations, and Start and Stop commands. The output displays the detected lanes and the obstacles in front of the vehicle and also emergency alarms. These outputs can be used by the user to take emergency actions if any,

The SCM acts as the execution unit for speed control in the vehicle. It bridges the gap between the high-level commands from the IMM and the physical operation of the motor for speed control. The system employs an induction motor operating at 184V and a peak current of 100A. The key functionalities of SCM are:

- **Speed Reference Reception:**The SCM receives the desired speed reference signal from the IMM. This signal translates the IMM's path-planning decisions into a specific motor speed target.
- **Induction Motor Control:** Based on the received speed reference, the SCM regulates the power delivered to the induction motor. V/f control scheme is used for controlling the motor
- **Real-time Speed Monitoring:** The SCM constantly monitors the actual speed of the induction motor using a Hall sensor. This feedback loop allows the SCM to make adjustments to the motor control signals if needed to ensure the car maintains the desired speed reference. This speed feedback information is also transmitted to the IMM for monitoring and also to make suitable adjustments in the planning.

The vehicle has two actuators for speed (propulsion) and steering control. An in-house designed induction motor is used for speed control. Steering control is done using a purchased power steering motor present in Maruti Alto. This is a DC steering motor operating at 12V and 6A peak current.

The vehicle is equipped with two sensors for perception and localization. The first sensor is a camera, specifically the Logitech C310 as shown in Fig. 2.3, which is utilized to capture the scene in front of the vehicle. This camera is securely mounted on a metal bar positioned at the

Figure 2.3: Logitech C310 camera used for perception

Parameter	Unit	Value
Dimension	mm	26 x 31 x 71
Weight	g	250
FOV	deg	60
Resolution		720p
Connector type		USB

Table 2.2: Technical details of the camera

front of the vehicle, providing a clear and stable view of the surroundings. Technical details of the camera are tabulated in table 2.2 [24].

The second sensor used is a GPS module by Adafruit for localization. The output of the GPS module provides the present latitude and longitude of the vehicle. This information is used for path planning and reference generation in the next time instant. The GPS module is fitted on the IMM PCB to reduce the interface cable length. The GPS module, Adafruit Ultimate GPS Breakout v3 as shown in Fig. 4.3 is built around the MTK3339 chipset and can track up to 22 satellites on 66 channels with an excellent high-sensitivity receiver (-165 dB). Technical details of the GPS module are tabulated in table 4.1 [25].

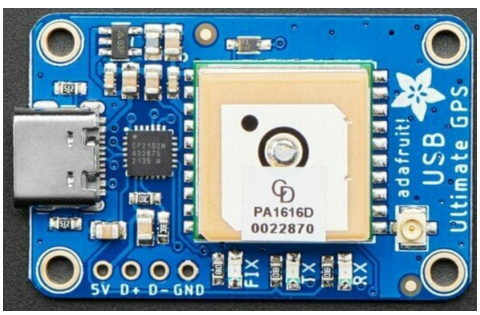


Figure 2.4: Adafruit Ultimate GPS Breakout v3

Parameter	Unit	Value
Dimension	mm	15 x 15 x 4
Update rate	Hz	1-10
Position accuracy	m	1.8
Acquisition sensitivity	dBm	-145
Connector type		USB

Table 2.3: Technical details of the GPS module

2.1 Design of IMM

The scope of this project is to design the Intelligent Mobility Module. Henceforth, the focus of the report will be on discussing the various technical details of the IMM. This section discusses in detail the design of the Intelligent Mobility Module. The block diagram view of the IMM is shown in Fig. 4.1.

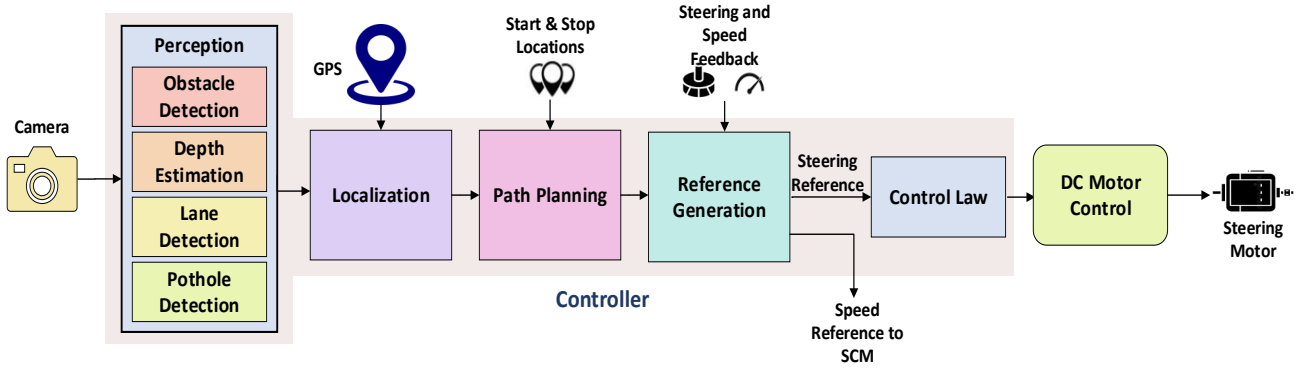


Figure 2.5: Block diagram of the IMM

The IMM consists of a Controller that implements the different algorithms and programs for sensors interface and reference generation. It also consists of analog circuits for implementing closed-loop control of the steering DC motor. Operations of the controller begin with the camera capturing the scene ahead, which the perception module processes to detect obstacles, estimate depth, identify lanes, and recognize potholes. This processed data, along with GPS inputs, feeds into the localization module to determine the vehicle's precise position. The localization information is then used by the path planning module to calculate the optimal route. The path planning outputs are translated into specific steering and speed references by the reference generation module. These references are processed by the control law module to formulate precise control actions, which are executed by the DC Motor Control module to adjust the

steering motor. Throughout this process, feedback on steering and speed is continuously monitored to ensure real-time adjustments and accurate vehicle movement. At a top level, the entire algorithm can be represented as an outer guidance loop and an inner control loop as shown in Fig. 6.21. The outer loop consisting of perception, localization, path planning, and reference generation is slow defined by the processing capability of the controller. However, the inner control loop runs at a faster rate for the bandwidth and stability requirements of the system.

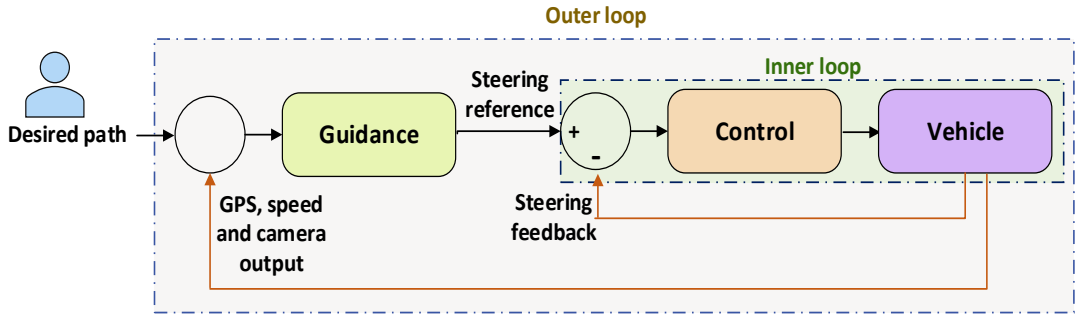


Figure 2.6: Control loops in the system

Details of the Controller and the DC motor control are discussed in the next subsections.

2.1.1 Controller

The controller in the autonomous driving vehicle is an intricate system designed to coordinate various subsystems, ensuring the vehicle navigates safely and efficiently. The following is a detailed breakdown of the controller's components and their functions.

2.1.1.1 Perception

Perception is the initial phase where the vehicle gathers environmental data. The primary sensor employed is a Logitech C310 camera, mounted on a metal bar at the front of the vehicle. This camera captures high-resolution images of the scene ahead, serving multiple critical functions:

Obstacle Detection: The system identifies potential obstacles such as pedestrians, vehicles, and other objects that may impede the vehicle's path. Advanced image processing algorithms detect and classify these obstacles, ensuring the vehicle can take appropriate action to avoid collisions.

Depth Estimation: By analyzing the images, the perception system estimates the distance

Figure 2.7: Obstacle detection

to various objects. Depth estimation is vital for understanding the spatial arrangement of the environment and making informed decisions about maneuvering.

Figure 2.8: Depth estimation

Lane Detection: The system recognizes lane markings on the road, which is essential for maintaining lane discipline and ensuring the vehicle follows the correct path. This involves detecting the side lanes to ensure the vehicle follows the proper track.

Figure 2.9: Lane detection

Pothole Detection: Identifying potholes and other road surface anomalies is crucial for vehicle safety and passenger comfort. The camera captures detailed images of the road, and specialized algorithms detect irregularities, allowing the vehicle to adjust its path or speed to avoid these hazards.

Figure 2.10: Pothole detection

The perception algorithm can be implemented in two different ways, computer vision algorithms and machine learning (ML) algorithms. Computer vision algorithms rely on predefined rules and handcrafted features to interpret visual data, using techniques such as edge detection, color thresholding, and geometric analysis. While these methods can be effective for specific tasks and simpler environments, they often struggle with complex, real-world scenarios due to their limited adaptability and robustness. In contrast, machine learning (ML) algorithms, particularly those based on deep learning, offer superior performance for perception tasks in autonomous driving. ML algorithms learn from vast amounts of data, enabling them to automatically extract and optimize features that capture intricate patterns and variations in the environment. They can automatically improve their performance as more data becomes available, making them more robust and scalable for complex perception tasks in autonomous driving. So considering these advantages of ML-based algorithms, they are employed for all of the perception algorithms instead of first principles-based computer vision algorithms.

2.1.1.2 Localization

Localization is the process of determining the vehicle's precise position in the world. This module integrates data from the GPS and the perception system. The GPS provides global positioning data, offering a rough estimate of the vehicle's location. The perception information is also used along with the GPS information to obtain a highly accurate position of the vehicle.

2.1.1.3 Path Planning

Path planning involves determining the optimal route from the vehicle's current position to the desired destination. Based on the Start and Destination locations, a route is generated for the journey. Based on the current location, the planning algorithm makes suitable adjustments for arriving at the destination.

2.1.1.4 Reference Generation

The reference generation module translates the path planning data into actionable commands. **Steering Reference**: The steering reference module calculates the precise steering angle needed to follow the planned path. It ensures the vehicle stays on course by adjusting the steering wheel as necessary.

Speed Reference: The speed reference module determines the optimal speed based on the path plan and current traffic conditions. It generates speed commands that ensure the vehicle travels at a safe and efficient pace, considering speed limits and traffic flow.

2.1.1.5 Control Law

To track the steering command at all conditions, a negative feedback control loop is implemented in the controller. The control law in the negative feedback loop is a critical component in ensuring precise and responsive vehicle steering. In this system, the control actions generated by the control law module are sent to the DC motor, which adjusts the steering mechanism according to the desired steering angle. A key element in this loop is the potentiometer, which measures the actual steering angle of the wheels. This real-time measurement is continuously fed back into the control system, allowing it to compare the actual steering angle with the desired angle specified by the reference generation module. Any discrepancy between the actual and desired angles is detected and corrected by the control system, which adjusts the motor's input to minimize the error. This negative feedback loop ensures that the steering adjustments are precise, stable, and responsive to changing conditions. The control law finally generates a Pulse Width Modulated (PWM) signal to the DC motor control.

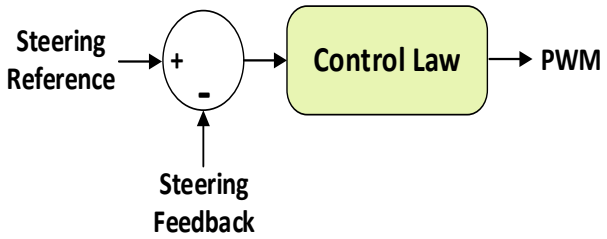


Figure 2.11: Control law for steering motor control

2.1.2 DC Motor Control

In the steering DC motor control system with a gate driver and H-bridge configuration, PWM is utilized as the input signal for controlling motor direction. When the PWM signal exceeds 50%, the motor is instructed to turn in the right direction, while if it falls below 50%, the motor turns in the left direction. When the PWM signal is exactly at 50%, there's no change in the motor's direction, maintaining its current position. This setup ensures precise and efficient control over the motor's movement, enabling smooth transitions between different directions based on the PWM input. The gate driver and H-bridge components play crucial roles in translating the PWM signals into the appropriate motor movements, ensuring reliable operation and accurate control of the steering system of the vehicle.

The previous sections highlighted brief details about the individual block diagrams of the IMM. A detailed discussion about these modules will be carried out in the subsequent chapters.

2.2 Controller Selection

In an autonomous driving vehicle, the controller serves as the central nervous system, orchestrating the seamless operation of various subsystems to ensure safe and efficient navigation. Its significance cannot be overstated. When selecting a controller for an autonomous driving vehicle, several parameters warrant careful consideration. Firstly, scalability is vital to accommodate the increasing complexity of algorithms and data processing as additional features are integrated. Secondly, affordability and availability are critical to ensure cost-effective production and avoid supply chain disruptions. Lastly, access to comprehensive online software support and a robust ecosystem of development tools and reference designs facilitates efficient algorithm development and integration, ultimately contributing to the success and viability of autonomous vehicle technology.

A thorough study is carried out to select a suitable controller that meets all the functionalities and requirements for the autonomous driving application. In the study, five different platforms were considered namely: a microprocessor, an FPGA, a digital signal processor, Raspberry Pi, and a CPU & GPU combination. Details of each of these devices are given in the subsequent sections.

2.2.1 Microprocessor - Atmega328

The ATmega328 microprocessor is a popular 8-bit microcontroller designed by Atmel, now owned by Microchip Technology. It operates at a clock speed of up to 20 MHz and features 32KB of flash memory, 2KB of SRAM, and 1KB of EEPROM [26]. It also supports a wide range of peripherals, including analog-to-digital converters, timers, and serial communication interfaces.

In terms of advantages for implementing a autonomous driving car algorithm, the ATmega328 offers low power consumption, making it suitable for applications that require efficient energy usage. It is also cost-effective and readily available, which can help reduce the overall production costs of a self-driving car system.

However, the ATmega328 microprocessor has certain limitations when it comes to implementing a autonomous driving car algorithm. Its 8-bit architecture and limited processing power make it less capable of handling complex and computationally intensive tasks required for real-time decision-making in autonomous driving scenarios. The limited amount of memory might also pose challenges when dealing with large datasets or complex algorithms.

Overall, while the ATmega328 microprocessor offers some advantages for implementing certain aspects of a autonomous driving car algorithm, its limitations in terms of processing power, memory, and I/O capabilities may hinder its ability to handle the demanding requirements of a comprehensive autonomous driving system.

2.2.2 FPGA - Artix 7

The Artix-7 FPGA is a high-performance field-programmable gate array developed by Xilinx. It offers a range of quantitative specifications, including logic cell capacity of up to 215,360, up to 13,500 slices, 1,800 kilobits of block RAM, and up to 6.6 million system logic cells [27]. It also supports a variety of high-speed transceivers, making it suitable for applications that require high-speed data processing and communication.

When it comes to implementing a autonomous driving car algorithm, the Artix-7 FPGA offers several advantages. Its high logic cell capacity and abundant resources allow for complex algorithm implementation, making it capable of handling computationally intensive tasks required for real-time decision-making in autonomous driving scenarios. The FPGA's parallel processing capabilities enable efficient execution of algorithms in parallel, further enhancing performance.

However, there are some disadvantages to consider when using the Artix-7 FPGA for autonomous driving car applications. Designing and programming an FPGA-based system requires specialized knowledge and expertise, making it a more complex task compared to traditional microcontroller-based implementations. FPGA development also typically involves longer development cycles compared to microcontrollers.

In summary, the Artix-7 FPGA provides significant advantages for implementing autonomous driving car algorithms, including high performance, parallel processing capabilities, and reprogrammability. However, the higher cost, complexity of design and programming, and longer development cycles make the FPGA less suitable for autonomous driving applications.

2.2.3 Digital Signal Processor - dsPIC/TMS320

The dsPIC (Digital Signal Controller) is a family of 16-bit microcontrollers developed by Microchip Technology. It combines the capabilities of a microcontroller and a digital signal processor (DSP), making it suitable for applications that require high-performance signal processing [28].

The dsPIC microcontroller provides several advantages for implementing autonomous driving car algorithms. Its DSP architecture and specialized instructions enable efficient and rapid signal processing, making it ideal for sensor data acquisition, filtering, and control calculations required in autonomous driving. The microcontroller's high-speed clock and generous memory capacity allow for real-time processing of complex algorithms, such as object detection and recognition.

However, there are some limitations to consider when using dsPIC for implementing autonomous driving car algorithms. Although it offers higher performance compared to traditional microcontrollers, it may still have limitations in terms of processing power and memory compared to more advanced processors or FPGAs. The development tools and software ecosystem for dsPIC may not be as extensive as those available for mainstream microprocessors, which can pose challenges in terms of algorithm development and debugging.

In summary, the dsPIC microcontroller offers significant advantages for implementing autonomous driving car algorithms, including its DSP capabilities, high-speed clock, and integrated peripherals. However, its limitations in terms of processing power, memory, and software support make it less suitable for autonomous driving applications.

2.2.4 Raspberry Pi

The Raspberry Pi is a credit card-sized single-board computer developed by the Raspberry Pi Foundation. It offers a range of models, but the latest Raspberry Pi 5 features a quad-core ARM Cortex-A76 processor running at 2.4GHz, up to 8GB of RAM, and support for various operating systems [29]. It also includes USB ports, HDMI output, GPIO pins, and onboard Wi-Fi and Bluetooth connectivity.

The Raspberry Pi provides several advantages for implementing autonomous driving car algorithms. Its compact size, low power consumption, and affordability make it an accessible choice for prototyping and testing autonomous driving systems. The GPIO pins allow for easy integration with sensors, actuators, and other external devices. Additionally, the Raspberry Pi's Linux-based operating system and wide community support provide access to a vast range of software libraries and resources, simplifying algorithm development and deployment.

However, there are certain limitations when using Raspberry Pi for implementing autonomous driving car algorithms. Its ARM-based architecture and lower processing power compared to dedicated microprocessors or FPGAs may restrict the performance and real-time capabilities needed for complex algorithms in real-world driving scenarios. Additionally, the device is not tailor-made for machine learning applications as demanded by the autonomous driving car.

In summary, the Raspberry Pi offers advantages such as affordability, community support, and ease of integration, making it a suitable choice for prototyping and testing autonomous driving car algorithms. However, its limitations in terms of processing power, memory, and real-time capabilities may impact its suitability for implementing comprehensive autonomous driving systems that require high-performance computing and real-time decision-making.

2.2.5 CPU-GPU Combination - Jetson Orin Nano

The Jetson Orin Nano is a small, low-power single-board computer developed by NVIDIA. It is specifically designed for AI and robotics applications, including autonomous driving cars. The Jetson Orin Nano features a six-core ARM Cortex-A78 CPU, a 1024-core NVIDIA Ampere GPU, 8GB of LPDDR4 RAM, and support for various peripherals and interfaces [30].

The Jetson Orin Nano offers several advantages for implementing autonomous driving car algorithms. Its powerful GPU and dedicated AI computing capabilities enable efficient and accelerated processing of machine learning and computer vision algorithms, making it suitable for tasks such as object detection, lane tracking, and depth estimation. The Jetson's hardware-

accelerated video decoding and encoding capabilities are beneficial for handling real-time camera feeds. Additionally, the availability of NVIDIA's extensive software development kits, such as CUDA and TensorRT, simplifies the optimization and deployment of AI algorithms on the Jetson Nano platform.

However, there are a few considerations when using the Jetson Orin Nano for autonomous driving car algorithms. The power consumption of the Jetson Orin Nano is relatively higher compared to other single-board computers, requiring appropriate power management for automotive applications. The cost of the Jetson Orin Nano is also higher compared to basic microcontrollers or Raspberry Pi, which may impact the overall production cost of an autonomous driving car system.

In summary, the Jetson Orin Nano offers significant advantages for implementing autonomous driving car algorithms, including its powerful GPU, AI computing capabilities, and extensive software development kits. However, considerations such as power consumption, cost, and limited I/O capabilities have to be taken into account when choosing it for autonomous driving applications.

Figure 2.12: Jetson Orin Nano module

A comparison of these platforms is tabulated in table 2.4.

From table 2.4, it can be observed that Jetson Orin Nano is superior to other platforms for the project on autonomous driving cars. Jetson Orin Nano has sufficiently good on-chip memory and computation capability (For storing one 720p image, 2.7MB memory is required). Also, the chip supports different communication peripherals and hence the custom PCB can be made scalable for future projects related to autonomous driving cars. (For eg: interfacing GPS, Lidar, Radar, etc). So Jetson Orin Nano is selected as controller for the project.

In this chapter, we have provided an in-depth examination of the system architecture for the autonomous driving electric vehicle (EV). The entire system is composed of several key com-

Feature	μ C	FPGA/SoC	DSP	Raspberry Pi	GPU
Device	Atmega 328	Xilinx Artix / ZynQ	dsPIC/ TMS320	Rpi 5	Jetson Orin Nano
Processor	Atmel 8 AVR RISC	CLB	C62x	Cortex A76 64bit	A78 & 1024 CUDA core
On-chip RAM	2kB	1.6MB	375kB	8GB	8GB
Image processing	Not supported	Yes, but computational speed limited		Yes, display/camera interface supported	
Clock speed	20MHz	$<$ 500MHz	$<$ 300MHz	2.4GHz	1.5GHz
Package	DIP/TQFP	BGA	BGA	Mountable system on module	

Table 2.4: Comparison of different embedded platforms

ponents: a central platform computer called the Intelligent Mobility Module (IMM), a Speed Control Module (SCM), a battery pack, various sensors, and actuation elements. We have also covered the design criteria and architectural overview of each of these components. Additionally, a study was conducted to choose a suitable controller for the IMM.

In the next chapter, we will discuss the various perception algorithms in detail.

Chapter 3

Perception Algorithms

Safe and reliable autonomous driving critically hinges on a car's ability to perceive its surroundings with high accuracy. Camera, a vital sensor in autonomous vehicles, is utilized in conjunction with powerful algorithms to understand the intricate details of the road environment. This chapter delves into the inner workings of these algorithms, focusing on four crucial tasks essential for safe navigation: object detection, depth estimation, lane detection, and pothole detection.

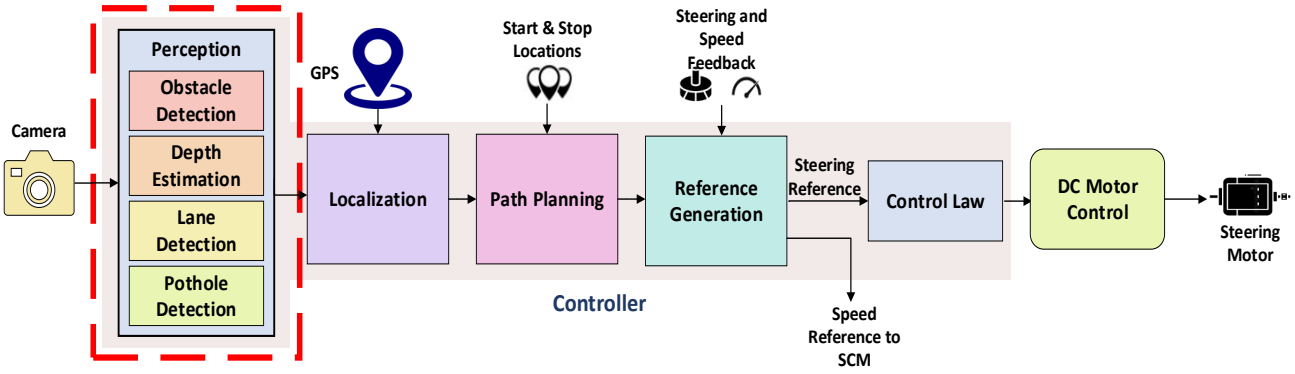


Figure 3.1: Block diagram of the IMM

Each section of this chapter will explore a specific perception algorithm in detail. The first section will examine object detection algorithms, which enable the car to identify and classify objects like pedestrians, vehicles, and traffic signs. This information empowers the car to make safe decisions regarding its path, such as stopping for pedestrians or maintaining a safe distance from other vehicles.

The second section will focus on depth estimation algorithms. These algorithms provide accurate distance information about objects in the environment, which is essential for safe navi-

gation. By understanding the distance to surrounding objects, the car can adjust its speed and trajectory accordingly.

The third section will explore lane detection algorithms. These algorithms ensure the car remains within its designated lane by identifying lane markings on the road. This is crucial for maintaining a safe and orderly flow of traffic.

Finally, the fourth section will delve into pothole detection algorithms. These algorithms play a vital role in identifying and avoiding potholes on the road surface. Pothole detection helps to prevent potential damage to the vehicle and maintains a smooth ride for passengers.

By exploring these four key perception algorithms, this chapter aims to provide a comprehensive understanding of how cameras, combined with powerful processing techniques, enable autonomous vehicles to see the road and navigate their surroundings safely and efficiently.

3.1 Object Detection

For autonomous electric vehicles (EVs) to navigate roadways safely and efficiently, a robust understanding of their surroundings is paramount. Object detection algorithms are fundamental to extracting meaningful information from the visual data. These algorithms analyze camera footage to identify and classify objects in the environment, such as pedestrians, vehicles, traffic signs, animals, etc. The output of this algorithm can be used to incorporate intelligence into the system. For instance, by recognizing pedestrians, the car can initiate braking procedures to ensure safe stopping distances. Similarly, the detection of other vehicles allows the car to maintain appropriate following distances or initiate overtake action and avoid collisions. Additionally, traffic sign recognition enables the car to adhere to traffic regulations. So object detection plays a vital role in enhancing the efficiency, reliability, and safety of autonomous driving vehicles.

There are several approaches to object detection, including both traditional computer vision techniques and modern machine learning algorithms. In traditional computer vision techniques, Histogram of Oriented Gradients (HOG) [31] computes gradients and orientation histograms to represent the appearance of objects in an image. It's effective for detecting objects with well-defined edges and shapes. Another algorithm Scale-Invariant Feature Transform (SIFT) [32] extracts key points and descriptors from images to detect objects invariant to scale, rotation, and illumination changes. These traditional computer vision algorithms suffer from limited flexibility due to their reliance on handcrafted features and predefined rules, often lead-

ing to degraded performance in complex scenarios. Moreover, they require extensive feature engineering and struggle with diverse object shapes and environmental variations. In contrast, machine learning-based algorithms offer end-to-end learning, adaptability to complex data, and superior generalization capabilities. By automatically learning hierarchical representations from raw data, these algorithms eliminate the need for handcrafted features, generalize well to unseen data, and continuously improve through access to large-scale datasets.

There exist several ML algorithms for object detection. Convolutional Neural Network (CNN) is one such algorithm that has revolutionized object detection by learning hierarchical features directly from raw pixel data. Among different CNN architectures for object detection, Single Shot MultiBox Detector (SSD) [33] and You Only Look Once (YOLO) [34] are two popular methods. SSD simultaneously predicts multiple bounding boxes and class probabilities within a single image. Whereas, YOLO divides the input image into a grid and predicts bounding boxes and class probabilities directly from grid cells. YOLO processes images in a single pass through the neural network, enabling faster inference speeds compared to SSD, which involves multiple stages of processing. This makes YOLO particularly well-suited for applications requiring rapid decision-making and real-time object detection, such as autonomous driving. Considering this advantage of YOLO architecture, it is used for object detection in perception. The next section discusses in detail the architecture of YOLO.

3.1.1 You Only Look Once (YOLO) Algorithm

You Only Look Once (YOLO), is a popular real-time object detection algorithm that was first published in 2015. Since then there have been several versions of the YOLO algorithm being published, each contributing an improvement over the previous versions. The latest version is YOLOv10 which was published in May 2024. Though this algorithm is the latest and most advanced version of YOLO in terms of performance and latency, the project is using the YOLOv5 algorithm published in 2020 which is the most mature and widely used algorithm in this family. In this section, firstly the YOLOv1 architecture is explained followed by highlighting the improvements made in YOLOv5 architecture.

YOLO is based on the idea of segmenting an image into smaller images. The image is split into a square grid of dimensions S x S as shown in Fig. 3.2.

The cell in which the center of an object, for instance, the center of the dog, resides, is the cell responsible for detecting that object [35]. Each cell will predict B bounding boxes and a confidence score for each box. These confidence levels capture the model's certainty that there

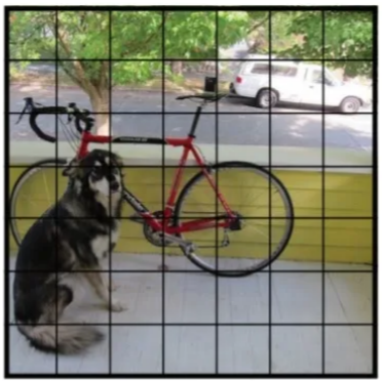


Figure 3.2: S x S grid on input image

exists an object in that cell and that the bounding box is accurate. Each of these bounding boxes is made up of 5 numbers: the *x* position, the *y* position, the width, the height, and the confidence. The coordinates *(x,y)* represent the location of the center of the predicted bounding box, and the width and height are fractions relative to the entire image size. In addition to outputting bounding boxes and confidence scores, each cell predicts the class of the object as shown in Fig. 3.3.

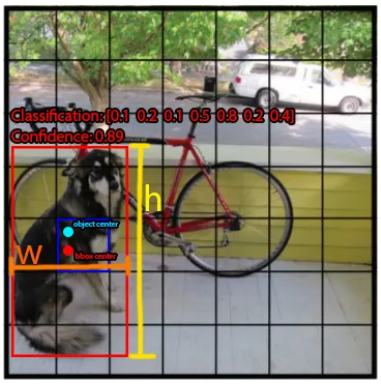


Figure 3.3: YOLO output prediction

This prediction of output by YOLO is based on a CNN architecture. This architecture has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 x 1 convolutional layers reduce the feature space from preceding layers. This architecture is shown in Fig. 3.4.

The YOLO algorithm is pretrained on the ImageNet 1000-class competition dataset [36]. During training, YOLO used the first 20 layers of the architecture followed by an average-pooling layer and a fully connected layer. With this training, YOLO achieved a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set. Even though YOLO was a big success at the time, YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the

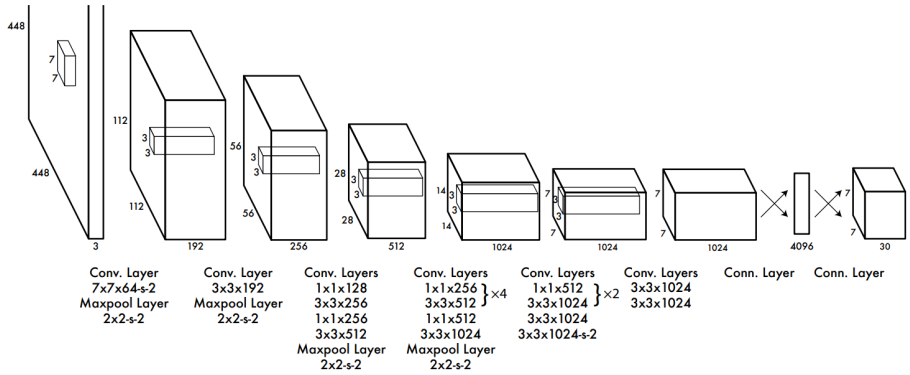


Figure 3.4: YOLO architecture

number of nearby objects that the model can predict. Also, since the model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. These shortcomings were improved in the next versions [37], [38] and [39], but the discussion is confined to that of modifications made in YOLOv5 architecture.

3.1.2 YOLOv5 Architecture

YOLOv5 was published in 2020 by Ultralytics [40]. The inference flow chart of YOLOv5 is shown in Fig. 3.5 [41].

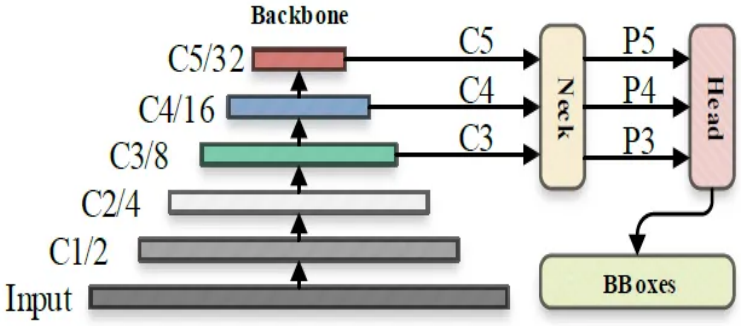


Figure 3.5: Inference flow chart of YOLOv5

The YOLOv5 object detection model begins by processing the image through an input layer, which then passes the data to the backbone for feature extraction. The backbone extracts feature maps of various sizes and subsequently fuses these features through the feature fusion network, known as the neck. This process generates three feature maps - P3, P4, and P5, corresponding to dimensions of 80x80, 40x40, and 20x20, respectively, to detect small, medium, and large objects in the image. These feature maps are then sent to the prediction head, where confidence

calculation and bounding-box regression are executed for each pixel using preset prior anchors. This results in a multi-dimensional array (BBoxes) that includes object class, class confidence, box coordinates, width, and height information. To refine the detection results, thresholds (*confthreshold and objthreshold*) are applied to filter out irrelevant information, and a non-maximum suppression (NMS) process is performed. This sequence of steps ensures that the final detection output is accurate and reliable, effectively identifying objects of varying sizes within the image.

The model architecture of YOLOv5 is shown in Fig. 3.6.

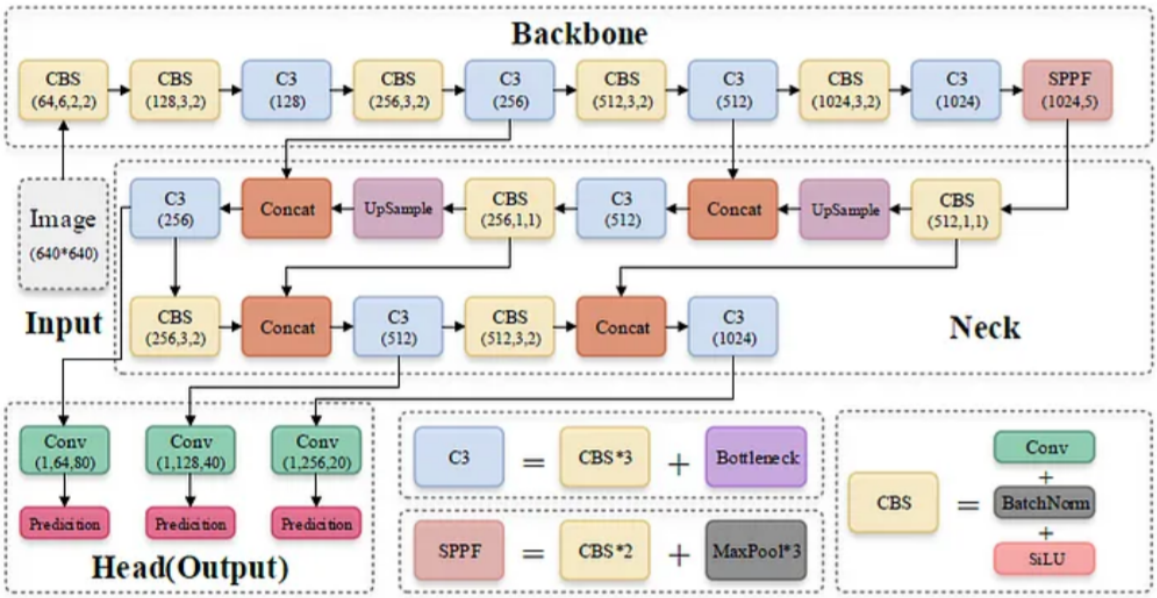


Figure 3.6: YOLOv5 architecture

The backbone of the YOLOv5 model is CSPDarknet53, a robust architecture designed for efficient feature extraction. It primarily consists of multiple stacked CBS (Conv + BatchNorm + SiLU) modules and *C3* modules, culminating in a single SPPF (Spatial Pyramid Pooling - Fast) module as shown in Fig. 3.7. The CBS modules assist the *C3* modules in extracting detailed features from the input image. The inclusion of the SPPF module enhances the feature expression capability of the backbone by pooling features at multiple scales, which enriches the receptive field without significantly increasing computational complexity.

The SPPF module improves upon the traditional SPP (Spatial Pyramid Pooling) method by avoiding redundant operations. Instead of performing multiple max-pooling operations sequentially as in SPPNet, SPPF maximizes the previous max-pooled features in one step. This streamlined approach significantly boosts the running speed of the module, making the YOLOv5 model more efficient and effective in processing images for object detection. Consequently,

YOLOv5 can quickly and accurately identify objects of various sizes, ensuring high performance in real-time applications.

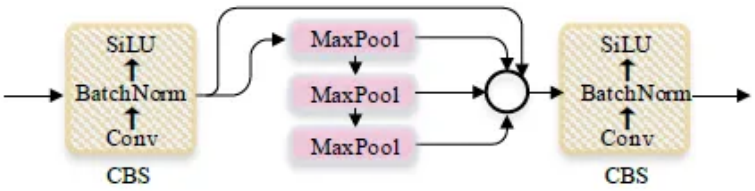


Figure 3.7: Structure of SPPF

The configuration of each of the sub blocks in the backbone and their parameters are mentioned in Fig. 3.8.

C	From	n	Params	Module	Arguments
0	−1	1	3520	CBS	[3, 32, 6, 2, 2]
1	−1	1	18,560	CBS	[32, 64, 3, 2]
2	−1	1	18,816	C3	[64, 64, 1]
3	−1	1	73,984	CBS	[64, 128, 3, 2]
4	−1	2	115,712	C3	[128, 128, 2]
5	−1	1	295,424	CBS	[128, 256, 3, 2]
6	−1	3	625,152	C3	[256, 256, 3]
7	−1	1	1,180,672	CBS	[256, 512, 3, 2]
8	−1	1	1,182,720	C3	[512, 512, 1]
9	−1	1	656,896	SPPF	[512, 512, 5]

Figure 3.8: Parameters of YOLOv5 backbone

In the neck of YOLOv5 (Fig. 3.9), it incorporates Feature Pyramid Network (FPN) and Path Aggregation Network (PAN) methods to enhance its object detection capabilities. The basic idea of FPN is to up-sample the output feature maps (*C3, C4,* and *C5*) generated by multiple convolutional down-sampling operations from the feature extraction network. This process creates multiple new feature maps (*P3, P4,* and *P5*) that are specifically designed to detect targets at different scales, thus improving the model's ability to recognize small, medium, and large objects.

On the other hand, PAN further refines this multi-scale feature representation by enhancing the information flow between different layers. It achieves this by implementing a bottom-up path augmentation, which strengthens the low-level features crucial for precise localization tasks and enhances the high-level semantic features for robust object classification. By integrating FPN and PAN, YOLOv5 ensures a comprehensive feature fusion process that significantly boosts detection accuracy and performance.

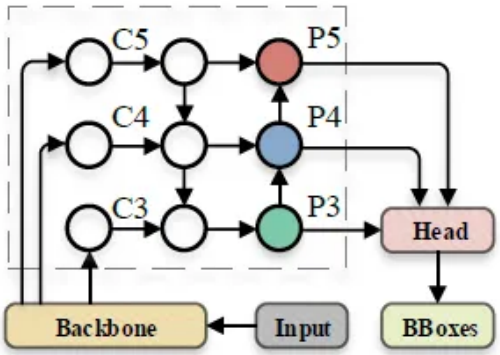


Figure 3.9: Neck of YOLOv5

YOLOv5 comes in five versions, each tailored to different requirements and computational constraints. YOLOv5x represents the largest variant, prioritizing high accuracy for complex object detection tasks, while YOLOv5l offers a slightly reduced parameter count while maintaining robust performance. YOLOv5m strikes a balance between speed and accuracy, making it suitable for real-time applications. YOLOv5s is a smaller and lighter option, designed for deployment on resource-constrained devices without sacrificing detection performance significantly. Lastly, YOLOv5n introduces a variant offering flexibility and adaptability for specialized tasks or custom applications. Since, resources are constrained in the Jetson Orin Nano and as real-time operation is needed for the autonomous driving application, the YOLOv5s model is used for object detection. The YOLOv5s has 10M parameters in total and achieves a mean average precision (mAP) of 37% in the COCO dataset.

3.1.3 Results and Discussion

The perception module utilizes the YOLOv5s pre-trained model for object detection, which has been trained to recognize 80 classes, encompassing a wide range of objects such as cars, motorbikes, people, dogs, bottles, pizzas, and toothbrushes. However, due to its general training, the model may detect objects that are not relevant to typical traffic scenarios. Consequently, the system focuses on the detection of 20 selected classes that are most likely to occur in traffic situations, ensuring accuracy and relevance in object identification. These 20 classes are categorized into 4 categories tabulated in table 3.1.

The output label of the YOLOv5 algorithm is checked to be one among these labels to avoid wrong classification. The camera captured output is passed through the pre-trained YOLOv5s model to obtain the bounding box and the classified label. Results obtained for some sample images are shown in Fig. 3.10.

Category	Classes
Vehicles	car, motorcycle, bus, train, truck, boat, bicycle
Animals	bird, cat, dog, horse, sheep, cow
Traffic control	traffic light, fire hydrant, stop sign, parking meter
Miscellaneous	person, plant, bench

Table 3.1: Classes for object detection

Figure 3.10: Output of YOLOv5 algorithm for object detection

As it can be seen from Fig. 3.10, the YOLOv5s algorithm is able to detect the obstacles with a very good amount of accuracy. However, it has been also noted that there are occasional misses in object detection, particularly when objects are distant or obscured within the image. This limitation is common in image-based object detection algorithms. To address this issue, alternative sensors such as Lidar or radar could be employed, offering complementary data for improved detection accuracy. Alternatively, upgrading to a higher version of the YOLO algorithm could enhance performance, albeit potentially impacting processing latency.

3.2 Depth Estimation

As autonomous driving technology advances, the accurate estimation of depth information emerges as a critical component for ensuring safe and efficient navigation. Depth estimation plays a pivotal role in providing autonomous vehicles with a comprehensive understanding of their surroundings, enabling them to make informed decisions and take appropriate actions. By accurately gauging the distance to obstacles, depth estimation facilitates collision avoidance, path planning, and maneuvering. Additionally, depth information enables precise localization and mapping, enhancing the overall reliability and effectiveness of autonomous driving.

Various sensors, including Lidar, radar, and cameras, are employed for depth estimation in autonomous vehicles. While each sensor type offers unique advantages, cameras have garnered significant attention due to their cost-effectiveness and versatility. Camera-based depth estimation leverages the rich visual information captured by cameras to infer depth cues from the scene. This approach offers high-resolution depth maps and facilitates seamless integration with existing camera-based perception systems in autonomous vehicles.

Camera-based depth estimation algorithms encompass a range of techniques, including stereo vision and single-camera methods based on both conventional computer vision algorithms and machine learning approaches. Stereo vision algorithms utilize pairs of synchronized cameras to triangulate depth information by analyzing disparities between corresponding image points. On the other hand, single-camera techniques exploit monocular cues such as motion parallax, perspective, and object size to estimate depth. These techniques encompass traditional methods like structure-from-motion and depth-from-focus, as well as modern ML-based approaches such as depth estimation networks (DENs) and depth prediction CNNs.

Compared to conventional computer vision techniques for depth estimation, ML algorithms can automatically learn intricate patterns and relationships from large datasets, enabling more accurate depth predictions in diverse and complex environments. Additionally, ML-based approaches exhibit greater adaptability to varying conditions, allowing for improved generalization and robustness in depth estimation tasks. Considering these advantages, an ML-based approach is employed for the depth estimation of obstacles from a single image frame. The present State of the Art monocular depth estimation algorithm (MDE) is Depth Anything by Tiktok [42]. Hence this algorithm is used for depth estimation in the perception. Details of this algorithm are presented in the next subsection.

3.2.1 Depth Anything

The depth Anything model was released by researchers from TikTok, The University of Hong Kong, and Zhejiang Lab in 2024. It marks a notable advancement in AI-driven depth perception, particularly in understanding and interpreting the depth of objects from a single image. At the heart of the model lies its training on a colossal dataset: 1.5 million labeled images, supplemented by over 62 million unlabeled images. This extensive training set is key to the model's proficiency. Unlike traditional MDE models that primarily rely on smaller, labeled datasets, Depth Anything leverages the sheer volume of unlabeled data. This allows the model to learn from a wider variety of scenes and lighting conditions, ultimately surpassing current

state-of-the-art models in depth perception tasks. The architecture of Depth Anything is shown in Fig. 3.11.

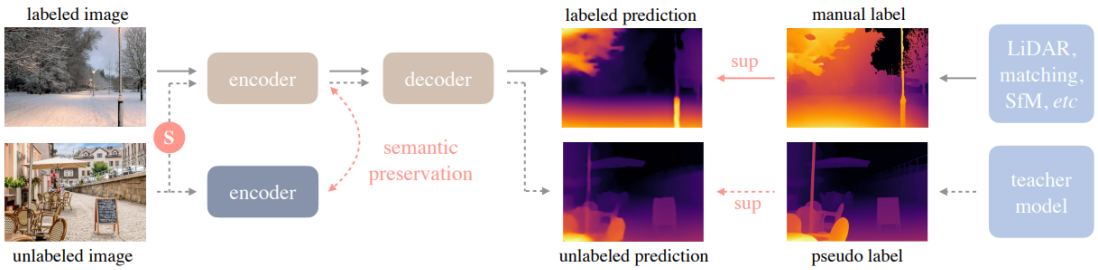


Figure 3.11: Depth Anything architecture

At its essence, Depth Anything employs a well-established encoder-decoder architecture. The encoder, particularly a DINOv2 encoder, serves as a feature extractor, meticulously analyzing the input image to capture essential details concerning shapes, colors, and spatial relationships between objects within the scene. These extracted features, representing a compressed essence of the image, are subsequently transmitted to the decoder, a Dense Prediction Transformer (DPT) decoder. The decoder assumes a pivotal role in translating these extracted features into a depth map.

Depth Anything incorporates a teacher-student learning paradigm to leverage the potential of unlabeled data effectively. This approach resembles a scenario wherein a well-trained model (the teacher), possessing expertise in depth estimation from labeled images, guides a student model during its training phase. The student model, however, accesses a more extensive dataset encompassing both labeled and unlabeled images. To harness the value of unlabeled data, the teacher model assumes a pivotal role by generating ”pseudo-labels” for these images. These pseudo-labels essentially encapsulate the teacher’s optimal estimation of the depth information inherent in the unlabeled images. Subsequently, the student model leverages both the original labeled data and these teacher-generated pseudo-labels to augment its learning process. This collaborative methodology empowers the student model to achieve superior performance on unseen images, surpassing the capabilities achievable with labeled data alone.

A key distinguishing feature of Depth Anything resides in its adept utilization of vast quantities of unlabeled data. Typically, this data may not be beneficial for conventional depth estimation models. However, Depth Anything leverages a distinct module called Vit-L (Vision Transformer Large) to address this challenge. Vit-L serves as a pre-training phase specifically tailored for processing unlabeled images. This pre-training stage equips Vit-L to distill meaningful information from the unlabeled data, thereby facilitating the generation of pivotal

pseudo-labels. These pseudo-labels, when combined with the original labeled data, substantially enrich the training process for the student model, ultimately culminating in heightened depth estimation capabilities.

The Depth Anything algorithm is producing an absolute relative error of 0.046 compared to 0.054 of the next best algorithm ZoeDepth. This highlights the superiority of the Depth Anything algorithm for monocular depth estimation.

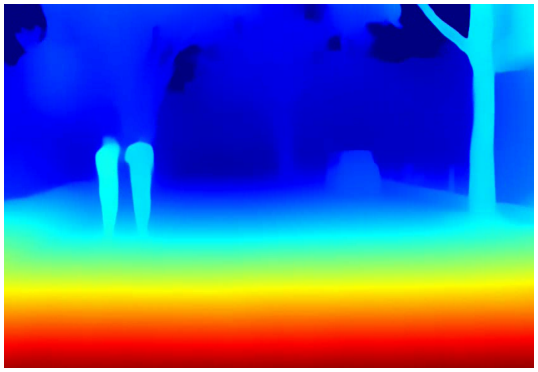
3.2.2 Results and Discussion

Depth Anything algorithm [43] is used for monocular depth estimation in the perception block of the autonomous driving feature. This algorithm is able to provide state-of-the-art output for the estimated depth. However, the drawback of this algorithm is the latency for transferring image and model parameters from the CPU memory to the GPU memory in the Jetson device. Through testing, it was found that this memory transfer comes to around 300 ms which will in turn limit the throughput of the depth estimation algorithm alone to 3 fps. To overcome this issue, the input feature vector size of the model was reduced from 518 x 518 to 200 x 200. Reducing the size resulted in a 10-fold decrease in memory transfer latency, now standing at 30 ms. However, this reduction in latency came with a trade-off, as an increase in depth estimation error was observed as shown in Fig. 3.12 (the higher the red content in the output, the closer the object is to the camera).

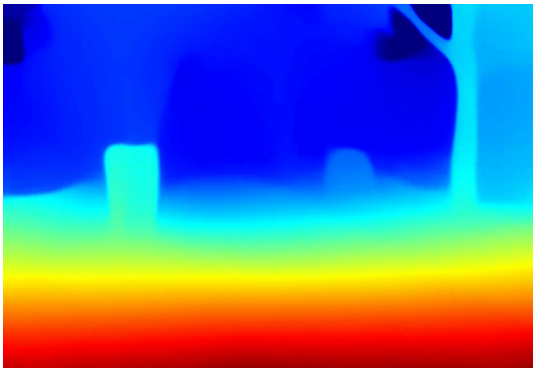
From Fig. 3.12, it can be seen that there is a reduction in the sharpness of the depth information when the input features are resized to 200 x 200. However, despite the resizing, the output is still accurate in providing depth information which is sufficient for the autonomous driving application. Hence, the reduction in sharpness is not considered and resizing to 200 x 200 input feature resizing is done to meet the real-time requirement of the application. Some more outputs of the depth estimation algorithm are provided in Fig. 3.13.

While the Depth Anything model has demonstrated commendable performance in recent outputs, its effectiveness diminishes in dynamic environmental scenarios characterized by fluctuating lighting conditions or the presence of fog or smoke on the road. Under such circumstances, the algorithm struggles to provide accurate depth information. To address these limitations, the integration of supplementary sensors such as Lidar, radar, or ultrasonic becomes imperative. These additional sensors offer enhanced capabilities in capturing depth information, particularly in challenging conditions, thereby augmenting the reliability and robustness of the depth estimation algorithm.

(a) Camera output



(b) Depth without resizing input feature



(c) Depth with resizing input feature

Figure 3.12: Increasing the throughput of Depth Anything model

Figure 3.13: Depth estimation output using Depth Anything model

3.3 Lane Detection

Lane detection is a critical component of autonomous driving systems, relying primarily on camera-based vision techniques to identify the boundaries and markings of lanes on the road. Cameras capture high-resolution images of the driving environment, which are then processed using advanced algorithms to detect lane markings, even in complex and dynamic conditions. This capability is essential for several key functionalities within autonomous vehicles and advanced driver-assistance systems (ADAS).

The importance of lane detection lies in its foundational role in ensuring vehicle safety and navigation accuracy. By accurately identifying lane boundaries, the system can perform lane-keeping, maintain the vehicle within its designated lane, and provide lane departure warnings to prevent unintentional drifts. Additionally, lane detection facilitates lane centering, enabling the vehicle to consistently position itself optimally within the lane.

Applications of lane detection extend to several critical aspects of autonomous driving. One of the primary applications is in adaptive cruise control systems, where the vehicle not only maintains a safe distance from other vehicles but also stays centered in its lane. Furthermore, lane detection aids in identifying vehicles on the same road, supporting collision avoidance systems by monitoring the relative position of other vehicles and ensuring safe maneuvers. Lane detection also underpins the functionality of automated lane-changing systems, allowing the vehicle to safely and efficiently change lanes on highways.

Lane detection from images can be approached through classical computer vision methods and machine learning techniques. Classical computer vision methods typically involve techniques like edge detection, color thresholding, and geometric modeling. One commonly used approach is the application of the Canny edge detector [44] combined with the Hough Transform [45]. This method identifies edges in an image and detects straight lines corresponding to lane markings. Another classical method is color-based segmentation, where specific colors such as white and yellow are isolated and analyzed to detect lane lines. These methods are efficient but may struggle with varying lighting conditions and complex road layouts.

In contrast, machine learning-based approaches leverage data-driven models to learn lane detection directly from images. CNNs are widely used in this context. For instance, a CNN can be trained on labeled datasets to recognize and segment lane markings from raw images, providing robustness to variations in lighting and road conditions. Another ML approach involves using Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks to consider temporal information from a sequence of frames, which enhances the stability and

accuracy of lane detection over time. The primary advantage of ML-based methods over classical computer vision techniques lies in their ability to generalize across diverse and complex scenarios by learning from extensive datasets. This results in improved accuracy and reliability in real-world driving conditions, making ML-based approaches more adaptable and effective for lane detection in autonomous driving systems. Considering these advantages an ML-based approach is selected for lane detection from the images.

Several Convolutional Neural Network (CNN) based models have been developed for lane detection, each leveraging deep learning to achieve high accuracy and robustness. Some of these include Spatial CNN (SCNN) [46] model and LaneNet. SCNN model enhances traditional CNN architectures by adding a layer that can capture spatial relationships between pixels. LaneNet [47] incorporates an instance segmentation approach to detect and segment lane lines from images. It processes the input image through a series of convolutional layers to extract features, and then uses clustering algorithms on the feature maps to identify and segment lane markings. However, a structure-aware deep lane detection algorithm proposed by Zequn et. al [48] is able to produce better results with faster execution time. Hence this method is used for lane detection in this autonomous driving project. The next section discusses in detail about this method for lane detection.

3.3.1 Ultra Fast Structure-Aware Deep Lane Detection

Ultra fast structure-aware deep lane detection algorithm was proposed by Zequn et. al [48] in 2020. In this method, the algorithm selects locations of lanes at predefined rows of the image using global features instead of segmenting every pixel of lanes based on a local receptive field, which significantly reduces the computational cost. Selecting these global features to predict has a larger receptive field than the usual segmentation formulation and thus addresses the no-visual-clue problem also. The overall architecture of this algorithm is shown in Fig. 3.14.

The architecture has a main branch and an auxiliary branch. The main branch is used while training and testing and the auxiliary branch is used only while testing. In the main branch, the input image is first passed through a Resnet block which does the feature extraction. These features are then passed through a group classification block which classifies the rows of the image into lanes if present. The auxiliary segmentation task in the training phase utilizes multi-scale features to model local features. The formulation of the problem by this method is shown in Fig. 3.15.

This method formulates lane detection as a row-based selection method leveraging global im-

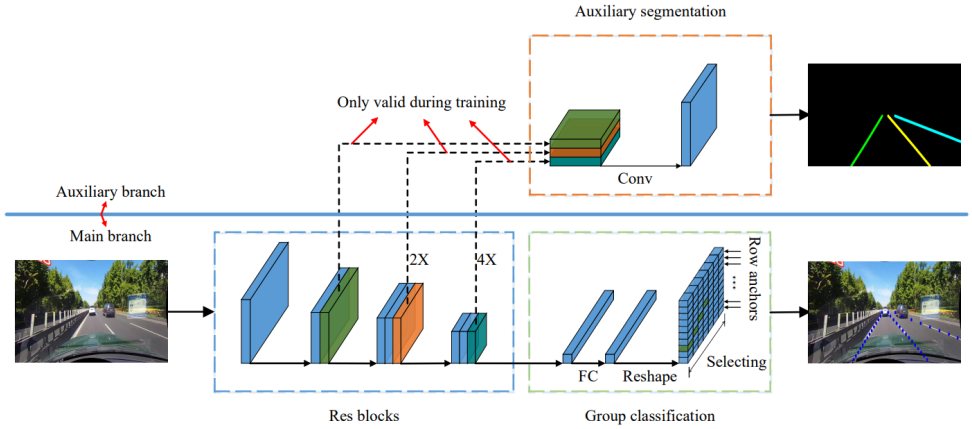


Figure 3.14: Overall architecture of lane detection algorithm

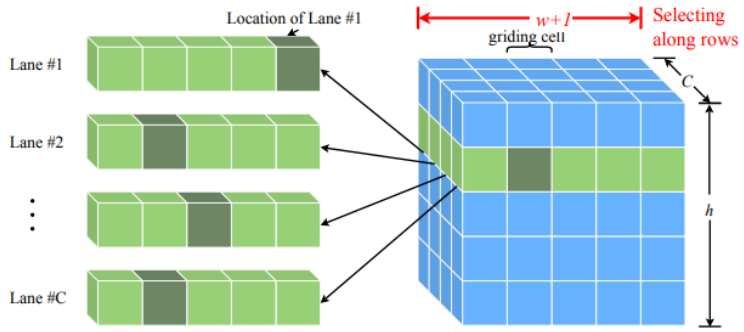


Figure 3.15: Problem formulation of lane detection algorithm

age features. It involves selecting the correct lane positions on each predefined row using these global features. Lanes are represented as a series of horizontal positions at predefined rows, referred to as row anchors. The process begins with gridding, where each row anchor is divided into multiple cells. Lane detection is then framed as selecting specific cells across these row anchors. This method effectively addresses the no-visual-clue problem, where direct visual information is absent, such as when a lane is occluded by a vehicle by utilizing contextual information from other lanes, the overall road shape, and even the direction of vehicles.

From a broader perspective, the formulation's receptive field encompasses the entire image, significantly larger than traditional segmentation methods. This extensive receptive field allows for the incorporation of context and information from other parts of the image to tackle the no-visual-clue issue effectively. Additionally, the approach facilitates learning prior information about lane shapes and directions through structural loss. By modeling lane locations in a row-based manner, it can explicitly establish relationships between different rows, bridging the semantic gap between low-level pixel-wise modeling and the high-level structural represen-

tation of lanes. Thereby this method enhances the accuracy and reliability of lane detection by integrating both local and global contextual information.

The algorithm is able to predict the lanes with an F1 score of 90.7 on the CULane normal dataset, while other algorithms have an F1 score of less than 90. The algorithm is also able to complete the task in 5.7 ms compared to greater than 10 ms of other algorithms when run on Nvidia GTX 1080Ti GPU.

3.3.2 Results and Discussion

The Ultra fast structure-aware deep lane detection algorithm [49] was implemented in Jetson Orin Nano for autonomous driving application. A pretrained model '*tusimple_18.pth*' was used for this purpose. Results obtained for some sample images are shown in Fig. 3.16.

Figure 3.16: Lane detection output for different road conditions

The lane detection output for different road conditions, as illustrated in Fig. 3.16, demonstrates that the algorithm produces satisfactory results across various scenarios. In the first image, the lane is accurately detected under normal conditions. In the second image, despite the lane being occluded by a pedestrian and a bicycle, the algorithm successfully identifies the lane. The third image features a slight left curve in the road, which is also detected accurately by the algorithm. However, in the fourth image, the algorithm fails to detect proper lanes when the vehicle approaches a junction. This failure is attributed to the lack of useful features or lane

markings in the image. Therefore, considering this limitation, the overall algorithm must rely on additional sensors, such as GPS, to navigate junctions and execute vehicle turns effectively.

3.4 Pothole Detection

Pothole detection is a crucial aspect of road safety and vehicle maintenance, particularly in urban environments where road infrastructure is often subject to wear and tear. Potholes, which are depressions or holes in the road surface, pose significant risks to both vehicles and passengers. A study conducted by the Ministry of Road Transport and Highways of India (MORTH) [50] (Fig. 3.17) has shown that nearly 1800 deaths happened in 2022 due to potholes in the Indian roads and potholes account for nearly 1% of deaths on road.

Road feature	Number of accidents			Persons killed			Persons injured		
	2021	2022	%age change	2021	2022	%age change	2021	2022	%age change
Straight road	2,78,218	3,09,247	11.2	1,02,623	1,11,815	9.0	2,59,402	2,97,694	14.8
Curved road	49,581	54,593	10.0	19,120	20,573	8.0	48,888	55,866	14.0
Bridge	12,709	14,111	11.0	5,337	6,258	17.3	11,546	13,062	13.1
Culvert	6,663	7,384	11.0	2,960	3,473	17.0	6,029	6,309	5.0
Potholes	3,625	4,446	22.6	1,481	1,856	25.3	3,103	3,734	20.3
Steep grade	3,967	4,475	13.0	1,635	2,056	26.0	3,398	4,089	20.0
Ongoing road works/ Under construction	9,075	9,211	1.5	4,014	4,054	1.0	7,539	7,955	5.5
Others	48,594	57,845	19.0	16,802	18,406	10.0	44,543	54,657	23.0
Total	**4,12,432**	**4,61,312**	**11.9**	**1,53,972**	**1,68,491**	**9.4**	**3,84,448**	**4,43,366**	**15.3**

Figure 3.17: Statistics of accidents on Indian roads by MORTH

The importance of pothole detection lies in its ability to prevent accidents, vehicle damage, and potential injuries to road users. By identifying potholes in real-time, vehicles can take appropriate actions to avoid or mitigate the impact of these road hazards. Applications of pothole detection systems include overtaking the pothole by steering away from it, slowing down the vehicle to reduce the impact of driving over the pothole, or even stopping the vehicle if necessary to avoid collision or damage. Additionally, pothole detection systems contribute to road maintenance efforts by providing valuable data on the location and severity of potholes. This information can be used by road authorities to prioritize repairs and allocate resources efficiently, thereby improving overall road quality and safety. Hence pothole detection is an important add-on feature for autonomous driving cars in the Indian roads.

Pothole detection from images can be approached using classical computer vision techniques as well as machine learning based methods. Classical computer vision techniques typically involve edge detection, texture analysis, and template matching to identify potholes. For example, edge detection algorithms such as Canny edge detection can be used to detect abrupt changes in intensity, which may indicate the presence of a pothole. Texture analysis methods analyze local patterns and variations in pixel intensity to identify irregularities characteristic of potholes. Another classical approach is template matching, where predefined templates of pothole shapes are compared with regions of the image to identify matches. These methods are computationally efficient but may struggle with variations in lighting, shadows, and road surface textures.

In contrast, machine learning-based approaches leverage data-driven models to learn and detect potholes directly from images. For instance, a CNN can be trained on labeled datasets to recognize and classify potholes based on their visual features. Another ML approach involves the use of image segmentation techniques to segment potholes from the surrounding road surface. This may include semantic segmentation, where each pixel in the image is classified as belonging to either the pothole or background class, or instance segmentation, where individual potholes are segmented as distinct objects. These ML-based methods offer improved accuracy and robustness compared to classical techniques, as they can learn complex patterns and features from large datasets, making them well-suited for real-world pothole detection applications.

In this project, the pothole detection from images is done based on Darknet and YOLO algorithms similar to those used for object detection. The next subsection discusses the architecture for pothole detection.

3.4.1 Pothole Detection Architecture

The overall architecture has 13 convolutional layers, 6 route layers, 1 maxpool layer, an upsample layer, and 2 YOLOv4 layers [51]. So in total, the architecture has 23 layers. Out of all the layers, the YOLOv4 layer is responsible for detecting the shape of the pothole and plays a major role in the architecture. The architecture of YOLOv4 is shown in Fig. 3.18.

The architecture of YOLOv4 integrates the robust CSPDarkNet-53 backbone to extract intricate features from input images. This backbone comprises five residual block modules, enabling efficient feature extraction. These features are then fused at the network's neck. Here, an SPP module enhances feature representation by concatenating max-pooling outputs from low-resolution feature maps, utilizing kernels of varying sizes with a stride value of 1. This

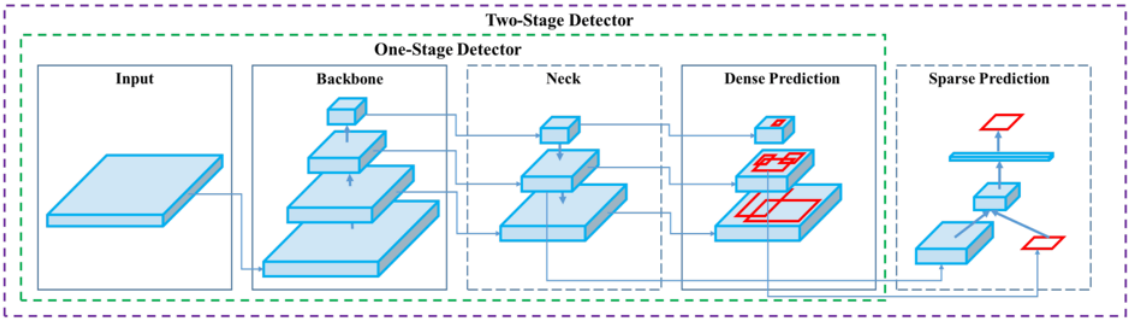


Figure 3.18: YOLOv4 Architecture

fusion strategy significantly augments the network's accuracy, particularly in detecting smaller objects, by expanding the receptive field of backbone features.

Subsequently, the PAN module plays a pivotal role in amalgamating concatenated feature maps from the SPP module with high-resolution feature maps. Employing a combination of upsampling and downsampling operations, PAN establishes bottom-up and top-down pathways to integrate low-level and high-level features effectively. This strategic fusion culminates in the generation of aggregated feature maps, crucial for subsequent predictions.

Finally, YOLOv4 boasts three detection heads, each akin to a YOLOv3 network, responsible for computing final predictions. These detection heads leverage feature maps of various sizes to predict bounding boxes, classification scores, and objectness scores, ensuring comprehensive object detection capabilities across different scales.

For training the model, a mix of datasets is used. The first dataset is provided by Roboflow [52] and has 465 training images. The second dataset is based on a research article by Nienaber et.al [53] published in 2015. Combining these two datasets forms 1265 training images and 401 validation images. Even though there are sufficient images for training, these images are captured from US and European road conditions. So they are not well suited for detecting potholes that are in Indian roads. So the above dataset was augmented with pothole images from Indian roads. This includes 400 pothole images captured from IISc, Mathikere, and Indian pothole images obtained from the internet, making it a total of 1665 images for training. Some of the pothole images captured from IISc for data augmentation are shown in Fig. 3.19.

Training of the model was done for 5000 iterations and the loss has converged to 0.9 as shown in Fig. 3.20. This trained model could produce a test mAP of 40%.

Figure 3.19: Sample pothole images captured from IISc for data augmentation

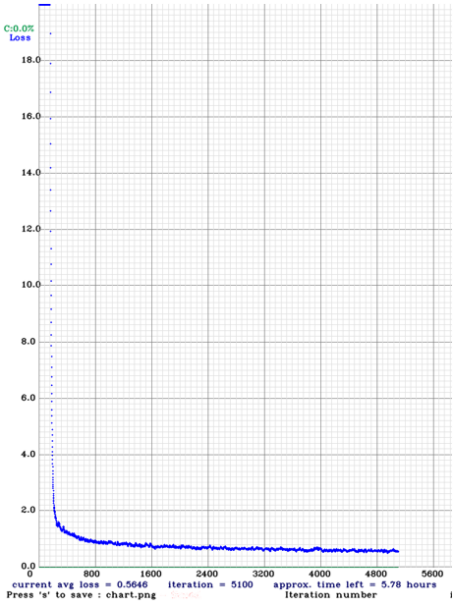


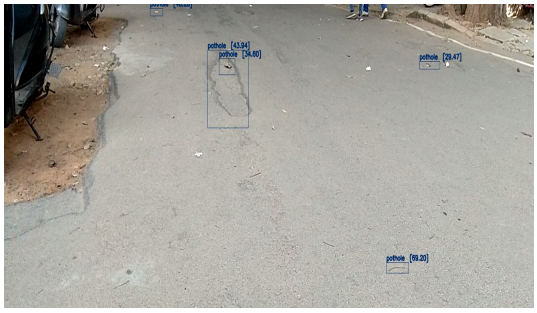
Figure 3.20: Loss plot of the model during training

3.4.2 Results and Discussion

As discussed in the previous section, to increase the accuracy in detecting potholes on the Indian roads, the training dataset was augmented with pothole images captured on the Indian roads. Fig. 3.21a shows a sample output of detected potholes in the Indian road without data augmentation. From the figure, it can be seen that there are some false negatives in the classification. To overcome that, data augmentation was done to produce the output shown in Fig. 3.21b. But it can be seen that there are some false positives in the image, especially when leaves or shadows are present on the road. To improve the classification, postprocessing is done on the model output which looks for the pothole size for proper classification and thereby ignores the small bounding boxes that might be attributed to leaves on the road Fig. 3.21c.

The trained model with postprocessing was used to detect potholes in the captured video frame. Results obtained for some sample images are shown in Fig. 3.22.

(a) Without data augmentation



(b) With data augmentation

(c) With post processing

Figure 3.21: Pothole detection with different techniques

These images were captured on the IISc campus and demonstrated satisfactory detection of potholes. However, it should be noted that some potholes are missed in the fourth image, and misclassifications occur when shadows are present. These issues can be addressed by increasing the size of the training dataset and implementing post-processing on the model output. Currently, the algorithm is designed such that whenever a pothole is detected the vehicle is slowed down to reduce its impact on the passenger and the vehicle. In the future, a more detailed classification of pothole severity could be developed to enable appropriate actions such as slowing down, stopping, or turning the vehicle.

This chapter, explored the perception capabilities of autonomous driving cars using cameras. Various models were implemented to achieve key perception tasks: object detection using YOLOv5, depth estimation using the Depth Anything model, lane detection using the Ultra Fast Structure-aware Deep Lane Detection model, and pothole detection using YOLOv4. The results from these implementations were satisfactory, demonstrating the effectiveness of these models in enhancing the perception system of autonomous vehicles.

However, there are areas for further improvement. Future work can focus on incorporating additional sensors, such as Lidar and radar, to complement the camera-based perception system. These sensors can provide more accurate depth information, better object detection in adverse

Figure 3.22: Pothole detection output of sample images

weather conditions, and improved robustness against occlusions and lighting variations. Integrating these sensor modalities will enhance the overall reliability and safety of the autonomous driving system. Additionally, expanding the training datasets and refining the models through post-processing techniques will further improve detection accuracy and performance.

In this chapter, we discussed different perception algorithms in detail. The object detection algorithm enables the vehicle to identify and classify objects like pedestrians, vehicles, and traffic signs. The depth estimation algorithm provides accurate distance information about objects in the environment, which is essential for safe navigation. The lane detection algorithm ensures the car remains within its designated lane by identifying lane markings on the road. Finally, the pothole detection algorithm detects potholes on the road which can be used for taking intelligent actions to prevent potential damage to the vehicle and maintain a smooth ride for passengers. We have also looked at the output obtained for each of these algorithms.

The next chapter discusses localization, path planning, reference generation, and control law implementation of the autonomous driving car.

Chapter 4

Route Planning and Control Algorithms

The previous chapter discussed different algorithms for perception in an autonomous driving vehicle in detail. In this chapter, a detailed study and design of localization, path planning, reference generation, and control law implementation are discussed.

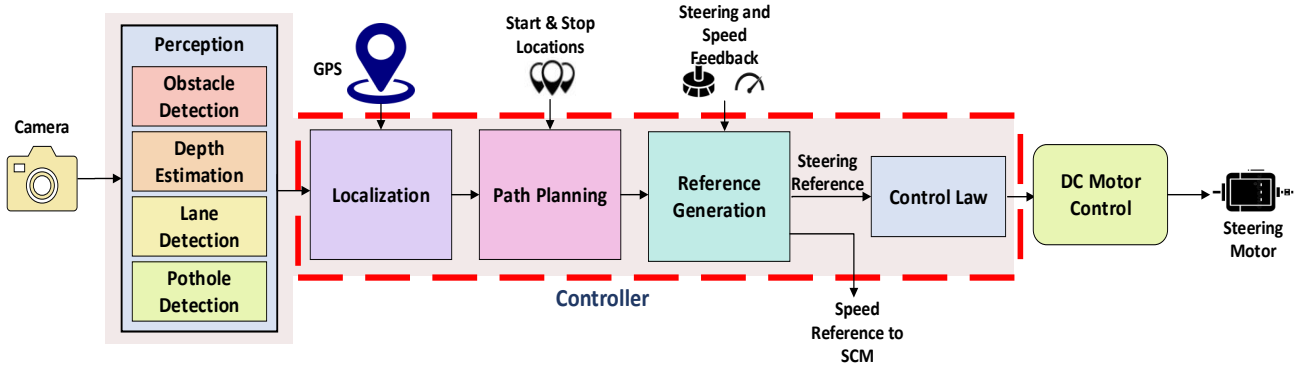


Figure 4.1: Block diagram of the IMM

Each section of this chapter will explore an element of planning and control in detail. The first section will examine localization, which enables the car to determine the vehicle's precise position in the world. This element integrates data from the Global Positioning System (GPS) and the vehicle's perception system to achieve accurate positioning. The second section will focus on path planning, involving the determination of the route from the vehicle's current position to its desired destination. In the third section, reference generation is discussed which translates these plans into actionable commands that the vehicle can execute. These references include speed and steering information for the car. In the last section, control law implementation, operating in a negative feedback loop, and ensuring precise and responsive vehicle steering are detailed.

By exploring localization, path planning, reference generation, and control law implementation in the car, this chapter aims to provide a comprehensive understanding of how these critical components, combined with powerful processing techniques, enable autonomous vehicles to precisely determine their position, plan routes, translate plans into actionable commands, and ensure responsive and precise vehicle control. Together, these modules empower autonomous vehicles to navigate their surroundings safely and efficiently, enhancing overall road safety and driving experience.

4.1 Localization

Localization is a fundamental process that determines the vehicle's precise position in the world. Accurate localization ensures safe and efficient operation by providing the vehicle with real-time awareness of its surroundings, facilitating reliable decision-making and trajectory planning. In the literature, there are several sensors that can be used for localization of an autonomous driving vehicle [54]. These sensors include the Global Positioning System (GPS) or Inertial Measurement Unit (IMU), camera, radar, Lidar, ultrasonic, etc. GPS provides global positioning data, but signals may degrade in urban environments. IMU measures acceleration and angular velocity, aiding orientation estimation, yet suffers from drift. Cameras capture rich visual information, aiding in landmark recognition and environment understanding. However, they are sensitive to lighting conditions and may lack depth perception, requiring robust processing algorithms for accurate localization. Radar excels in adverse weather conditions and offers reliable long-range sensing capabilities. Nevertheless, it has limited angular resolution and may struggle with precise object localization. Lidar provides precise distance measurements and accurate object localization, especially in urban environments with complex structures. Despite its accuracy, Lidar systems can be costly and susceptible to interference from ambient light and adverse weather.

To start with and to make the system simple, this project implements localization using information from GPS and the vehicle's perception system to achieve accurate positioning. While the GPS provides global positioning data, offering a rough estimate of the vehicle's location, the perception system incorporates information about surrounding features like buildings or potholes. By fusing these data sources, the localization module ensures a highly accurate understanding of the vehicle's position, which is crucial for safe and efficient navigation.

4.1.1 GPS Interface

As mentioned earlier, GPS provides global positioning data, offering a rough estimate of the vehicle's location. The GPS receiver in an autonomous vehicle communicates with a constellation of orbiting satellites, each broadcasting precise timing information along with its own position as shown in Fig. 4.2 [55]. By receiving signals from multiple satellites simultaneously, the GPS receiver calculates its distance from each satellite based on the time it takes for the signals to reach it. Through a process called trilateration, the receiver then determines its position by intersecting these distance measurements. This data includes latitude, longitude, and altitude, forming a three-dimensional coordinate system that precisely defines the vehicle's location on Earth's surface.

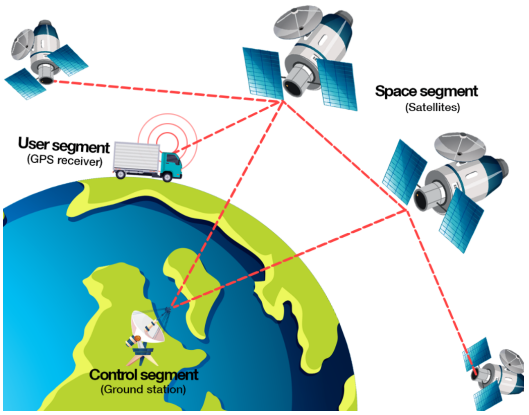


Figure 4.2: Working of GPS

There are several GPS modules available in the market like NEO-6M, Raspberry Pi GPS Module, Adafruit Ultimate GPS module, etc. Among them, the product by Adafruit is widely used for autonomous driving applications because of its accuracy in the location information. Technical details of the product are tabulated in table 4.1 [25].

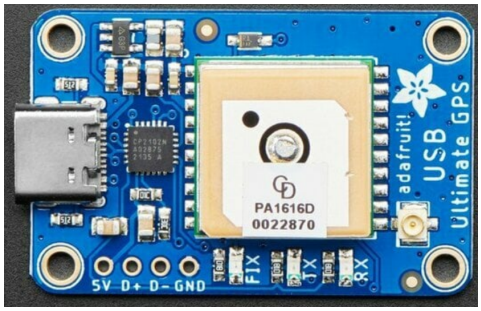


Figure 4.3: Adafruit Ultimate GPS Breakout v3

Parameter	Unit	Value
Dimension	mm	15 x 15 x 4
Update rate	Hz	1-10
Position accuracy	m	1.8
Acquisition sensitivity	dBm	-145
Connector type		USB

Table 4.1: Technical details of the GPS module

The GPS module is equipped with an RTC battery of CR1220 size and includes the option to attach an external antenna. Connecting to the Jetson board is facilitated through a USB interface. To retrieve data from the module, the Python environment is configured with the Adafruit_Blinka library.

The module's latitude and longitude yield the current vehicle location. Complementing this, data from the pothole detection and obstacle detection modules contribute to understanding the surrounding environment, enhancing localization capabilities.

4.2 Path Planning

Following localization, path planning emerges as a critical task, entailing the identification of the most efficient route from the vehicle's present location to its intended destination. This pivotal process initiates with the creation of a route derived from the starting and destination points. Path planning holds paramount importance in autonomous driving systems as it ensures efficient navigation, minimizes travel time, optimizes fuel consumption, and enhances overall road safety. Various path planning algorithms, such as the Hybrid-State A* Search [56], Dijkstra's Algorithm, Rapidly-exploring Random Tree [57], and Probabilistic Roadmap (PRM) [58] offer distinct methodologies for determining optimal routes in autonomous driving scenarios. Each of these algorithms has its own strengths and weaknesses, making them suitable for different types of environments and scenarios encountered in autonomous driving. However, implementing these algorithms requires a considerable amount of time in mapping the area, finding appropriate routes, etc. Hence, to make the system simple, instead of computing the path in the onboard electronics in real-time, the path information based on a Start and Destination location is saved as a csv file in the system and used for testing. This path information contains the latitude and the longitude of the predefined path obtained from Google Maps.

The location data obtained from maps is visualized on a 2D plane, as depicted in Fig. 4.4. The path planning and reference generation modules collaborate to guide the vehicle along the lane detected by the perception module. However, there are scenarios where the lane detection algorithm fails, such as when the vehicle approaches a junction or needs to make a right/left turn. In such cases, the localization module becomes crucial. It provides real-time location information of the vehicle. Consequently, if the path requires a turn in the upcoming instance, the steering reference is determined based on this real-time location data.

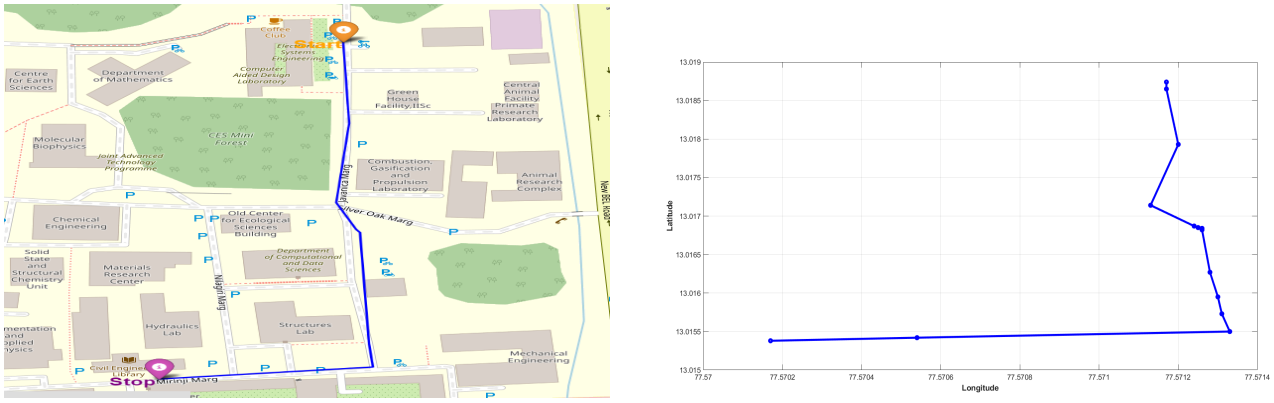


Figure 4.4: Path obtained from Google Maps and its projection onto a 2D plane

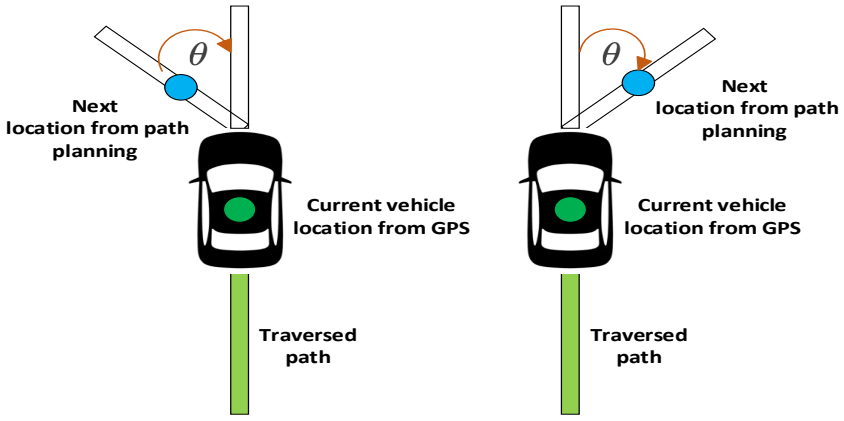


Figure 4.5: Vehicle turning scenarios

Fig. 4.5 depicts 2 such scenarios. In the first scenario, the next location for the vehicle obtained from the stored path is towards the left of the current location at an angle of θ. Similarly, in the second scenario, it is at an angle of θ to the right. In both scenarios, the steering angle of the vehicle has to be adjusted by θ degrees to the left or right to follow the pre-stored path. In this way, the vehicle will be able to make the required turns as demanded by the stored path. This ensures the vehicle remains on the correct path and can navigate complex routes with necessary turns, thus enhancing the overall reliability and accuracy of the autonomous navigation system.

4.3 Reference Generation

Once the path is planned, the reference generation module translates this data into actionable commands that the vehicle can execute. The steering reference module calculates the precise steering angles required to follow the planned path, ensuring the vehicle remains on course. Concurrently, the speed reference module determines the optimal speed, taking into account the path plan and current traffic conditions, to generate speed commands that maintain a safe and efficient pace. In this section, reference generation for scenarios involving obstacles on the road and lane-following will be discussed in detail.

4.3.1 Handling Obstacles

For handling obstacles, only the obstacles that are in the same lane as the ego vehicle are considered. In Fig. 4.6, only the primary obstacle is considered for making the decisions, and secondary obstacles are only considered for safety aspects.

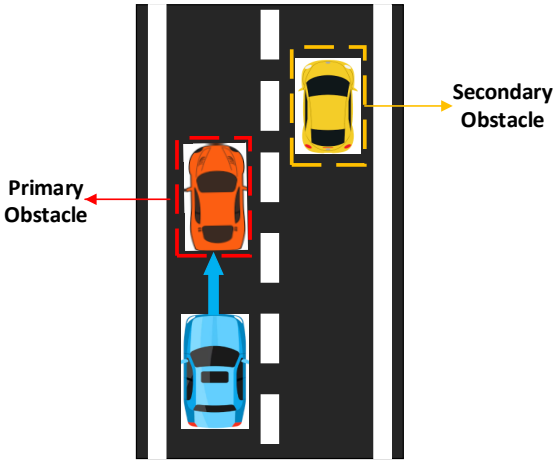
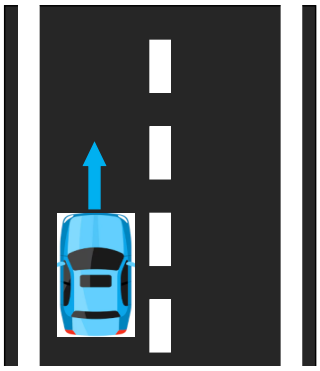


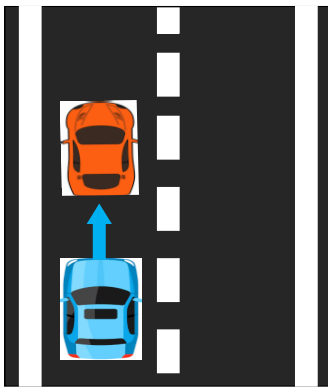
Figure 4.6: Primary and secondary obstacles

Six test cases are considered for handling obstacles, with some cases including subcases. These cases are shown in Fig. 4.13. In these images, the ego vehicle (autonomous vehicle) is shown in blue colour, an obstacle in the same lane in orange, and the obstacle in the different lanes in yellow. The first scenario considers a situation where there are no obstacles on the road. The second scenario involves an obstacle in front of the vehicle, with three subcases based on different vehicle speeds. The third scenario addresses an obstacle that suddenly appears very close to the vehicle. The fourth scenario examines situations where another obstacle is present in the

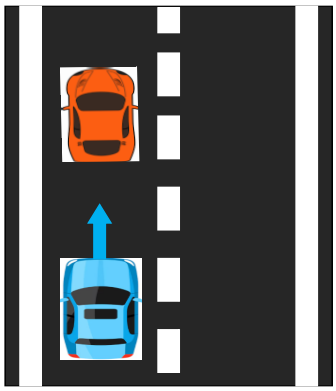
overtaking lane while the primary obstacle is being overtaken, with four subcases depending on the distance and speed of the other obstacle. The fifth scenario involves a moving obstacle, considering subcases where the relative velocity of the ego vehicle is either negative or positive. Finally, the sixth scenario analyzes a single-lane situation where an obstacle approaches the ego vehicle without posing a collision risk. To make the system simple, a condition where both the primary and secondary obstacles are moving is not considered for the analyses.



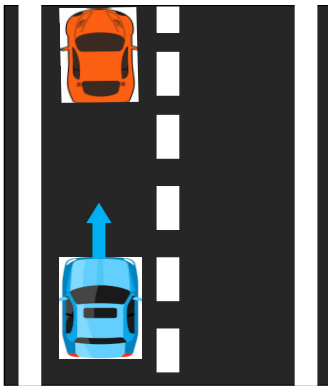
Case 1: No obstacles



Case 2a: Moving at 10 kmph

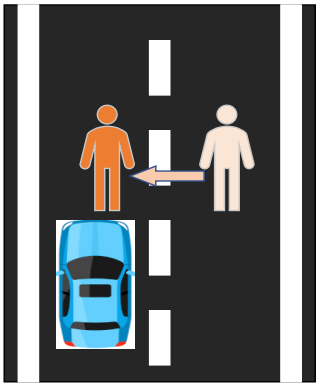


Case 2b: Moving at 20 kmph

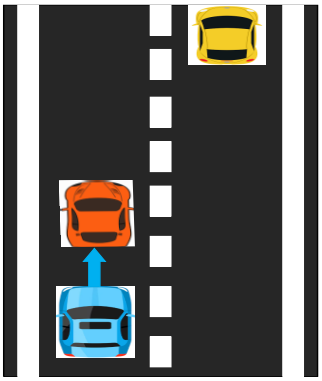


Case 2c: Moving at 30 kmph

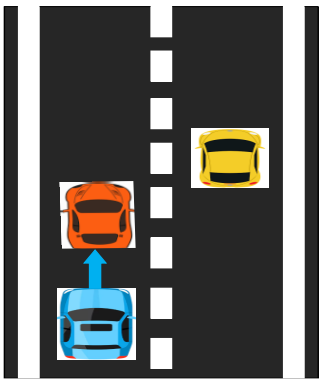
The actions expected by the autonomous driving car in each of these cases are different. In the first case, where there are no obstacles, the car is expected to follow the lane at the same speed. In the second case involving a stationary primary obstacle and the ego vehicle moving at various speeds, the action expected from the vehicle is the same, i.e. overtake. However, since the ego vehicle speeds are different, the distance of the obstacle at which the ego vehicle initiates an overtake varies. When the car is moving at 30 km/h, the overtake action has to be taken much ahead of than when the car is moving at 10 km/h. In the third case where the obstacle is too close, then an emergency Stop action has to be initiated. The deceleration rate of the vehicle in this case has to be decided based on the comfort of the passenger and the stopping distance



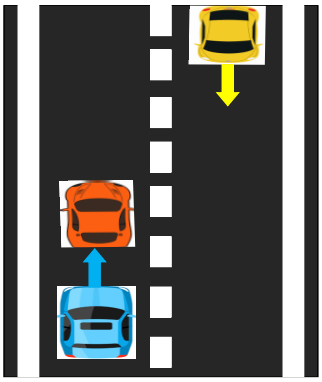
Case 3: Obstacle too close



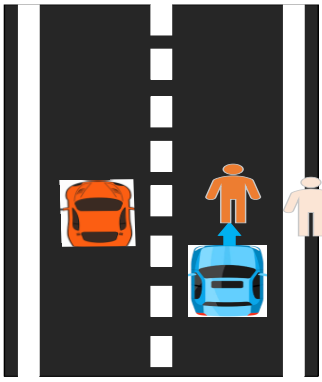
Case 4a: Secondary obstacle too far



Case 4b: Secondary obstacle nearby

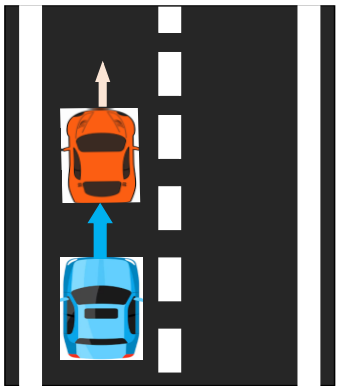


Case 4c: Secondary obstacle far away, but moving

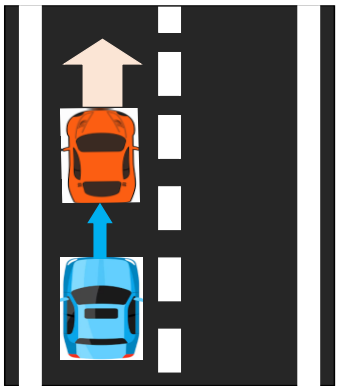


Case 4d: Secondary obstacle closeby

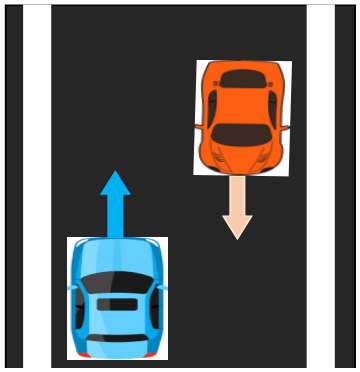
of the vehicle. The fourth case analyzes different conditions of a secondary obstacle while overtaking a primary obstacle. If the secondary obstacle is too far, then the overtake action can be taken similar to case 2. However, if the secondary obstacle is nearby or is moving closer to the ego vehicle, then overtake action shouldn't be taken to avoid a collision with the secondary obstacle, instead the ego vehicle should be stopped in front of the primary obstacle with a



Case 5a: Obstacle moving slower



Case 5b: Obstacle moving faster



Case 6: Obstacle approaching in single lane road

Figure 4.13: Test cases considered for handling obstacles

suitable deceleration rate. In the fifth case where the primary obstacle is moving, the relative speed of the primary obstacle can be positive or negative than the ego vehicle. If the speed is negative, this implies that the ego vehicle is faster than the obstacle and the overtake action can be initiated. In the other condition where the relative speed is negative, the ego vehicle is slower than the obstacle, and hence an overtake action cannot be taken. In the final case where the road is single lane, we have to take appropriate action when an obstacle is approaching in the opposite direction. In this condition, if the obstacle is coming through the right side of the road, then the ego vehicle can move without any change in its speed or direction.

For each of these test cases, the parameters that should be monitored to take the action are tabulated in table 4.2.

From table 4.2, it can be seen that 5 parameters are needed to take an optimum action to handle obstacles. Out of these 5 parameters, the ego vehicle speed and steering angles are obtained directly from a Hall sensor and a potentiometer respectively. The obstacle depth is estimated

Scenario	Ego vehicle speed	Ego vehicle steering	Obstacle depth	Obstacle relative speed	Obstacle position on lane
Case 1	✓	✓			
Case 2	✓	✓	✓		
Case 3	✓	✓	✓		
Case 4a	✓	✓	✓		
Case 4b	✓	✓	✓		
Case 4c	✓	✓	✓	✓	
Case 4b	✓	✓	✓		
Case 5a	✓	✓	✓	✓	
Case 5b	✓	✓	✓	✓	
Case 6	✓	✓			✓

Table 4.2: Parameters to be monitored for handling obstacles

using the Depth Anything model discussed in the third chapter. Obstacle relative speed is derived from the depth information as shown in equation 4.11. Here, the consecutive depth information and the processing time represented by frames per second (*fps*) are used to derive the relative speed of the obstacle. Obstacle position on the road is required to handle single-lane scenarios. In this case, if the obstacle is found to be at a location more than 50% to the right of the detected lane, then the obstacle is considered to be on the right side of the road and no change in action will be taken for it.

$$ObstacleRelativeSpeed = (depth[t] - depth[t+1])fps \tag{4.1}$$

The algorithm for handling obstacles is developed based on these six scenarios and incorporates these 5 parameters. The flowchart for handling obstacles is shown in Fig. 4.14.

When an obstacle is detected, the algorithm first checks if it is a primary obstacle, meaning it is in the same lane and poses the most immediate threat. If the obstacle is not in the same lane, the algorithm then determines if it is on the right side of the road. If it is, it does not pose a threat, and no action change is needed, allowing the vehicle to maintain its current speed and steering angle. However, if the obstacle is not primary or if it is on the left side of the road, additional information is needed to make a decision.

Next, the algorithm assesses the relative speed of the obstacle. If the relative speed is positive, indicating that the obstacle is moving faster than the ego vehicle, it does not pose a safety threat.

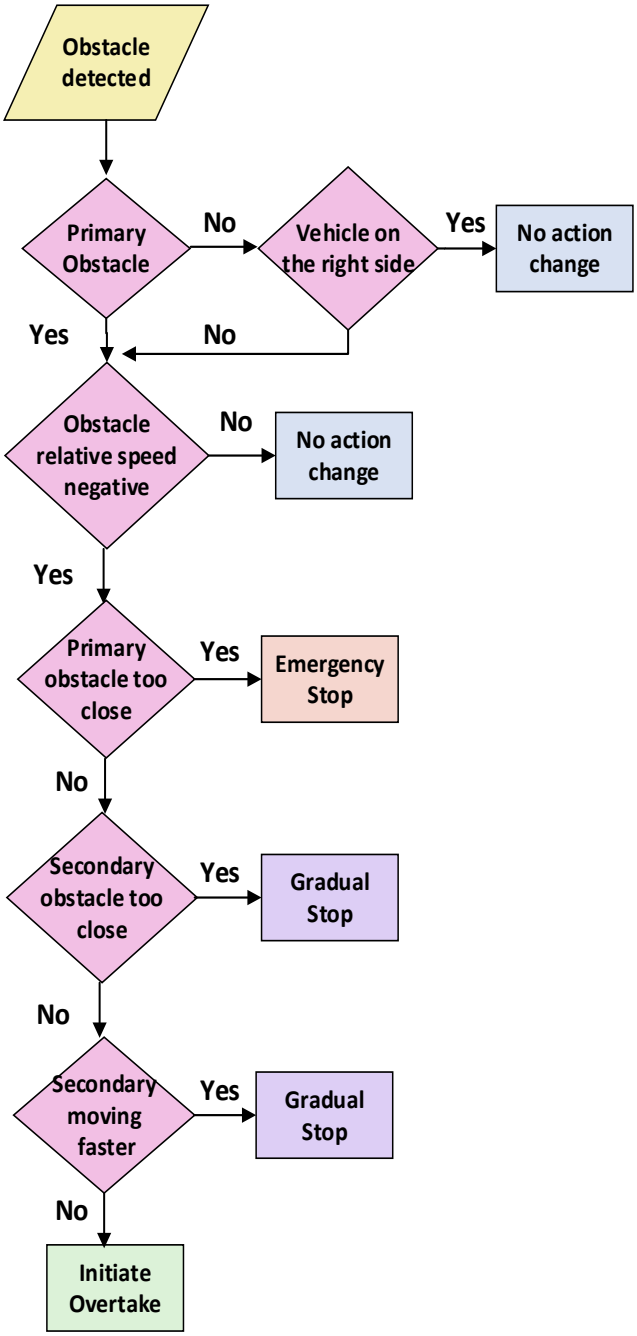


Figure 4.14: Flow chart to handle obstacles

Conversely, if the relative speed is negative, further considerations are required. If the obstacle is too close, an overtaking maneuver could lead to a collision, so the ego vehicle must gradually slow down to avoid a crash with the primary obstacle. On the other hand, if the obstacle is at a sufficient distance and any secondary obstacles are either too far or not moving toward the ego vehicle at a higher speed, the ego vehicle can initiate an overtaking action. If these conditions

are not met, the vehicle must gradually stop to prevent a collision.

There are different parameters used for achieving this algorithm. These parameters are discussed in the next subsection. And details about the three subprocesses - Gradual Stop, Emergency Stop, and Initiate Overtake are discussed in the subsequent subsections.

4.3.1.1 Algorithm Parameters

In the overtake algorithm, there are two types of parameters, viz., decision-making parameters and action parameters. Decision-making parameters are obtained directly from the sensor or derived from the sensor/ perception. These parameters are used to make optimum decisions for initiating actions. Here depth threshold is a dependable parameter that changes based on the other decision-making parameters. Action parameters are the outputs of the reference generation module for controlling the speed and steering of the vehicle. Decision-making parameters and their short forms are tabulated in table 4.3 and Action parameters are tabulated in 4.4.

Parameter	Short form
Ego Vehicle Current Speed	ev_cs
Primary Obstacle Relative Speed	po_rs
Primary Obstacle Depth	po_dp
Primary Obstacle Depth Threshold	po_dt
Secondary Obstacle Depth	so_dp
Secondary Obstacle Relative Speed	so_rs
Secondary Obstacle Depth Threshold	so_dt
Obstacle Safe Distance	ob_sd

Table 4.3: Decision-making parameters for handling obstacles

In the action parameters, *ev_ot* represents the time it takes to overtake an obstacle, and *ev_tt* represents the time taken to execute a left or right turn in the lane. These parameters depend on the current speed of the ego vehicle. Time to gradual stop and the time to emergency stop indicate the time taken for the vehicle to achieve a gradual stop and emergency stop respectively. Next, we discuss how the action parameters and the depth threshold are updated based on the decision-making parameters. All the equations used for determining these parameters in this

Parameter	Short form
Ego Vehicle Reference Speed	ev_rs
Ego Vehicle Reference Steering Angle	ev_ra
Ego Vehicle Overtake Time	ev_ot
Ego Vehicle Turn Time	ev_tt
Ego Vehicle Time to Gradual Stop	ev_tg
Ego Vehicle Time to Emergency Stop	ev_te

Table 4.4: Action parameters for handling obstacles

section are obtained from the fundamental three equations of motion as shown below.

$$v = u + at$$
$$s = ut + \frac{1}{2}at^2$$
$$v^2 = u^2 + 2as$$

At first, the acceleration that is possible for the vehicle needs to be found out. This information is needed to find out the limit of the vehicle and make decisions that operate safely in this region. The notations used for this analysis are tabulated in 4.5.

Aero force and rolling force experienced by the vehicle are shown in the equations 4.2 and 4.3 respectively.

$$F_{\text{aero}} = \frac{1}{2}\rho A_{\text{f}} C_{\text{d}} v^2 \tag{4.2}$$

$$F_{\text{r}} = C_{\text{r}} M g \tag{4.3}$$

Using this the actual acceleration force on the vehicle is

$$F_{\text{a}} = \frac{T_{\text{M}} N}{R} - F_{\text{aero}} - F_{\text{r}} - F_{\text{g}} \tag{4.4}$$

Now considering the capability of the vehicle only to operate on flat roads, i.e. without any gradient, the gradient force can be neglected in equation 4.4. Substituting the respective values, the actual acceleration force experienced by the vehicle is 2296 N. Using this the actual vehicle acceleration is obtained as 4.4 m/s^2.

Parameter	Notation	Value	Unit
Motor torque	T_M	90	Nm
Gear ratio	N	6	
Wheel radius	R	0.2	m
Air density	ρ	1.225	kg/m^3
Frontal area	A_f	1	m^2
Drag coefficient	C_d	0.3	
Friction coefficient	C_r	0.02	
Vehicle mass	M	520	kg
Gravity	g	9.8	m/s^2
Vehicle speed	v		km/h
Actual acceleration force	F_a		N
Actual acceleration	a		m/s^2
Gradual acceleration	a_g		m/s^2
Emergency acceleration	a_e		m/s^2
Aero force	F_{aero}		N
Rolling force	F_r		N
Gradient force	F_g		N

Table 4.5: Notation of used parameters

The value of acceleration obtained by equation 4.4 is the maximum possible acceleration that can be achieved by the vehicle. However, executing this acceleration rate would affect passenger comfort as well as will have safety concerns. Hence, while considering acceleration/ deceleration of the vehicle, a value that is lower than this limit and comfortable for the passenger is considered. Assuming the acceleration to bring a vehicle from rest to a speed of 30 km/h in 2.7 seconds would be comfortable for the passenger, this acceleration/ deceleration rate is used for all actions where the vehicle is brought to rest or when the speed is modified, except for emergency stop. This acceleration value turns out to be **3 m/s^2**. However, while considering an emergency stop action, the safety of the passenger, ego vehicle, and the obstacle has to be given importance over the passenger's comfort. In this scenario, the vehicle deceleration is fixed to happen at a higher rate of **4 m/s^2** (decelerating the vehicle from 30 km/h to rest in 2.1 seconds).

Now, the dependent decision-making parameter, Primary Obstacle Depth Threshold (*po_dt*) depends on the ego vehicle velocity. It is the depth of the obstacle at which the overtake or

gradual stop action will be initiated. If the vehicle velocity is higher, then overtake action needs to be taken much ahead of than when the vehicle is moving slowly. Now, considering a safe stoppage distance (distance away at which the ego vehicle stops in front of the obstacle) of 1 m, the depth threshold is given by equation 4.5.

$$po\_dt = \frac{-v^2}{-2a_{\mathrm{g}}} + 1 \tag{4.5}$$

In equation 4.5, a_g = -3 m/s^2. This value of the Primary Obstacle Depth Threshold is used to check the closeness of the obstacle with the vehicle and so used for initiating an overtake or gradual stop action.

The second dependent decision-making parameter is the Secondary Obstacle Depth Threshold (*so_dt*). This parameter indicates the threshold depth of the secondary obstacle below which the overtake action won't be initiated. So if the depth of the secondary obstacle is below the secondary obstacle depth threshold, then the ego vehicle will be gradually stopped instead of overtaking. The secondary obstacle depth threshold is a safety parameter crucial while overtaking the vehicle. This parameter has to be determined by considering two factors, the primary obstacle depth threshold and the relative velocity of the secondary obstacle. If the secondary obstacle is moving, then the *so_dt* has to be higher than when the obstacle is at rest. Also for safety, a margin of 10 m is put above these two factors. It means that if the stationary secondary obstacle is at a distance less than 10 m from the primary obstacle, then the overtake action won't be initiated and instead the ego vehicle will be gradually stopped. The margin of 10 m is arrived at based on the consideration that even a vehicle moving at 30 km/h would be able to overtake the primary obstacle with a sufficient distance from the secondary obstacle.

$$so\_dt = po\_dt + (so\_rs - v)ev\_ot + 10 \tag{4.6}$$

In equation 4.6, the first term represents the primary obstacle depth threshold and the second term indicates the distance traveled by the secondary obstacle in the time when the primary obstacle is overtaken by the ego vehicle.

Next, the safe distance of an obstacle in case of an emergency stop needs to be identified to define the safe operating region of the vehicle. The vehicle is guaranteed to operate safely if the obstacle is beyond this distance from the vehicle. On the other hand, if an obstacle appears with a depth shorter than this distance, then the passenger has to take a manual action (sudden brake) to stop the vehicle. This obstacle-safe distance (*ob_sd*) is given by equation 4.7.

$$ob\_sd = \frac{-v^2}{-2a_e} + 1 \tag{4.7}$$

Now all the decision-making parameters are obtained. Next, we discuss how different action parameter values are arrived at. The Ego Vehicle Time to Gradual Stop (*ev_tg*) indicates the time taken for the vehicle to gradually stop once the gradual stop action is initiated. This parameter depends on the fixed gradual deceleration of the vehicle i.e. -3 m/s^2 and the current vehicle speed equation 4.8. Similarly, the Ego Vehicle Time to Emergency Stop (*ev_te*) indicates the time taken for the vehicle to emergency stop once the emergency stop action is initiated. This parameter depends on the fixed emergency deceleration of the vehicle i.e. -4 m/s^2 and the current vehicle speed given by equation 4.9.

$$ev\_tg = \frac{-v}{-a_g} \tag{4.8}$$

$$ev\_te = \frac{-v}{-a_e} \tag{4.9}$$

Similar to the time to stop, Ego Vehicle Overtake Time (*ev_ot*) is also important to decide the instance where the vehicle has to return to the previous lane after overtaking. While overtaking, the vehicle has to take a lane change to the right and travel *po_dt* distance for a duration of *ev_ot* at the same speed before returning to the same lane. This time depends on the distance of the primary obstacle and the relative speed of the ego vehicle with the obstacle. In addition to the distance to the primary obstacle, a margin of 2 m is given in the distance for safety. Ego Vehicle Overtake Time is given by the equation 4.10.

$$ev\_ot = \frac{po\_dt + 2}{po\_rs} \tag{4.10}$$

The Ego Vehicle Reference Speed (*ev_rs*) is adjusted when the vehicle is gradually or emergency stopped, or when transitioning from a stationary to a moving state. However, the speed remains unchanged during obstacle overtaking or when navigating turns.

In the first condition of gradual stop, the vehicle deceleration is fixed at -3 m/s^2. So in *ev_tg* time, the vehicle has to be brought to rest by modifying the reference speed. The speed reference to the vehicle is updated in every iteration in the outer guidance loop. So the speed reference generated in every iteration till the vehicle achieves rest is given by equation 4.11.

$$ev\_rs[t] = v[t-1] - \frac{a_g}{fps} \tag{4.11}$$

The emergency stop is similar to the condition of a gradual stop, with the only difference in the deceleration rate a_e given by equation 4.12.

$$ev\_rs[t] = v[t-1] - \frac{a_{\mathrm{e}}}{fps} \tag{4.12}$$

In the third condition of transitioning from a stationary to a moving state, it is assumed that the final speed needed to be achieved for the ego vehicle is 30 km/h. With this, the vehicle speed is increased at a constant rate till it achieves 30 km/h given by equation 4.13.

$$ev\_rs[t] = v[t-1] + \frac{a_{\mathrm{g}}}{fps}, \text{ where } v[0] = 0 \tag{4.13}$$

Now, the final action parameter to be obtained is the Ego Vehicle Reference Steering Angle (*ev_ra*). This angle represents the amount of turn to be applied on the steering wheel for overtaking the obstacle. When the overtake action is initiated, the ego vehicle has to turn right to avoid the obstacle, and when the obstacle is crossed, the vehicle has to make a left turn to bring it to the previous lane. Here, the right turn is represented as a positive angle, and the left turn as negative.

The steering angle to overtake the obstacle strongly depends on the speed of the vehicle. If the vehicle speed is high, then a small turn is sufficient to change the vehicle lane to avoid the obstacle. On the other hand, if the vehicle speed is low, then a higher turn is required to avoid the obstacle. Hence the angle and speed of the vehicle have an inverse relationship. This relationship is modeled in equation 4.14.

$$ev\_ra = ev\_ra_0(1 + \frac{1}{v}) \tag{4.14}$$

In equation 4.14, ev_ra_0 represents a base angle to turn the vehicle which has to be found out through practical testing. It can be also seen that as the vehicle velocity v increases, then the turn angle needed *ev_ra* decreases satisfying the requirement of inverse relationship of steering angle with the vehicle speed. Furthermore, to turn left, only the polarity of the steering angle is reversed to negative.

Now, Ego Vehicle Turn Time (*ev_tt*) needs to be determined. This time indicates the duration for which we have to turn the vehicle left or right to change the lane while overtaking. For arriving at this time, it is assumed that a typical lane width is 2 m and the time the vehicle takes to cover this horizontal distance of 2 m is considered as the turn time. When the vehicle is turned by an angle of θ as shown in Fig. 4.15, then the vertical component of the vehicle speed

is responsible for the horizontal movement. Also, θ is the vehicle wheel angle and hence the gear ratio has to be applied to get the steering corresponding angle. Applying this the turn time is determined by equation 4.15.

$$ev\_tt = \frac{2}{vsin(\frac{ev\_ra}{N})} \tag{4.15}$$

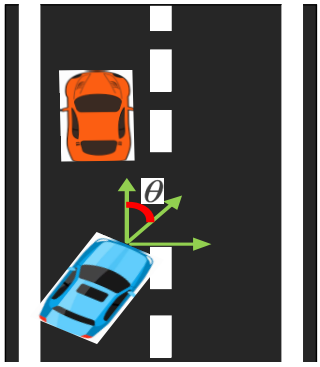


Figure 4.15: Turning of the ego vehicle

Equations 4.5 - 4.15 have given insight into how to find out different decision-making and action parameters. These parameters and their equations are tabulated in table 4.6.

Parameter	Short form	Equation
Primary Obstacle Depth Threshold	po_dt	$\frac{v^2}{2a_g}+1$
Secondary Obstacle Depth	so_dp	$po\_dt+(so\_rs-v)ev\_ot+10$
Obstacle Safe Distance	ob_sd	$\frac{v^2}{2a_e}+1$
Ego Vehicle Reference Speed	ev_rs	$v[t-1]-\frac{a_g}{fps}$, gradual stop $v[t-1]-\frac{a_e}{fps}$, emergency stop $v[t-1]+\frac{a_g}{fps}$, acceleration
Ego Vehicle Reference Steering Angle	ev_ra	$ev\_ra_0(1+\frac{1}{v})$
Ego Vehicle Turn Time	ev_tt	$\frac{2}{vsin(\frac{ev\_ra}{N})}$
Ego Vehicle Overtake Time	ev_ot	$\frac{po\_dt+2}{po\_rs}$
Ego Vehicle Time to Gradual Stop	ev_tg	$\frac{v}{a_g}$
Ego Vehicle Time to Emergency Stop	ev_te	$\frac{v}{a_e}$

Table 4.6: Equations for finding parameters

These equations are implemented in the outer guidance loop for decision-making and taking

action accordingly. In the next subsection details about the subprocesses - Gardual Stop, Emergency Stop, and Initiate Overtake are discussed.

4.3.1.2 Obstacle Handling Subprocesses

Gradual Stop and Emergency Stop subprocesses are similar as shown in Fig. 4.16 and Fig. 4.17.

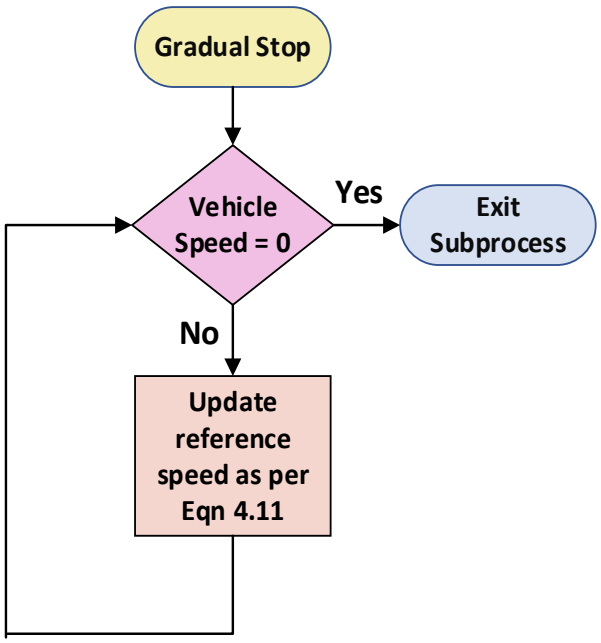


Figure 4.16: Gradual stop subprocess

In both these subprocesses, until the vehicle speed reaches 0, the vehicle speed reference is updated based on the respective equations. This updation happens in every iteration of the outer loop.

The Initiate Overtake subprocess is a bit more complex than the Stop subprocess as it also needs to keep a check on the secondary obstacle. The flow chart of the Initiate Overtake subprocess is shown in Fig. 4.18.

In the overtake subprocess, the ego vehicle is first turned right to avoid the obstacle and keep a safe horizontal distance from the obstacle. The steering reference angle for this is provided in equation 4.14 and the time for which the turning action has to be executed is given in equation 4.15. Once this step is executed, the vehicle is expected to be in the right lane. Now the vehicle has to move forward till the obstacle is overtaken. This straight movement is executed for ev_ot time duration given in 4.10. While moving straight, the ego vehicle also has to check for

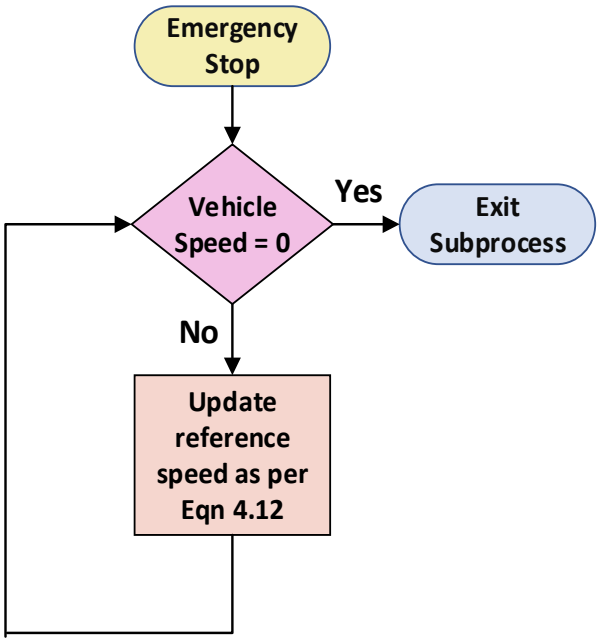


Figure 4.17: Emergency stop subprocess

any secondary obstacles that can appear suddenly. If the obstacle is at a close depth indicated by *ob_sd*, then the vehicle has to be brought to rest by the passenger manually. Or else if the secondary obstacle is at a depth less than the secondary obstacle depth threshold, then the autonomous system has to execute an emergency stop to bring the vehicle to rest considering the safety of the passenger and the vehicle. Now, once the ego vehicle crosses the primary obstacle, then the vehicle has to be brought back to the previous lane by executing a left turn (equation 4.14) for a duration of turn time. Once this last step is executed, then the ego vehicle is brought back to normal condition to further move ahead.

By executing the main obstacle-handling process and these three subprocesses, the autonomous driving system will be able to handle obstacle avoidance under the specified six scenarios. In the next section, we will discuss the steps involved in implementing the lane-following algorithm.

4.3.2 Lane Following

Lane following is a fundamental component of autonomous driving systems, ensuring that vehicles can accurately track and maintain their position within a designated lane on the road. By continuously monitoring lane markings, lane-following systems can make precise adjustments to keep the vehicle centered within its lane. This not only enhances safety by preventing unin-

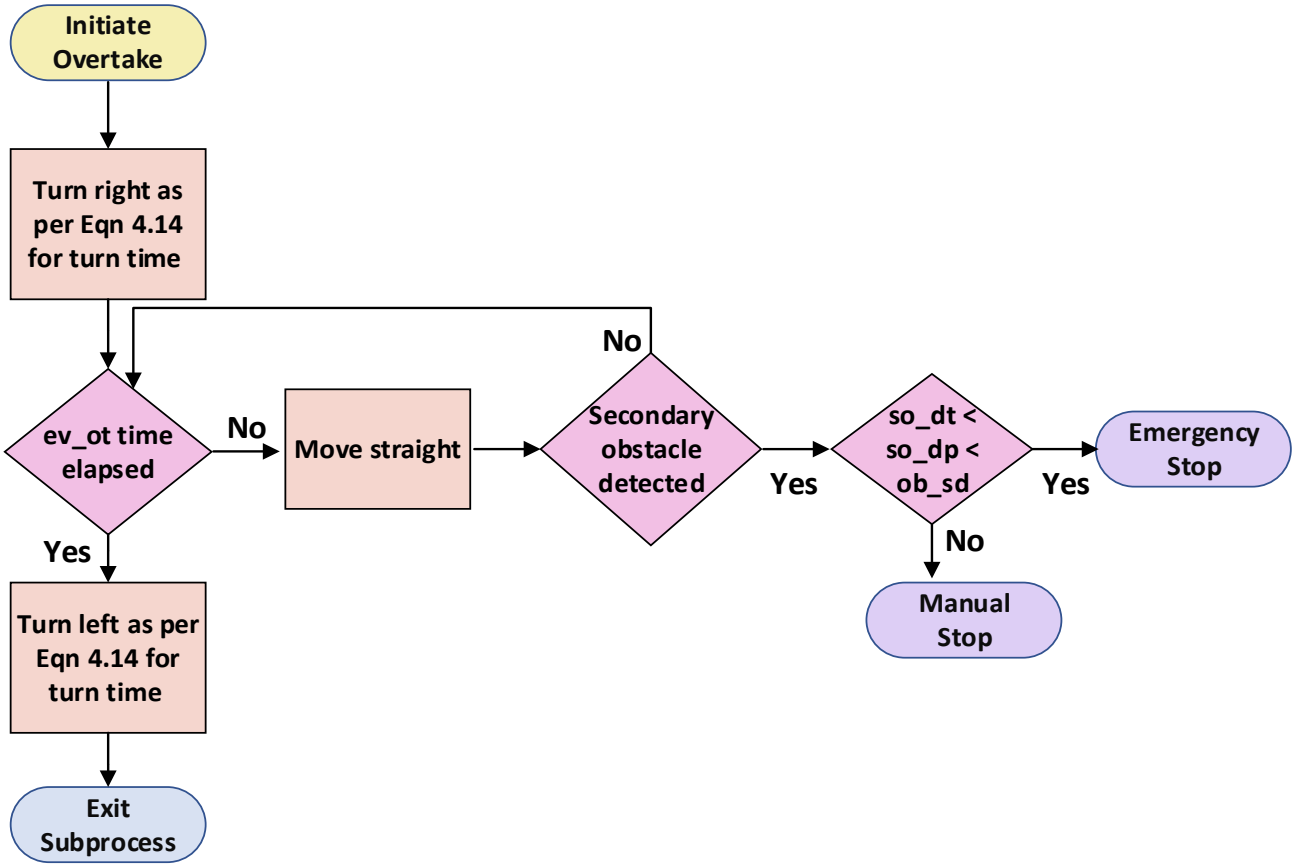
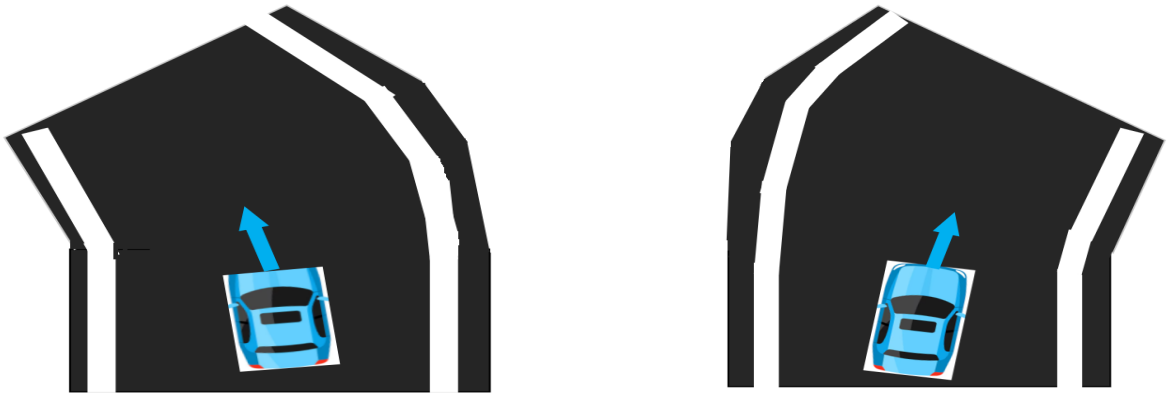


Figure 4.18: Initiate Overtake subprocess

tended lane departures but also contributes to a smoother and more reliable driving experience.

When an autonomous driving car is on a straight road, it generally does not need to make any steering adjustments unless there are obstacles to navigate around. However, lane following becomes particularly important when the road includes implicit turns. In these situations, the vehicle must follow the curvature of the road while maintaining its position in the center of the lane. This section will discuss how autonomous vehicles handle these intrinsic turns on the road. There are two cases for handling these turns, viz., left curvature and right curvature on the road as shown in Fig. 4.19.

In both the left and right curvatures, the steering angle of the vehicle has to be modified to follow the lane. To develop an algorithm for lane following and to understand the scenario further, the lane detection algorithm is run on three scenarios - straight lane, left curvature, and right curvature as shown in Fig. 4.20. Here, two points are introduced in the lane detection algorithm - the projected point and the target point [59]. A projected point is a point located in the imaginary line drawn from the center of the bottom of the image to the top of the image



Case 1a: Left curvature

Case 1b: Right curvature

Figure 4.19: Lane following on-road curvatures

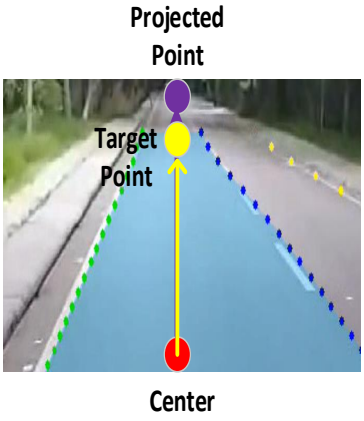
making it perpendicular to the horizontal axis. The target point is the point at the center of an imaginary line drawn between the last point in the left and the right lane. On a straight lane, the lines drawn from the image's bottom center to the projected point and the target point are collinear. However, if there is any curvature in the road, then these two lines form an angle between them which is an indication of the amount of curvature. So, the vehicle has to make a suitable correction equivalent to this angle to follow the lane.

This information can be incorporated into the reference steering angle. When the curvature angle is θ, the vehicle has to turn by the same angle θ. However, the gear ratio of the vehicle also has to be incorporated while deciding the reference steering angle given by equation 4.16.

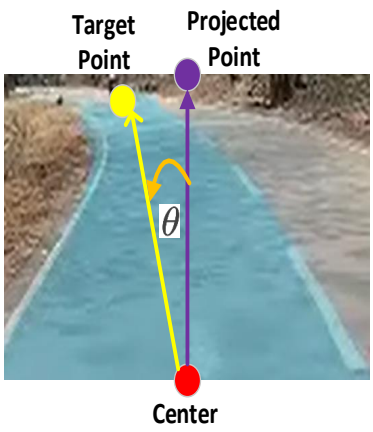
$$ev\_ra = N\theta, \text{ where } \theta \text{ is the curvature angle} \tag{4.16}$$

By implementing the equation 4.16, the autonomous driving vehicle will be able to follow its lane. However, it can be noted from equation 4.14 in the object detection and equation 4.16 in the lane following, the same variable *ev_ra* is updated. Hence, a priority for a task needs to be provided to avoid wrong calculations. Since object detection is of higher importance considering the safety aspects, reference angle updation in the object detection algorithm is given priority over lane detection. This means that the lane detection algorithm will be implemented only when there are no primary obstacles on the road. Hence, this is one of the limitations of the system to handle obstacles on a curved road.

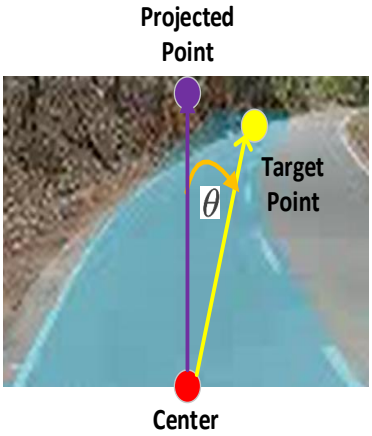
During the lane-following process, maintaining an appropriate speed to avoid collisions and ensuring safety is crucial. To ensure that a dynamic updation of the speed reference based on the traffic scenario is implemented. The algorithm for it is shown in Fig. 4.21.



(a) On straight lane



(b) On lane left curvature



(c) On lane right curvature

Figure 4.20: Projected and target points for different lane conditions

In the algorithm, when the car starts and enters the lane, the system immediately checks for obstacles. If the number of obstacles is greater than 10, the system sets the target speed to 10 km/h, ensuring the car navigates cautiously in a highly congested environment. If the number of obstacles is greater than 5 but not more than 10, the system sets the target speed to 20 km/h, indicating a moderate level of obstacles where a slightly higher speed is safe. If the number of obstacles is 5 or fewer, the car sets the target speed to 30 km/h, allowing for faster travel in relatively clear conditions. This system ensures that the car adjusts its speed dynamically based on real-time obstacle detection, balancing safety and efficiency.

So far in the document, two references - steering angle and speed were discussed. However, the speed control module is expecting a torque reference instead of a speed reference, and the term speed reference was used for clarity and conciseness. Now, the conversion of speed reference

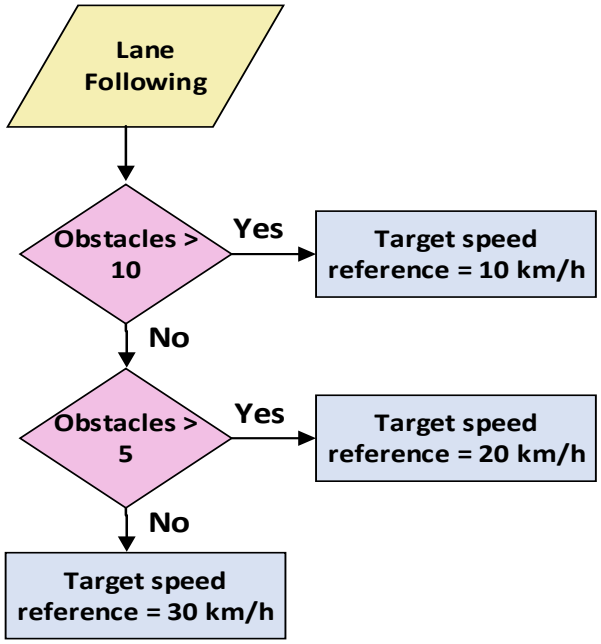


Figure 4.21: Speed reference updation during lane following

to torque reference will be discussed.

There are three different desired accelerations (a_d) the vehicle is expected to carry out depending on the situation - Gradual acceleration (a_g), Gradual deceleration (a_g), and Emergency deceleration (a_e). The desired acceleration force of the vehicle is given by equation 4.17.

$$ev\_df = Ma_{\mathrm{d}} \tag{4.17}$$

In addition to the desired acceleration force, the motor also has to generate force to overcome the aerodynamic drag and the rolling force (ignoring gradient force) of the vehicle mentioned in equations 4.2 and 4.4. So the torque reference to the SCM is given by equation 4.18.

$$ev\_dt = R(Ma_{\mathrm{d}} + \frac{1}{2}\rho A_{\mathrm{f}} C_{\mathrm{d}} v^2 + C_{\mathrm{r}} Mg) \tag{4.18}$$

This desired torque is given as a reference to the torque control loop in the speed control module.

So far, different approaches, algorithms, equations, and limitations of localization, path planning, and reference generation have been discussed. In the next section, some results obtained for these modules will be discussed.

4.3.3 Results and Discussion

All the algorithms and equations detailed in the previous sections were implemented on the Jetson Orin Nano. In this section, we discuss some of the results obtained for reference generation. First, we analyze the outputs of the equations listed in Table 4.6 under different input conditions. This analysis is crucial to understanding the system's performance across various operating conditions and to define the vehicle's operating region.

Firstly, Fig. 4.22 illustrates the variation in different depth thresholds for various speeds of the ego vehicle and secondary obstacles. In the first figure, the Primary Obstacle Depth Threshold and the Safe Obstacle Distance for different ego vehicle speeds are depicted. The depth threshold required for a gradual stop or to initiate an overtaking maneuver is greater than that for initiating a manual stop action. For example, when the ego vehicle is traveling at 20 km/h, it initiates an overtaking or gradual stop maneuver when the primary obstacle is 6 meters away. Conversely, if the obstacle is between 4 and 6 meters away, an emergency stop is executed. If the obstacle is closer than 4 meters, the passenger must manually stop the vehicle.

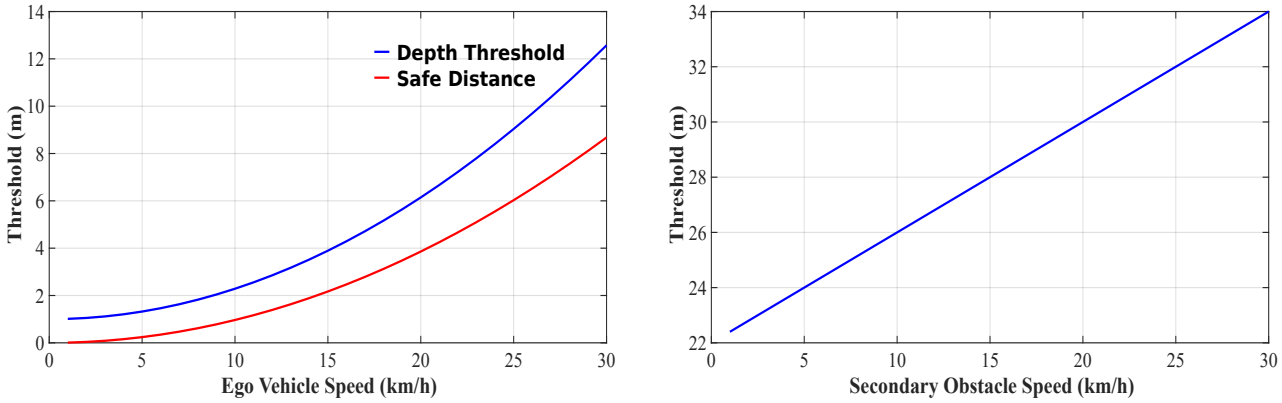
The safe stationary secondary obstacle ranges for some ego vehicle speeds are tabulated in table 4.7. This information needs to be considered by the passenger to take emergency manual action for safety aspects.

Ego Vehicle Speed (km/h)	Safe Obstacle Distance (m)
10	1
20	3.8
30	8.6

Table 4.7: Safe obstacle ranges for different vehicle speeds

In the second figure, the Secondary Obstacle Depth Threshold during overtaking for various absolute speeds of the secondary obstacle is shown. Here, the ego vehicle speed is set at 30 km/h. The plot reveals that when the secondary obstacle is stationary, the secondary obstacle depth threshold is 22 meters, meaning an overtaking maneuver will only be initiated if the secondary obstacle is at a depth higher than 22 m. It can be also noticed that the threshold increases as the obstacle speed increases.

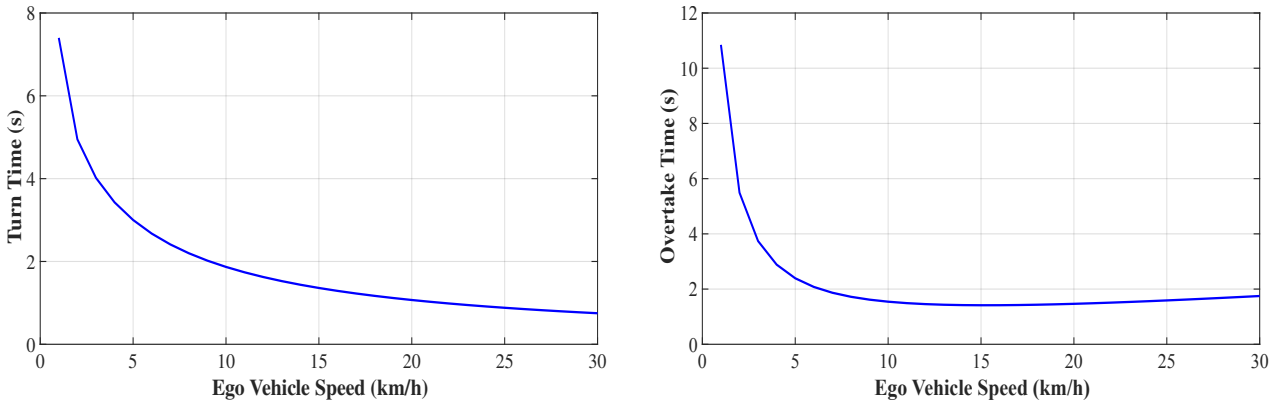
In Fig. 4.23, the turn time and overtake time for the ego vehicle at various speeds are depicted. The first graph shows that the time needed for the vehicle to turn decreases as the speed of the ego vehicle increases. Meanwhile, the second graph indicates that the overtake time decreases with increasing speed up to 8.6 km/h, after which it begins to increase as the vehicle speed



(a) Depth threshold and safe distance limit (b) Secondary obstacle depth threshold

Figure 4.22: Different depth thresholds vs speed of the vehicle/obstacle

continues to rise.



(a) Turn time (b) Overtake time

Figure 4.23: Turn time and overtake time for different ego vehicle speeds

Additionally in Fig. 4.24, the time taken for the ego vehicle to stop gradually and suddenly in case of an emergency situation is shown. The linear proportionality of stop time with the vehicle velocity can be seen in the figure. Also, the vehicle stops quickly in an emergency situation compared to a gradual stop considering the safety requirement.

Finally, Fig. 4.25 shows the steering and speed references generated for various conditions. In the first plot, the vehicle reference steering angle for various speed conditions is shown. It is seen that the steering reference angle has an inverse relationship with the ego vehicle speed. This can be justified as the vehicle moving at a faster rate needs to turn less to change its lane compared to a slower vehicle. The second plot shows the reference speed at different time instants for various conditions - when the vehicle starts from rest, when the vehicle stops

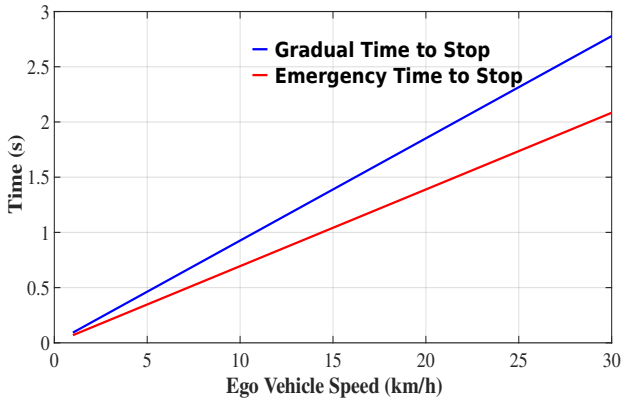
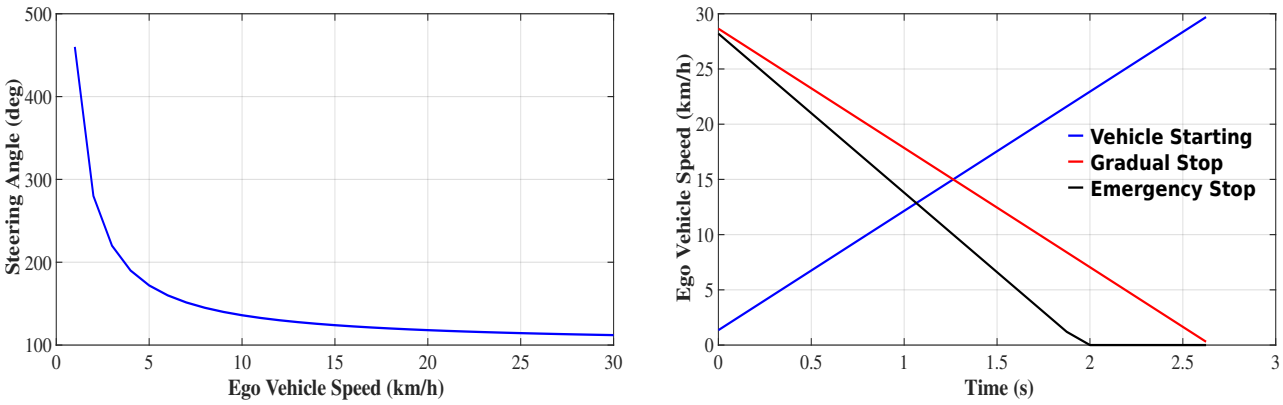


Figure 4.24: Time to stop the vehicle for different vehicle speeds

gradually, and on an emergency stop. For all these three conditions, the vehicle speed is fixed at 30 km/h. It can be noticed that the vehicle takes 2.7 seconds to reach the allowed maximum speed of 30 km/h from rest and the same time to bring it to a gradual stop. On the other hand, an emergency stop will bring the vehicle to rest in 2 s.



(a) Reference steering angle

(b) Reference speed

Figure 4.25: Vehicle reference commands

So far the individual outputs of each of the equations were outlined. Now the response of the autonomous driving system for different scenarios mentioned in the previous sections will be discussed. At first, scenario 2, i.e., overtaking a primary obstacle at different speeds of 10 km/h, 20 km/h, and 30 km/h is considered. The steering angle for each of these cases is shown in Fig. 4.26.

From the figure, it can be seen that when the ego vehicle is moving at 10 km/h, then the overtake action is initiated at a primary obstacle depth of 4 m. However when the speed is 20 km/h and 30 km/h, the action depth increases to 7 m and 11 m respectively. Additionally, it can be also noticed that the turn time and the overtake time for all three cases are of comparable magnitude.

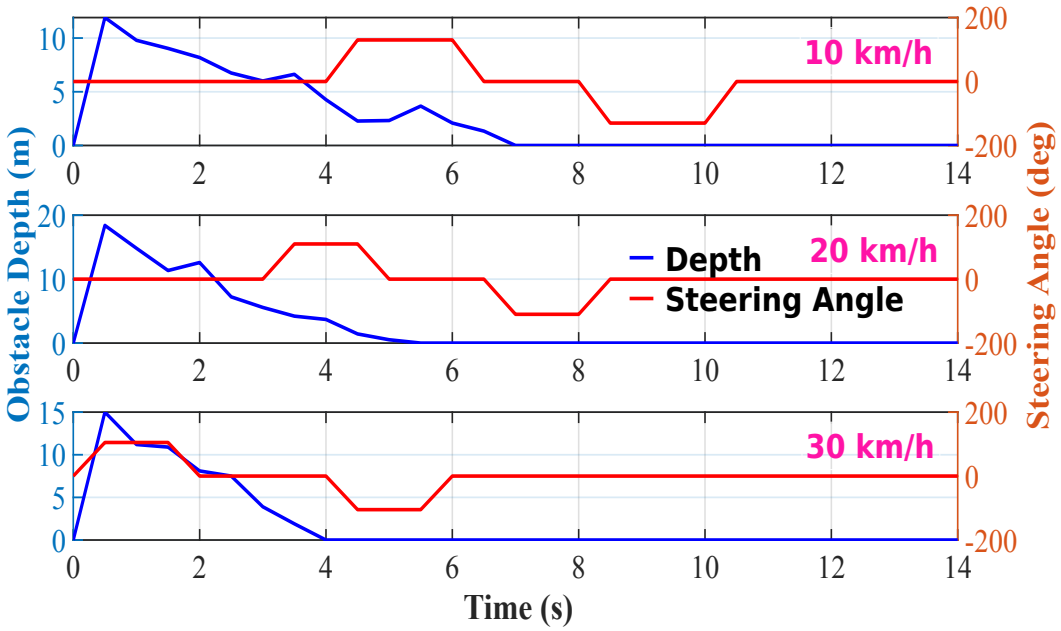


Figure 4.26: Steering angle for overtaking primary obstacle at various vehicle speeds

In the next scenario, case 3, an obstacle appears suddenly at a depth of 1.8 m in front of the ego vehicle traveling at 10 km/h, Fig. 4.27. Since this depth is less than the depth threshold but greater than the safe distance limit, the vehicle executes an emergency stop command, whereby the vehicle is brought to rest at a deceleration rate of -4m/s^2. In this case, the vehicle stops with a gap of 0.84 m with the primary obstacle.

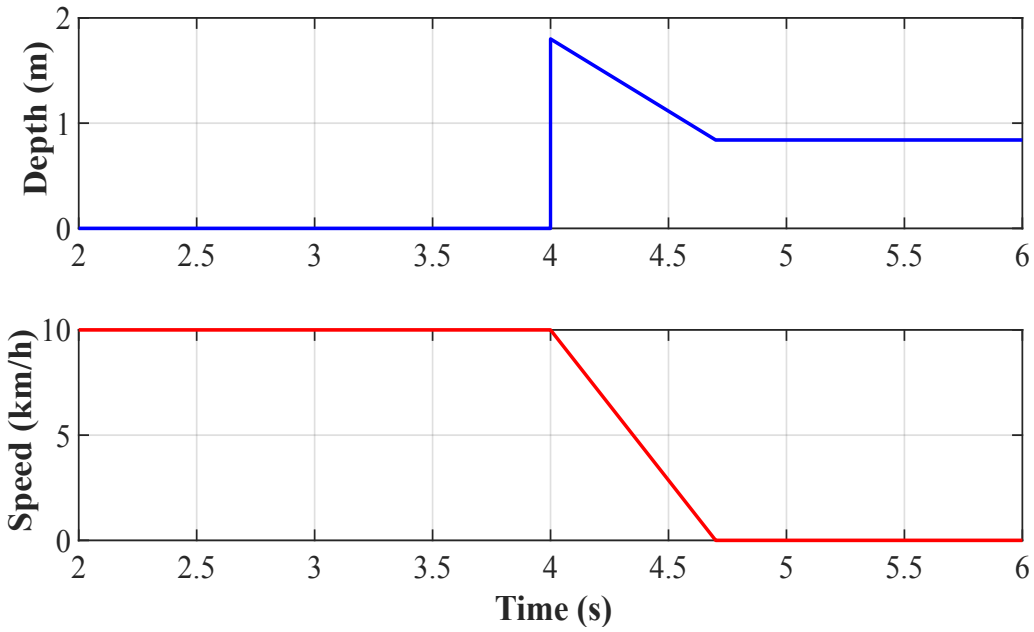


Figure 4.27: Emergency stop of the vehicle

The fourth scenario considers the presence of a secondary obstacle on the overtaking lane. This scenario has 4 subcases - secondary obstacle far away, secondary obstacle nearby, secondary obstacle moving towards the ego vehicle, and the secondary obstacle appearing suddenly at a shorter depth triggering emergency stop action. Response of the system for these subcases are illustrated in Fig. 4.28 - 4.30. In all these subcases, the ego vehicle is initially traveling at a speed of 10 km/h.

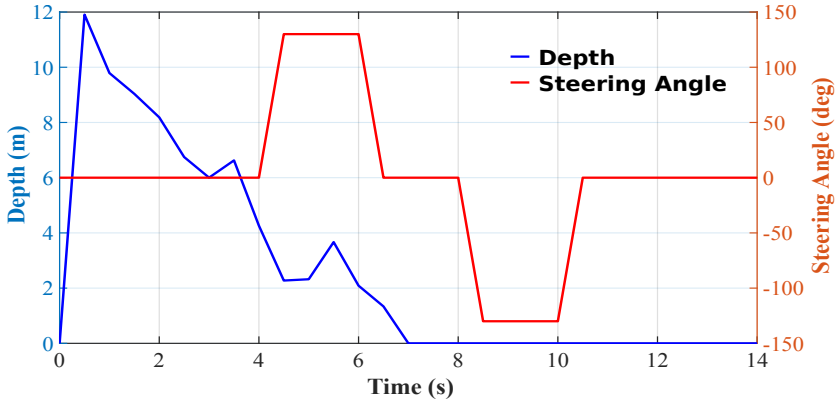


Figure 4.28: Case 4b: Overtaking when the secondary obstacle is far away

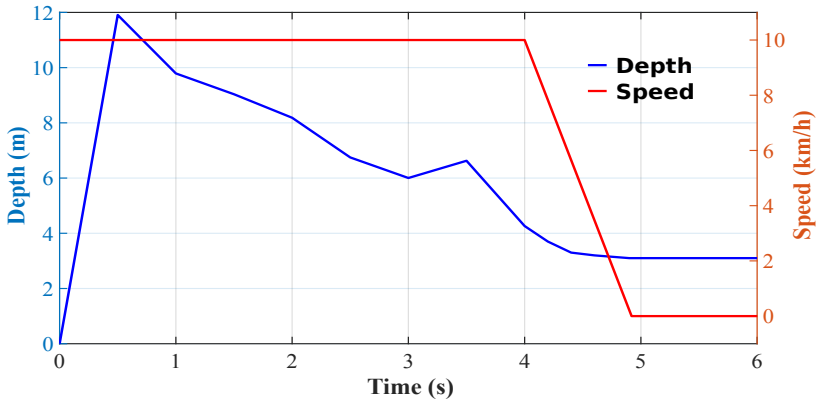


Figure 4.29: Case 4b and 4c: Secondary obstacle poses potential collision

In case 4a, since the secondary obstacle is far away beyond *so_dt*, the ego vehicle overtakes the primary obstacle similar to case 2b. In cases 4b and 4c, the secondary obstacle can become a potential collision source. So in these two cases, the vehicle is gradually brought to rest at a deceleration rate of -3 m/s^2. It is also to be noted that the vehicle stops at a safe distance of 3 m away from the primary obstacle. In the final subcase 4d, the secondary obstacle appears suddenly when the ego vehicle is overtaking the primary obstacle at a time instant of 4.5 seconds. In this case, as the secondary obstacle is within the safe distance for the emergency action, the vehicle decelerates at -4 m/s^2 and achieves rest by 5.2 s.

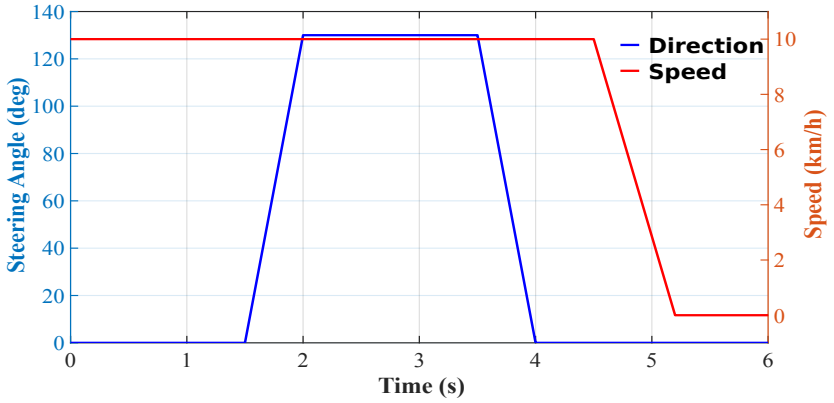


Figure 4.30: Case 4d: Secondary obstacle appearing closeby

4.4 Steering Control Law

The reference generation block has generated speed and steering angle references of the vehicle. Now, a negative closed-loop control of these parameters has to be implemented. Closed-loop control is essential for this application as it ensures robustness to disturbances, accuracy in trajectory following, stability, and safety. By continuously monitoring and adjusting the vehicle's speed and steering angle based on real-time feedback, the system can handle dynamic environments and unexpected changes, maintaining desired performance and ensuring safe, stable operation. In the system implementation of the vehicle, the speed control loop is implemented in a different Speed Control Module (SCM). So the generated speed reference which is converted to a torque reference is given to the SCM through a DAC module. In this section, only the steering control law implementation is discussed.

The control loop of the steering reference is shown in Fig. 4.31. The control loop has a comparator that compares the steering reference with the steering feedback signal from a potentiometer. The generated error is used to compute corrective actions. This error is passed through a control law and outputs a Pulse Width Modulated (PWM) signal to the DC motor control unit. The control law essentially translates the car's understanding of the situation (via potentiometer feedback) into real-time control actions (steering). This continuous feedback loop guided by the control law ensures that the car stays on track, adapts to changing environments, and maneuvers safely.

The DC motor steering control can utilize various control laws like classical control using proportional, integral, and derivative (PID) units or State Space control or Model Predictive Control (MPC). PID control offers a simple and widely used approach, adjusting motor input based on proportional, integral, and derivative terms of the error signal. State-space control

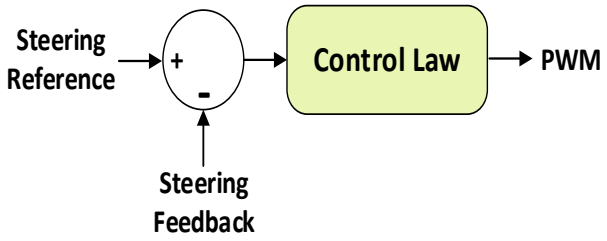


Figure 4.31: Control law for steering motor control

leverages a mathematical model of the motor to calculate optimal control inputs, while Model Predictive Control (MPC) goes a step further by predicting future behavior and optimizing control sequences for complex systems with constraints. The classical PID control is simple and achieves good performance without complex modeling. So a simple PID control is employed for controlling the steering motor in a closed loop.

Each of the elements in the PID control plays a crucial role. The proportional term provides immediate corrective action based on the magnitude of the error. The integral term accumulates past errors over time and helps eliminate any steady-state error by continuously adjusting the output. Whereas, the derivative term predicts future behavior based on the rate of change of the error, providing damping and stability to the system by counteracting rapid changes in the error signal. Since a rapid action is not demanded from the steering motor and to make the system simple, the derivative term is not included in the classical control method. The continuous domain representation of the PI module is given by equation 4.19 where e is the error signal, K_p is the proportional term, and K_i is the integral term.

$$PWM = (K_\mathrm{p} + \frac{K_\mathrm{i}}{s})e \tag{4.19}$$

Since this control law is implemented on a digital controller, the continuous domain representation of the control law has to be modified to the discrete domain by using Tustin's transformation as given by equation 4.20.

$$PWM = (K_\mathrm{p} + \frac{K_\mathrm{i}T}{2}(\frac{z+1}{z-1}))e \tag{4.20}$$

In this equation, the parameters K_p and K_i are obtained by tuning the system to meet the desired response of the steering motor.

There are different sensors possible to measure the steering feedback like encoder, potentiometer, LVDT, etc. However, considering the simplicity, low cost, and ease of integration, a po-

tentiometer is used as the sensor for steering feedback measurement. The potentiometer used for this is shown in Fig. 4.32 and its technical details are tabulated in table 4.8. The two fixed ends of the potentiometer are connected to a rigid point and the moving element is connected to the steering column. The potentiometer is powered with 5 V and its output is taken for implementing closed-loop control of the steering.

Figure 4.32: Potentiometer used for steering feedback measurement

Parameter	Value
Mechanical travel	125 mm
Electrical travel	100 mm
Total resistance	10 KΩ

Table 4.8: Technical details of the potentiometer used for steering feedback measurement

In this chapter, we looked at the different components of route planning and control. The localization module enabled the car to determine the vehicle's precise position in the world. The path planning module determined the route from the vehicle's current position to the desired destination. The reference generation module translated these plans into actionable commands that the vehicle can execute. And finally, the control law ensured precise and responsive vehicle steering. In this chapter, we also looked at the results obtained for various algorithms executed for different scenarios on the road. In the next chapter, we will look at the drawbacks of the present method of reference generation and discuss a machine learning-based approach for reference generation in an autonomous driving vehicle.

Chapter 5

ML Based Reference Generation

In the context of autonomous driving, reference generation is crucial for defining the desired trajectory of the vehicle. In the previous chapter, a modular architecture was discussed, consisting of distinct modules for perception, planning, reference generation, and control. In addition to that the reference generation was done using a rule-based approach. This modular approach offers several advantages. Firstly, it enhances modularity, allowing developers to focus on optimizing each module independently. This segmentation simplifies the debugging and maintenance process, as issues can be isolated within specific modules. Additionally, it promotes ease of understanding and collaboration among teams, as each module has a well-defined purpose and function.

However, the modular architecture also presents notable disadvantages. One major drawback is the lack of task coordination across modules. Each module operates independently, which can lead to inefficiencies and suboptimal performance. For instance, the planning module might generate a path that is difficult for the control module to follow accurately, resulting in a less smooth driving experience. Furthermore, modules are often not optimized for the final goal of the entire system. This can lead to situations where the cumulative outcome is not as effective as it could be if the system were designed and optimized holistically from the start. In addition to that, since the approach is based on predefined rules, this model is not scalable to include all the scenarios expected on the road.

To overcome these issues, machine learning-based data-driven approaches can be pursued. In this approach for reference generation, it involves training models on large datasets of driving behavior to predict optimal driving trajectories. These models learn to generate reference paths by identifying patterns and making decisions based on the input data, such as sensor readings and environmental conditions. It essentially combines the perception and reference generation

blocks into a single machine-learning model as shown in Fig. 5.1. There are two different learning methods to implement data-driven reference generation in the autonomous driving vehicle namely End-to-end learning and Reinforcement learning. In this chapter, a detailed study of both these approaches for reference generation is done.

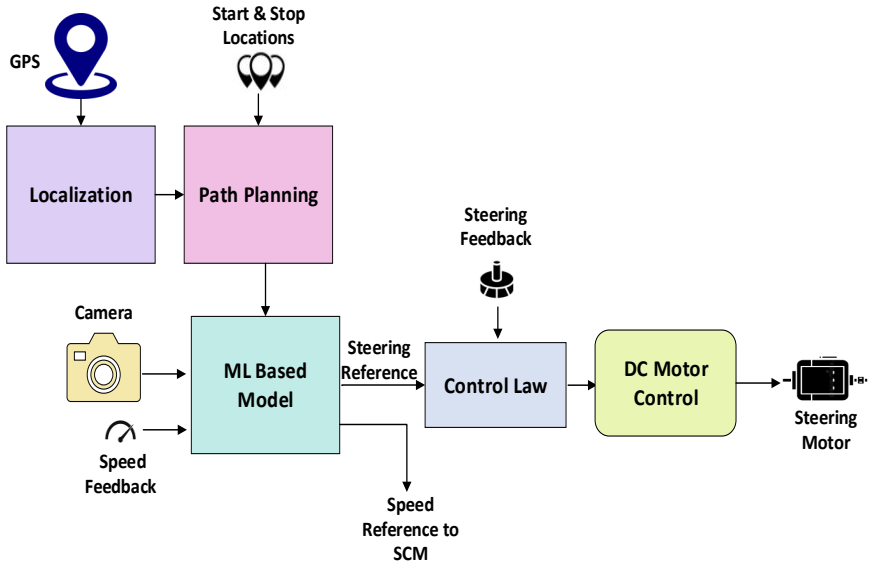


Figure 5.1: Machine learning-based reference generation

5.1 End-to-End Learning

Supervised end-to-end learning entails training a neural network to directly map visual data from the camera to driving actions, such as steering and speed commands, without explicit intermediate representations or modules. When an input camera is fed to the neural network, the neural network processes these images, extracting relevant features and patterns directly related to driving behaviors. Through extensive training on a diverse dataset, the network learns to associate specific visual cues with appropriate steering and speed commands, effectively generating the desired driving references in real-time. This approach bypasses the need for handcrafted features or intermediate representations, allowing for a more streamlined and data-driven solution to reference generation in autonomous driving systems. There are several research works that have implemented end-to-end learning in autonomous driving vehicles like [60], [61], [62] etc. In this study, we implement an end-to-end learning-based reference generation based on the work by Bojarski et. al [60].

The deep learning model for implementing the end-to-end learning-based steering angle refer-

ence generation is shown in Fig. 5.2. Initially, the model takes in RGB images of size 66x200 pixels. These images are normalized to improve the training process by scaling pixel values to a consistent range. At the heart of the model lies a sequence of convolutional layers designed to extract and refine features from the input images progressively. The initial convolutional layer employs 24 filters with a 5x5 kernel, reducing the spatial dimensions to 31x98. This is followed by a second convolutional layer with 36 filters, maintaining the 5x5 kernel size and further reducing the dimensions to 14x47. The third convolutional layer increases the filter count to 48 while continuing with the 5x5 kernels, resulting in feature maps of size 5x22. The subsequent convolutional layers utilize smaller 3x3 kernels; the fourth and fifth layers each apply 64 filters, producing final feature maps of 3x20 and 1x18, respectively. These layers work together to distill the raw image data into a compact and informative representation.

The extracted features from the convolutional layers are then flattened into a one-dimensional vector, which feeds into a series of fully connected layers, serving as the high-level reasoning component of the model. These fully connected layers consist of 1164, 100, 50, and 10 neurons sequentially, integrating the refined features to form a coherent understanding of the driving environment. The final output layer generates a single value representing the steering angle, which the vehicle uses to navigate. This end-to-end approach allows the model to learn directly from raw image data to control commands, streamlining the development process by eliminating the need for handcrafted features and enabling the system to adapt and improve through continuous learning from its driving experiences.

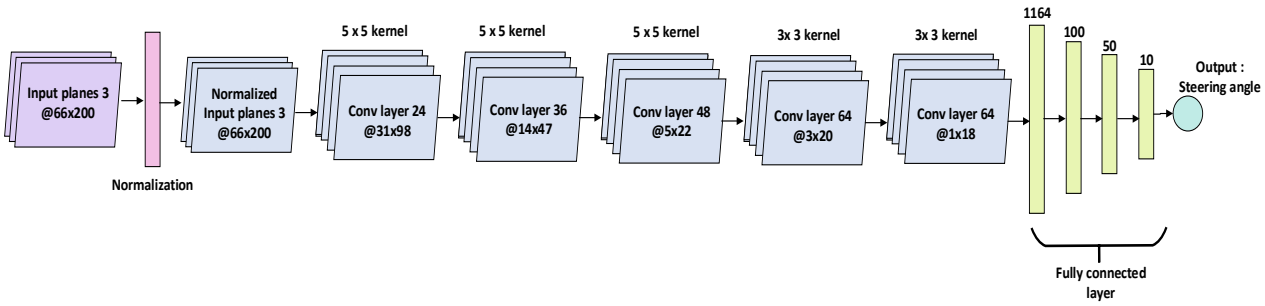


Figure 5.2: End-to-end learning model for steering angle reference generation

The training architecture for the model is shown in Fig. 5.3. The training process begins with images captured by three cameras positioned on the left, center, and right of the vehicle. These images are subjected to random shifts and rotations to augment the training data, enhancing the model's robustness to variations in the driving environment. The recorded steering wheel angle, which serves as the ground truth, is also adjusted for these shifts and rotations to maintain alignment with the augmented images.

Once the images are preprocessed, they are fed into the model, which extracts relevant features and computes a steering command based on the input images. This computed steering command is compared to the desired steering command, derived from the adjusted recorded steering wheel angle, to calculate an error. The error represents the difference between the network's prediction and the actual steering command. To minimize this error, backpropagation is employed to adjust the weights of the neural network. This iterative process involves propagating the error backward through the network and updating the weights to reduce the discrepancy between the predicted and actual steering commands in future iterations. By continuously refining the network through this feedback loop, the model learns to accurately predict the appropriate steering angle from the input images, ultimately improving its performance in real-world driving scenarios.

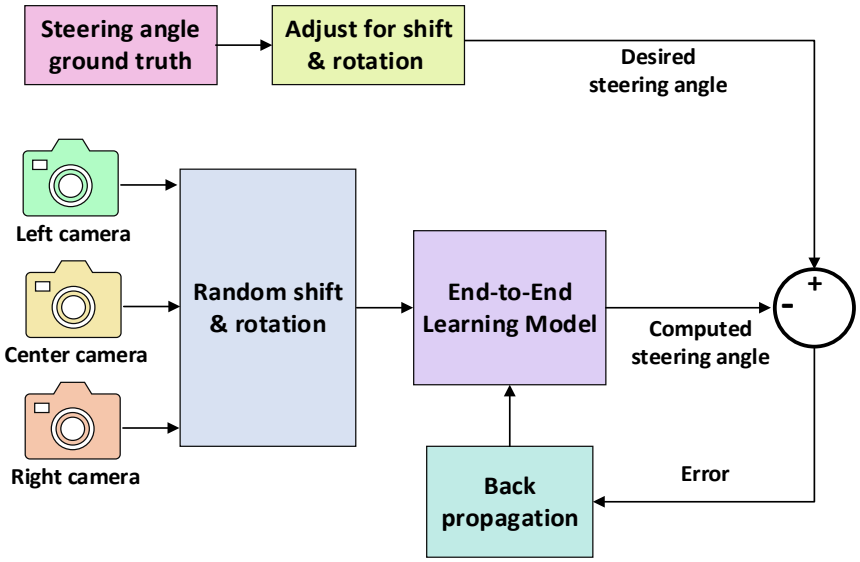


Figure 5.3: Training architecture for steering angle reference generation

The inference architecture for the model is shown in Fig. 5.4. During the inference phase, the setup involves the vehicle's center camera continuously capturing real-time images of the driving environment. These images serve as the input to the previously trained model. The trained model processes the incoming images to extract relevant features and compute the appropriate steering command based on its learned patterns and knowledge. This steering command is used to drive the steering wheel to follow the lane.

Training of the model was done using the dataset provided in [63], [64]. Some of the sample images of the training dataset are shown in Fig. 5.5. This training set contains data from a wide variety of conditions like different lighting and weather settings. However, this training dataset

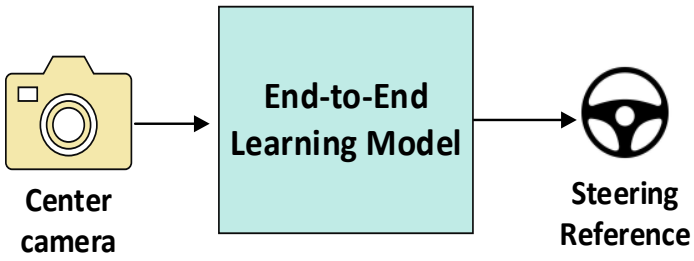


Figure 5.4: Inference architecture for steering angle reference generation

is limited to the roads and conditions of the United States of America.

Figure 5.5: Sample images used for training the end-to-end model for steering angle generation

Once the training was completed, the model was evaluated on the testing dataset. The steering reference obtained from the inference model for the testing video, along with the actual steering angle expected for the same dataset, is plotted in Fig. 5.6. In this figure, a positive angle indicates a right turn, while a negative angle indicates a left turn.

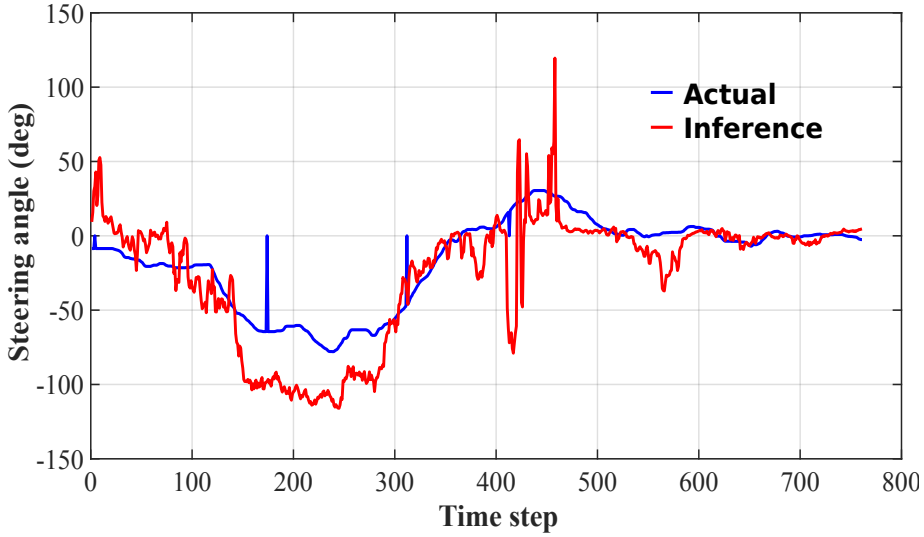


Figure 5.6: Actual and inference steering angle reference

It can be observed from the plot that the inference angle pattern generated by the model matches

the pattern of the actual desired angle. For instance, during the first 350 time steps, the vehicle is required to turn left, and the model's output also suggests a left turn. However, it is also noticeable that there is a significant error in matching both angles. There are occasional spikes in the model output, and the angle reference provided by the model does not exhibit a smooth profile, which would be necessary for a comfortable journey. These mismatches with the desired angle can be improved by increasing the complexity of the model and training it with more extensive datasets. Additionally, to achieve a smoother profile, imitation learning techniques can be pursued, or a smoothing function can be implemented as part of the post-processing steps.

The implementation of end-to-end learning for steering angle reference generation has shown promising results, with the model capable of generating steering angles satisfactorily. However, there remains significant potential for enhancement in accuracy and robustness. Future work could focus on refining the model architecture, incorporating additional sensory inputs, and leveraging advanced training techniques to further improve performance. The scope for end-to-end learning in steering angle prediction is vast, offering the potential for more adaptive, efficient, and intelligent driving systems that can better handle a wide range of driving conditions and scenarios.

In the next section, we discuss another machine learning approach for reference generation - Reinforcement Learning.

5.2 Reinforcement Learning

Reinforcement learning (RL) is a subset of machine learning where an agent learns to make decisions by performing actions in an environment to maximize cumulative rewards. Unlike supervised learning, which relies on labeled data, RL is based on the trial-and-error approach where the agent explores various actions and receives feedback in the form of rewards or penalties. The core components of RL include the agent, the environment, the policy, the reward signal, and the value function. The agent interacts with the environment in discrete time steps, and the policy determines the action taken by the agent at each state. The goal is to find an optimal policy that maximizes the expected cumulative reward over time. The value function estimates the expected reward for being in a given state and following a particular policy thereafter, guiding the agent toward more rewarding actions.

Reinforcement learning offers significant advantages over traditional rule-based modular architecture in the steering angle reference generation for autonomous driving. RL enables the

development of adaptive systems that learn optimal driving strategies directly from interactions with the environment, allowing for continuous improvement and adaptation to new situations. This ability to learn from experience and refine strategies based on actual driving conditions leads to more robust and flexible autonomous driving solutions, capable of handling the complexities and variability inherent in real-world driving environments. Because of these advantages offered by RL-based decision-making, the current state-of-the-art autonomous driving vehicles heavily depend on it for speed and steering reference generation [65], [66], [67]. In this study of reinforcement learning-based reference generation, a Deep Deterministic Policy Gradient (DDPG) based algorithm [68],[69] is implemented and analyzed.

Deep Deterministic Policy Gradients is a reinforcement learning algorithm to solve continuous action space problems, making it particularly suitable for tasks such as autonomous driving. Unlike traditional RL algorithms like Q-learning and Deep Q-Networks (DQN), which struggle with continuous action spaces, DDPG employs an actor-critic architecture that learns a deterministic policy, enabling it to handle continuous actions more effectively. One of the key advantages of DDPG is its ability to learn policies in high-dimensional action spaces, making it well-suited for tasks where fine-grained control is necessary, such as steering, acceleration, and braking in autonomous driving scenarios. In addition to that, DDPG incorporates experience replay and target networks to stabilize training and improve convergence.

The DDPG learning framework is shown in Fig. 5.7. In the DDPG algorithm, an Actor-Critic framework is employed to enable efficient learning in continuous action spaces. In the framework, the Actor-network serves as the decision-maker, considered as the car's brain. It analyzes the environment surrounding the vehicle, represented as a state vector derived from sensor data. Based on this analysis, the Actor determines appropriate actions (steering angle, acceleration, and braking). Its primary objective is to learn a policy that maximizes the cumulative reward obtained during driving, aiming for both safety and efficiency.

Complementing the Actor, the Critic network acts as a reviewer. It evaluates the value of the state-action combinations suggested by the Actor, essentially predicting the future reward potential associated with each action in a given state. The Critic's feedback is crucial for refining the Actor's policy, guiding it towards decisions that lead to states with higher expected rewards.

The learning process in DDPG unfolds in a continuous loop. First, the car observes its surroundings through sensors, translating this information into a state representation. Based on this state, the Actor recommends an action. Subsequently, the car executes the action, receiving a reward signal based on the outcome, which could include positive rewards for safe driving

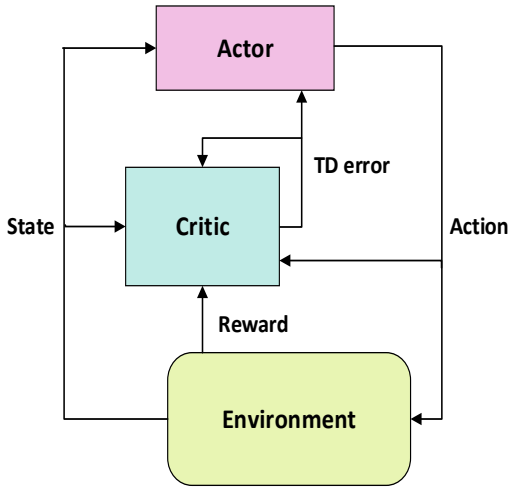


Figure 5.7: DDPG actor-critic network

behaviors and penalties for collisions or violations. After experiencing the action, the Critic evaluates the state-action pair, updating its understanding of the expected future rewards. This evaluation is crucial for learning from past experiences and guiding future decision-making.

The Actor in this case is a neural network with two hidden layers of 300 and 600 neurons each as shown in Fig. 5.8. The input states (discussed later) are passed through this hidden layer to obtain three outputs - Steering angle (-1 means max right turn and +1 means max left turn), Acceleration (0 means no acceleration and 1 means full acceleration), and Brake (0 means no brake, 1 means full brake). The activation functions of the output layer for each of them are different. Steering angle uses a tanh activation function, whereas acceleration and brake use a sigmoid activation function.

The critic network takes both the states and the action as inputs and passes through a set of hidden layers after which they are merged together to form the output as shown in Fig. 5.9.

Since the DDPG model learns its parameters by performing actions in the environment, a software simulator is required to train it. The Open Racing Car Simulator (TORCS) simulator is used for that purpose. TORCS is a simulator for testing AI-driven autonomous driving systems. It offers a wide range of cars, tracks, and customizable options to train and do the inference of the model. Also, TORCS is open source and provides a realistic, controlled, and accessible environment for testing and refining control strategies. TORCS has provision for interfacing up to 18 different sensors. However, for this DDPG-based implementation, only 8 sensors are used to form the vehicle state as tabulated in table 5.1. These 8 sensor readings form the state vector for the actor and critic networks.

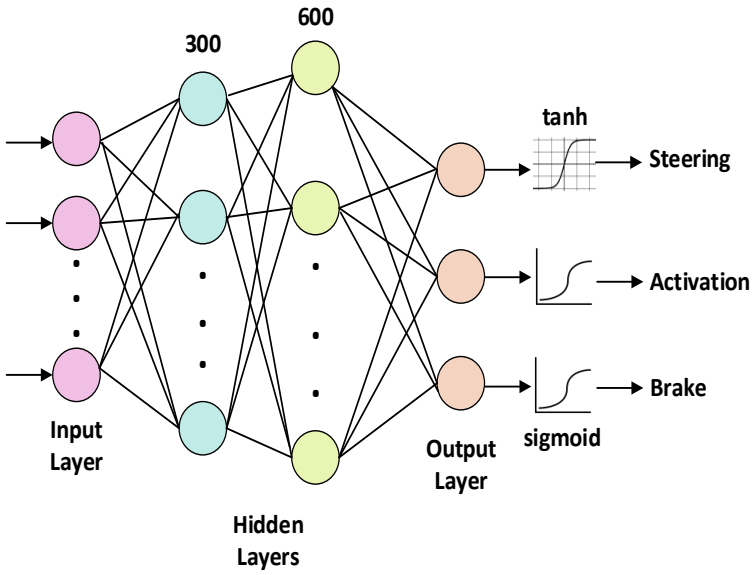


Figure 5.8: DDPG actor network

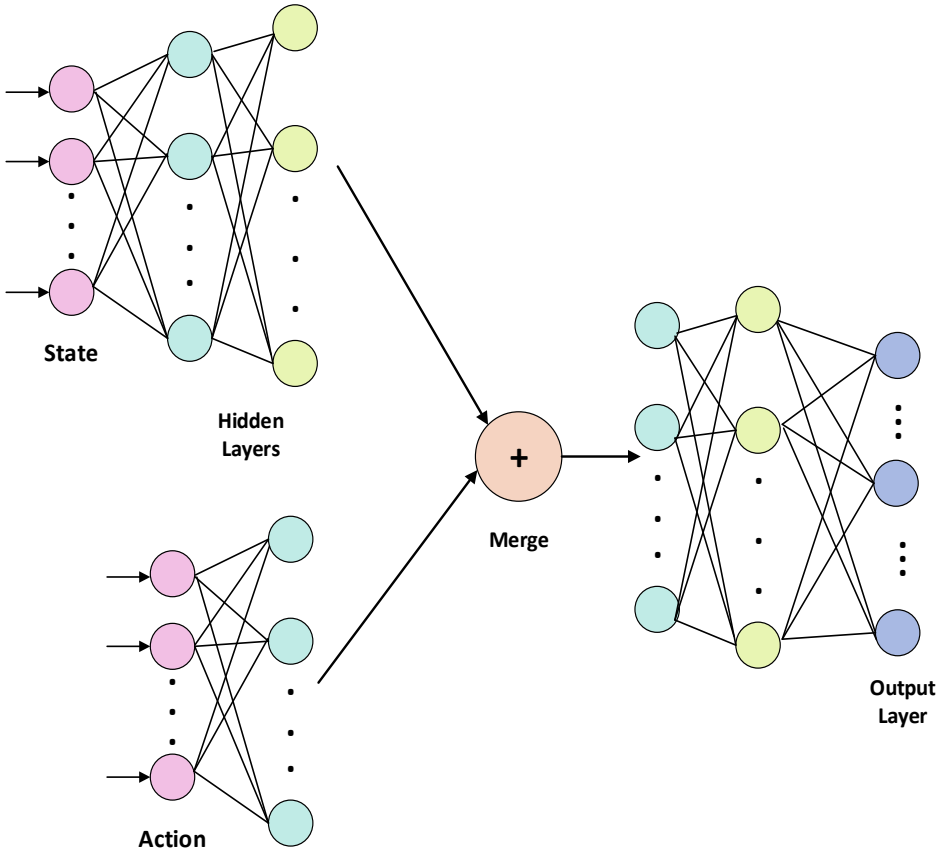


Figure 5.9: DDPG critic network

Name	Description
Angle	Angle between the car direction and the direction of the track axis
Track	Distance between the track edge and the car
TrackPos	Distance between the car and the track axis
speedX	Speed of the car along the longitudinal axis of the car
speedY	Speed of the car along the transverse axis of the car
speedZ	Speed of the car along the Z-axis of the car
WheelSpinVel	Rotation speed of wheels
RPM	Number of rotation per minute of the car engine

Table 5.1: Sensors used for DDPG implementation

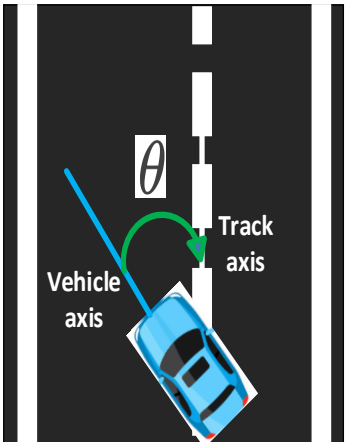


Figure 5.10: Angle formed by vehicle axis and track axis

The next step is to arrive at an optimum reward function. The reward function quantifies the desirability of the given state-action pair, guiding the actor to maximize long-term returns by learning to select actions that yield higher cumulative rewards. The reward function chosen for the DDPG algorithm is given in equation 5.1. In the equation, V is the speed of the vehicle, and θ is the angle made by the vehicle axis with the track axis as shown in Fig. 5.10. The reward function is designed to maximize the longitudinal velocity (first term), minimize the transverse velocity (second term), and penalize if it drives very off-center of the track (third term).

$$R_t = Vcos(\theta) - Vsin(\theta) - V|TrackPos| \tag{5.1}$$

Training of the model is done in TORCS. The system is trained to follow the lane in a traffic environment. Sample images of the TORCS environment are shown in Fig. 5.11. The steering angle and rewards obtained during the training are shown in Fig. 5.12.

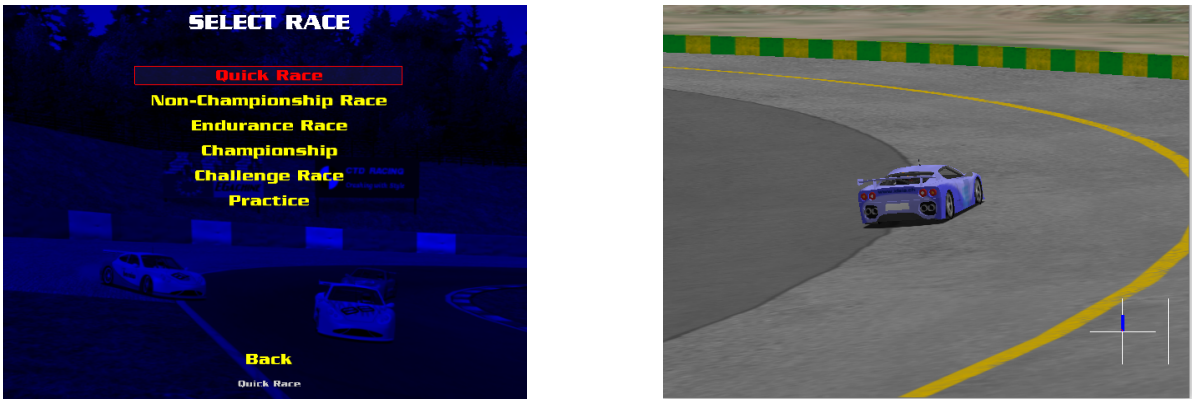


Figure 5.11: TORCS environment for DDPG training

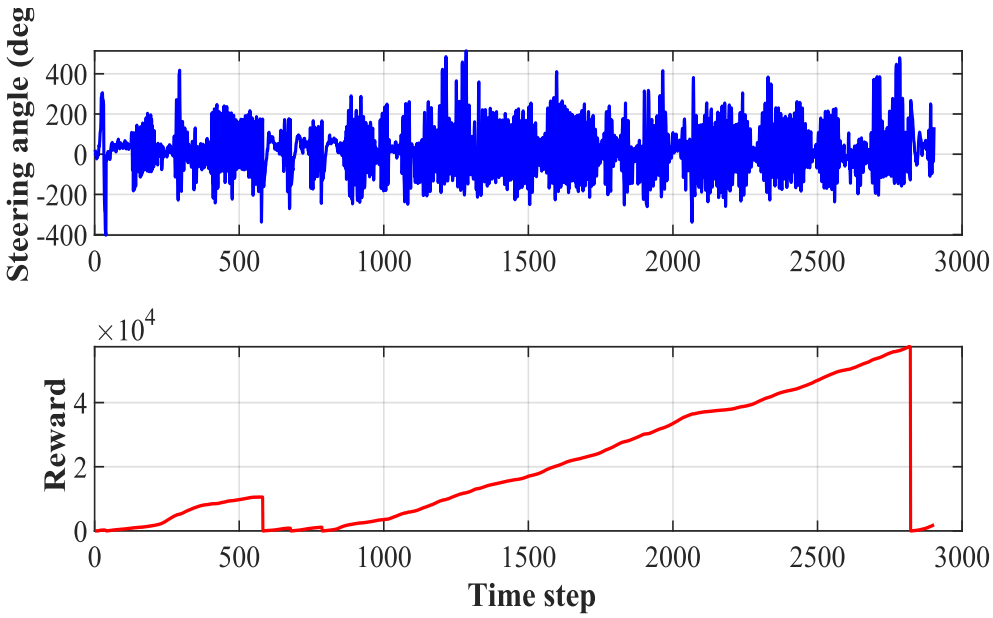


Figure 5.12: Steering angle and rewards obtained during training

From the training plot, it can be seen that the steering angle fluctuates significantly throughout the training process, indicating the car's attempts to learn how to steer effectively in various situations. There are periods of high volatility and rapid changes in steering angle, suggesting that the car is exploring different steering actions to find an optimal policy. From the reward plot, it can be observed that the reward starts from zero and increases over time. The reward increases steadily, with occasional drops and plateaus. The drops indicate situations where the car made poor decisions leading to lower rewards, and plateaus representing periods of learning

where the car is exploring different actions without significant improvement. By training the model for longer time steps, the system is able to learn better to make decisions to keep the lane and make suitable turns wherever demanded by the road conditions.

In this chapter, we have explored two prominent machine learning approaches for reference generation in autonomous driving vehicles: the end-to-end learning method and the Deep Deterministic Policy Gradient (DDPG) reinforcement learning-based method. Both methods offer compelling advantages over rule-based modular architectures for autonomous driving. End-to-end learning directly mapped raw inputs to commands, simplifying integration and leveraging neural networks for adaptive feature extraction. DDPG RL exceled in learning complex policies through continuous interaction, balancing exploration, and exploitation for precise decision-making. In contrast, rule-based systems struggle with real-world variability, while machine learning methods adapt and improve with diverse experiences, crucial for navigating dynamic driving environments.

While the results obtained from both methods are satisfactory, there remains substantial room for improvement. Future research should focus on training with a wider dataset including the Indian road scenarios and more diverse driving environments to enhance generalization and robustness. Developing more complex models can capture intricate driving patterns and interactions, while the inclusion of traffic elements in the training set can provide realistic scenarios for better decision-making. Additionally, combining these methods with advanced sensor fusion techniques and incorporating feedback from real-world deployments will further refine their performance and reliability.

In the next chapter, we will discuss the hardware and software design of the project, particularly the circuit design of different modules and the software implementation of the algorithms.

Chapter 6

Hardware and Software Design

In the previous chapters, we have looked at different software algorithms to achieve autonomous features on an EV. These chapters have covered a detailed discussion on perception, localization, path planning, reference generation, and control. In this chapter, we will discuss the hardware implementation aspects of the system as well as the software design of the algorithms previously discussed. The first four sections of the chapter cover the hardware design and the final section covers the software design aspects. In the hardware design, mainly the discussion focuses on the circuit design and the interface of different sensors to the Intelligent Mobility Module (IMM) and the software section discusses the implementation of different algorithms on the Jetson Orin Nano controller.

6.1 Hardware Design

The Intelligent Mobility Module (IMM) acts as the brain of the autonomous car. It's a powerful computer responsible for high-level decision-making, processing sensor data, and controlling the vehicle's movement. The IMM so designed has to be compatible with a wide range of sensors, including potentiometers, cameras, and speed sensors. It should provide flexible interfaces for seamless integration, modular for easy expansion, accommodating additional sensors and peripheral boards without significant redesign. It also has to be reliable to ensure the safety of the vehicle. Additionally, the module also has to execute algorithms with low latency and in a power-efficient manner.

The different hardware requirements of the Intelligent Mobility Module are tabulated in table 6.1 and the external interface of IMM with other subsystems is shown in Fig. 6.1.

Requirement	Input	Output
Controller	Sensors output	Speed reference / PWM
ADC	Sensors	Digital data to the controller
DAC	Speed reference	Speed Control Module
DC motor control	PWM	DC motor
User interface	Passenger	Information to the controller
Power distribution network	48 V battery	15V, 5 V and 12 V

Table 6.1: Hardware requirements of the IMM

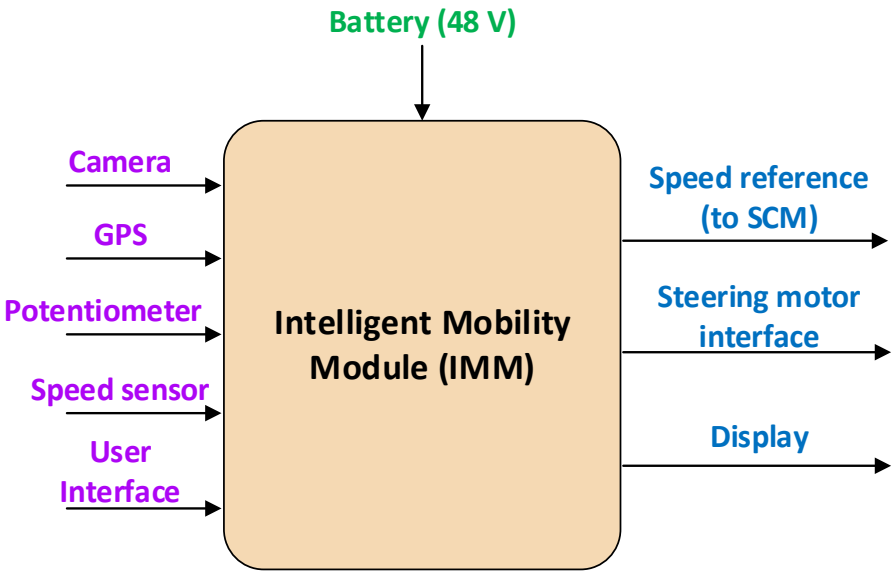


Figure 6.1: External interface of IMM

The brain of the IMM is the controller that implements all the software algorithms discussed in the previous chapters. Different sensors like camera, GPS, speed sensor, and potentiometer are interfaced with the controller. Based on these sensors, the controller runs the algorithms to provide a speed reference and a PWM. The speed sensor is connected to a digital-to-analog converter (DAC) to provide the analog speed reference signal to the Speed Control Module (SCM). As discussed in Chapter 2, based on the study conducted, Jetson Orin Nano was selected as the controller module. Some of the sensors like the speed sensor and potentiometer give an analog output. So to interface this with the controller, an analog-to-digital converter (ADC) is used. Now, the PWM generated from the controller is used for controlling the position of a steering DC motor. To achieve this, a DC motor control circuit consisting of an H bridge and its gate driver is required. This circuit is responsible for converting the PWM signal

into a high-voltage alternating pulse to operate the motor. Next, the IMM should also provide an interface to the user to get commands and display information to the user. Finally, the module should also have onboard circuits to generate the necessary power supply levels to power the above said circuits. The board needs to have two kinds of supplies - one for powering the controller and its associated circuits and another for powering the DC motor and its associated circuits. The input to these two supplies comes from two different batteries. This input to the power distribution network (PDN) is 48 V from two Li-Ion batteries and its outputs are 15V and 5V for powering the controller circuits and 12V for powering the motor control circuits.

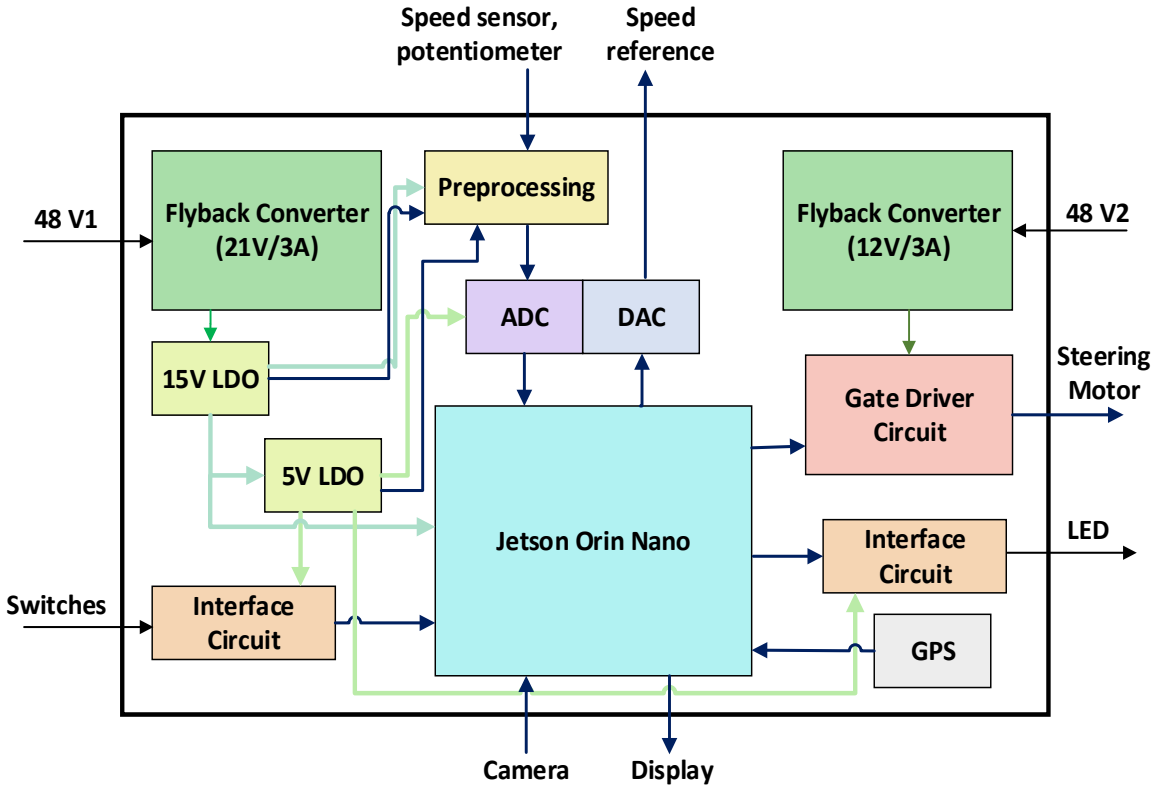


Figure 6.2: Block diagram of IMM

The complete block diagram representation of the IMM is shown in Fig. 6.2. The Jetson Orin Nano forms the main component of the board. It is directly interfaced with a camera and a display unit that are mounted at suitable locations on the vehicle. The camera provides video of the road in front of the vehicle and the display provides a GUI for user interface which consists of a provision for user input and displaying the perception outputs. Other than the camera, the module also has a GPS for localization. The GPS is connected directly to the Jetson through a USB interface. The module also has provision for user interface through switches and LEDs. The switch provides an option for emergency actions and the LED is used

to display any warnings to the user. Both the switches and LEDs are connected to the Jetson through an interface circuit.

The module has provision for converting analog signals to digital and digital signals from the Jetson to analog. The analog signals coming from outside as well as from within the board are first preprocessed to filter noise and clamp the voltage. Output of the preprocessing stage is connected to the ADC and the ADC output is given to the Jetson. The digital output signal from the Jetson i.e., the speed reference is converted to analog using a DAC and is interfaced to the Speed Control Module.

As mentioned earlier, a power distribution network is needed to deliver adequate power levels to the circuits in the board. The controller side circuits need 15 V and 5 V, whereas the motor control side needs 12 V. The input to the PDN is from two 48 V Li-Ion batteries. Since these batteries are used by other subsystems in the vehicle, an electric isolation with the battery is crucial to avoid ground bounce or EMI issues on the module. So to provide the electric isolation, a flyback converter is used at the front end of the PDN. The first flyback converter is designed to produce an output of 21 V at a maximum current rating of 3 A and the second converter produces an output of 12 V at a maximum current rating of 3 A. The 12 V output of the second converter is connected directly to the gate driver circuit. However, since the controller side demands a very stable, ripple free output, the 21 V output of the converter is connected to a 15 V linear regulator. This regulator output is used to power the Jetson, and circuits in the preprocessing stage. In addition to that, a 5 V is also generated from the 15 V using another linear regulator to power ADC, DAC, interface circuits, etc.

In the upcoming sections, design of the circuits of each of these requirements is discussed. Since the Jetson Orin Nano comes as a standalone developer kit, it is not discussed here.

6.1.1 ADC and DAC Interface

An ADC is required to convert the analog signals to digital and interface them to the controller. The parameters to be converted by the ADC include vehicle speed, steering angle, steering motor currents, and PDN voltage output. The vehicle speed is obtained from a Hall sensor connected to the propulsion motor. The Hall sensor provides the propulsion motor speed that needs to be converted to the vehicle speed. The steering angle obtained from the potentiometer connected to the steering column of the vehicle is used for the closed-loop implementation of the steering control. Steering motor currents and the power distribution network output currents are monitored for any failure detection and indication to the user for safety aspects.

When an ADC is selected for this application, it needs to have the requirements of good resolution, a higher sampling rate, multichannel provision, and a suitable interface protocol to connect to the Jetson and also it should meet the voltage levels of the input signals and the power supply voltage generated by the PDN circuits. There are several ADC modules available in the market that meet these requirements. One such module is the High Precision AD/DA board by Waveshare shown in Fig. 6.3.

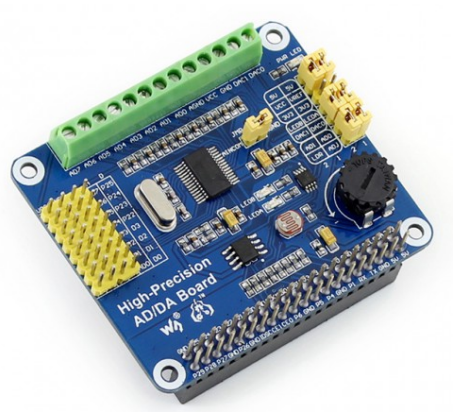


Figure 6.3: Waveshare High Precision AD/DA board

Waveshare High Precision AD/DA board comes with an 8-channel ADC and 2-channel DAC [70]. The board has onboard ADS1256, an 8-channel 24-bit high-precision ADC, and DAC8552, a 2-channel 16-bit high-precision DAC making it highly suitable for usage in the IMM. The board communicates to the controller through an SPI interface that is also present in the Jetson board making the interface adaptable. The board is powered from a 5 V which is generated by the PDN circuit. Since the Waveshare module comes with a standalone board, it is mounted on the IMM PCB next to the Jetson module and connected through an external flat ribbon cable.

Before interfacing external sensor signals with the ADC, filtering, and voltage clamping are necessary to ensure accurate and reliable measurements. Filtering removes noise and prevents aliasing, enhancing signal quality and reducing errors. Voltage clamping protects against overvoltage conditions, safeguarding both the sensor and the ADC from damage while ensuring compatibility with the ADC's input range. These preprocessing steps are critical for improving the quality, reliability, and safety of the digitized signals.

In the first step of the preprocessing stage, a voltage divider is implemented to scale the analog voltages to within the range of the ADC. For instance, the 15 V output from the voltage regulator is beyond the input range of the ADC (0 - 5V). Hence, it has to be attenuated to be within 5V. In the next step, a Sallen key second-order low pass filter is used to remove high-frequency noise in the signal and to band limit the signal to meet the Nyquist rate. The sampling rate of

the ADC is at 1 kHz. Hence the signal has to be bandlimited at a frequency less than 500 Hz. With a margin of 2.5 times, the cutoff frequency of the filter is chosen as 200 Hz. Now, in the third step, the output voltage of the preprocessing state has to be clamped to 4.7 V on the upper side and 0 V on the lower side to ensure that even under the failure of the preprocessing circuit, the ADC input range is always met.

The preprocessing circuit is shown in Fig. 6.4. In the circuit, resistors R_1 and R_2 form the voltage divider to attenuate the sensor output to within the input range of the ADC. The voltage divider output is given by the equation 6.1. Since the sensor output range for each of the sensors is different, the values of resistors R_1 and R_2 are also different according to the sensor. These values for each of the inputs are tabulated in table 6.2.

$$V_{\text{div}} = V_{\text{in}} \frac{R_2}{R_1 + R_2} \tag{6.1}$$

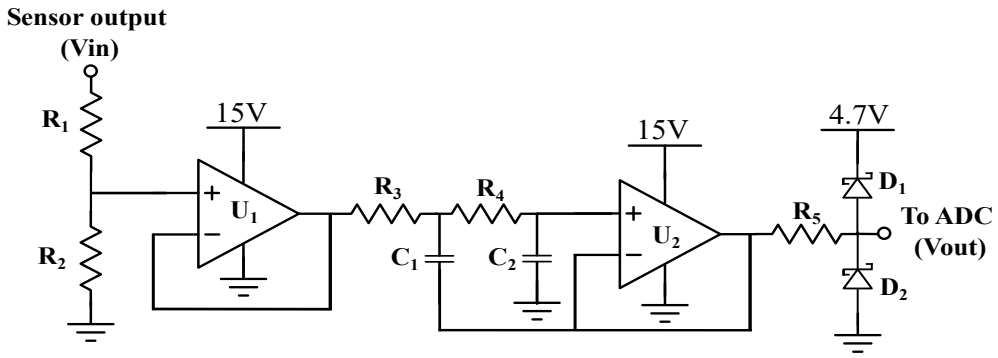


Figure 6.4: ADC preprocessing circuit

Signal	R_1 (KΩ)	R_2 (KΩ)	Output range (V)
Potentiometer	1	2	0 - 3.33
Speed sensor	1	2	0 - 3.33
15 V	1	3.3	3.5
5 V	1	2	2.5

Table 6.2: Voltage divider circuit values

After the voltage divider, the second order Sallen key low pass filter is implemented using Opamps U_1 and U_2 that are powered by 15 V from the PDN. The resistors R_3 and R_4 and capacitors C_1 and C_2 are kept the same value to simplify the computations. The cutoff frequency

of the filter is given by the equation 6.2. Considering a cutoff frequency of 200 Hz, the resistors R_3 and R_4 are selected as $8.2K\Omega$ and capacitors C_1 and C_2 as $100\ nF$.

$$f_{3\text{dB}} = \frac{1}{2\pi R_3 C_1} \tag{6.2}$$

At the output of the preprocessing stage, there are zener diodes D_1 and D_2 that are connected to 4.7 V and the ground respectively to clamp the output for safety. The output of this preprocessing stage is connected to the Waveshare ADC module.

DAC interface is easier compared to the ADC. The digital value of the speed reference signal is given directly to the Waveshare module through SPI without any intermediate circuit. And the DAC output is connected to the Speed Control Module. The clock and data signals generated from the Jetson during the SPI communication to the Waveshare module are shown in Fig. 6.5.

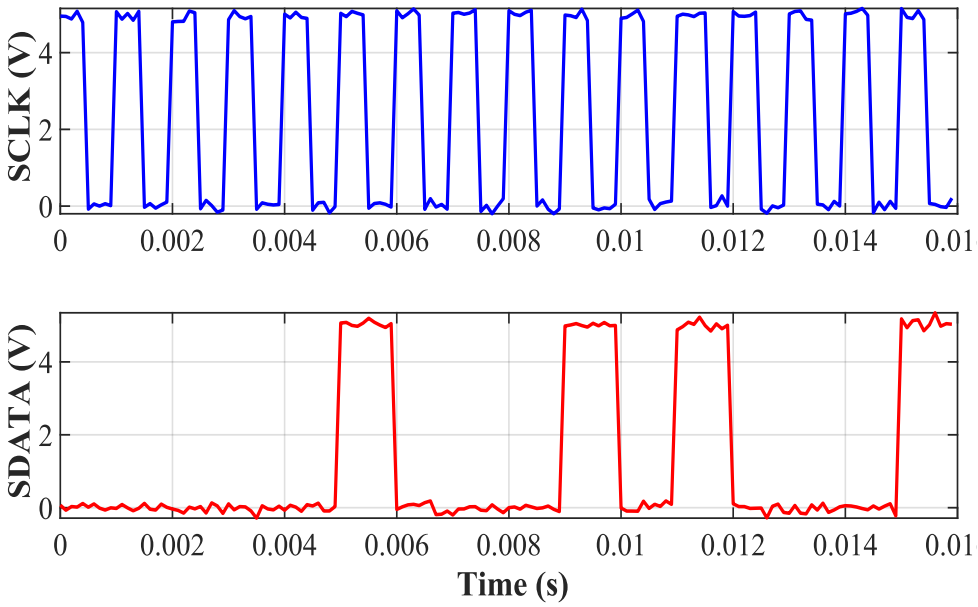


Figure 6.5: Clock and data signals of a sample SPI communication

6.1.2 DC Motor Control

For controlling the direction of the vehicle, a power steering module of the Maruti Alto car is used which comes with a DC motor operating at 12V. This DC motor drives the steering column attached to the vehicle wheel. The DC motor control block is an interface between the controller and the motor. This block should translate the 5V signal from the Jetson to the higher power levels required to operate the motor effectively. So input to this block is a PWM

signal from the Jetson and the output is a high power signal to the motor. The control loop in the Jetson is implemented such that the motor stays at rest when the PWM duty ratio is 50%, turns clockwise when it is greater than 50%, and turns anticlockwise when the PWM is less than 50%.

To control the DC motor, a module with a built-in gate driver and H-bridge (Fig. 6.6) is required. It ensures seamless interfacing between the control signals and the high-power motor, preventing damage and enhancing performance. This module is expected to have high-current handling, fast switching capabilities, robust thermal management, and protection features against overcurrent and short circuits. There are several such modules available in the market at different form factors and meeting the requirements of the project. One such module is the MD10C module of Cytron as shown in Fig. 6.7. Technical details of this module are tabulated in table 6.3 [71].

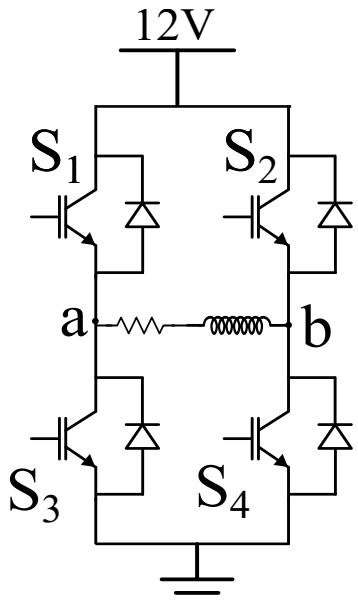


Figure 6.6: H bridge for motor control

Parameter	Value	Unit
Input voltage	30	V
Continuous current	13	A
Peak current	30	A
PWM frequency	20	kHz

Table 6.3: Absolute maximum rating of Cytron MD10C

Based on the absolute maximum rating of the module, it is operated at a voltage of 12 V, a

Figure 6.7: Cytron MD10C

maximum motor current of 6 A, and at a PWM frequency of 10 kHz. Input to the module is PWM at 5 V level and the output is connected to the DC motor at points *a* and *b* in Fig. 6.6. A sample PWM generated from the Jetson is shown in Fig. 6.8.

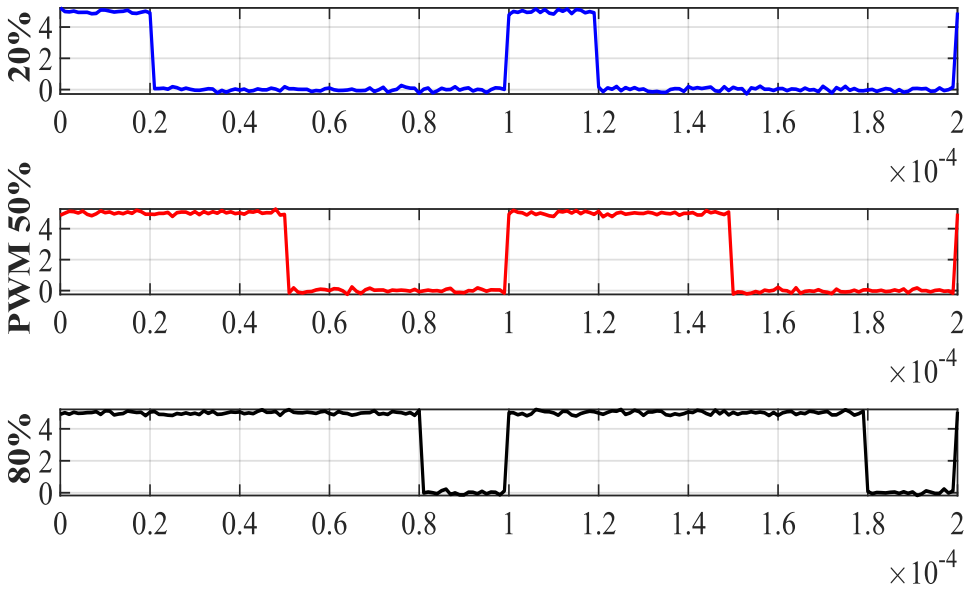


Figure 6.8: Different duty ratio PWMs generated (20%, 50% & 80%)

6.1.3 User Interface

The IMM should have an interface with the user for taking inputs and showing alarms/warnings to the user. The input to the IMM from the user can be either through the GUI in the display connected directly to the Jetson module or through external switches. The user primarily depends on the GUI for communicating with the IMM. However, in case of an emergency, the

user can also feed in important actions like stop, slow down, mode change, etc. Similarly, the output to the user is also mainly through the GUI on the display unit. This output consists of perception output, emergency alarms, and information like vehicle speed or current mode of operation. However, in addition to the display unit, the IMM also raises emergency indications to the user through an LED or alarm placed on the dashboard of the vehicle. This information is very crucial to take necessary action by the user for safety concerns.

The display unit can either be connected to the Jetson directly through the USB interface or it can be interfaced to a laptop either through a wired USB or ethernet connection or a wireless WiFi connection. However, interfacing switches, LEDs, and alarms require an analog circuit. The circuit to interface the switch is shown in Fig. 6.9.

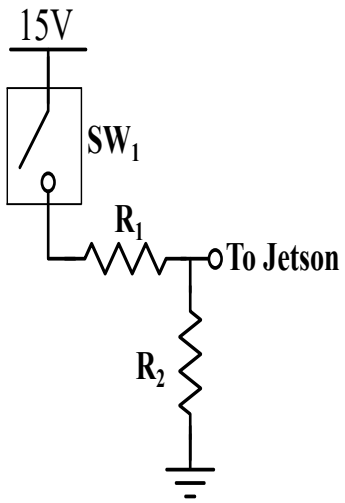


Figure 6.9: Switch interface to the IMM for emergency input

In the figure, the Single Pole Single Throw (SPST) switch SW_1 is placed on the dashboard of the vehicle. One end of the switch is connected to 15 V and the other end to a voltage divider circuit. The middle point of the voltage divider circuit is connected to a digital GPIO of the Jetson. In the normal case when the switch is not pressed, the switch is in open condition and the GND signal gets connected to the Jetson. On the other hand, when the switch is pressed, 15V is connected to the voltage divider circuit which is then connected to the Jetson. The resistors R_1 and R_2 are selected such that when the switch is pressed, Jetson GPIO gets 5 V. So for that R_1 is selected as 1KΩ and R_2 as 2KΩ.

The interface of the alarm or LED is straightforward compared to that of the switch. A series resistor is put at the GPIO output of Jetson and the resistor's other end is connected to the LED or the alarm as shown in Fig. 6.9. The resistor is selected to limit the current to the LED/ alarm.

Considering a current of 100 mA to flow through the LED, the resistance of R_1 is selected as 50Ω.

Figure 6.10: LED/ alarm interface to the IMM for emergency output

6.1.4 Power Distribution Network

The Power Distribution Network (PDN) in the IMM ensures stable and efficient delivery of power to all components, minimizing noise and voltage fluctuations. It is crucial for maintaining signal integrity and overall system performance. As mentioned earlier, there are two types of power distribution networks in the module - PDN 1 and PDN 2. The PDN 1 generates the required power levels for the controller and the associated circuits including 15 V and 5 V. PDN 2 is responsible for generating 12 V to the motor control circuit. Input to the PDN 1 is from the first set of 48 V battery and to PDN 2 from the second set of 48 V battery. Since these batteries are also used to power other subsystems in the vehicle, the front end of both PDNs has an electric isolation achieved through a transformer to reduce the ground bounce effects and EMI. This transformer is part of a flyback converter which converts the input 48 V to a lower output. In the case of PDN 1, this output voltage is 21 V and for PDN 2, it is 12 V. The 12 V from PDN 2 is used directly to power the motor control circuit. On the other hand, since the controller and the associated circuits need a highly stable, low noise/ ripple supply, a linear regulator is further used to convert the 21 V to 15 V and 5 V levels. In the next section, the flyback converter designs of both PDN 1 and PDN 2 are discussed and in the subsequent section, voltage regulator design for getting 15 V and 5 V are discussed.

6.1.4.1 Flyback Converter Design

The flyback converter is a type of DC-DC converter that uses a transformer to store energy when the input voltage is applied and releases it to the output when the input is turned off. It is widely used for its simplicity, efficiency, and ability to provide electrical isolation between input and output. Even though both PDN 1 and PDN 2 flyback converter output voltages are different, the design procedure is the same. Hence the design steps for only the PDN 1 transformer are discussed here.

Firstly, the output power level required for each of the flyback converters is discussed. On the controller side, the Jetson is expected to consume up to 1 A at its peak operation. In addition to that, the ADC preprocessing circuits and other interface circuits can consume up to 0.5 A. Now, with a margin of 2 times, the flyback converter for PDN 1 is designed to give up to 3 A at an output of 21 V. On the motor control side, the DC motor usually operates with a current of less than 2.5 A. An additional margin of 0.5 A is also given whereby the PDN 2 flyback converter is designed to give up to 3 A at an output of 12 V.

A standard flyback converter circuit is shown in Fig. 6.11. A detailed explanation of the flyback converter is provided in [72]. Hence, in this section, only the relevant equations for determining the component values are mentioned for brevity.

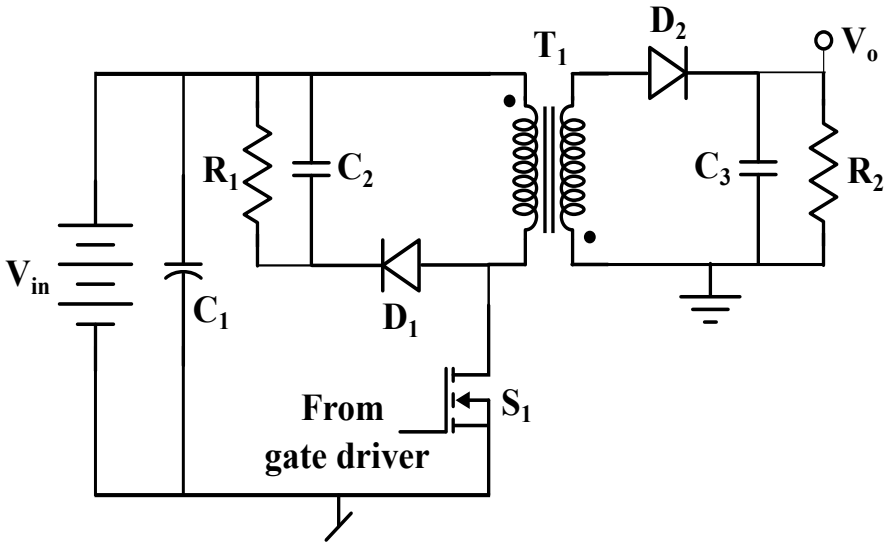


Figure 6.11: Flyback converter circuit

In the figure, V_{in} is the input to the converter from the battery which is 48 V. At the front end of the converter, there is an electrolytic bus capacitor C_1 of value 100 μF to reduce the ripple in the input voltage and also to provide necessary charge levels during the operation. The flyback converter consists of a primary side and a secondary side separated by a transformer T_1. The transformer is used to store the energy and transfer it to the secondary at an interval decided by the switch using a MOSFET S_1. The transformer also provides a galvanic isolation between the primary and the secondary. When the MOSFET is ON for a certain duration of a cycle, the primary current flows through the transformer windings, and the transformer stores the energy. In the remaining duration of the cycle, the transformer is OFF and the stored energy is transferred to the secondary side. On the secondary side, there is a diode D_2 which acts as a rectifier followed by a capacitor C_3 and a bleeder resistor R_2 to reduce the output voltage ripple

and to provide a path for discharge respectively. On the primary side, there is an RCD snubber formed by R_1, C_2 and D_1 to dampen the noise produced by the switching signal that interacts with the leakage inductance of the transformer.

The notations used for the analysis of the flyback converter are tabulated in table 6.4.

Parameter	Notation
Input Voltage	V_{in}
Output Voltage	V_o
Input Current	I_{in}
Output Current	I_o
Switching frequency	f_s
Transformer turns ration	n
Max primary current	I_{p+}
Min primary current	I_{p-}
Max secondary current	I_{s+}
Min secondary current	I_{s-}
Transformer primary inductance	L_p
Transformer secondary inductance	L_s
Transformer primary energy	E_L

Table 6.4: Notations of parameters

The turns ratio of the transformer and the duty ratio of the MOSFET turn ON are determined by the equation 6.3.

$$V_0 = nV_{\text{in}}(\frac{d}{1-d}) \tag{6.3}$$

For an input voltage of 48 V and a desired output voltage of 21 V, the duty ratio d is chosen as 0.4, which gives a turns ratio n of 0.65. Now the switching frequency of the MOSFET f_s is selected as 125 kHz. The maximum and minimum values of the secondary current ripple are given by equations 6.4 and 6.5.

$$I_{\text{s+}} = \frac{I_{\text{o}}}{1-d} + \frac{V_{\text{o}}(1-d)}{2L_{\text{s}}f_{\text{s}}} \tag{6.4}$$

$$I_{\text{s-}} = \frac{I_{\text{o}}}{1-d} - \frac{V_{\text{o}}(1-d)}{2L_{\text{s}}f_{\text{s}}} \tag{6.5}$$

For a desired output current of 3 A, from these equations, the maximum and minimum values of the secondary currents are 5.5 A and 4.5 A respectively. Similar to the previous equations, the maximum and minimum values of the primary current ripple are given by equations 6.6 and 6.7.

$$I_{\text{p+}} = \frac{I_{\text{in}}}{d} + \frac{V_{\text{in}} d}{2 L_{\text{p}} f_{\text{s}}} \tag{6.6}$$

$$I_{\text{p-}} = \frac{I_{\text{in}}}{d} - \frac{V_{\text{in}} d}{2 L_{\text{p}} f_{\text{s}}} \tag{6.7}$$

From these two equations, the maximum and minimum values of the primary currents are 5.16 A and 4.22 A respectively and the primary side inductance of the transformer is 0.15 mH. Using these obtained values, the MOSFET and the secondary diode ratings are decided. The MOSFET blocking voltage, maximum current, and rms current requirements are given by equations 6.8 - 6.10. From the equations, the selected MOSFET should have a voltage-blocking capability of at least 81 V, a maximum current capability of 5.16 A, and an RMS current rating of 2.95 A. Considering these parameters, a Vishay make MOSFET IRF840 is selected. IRF840 has a blocking voltage of 500 V, a maximum pulsed current capability of 32A, and a continuous current rating of 8A which makes it suitable for the application.

$$V_{\text{block}} = \frac{V_{\text{o}}}{n} + V_{\text{in}} \tag{6.8}$$

$$I_{\text{max}} = I_{\text{p+}} \tag{6.9}$$

$$I_{\text{rms}} = \left(\frac{I_{\text{p+}} + I_{\text{p-}}}{2}\right)\sqrt{d} \tag{6.10}$$

The secondary diode D_2, peak inverse voltage (PIV) and maximum current ratings are given by the equations 6.11 and 6.12. From the equations, the selected diode should have a minimum PIV of 40 V and a maximum current capability of 5.5 A. Based on this, ROHM semiconductor make RB088NS150FH power diode is selected. This diode has a PIV of 150 A and a maximum current capability of 10 A.

$$V_{\text{PIV}} = n V_{\text{in}} + V_{\text{o}} \tag{6.11}$$

$$I_{max} = I_{s+} \tag{6.12}$$

The capacitor on the secondary, C_3 is chosen as a parallel combination of an electrolytic capacitor of value 47 μF and 100 nF to reduce the output voltage ripple. Also, the bleeder resistor has a value of 11 KΩ. The RCD snubber design is based on [73] and the values are obtained as R_1 of 230 Ω of 10W rating, C_2 as 300 nF and RB088NS150FH as the power diode in the snubber.

The design of all the components of the flyback converter has been discussed so far. Now the design of the energy-storing element, the transformer needs to be discussed. The design of the transformer is based on the window area method. An EE core is used for winding the transformer. The notations used for the design of the transformer are tabulated in table 6.5.

Parameter	Notation
Transformer primary energy	E_L
Window utilization factor	K_w
Current density	J
Crest factor	K_c
Window area	A_w
Area product	A_p
Flux density	B_m

Table 6.5: Notations of parameters

Energy to be stored in the transformer is given by equation 6.13 and the area product of the transformer is given by equation 6.14. On substituting the values, we obtain the desired area product of the transformer EE core as 1.6 cm^4. Considering different cores from the brochure, the E42/21/20 core is selected which has an area product of 6 cm^4.

$$E_L = \frac{1}{2} L_p {I_{p+}}^2 \tag{6.13}$$

$$A_p = \frac{2E_L}{K_w K_c J B_m} \tag{6.14}$$

Now based on other equations mentioned in [72] (not discussed here for brevity), the number of turns in the primary is obtained as 17 with a wire gauge of 18 SWG and the secondary having 13 turns at 17 SWG. The air gap required in the design is 0.57 mm. This completes the design

of all the components in the flyback converter. The design values are tabulated in table 6.6 for both the PDN 1 and PDN 2 flyback converters.

Component	PDN 1	PDN 2	Unit
V_{in}	48	48	V
V_o	21	12	V
I_o	3	3	A
C_1	100	100	μF
C_2	300	300	nF
C_3	47	47	μF
R_1	230	230	Ω
R_2	11	11	KΩ
N_p	17	17	
N_{pSWG}	18	18	SWG
N_s	13	8	
N_{pSWG}	17	16	SWG
L_p	0.15	0.15	mH
Airgap	0.57	0.56	mm
Core	E42/21/20	E42/21/20	

Table 6.6: Flyback converter circuit values

So far in the section, we have discussed the design of all the components of the flyback converter. Now the design of the gate driver circuit to the MOSFET S_1 of the flyback circuit is discussed. The gate driver circuit is required to generate pulses at a sufficient power level to turn ON/OFF the power MOSFET at the desired duty ratio and switching frequency. An IC TL494 is used to achieve this. The TL494 IC is a versatile PWM controller used in various power supply and DC-DC converter designs. It integrates key functions like error amplifiers, an oscillator, and a PWM comparator, making it a popular choice for efficient power management [74]. TL494 circuit with suitable passive components is shown in Fig. 6.12.

At the front end of the TL494, it has an error amplifier. The input voltage to the error amplifier determines the duty cycle of the output square wave. Voltage to the error amplifier is derived from a reference voltage V_{ref} of 5 V provided by the IC. The error amplifier input is obtained from the equation 6.15.

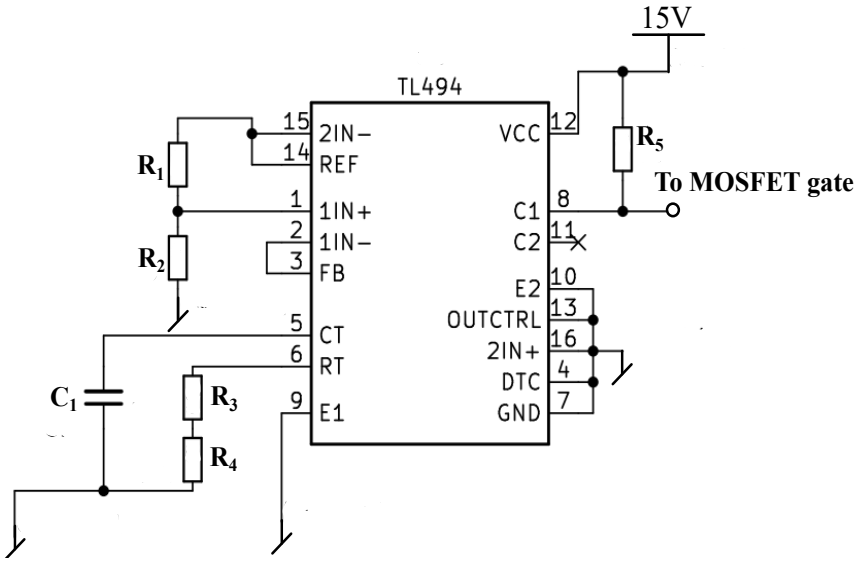


Figure 6.12: MOSFET gate driver circuit using TL494

$$V_{\text{err}} = \frac{V_{\text{ref}} R_2}{R_1 + R_2} \tag{6.15}$$

For a duty ratio of 0.4, i.e., 2 V (2 V/ 5 V gives 0.4), resistors R_1 and R_2 are selected as 1.5 KΩ and 1 KΩ respectively. Now the frequency of the output square pulse is determined by the series combination of resistors R_3 and R_4 and the capacitor C_1 as given in the equation 6.16.

$$f_s = \frac{1}{(R_3 + R_4) C_1} \tag{6.16}$$

For a switching frequency of 125 kHz, R_3 is selected as 8.2 KΩ, R_4 as 0 Ω and C_1 as 1 nF. Finally, R_5 forms a pull-up resistor at the output stage of the IC. This resistor is selected as 200Ω to provide a maximum gate current to the MOSFET of 300 mA. The gate pulse generated by the TL494 at pin no 8 of the IC is connected to the flyback converter MOSFET. The gate pulses generated by the TL494 and the MOSFET drain-source voltage are shown in Fig. 6.13. The output voltage of both PDN 1 and PDN 2 flyback converters for varying input voltages is tabulated in table 6.7 (load current fixed at 0.5 A) and the output voltages for varying load conditions in table 6.8 (input voltage fixed at 48 V).

Using the table data, line regulation and load regulation of both the converters are obtained using equations 6.17 and 6.18 respectively and mentioned in table 6.9. It can be seen that the line and load regulations of the converter are quite high. This is attributed to the open-loop implementation of the flyback converter. However, this doesn't affect the performance of the

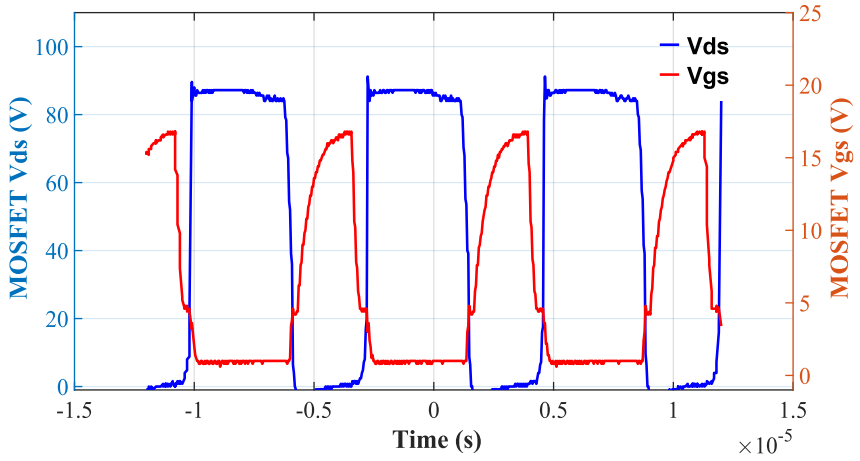


Figure 6.13: Flyback converter MOSFET gate-source and drain-source voltages

Input voltage (V)	PDN 1 flyback output (V)	PDN 2 flyback output (V)
52	23	14
50	22.1	12.5
48	21.2	12.1
46	20.5	11.8
44	19.6	11.2

Table 6.7: Flyback converters output voltage for different input voltages

system, as there is a 15 V linear regulator at the PDN 1 flyback converter output powering the controller and operating in a closed loop to provide a more accurate output voltage. The PDN 2 flyback converter also doesn't demand a very tight regulation parameters. In the future design, closed-loop feature can be implemented in the flyback converter for better output voltage control.

$$line_{\text{reg}} = \frac{V_{\text{o(52V)}} - V_{\text{o(44V)}}}{V_{\text{o(48V)}}} \tag{6.17}$$

$$load_{\text{reg}} = \frac{V_{\text{o(0A)}} - V_{\text{o(3A)}}}{V_{\text{o(0.5A)}}} \tag{6.18}$$

This completes the design of the flyback converter.

Output current (A)	PDN 1 flyback output (V)	PDN 2 flyback output (V)
0	23.6	14.9
0.2	22.1	12.5
0.5	21.2	12.1
1	20.1	11.9
2	19.1	11.4
3	18.3	11.01

Table 6.8: Flyback converters output voltage for different load currents

Parameter	PDN 1 flyback (%)	PDN 2 flyback (%)
Line regulation	16	23
Load regulation	25	32

Table 6.9: Flyback converters line and load regulation parameters

6.1.4.2 Voltage Regulator Design

As discussed earlier, the output of the switching flyback converter has higher ripple and noise content. In addition to that, since the converter is not operating in a closed loop, variation in the output voltage w.r.t change in the input voltage and load conditions is high. So this converter output is not good enough to power a voltage-sensitive device like Jetson Orin Nano. Hence, a linear voltage regulator is put at the output of the flyback converter to produce 15 V output, which is used to power Jetson and other associated circuits. In addition to that, to power ADC/ DAC and user interface circuits, a 5 V is also required. This 5 V is generated using another linear regulator connected to the output of the 15 V regulator.

The current requirement from the 15 V regulator is around 1.2 A. This includes 1 A requirement by the Jetson and another 0.2 A by preprocessing and other circuits. With a margin of another 0.8 A, the 15 V linear regulator should be capable of providing at least 2 A current. To meet this requirement, the LM338 linear voltage regulator of Texas Instruments is selected. This voltage regulator comes as an adjustable 3-terminal device, is exceptionally easy to use, and requires only 2 resistors to set the output voltage. Technical details of the device are tabulated in table 6.10 [75].

The circuit diagram for generating 15 V from an input of 21 V using LM338 is shown in Fig. 6.14.

Parameter	Value	Unit
Input to output voltage differential	40	V
Output current	5	A
Line regulation	0.06	%
Load regulation	1	%

Table 6.10: Technical details of LM338

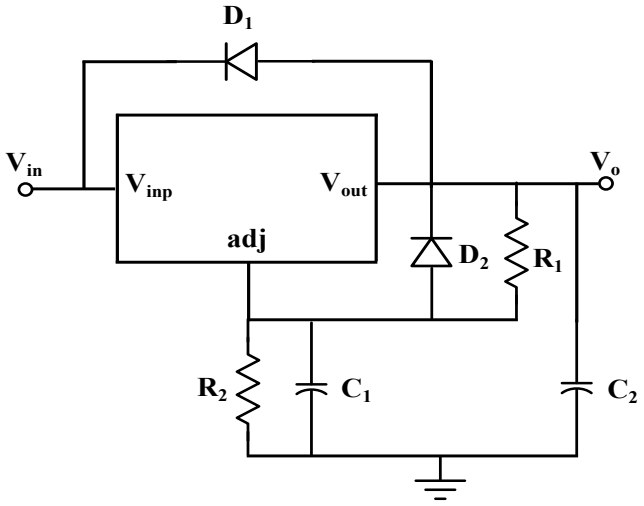


Figure 6.14: Voltage regulator circuit using LM338

In the circuit, the input 21 V generated from the flyback converter is connected to the V_{inp} pin of LM338. The output 15 V is obtained from the V_{out} of the IC. In addition to these two pins, the IC also has an *adj* pin which acts as a negative feedback loop for output voltage control. The output voltage V_o is voltage divided using resistors R_1 and R_2 and connected to the *adj* pin. These resistors are selected based on the equation 6.19, where V_{adj} is the expected voltage at the *adj* pin of the IC to form zero error in the voltage control loop. For a V_{adj} of 1.25 V (obtained from the datasheet) and an output voltage V_o of 15 V, resistors R_1 and R_2 are obtained as 1 KΩ and 11 KΩ respectively. Capacitors C_1 and C_2 in the circuit are provided for a noise and ripple free V_{adj} and V_o respectively. C_1 is selected as 10 μF and C_2 as a parallel combination of a 100 μF electrolytic capacitor and a 100 nF ceramic capacitor. Finally the diodes D_1 and D_2 act as protection diodes to prevent the capacitors from discharging through low current points into the regulator.

$$V_{\mathrm{o}} = V_{\mathrm{adj}}(1 + \frac{R_2}{R_1}) \tag{6.19}$$

The output voltage of the regulator for various input voltages and load conditions is tabulated in table 6.11. From the table, the line regulation and load regulation for the voltage regulator are obtained as 2.7% and 3.2% respectively. These values are very low compared to the flyback converter regulation parameters making it adequate for powering the controller.

Input voltage (V)	Output voltage (V)	Output current (A)	Output voltage (V)
23	15.34	0	15.35
21	15.21	0.5	15.24
19	15.09	1	15.01
17	14.92	2	14.85

Table 6.11: Voltage regulator output values for different conditions

The circuit schematic of the complete IMM module is shown from Fig. 6.15 - 6.16. Layout of both the top and bottom sides of the PCB are also shown in Fig. 6.17 and 6.18 respectively. Finally, the top side and bottom side photos of the realized PCB are shown in Fig. 6.19 and 6.20 respectively.

In the previous sections, we have looked at the design of hardware circuits in the Intelligent Mobility Module in detail. In the coming section, the software design of perception and reference generation algorithms discussed in chapters 3 and 4 respectively will be discussed.

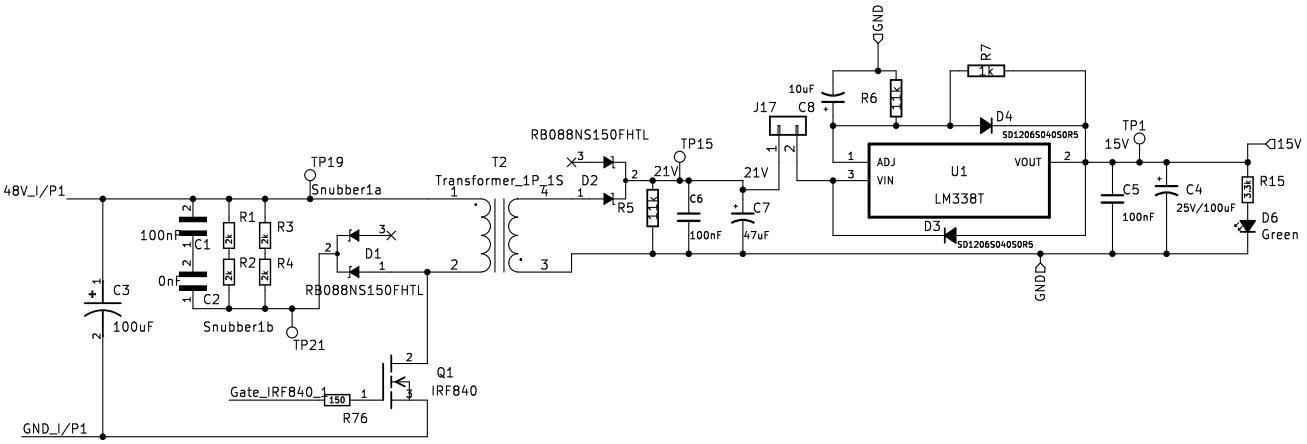
15V
GND
5V
15V
GND
5V
Controller
PWM1
File: Controller.kicad_sch
PWM1

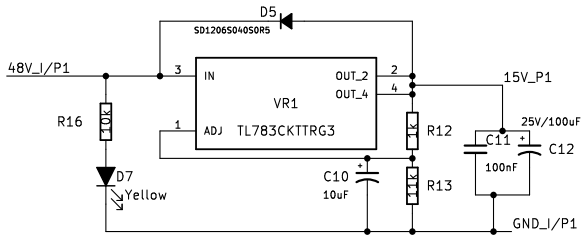
GND
V_motor
GND
V_motor
Power_circuit2
File: Power_circuit2.kicad_sch

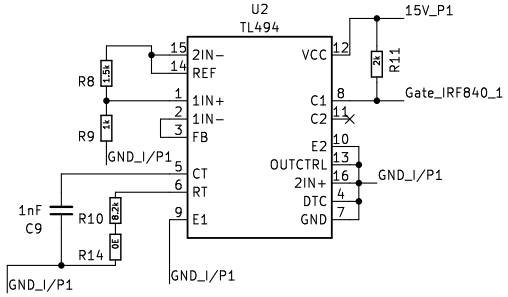
4.7V
15V
5V
GND
4.7V
15V
5V
GND
Sense
Curr2_sense
Curr1_sense
File: Sense.kicad_sch
Curr2_sense
Curr1_sense

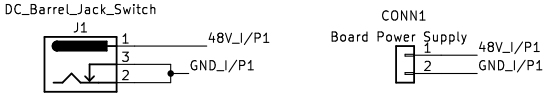
15V
GND
5V
4.7V
15V
GND
5V
4.7V
Power_circuit1
File: Power_circuit.kicad_sch

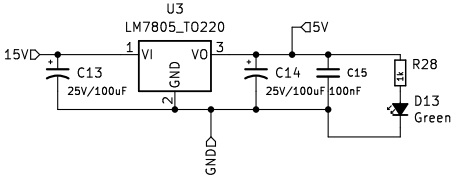
V_motor
15V
GND
5V
4.7V
V_motor
15V
GND
5V
4.7V
Motor_Driver
Curr1_sense
Curr2_sense
PWM1
File: Motor_Driver.kicad_sch
Curr1_sense
Curr2_sense
PWM1

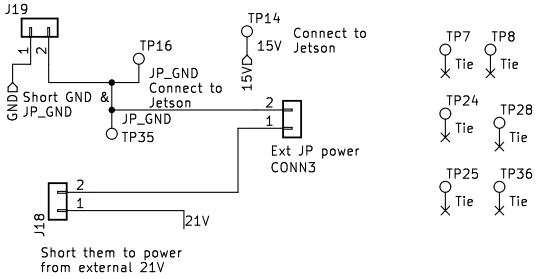

GND
R7
1k
10uF
J17
C8
R6
11k
D4
SD1206S040S0R5
TP1
15V
15V
RB088NS150FHTL
TP15
21V
21V
ADJ
VIN
U1
LM338T
VOUT
C5
100nF
C4
25V/100uF
R15
3.3k
D6
Green
TP19
Snubber1a
T2
Transformer_1P_1S
D2
48V_I/P1
R1
R3
100nF
C1
2k
2k
R2
R4
0nF
C2
2k
2k
D1
RB088NS150FHTL
C6
100nF
C7
47uF
R5
111k
C3
100uF
Snubber1b
TP21
D3
SD1206S040S0R5
GND
Q1
IRF840
Gate_IRF840_1
150
R76
GND_I/P1


D5
SD1206S040S0R5
48V_I/P1
IN
OUT_2
OUT_4
VR1
ADJ
TL783CKTTRG3
15V_P1
R16
10k
R12
1k
25V/100uF
C11
100nF
C12
D7
Yellow
C10
10uF
R13
11k
GND_I/P1

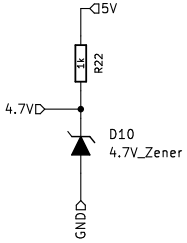

U2
TL494
15V_P1
R8
1.5k
2IN−
REF
VCC
R11
2k
Gate_IRF840_1
1IN+
1IN−
FB
C1
C2
R9
1k
GND_I/P1
E2
OUTCTRL
2IN+
DTC
GND
GND_I/P1
CT
RT
E1
1nF
C9
R10
8.2k
R14
0R
GND_I/P1
GND_I/P1

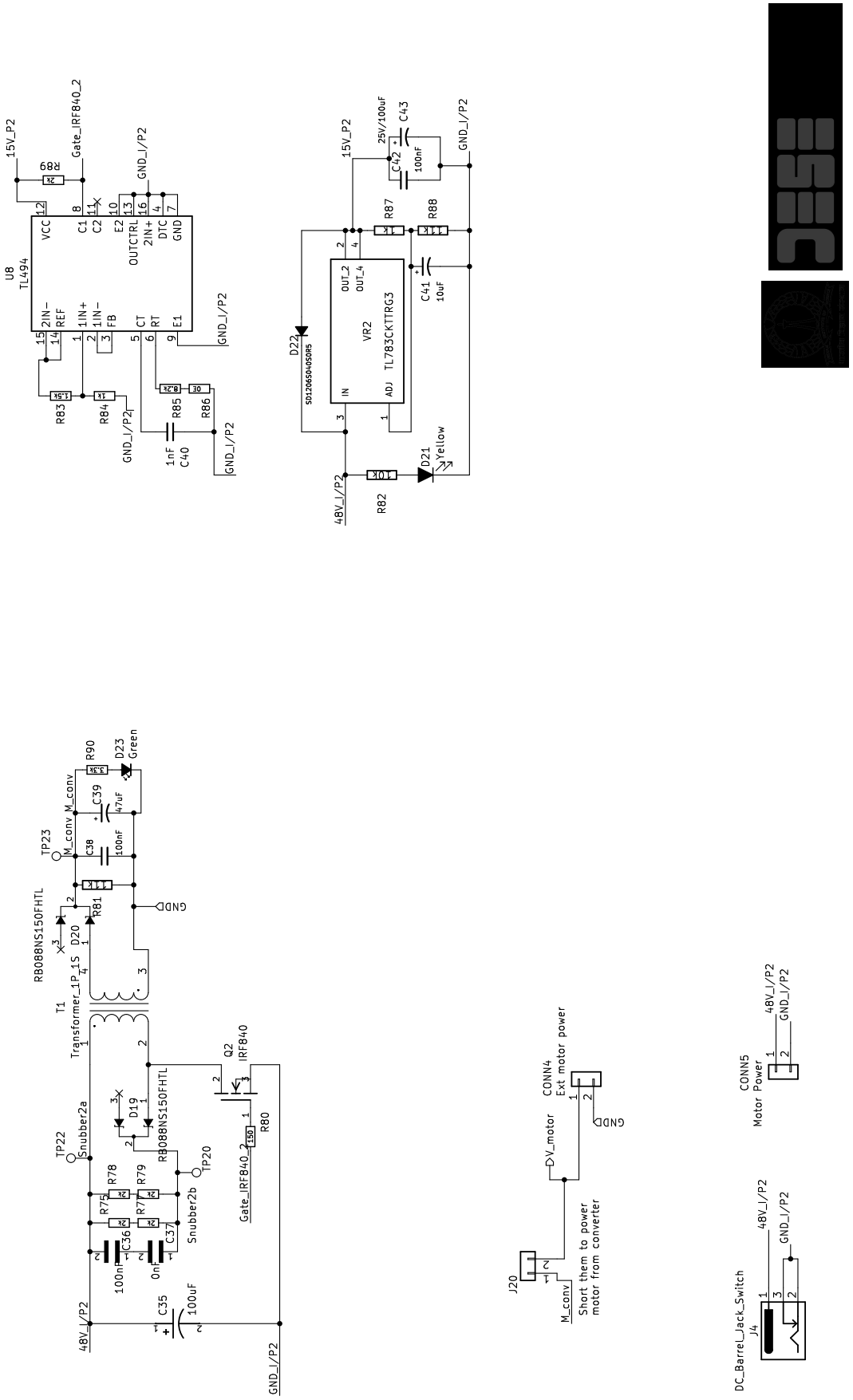

DC_Barrel_Jack_Switch
J1
48V_I/P1
GND_I/P1
CONN1
Board Power Supply
48V_I/P1
GND_I/P1


U3
LM7805_TO220
15VD
VI
VO
GND
5V
C13
25V/100uF
C14
25V/100uF
C15
100nF
R28
1k
D13
Green
GNDD

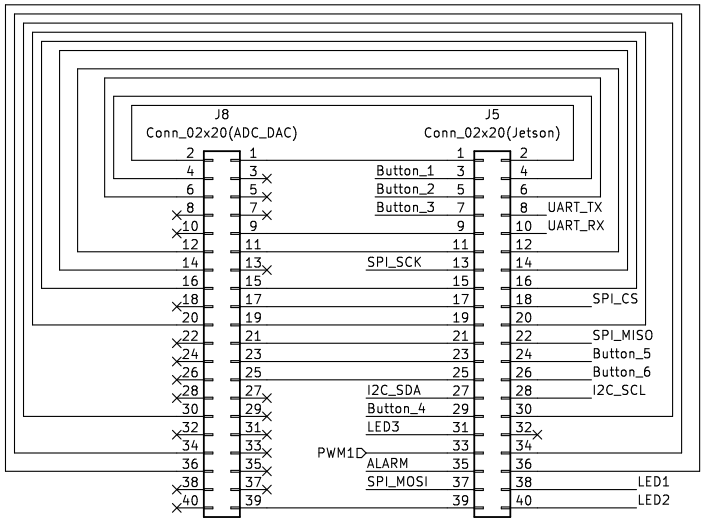

J19
TP16
JP_GND
Connect to
Jetson
Short GND &
JP_GND
JP_GND
TP35
TP14
15V
15VD
Connect to
Jetson
Ext JP power
CONN3
J18
21V
Short them to power
from external 21V

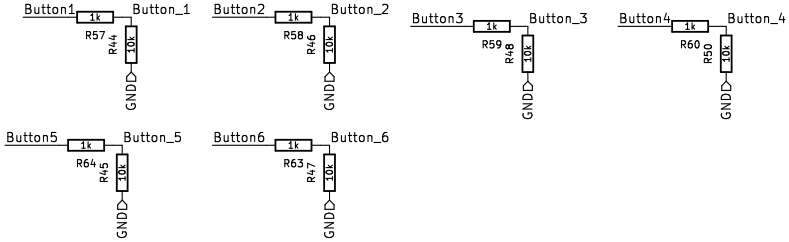
TP7
TP8
Tie
Tie
TP24
TP28
Tie
Tie
TP25
TP36
Tie
Tie

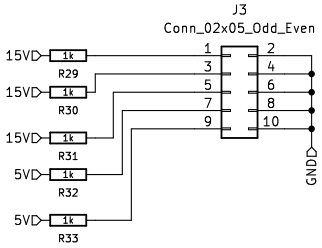

5V
1k
R22
4.7VD
D10
4.7V_Zener
GNDD

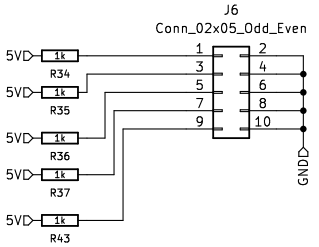

15V_P2
Gate_IRF840_2
GND_I/P2
U8
TL494
R89
R83
R84
R85
R86
C40
1nF
R87
R88
C42
C43
25V/100uF
100nF
C41
10uF
VR2
TL783CKTTRG3
D22
D21
Yellow
R82
48V_I/P2
TP23
M_conv
C39
47uF
C38
100nF
R81
D20
RB088NS150FHTL
T1
Transformer_1P_1S
TP22
Snubber2a
D19
R78
R79
R75
R77
C36
C37
Snubber2b
TP20
Q2
IRF840
R80
C35
100uF
D23
Green
R90
GND
J20
V_motor
CONN4
Ext motor power
Short them to power
motor from converter
CONN5
Motor Power
DC_Barrel_Jack_Switch
J4

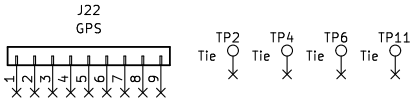
USonic_sense
DClink
TP29
4.7V
GND
SD1206S040S0R5
D14
D28
1k
R42
LM324PW
U9B
8.2k
R41
C21
100nF
C20
100nF
8.2k
R40
GND
C15V
C33
100nF
U9A
LM324PW
GND
USonic
1k
R38
open
R39
GND
U9D
LM324PW
U9C
LM324PW
J7
Conn_01x08
15V_sense
5V_sense
Curr1_sense
Curr2_sense
Pot1_sense
Pot2_sense
USonic_sense
USonic
J13
5VD
USonic
GND
Pot2
J2
5VD
Pot2
GND
Pot1
J9
5VD
Pot1
GND
J15
R102
1k
R49
1k
GND
DAC o/p
J14
Speed
R101
1k
R51
1k
GND
J23
15V_sense
DClink
TP3
4.7V
GND
SD1206S040S0R5
D8
D9
1k
R21
U4B
LM324PW
8.2k
R20
C17
100nF
C16
100nF
8.2k
R19
C15V
C22
100nF
GND
U4A
LM324PW
GND
15V
1k
R17
3.3k
R18
GND
5V_sense
DClink
TP5
4.7V
GND
SD1206S040S0R5
D11
D12
1k
R27
U4D
LM324PW
8.2k
R26
C19
100nF
C18
100nF
8.2k
R25
U4C
LM324PW
5V
1k
R23
1.5k
R24
GND
Pot1_sense
DClink
TP27
4.7V
GND
SD1206S040S0R5
D26
D27
1k
R100
U5B
LM324PW
4.5k
R98
C47
100nF
C45
100nF
4.5k
R96
C15V
C32
100nF
GND
U5A
LM324PW
GND
Pot1
1k
R93
2k
R94
GND
Pot2_sense
DClink
TP26
4.7V
GND
SD1206S040S0R5
D24
D25
1k
R99
U5D
LM324PW
4.5k
R97
C46
100nF
C44
100nF
4.5k
R95
U5C
LM324PW
Pot2
1k
R91
2k
R92
GND











Mounting Jetson

H9 MountingHole
H10 MountingHole
H11 MountingHole
H12 MountingHole

Mounting ADC/DAC

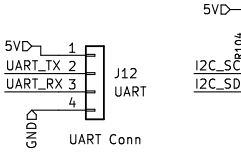
H5 MountingHole
H6 MountingHole
H7 MountingHole
H8 MountingHole

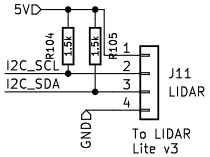
Mounting PCB

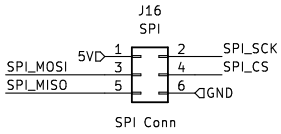
H1 MountingHole
H2 MountingHole
H3 MountingHole
H4 MountingHole

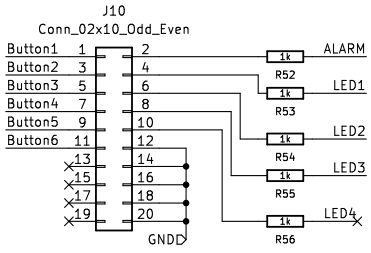
Mounting GPS

H13 MountingHole
H14 MountingHole

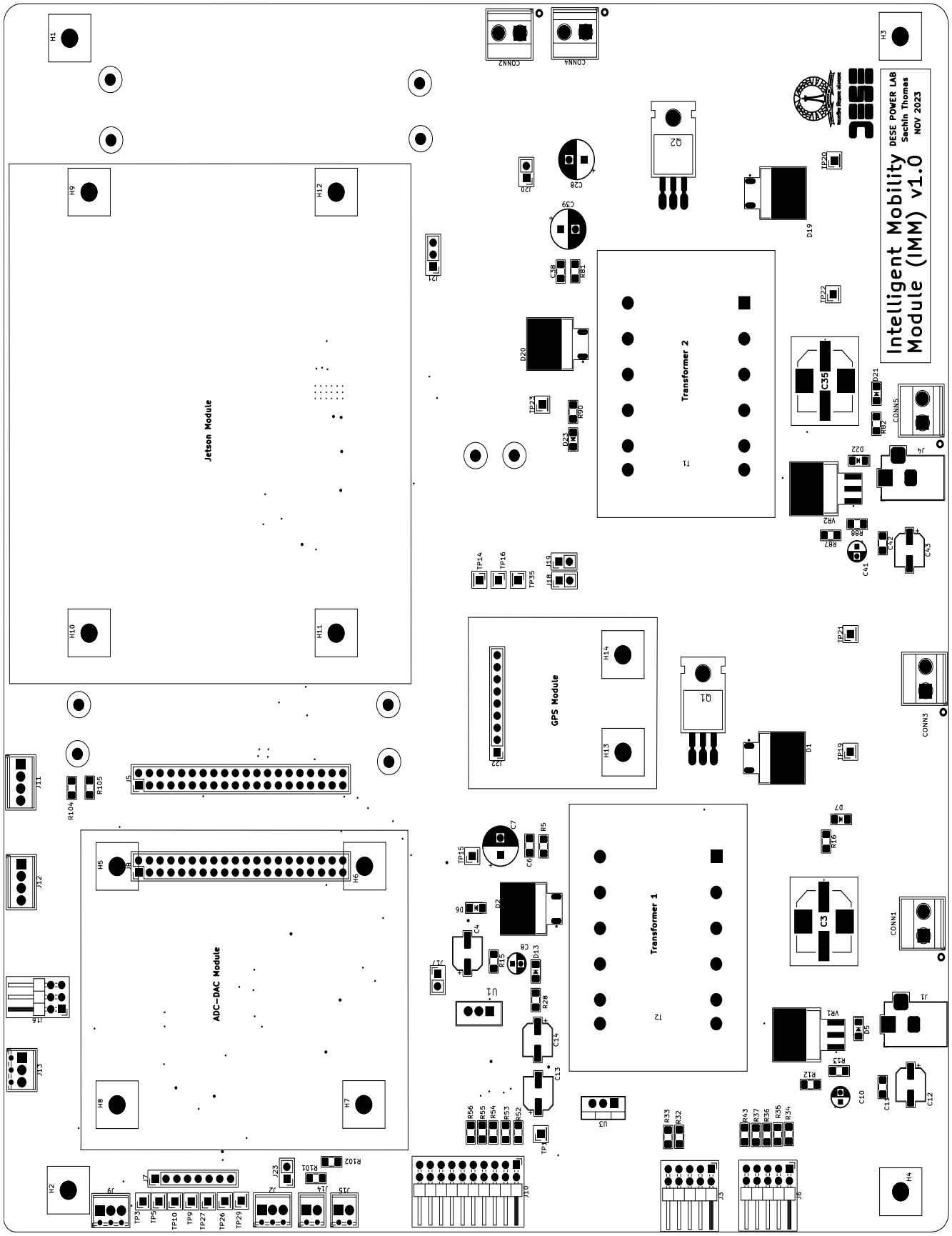


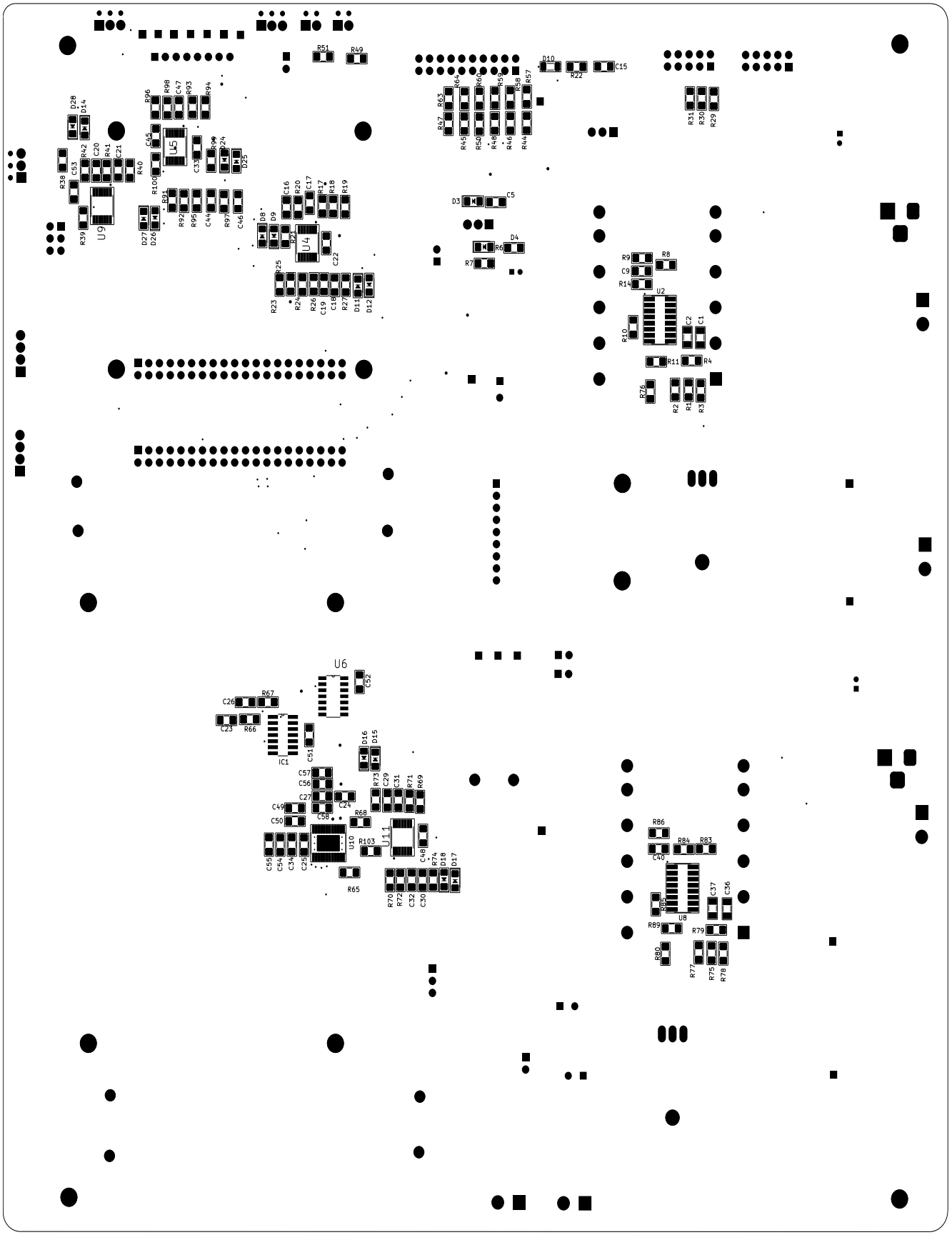






U10
DRV8873HPWPRQ1
DVDD
VCP
VM
CPH
CPL
OUT1
OUT2
SRC
NFAULT
IPROPI1
IPROPI2
EXP
GND
EN/IN1
PH/IN2
NSLEEP
DISABLE
NOL
SR
MODE
NITRIP
CONN2
Motor
Mot1
Mot2
Curr1
Curr2
R103
R68
R65
J21
PH_IN1
PH_IN2
5V_motor
Short to 5V to
disable current limit
U6
74HC04D
IC1B
74HC08D
IC1C
74HC08D
IC1D
74HC08D
IC1E
74HC08D
IC1A
74HC08D
PWM1
PH_IN1
Curr1_sense
Curr2_sense
LM324PW
SD1206S040S0R5
SD1206S040S0R5R
TP10
TP9
DCLink
4.7V
15V
5V
GND


Intelligent Mobility
Module (IMM) v1.0
DESE POWER LAB
Sachin Thomas
NOV 2023
Jetson Module
ADC-DAC Module
GPS Module
Transformer 1
Transformer 2
CONN1
CONN2
CONN3
CONN4
CONN5


U6
U5
U4
U9
U2
U11
U10
U8
IC1

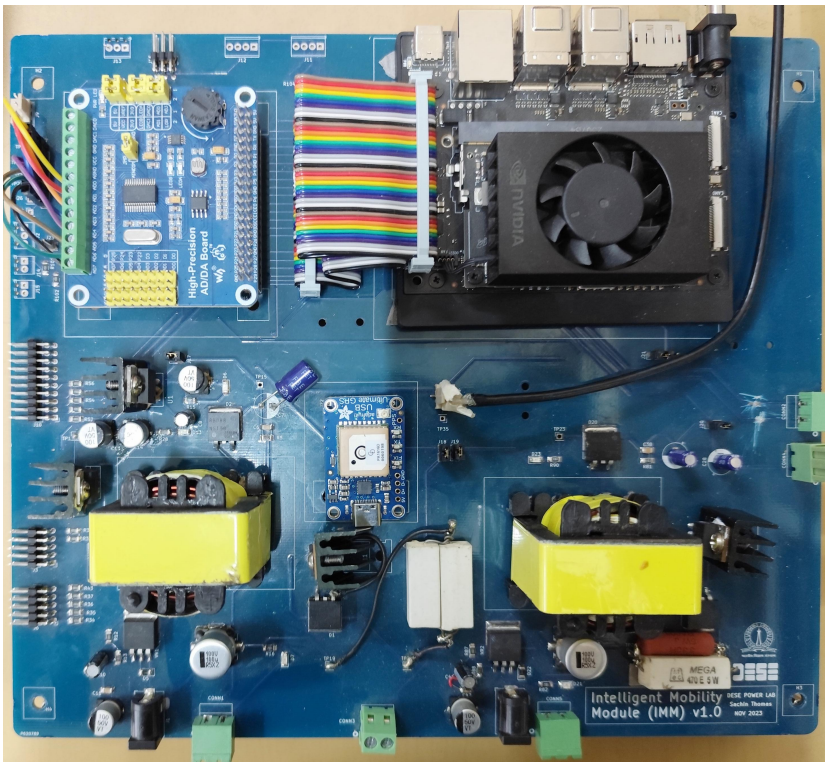


Figure 6.19: Top side of the PCB

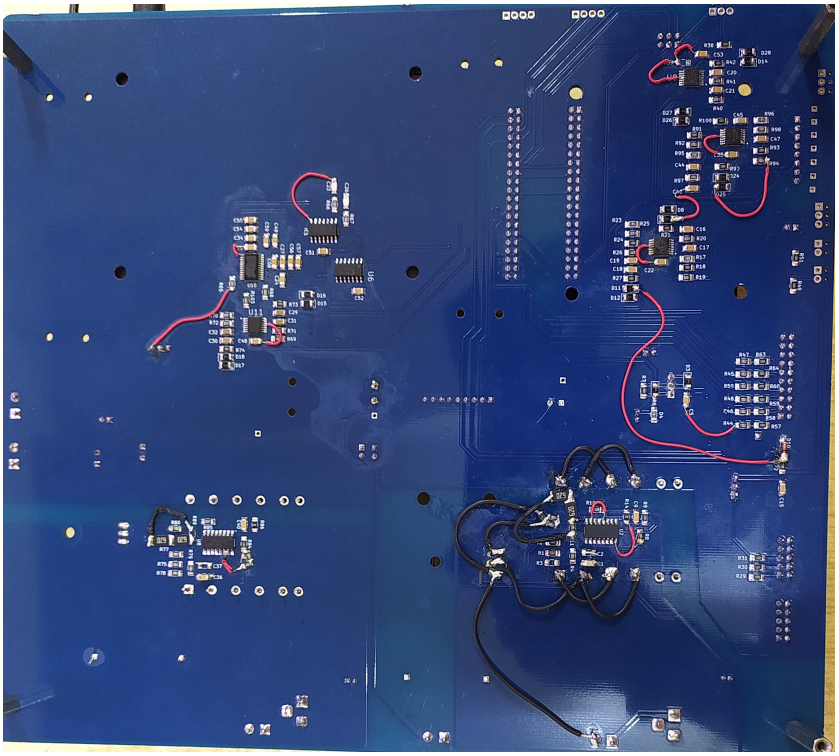


Figure 6.20: Bottom side of the PCB

6.2 Software Design

Designing software for the autonomous driving car on the Jetson Orin Nano requires several critical features for optimal performance and safety. Real-time processing is essential for handling vast sensor data and making instantaneous decisions. Additionally, safety and redundancy are crucial, with fail-safe mechanisms and error handling to ensure continuous operation. Furthermore, scalability and modularity in the software architecture allow adaptation to different vehicle models and easy updates. Efficient resource management is also vital to maximize computational capabilities while minimizing power consumption. Moreover, a user-friendly interface is necessary for monitoring and diagnostics. Finally, optimizing hardware resource utilization ensures efficient and effective operation, leveraging Jetson Orin Nano's strengths. By integrating these features, the software becomes robust, adaptive, and capable of delivering a reliable autonomous driving experience.

The software can be implemented using several programming languages like C, C++, Python, Java, etc. However, Python has several advantages over other languages. Python's simplicity and readability make development faster and more accessible. Its extensive libraries and frameworks, such as TensorFlow, Pytorch, and OpenCV, are highly beneficial for machine learning and computer vision applications. Additionally, Python's strong community support and cross-platform compatibility streamline troubleshooting and integration with various hardware and software components, whereas languages like C++ or Java might offer better performance but at the cost of increased complexity and development time. Hence the software implementation of the autonomous driving algorithms is done using Python language in Jetson Orin Nano.

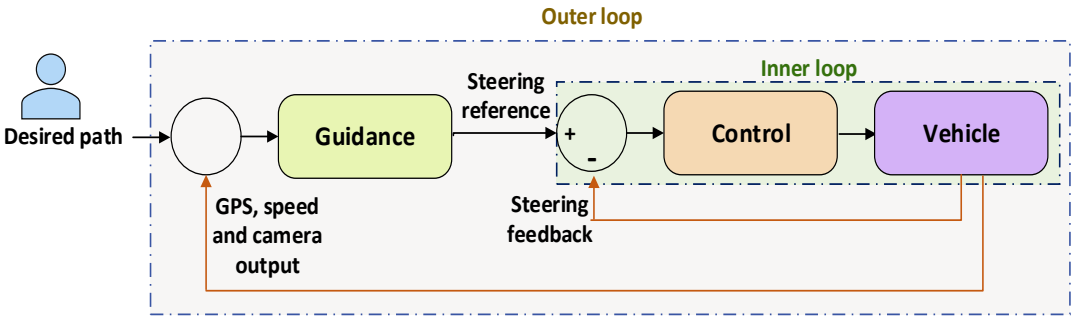


Figure 6.21: Control loops in the system

As mentioned in Chapter 2, the software for autonomous driving vehicle consists of two processes or loops operating at different periodicity. The outer guidance loop consisting of perception, localization, path planning, and reference generation operates at a slower rate defined

by the processing capability of the controller. However, the inner control loop runs at a faster rate for the bandwidth and stability requirements of the system. The inner loop operates at a periodicity of 1 ms. The speed reference and steering reference generated by the outer loop are transferred to the inner loop for implementing the steering motor control and converting the digital speed reference to analog using the DAC.

Pseudo code of both the processes - Outer Loop process and Inner Loop process are given in Algorithms 1 and 2 respectively.

The inputs to the outer loop process are the frames captured by the camera and the GPS output and the outputs from this process are the steering and speed references given to the inner loop. When the process is entered, the algorithm parameters mentioned in the fourth chapter and the perception model parameters are initialized. After that, a while loop is run inside which the perception models like object detection, depth estimation, lane detection, and pothole detection are implemented. The output of these individual models are used to generate a perception of the surroundings. Subsequently, the latitude and longitude from the GPS sensor are obtained for localization. Now, the speed reference and steering references are generated based on the perception output and the present location of the vehicle. This while is continuously run when the camera is open to capture frames.

The inner loop process implements the control loop. Inputs to this process are the references generated by the outer loop and the data converted by the ADC which includes the potentiometer and speed values. Inside the process, initially, ADC, DAC, control model, and the PWM are initialized to their required values. After that a always while loop runs which implements the control loop. Inside the while loop, firstly the ADC parameters are obtained followed by running the PI control loop to generate PWM for steering motor control. This generated PWM is connected to the DC motor control circuit. In addition to that, the while loop also provides the speed reference to the DAC for analog conversion. The output of the DAC is given to the Speed Control Module.

The software algorithms referenced here need to be implemented on Jetson Orin Nano hardware, leveraging its resources efficiently to achieve higher throughput and lower power consumption. Jetson Orin Nano features an 8-core Arm Cortex-A78AE CPU and a 1024-core Nvidia Ampere GPU with 32 Tensor Cores. It offers up to 40 TOPS of AI performance and supports up to 8 GB of LPDDR5 memory. The device includes various peripherals such as PWM, USB, Ethernet, HDMI, MIPI-CSI, SPI, UART, and I2C expanding its peripheral connectivity for a wide range of embedded system designs and IoT applications. Effectively utilizing these interfaces is crucial in the software design.

Algorithm 1: Outer loop process

Input: Image, GPS

Output: Steering reference, Speed reference

Initialization :

Initialize parameters
Initialize perception models
while camera is open **do**
    run object detection algorithm
    run depth estimation algorithm
    run lane detection algorithm
    run pothole detection algorithm
    generate surrounding perception
    find the current lat/ long from the GPS
    generate speed (V_{ref}) and steering (θ) references
return V_{ref} θ

Algorithm 2: Inner loop process

Input: Steering reference, Speed reference, ADC output

Output: PWM, DAC input

Initialization :

Initialize ADC and DAC parameters
Initialize control model parameters
Enable PWM
always do
    get ADC output
    run the PI control model
    generate PWM for steering motor control
    Provide speed reference as input to the DAC
return V_{ref} PWM

As mentioned earlier, the periodicity of the outer loop process and the inner loop process are different. The inner loop has to run at a periodicity of 1 ms, whereas the outer loop runs at a slower rate determined by the processing capability of the controller. In addition to that the processing capability needed for the inner loop is very less compared to that of the outer loop. Considering these needs and constraints, a single CPU core is dedicatedly assigned to the inner loop for implementing the control loop and the remaining five cores are used for the outer loop process. In the outer loop itself, to accelerate the machine learning models in the perception, a 1024-core Ampere GPU is used. The overall Jetson Orin Nano resources utilized for autonomous driving software implementation are tabulated in table 6.12.

Resource	Usage
Ampere GPU	Perception models
CPU cores 1 - 5	Outer loop process
CPU core 6	Inner loop process
SPI	ADC & DAC
PWM	Steering motor control
USB	Camera, GPS
WiFi, ethernet	Device login

Table 6.12: Jetson Orin Nano resources used

To configure the Jetson Orin Nano, a software development kit (SDK) developed by Nvidia needs to be installed on the device. JetPack SDK 5.1.2 is the particular SDK used for this project. It offers tools and libraries for AI, computer vision, and deep learning applications. It includes TensorRT, cuDNN, and support for the latest CUDA toolkit. This SDK streamlines development by providing pre-configured packages for seamless deployment on Jetson devices.

The Python implementation of the algorithms, especially the machine learning models used requires several dependent packages to be installed. These packages and their functionality are tabulated in table 6.13.

The directory structure for the project is meticulously organized to streamline the management and access of various components necessary for its functioning as shown in Fig. 6.22. *Autonomous_driving_project* is the main directory that has several subdirectories. The *Input video* subdirectory contains multiple video files (e.g., *video1.mp4, video2.mp4, ..., videon.mp4*). These videos serve as the raw data that the autonomous driving system processes to perform various tasks such as perception and reference generation.

Package	Functionality
Numpy	Numerical computing
Pandas	Data manipulation
Scikit-learn	Machine learning
Seaborn	Data visualization
OpenCV	Computer vision
Pillow	Image processing
PyYAML	YAML parsing
psutil	System monitoring
imutils	Image utilities
PyTorch	Deep learning
Torchvision	Image processing

Table 6.13: Python dependent packages

Within the *Perception models* directory, there are subdirectories for different types of perception tasks, each containing the required files for their specific function. The *Object detection* subdirectory includes a TensorRT engine file (*object_detection.trt*), a weight file (*object_detection.wt*), and a Python script (*object_detection.py*) to perform object detection tasks. Similarly, there are subdirectories for depth estimation, lane detection, and pothole detection.

The *Desired route* folder consists of CSV files (*route1.csv, route2.csv, ..., routen.csv*) that describe the intended routes for the autonomous vehicle to follow. These files contain the latitude and longitude information of the path that guides the vehicle along its journey.

The *Output* directory is designated for storing the results generated by the system after processing the input videos. This includes the processed video file (*output_video.mp4*), a CSV file containing the output path data (*output_path.csv*), a video file showing depth information (*output_depth.mp4*), and an action video file (*output_action.csv*). These outputs are crucial for analyzing the performance and effectiveness of the perception models.

Lastly, the directory includes configuration and script files essential for initializing and running the project. The *init_parameters.csv* file contains the initial parameters required to set up the project, while the *main.py* script serves as the main executable Python script that orchestrates the overall functioning of the autonomous driving system.

In this chapter, we have discussed the hardware and software design of the Intelligent Mobility Module. In the hardware design, circuits of the Controller, ADC & DAC, DC motor control,

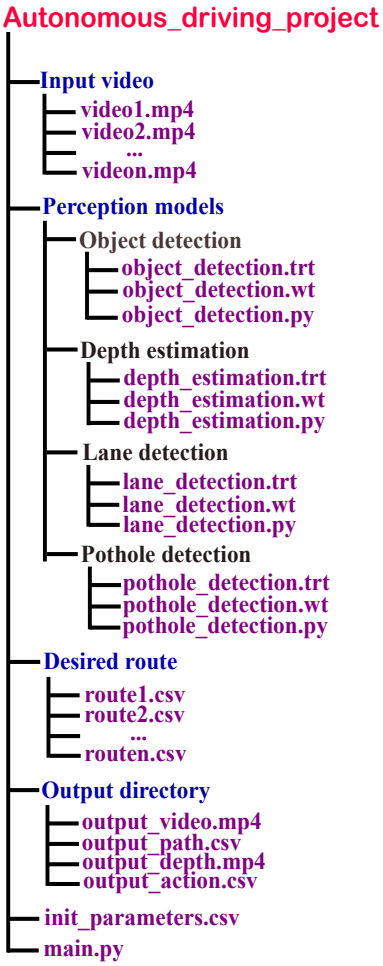


Figure 6.22: Directory structure of the autonomous driving project

User Interface, and Power Distribution Network were discussed. And in the software design, the pseudo code of the algorithm and the hardware resource utilization of the Jetson Orin Nano were looked at. In the next chapter, we will see some of the results obtained in this project and a discussion on the same.

Chapter 7

Results and Discussion

In the previous chapters, we have looked at the algorithm design (including perception and reference generation) and hardware & software design of the autonomous driving car project. In the current chapter, we will look at some of the results obtained from this project. In the first section, we will discuss the performance of the hardware module, the Intelligent Mobility Module (IMM), followed by the software performance in the next section. In the final section, the performance of the system in an emulated environment is presented.

7.1 Hardware Performace

The performance of the submodules and circuits of the Intelligent Mobility Module was already discussed in the previous chapter. In this section, we look at the overall performance of the module in the system. The hardware setup for the evaluation of the module is shown in Fig. 7.1.

In the test setup, the IMM serves as the core component, processing inputs from various sensors and feedback devices to manage the autonomous driving functions. It interfaces with input devices such as a camera, which provides visual data enabling the IMM to process and analyze the environment. Additionally, feedback mechanisms for steering and speed are incorporated to provide real-time data on the vehicle's steering angle and speed. The IMM sends control signals to the Speed Control Module (SCM) to regulate the vehicle's speed and adjusts the steering angle by sending commands to the steering motor. To power the system, two 48 V batteries are present. These batteries provide the necessary power for the IMM and other connected components, with each battery connected to the IMM via individual switches (Switch 1 and Switch 2). These switches ensure controlled and safe power delivery, allowing the system to be

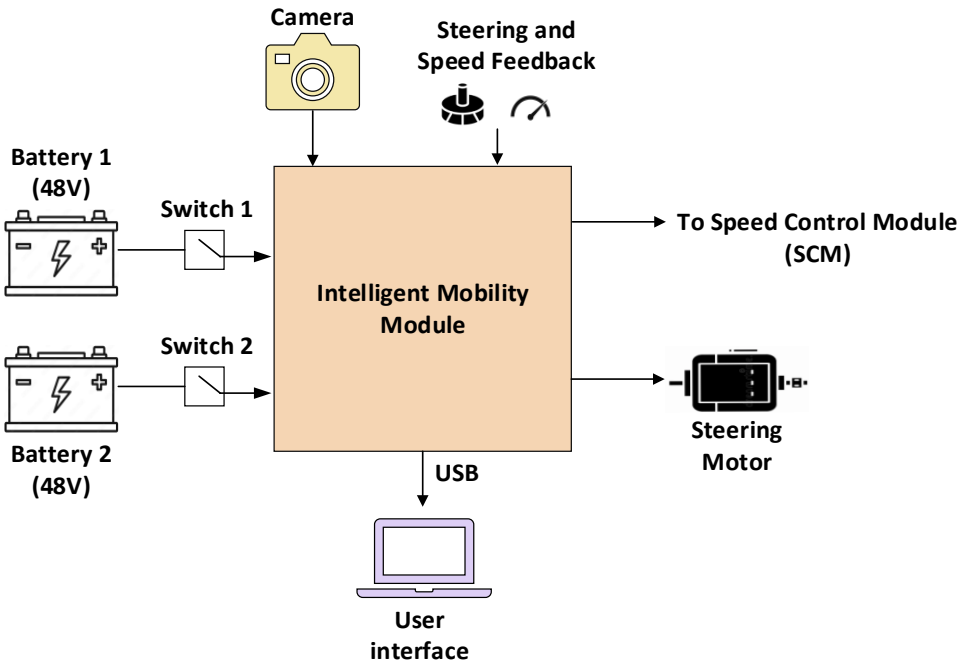


Figure 7.1: Hardware setup of IMM

powered on or off securely. Finally, to monitor the system performance and provide necessary commands to the system a user laptop is interfaced to the IMM through the USB connection. In this test setup, the primary objectives include validating sensor integration to ensure accurate data provision from the camera and feedback sensors to the IMM, verifying control accuracy to ensure the IMM can effectively control the speed and steering of the vehicle, and finally monitoring the crucial parameters to improve the system.

The average current and power consumed from the batteries are tabulated in table 7.1. This information is useful to estimate the life of the battery.

	Battery 1	**Battery 2**
Current (A)	0.61	0.5
Power (W)	29.3	24

Table 7.1: Average current and power consumed from the batteries

A typical current profile of battery 1 is shown in Fig. 7.2. From the plot, it can be inferred that when the IMM is powered, it consumes 0.4 A from the battery. This is the static power of the module before running the Autonomous driving program. Now, when the program is run, it consumes an additional 0.2 A. This current is mainly consumed in running the computationally intensive perception algorithms.

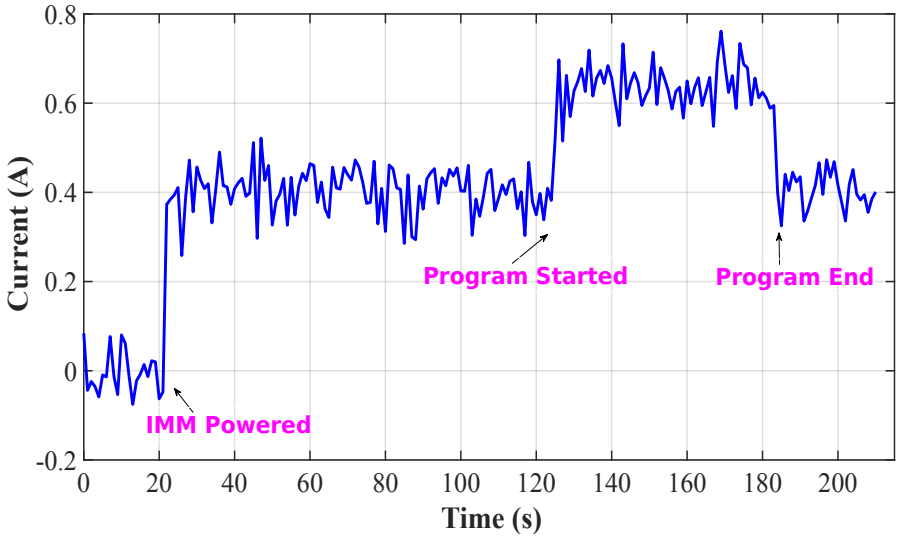


Figure 7.2: Typical battery 1 current plot

Similarly, the current profile of battery 2 is shown in Fig. 7.3. When the motor is powered ON by turning ON the switch 2, a current of 0.2 A is drawn from the battery. This is the current to power the flyback converter in the PDN 2. Now, when motoring action is carried out, a peak current of 3.5 A is withdrawn from the battery. This current is used to provide sufficient torque to the steering motor to overcome the steering system inertia. Finally, when the battery 2 is switched OFF, the current goes to zero.

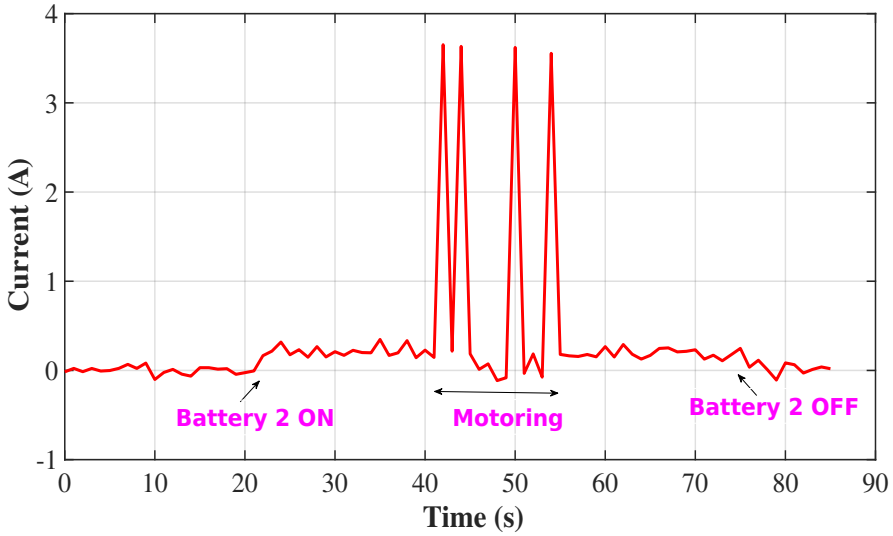


Figure 7.3: Typical battery 2 current plot

7.2 Software Performace

Evaluating the software's performance is critical to understanding its response to various situations. Firstly, the latency of each of the modules in the software is analyzed. A comprehensive breakdown of their performance times, illustrated in the pie chart 7.4 provides insights into their respective contributions to the overall processing time. Object detection required 27 ms, accounting for 22.4% of the total processing time. Lane detection took 37 ms, representing 30.7% of the total. Depth estimation, being the most time-consuming function, utilized 55 ms, which is 45.6% of the total processing time. In contrast, reference generation was significantly faster as it doesn't contain any computationally intensive ML model, requiring only 1.5 ms and contributing only 1.2% to the total processing time. Finally, the overall system throughput is obtained as 8 frames per second (fps). From this analysis, it can be inferred that by improving the throughput of the depth estimation algorithm, a significant improvement in the overall system throughput can be achieved.

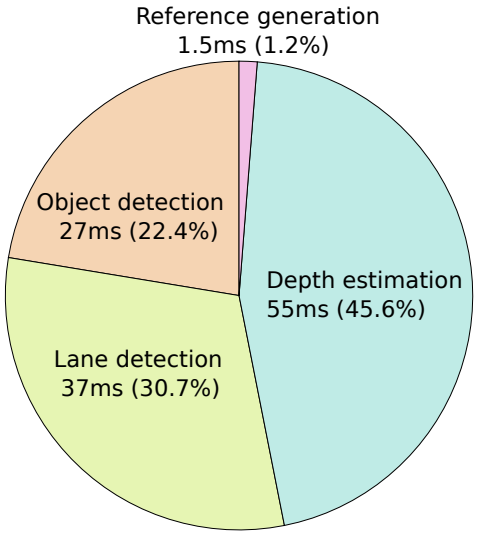


Figure 7.4: Breakdown of latency of algorithms

Next, we analyze the performance of Jetson when the autonomous driving project program is running. Firstly, the utilization of the CPU cores by the program is illustrated in Fig. 7.5. It can be seen that the CPU cores (cores 1 - 5) used for implementing the outer loop process utilize less than 45 % of the CPU. In contrast, the CPU core 6 used for implementing the inner loop process utilizes the complete CPU. This is expected as the perception models in the outer loop are implemented in the GPU and the CPU is used only for minor functionalities in the perception and reference generation. However, the inner loop is operating at a faster periodicity and hence the CPU utilization is also higher.

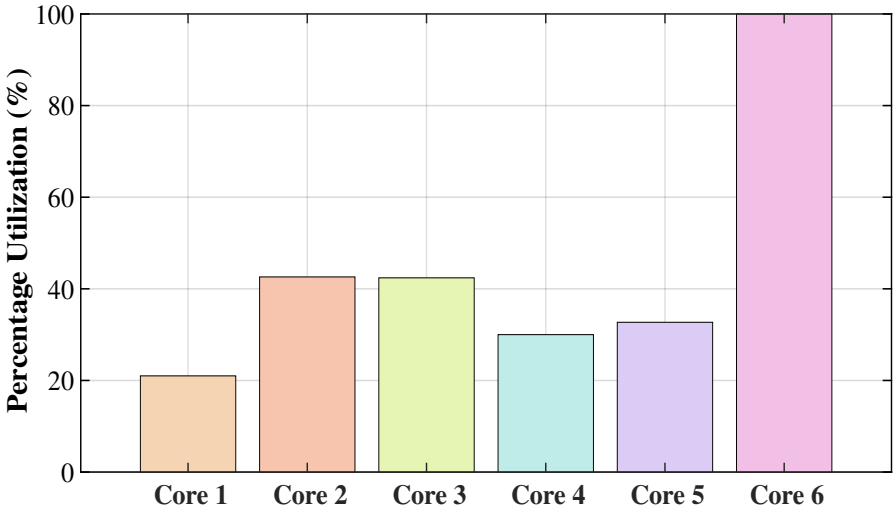


Figure 7.5: CPU core utilization

Some of the other important measurements of the Jetson Orin Nano during the execution of the program are tabulated in table 7.2. The temperature of the device measured here is after 20 minutes of the program execution.

Parameter	Value	Unit
Power	9.2	W
Temperature	56	° C
RAM utilized	4.9	GB
GPU utilization	92.3	%

Table 7.2: Jetson Orin Nano performance

7.3 System Performance

In this section, the system performance of the autonomous driving vehicle implemented in the IMM is evaluated. The test setup for this is as shown in Fig. 7.1. Firstly, the turn ON and turn OFF procedures of the system are tabulated in table 7.3 and 7.4 respectively. Following the sequence of operations given in these tables is extremely crucial for the normal operation of the system. In the turn-ON sequence of the system, a health check is provided to ensure that the system modules are functioning as expected. To implement that, the ADC output of 15 V and 5 V regulator voltages are compared against an expected bound. If the ADC output voltages

are outside this bound, then it indicates a failure in the power distribution network, and hence the sequence of operations have to be aborted.

Event	Action
1	Ensure proper connection as per Fig. 7.1
2	Enable switch 1
3	Login to Jetson Orin Nano
4	Execute the program
5	Enable switch 2 after health check

Table 7.3: Turn ON procedure of the system

Event	Action
1	Disable switch 2
2	Quit the program
3	Shutdown Jetson Orin Nano
4	Disable switch 1

Table 7.4: Turn OFF procedure of the system

To evaluate the performance of the system under various conditions an emulation setup is created. In this setup, the IMM is mounted on the tabby vehicle below the seat, the battery pack is mounted behind the seat in an open space, the steering motor is attached to the steering wheel and the camera is mounted at the front of the vehicle facing the road as shown in Fig. 7.6.

This emulation setup for the autonomous driving vehicle is essential for conducting comprehensive testing of various scenarios before deploying it on real roads. By simulating diverse conditions, we can identify potential issues and improve the system's robustness. Additionally, it allows for safer and more cost-effective testing, minimizing risks associated with real-world trials. In this setup, the camera is not utilized for perception. Instead, a pre-recorded video is stored on the Jetson and used as the input for testing the autonomous driving feature. This video serves as the primary input to the test setup, allowing for the evaluation of the system's performance in a controlled environment. The output of the system is the steering angle movement and the speed reference generated. Behaviour of the steering wheel for the dynamic conditions on the road is observed and recorded.

Four different routes within the IISc campus were selected to emulate the car's performance, tabulated in table 7.5. Videos along these routes were recorded offline (emulating the car's

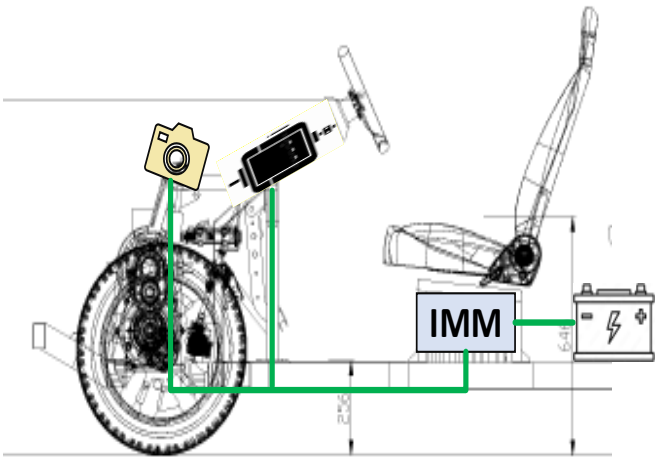


Figure 7.6: Physical placement of components on the tabby vehicle

camera) and saved in the Jetson device. These videos were used for the emulation tests.

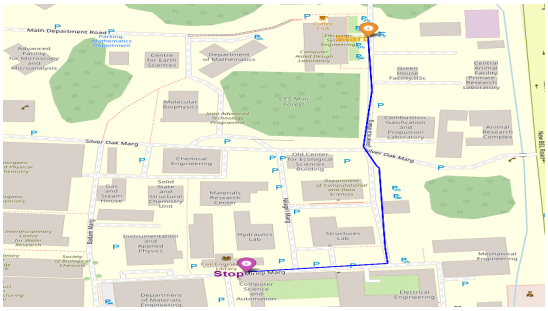
Test Number	Start Location	Destination
Test 1	DESE	CSA
Test 2	CSA	DESE
Test 3	ECE	Aero
Test 4	Aero	ECE

Table 7.5: Routes selected for the emulation

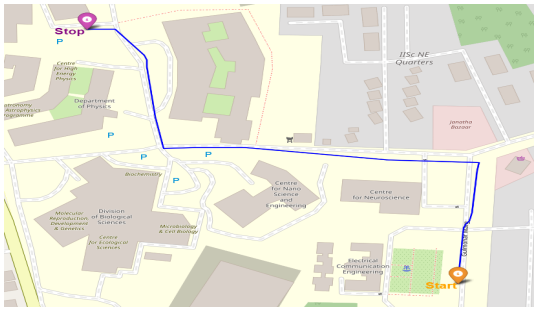
Firstly, A GUI was developed for the user to enter the source and destination locations and initiate the test as shown in Fig. 7.7. The user also gets information about the necessary system parameters and the speed of the vehicle in the GUI. At the top of the GUI, the user can select the start and destination locations. Once these locations are entered, the corresponding prerecorded video of the route and the trajectory from the start to the destination will be loaded. Once this has been loaded, the user has to press the enter button at the bottom of the GUI to start the autonomous driving action. While the program is running, the current location of the vehicle is shown as a red dot in the map plot, and the system parameters like the 15 V regulator output and the current speed of the vehicle are displayed for the user information.

Now, the planned path for all four test cases is shown in Fig. 7.8. The planned path obtained from Google Maps is projected onto a 2D plane for path planning as shown in Fig. 7.9. The pre-recorded video and the latitude/ longitude information of the route form the input to the emulation setup. The program is run to get the speed and steering references as the output.

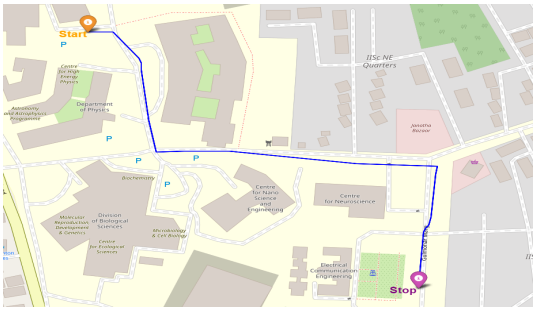
Figure 7.7: GUI for autonomous driving vehicle



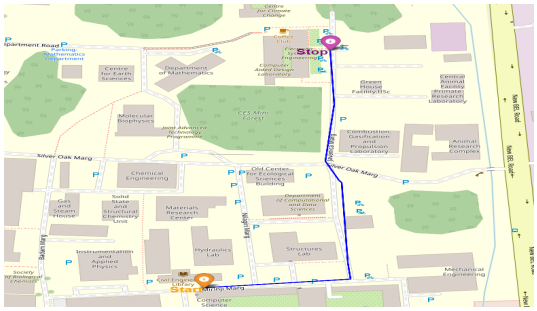
(a) DESE to CSA



(b) ECE to Aero



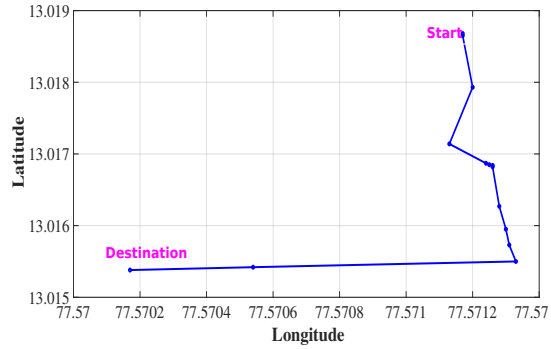
(c) Aero to ECE



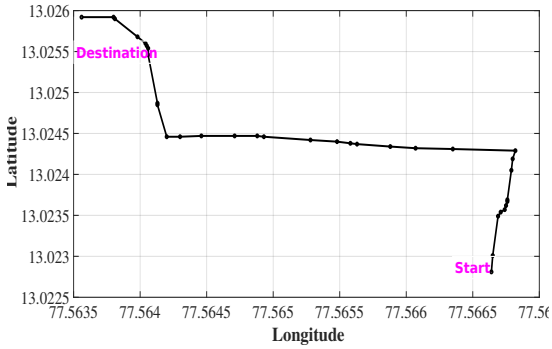
(d) CSA to DESE

Figure 7.8: Planned path for different test locations

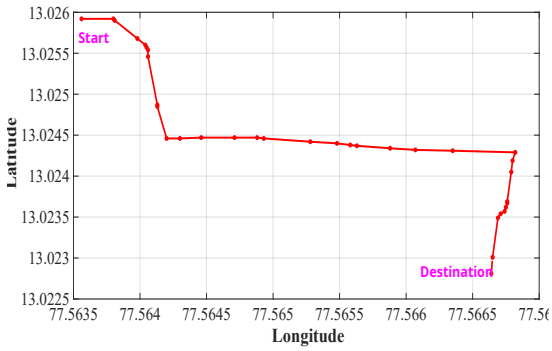
The speed and steering references generated for each of the four tests are plotted in Fig. 7.10 - 7.13. In the presented plots, the vehicle's speed and steering angle are analyzed over a specific route. Initially, the vehicle accelerates from zero to 30 km/h as it starts moving. This



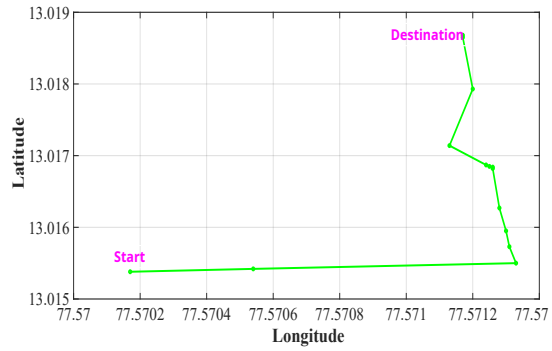
(a) DESE to CSA



(b) ECE to Aero



(c) Aero to ECE



(d) CSA to DESE

Figure 7.9: 2D plot of planned path for different test locations

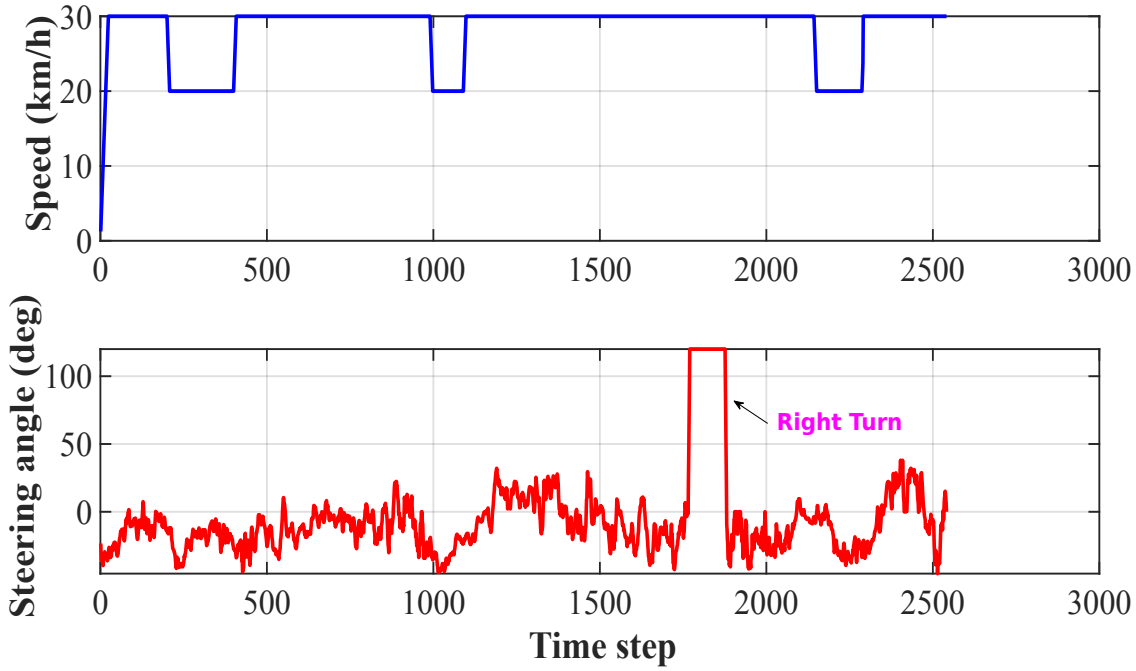


Figure 7.10: Reference generated for DESE to CSA route

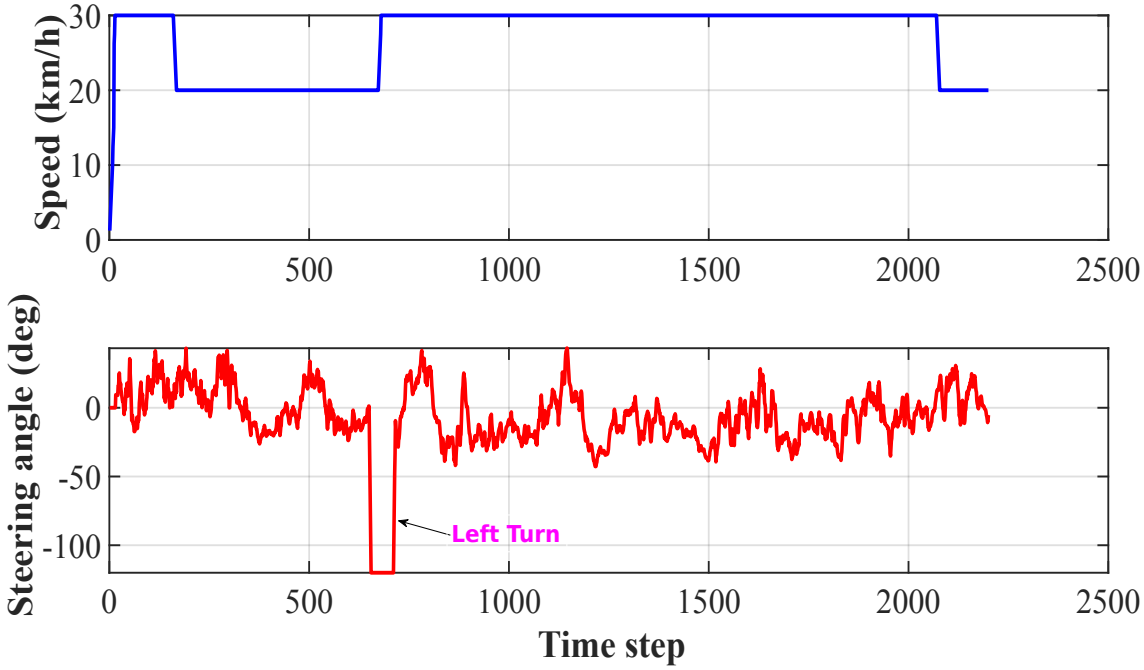


Figure 7.11: Reference generated for CSA to DESE route

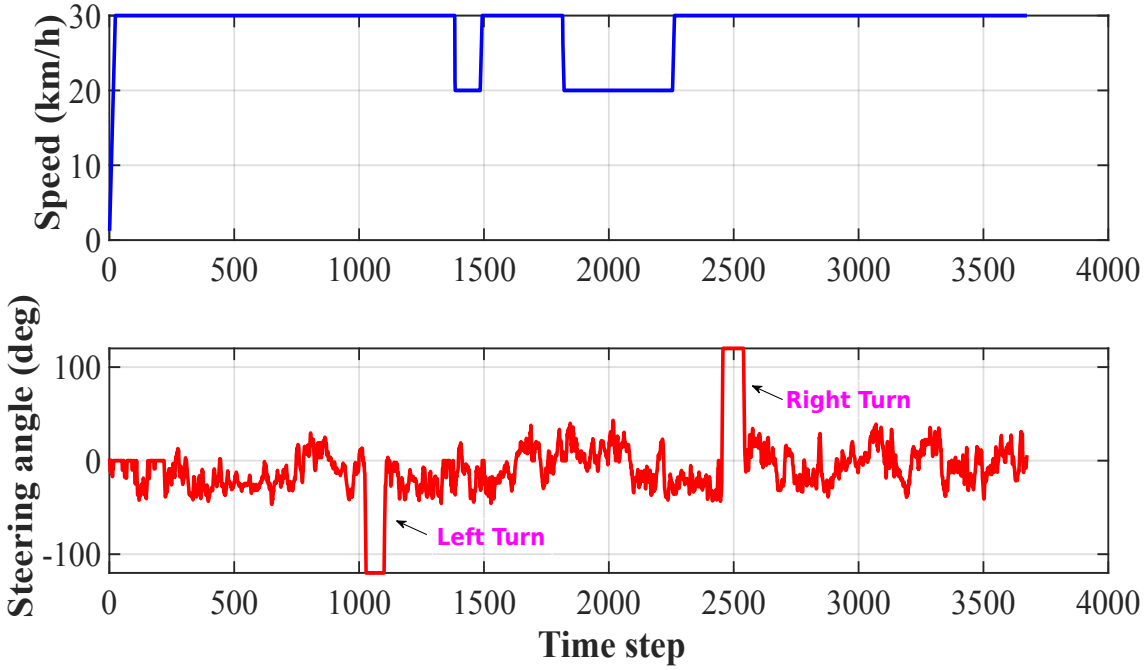


Figure 7.12: Reference generated for ECE to Aero route

acceleration occurs linearly and reaches the target speed of 30 km/h. The vehicle maintains this speed until it encounters a medium-traffic situation, which is characterized by having more than five obstacles in front of it. Upon encountering this medium traffic, the vehicle decreases

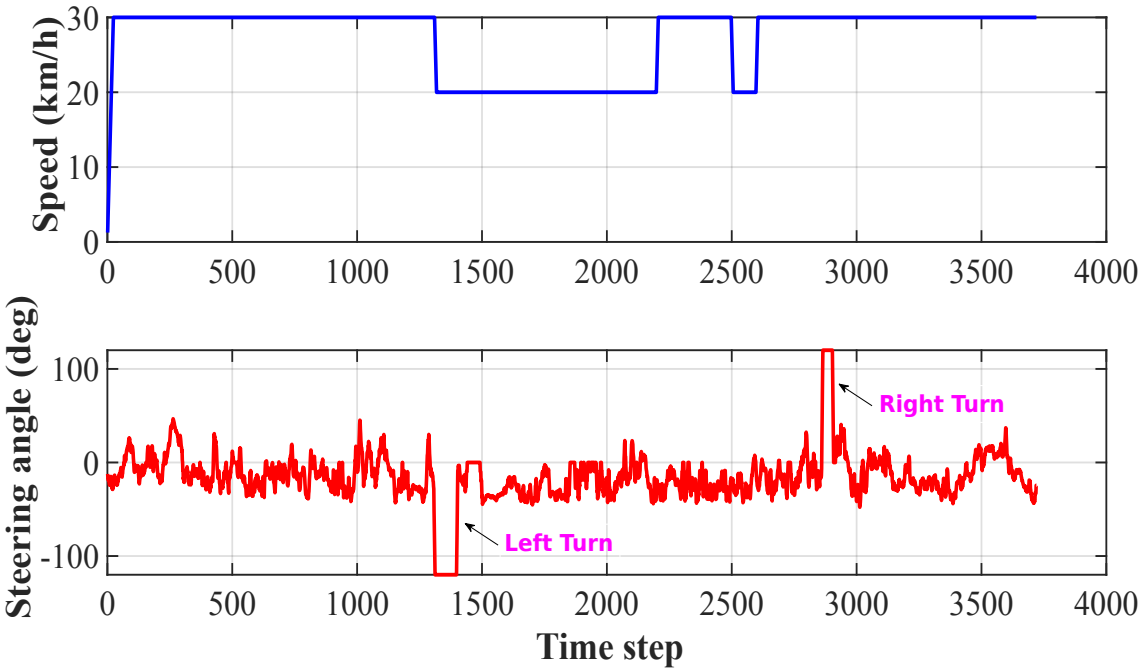


Figure 7.13: Reference generated for Aero to ECE route

its speed to 20 km/h and continues at this reduced speed until the traffic conditions improve and the obstacles are fewer. When the traffic is considered low again, the vehicle resumes its speed of 30 km/h.

The steering angle reference plot shows how the vehicle adjusts its steering to follow a target point obtained from the lane detection algorithm. The vehicle makes continuous adjustments to the steering angle to stay within the lane. When a turn is required, as determined by the vehicle's GPS location, the steering angle changes significantly to facilitate the turn. For example, a left turn is indicated by a sharp negative spike in the steering angle, while a right turn shows a sharp positive spike. These adjustments ensure that the vehicle can navigate the route effectively, responding dynamically to the path and traffic conditions. From the plots, it can be observed that the algorithm is functioning as expected for the given scenario. However, it is important to note the significant fluctuations in the steering angle reference, which may impact passenger comfort. To enhance this, a suitable post-processing stage could be implemented at the output of the steering reference angle algorithm to smooth out these fluctuations.

In this chapter, we have seen the results obtained for the hardware, software, and the complete system. The results indicate that the algorithms and the system are performing satisfactorily in the emulation setup. However, the implementation has to be extended to demonstrate the capability of the vehicle on the road. Due to the timing constraints, this was not implemented in the scope of the current project. In the next chapter, we conclude the project and discuss the performance of the vehicle against the target specifications intended in the prestudy phase of the project. In addition to that, we also discuss the future work to be carried out in the project to develop a fully autonomous vehicle that can be deployed on the road.

Chapter 8

Conclusion

Autonomous vehicles promise to enhance road safety, reduce traffic congestion, and provide greater mobility. With the rising trend and importance of autonomous driving vehicles, there is a significant push towards developing sophisticated architectures that can handle complex driving environments. This project was aimed at achieving SAE Level 3 autonomy, where the vehicle can handle most driving tasks independently while still allowing human intervention when necessary. The focus was on designing and implementing both the hardware and software components necessary to create a robust and reliable autonomous driving system.

This project has effectively illustrated the fundamental requirements for the control architecture of an autonomous driving vehicle. The design integrated both hardware and software components necessary to achieve autonomous functionality using a single camera. The hardware, known as the Intelligent Mobility Module (IMM), was built around a Jetson Orin Nano controller and included critical circuits such as Analog-to-Digital Converters (ADC), Digital-to-Analog Converters (DAC), DC motor control, user interface, and a power distribution network. Each of these components was meticulously designed, implemented, and evaluated for performance, ensuring their suitability for autonomous driving applications.

In the software domain, the control architecture encompassed advanced perception and reference generation algorithms. The perception system was particularly sophisticated, employing machine learning models to perform object detection, depth estimation, lane detection, and pothole detection. These capabilities ensured that the vehicle could accurately perceive and interpret its environment, which was crucial for safe and efficient autonomous operation. The reference generation system used a modular, rule-based approach to handle localization via GPS, path following based on a pre-stored route, and the generation of steering angle and speed references. The speed reference was given to a Speed Control Module (SCM) to control

an induction motor and the steering angle reference was used to control the steering wheel with a DC steering motor.

Additionally, the project report discussed the limitations of the current modular and rule-based approach in reference generation. While effective, this method could be improved upon by integrating more advanced techniques such as end-to-end learning and Deep Deterministic Policy Gradient (DDPG) based reinforcement learning. These approaches promised to enhance the system's adaptability and decision-making capabilities by allowing it to learn and optimize directly from environmental interactions, rather than relying solely on pre-defined rules.

Finally, the IMM was integrated into an open-source tabby vehicle, and emulation of the design was carried out using a pre-recorded offline video sequence, demonstrating obstacle handling and lane following with videos captured on the IISc campus. The results indicate that the designed architecture can handle different environments and maintain reliable performance, meeting the core objectives of autonomous navigation. However, the driving capability of the vehicle on the road was not demonstrated due to the time constraints of the project.

The final specifications of the vehicle implementing the autonomous driving feature are compared against the target features in the prestudy phase and tabulated in table 8.1. It can be observed that many of the qualitative features targeted in the study phase are matching with the final specifications. However, some parameters have slight deviations from the desired specification. The acceleration and deceleration targeted for the vehicle was 6 m/s^2. But, the achieved value is only 3 m/s^2. The achieved value is sufficient to operate the vehicle in normal conditions. However, the reduced deceleration capability of the vehicle has implications on handling the obstacles especially when the obstacles are nearby. The lower deceleration rate of the vehicle demands the IMM to initiate the obstacle avoidance action much ahead of the obstacle and safely bring the vehicle to rest. This can be improved by using a higher torque motor in the future design of the vehicle.

In the prestudy phase, the vehicle was also expected to implement the platooning feature. In platooning mode, the vehicle travels closely together with other vehicles in a convoy, coordinated by advanced communication and control systems, to improve efficiency and safety. This requires the implementation of different layers of the communication protocol stack to enable wireless communication with other vehicles and infrastructure. Due to the time constraints of the project, this feature was not implemented. However, it can be incorporated in the next version.

Also, in the initial study, it was planned to operate the vehicles on village roads. Village roads account for more than 50% of the Indian roads and hence is a useful feature to be added in

the autonomous driving vehicle. However, these roads lack proper lane markings making them less suitable for the lane detection algorithm in the perception module. Hence, adapting the vehicle to operate in these conditions becomes challenging in the present design. However, by incorporating a reinforcement learning-based approach [76], this feature can be also achieved in the next version of the design.

Parameter	**Target**	**Achieved**	**Unit**
Vehicle weight (with battery)	520	520	kg
Maximum passengers	2	2	Nos
Maximum vehicle speed	30	30	km/h
Vehicle designed acceleration/ deceleration	6	3	m/s^2
Emergency deceleration	-8	-4	m/s^2
Operation modes	Manual, Autonomous		
Primary features	Obstacle avoidance, Lane following Platooning	Obstacle avoidance, Lane following	
Road lanes	Single, Double Village	Single, Double	
Environment condition	Daylight		
Input Sensor	Camera		
Battery Requirement	Two 48 V		
No. of obstacles handled per instance	> 5	1	Nos
Maximum obstacle distance from the vehicle that will be detected	50	25	m
Throughput	30	8	fps
IMM total power	54		W
IMM dimension	260 x 225		mm

Table 8.1: Targer and final specifications of the designed vehicle

Moreover, in the target specification, the design was expected to handle more than 5 obstacles in an instant. This feature is suitable to operate the vehicle in a high-traffic scenario observed in the cities. However, due to the complexity of the reference generation algorithm, this feature is postponed to be implemented in the next version after successfully demonstrating the current version on the road. Furthermore, the project aimed to achieve a maximum obstacle detection distance of 50 meters but realized a distance of 25 meters. A longer detection range enables

the vehicle to make more informed decisions. However, this is constrained by the effectiveness of perception algorithms in detecting obstacles and estimating their depth. To enhance this capability in future iterations, additional sensors such as radar or Lidar could be integrated.

Finally, the targeted throughput of the IMM module was 30 fps against the achieved 8 fps. A higher throughput in the was targeted to make the system adapt to the complex conditions arising in the dynamic environment. However, the throughput is limited by the computational capability of the Jetson Orin Nano to process complex machine-learning models in the perception algorithm. This can be enhanced by profiling the program extensively and pruning the models to tradeoff the accuracy with the throughput. It is expected that by employing state-of-the-art pruning techniques the throughput can be improved. Moreover, in the future design, a more advanced controller like Jetson Orin NX or AGX can be used which is capable of achieving up to 100 TOPS and 275 TOPS respectively against the 40 TOPS of Jetson Orin Nano.

8.1 Future Work

The field of autonomous driving is highly active in research, with ongoing advancements in several key areas. The present research activities mostly focus on improving sensor fusion techniques to enhance environmental perception, developing more sophisticated machine learning models for better decision-making, and enhancing vehicle-to-everything (V2X) communication for safer and more efficient traffic management. Additionally, the autonomous driving Tabby vehicle presents opportunities for interdisciplinary development activities, fostering collaboration across fields such as embedded systems, IoT, robotics, artificial intelligence, and transportation engineering. Considering these scopes for research and development, a roadmap for future activities is shown in Fig. 8.1.

The roadmap for the autonomous driving vehicle project, as illustrated in the provided diagram, is divided into four distinct phases, each focusing on critical advancements in technology and infrastructure. In the initial phase, termed Phase 0, the primary objective is to demonstrate the vehicle's capabilities on the road. In this current project, only an emulation was carried out due to time constraints. This work has to be extended to demonstrate the algorithm's capability to operate on the road. This phase is crucial for showcasing the baseline functionalities and setting the stage for further development.

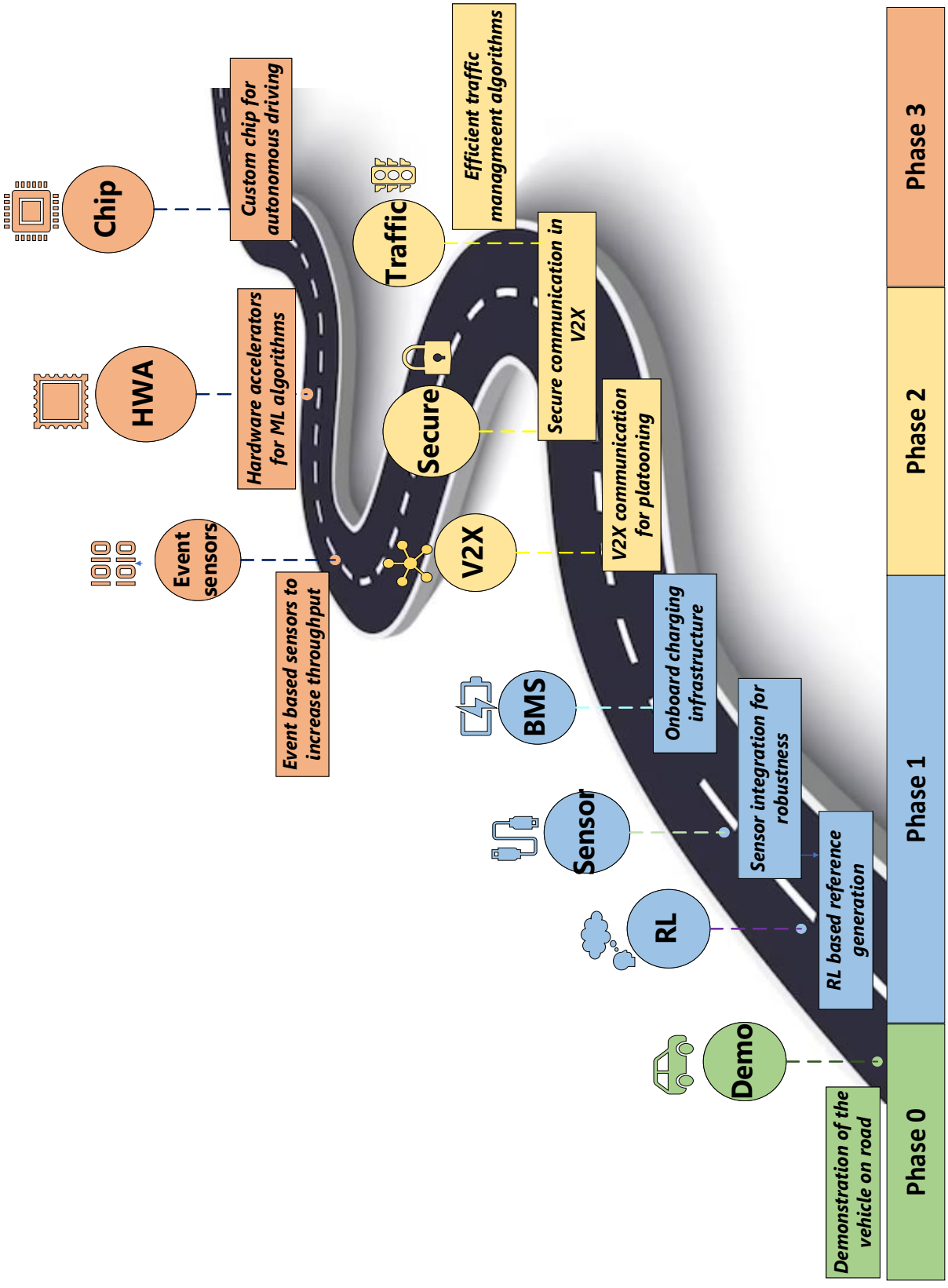


Figure 8.1: Future roadmap of the project

In the next phase, Phase 1 focuses on the integration of multiple sensors and the development of robustness in the system. Key components in this phase include the incorporation of Reinforcement Learning (RL) for reference generation and the deployment of onboard charging infrastructure through Battery Management Systems (BMS). Additionally, multiple sensor integration (Lidar, radar, ultrasonic) aims to enhance the overall robustness of the vehicle, ensuring that it can handle diverse driving conditions and environments effectively.

Phase 2 focuses on establishing secure Vehicle-to-Everything (V2X) communication, which is critical for vehicle platooning and other collaborative driving scenarios. Secure communication ensures that data exchanged between vehicles and infrastructure is protected against cyber threats. This phase also involves the development and implementation of efficient traffic management algorithms, which are essential for optimizing traffic flow and reducing congestion in urban environments.

The final phase, Phase 3, involves the deployment of advanced hardware accelerators (HWA) for machine learning algorithms and the development of custom chips specifically designed for autonomous driving, similar to the Full Self-Driving (FSD) chip of Tesla. These hardware advancements are crucial for enhancing computational efficiency and the overall performance of the autonomous driving system. Additionally, this phase introduces event-based sensors aimed at increasing data throughput, thus improving the vehicle's ability to process information rapidly and accurately.

Each phase builds upon the previous one, creating a comprehensive and scalable approach to developing a fully autonomous driving system. To achieve this, it requires collaboration with different groups within the Department of Electronics Engineering (DESE) and necessitates an interdisciplinary approach. Integrating expertise from various domains such as machine learning, embedded systems, communication, power electronics, systems engineering, security, VLSI & hardware acceleration, and traffic engineering is essential for the successful development and deployment of a fully autonomous driving technology for the future. By incorporating this, the concept of Mobility as a Service can be achieved for enhancing transportation efficiency, reducing traffic congestion, improving passenger safety, and providing on-demand mobility solutions to users.

Bibliography

[1] National Crime Records Bureau. Adsi 2021 full report. https://ncrb.gov.in/sites/default/files/ADSI-2021/ADSI2021FULLREPORT.pdf/, 2021.

[2] National Center for Sustainable Transportation. Menace of road congestion in india. https://www.jatinverma.org/menace-of-road-congestion-in-india/, 2019.

[3] Wall-Y. Self-driving cars are safer than human drivers, study shows. https://www.warpnews.org/transportation/self-driving-cars-are-safer-than-human-drivers-study-shows/, 2023.

[4] Wikipedia. History of self-driving cars. https://en.wikipedia.org/wiki/Self-driving_car/, 2024.

[5] New York Times. Electronic roads called practical. https://www.nytimes.com/1960/06/06/archives/electronic-roads-called-practical-new-system-of-guiding-cars-safely.html/, 1960.

[6] Dean Pomerleau. Alvinn: an autonomous land vehicle in a neural networkl. https://kilthub.cmu.edu/, 1989.

[7] Willie D Jones. Keeping cars from crashing. *IEEE spectrum*, 38(9):40–45, 2001.

[8] John R. Quain. Toward robotic cars. https://cacm.acm.org/research/toward-robotic-cars/, 2010.

[9] Sebastian Thrun. The hype of self-driving cars versus the reality. https://www.aarp.org/auto/trends-technology/self-driving-cars-future/, 2020.

[10] Ben Popper. Guiding light. the billion-dollar widget steering the driverless car industry. https://www.theverge.com/2017/10/18/16491052/velodyne-lidar-mapping-self-driving-car-david-hall-interview/, 2017.

[11] Chris Urmson. Just press go: designing a self-driving vehicle. https://blog.google/alphabet/just-press-go-designing-self-driving/, 2014.

[12] Josh Lowensohn. This is tesla's d: an all-wheel-drive model s with eyes on the road. https://www.theverge.com/2014/10/9/6955357/this-is-tesla-s-d-an-all-wheel-drive-car-with-eyes-on-the-road/, 2014.

[13] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6):2140, 2021.

[14] Farrukh Hafeez, Usman Ullah Sheikh, Nasser Alkhaldi, Hassan Zuhair Al Garni, Zeeshan Ahmad Arfeen, and Saifulnizam A Khalid. Insights and strategies for an autonomous vehicle with a sensor fusion innovation: A fictional outlook. *IEEE access*, 8:135162–135175, 2020.

[15] Valerian Mannoni, Vincent Berg, Stefania Sesia, and Eric Perraud. A comparison of the v2x communication systems: Its-g5 and c-v2x. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–5. IEEE, 2019.

[16] Peraphan Jittrapirom, Valeria Caiati, Anna Maria Feneri, Shima Ebrahimigharehbaghi, María J Alonso-González, and Jishnu Narayan. Mobility as a service: A critical review of definitions, assessments of schemes, and key challenges. *Urban Planning*, 2(2):13–25, 2017.

[17] Warwick Goodall, Tiffany Dovey, Justine Bornstein, and Brett Bonthron. The rise of mobility as a service. *Deloitte Rev*, 20(1):112–129, 2017.

[18] Google Support. How our cars drive. https://support.google.com/waymo/answer/9190838?hl=en/, 2023.

[19] Anthony Navarro, Jendrik Joerdening, Rana Khalil, Aaron Brown, and Zachary Asher. Development of an autonomous vehicle control strategy using a single camera and deep neural networks. Technical report, SAE Technical Paper, 2018.

[20] Buse Pehlivan and Kahraman. Real-time implementation of mini autonomous car based on mobilenet-single shot detector. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 1–6. IEEE, 2020.

[21] Alexandru Constantin Serban, Erik Poll, and Joost Visser. A standard driven software architecture for fully autonomous vehicles. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 120–127. IEEE, 2018.

[22] Synopsys. The 6 levels of vehicle autonomy explained. https://www.synopsys.com/automotive/autonomous-driving-levels.html.

[23] Open Motors. Tabby evo. https://www.openmotors.co/product/tabbyevo/.

[24] Logitech. Logitech c310 digital hd webcam with widescreen hd video calling, hd light correction, noise-reducing mic. https://www.amazon.in/Logitech-Widescreen-Correction-Noise-Reducing-FaceTime/dp/B008QS9KMW?th=1.

[25] Adafruit. Adafruit ultimate gps. https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf, 2024.

[26] Atmel. 8-bit avr microcontroller with 32k bytes in-system programmable flash. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[27] Xilinx. Artix-7 datasheet. https://docs.amd.com/v/u/en-US/ds181_Artix_7_Data_Sheet.

[28] Microchip. dspic33f family data sheet. https://ww1.microchip.com/downloads/en/DeviceDoc.

[29] Raspberry Pi. Raspberry pi 5 datasheet. https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf.

[30] Nvidia. Jetson orin nano developer kit carrier board. https://developer.download.nvidia.com/assets/embedded/secure/jetson/orin_nano/docs/Jetson-Orin-Nano-DevKit-Carrier-Board-Specification_SP.

[31] Haoyu Ren and Ze-Nian Li. Object detection using edge histogram of oriented gradient. In *2014 IEEE international conference on image processing (ICIP)*, pages 4057–4061. IEEE, 2014.

[32] Thao Nguyen, Eun-Ae Park, Jiho Han, Dong-Chul Park, and Soo-Young Min. Object detection using scale invariant feature transform. In *Genetic and Evolutionary Computing: Proceedings of the Seventh International Conference on Genetic and Evolutionary Computing, ICGEC 2013, August 25-27, 2013-Prague, Czech Republic*, pages 65–72. Springer, 2014.

[33] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[34] 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.

[35] Ani Aggarwal. Yolo explained. https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31, 2020.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[37] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[38] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[39] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[40] Glenn Jocher. YOLOv5 by Ultralytics, May 2020.

[41] Sik-Ho Tsang. Brief review: Yolov5 for object detection. https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a, 2023.

[42] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. *arXiv preprint arXiv:2401.10891*, 2024.

[43] LiheYoung. Depth anything: Unleashing the power of large-scale unlabeled data. https://github.com/LiheYoung/Depth-Anything, 2024.

[44] Xuqin Yan and Yanqiang Li. A method of lane edge detection based on canny algorithm. In *2017 Chinese Automation Congress (CAC)*, pages 2120–2124. IEEE, 2017.

[45] Chan Yee Low, Hairi Zamzuri, and Saiful Amri Mazlan. Simple robust road lane detection algorithm. In *2014 5th international conference on intelligent and advanced systems (ICIAS)*, pages 1–4. Ieee, 2014.

[46] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[47] Ze Wang, Weiqiang Ren, and Qiang Qiu. Lanenet: Real-time lane detection networks for autonomous driving. *arXiv preprint arXiv:1807.01726*, 2018.

[48] Zequn Qin, Pengyi Zhang, and Xi Li. Ultra fast deep lane detection with hybrid anchor driven ordinal classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–14, 2022.

[49] ibaiGorordo. Ultrafast lane detection inference pytorch. https://github.com/ibaiGorordo/Ultrafast-Lane-Detection-Inference-Pytorch-, 2022.

[50] Ministry of Road Transport and Highways of India (MORTH). Road accidents in india 2022. https://morth.nic.in/sites/default/files/RA_2022_30_Oct.pdf, 2023.

[51] Sovit Rath. Yolov4 and darknet for pothole detection. https://learnopencv.com/pothole-detection-using-yolov4-and-darknet/, 2022.

[52] Roboflow. Pothole dataset raw. https://public.roboflow.com/object-detection/pothole/1, 2020.

[53] Sonja Nienaber. *Detecting potholes with monocular computer vision: A performance evaluation of techniques*. PhD thesis, Stellenbosch: Stellenbosch University, 2016.

[54] Sampo Kuutti, Saber Fallah, Konstantinos Katsaros, Mehrdad Dianati, Francis Mccullough, and Alexandros Mouzakitis. A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications. *IEEE Internet of Things Journal*, 5(2):829–846, 2018.

[55] Onelap Telematics. How gps works and its applications. https://www.onelap.in/blog/how-gps-works/, 2019.

[56] 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.

[57] Ruinan Chen, Jie Hu, and Wencai Xu. An rrt-dijkstra-based path planning strategy for autonomous vehicles. *Applied Sciences*, 12(23):11982, 2022.

[58] Per Olof Pettersson and Patrick Doherty. Probabilistic roadmap based path planning for an autonomous unmanned helicopter. *Journal of Intelligent & Fuzzy Systems*, 17(4):395–405, 2006.

[59] Aakanksha Vivek Gadagkar and Loganathan Umanand. A novel monocular camera obstacle perception algorithm for real-time assist in autonomous vehicles. In *2020 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS)*, pages 196–201. IEEE, 2020.

[60] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[61] Zhilu Chen and Xinming Huang. End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE intelligent vehicles symposium (IV)*, pages 1856–1860. IEEE, 2017.

[62] Prashanth Viswanath, Soyeb Nagori, Mihir Mody, Manu Mathew, and Pramod Swami. End to end learning based self-driving using jacintonet. In *2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*, pages 1–4. IEEE, 2018.

[63] Sully Chen. End-to-end learning training dataset. https://drive.google.com/file/d/0B-KJCaaF7elleG1RbzVPZWV4Tlk/view.

[64] Aditya Gupta. Self driving car. https://github.com/adityaguptai/Self-Driving-Car-/tree/master, 2018.

[65] Yunxiao Shan, Boli Zheng, Longsheng Chen, Long Chen, and De Chen. A reinforcement learning-based adaptive path tracking approach for autonomous driving. *IEEE Transactions on Vehicular Technology*, 69(10):10581–10595, 2020.

[66] Sen Wang, Daoyuan Jia, and Xinshuo Weng. Deep reinforcement learning for autonomous driving. *arXiv preprint arXiv:1811.11329*, 2018.

[67] Dong Li, Dongbin Zhao, Qichao Zhang, and Yaran Chen. Reinforcement learning and deep learning based lateral control for autonomous driving [application notes]. *IEEE Computational Intelligence Magazine*, 14(2):83–98, 2019.

[68] Ben Lau. Using keras and deep deterministic policy gradient to play torcs. https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html, 2016.

[69] djo10. Deep reinforcement learning - deep deterministic policy gradient - torcs self-driving car. https://github.com/djo10/deep-rl-ddpg-self-driving-car, 2016.

[70] Waveshare. High-precision ad/da board. https://www.waveshare.com/wiki/High-Precision_AD/DA_Board.

[71] Cytron. Md10c r3.0 10amp dc motor driver user's manual rev3.0. https://images-na.ssl-images-amazon.com/images/I/A1TemgvjKjL.pdf.

[72] L Umanand. *Power electronics: essentials and applications*. Wiley India Pvt. Limited, 2009.

[73] Fairchild. Design guidelines for rcd snubber of flyback converters. https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/196/Design-Guidelines-for-RCD-Snubber-of-Flyback-Converters_2D00_Fairchild-AN4147.pdf.

[74] Texas Instruments. Designing switching voltage regulators with the tl494. https://www.ti.com/lit/an/slva001e/slva001e.pdf?ts=1716610728858.

[75] Texas Instruments. Lm138 and lm338 5-amp adjustable regulators - datasheet. https://www.ti.com/lit/ds/symlink/lm338.pdf.

[76] NDTV. Swaayatt robot ceo reveals success in autonomous driving on indian roads. https://www.ndtv.com/india-news/swaayatt-robot-ceo-sanjeev-sharma-reveals-success-in-autonomous-driving-on-indian-roads-5155756, 2024.