VISION DRIVEN LANE CHANGING SYSTEM OF SELF DRIVING CAR

By

Mashook Mohammad Meshkat
ID: 20121006

Shehab Mustafa
ID: 20121009

Tahmid Al Deen
ID: 20121073

Mrinmoy Das
ID: 20121001

A Final Year Design Project (FYDP) submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical and Electronic Engineering

Academic Technical Committee (ATC) Panel Member:

A. H. M. Abdur Rahim, PhD (Chair)

Professor, Department of EEE, BRAC University

Md. Mehedi Hasan Shawon (Member)

Lecturer, Department of EEE, BRAC University

Tasfin Mahmud (Member)

Lecturer, Department of EEE, BRAC University

Department of Electrical and Electronic Engineering
Brac University
December 2023

Declaration

It is hereby declared that

1. The Final Year Design Project (FYDP) submitted is my/our own original work while completing degree at Brac University.
2. The Final Year Design Project (FYDP) does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The Final Year Design Project (FYDP) does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. I/We have acknowledged all main sources of help.

Student's Full Name & Signature:

Mashook Mohammad Meshkat 20121006	**Shehab Mustafa** 20121009
Tahmid Al Deen 20121073	**Mrinmoy Das** 20121001

Approval

The Final Year Design Project (FYDP) titled "Vision Driven Lane Changing System of Self Driving Car" submitted by

1. Mashook Mohammad Meshkat (20121006)
2. Shehab Mustafa (20121009)
3. Tahmid Al Deen (20121073)
4. Mrinmoy Das (20121001)

of FALL, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Bachelor of Science in Electrical and Electronic Engineering on 14-12-2023.

Examining Committee:

Academic Technical Committee (ATC):

(Chair)

_____

A. H. M. Abdur Rahim, PhD

Professor, Department of EEE,

BRAC University

Final Year Design Project Coordination Committee:

(Chair)

_____

Abu S.M. Mohsin, PhD

Associate Professor, Department of EEE,

BRAC University

Department Chair:

_____

Md. Mosaddequr Rahman, PhD

Chairperson and Professor, Department of EEE,

BRAC University

Ethics Statement

While conducting this Final Year Design Project, we needed to do high amount of research. The highest ethical standards have been upheld by FYDP Group 2. Any data or information used in the FYDP has been collected, cited and used in an ethical manner, in accordance with proper guidelines of IEEE. This FYDP by Group 2 has been conducted with honesty, integrity, impartiality and objectivity and any potential conflicts of interest have been disclosed. The sources used have been appropriately cited and acknowledged. Additionally, a Similarity check has been conducted through Turnitin from Brac University by the help of the ATC Panel. The similarity index achieved was 6%. Overall, the ethics statement serves as a clear and transparent declaration of our commitment to conducting the study in a responsible manner and in compliance with the applicable practices and standards set forth by the IEEE.

Abstract/ Executive Summary

Accidents due to driver error persists despite road rules and regulations being updated regularly, mostly because of reckless driving and non-measured actions while lane changing. Therefore, the inclusion of a lane changing system in self-driving cars can greatly reduce these fatal tragedies resulting in safer streets along with an easy mode of transportation for people of all ages and conditions. The system we designed uses image processing which consists of creation of region of interest, perspective transformation, threshold operations, canny edge detections, histogram and various calculations for lane detection and lane maintenance. It further recognizes object detection models to detect vehicles, stop signs and traffic signals using the Haar cascade Deep learning method to make decisions and maneuver our vehicle accordingly. The output of the analysis is deployed on a protype level build with the help of Raspberry Pi 3B+ having integrated connection with Arduino and motor driver system.

Keywords: self-driving car, lane changing, machine learning, object detection, decision criteria, image processing.

Dedication

This Final Year Design Project FYDP is entirely dedicated to the uncountable lives lost due to road accidents. We pay portentous tribute towards all the martyrs whose journeys of life were tragically cut short on the roads.

Acknowledgement

Starting by the name of the Almighty, who has given us the strength and patience to consistently work hard and make the ends meet. We would like to thank our parents for their constant support, consistent encouragement and crucial sacrifices for which we are able to reach this stage in life.

Firstly, we would like to thank our Academic Technical Committee Chair, Professor A. H. M. Abdur Rahim, PhD Sir who had been consistently beside us in all our needs of logistics and support. Special thanks to the Department Chairperson, Professor Md. Mosaddequr Rahman, PhD Sir for assisting us with his invaluable guidance. Moreover, heartfelt gratitude to Abu S.M. Mohsin, PhD Sir for his insightful feedback whenever we reached him for guidance. This endeavor would never have been possible without them. We are as well grateful to Tasfin Mahmud Sir for guiding and supporting us with his indispensable resources in Machine Learning. We would also like to thank Research Assistant, Md. Jobayer for helping us with his vital knowledge in Raspberry Pi.

We would always be indebted to Md. Mehedi Hasan Shawon Sir, who generously provided invaluable expertise and guidance. Special thanks to him for his greatest contribution in motivating us to work beyond our limits. We all pray for his long and fulfilling life.

Last but not the least we would like to thank MD Fiaz Islam Bhuiyan for lending his industry level expertise to broaden our knowledge on the subject matter. Profound thanks to Md. Gholam Mustafa Patwary for sparing his own time to bring us the required components from abroad. Special heartfelt gratitude towards Aunt Raunak Jahan for letting us use her home to work when the University was on Shifting schedule.

Table of Contents

Declaration i

Approval ii

Ethics Statement iii

Abstract/ Executive Summary iv

Dedication v

Acknowledgement vi

Table of Contents vii

List of Tables xii

List of Figures xiii

List of Acronyms xvi

Glossary xvii

Chapter 1: Introduction 1

1.1 Introduction 1

1.1.1 Problem Statement 1

1.1.2 Background Study 1

1.1.3 Literature Gap 3

1.1.4 Relevance to current and future Industry 3

1.2 Objectives, Requirements, Specification and constraints 5

1.2.1 Objectives 5

1.2.2 Functional and Non-functional Requirements 6

1.2.3 Specifications 7

1.2.4 Technical and Non-technical consideration and constraint in design process 9

1.3 Applicable compliance, standards, and codes 10

1.4 Systematic Overview/summary of the proposed project 11

1.5 Conclusion 11

Chapter 2: Project Design Approach 12

2.1 Introduction 12

2.2 Multiple Design Approach 12

2.2.1 Design Approach-1 (Rule-Based) 12
2.2.2 Design Approach-2 (Machine Learning) 13
2.3 Description of multiple design approach 14
2.3.1 Description of Methodology Approach-1 (Rule-Based) 14
2.3.2 Description of Methodology Approach-2 (Machine Learning) (Software) 15
2.3.3 Methodology Approach-2 (Machine Learning) for hardware 20
2.4 Analysis of multiple design approach 25
2.4.1 Analysis of Design-1 (Rule-Based):..... 25
2.4.2 Design-2 (Machine Learning) in CARLA 28
2.5 Conclusion 42
Chapter 3: Use of Modern Engineering and IT Tool 43
3.1 Introduction 43
3.2 Select appropriate engineering and IT tools 43
3.2.1 CARLA Simulator 0.9.14 43
3.2.2 Python 3.7 44
3.2.3 Sublime Text 3 44
3.2.4 Anaconda Prompt 44
3.2.5 Google Collab 44
3.2.6 Proteus 44
3.2.7 Arduino IDE 45
3.2.8 Cascade Trainer GUI. 45
3.2.9 CodeBlocks 45
3.2.10 C++ Programming Language 46
3.2.11 VNC Viewer 46
3.2.12 PuTTY 46
3.2.13 Geany Programming Editor 46
3.2.14 OpenCV 46
3.3 Use of modern engineering and IT tools 46
3.3.1 CARLA Simulator 0.9.14 46

3.3.2 Python 3.7 46
3.3.3 Sublime Text 3 47
3.3.4 Anaconda Prompt 47
3.3.5 Google Collab 47
3.3.6 Proteus 47
3.3.7 Arduino IDE 47
3.3.8 Cascade Trainer GUI. 47
3.3.9 CodeBlocks 48
3.3.10 C++ Programming Language 48
3.3.11 VNC Viewer 48
3.3.12 PuTTY 48
3.3.13 Geany Programming Editor 48
3.3.14 OpenCV 48
3.4 Conclusion 48
Chapter 4: Finding the Optimal Solution and Optimization of the Optimal Solution 49
4.1 Introduction 49
4.2 Identification of optimal design approach 49
4.3 Optimization of optimal design approach 50
4.3.1 Optimization of the optimal design in Software 50
4.3.2 Optimization of the optimal design in Hardware 52
4.4 Performance Evaluation of Developed Solution 52
4.4.1 Performance Evaluation for Software design 52
4.4.2 Performance Evaluation for Hardware design 52
4.5 Conclusion 54
Chapter 5: Completion of Final Design and Validation 55
5.1 Introduction 55
5.2 Completion of Final Design 55
5.2.1 Image Processing 56
5.2.2 Object Detection using Haar Cascade Algorithm 56

5.2.3 Decision Model ……. 57
5.3 Evaluation of the solution to meet desired needs …….. 58
5.3.1 Maintaining Lane ……... 58
5.3.2 U-Turn Operation at Lane End …….. 58
5.3.3 Lane Change operation at car detection …… 61
5.3.4 Traffic Signal operation …… 62
5.3.5 Stop Sign operation …… 63
5.3.5 Performance Analysis …… 64
5.4 Conclusion ……..... 65
Chapter 6: Impact Analysis and Project Sustainability ……..... 66
6.1 Introduction …… 66
6.2 Assess the impact of solution …….. 66
6.2.1 Safety Impact …….. 66
6.2.2 Societal Impact ……...… 66
6.2.3 Environmental Impact …… 67
6.2.4 Economical Impact …… 67
6.2.5 Legal Impact ……... 67
6.3 Evaluate the sustainability …….. 67
6.4 Conclusion …… 68
Chapter 7: Engineering Project Management …… 69
7.1 Introduction ……... 69
7.2 Define, plan and manage engineering project …… 69
7.2.1 Initiation ……..... 69
7.2.2 Planning …… 69
7.2.3 Execution ……...… 70
7.2.4 Monitoring and Control …… 70
7.2.5 Closure ……...… 70
7.3 Gantt Chart/Project Timeline …… 70
7.4 Conclusion …… 71

Chapter 8: Economical Analysis 72
8.1 Introduction 72
8.2 Economic analysis 72
8.2.1 Budget for working prototype 72
8.2.2 Estimated Cost 72
8.3 Cost benefit analysis 74
8.4 Evaluate economic and financial aspects 74
8.5 Conclusion 75
Chapter 9: Ethics and Professional Responsibilities 76
9.1 Introduction 76
9.2 Identify ethical issues and professional responsibility 76
9.2.1 Informed Consent 76
9.2.2 Privacy and Confidentiality Protection 76
9.2.3 Acknowledgement of Proper Sources 76
9.2.4 Approval from Respective Bodies 76
9.3 Apply Ethical Issues and professional responsibility 77
9.3.1 Informed Consent 77
9.3.2 Privacy and Confidentiality Protection 77
9.3.3 Acknowledgement of Proper Sources 77
9.3.4 Approval from Respective Bodies 77
9.4 Conclusion 77
Chapter 10: Conclusion and Future Work 78
10.1 Project summary 78
10.2 Future work 78
Chapter 11: Identification of Complex Engineering Problems and Activities 79
11.1 Attributes of Complex Engineering Problems (EP) 79
11.2 Attributes of Complex Engineering Activities (EA) 79
References 80
Appendix 83

List of Tables:

Table 1. System level specification ……... 7

Table 2. Subsystem level specification ……. 7

Table 3. Applicable codes and standards ……. 10

Table 4. Camera position used in CARLA …….. 16

Table 5. Comparison analysis of multiple designs ……...… 41

Table 6. Reasons why to choose Carla over Udacity …….. 44

Table 7. Comparison Analysis summary of the three multiple design approaches ……. 49

Table 8 Accuracy of correct operation for our prototype car before and after optimization .. 53

Table 9. Accuracy of our self-driving car's operations …….. 64

Table 10. Budget for the Final Design …… 72

Table 11. Estimated cost for the working prototype, first unit and commercial product …… 73

Table 12. Profit Estimation with respect to Number of Companies in Contract ……. 73

Table 13. Profit Estimation from Number of Units Sold per Year …….. 74

Table 14. Attributes of Complex Engineering Problems (EP) …… 79

Table 15. Attributes of Complex Engineering Activities (EA) …… 79

List of Figures:

Fig 1 Picture collected from Synopsys.com depicting the various systems it possesses …….. 4

Fig 2 Design approach for rule-based ……..…… 12

Fig 3 Design approach for machine learning ……..… 13

Fig 4 Methodology for rule-based approach ……... 14

Fig 5 Methodology for machine learning approach in CARLA simulator ……. 15

Fig 6 Collected image samples from CARLA Simulator …… 16

Fig 7 Image sample with annotations of four classes …….. 17

Fig 8 Determining distance (z) between our car and front vehicle …….. 18

Fig 9 Decision Model for machine learning approach in CARLA …….. 18

Fig 10 Changing lane to the left in CARLA …… 19

Fig 11 Changing lane to the right in CARLA ……. 19

Fig 12 Methodology for machine learning approach in hardware system …….. 20

Fig 13 Motor driver with DC motors and wheels in car chassis ……. 20

Fig 14 Arduino UNO and power bank ……. 20

Fig 15 Raspberry Pi and Raspberry Pi Cam module …… 20

Fig 16 Collected image samples from Raspberry Pi camera ……... 22

Fig 17 Cropped positive image samples …….. 23

Fig 18 Decision Model for machine learning approach in hardware …….. 23

Fig 19 Vehicle detects person and stops …….. 25

Fig 20 Vehicle detects no more person and starts to move ……. 25

Fig 21 Vehicle detects a car ahead and stops ……..…… 26

Fig 22 Vehicle moves when front car moves, maintaining some distance of 2 feet …… 26

Fig 23 Vehicle changing direction while maintaining lane ……. 27

Fig 24 Vehicle detects other vehicle in front of it …… 27

Fig 25 Vehicle changing lane …….. 28

Fig 26 Bounding Box applied on images for Vehicles and Motorcycles …… 28

Fig 27 Bounding Box applied on images for Bicycles and Person ……. 29

Fig 28 Mean average precision vs epochs for test-1 …… 29

Fig 29 Box loss, Class loss and Object Loss vs epochs for test-1 ……...… 30

Fig 30 Loss, precision and recall graphs vs epochs for test-1 …….. 30

Fig 31 Showing the Average Precision by class for Validation set results for test-1 …….. 31

Fig 32 Showing the Average Precision by class for Test set results for test-1 …… 31

Fig 33 Mean average precision vs epochs for test-2 …… 31

Fig 34 Box loss, Class loss and Object Loss vs epochs for test-2 ……...… 32

Fig 35 Loss, precision and recall graphs vs epochs for test-2 ……. 32

Fig 36 Showing the Average Precision by class for Validation set results for test-2 …….. 33

Fig 37 Showing the Average Precision by class for Test set results for test-2 …… 33

Fig 38 Vehicle in front but right lane is clear ……. 34

Fig 39 Motorcycle in front but a bit far …… 34

Fig 40 Vehicle detecting front vehicle but still continuing as it is still not within braking distance ……...… 35

Fig 41 Vehicle stops, keeping a minimum braking distance between it and the front vehicle…35

Fig 42 Vehicle stops, keeping a minimum braking distance between it and the front person…36

Fig 43 Vehicle continues as it finds the lane clear of obstacles ……..… 36

Fig 44 Vehicle detects other vehicle in front of it …… 37

Fig 45 Vehicle changing lane …….. 37

Fig 46 Simulation Design in Proteus for Hardware implementation …….. 38

Fig 47 All four wheels and LEDs turn OFF (Car at Halt) ……...… 38

Fig 48 All four wheels moving forward and front LED turn ON (Car moving forward) ….. 39

Fig 49 All 3 wheels, except front right wheel, moving and right LED turn ON (Car moving to the right lane) 39

Fig 50 All 3 wheels, except front left wheel, moving and left LED turn ON (Car moving to the left lane) ……..…… 40

Fig 51 Mean average precision vs epochs for test-3 ……..….. 51

Fig 52 Box loss, Class loss and Object Loss vs epochs for test-3 …….. 51

Fig 53 Loss, precision and recall graphs vs epochs for test-3 ……. 51

Fig 54 Showing the Average Precision by class for Validation set results for test-3 …….. 52

Fig 55 Showing the Average Precision by class for Test set results for test-3 …… 53

Fig 56 Final Design of self-driving car prototype, along with other objects ……..…. 55

Fig 57 Final Design System flowchart ……. 56

Fig 58 Decision model of final design ……. 57

Fig 59 Vehicle having its frame center match with lane center perfectly, with no deviation…58

Fig 60 Vehicle moving forward after getting a deviation value of 0 ……..… 59

Fig 61 Vehicle having a difference of 12 units between its frame center (blue) and lane center (green) ……..... 59

Fig 62 Vehicle having a difference of -21 units between its frame center (blue) and lane center (left green) …….. 60

Fig 63 Vehicle detecting Lane End after reaching a value of 4500 and deciding on taking a U-Turn ……...…. 60

Fig 64 Vehicle taking a U-Turn …….. 61

Fig 65: Vehicle detecting car ahead with bounding box width of 93 pixels …… 61

Fig 66: Vehicle detecting car ahead within 5 to 30 cm and changing lanes to the left …….... 61

Fig 67: Vehicle detecting traffic signal with red light ahead with bounding box width of 87 pixels ……..…. 62

Fig 68 Vehicle detecting red traffic signal ahead within 5 to 20 cm and decides to stop ….. 62

Fig 69 Vehicle no more detecting red traffic signal ahead within 5 to 20 cm and decides to move on ……...… 63

Fig 70 Vehicle detecting stop sign ahead with bounding box width of 90 pixels …….. 63

Fig 71: Vehicle detecting stop sign ahead and decides to stop for 4 seconds before moving on for 1 second ……..... 64

Fig 72 Five Key Factors of Engineering Project Management …….. 69

Fig 73 Gantt Chart/Project Timeline for EEE499P …… 71

Fig 74 Gantt Chart/Project Timeline for EEE499D …… 71

Fig 75 Gantt Chart/Project Timeline for EEE499C ……. 71

List of Acronyms:

AV	Autonomous Vehicle
MPC	Model Predictive Framework
ECU	Electronic controller unit
IoT	Internet of Things
ML	Machine Learning
ADAS	Adaptive Driving Assistance System
GPS	Global Positioning System
V2I	Vehicle-to-infrastructure
V2V	Vehicle-to-vehicle
C-ITS	Cooperative intelligent transport systems
BRTA	Bangladesh Road Transport Authority
IRB	Institutional Review Board

Glossary:

Autonomous Vehicle (AV)	Vehicle which is able to sense its environment and react without the involvement of humans.
Internet of Things (IoT)	A network consisting of connected devices that process and send data through the internet.
Object Detection	It is the entire process of identifying, locating an object or multiple objects in an image or a video.
OpenCV	It is a machine learning library which contains a rich set of tools and its main function is to facilitate the various image processing tasks and machine learning implementations.
GPS Sensor	The full form is Global Positioning System, these signals are radio frequency signals transmitted by a constellation of satellites in Earth's orbit.
mAP	This parameter is used in the field of object detection and image segmentation to employ and evaluate the performance of models by assessing the precision and recall of their predictions across multiple categories or classes.
Image processing	It is the manipulation and analysis of digital images using various algorithms and techniques.

Chapter 1

Introduction

1.1 Introduction

1.1.1 Problem Statement

Statistics suggest that about 94% of the accidents seen on the roads are mainly due to human error [1]. Self-driving cars can be a solution to the problem as automated vehicles can take the decision based on what it 'sees' in its multiple direction which is mainly by the help of a camera and other sensors. In addition, modern automated vehicle systems have been seen to possess multiple other sensors as well which helps to improve the decision making. Moreover, high speed overtaking has been a major cause of accidents. Due to lack of sensors the victims do not even get notified on what obstacles they are surrounded with. Most of the time rough driving not only puts themselves at risk but also risks the lives of the other vehicles surrounding them. Moreover, accidents in squeezed roads due to lack of practice of driving in overcrowded roads have been seen as another problem. In addition, impaired and distracted driving is another major reason for accidents which lead to major fatalities.

Various different literatures have addressed the wide ranges of approaches that Autonomous Vehicles use for their control [3, 4, 5, 6, 7]. Mihalea et al. [3] have stated that driver assistance using an 'end-to-end' approach can be a suggested solution in which the overall process of driving is a learned transformation from image to output. Another reason can be limited transportation access for individuals with driving disabilities, elderly adults, and those lacking driving experience hinders their ability to lead independent lives and access essential services.

1.1.2 Background Study

In this period of modernization, automated vehicles still have a huge path to proceed. Some people are still afraid of how the automated car performs when it is on the road, while some more are anxious about how the software system will function [10]. But despite all these comments, the work of autonomous vehicles is proceeding and so is the lane changing part of it. Statistics say that over 10 percent of the accidents on highways occur due to lane changing operations.

Total kinetic energy minimization approach integrated with solving a polynomial equation can be basically used to determine a trajectory for easy and running lane change [10]. However, the change angle and accuracy is calculated by resolving a time-optimal control problem that decreases runtime with fixed end constraints and bounds on lateral speeding up and lurch.

The idea of lane changing or shifting the car to the adjacent lanes comes with ideas based on a set of rules, which includes:
1. subjective rules
2. functions of the positions
3. relative velocities of surrounding vehicles

In the paper “A machine learning approach for personalized autonomous lane change initiation and control” by Vallon, an approach is established where the characteristics of a driver are being tried to imitate [10]. The main ideology behind this is the dataset which leads to calling the model a data-driven model.

Data were basically collected in situations of complexity and moments when the decision making is quite difficult, whether or not to change the lane.

Which led to the generation of Model Predictive Framework (MPC) where the certain decision logic was being integrated with more autonomous vehicles in order to test for safety, reliability and resilience.

Another trajectory idea is the learning-based trajectory where the dataset is nothing but real live driving data [10]. K-nearest instance is being used to basically establish a lane changing method by machine learning where the datasets are obtained from online. The main motive of so much experimentation is that the lane changing and particularly roads become a safer place.

The primary purpose of the trajectory planning algorithm is to generate a smooth trajectory, while ensuring safety and following the global route. As illustrated in Fig.1, the proposed method can be decomposed into four portions. First, the center line is constructed by a spline polynomial, which is based on a batch of center waypoints on the road (see Fig. 2).

There are other methods such as real-time trajectory planning [8]. This algorithm basically works on an optimal trajectory accompanying the speed for autonomous driving as well as the acceleration. Predesigned center waypoints are the initial approach as they are used as the reference line. Moreover, arc length and offset to the center line in the Frenet coordinate system is a huge contributor in producing the path candidate, also the polynomial function is being used to discretize the speed space [8].

The trajectory planning algorithm is divided in 4 portions [8]. Firstly, spline polynomials are used in which there are centerlines being manufactured on the road. After that the vehicle existence can be understood by using pinpointed coordinate transformations, a path planning algorithm being developed is used to produce the path candidates at which a range of lateral offsets are being sampled along the center line considering vehicle speed and heading direction [8]. Finally, the optimal path is being picked by decreasing the cost function of the previous path candidates. Then the speed generation algorithm is being seen to be very close to that of the path generation algorithm. Sample of 100 Hz is used as the sampling frequency in the trajectory planning approach which in turn overhauls the trajectory in real time. In a nutshell, optimization of safety as well as better energy efficiency was the main objective for the dynamic trajectory system.

In this paper, a dynamic trajectory planning algorithm is put forward to avoid the static and moving obstacles while ensuring safety, comfort and energy consumption reduction. The presented approach first uses the cubic spline to construct a center line based on traffic conditions.

Another process incorporates the motion planning modules [9]. The full process consists of 2 steps. Initially it uses the start position, end position as well as the map information as the input

to basically select the bestest route that has the least traffic congestion. Changhao. J et al. [9] named this step as route planning.

Secondly, due to its geometric nature as a list of points, the global route may not adhere to the constraints of the vehicle dynamics model and lacks speed information. The local motion planning modules, which we will refer to as trajectory planning in this article, take as inputs the global route, the ego vehicle's current position supplied by the localization modules, the position of static obstacles supplied by the perception modules, and the trajectory possibility distribution of dynamic obstacles supplied by the prediction modules. They then output the trajectory to the control modules over a T time step horizon. Figure 1 illustrates how the motion planning components are organized.

1.1.3 Literature Gap

Literature resources on Autonomous Vehicle (AV) have been seen to work on prototype level. However, we have faced difficulty to manage resources from industry level which is highly confidential due patents and other legal requirements.

While conducting our background research we came across the realization that our literature resources regarding our project only show theoretical and experimental approaches. From marketing advertising to media presentation we can see most major car manufacturing companies have varying degrees of self-driving features. Majority claim to have SAE level 2 automations, some even reaching Level 3, however, we lack the in-depth knowledge of these vehicles. This lack of industry level knowledge regarding our project can be attributed to proprietary technologies and methodologies which are subject to high confidentiality due to patents and trademarks. Thus, we have faced great difficulty to manage any resource from the industry level due to these confidential legal requirements.

1.1.4 Relevance to current and future Industry

The modern industry has been seen to use multiple solutions. One being better than the other. ADAS is an algorithm where all the cars are connected with each other via the Internet of Things (IoT) then comes the Connected Vehicles where the connected vehicles use vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) where the cars basically share information regarding how much traffic some other part of the city has [11, 12]. As well as the most common solution of modern times, Autonomous Vehicles (AV) as they have been witnessed by modern industries to significantly reduce the risk of accidents caused by impaired or distracted driving. By removing the human element from driving, AVs can eliminate the risk of driver error altogether.

The car industry has invested a lot in putting safety precautions for their users. They have been seen to incorporate car safety features such as three-point seatbelts, airbags, and shatter-resistant glass. These precautionary measures have been taken just to prioritize passive safety measures to minimize injuries during an accident on the roads. This ADAS system has been seen to actively participate in minimizing the numbers of occupant injuries and collisions during an accident with the aid of embedded systems [11].

The use of multiple sensors fused with object detection, identification as well as image processing with the help of camera systems connected actively with the car have been seen as a common practice in the car industry. Finally using all the parameters and merging them using the novel AI. Multiple image/ object recognition devices are being used in the current time. The most common one of them are the radar, lidar, ultrasound sensors and ultrasonic sensors finally using a specific algorithm for sensor fusion. These are the common parameters used by the ADAS algorithm [11]. The image recognition software as well uses these parameters similar to how the human brain processes information by combining a lot of data. The main motive of this technology is to react even quicker than a human driver, which in turn reduces the reaction time, therefore in modern times, it is possible to react quickly by a machine to simultaneously by streaming video, object recognition, as well as planning the response and reacting to it.

Here are some of the most common ADAS applications:

Adaptive Cruise Control: This specific technology is useful for highway driving [12]. While driving for long hours it is very much difficult to constantly maintain and keep an eye on the speed as well as how other vehicles are approaching. So this technology judges these parameters and tunes accordingly as a result helps the car automatically accelerate, decelerate, and even stop in fatal situations [12]. The main advantage when equipped with this technology is it will help to maintain an appropriate distance with other vehicles in all of its directions. This space-aware feature will specifically help the driver from being prevented from a mishap happening especially in situations where blocked vision or vehicles are really close to it.

Glare-Free High Beam and Pixel Light: This feature uses the sensors to sense and tune its glare-free high beam and pixel light when the car is in darkness. One common problem of driving on highways at night is momentary blinding, this feature basically works on that issue. It helps to detect the light of the vehicle approaching from the opposite and diverts the vehicle's light away.

Some other features are the Adaptive Light Control, Automatic Parking, Autonomous Valet Parking, Navigation System, Night Vision, Unseen Area Monitoring, Automatic Emergency Braking, Crosswind Stabilization, Driver Drowsiness Detection, Driver Monitoring System, 5G and V2X,

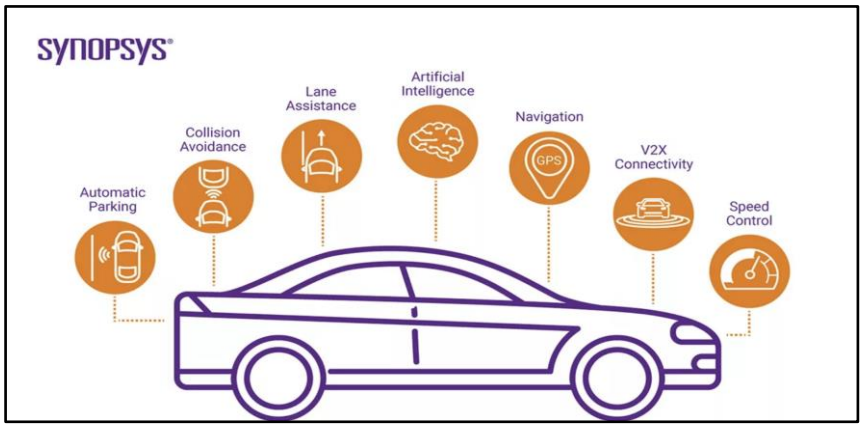

Fig 1: Picture collected from synopsys.com depicting the various systems it possesses.

In the recent world there has been an increased demand for the automobile electronic software and hardware necessitates. The manufacturer has to ensure the enhanced reliability, reduced expenses, and shorter development cycles.A developed and integrated ADAS Electronic controller unit (ECUs) has been seen to be in demand putting the scattered ADAS domain controller at less demand [13].

Connected Vehicles on the other hand is a term which refers to services, technologies and apps that pair up a car to the environment. As the AUTO Connected Car News suggests, a connected car can be termed any vehicle which has the necessary equipment needed to connect to the equipment of the other cars, notably the networks, devices, services, apps which are outside the vehicle. The applications used can range from the safety sensors, efficiency, traffic safety feature as well as can be an infotainment system, road assistance, parking assistance as well as Global Positioning System (GPS) [13]. Most commonly vehicles need to have two certain features to be connected which are the cooperative intelligent transport systems (C-ITS) and the advanced driver-assistance systems (ADASs). The safety application which connected vehicles possess is being designed to escalate situation awareness to achieve the objective of mitigating traffic through vehicle-to-infrastructure (V2I) communications and vehicle-to-vehicle (V2V) all by the help of sensors, vehicle data networks as well as camera technology

1.2 Objectives, Requirements, Specification and constraints

1.2.1. Objectives

The objectives are as follows:

1. Detecting lanes, vehicles, traffic signals and road signs present in real time driving conditions.
2. Analyzing the parameters obtained and taking appropriate decisions from the object detection model.
3. Advancing the decision taken to the vehicle maneuver system.

By integrating cameras for the use of image processing and object detection models for detecting the specific parameters such as red and green traffic signal, stop sign as well as the surrounding vehicles and the lanes, our system will be able to make specific decisions on whether to change the lane or to be present on the same lane, whether to accelerate, decelerate or to maintain that constant speed, specifically intended to reduce the number of accidents caused due to the changing of lanes all across the world. This system also helps to minimize the number of accidents occurring due to human error, thus reducing human error while driving. In addition, Haar Cascade Deep Learning technique is used in order for the identification model for the specified parameters. Then incorporating the output in our prototype level using a Raspberry Pi 3B+ which has integrated connections with Arduino and motor driver system. Thus, improved safety on the road for all as well as better maintenance of road rules.

1.2.2 Functional and Nonfunctional Requirements

Functional requirements:

1. Object detection: Our designed system will be able to detect vehicles and people, and find out the distance between it and them. along with the detection of traffic signals. If the signal is red, it stops. If it is green, it continues [8]. Along with these detections, our system will specifically be able to detect stop signs that accompany the road. For the detection of the stop sign, our vehicle will come to a stop and wait for 4 seconds before continuing on its course.
2. Braking distance: Our system will slow down and accelerate the car according to the situation it detects. However, for coming to a complete halt, our vehicle should maintain a distance of 2 feet (5 to 15 cm in prototype model) from the car/ traffic signal or stop sign ahead with a braking time of 3 seconds (1 second in prototype model).
3. Operating velocity: Depending on the situation our system detects ahead of itself, it can change its velocity accordingly. If there is nothing ahead of itself, it can accelerate. If there is another car in front of it that is moving, it will move with a constant velocity. If there is a car in front which is at a halt, our system will slow down and come to halt. Velocity will be different for different curved lane maintaining and lane changing, and our car will do it accordingly.
4. Lane change: If the car ahead is detected closeby for more than 2 seconds, our vehicle will check the adjacent lanes. If any of the adjacent lanes are free, the system will make a decision to change lanes.

Non-functional requirements:

- Navigation system: Our automated car system will be equipped with a GPS system that will allow the passenger to select their preferred route.
- Security system: As the system can collect data from one or several sensors, it is paramount that there is a security system in place to protect these data.
- Maintainability: The system should be easy to maintain, update and repair when required.
- User experience: The system should provide seamless and enjoyable interference for easy communication with the passengers.

1.2.3 Specifications

Project Specification:

Table 1: System level specification

System	**Project Specification**
Object detection	For the detection of the obstacles: 1. Detect vehicles and people (**stop** and check again) 2. Stop sign (**stop** and wait for **4 seconds**) **(In Hardware)**
Lane change	If obstacles is found ahead, the vehicle system will check the adjacent lanes: 1. If any of the **adjacent lanes are free** 2. The system will make a decision to **change lanes**.
Operating velocity and distance maintained	Maintain specific distance between adjacent vehicles: 1. running at **30km/h** maintains a **distance of 4m**, with braking **distance of 2 feet.** Hardware prototype model will have smaller parameters
Braking distance	**Slow down** and **accelerate** the car according to the situation it detects. 1. Will **calculate** the **braking distance** for its **instantaneous speed** according to the object ahead 2. When it **completely** stops, it maintains **1m distance** (5 to 15 cm in prototype model)

Component Specification:

Table 2: Subsystem Level Specification

Sub System	**Tentative Components**	**Tentative Component Specifications**
Object Detection	Raspberry Pi Camera Module with cable	Dimension: ~25 × 24 × 9 mm Weight: 3g Video mode: 1080p47, 1640 × 1232p41 and 640 × 480p206 @ 8MP Sensor: Sony IMX219 @3280 × 2464 pixels Depth of field: ~ 10 cm to ∞ Focal length: 3.04 mm Cable length: 300mm [15]

Processing Unit	Raspberry Pi 3 B+ with case	Clock Cycle: 1.4GHz SDRAM: 1GB LPDDR2 IEEE standard: 802.11.b/g/n/ac Wireless connectivity: ● 2.4GHzand 5GHz wireless LAN ● Bluetooth 4.2 ● BLE Connectivity: ● Gigabit Ethernet over USB 2.0 (maximum 300 Mbps), ● 4 USB 2.0 ports ● CSI camera port ● DSI display port Terminals: 40-pin GPIO header Storage: Micro SD port Operational Voltage: 5V DC current: 2.5A [16]
	Ultra 16GB Micro SDHC UHS-I	Capacity: 16GB Read speed: ~98 MB/s Dimensions: 0.04" x 0.59" x 0.43" [17]
Power Supply	Meko Power Bank	Capacity: 10000mAh Output: Double USB Input: Micro USB & Type-C [18]
	Battery with charger	Battery: ● Capacity: 750-1300 mAh ● Output Voltage: 3.7V Charger: ● Input AC: 100-240V @47-63Hz ● Output DC: 3.7V @500mA ● Terminal Voltage:4.2V±1% [19]
Vehicle Maneuver	4 Wheel 2 Layer Robot Smart Car Chassis Kits with Speed Encoder	Body: 2 layered acrylic chassis with screw and supportive parts Motor: ● Rated Voltage: 3-12 V DC ● Unloaded speed: 120 RPM. ● Load current: 190 mA (250 mA MAX) ● maximum torque: 800 gf. Cm min. Wheel dimensions: 30 x 65 mm [21]

	Arduino Uno R3	Clock cycle: 16MHz Memory: 2KB SRAM, 32KB FLASH, 1KB EEPROM Built-in LED Pin: 13 Digital I/O Pins: 14 Analog input Pins: 6 PWM Pins: 6 I/O Voltage: 5V Input voltage (nominal): 7-12V DC Current per I/O Pin: 20 mA Dimensions: 53.4 x 68.6 mm [22]
	L298N H-Bridge Dual Motor Driver	Motor Supply Voltage (Maximum): 46V Motor Supply Current (Maximum): 2A Logic Voltage: 5V Driver Voltage: 5-35V Driver Current:2A Logical Current:0-36mA Maximum Power (W): 25W [23]

Several components along with some specific parameters and descriptions have been categorized into some subsystems in Table. The object detection sub-system detects the environment for the main system and relays it to the processing unit which processes the information to make decisions for the action of the vehicle. These two subsystems mainly compose the main system. The action of the vehicle is dictated by the vehicle maneuver sub-system which is completely customizable to stakeholders' needs. To power all the subsystems we require a power supply subsystem which is directly correlated to the vehicle maneuver subsystem. The specifications of these two subsystems are just a placeholder and customizable.

1.2.4 Technical and Non-technical consideration and constraint in design process

1. Implementing this self-driving car can increase the traffic congestion that is already present in countries like Bangladesh.
2. If a system or subsystem fails somehow, it may lead to a serious accident and so, we need a backup system and an emergency system to shut down all operations of the car in case any crucial system fails.
3. The sensors will affect the health of nearby people negatively, as they will emit radiation and so, the amount of sensors used should be as low as possible.
4. Time is another constraint for this project. We were bound to complete the project in one year.
5. We also have financial constraints as the project needs to be implemented within a certain budget. We have done a detailed cost analysis for the whole system.
6. Regulatory challenges: Self-driving cars are subject to a complex web of regulations at the local, state, and federal levels. Regulations must be

developed to ensure the safety and reliability of self-driving vehicles, while also addressing liability and insurance issues.

1.3 Applicable compliance, standards, and codes

Table 3. Applicable codes and standards

Device & Technology	Standards & Code	Description	Application
Practice	SAE J3016	The Society of Automotive Engineers have developed a report explaining six levels of driving automation from level 0 to level 5.It also describes the systems and role of the drivers in each system [24].	Objective of project
	IEEE 2846	This standard measures the performance of autonomous vehicles by providing a set of metrics for analyzing the safety and reliability of ADS. It also measures the performance, functionality, and robustness of the system [25].	Validation parameters
	ISO/PAS 21448	This is a new developed standard which also describes the safety issues of autonomous vehicles. It includes factors such as hazard identification, safety validation, risk assessment and provides a backbone for the safety of ADS [26].	Validation process
IoT	IEEE P1451.99	Explains how to share data, make sensors compatible, and send wireless network messages in communication technology [27].	User interface with system.
Software Simulator	CARLA simulators	This is an open-source simulator that provides a virtual environment to researchers for testing the performance of different algorithms and safety of autonomous vehicles. It is also used to evaluate many scenarios which may provide life threatening consequences to replicate in real life [28].	Analyze multiple approaches.
Sensor	IEEE 2700-2017	Common framework code concerning sensor performance specifications, units, conditions and limits [29].	Object detection camera.
Neural Network	ISO/IEC TR 24029-1:2021	Standards to evaluate the robustness of neural networks [30].	Applicable for second approach

Body	ISO 26262	This code stands for the international standard for functional safety in road vehicles. This code provides a structure for designing and testing for the safety of electrical systems in passenger vehicles [31]. It covers all aspects of safety like software and hardware designs and all types of validations.	For the construction of vehicle maneuver subsystem

1.4 Systematic Overview/summary of the proposed project

The problem of road accidents has been decreasing but not eradicating. This matter can only be addressed keeping a certain fact in mind that the majority of car accidents occur due to human error and lack of driving experience. Keeping certain scenarios and statistics in mind a potential solution to this problem is autonomous vehicles or self-driving cars.

The project we design focuses on a specific part, the lane-changing mechanism of self-driving cars. We have developed a system by training it with specific parameters; that is lanes, traffic signals, stop signs, and of course the other neighboring vehicles. The system uses image processing, object detection, decision model, and vehicle maneuver model. The system has been deployed in Raspberry Pi 3B+ and tested through a prototype. Further, it has been optimized to make the results better, aspiring to make the roads a safer place than before.

1.5 Conclusion

In a nutshell, self-driving cars can be stated as the potential solution to mitigate traffic rules negligence as well as the massive accidents occurring worldwide. By going through an enormous range of literature, we came across the matter that ADAS is the system being used currently. However, there are still probable solutions under development, of which the approach we are working on might be an exemplary one.

Chapter 2

Project Design Approach

2.1 Introduction

According to the objectives defined in Chapter 1, 2 unique designs were analyzed. The first design consists of a rule-based approach whereas the second design makes use of a machine learning approach, which itself has two types of approach, one in designing a self-driving car in CARLA simulator, and another in real-life using a prototype model of a self-driving car with the help of raspberry pi, camera, arduino, and motors.

2.2 Identify multiple design approach

2.2.1 Design Approach-1 (Rule-Based)

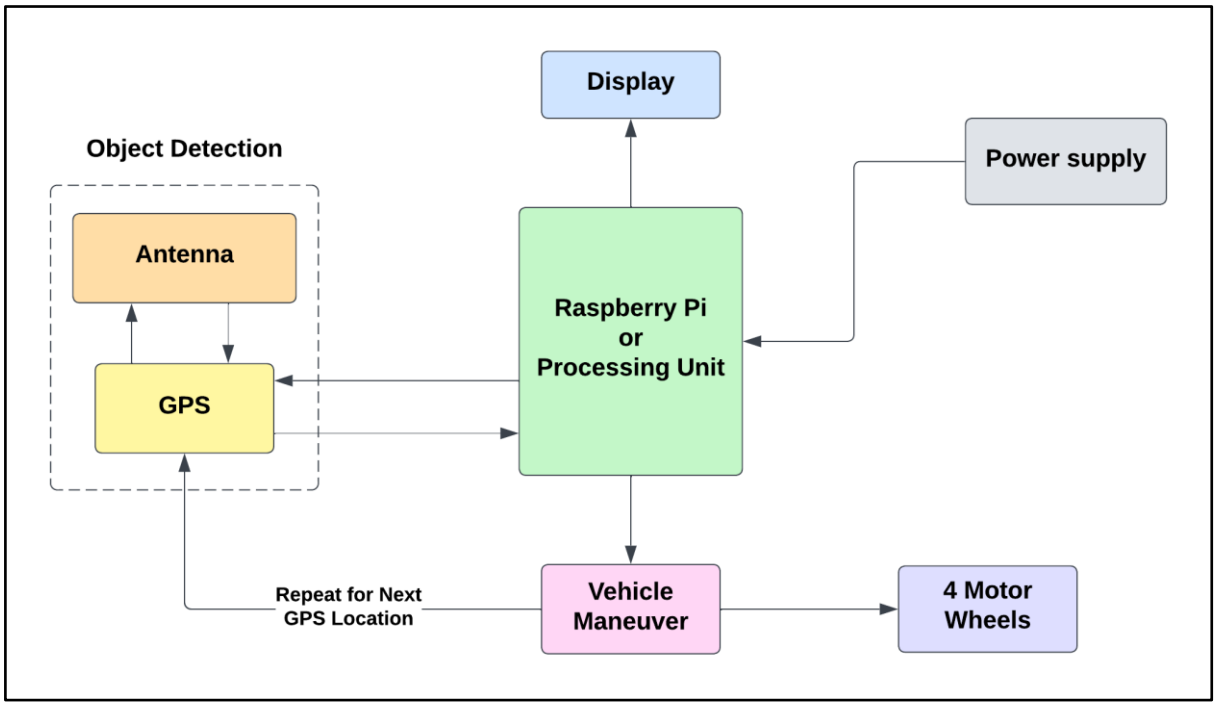

Fig 2: Design approach for rule-based

Our first approach is rule-based, meaning all the actions of our systems are dictated by the information of the system and our preferred thresholds. It mainly consists of three primary systems. They are: object detection, processing unit and vehicle maneuver. All the systems communicate with each other through the processing unit or Raspberry Pi. The object detection system mainly contains a GPS module along with an antenna. The lane changing system, operating velocity, and braking system, all are maintained in the vehicle maneuver system which drives four DC motors with the help of a motor driver, to represent the wheels of our vehicles.

2.2.2 Design Approach-2 (Machine Learning)

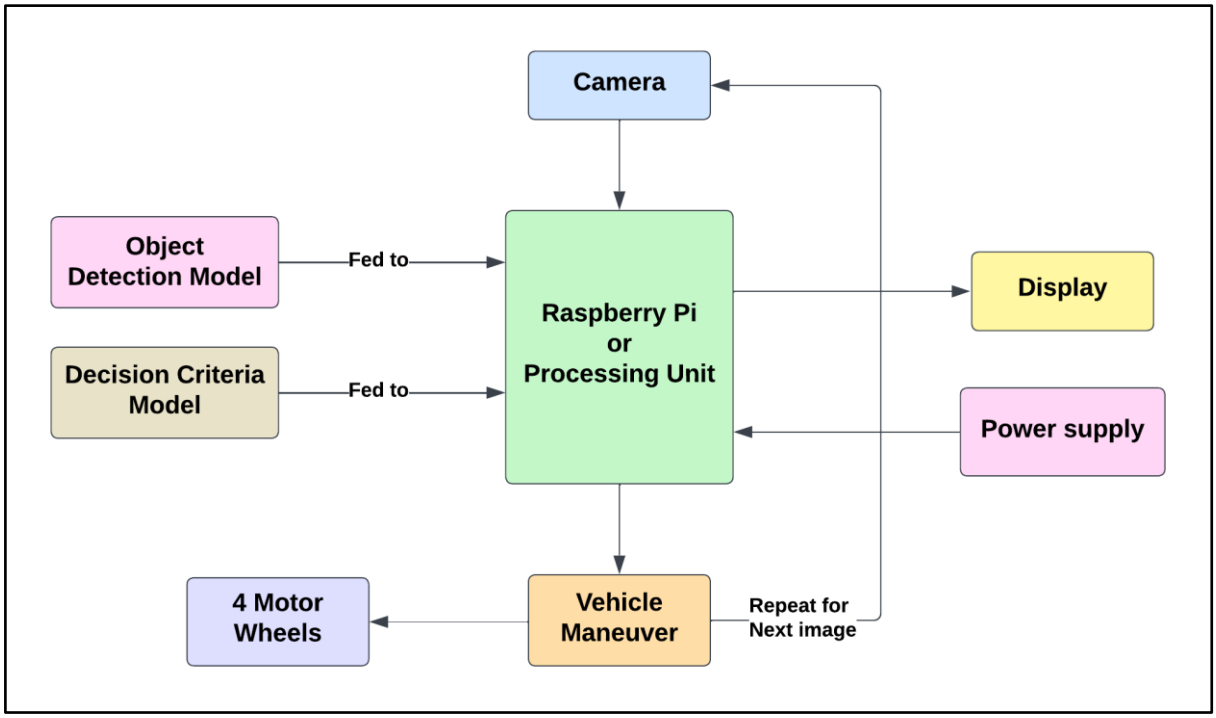

Fig 3: Design approach for machine learning

Our second approach is consists of mainly four different systems. The systems are: object detection, lane changing, operating velocity and braking systems. Each of the systems are connected to each other through the processing unit which consists of a Raspberry Pi. The object detection system consists of a camera. The lane changing and operating velocity system is composed of the motor driver accompanied with four DC motors to represent the wheels of our vehicles. The four wheels also have to be fitted with brake pads to make up the braking system. All the components will be powered up by a battery of sufficient capacity.

2.3 Description of multiple design approach

2.3.1 Description of Methodology Approach-1 (Rule-Based)

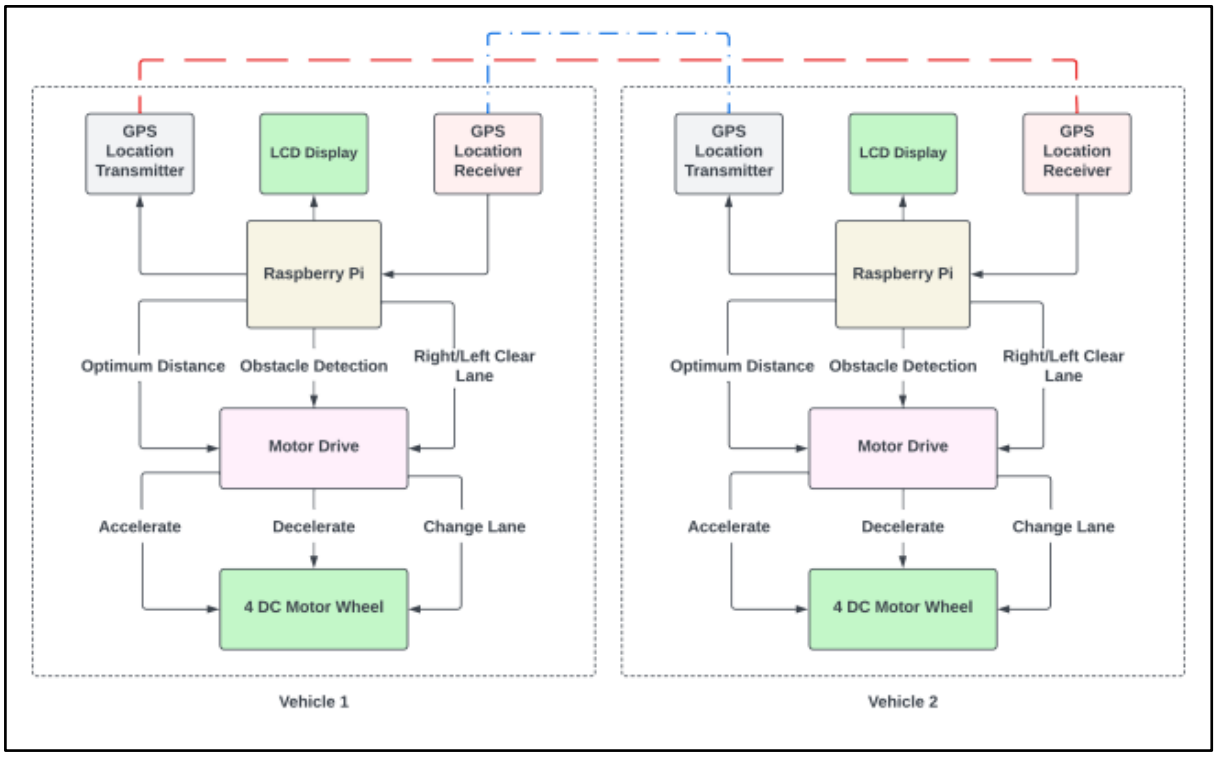

Fig 4: Methodology for rule-based approach

The methodology of our first approach begins with the object detection system. This detection is done by GPS location tracking using GPS modules. This GPS information is collected by the Raspberry Pi, which already has our set rules of instructions on the thresholds that it has to maintain. If the distance between the agent and the detected object is greater than the threshold of optimum distance, the motor drive accelerates the four DC motors. However, if the distance is less than the optimum threshold, the agent decelerates. Lastly, if an object is detected to be halted at a location for too long, the agent resorts to the GPS information of vehicles from adjacent lanes. If one of them is free, the agent will change lanes. The vital importance of this approach is that the agent is completely dependent on the surrounding vehicles and their ability to communicate with the agent.

2.3.2 Description of Methodology Approach-2 (Machine Learning) for Software

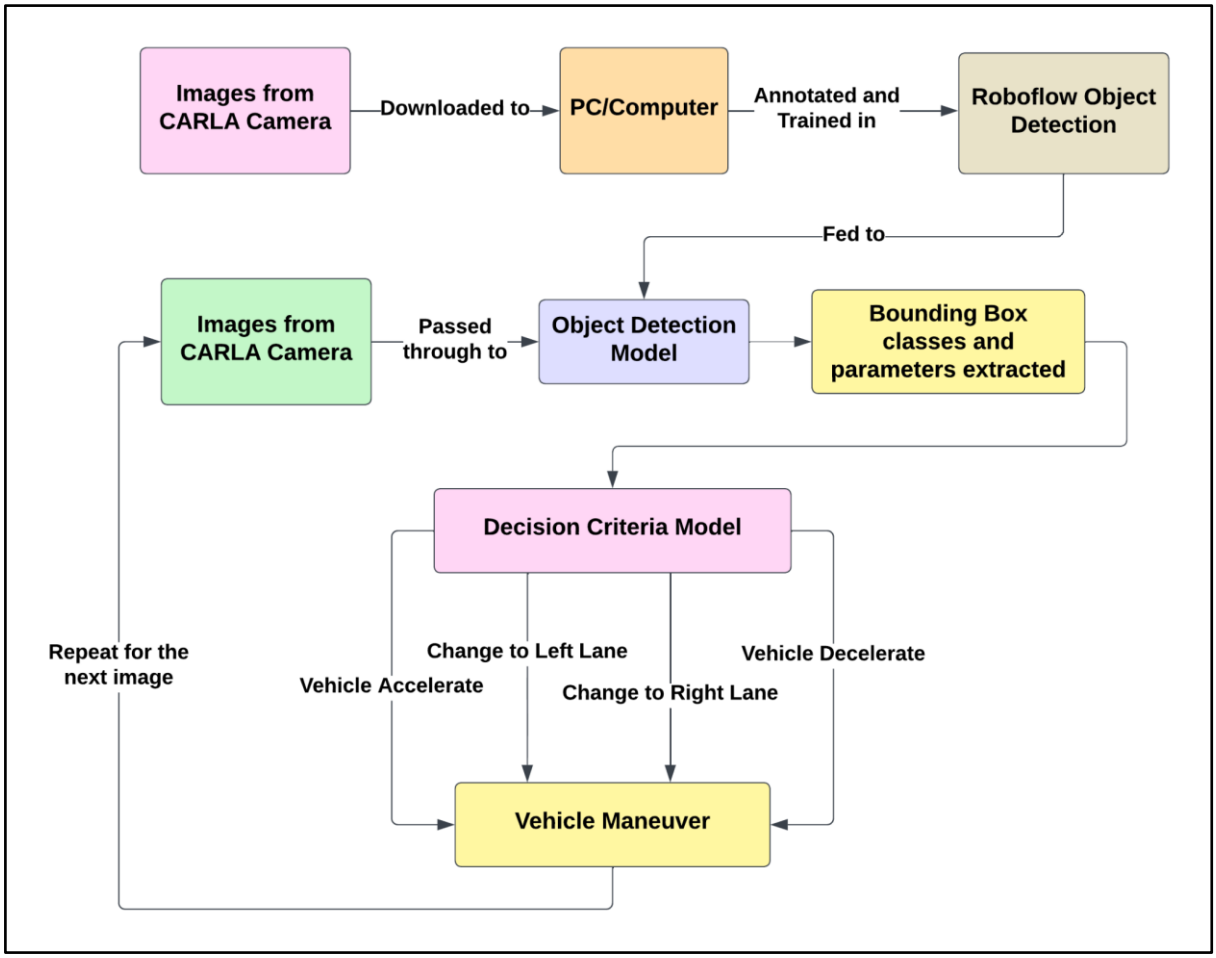

Fig 5: Methodology for machine learning approach in CARLA simulator

For our second approach, we devised a machine learning methodology for the CARLA simulator, where we showed the steps necessary to build a self-driving car system in the CARLA simulator. The focus is based on the simulation of the main models required for the system to function properly. Initially, information of the surroundings is taken into account with the help of a camera from our car in CARLA. These are displayed in an image window and downloaded as images using our python code in sublime text, which are then used to annotate and train our machine-learning model in Roboflow. After training and loading the model, we can then instead pass images from our car's camera in CARLA through the trained object detection model where information such as the position of detected objects and their distance from the objects are to be extracted with the help of bounding boxes. This information is passed through to the decision criteria model which determines the action our car has to take according to its surroundings. Based on the decision received, our vehicle maneuvers accordingly to the left lane, right lane, or straight. If there is more distance than optimum distance between our car and the detected object in front, it accelerates. If the detected object is close enough, our car decelerates. The car also takes information from adjacent lanes, and if they are clear of objects, then it changes lanes accordingly. Our focus is to train and test the models in Roboflow to be able to react to the simulated environment. The performance of the self-driving car depends on the robustness of the training of the models.

We implemented our machine learning based lane changing system of self-driving cars in CARLA Simulator 0.9.14. The programming language we used was Python 3.7 that is highly suitable for usage for CARLA. It was implemented in sublime text 3 which is a type of shareware text and source code editor. To implement our code from sublime text in the CARLA simulator and run our models, we used anaconda prompt from anaconda which allowed us to connect with CARLA and interact with Python and Anaconda packages through the command line. It provided a way to manage Python environments, install packages, and run sublime text files.

Numerous simple-to-use sensors are available from CARLA, including open-source Python APIs for RGB cameras and depth cameras . It is an easy plug and play addition of these sensors to get desired data without the consideration of underlying drivers of the sensors. We added only one RGB camera to our self-driving car to collect images with the following position parameters as following:

Table 4: Camera Position used in CARLA

x	y	z	fov
2.5	0	0.7	110

In Table 1, x stands for the camera's front and back positions in relation to the vehicle, y for its left and right positions in relation to the vehicle, and z for its vertical position in relation to the vehicle. Lastly, fov stands for field of view which indicates the area that can be seen by the camera at a given moment [5]. We manually took photos from the attached RGB camera while our car was in various scenarios within the simulated environment. We chose this process to avoid any image cleaning process where we had to discard images we could not use due to similarity. In this manner, we collected various images at the resolution of 640*480 keeping the standard size. Examples of collected images for the dataset are in Fig. 6.

Fig 6: Collected image samples from CARLA Simulator

After the image acquisition process, we annotate the image dataset using roboflow. We labeled a total of 1500 images into four classes: vehicle, person, motorcycle and bicycle. Our images contained bounding boxes for single class, muti-classes as well as occluded single and multi-classes. The annotation had to be done manually using roboflow which results in the generation of text, CSV or XML file of the bounding box for the dataset. Not only this, roboflow also splits the dataset into training, validation and testing sets which are fed into the object detection model for further operation. Examples of annotated images are shown in Fig. 7.

Fig 7: Image sample with annotations of four classes

For our object detection model, we used the Roboflow 3.0 training algorithm to train, test, and validate our model and further use its output image's information as input to the decision criteria model.

After extracting our dataset images from the CARLA simulator and manually applying bounding boxes on images from CARLA for 4 classes, we then trained our model using the Roboflow 3.0 object detection algorithm in roboflow. As there are no or very few stop signs or pedestrian signs in CARLA, we are omitting them here and not including them in our object detection model as they are not necessary for lane-changing mechanisms. Therefore, we are only considering the objects that are on the streets to implement our lane-changing system.

Decision Model:

As we already know, our decision model is needed for our self-driving car to decide whether it will accelerate forward, or change lane to right or left, or decelerate. Our object detection model discussed in the previous section takes in an input image from the front of the car and can give out an output image with bounding boxes, along with the class detected, boxes' x and y pixel coordinates of the 640x480 image and the width and height of the boxes in pixel. Using these parameters, we made a decision model which helps our car to detect whether a vehicle, person, motorcycle, or bicycle is in front of us, or on the left, or on the right lane and take decisions of acceleration, lane changing, or deceleration accordingly.

Equation 1 was used to find the distance between our car and the object in the front, using the 'y' and height parameter from object detection bounding box, also shown as an example in Fig. 8 as well:

$$z = a - value\ of\ y - (\frac{value\ of\ height}{2})(1)$$

where a = total number of pixels in y dimension (480 for us)

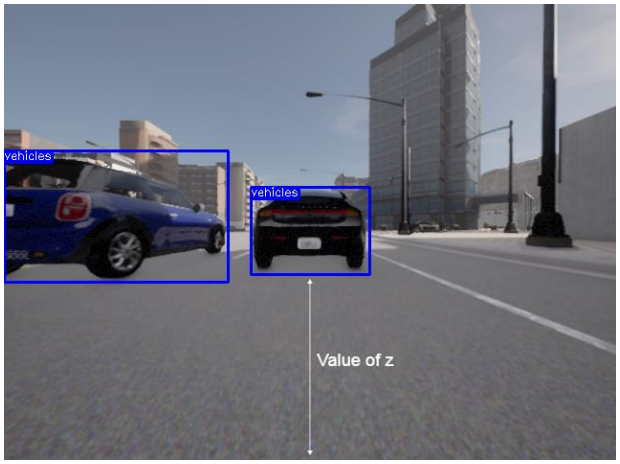

Fig 8: Determining distance (z) between our car and front vehicle

We have considered an algorithm for this model after much trial and error, and considerations. We first call our object detection model from Roboflow and use it, setting 40% confidence and 30% overlap, on images that our car displays in CARLA continuously, as shown in Fig. 8.

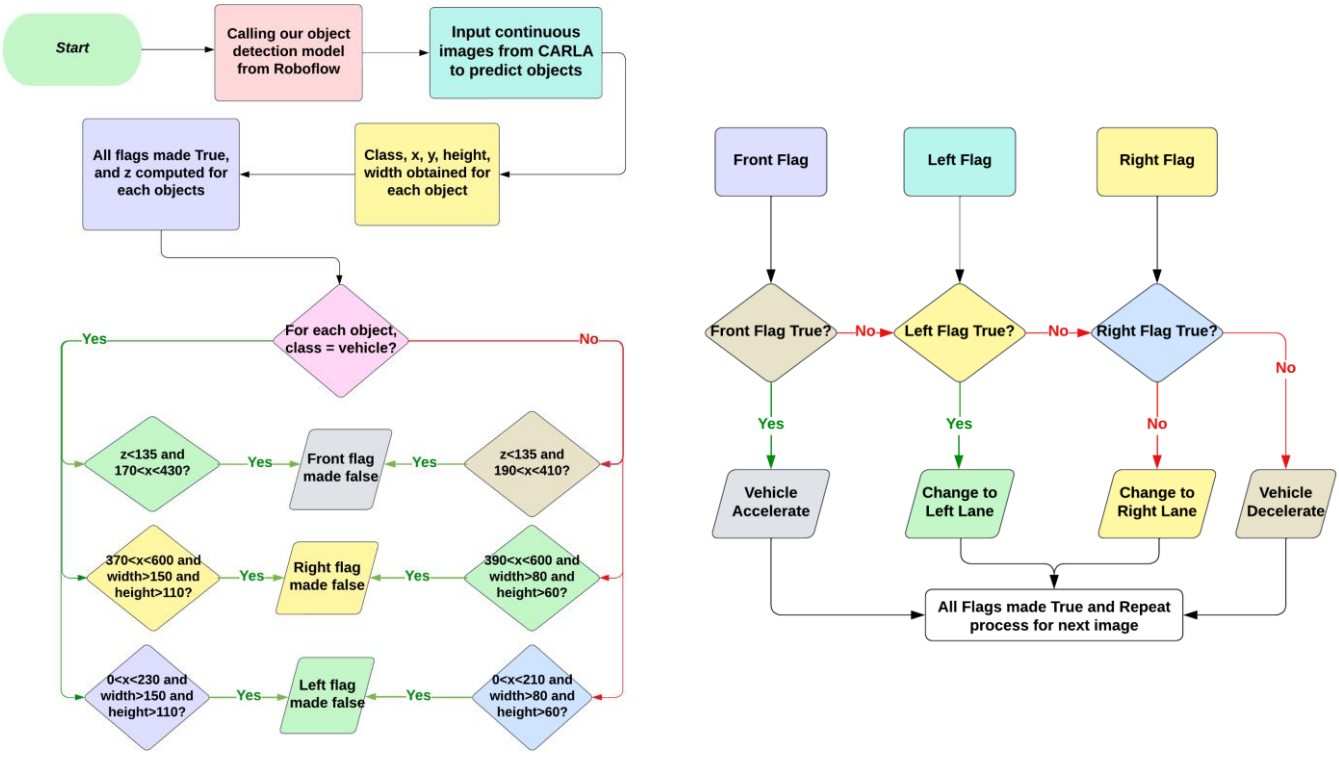

Fig 9: Decision Model for machine learning approach in CARLA

For **every image**, we ran the algorithm again and again in a loop to give decisions continuously, assuming the car is in the right or left lane at the very beginning. The left or right lane flags are also made false if our car is already in the right or left lane respectively, as it cannot go further right or further left.

It decides its action using the algorithm from Fig. 9, comparing the 3 flag variables for every image. Then all three flags are made true and the whole algorithm is run for the next image as well, and a new decision can be made for that image. This goes on again and again.

To indicate an object is really close in the front lane, we considered a value of z to be less than 135 pixels. To indicate a close object in the left lane or right lane, we have considered a larger value of width for vehicles compared to other classes i.e. person, motorcycle, and bicycle, as vehicles are generally wider. On the other hand, a person, motorcycle, or bicycle has generally lower width and so lower width is used to consider that it is present in the adjacent lanes. Also, height considered to identify them as obstacles, is also set lower as they can be close in the adjacent lanes and still be a bit far away from the camera, compared to vehicles. A person does not walk in the adjacent lanes in CARLA and so, their low width is not considered while checking the adjacent lanes for the classes other than vehicles.

Vehicle Maneuver:

After our decision model gives a decision whether to accelerate, change lane, or decelerate, our vehicle gets some specific commands to move accordingly. These commands were given with the help of the "apply_control" method to our vehicle in CARLA.

The throttle value indicates whether to accelerate or decelerate. If zero is given, the vehicle decelerates to a stop. If one is given, the vehicle accelerates. If -1 is given, the vehicle goes back. The steer value indicates which direction the vehicle should turn. The value ranges from -1 to 1. If the value is given zero, it goes straight. If the value is given more than 0 till 1, it steers to the right. The more the value, the more right it goes. If the value is given less than 0 till -1, it steers to the left. The more negative the value, the more left it goes. "time.sleep()" command is used to delay the execution of the next command by the time given in seconds. Using these commands and through much trial and error, we generated the action commands, which are integrated with the decision model's output.

For accelerating, the throttle value is set to 1 and the steer is set to 0. Similarly, for decelerating to a stop, the throttle value is set to 0 and the steer is 0.

For changing the lane to the left lane, we keep throttle 1 and make the car steer -0.5 to the left for 0.33 seconds and then steer 0.5 to the right for 0.39 seconds. Then, finally make it go straight in the new lane with steer 0. The order of steps are shown Fig. 10.

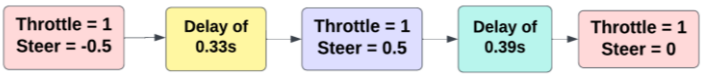

Fig 10: Changing lane to the left in CARLA

Similarly, for changing the lane to the right lane, we keep throttle 1 too but make the car steer 0.5 to the right for 0.33 seconds and then steer -0.5 to the left for 0.39 seconds this time. Then, finally make it go straight in the new lane with steer 0. The order of steps are shown Fig. 11.

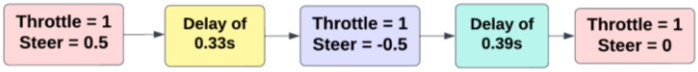

Fig 11: Changing lane to the right in CARLA

2.3.3 Methodology Approach-2 (Machine Learning) for hardware

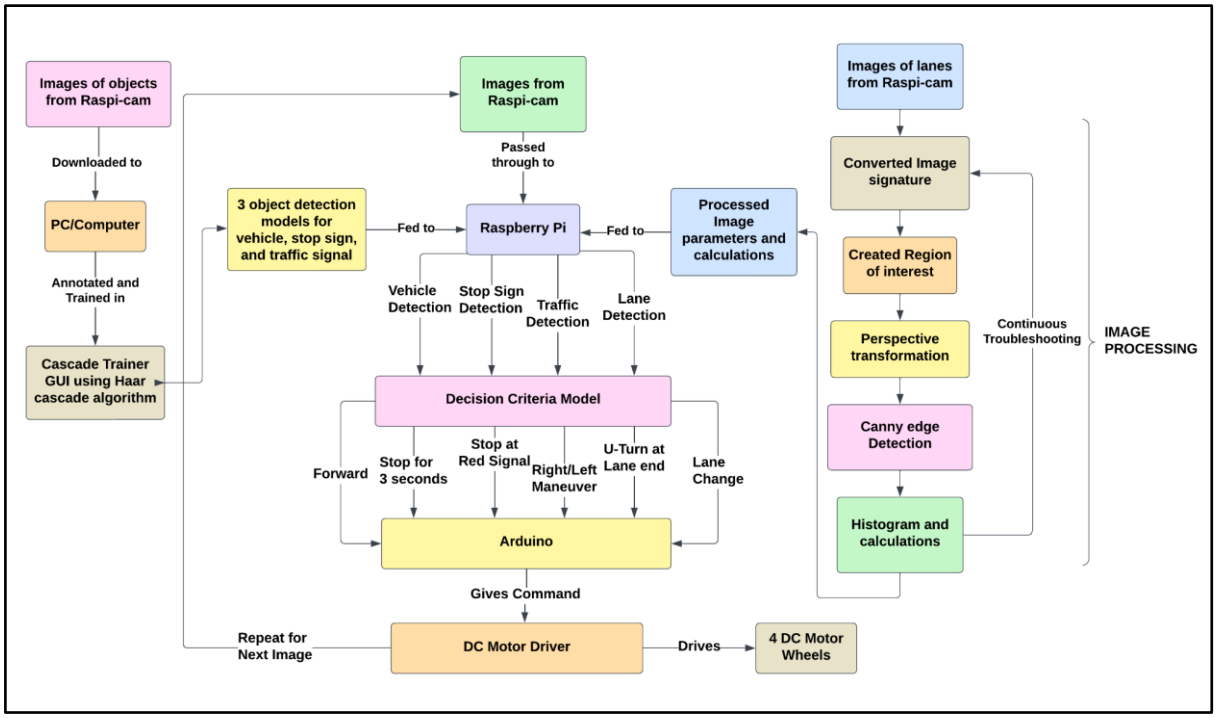

Fig 12: Methodology for machine learning approach in hardware system

For our machine learning methodology in hardware prototype, we used deep learning (Haar cascade), openCV, image processing and decision criteria to effectively train, detect, and run our self-driving car in our lanes. This is an embedded IOT approach as well, with machine learning. We first had to buy all our required hardware components like Raspberry Pi, Raspberry Pi camera, Arduino, SD card, motor driver, motors, car chassis, power bank, batteries, etc. We had to construct our car structure by connecting all those components.

Fig 13: Motor driver with DC motors and wheels in car chassis

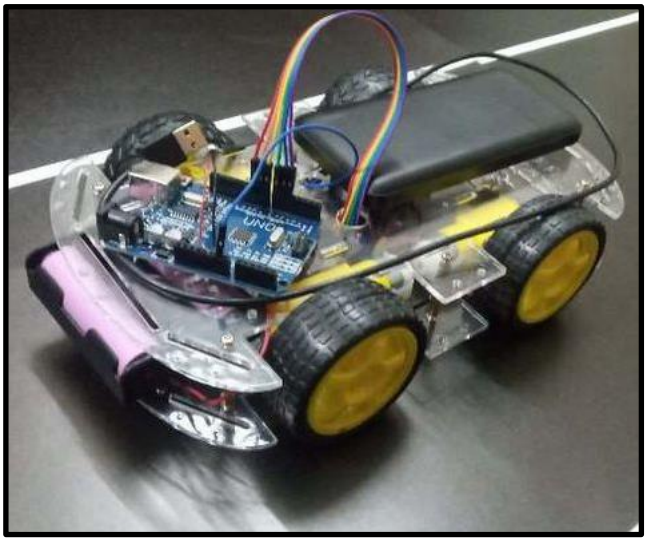

Fig 14: Arduino UNO and power bank

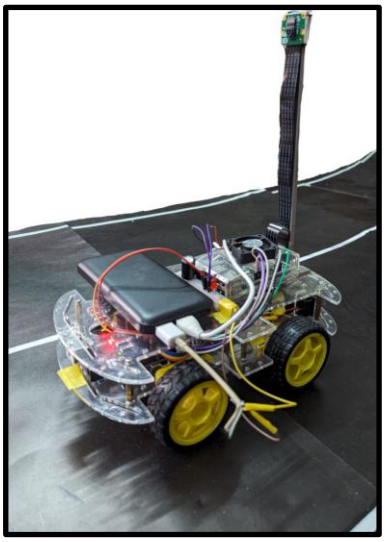

Fig 15: Raspberry Pi and Raspberry Pi camera module

Our main subsystems will have to be fed into Raspberry Pi, which will detect, decide and send commands to the Arduino to maneuver the wheels accordingly using the motor driver and motors.

To set up our Raspberry Pi 3B+, we first booted the 32-bit Raspberry Pi OS Full in our 32 GB SD card using Raspberry Pi imager software and accessed our Raspberry Pi interface using PuTTY and VNC viewer. We downloaded our required packages in Raspberry Pi and installed OpenCV and Raspi-cam. We captured images and videos, and calculated frames per second. Then, we had to do image processing for lane detection, where we converted image signature (BGR to RGB), created region of interest, did perspective transformation (Bird Eye View) on that region, threshold operations to extract white lane lines, canny edge detections on those lines, continuous troubleshooting, found exact lane positions from the lines using histogram and various calculations, calibration to map lane center to frame center and finally find the relative difference between them to know how much our car needs to move left or right to match its frame center to the lane center. This is sent to the arduino for command.

We also trained 3 object detection models in Cascade Trainer GUI software that uses the Haar cascade algorithm. Positive and negative samples were made from our Raspi-Cam for each 3 objects (vehicle, stop sign, and traffic signal) individually and trained individually, after which the 3 trained models were fed into the Raspberry Pi to detect them in running conditions. For all the objects, Raspberry Pi takes decisions when it is detected to exist between 5 to 20 or 30 centimeters ahead of the car, and sends commands accordingly to the Arduino for vehicle maneuver with the help of 4 digital pins.

For our Arduino, we kept some specific action commands, such as forward, backward, stop, and U-turn to send to our motor drivers according to the decisions taken by Raspberry Pi. We kept 3 types of left movements and 3 types of right movements, each having a different magnitude of turn, and only one of them occurs every time our raspberry decides on relative lane center position after analyzing the image it is getting, and sends its decision to the arduino through its 4 digital pins to adjust and maintain the lane.

The whole process of detecting, analyzing, making decisions, and moving is repeated for the next image obtained from a Raspberry Pi camera, for the continuous operation of our self-driving car.

We used some specific commands to install packages and OpenCV in our Raspberry Pi, after which we did C++ coding in the Geany programming editor. We did specific coding for our Raspberry Pi to capture images from the Raspberry Pi camera to use in image processing and training machine learning models. The images' width was 400 pixels, and the height was 240 pixels, with brightness, contrast, saturation, and gain set to 50. Examples of the taken images from Raspi-cam are shown below in Fig. 16.

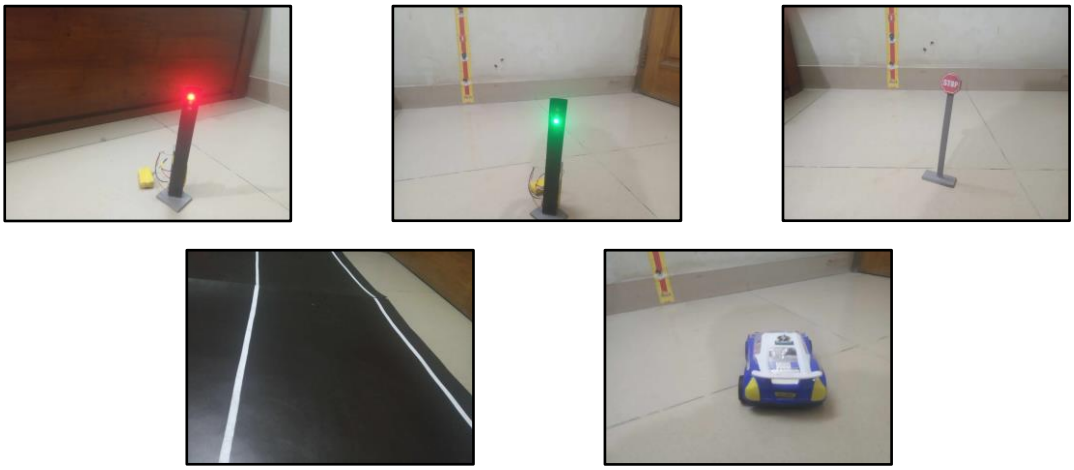

Fig 16: Collected image samples from Raspberry Pi camera

After the image acquisition process, we did image processing on only the lane image samples to detect lanes, as discussed before. To detect lane end, we also used the histogram process by creating a new dynamic array to store the intensity values of the region of interest that contains lane end lines.

The rest of the image datasets of car, stop sign, and traffic singal was used for cropping in Cascade Trainer GUI software for Haar cascade training to create positive and negative samples for each. We labeled a total of 300 images with positive and negative samples for car, vehicle, and traffic signals separately. Our cropped images contained full objects as well as occluded object for better training. The cropping had to be done manually in Cascade Trainer GUI, which automatically crops and saves the bounding box that was made on an object. These were considered positive samples to detect objects, whereas the negative samples considered images which had no object. So each object had both positive and negative sample set. Examples of cropped positive images are shown in Fig. 17.

Fig 17: Cropped positive image samples

We then trained 3 times for 3 objects using 3 sets of positive and negative image samples in Cascade Trainer GUI to obtain 3 object detection models, which were used and called in the Raspberry Pi to detect the 3 objects, besides the lane detection.

Decision Model:

The width (in pixels) of the bounded box, that we get from the detected object, was used to find out how far away that object is. We did manual calculations to solve 3 linear equations to convert pixel dimensions to the distance from the object and got the following equations:

For both stop sign and traffic signal detection, distance = $-1.07 \times \text{width} + 102.597$

For car detection, distance = $-0.48 \times \text{width} + 56.6$

We also can find out our deviation value from our lane, and the lane end intensity value using image processing. We will make decisions based on them, as well as the 3 object detections. Finally, there will be 11 conditions to choose from and the respective commands, in the form of digital numbers of 4 bits, will be sent to the Arduino for vehicle maneuver, as shown in Fig. 18.

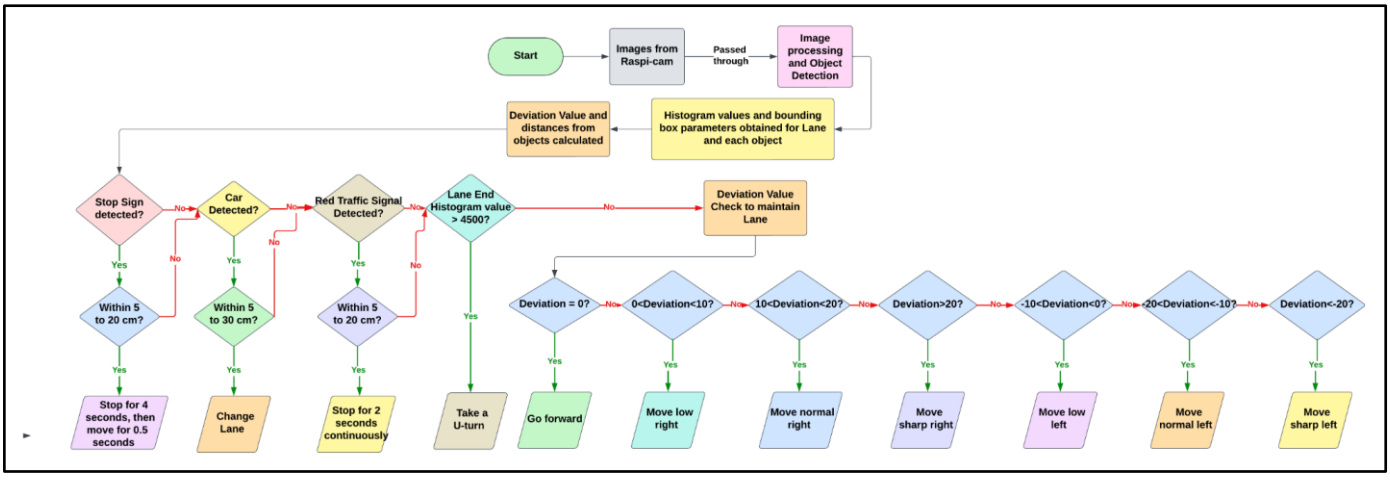

Fig 18: Decision Model for machine learning approach in hardware

If a stop sign is detected and the distance is more than 5 cm but less than 20 cm, then the Raspberry Pi sends commands to the Arduino to stop the car for 4 seconds, and then move ahead for 0.5 seconds to overtake the stop sign and take the next decision.

If a traffic signal with red light is detected and distance is more than 5 cm but less than 20 cm, then the Raspberry Pi sends commands to the arduino to keep on stopping the car for 2 seconds continuously, until the red light changes to green light, or turns off completely.

If a car ahead is detected and the distance is more than 5 cm but less than 30 cm, then the Raspberry Pi sends commands to the arduino to make the car change its lane. For this, a turn is taken using motor drivers, followed by a forward operation and then a reverse turn to align with the new lane. Then, after continuing forward on the new lane for some time while checking for other detections, the car comes back to its previous lane in a similar way with a different direction, ultimately overtaking the car that it had detected before.

If the histogram value of the region of interest exceeds 4500, then it indicates a Lane End, and the arduino makes our car take a U-Turn. It rotates 90 degrees in one direction, moves forward for 1 second, then again rotates 90 degrees in the same direction, before continuing to run on the second lane.

The rest 7 conditions are kept to maintain our car on the lane. If the deviation value is 0, the car moves forward. If the deviation value comes out to be between 0 to 10, then a low right turn is executed by varying the velocities of the left and right wheels. If deviation value comes out to be between 10 to 20, then a normal right turn is executed. If deviation value comes out to be more than 20, then a high right turn is executed. Similarly low, normal, and high left turn is executed but for negative deviation values, depending on the magnitude.

2.4 Analysis of multiple design approach

2.4.1 Analysis of Design-1 (Rule-Based):

For the rule-based approach, as discussed previously, our vehicle knows the GPS location of the vehicles and mobile phones, and drives on the road accordingly. At the bottom left side of the pictures below, there are multiple vehicles that it is detecting through GPS and knows its location while avoiding collision with them.

1) When our vehicle detects a person, it stops at a braking distance of 2 feet and waits for the person to pass by. When our vehicle doesn't detect any more people, it starts to move again.

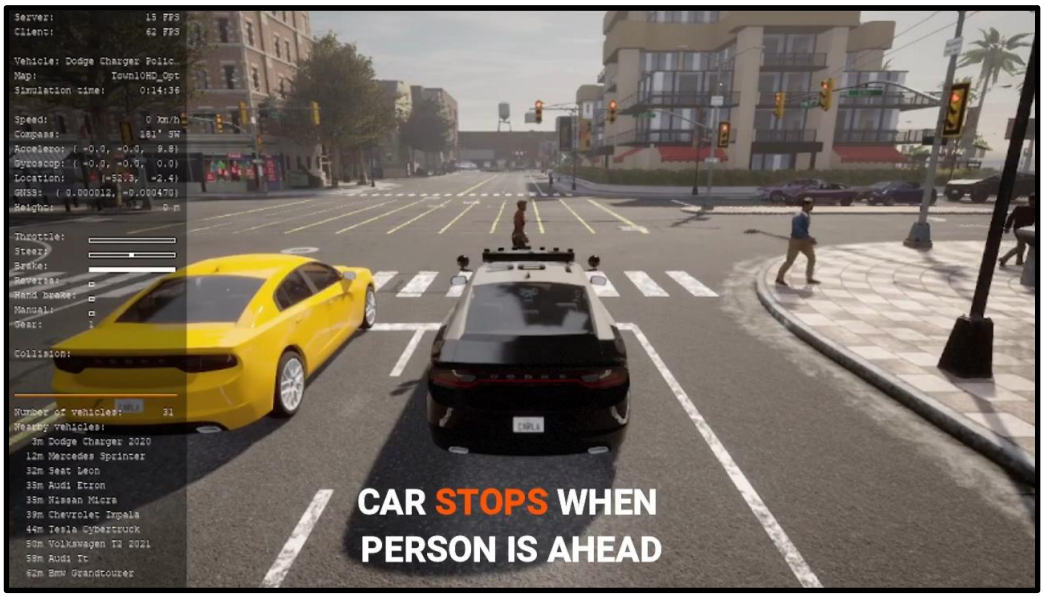

Fig 19: Vehicle detects person and stops

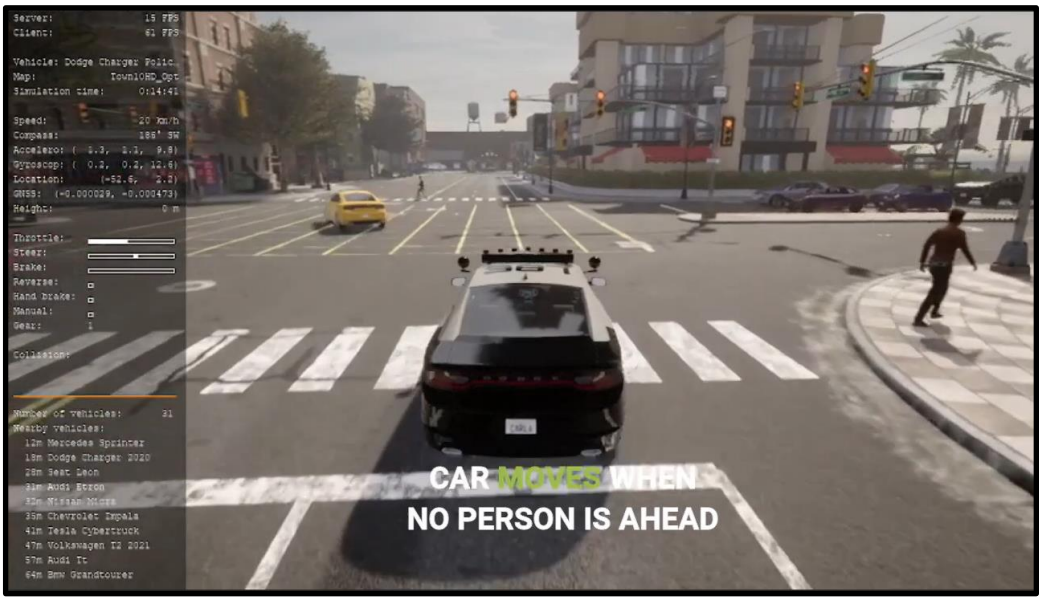

Fig 20: Vehicle detects no more person and starts to move

2) When our vehicle detects a car ahead, it stops at a braking distance of 2 feet and waits for the front car to move. When the front car moves, our vehicle starts to move, maintaining the braking distance of at least 2 feet.

Fig 21: Vehicle detects a car ahead and stops

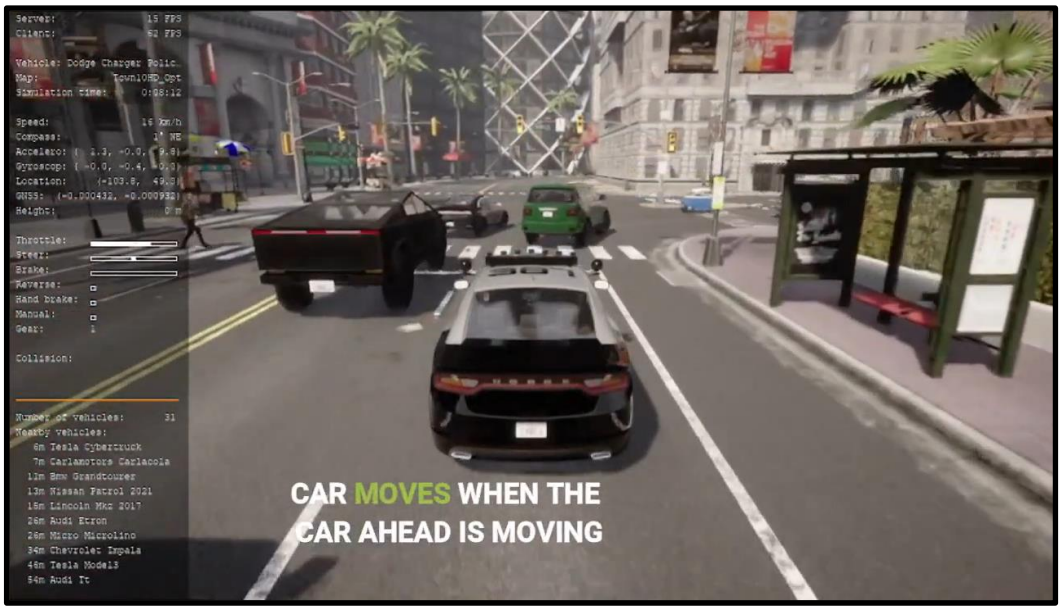

Fig 22: Vehicle moves when front car moves, maintaining some distance of 2 feet

3) Our vehicle maintains a lane when it finds that the forward path is open, also while changing direction.

Fig 23: Vehicle changing direction while maintaining lane

4) If our vehicle detects a vehicle in front of it for more than 2 seconds, it checks its right or left lane whether there are cars or not. If not, it changes lanes accordingly, which also meets our functional requirements.

Fig: 24 Vehicle detects other vehicles in front of it

Fig 25: Vehicle changing lane

2.4.2 Analysis of Design-2 (Machine Learning) in CARLA:

For this design, we extracted some dataset images from the CARLA simulator to detect our obstacles. We have used Roboflow to do bounding boxes on images from CARLA for 4 classes: Vehicle, person, motorcycle, and Bicycle. Then, trained our model using the YOLOv4 object detection algorithm. We could have considered the class of traffic signals as well. However, it is too small for our vehicle in the CARLA simulator to detect the light signals from far away. We will try to detect it in hardware design, where the signals will be closer and clearer to see and the detection process will be the same as the other 4 classes we implemented here. Also, as there are no stop signs or pedestrian signs in CARLA, we are omitting it here. However, we will detect it in hardware design in a similar manner. CARLA also experiences heavy lagging for more classes and processing, which was also a problem for us.

Fig 26: Bounding Box applied on images for Vehicles and Motorcycles

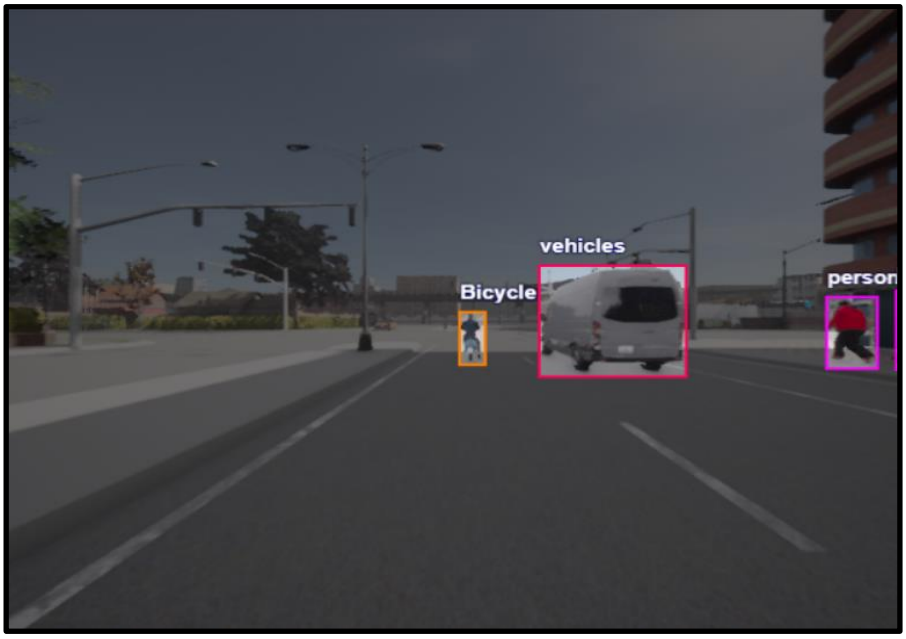

Fig 27: Bounding Box applied on images for Bicycles and Person

We trained our object detection model three times and obtained 3 different results. The dataset images were divided into 70% for training images, 20% for validation images, and 10% for test images.

Test-1:

We have used around 200 dataset images to train in test-1 with 200 epochs. We obtained mean average precision (mAP) of 87.3%, precision of 85.9% but a recall of 54.8%. Recall indicates how much percentage of a sample from the test image that our machine learning model correctly identifies as belonging to the correct class, out of the total test image samples for that class.

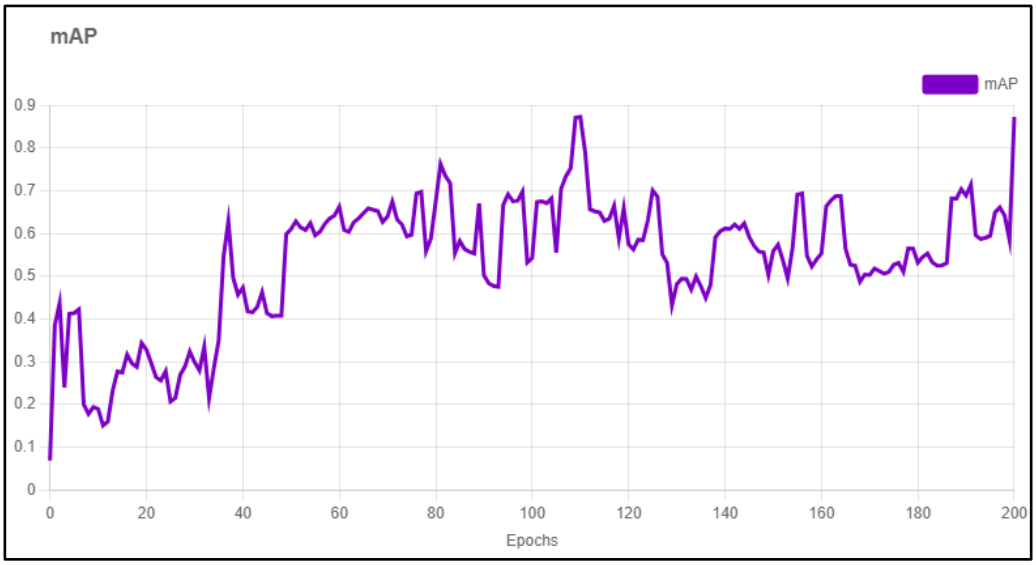

Fig 28: Mean average precision vs epochs for test-1

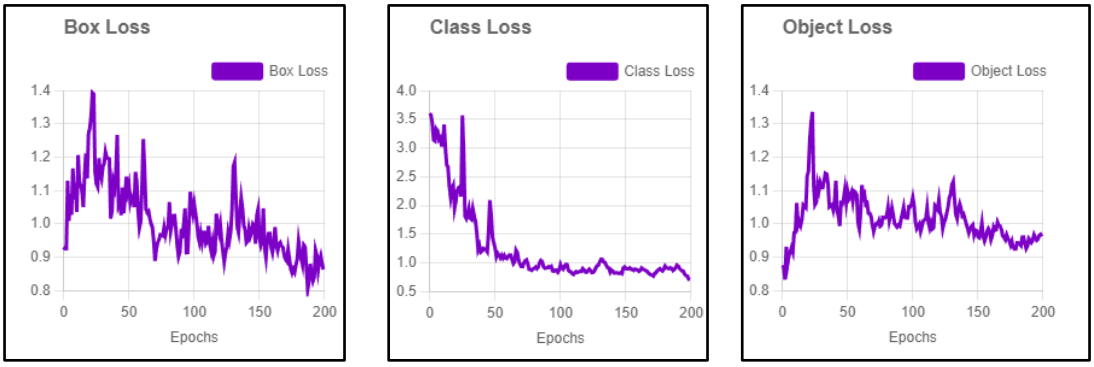

Fig 29: Box loss, Class loss and Object Loss vs epochs for test-1

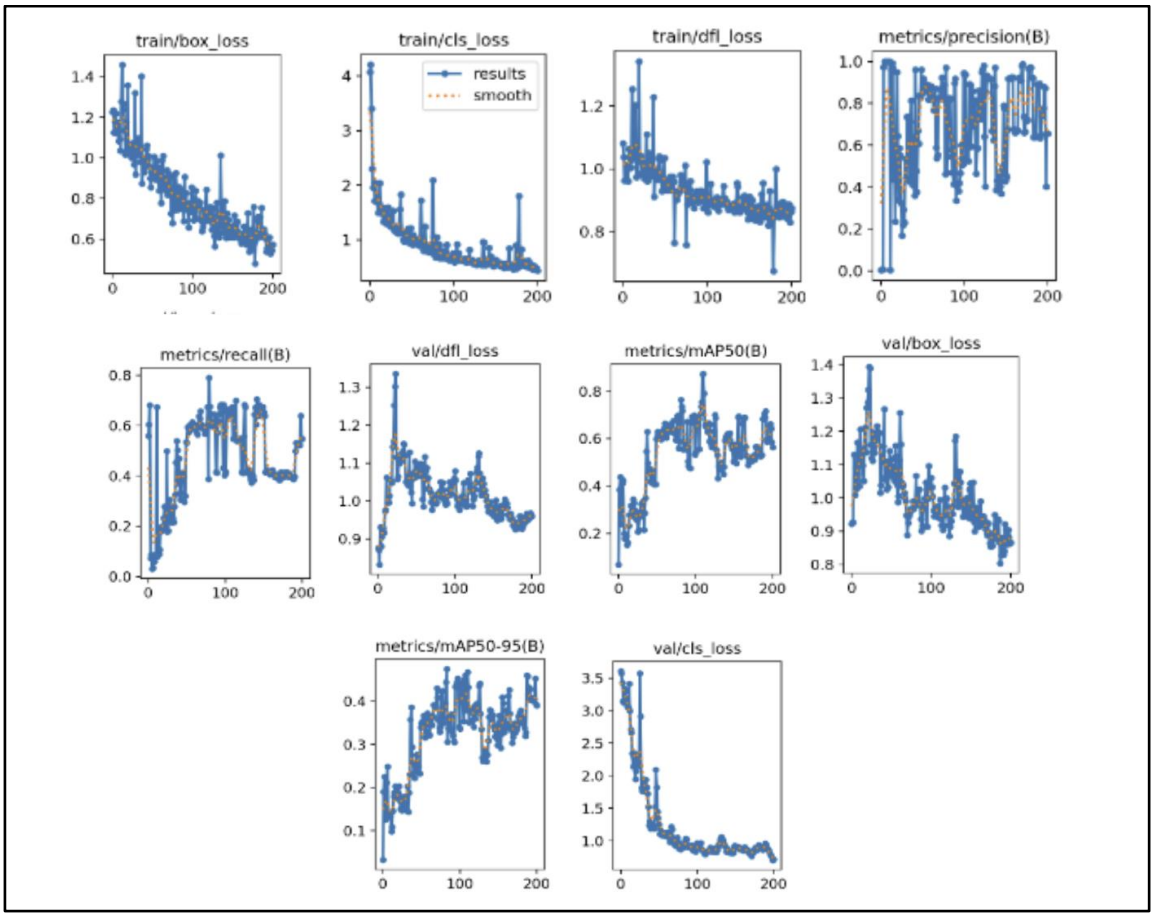

Fig 30: Loss, precision and recall graphs vs epochs for test-1

Above, we have extracted the training and validation graphs of Box loss, class loss, distribution focal loss (DFL), precision, recall, and mAP with respect to epochs, for test-1 but we noticed there are a lot of fluctuations and spikes in the graphs while training our model, ultimately giving a very low recall of 54.8%.

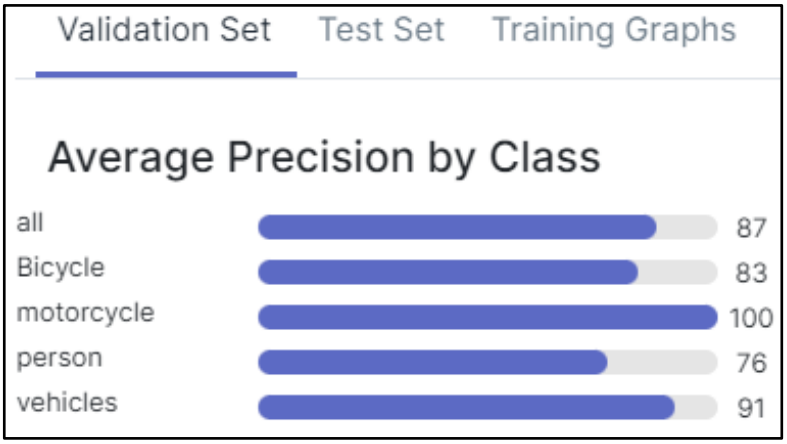

Fig 31: Showing the Average Precision by class for Validation set results for test-1

Fig 32: Showing the Average Precision by class for Test set results for test-1

Although the precision also looks acceptable here, however, we can see that for the class of motorcycle, the model overfit. So, it could not detect any motorcycle in the test images. For that reason also, we needed to train our model again with more dataset images and epochs this time, in test-2.

Test-2:

We have used around 1000 dataset images to train in test-2 with 300 epochs this time. We obtained mean average precision (mAP) of 88.6%, precision of 85.9% and a recall of 78.9%.

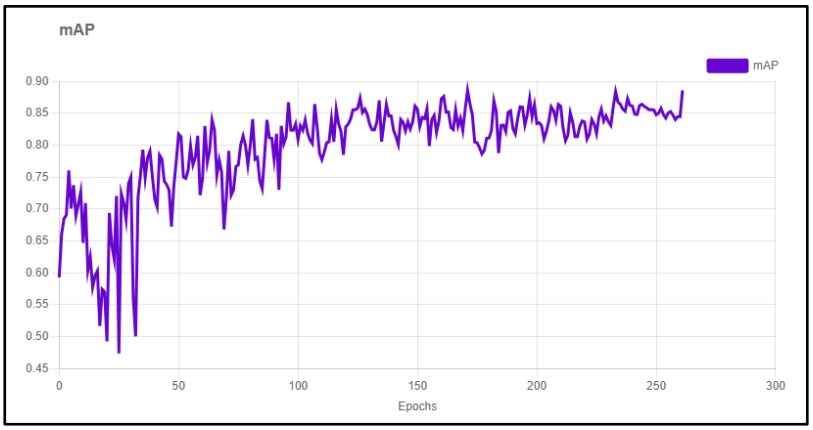

Fig 33: Mean average precision vs epochs for test-2

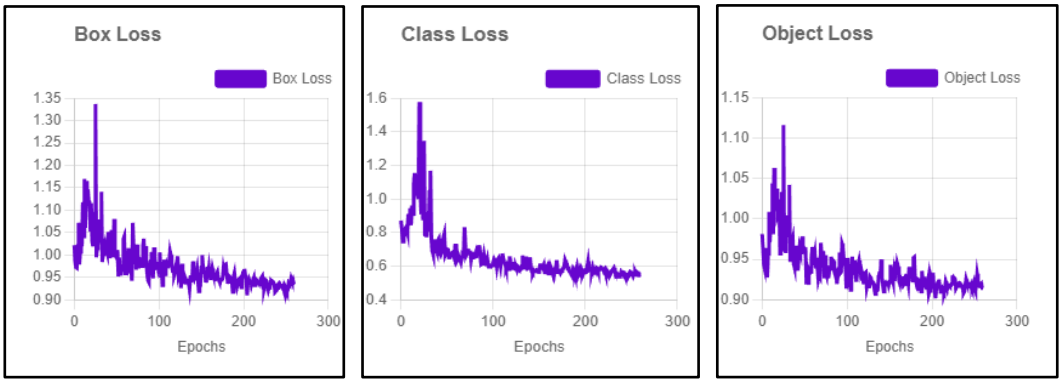

Fig 34: Box loss, Class loss and Object Loss vs epochs for test-2

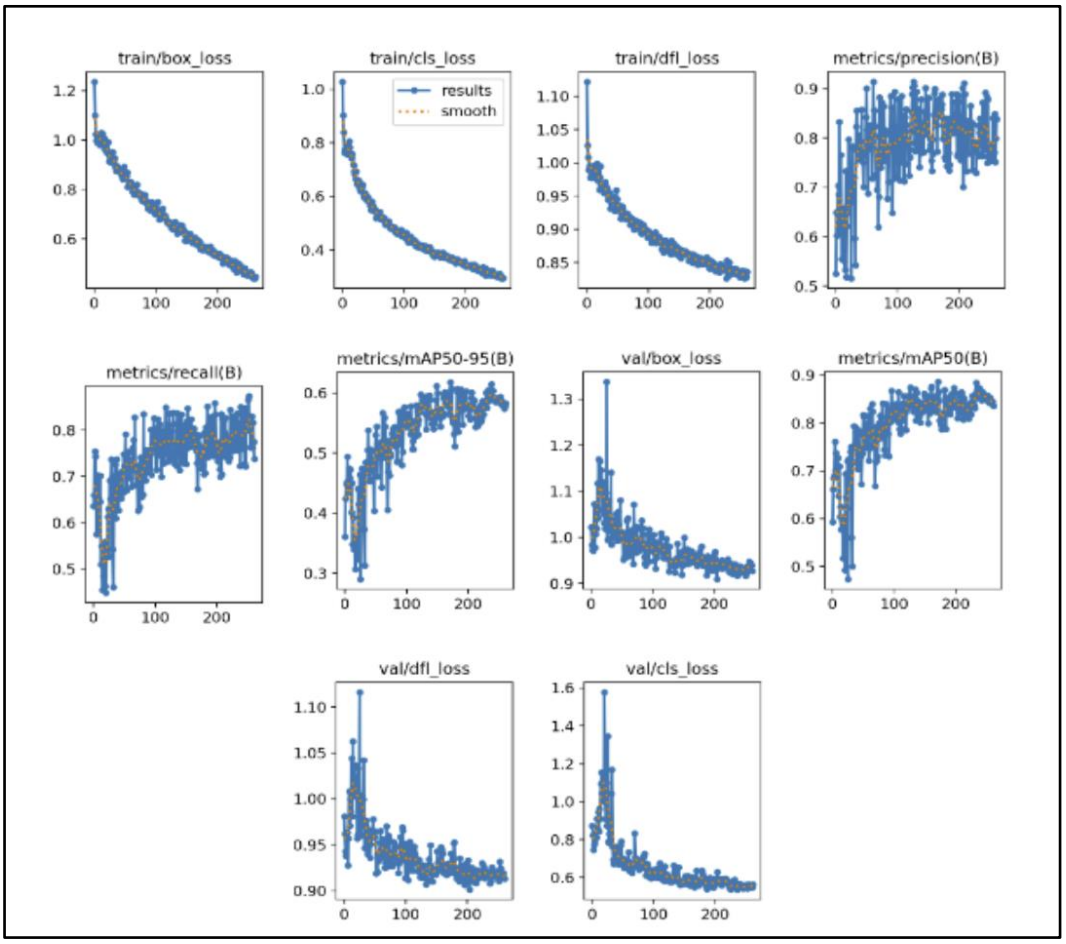

Fig 35: Loss, precision and recall graphs vs epochs for test-2

This time, we notice that the fluctuations and spikes in the graphs decreased while training our model in test-2 compared to test-1, ultimately giving a higher recall of 78.9%

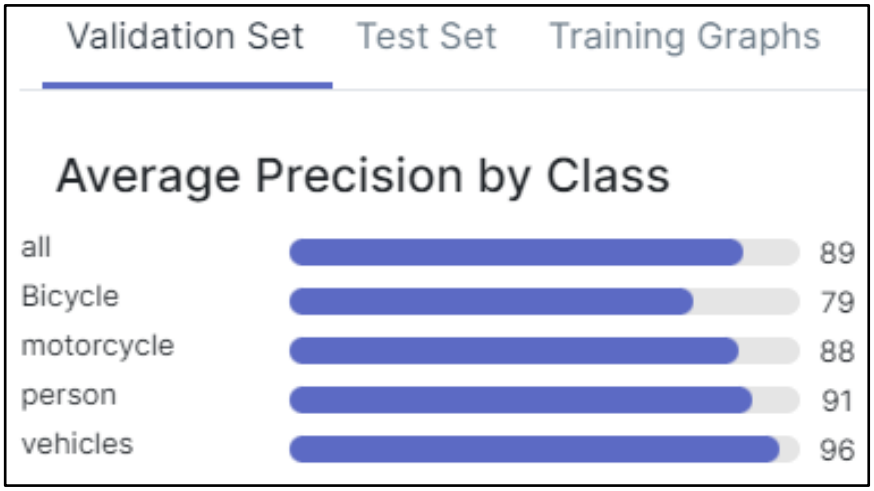

Fig 36: Showing the Average Precision by class for Validation set results for test-2

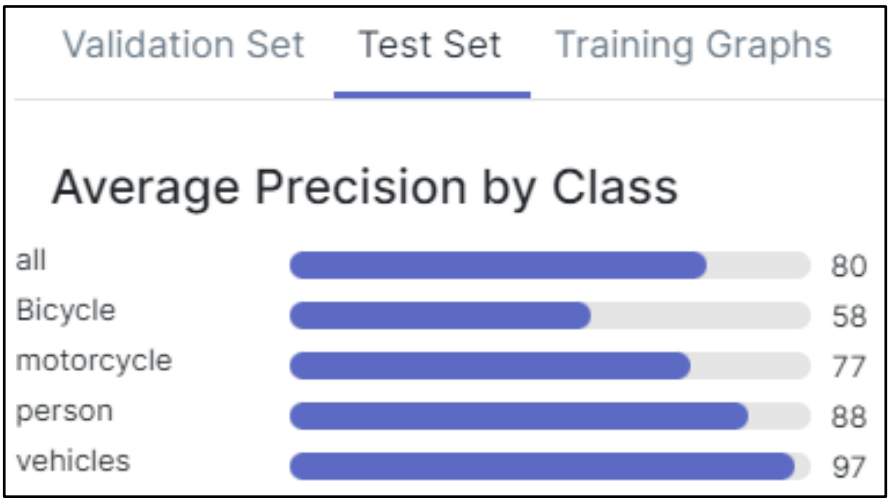

Fig 37: Showing the Average Precision by class for Test set results for test-2

It is clear here that the precision for validation and test, for all the images increased. For further improvement, we would be needing more dataset images. As we had to manually take dataset images from the CARLA simulator, it was a really lengthy process to take all the images and annotate them all. So, extracting these images turned out to be more time consuming than expected and so we have ultimately used our test-2 model to detect objects and conduct simulation in CARLA.

Decision Model Results for CARLA

For the result of the decision model, we obtained good results for the parameters specified for the model to give its decision. The decision model's accuracy also depends on the accuracy of the object detection. As we obtained around 80% average precision for it, our decision model can give close to accurate results for the 4 classes detected. Below are some of the case examples we got from our decision model.

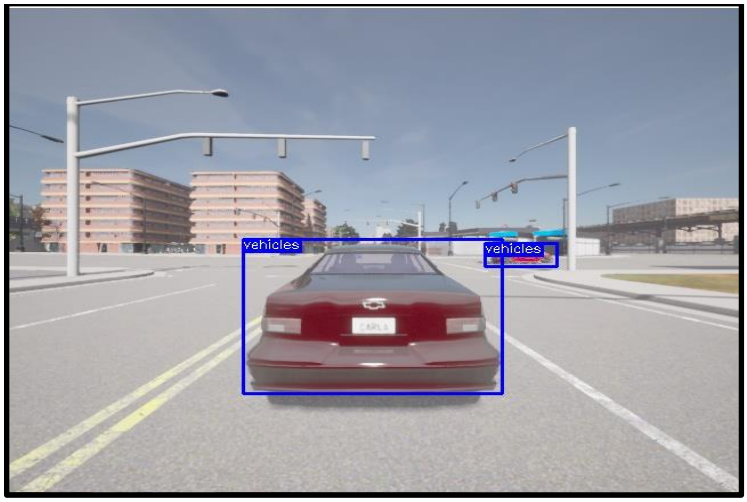

Fig 38: Vehicle in front but right lane is clear

For the front vehicle in image from Fig. 38, we got distance, z = 480 - 308 - 153/2 = 95.5 which is less than 135 and x = 319 which is between 170 and 430. So, going straight is not possible. No object is detected in the right lane and so, our decision model gives the decision to change lane to the right.

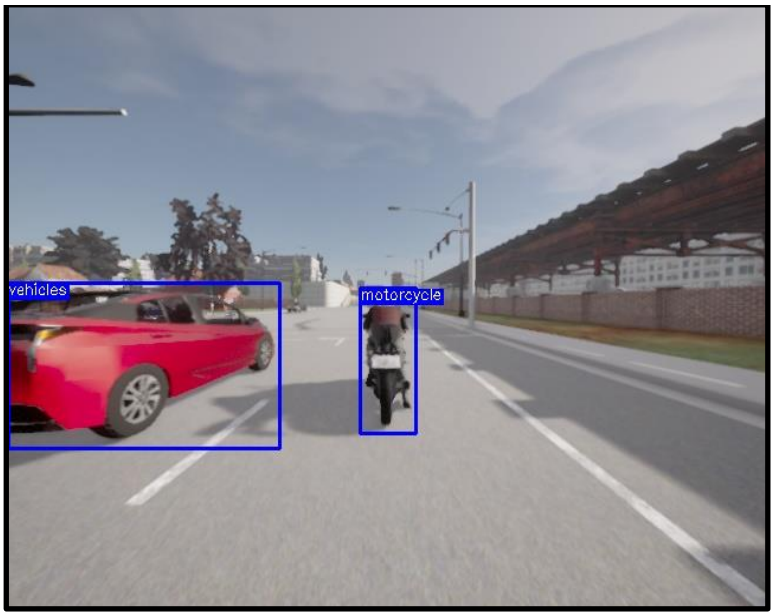

Fig 39: Motorcycle in front but a bit far

Similarly, for the front motorcycle in image from Fig. 39, we got distance, z = 480 - 282 - 116/2 = 140 which is barely more than 135 and we got x = 319 which is between 170 and 430. So, our decision model gives the decision to accelerate forward. However, if the motorcycle was a bit closer so much so that the value of z was less than 135, then our decision model would have checked the adjacent lanes, only to find here that there is a vehicle at the left as well. So, it would decelerate.

As for the maneuver of our vehicle, it stops, accelerates, or changes lanes when required from the decision model. Sometimes, the lane changing is not always perfectly executed and so, a lane invasion sensor from CARLA is used to prevent our car from crossing the lane lines after

changing lane, unless another decision comes up from the decision criteria model to change lane. This ensures our vehicle maintains its lane at all cost.

Simulation in CARLA based on Machine Learning:

We passed the camera window from CARLA through our trained object detection model test-2 to detect objects and then used the decision criteria for our detection model from camera and also from LiDAR to control our self-driving vehicle accordingly.

1) Our vehicle detects a vehicle in front of it using a camera and stops, maintaining a minimum braking distance of 2 feet.

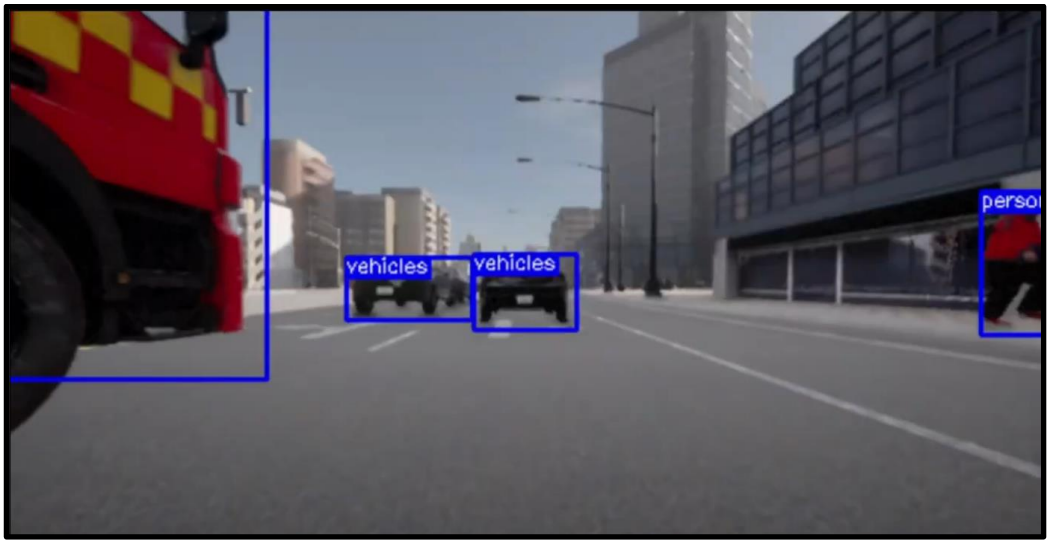

Fig 40: Vehicle detecting front vehicle but still continuing as it is still not within braking distance

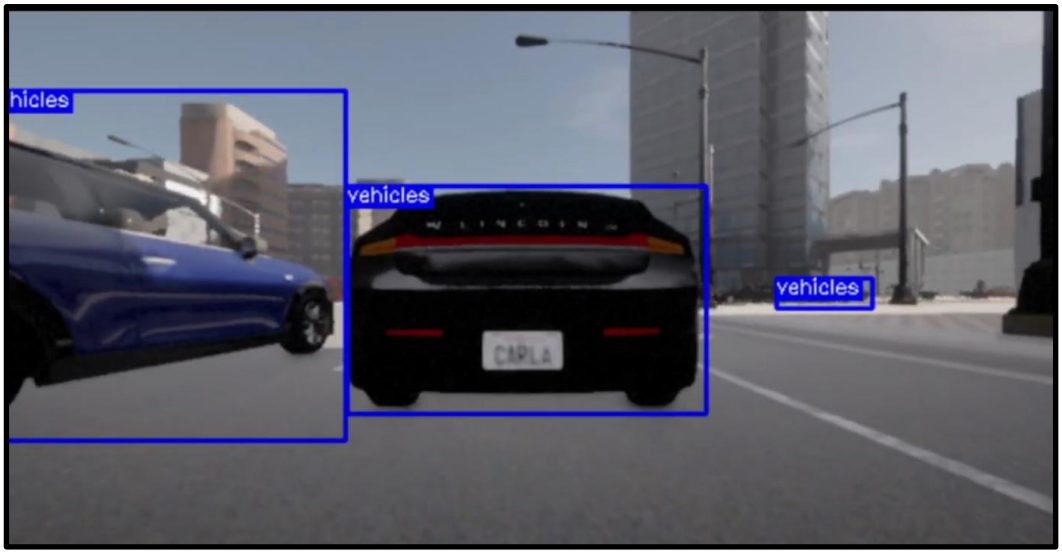

Fig 41: Vehicle stops, keeping a minimum braking distance between it and the front vehicle

2) Our vehicle detects a person in front of it using Camera or LiDAR and stops, maintaining a minimum braking distance of 2 feet

Fig 42: Vehicle stops, keeping a minimum braking distance between it and the front person

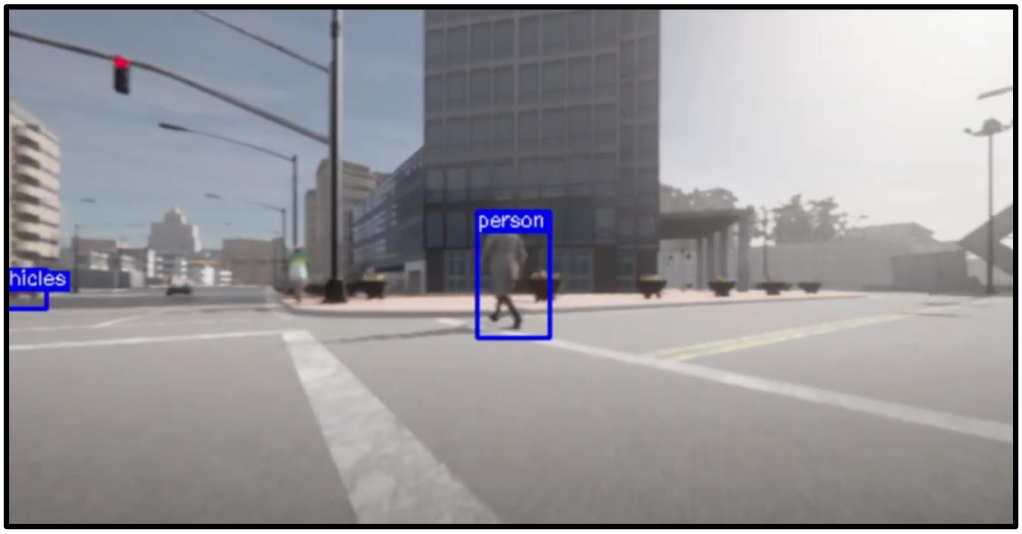

Fig: 43 Vehicle continues as it finds the lane clear of obstacles

3) If our vehicle detects a vehicle in front of it, it checks its right or left lane whether there are obstacles or not. If not, it changes lane accordingly, which also meets our functional requirement.

Fig 44: Vehicle detects other vehicle in front of it

Fig 45: Vehicle changing lane

We have considered the operating velocity in CARLA as default, either zero or moving at a specific speed, as all the other cars stop and move with the same speed all the time. However, in hardware implementation, we will be considering the speed while designing our vehicle.

Proteus Simulation:

We also implemented our design in proteus software for lane changing for our self-driving car.

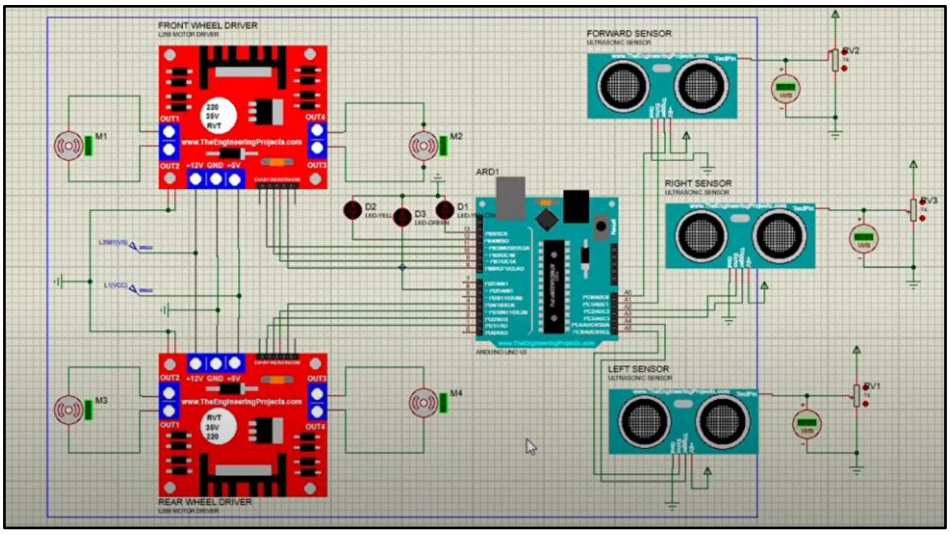

Fig 46: Simulation Design in Proteus for Hardware implementation

Implementing cameras and LiDAR is not possible in proteus software. So, in this design, instead of a LiDAR, we considered 3 ultrasonic sensors to detect objects in 3 lanes, and used 3 LEDs, representing the car lights, to give indication for 3 lanes: forward, right, and left. When voltage in the test pins increases above 4.5 for the ultrasonic sensors, it indicates that it is not detecting obstacles in that direction. The forward sensor detection was prioritized compared to the left or right sensor in the code.

1) When all of the 3 ultrasonic sensors are detecting obstacles ahead, the car stays or comes to a halt and all the LEDs turn off.

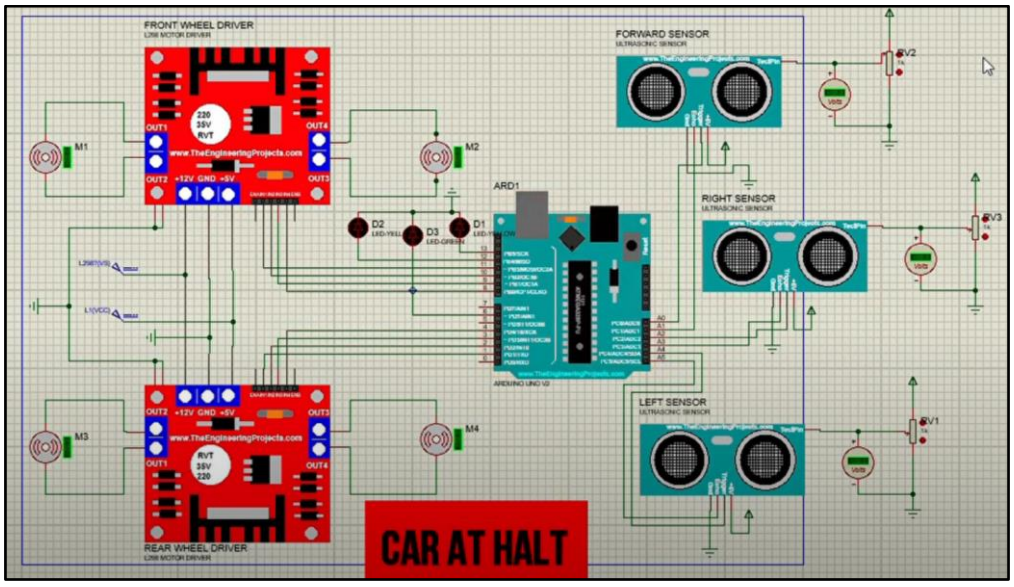

Fig 47: All four wheels and LEDs turn OFF (Car at Halt)

2) When the obstruction at the front clears up, voltage increases at the test pin of forward sensor and all four wheels start to move forward.

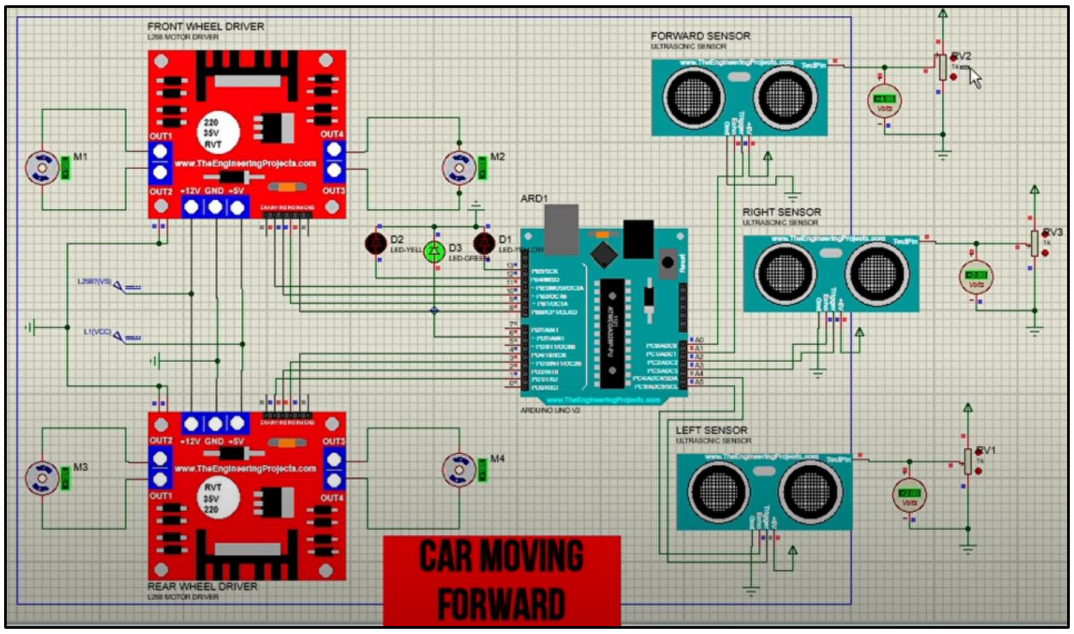

Fig 48: All four wheels moving forward and front LED turn ON (Car moving forward)

3) When there is an obstacle at the front but the right lane is free of obstacles, then voltage increases at the test pin of the right sensor and all 3 wheels, except the front right wheel, start to move forward. This causes the car to turn and go to the right lane due to right rotational motion at the front right wheel (pivot) that is created by friction.

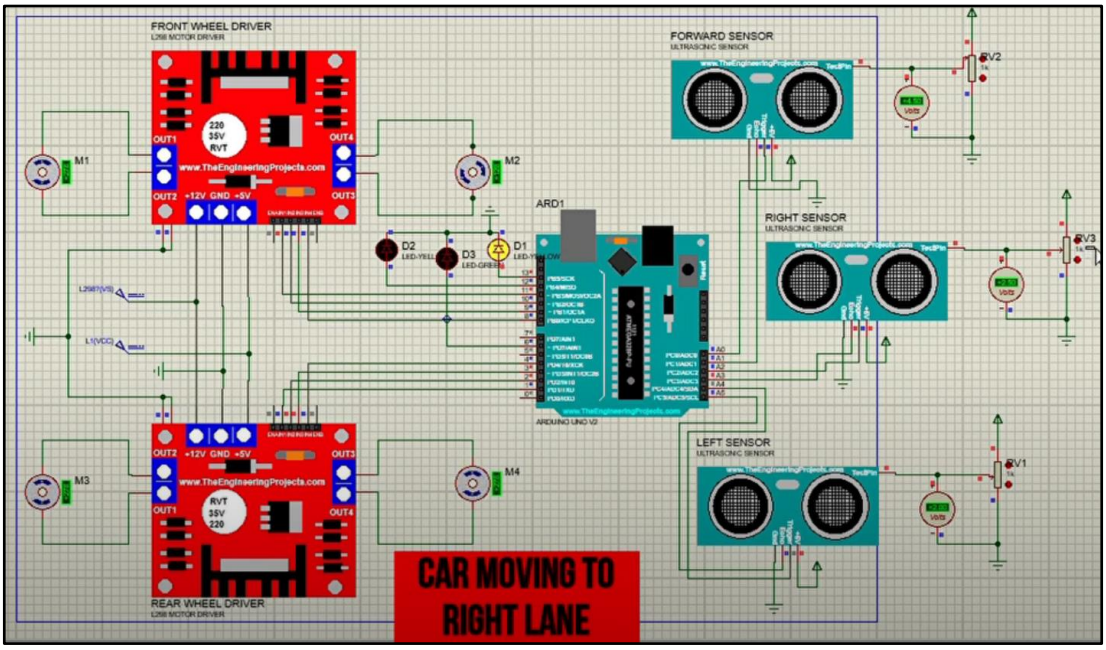

Fig 49: All 3 wheels, except front right wheel, moving and right LED turn ON (Car moving to the right lane)

4) When there is an obstacle at the front but the left lane is free of obstacles, then voltage increases at the test pin of the left sensor and all 3 wheels, except the front left wheel, start to move forward. This causes the car to turn and go to the left lane due to left rotational motion at the front left wheel (pivot) that is created by friction.

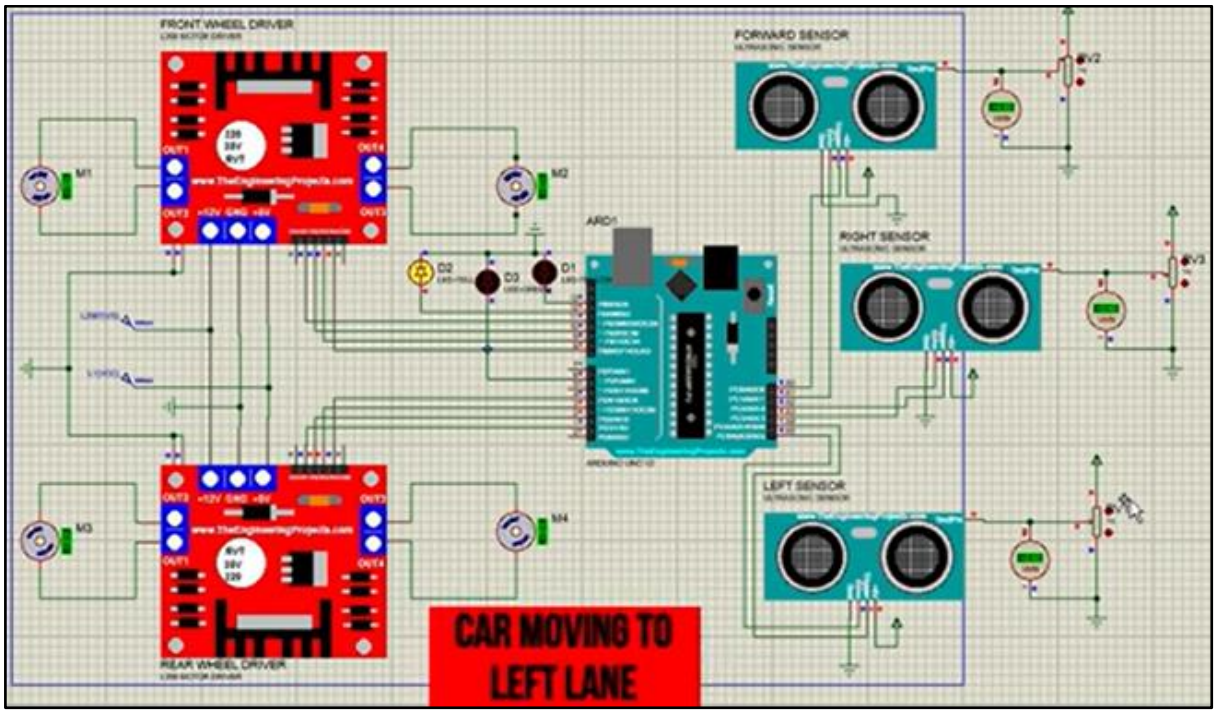

Fig 50: All 3 wheels, except front left wheel, moving and left LED turn ON (Car moving to the left lane)

We have not implemented the braking system here, that causes the car to come to an immediate stop, as it is not possible for the motors used in proteus to do so. However, there will be a braking mechanism in the hardware design, and the car will come to a stop depending on the decisions taken. We tried to implement the design in proteus as much as possible to represent a self-driving car with LiDAR/ultrasonic sensors.

Comparison Analysis:

We compared the two design approaches in the table below

Table 5. Comparison Analysis of multiple designs

Parameters	DESIGN APPROACH 1 (Rule-Based)	DESIGN APPROACH 2 (Machine Learning)
Cost	28,150 TK	28323 TK
Precision	Almost 100% precise but 0% for vehicles or obstacles outside of its GPS detection	Around 89% precise for all obstacles
Adaptability	Limited to the predefined set of rules and conditions	Can adapt to dynamic and ambiguous conditions
Knowledge Representation	Explicit rules are defined by human experts to guide the behavior of the self-driving car	The system learns from data without explicit programmed rules
Real-time Data Requirements	Need real-time coordinates of all objects in the entire region to function properly	Need only real-time image of objects in front of our car to function properly
Complexity	Tracking cars will increase complexity exponentially as every car will be dependent on each other	Less complex as cars are independent from each other
Safety and Liability	Can't detect objects that do not give GPS signals, endangering people without phones.	Safety higher since decision-making relies on learned patterns data of nearby environments
Availability of components	There is only one sensor (GPS) needed which is very common	More sensors may be required for better performance, which may not be available
Interpretability and Explainability	High since it's decision-making relies on clearly defined rules and logic	Often lack interpretability as difficult to explain the reasoning behind it's decisions
Maintainability	Easier to maintain as one sensor can be fixed easily when damaged	Harder to maintain as tough to fix multiple sensors

From the table, we can see a brief analysis of the two design approaches. We can see that both approaches have their merits by being advantageous in five parameters each.

Both the approaches seem to have more or less the same costs to produce similar functionality. As we will be using only a single GPS sensor in our rule-based approach, it will be easier to maintain as well as manufacture due to the availability of components. Also, as this approach's rules are constructed by us, it is easy to interpret and explain. Despite the pros of approach 1, there are some vital cons. The major one being the complexity. As each vehicle has to know the location of all vehicles in its vicinity, this creates an exponentially increasing complexity as the number of vehicles increases. Due to dependency on other vehicles, approach 1 can be considered less safe. Despite having a 100% precision, there is the possibility of a rogue vehicle

without any means of tracking it, which will lead to the precision to fall to 0%. Not only rogue vehicles, any person without a phone or means of transmitting location information is in danger of not being detected by the vehicles which can become very fatal if the particular situation is not declared in its predefined rules.

On the other hand, the machine learning approach is extremely effective in compensating for the failings of the rule-based approach. It is less complex as it only processes real-time information in close proximity, while providing greater safety as the vehicle can be enabled to learn from its environments and thus be able to react properly to dynamic and ambiguous conditions on the road with the help of its camera, object detection models and decision criteria algorithms.

2.5 Conclusion

After the specific objectives and functional requirements were discussed in chapter 1, 2 types of design approaches were simulated and analyzed in CARLA simulator which meets the specified goals in different ways. The 2 design approach's methodology have been explained in quite a lot of details along with their results. Finally, a comparison is made between the two approaches in terms of cost, precision, adaptability, knowledge representation, real-time data requirements, complexity, safety and liability, availability of components, interpretability and explainability, and maintainability.

After analyzing, we can safely conclude that design approach 2 is the optimum solution for our desired outcome. We choose the machine-learning approach mainly because it can effectively work for unexpected situations and cars nearby by sensing them with only requiring real time front images. Rule-based approaches cannot consider cars outside of their set of rules, and complexity increases exponentially as cars are added to the set of rules, with the problem of not getting GPS signals from every obstacle and requiring real-time coordinates for all vehicles in the region.

Chapter 3

Use of Modern Engineering and IT Tool

3.1 Introduction

Numerous IT tools have been employed to ensure the project's successful execution. Prior to the actual hardware implementation, they assisted in assessing the design and performance of each design method and identifying areas that needed improvement. As a result, the ideal design could be confirmed.

3.2 Select appropriate engineering and IT tools

3.2.1 CARLA Simulator 0.9.14

CARLA Simulator is an open-source software which helps in the simulation of autonomous vehicles [28]. CARLA simulator is famous for its reliability and user-friendly interface. The Simulator has certain Characteristics:

1. Support development of autonomous driving systems.
2. Training of autonomous driving systems.
3. Validation of autonomous driving systems.

The Simulator has certain features which help it in imitating the real-world interface. It has urban layouts, buildings, vehicles this is pre-built and can be used with ease. Users can tune the various environmental settings along with the environmental conditions, sensor suites, maps and various other options [28]. The software has certain Key features:

- Quickstart: It is extremely effortless to get started with a CARLA simulator.
- Actors: This simulation tool has pedestrians, traffic signals as well as pedestrians in order to give a real world taste.
- Sensors: CARLA has a wide variety of sensors to use like LIDAR, RADAR as well as cameras [28].
- ROS bridge: The Robot Operating System of CARLA helps the endless connection.
- Scalability: It has a multi-client architecture which is server based.
- Flexible API: The different interfaces can be tuned such as the simulation, including pedestrian behaviors, weathers, traffic generation, sensors, etc.
- Fast simulation for planning and control: Fast execution is possible in CARLA which enables fast execution of traffic simulation and road behaviors for which graphics are not required [28].

Table 6. Reasons why to choose Carla over Udacity:

	Open-Source	Complex Environments	Sensor Support	Customization	Real-World Applications
Carla Simulator	Yes	Yes	Wide range	Wide range possible	Yes
Udacity Simulator	Yes (but very Limited)	Very Limited	Few available	Very Limited	No

3.2.2 Python 3.7

It is a type of computer programming language mainly used to build software and do software related works, automate tasks, and conduct data analysis. It is highly suitable for usage in CARLA.

3.2.3 Sublime Text 3

It is a type of shareware text and source code editor. The main work of it is to support and edit the programming languages and markup languages including python.

3.2.4 Anaconda Prompt

Anaconda is an open-source platform that is used for data science and machine learning tasks. Anaconda Prompt allows users to interact with Python and Anaconda packages through the command line. It provides a way to manage Python environments, install packages, and run scripts or applications.

3.2.5 Google Collab

Google Colab is an online platform that allows users to write and execute Python code collaboratively. It provides a Python programming environment in the form of Jupyter Notebooks.

3.2.6 Proteus

It is basically a software used to design and analyze the electronic circuit. We have used it for our hardware design implementation of our self-driving car. It has the following characteristics:

1. Electronic circuit design.
2. Simulation of the designed circuit.
3. PCB (Printed Circuit Board) of the designed circuit.

It has a lot of key features which makes it the suitable software for designing circuits [32].

- Library of Components: Proteus has a wide range of library components which electrical engineers need in their daily life. We will be using Arduino UNO, motor drivers, motors, LEDs, ultrasonic sensors and potentiometers to design our circuit.
- Schematic Capture: Proteus allows the users to create electronic circuit schematics with a range of components like microcontrollers, resistors, sensors, capacitors, transistors, and many more so that the user can get a real world experience.
- Simulation: Proteus has the ability to showcase simulation as a result, the users can access the electronic circuits by analyzing its behavior, tuning necessary voltage, current flow, and finally observe how the circuit functions [32].
- Microcontroller Simulation: Proteus has a library which contains microcontroller models used in various embedded systems. We can write code in the required language and then add the file in the proteus which inturn helps the microcontroller to run properly.
- PCB Design: Proteus has a library and component list which helps in routing, power ring management, power meshing then arranging components, defining board layers, etc [32]. However, we will not be using this.
- Interactive Debugging: Proteus has a very user-friendly troubleshooting and debugging platform that helps users to analyze, energize then re-energize their circuits if needed.

3.2.7 Arduino IDE

Arduino IDE or Arduino Integrated Development Environment is nothing but a software which gives a platform to electrical engineers to develop various circuits and modules using the Arduino microcontroller board. Arduino UNO has the following characteristics:

1. Open-source hardware.
2. Interactive electronic projects design.
3. Quick Prototype generation [33].

3.2.8 Cascade Trainer GUI

A tool for training, testing, and refining cascade classifier models. To establish the parameters and make using OpenCV tools for training and testing classifiers simple, it employs a graphical user interface [34].

3.2.9 CodeBlocks

The cross-platform, free, and open-source IDE that is compatible with GCC, Clang, and Visual C++ among other compilers. It is created in C++ with the GUI toolkit wxWidgets [35].

3.2.10 C++ Programming Language

An expansion of the C programming language, C++ is a strong and flexible programming language. It was created to supplement C's procedural programming tools with object-oriented programming functionality [36].

3.2.11 VNC Viewer

VNC (Virtual Network Computing) Viewer is a client application that allows users to connect to and control a remote computer running a VNC server. VNC is a graphical desktop-sharing system that enables remote access and control of another computer's desktop environment [37].

3.2.12 PuTTY

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. It can also connect to a serial port [38].

3.2.13 Geany Programming Editor

Using Scintilla and GTK, Geany is a lightweight GUI text editor that is free and open-source and offers rudimentary IDE functionality. It exists inside Raspberry Pi interface for coding purposes. It is intended to load quickly and require little in the way of external libraries or distinct packages on Linux [39].

3.2.14 OpenCV

OpenCV is an open-source computer vision and machine learning library widely used for image and video processing. It provides tools for tasks such as image manipulation, feature detection, object recognition, machine learning, camera calibration, and deep learning. OpenCV is cross-platform, supports various programming languages, and is commonly used in applications like computer vision research, robotics, and augmented reality [40].

3.3 Use of modern engineering and IT tools

3.3.1 CARLA Simulator 0.9.14

CARLA Simulator was used as the primary environment for both approach one and two software simulations. It was ideal for its easy-to-use user interface and keen use of real-world physics. The environmental elements and sensors were easily accessible through appropriate codes which were conveniently provided in their official website. The allowed use of GPS and camera modules perfectly aligned with our design approaches, providing us the perfect scenario to conduct our design and identify improvements.

3.3.2 Python 3.7

We used Python programming language during the simulation to write codes for both of our approaches and mold the environment in CARLA Simulator. The ease of use of the python language helped us to easily grasp the working understanding of the required models and to

quickly write up our code. The vast resources of the Python language on the internet were also a reason for choosing Python as our language of choice in simulation.

3.3.3 Sublime Text 3

We used Sublime Text as our primary source code editor tool to write codes to control the environmental elements within the CARLA Simulator. Its simple interface provided an excellent editing experience.

3.3.4 Anaconda Prompt

Anaconda Prompt served as the intermediary between our codes compiled in Sublime Text, Google Collab, and CARLA Simulator. Anaconda Prompt allowed us to run our codes to change the environmental elements and extract data from sensor modules and execute them in the CARLA Simulator. It also displays extracted data from the sensor modules like GPS and camera. Anaconda Prompt basically provides an easy and swift method to communicate with CARLA Simulator with Sublime Text. It also allowed us to deploy our trained models from Google Collab to CARLA Simulator, allowing the agent vehicle to work autonomously. The model was also coded to extract data which were displayed again in Anaconda Prompt.

3.3.5 Google Collab

We used Collab to run and test our model training. We used its innovative and revolutionary concept of sharing computing resources to run our object detection and decision criteria models. Running a model is very taxing on the computer and requires a very high configuration to train in a timely fashion. Collab allowed us to use a fraction of the computing power available at Google to run our models, leaving us to work efficiently on our personal computers. The fact that Collab is compatible with Python language helped us to work with ease and convenience.

3.3.6 Proteus

Proteus was used in the simulation of the vehicle maneuver subsystem using Arduino UNO and L298N Motor Driver. Proteus allowed us to run and tune a software simulation for our vehicle maneuver subsystem. Using Proteus, we were able to understand the required connection that is needed between the L298H Motor Drive and Arduino, also we were able to deploy the code required by the Arduino to move the vehicle.

3.3.7 Arduino IDE

Arduino IDE provided us with a source code editor to code our Arduino UNO to run to specification. All necessary coding done for the Arduino was done through this IDE. We implemented the code here for our Arduino UNO in proteus software, and to control the motors in hardware according to the decisions made by the Raspberry Pi.

3.3.8 Cascade Trainer GUI.

We used this in hardware to train our 3 object detection models using positive and negative samples for each, and then detecting them in running conditions. It provided us with a simple way of detecting objects, consuming less power for more accuracy when used in Raspberry Pi.

3.3.9 CodeBlocks

We did C++ coding in the hardware system to verify and put in Raspberry Pi for operation. It seemed more convenient to do the coding here.

3.3.10 C++ Programming Language

Our Raspberry Pi provided good support for C++ programming and it is convenient for our project so, we used it to develop our coding for Raspberry Pi during hardware implementation.

3.3.11 VNC Viewer

We used this to connect to our Raspberry Pi for any operation, even when in running conditions in hardware design. It was faster and more efficient to use rather than using our PC's built-in remote desktop connection app to connect and view our Raspberry Pi desktop and real-time camera displays of Raspberry Pi camera.

3.3.12 PuTTY

We initially used PuTTY in hardware to connect to our Raspberry Pi from PC, installed the graphical user interface and other packages, and enabled VNC viewer for our Raspberry Pi, after which we switched to VNC viewer for further operations as PuTTY cannot provide a desktop environment and was comparatively slower than VNC viewer.

3.3.13 Geany Programming Editor

We used the internal Geany Programming Editor inside the Raspberry Pi to build and run our codes, after we designed our code in CodeBlocks during hardware implementation. We further had to modify or tune our code in Geany Programming Editor after running some test runs, to improve our results. So, it helped in doing trials and errors for our system.

3.3.14 OpenCV

We installed OpenCV in our Raspberry Pi for important operations related to our image processing and deep learning models. It provided a comprehensive set of tools and algorithms for image and video processing. It also supported our Raspberry Pi camera, providing an interface to capture and process frames from cameras and enabling seamless integration with our Raspberry Pi camera in hardware.

3.4 Conclusion

Throughout the project's development and execution stages, the aforementioned engineering and IT tools have been employed. With the use of these tools, we were able to evaluate the practicality of each design strategy and select the optimal one for hardware implementation. These tools have sped up the design process by enabling the identification of various strengths and restrictions of the design methods. For the development of the final prototype, these tools also proved to be efficient and helpful to enable a smooth design process for us.

Chapter 4

Optimal Solution and Optimization of the Optimal Solution

4.1 Introduction

After analyzing and comparing the multiple design approaches in simulation, design approach 2 was found to be the optimal design, considering multiple parameters. This design approach was implemented extensively in hardware and several test runs were conducted to analyze its performance. The design fulfilled all the objectives and requirements with acceptable accuracy. However, there was a potential to further increase the accuracy of its operations and so, several optimizations were done on both software and hardware design which gave better results and increased their performance.

4.2 Identification of Optimal Design Approach

We compared the two design approaches in Table 7 below to find out the optimal design approach.

Table 7 Comparison Analysis summary of the three multiple design approaches

Parameters	DESIGN APPROACH 1 (Rule-Based)	DESIGN APPROACH 2 (Machine Learning)
Cost	28,150 TK	28323 TK
Precision	Almost 100% precise but 0% for vehicles or obstacles outside of its GPS detection	Around 89% precise for all obstacles
Adaptability	Limited to the predefined set of rules and conditions	Can adapt to dynamic and ambiguous conditions
Knowledge Representation	Explicit rules are defined by human experts to guide the behavior of the self-driving car	The system learns from data without explicit programmed rules
Real-time Data Requirements	Need real-time coordinates of all objects in the entire region to function properly	Need only real-time image of objects in front of our car to function properly
Complexity	Tracking cars will increase complexity exponentially as every car will be dependent on each other	Less complex as cars are independent from each other
Safety and Liability	Can't detect objects that do not give GPS signals, endangering people without phones.	Safety is higher since decision-making relies on learned patterns data of nearby environments
Availability of components	There is only one sensor (GPS) needed which is very common	More sensors may be required for better performance, which may not be available
Interpretability and Explainability	High since it's decision-making relies on clearly defined rules and logic	Often lack interpretability as difficult to explain the reasoning behind it's decisions
Maintainability	Easier to maintain as one sensor can be fixed easily when damaged	Harder to maintain as tough to fix multiple sensors

From Table 7, we can see a brief analysis of the two design approaches. We can see that both approaches have their merits by being advantageous in five parameters each.

Both approaches seem to have more or less the same costs to produce similar functionality. As we will be using only a single GPS sensor in our rule-based approach, it will be easier to maintain as well as manufacture due to the availability of components. Also, as this approach's rules are constructed by us, it is easy to interpret and explain. Despite the pros of approach 1, there are some vital cons. The major one being the complexity. As each vehicle has to know the location of all vehicles in its vicinity, this creates an exponentially increasing complexity as the number of vehicles increases. Due to dependency on other vehicles, approach 1 can be considered less safe. Despite having a 100% precision, there is the possibility of a rogue vehicle without any means of tracking it, which will lead to the precision to fall to 0%. Not only rogue vehicles, any person without a phone or means of transmitting location information is in danger of not being detected by the vehicles which can become very fatal if the particular situation is not declared in its predefined rules.

On the other hand, the machine learning approach is extremely effective in compensating for the failings of the rule-based approach. It is less complex as it only processes real-time information in close proximity, while providing greater safety as the vehicle can be enabled to learn from its environments and thus be able to react properly to dynamic and ambiguous conditions on the road with the help of its camera, object detection models and decision criteria algorithms.

After analyzing, we can safely conclude that design approach 2 is the optimum solution for our desired outcome. We choose the machine-learning approach mainly because it can effectively work for unexpected situations and cars nearby by sensing them with only requiring real-time front images. Rule-based approaches cannot consider cars outside of their set of rules, and complexity increases exponentially as cars are added to the set of rules, with the problem of not getting GPS signals from every obstacle and requiring real-time coordinates for all vehicles in the region.

4.3 Optimization of the optimal design approach

4.3.1 Optimization of the optimal design in Software

As our design approach-2, the machine learning approach, turned out to be the optimum solution for our desired outcome, to optimize it further we have tried getting more dataset images from CARLA and trained our model again for a third time (Test-3) to get a more accurate model.

We have used around 1500 dataset images to train this time in test-3. We obtained mean average precision (mAP) of 88.2%, a precision of 82.9%, and a recall of 83.2%.

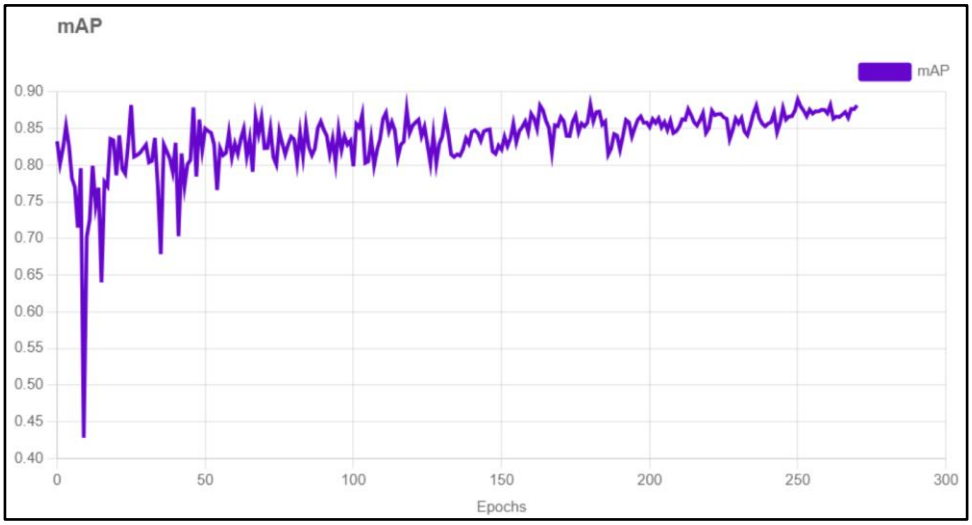

Fig 51: Mean average precision vs epochs for test-3

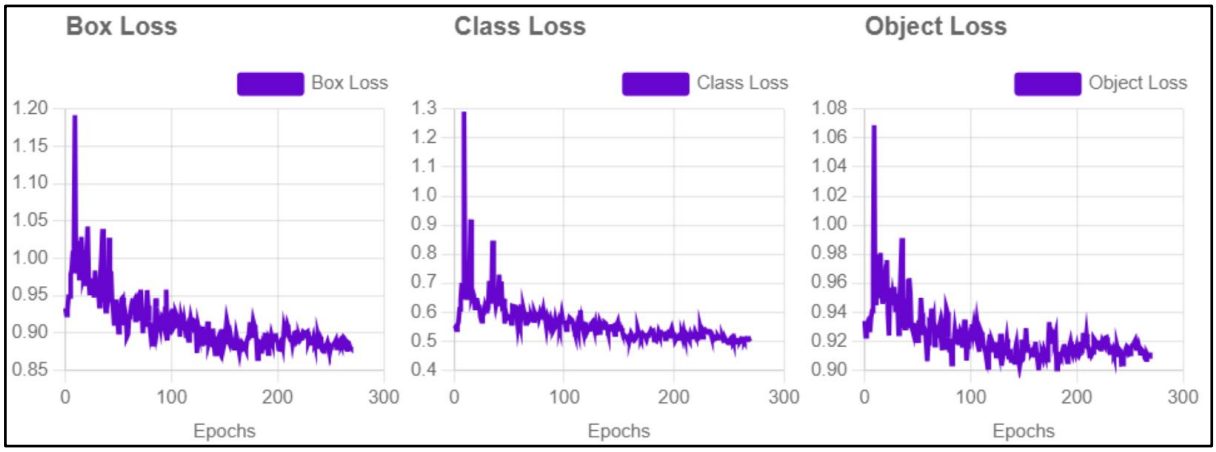

Fig 52: Box loss, Class loss and Object Loss vs epochs for test-3

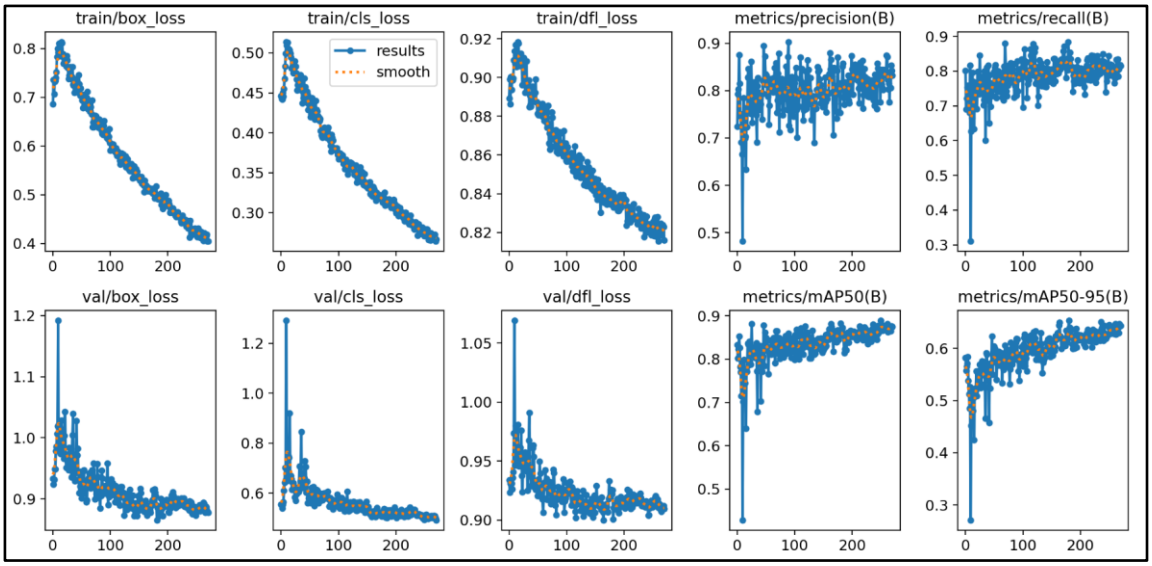

Fig 53: Loss, precision and recall graphs vs epochs for test-3

This time, we notice that the fluctuations and spikes in the graphs further decreased while training our model in test-3 compared to test-2, ultimately giving a higher recall of 83.2%

4.3.2 Optimization of the optimal design in Hardware

Similarly, to optimize our optimal design in hardware further, we made more dataset images using Raspi-cam and used them to train in Cascade Trainer GUI again to get more accurate object detection models for the 3 objects to get detected better. We used around 500 to 600 images to train the models this time and verified the accuracy by running several test runs of our prototype car on our track.

We also had to fine-tune our histogram calculations for more accurate lane detection. Also, the car could not turn enough to maintain its position on a highly curved lane. So, we had to increase the turning speed for those turns in Arduino. When a higher left turn was needed, the speed for the two left-side wheels was decreased further, whereas the speed for the two right-side wheels was increased further. Similarly, a higher right turn was executed.

Moreover previously, when we decided on our action of moving 0.5 seconds after stopping for 4 seconds at a stop sign, we sometimes encountered the problem where even after moving for 0.5 seconds, our car still detected the stop sign for the second time and stopped for a second time as well for 4 seconds. So, to solve this, we increased the time our car has to travel to 1 second from 0.5 seconds, after stopping for 4 seconds, before taking the next decision. In this way, it does not detect the stop sign for a second time after moving 1 second ahead.

4.4 Performance Evaluation of Developed Solution

4.4.1 Performance Evaluation for Software Design

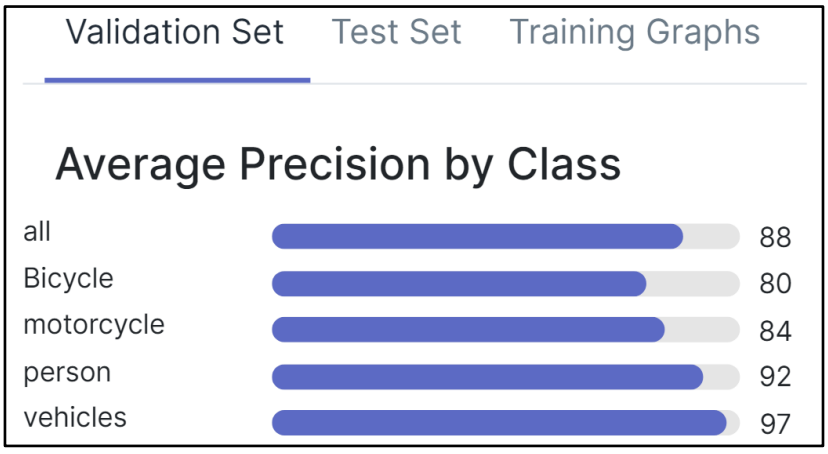

Fig 54: Showing the Average Precision by class for Validation set results for test-3

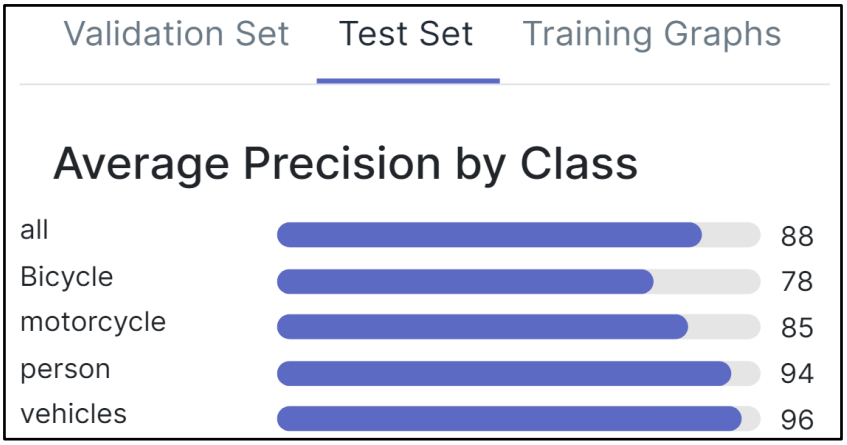

Fig 55: Showing the Average Precision by class for Test set results for test-3

It is clear here that the precision for validation and test, for all the images increased. Our average precision for test images increased from 80% in test-2 to 88% in test-3 for all classes. Especially for the bicycle and motorcycle, their average precision in test images increased from 58% to 78% and from 77% to 85% respectively, which is a huge improvement. If we have more time, these statistics can be further improved by collecting more and more dataset images and annotating them further to train and improve our model.

4.4.2 Performance Evaluation for Hardware Design

We ran several test runs of our prototype car on our track and verified the accuracy before and after optimization. The accuracy of image processing, object detection, decision model, and vehicle maneuver were all found out collectively by determining the successful path our car takes on Lane lines, Lane End, traffic signal, stop sign, and car detection.

Table 8. Accuracy of correct operation for our prototype car before and after optimization

Operation	No. of tests	Accurate Operations before optimization	Accurate Operations after optimization	Accuracy before optimization	Accuracy after optimization
Lane Maintaining operation	23	18	22	78.3%	95.6%
U-Turn operation at Lane End	16	10	12	62.5%	75%
Lane Changing operation at car detection	19	13	16	68.4%	84.2%
Traffic Signal operation	16	12	14	75%	87.5%
Stop Sign operation	18	15	17	83.3%	94.4%

From Table 8 above, we have compared results before and after optimization. It can be seen that accuracy for all of the operations increases due to the optimization done. Our car maintains lane and stops at stop signs with around 95% accuracy, whereas lane changing operation and traffic signal operation had an accuracy of around 85%.

U-Turn operation has relatively lower accuracy (75%) than the others and this is due to the varying speed of our motor. As we run our prototype again and again, the battery that drives the motors depletes continuously. This causes our motors to run slower and this hampers the U-Turn operation. When our car has to take a U-Turn, specific Arduino commands are run with fixed specific speeds to the motor drivers for our car to turn, go forward, turn again, and continue. During this time, our car does not detect any other things so, if speed varies at this time, it may move out of the lane sometimes. We can improve this by using a stable power supply for our motor drivers, instead of batteries that deplete and change speed over time.

So, overall we got 81 accurate operations out of 92 test runs and so, accuracy of our optimized system was 88% altogether.

4.5 Conclusion

In this chapter, we found out that the second design was the most optimal one based on the analysis in Table 7 considering cost, precision, adaptability, knowledge representation, real-time Data, requirements, complexity, safety and liability, availability of components, interpretability and explainability, and maintainability. This design approach was implemented in hardware with some general modifications and its performance was evaluated. For further optimization of the optimal design, machine learning models were trained with more datasets, histogram calculations were further fine-tuned, turning speed was increased for higher turns, and time for moving past stop sign was increased. After these optimizations, the system gave much better results and accuracy during test runs. The overall accuracy of our optimized system was 88%, combining all the test runs.

Chapter 5

Completion of Final Design and Validation

5.1 Introduction

After various optimizations and considerations, we got our final design for the machine learning approach in the hardware system. Our final system was judged based on 5 actions: Maintaining Lane operation, U-turn operation at Lane End, Lane Changing operation at car detection, Traffic Signal operation, and Stop Sign operation. The optimized design was tested 92 times and in 81 of those cases, our car was found to be working properly. In this chapter, we will elaborate on our final design and evaluate each and every operation of our car.

5.2 Completion of Final Design

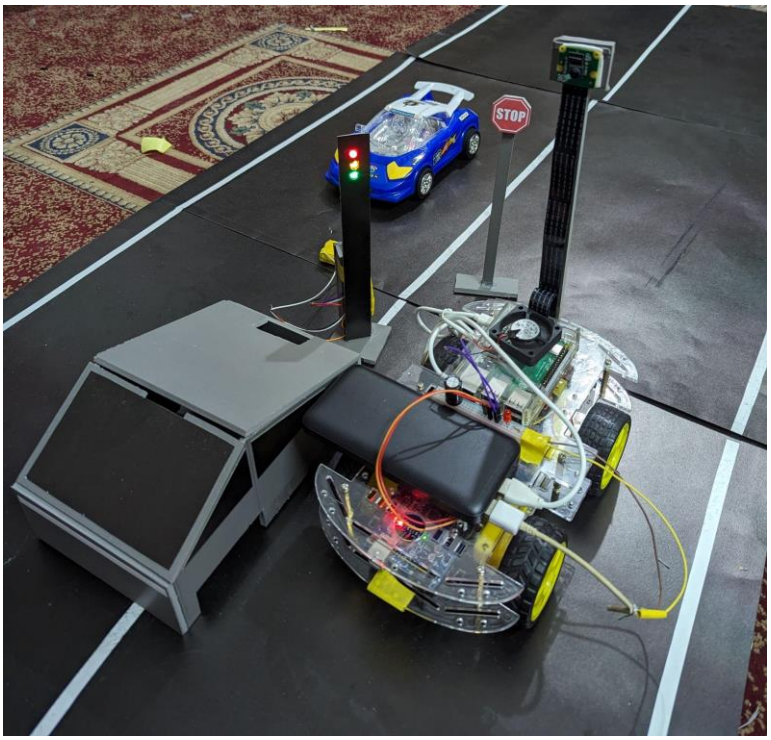

Fig 56: Final Design of self-driving car prototype, along with other objects

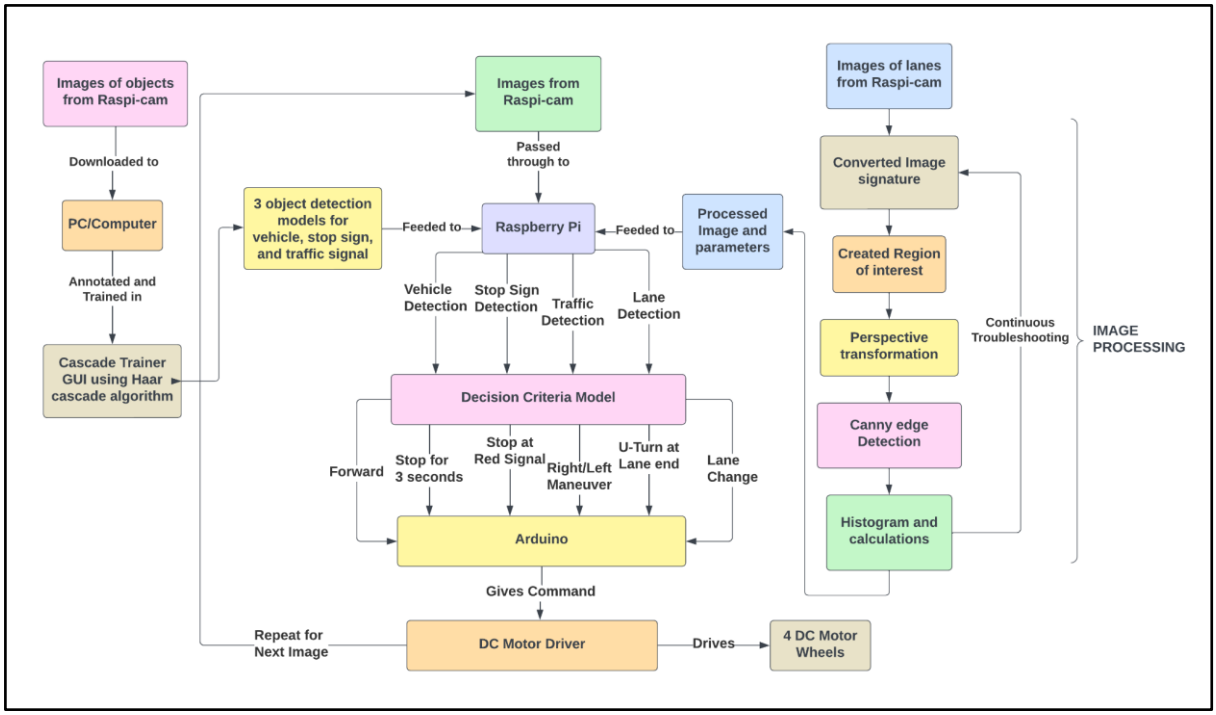

Fig 57: Final Design System flowchart

For our final design, we used deep learning (Haar cascade), openCV, image processing and decision criteria to effectively train, detect, and run our self-driving car in our lanes. This is an embedded IOT design as well, with machine learning.

Our main subsystems are fed into Raspberry Pi, which will detect, decide, and send commands to the Arduino to maneuver the wheels accordingly using the motor driver and motors.

Our Arduino can conduct some specific action commands, such as forward, backward, stop, and U-turn to send to our motor drivers according to the decisions taken by Raspberry Pi. 3 types of left movements and 3 types of right movements exist, each having a different magnitude of turn, and only one of them occurs every time our raspberry decides on relative lane center position after analyzing the image it is getting, and sends its decision to the Arduino through its 4 digital pins to adjust and maintain the lane.

5.2.1 Image processing

Our Raspberry Pi does image processing for lane detection, where it converts image signature (BGR to RGB), creates a region of interest, does perspective transformation (Bird Eye View) on that region, threshold operations to extract white lane lines, canny edge detections on those lines, finds exact lane positions from the lines using histogram and various calculations, calibrates to map lane center to frame center and finally find the relative difference between them to know how much our car needs to move left or right to match its frame center to the lane center. This is sent to the Arduino for command.

5.2.2 Object Detection using Haar Cascade Algorithm

Our Raspberry Pi also uses 3 trained object detection models from Cascade Trainer GUI software that uses the Haar cascade algorithm. These were trained 3 times for 3 objects using 3 sets of positive and negative image samples in Cascade Trainer GUI to obtain 3 models. These are fed into the Raspberry Pi to detect them in running conditions. For all the objects, Raspberry Pi makes decisions when it is detected to exist between 5 to 20 centimeters ahead of the car, and sends commands accordingly to the Arduino for vehicle maneuver with the help of 4 digital pins.

The whole process of detecting, analyzing, making decisions, and moving is repeated for the next image obtained from a Raspberry Pi camera, for the continuous operation of our self-driving car.

5.2.3 Decision Model

To use the pixel dimensions of the bounding boxes and determine the distance from the objects, the following equations were used by our Raspberry Pi:

For both stop sign and traffic signal detection, distance = $-1.07 \times \text{width} + 102.597$

For car detection, distance = $-0.48 \times \text{width} + 56.6$

Additionally, our Raspberry Pi can find out our deviation value from our lane, and the Lane End intensity value using image processing. It makes decisions based on them, as well as the 3 object detections. Finally, there will be 11 conditions to choose from and the respective commands, in the form of digital numbers of 4 bits, will be sent to the Arduino for vehicle maneuver, as shown in Fig. 58.

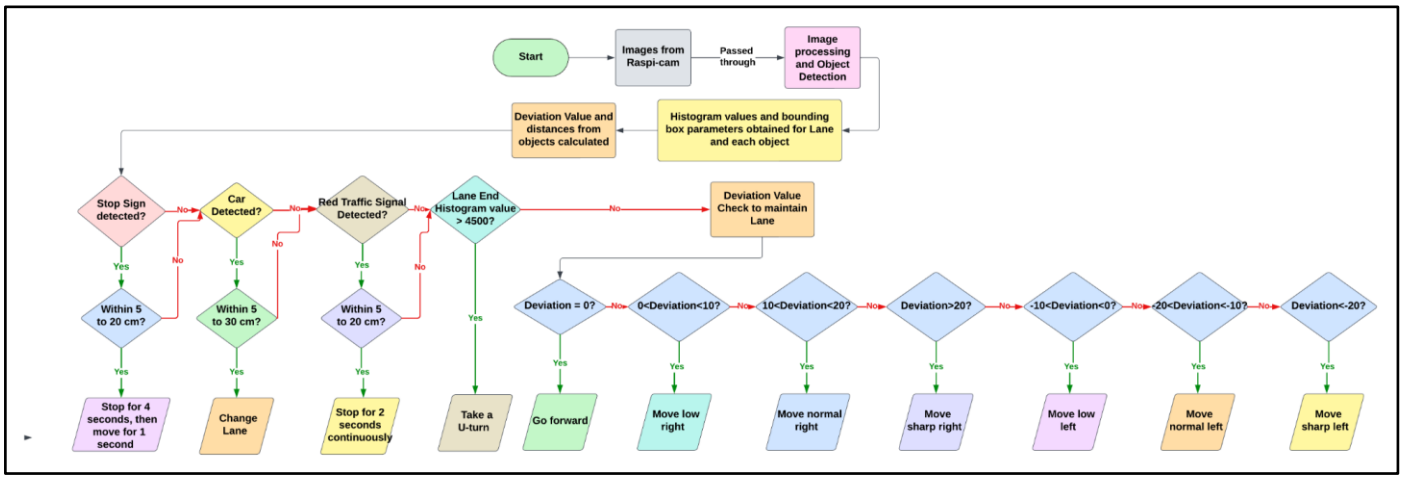

Fig 58. Decision model of final design

If a stop sign is detected and the distance is more than 5 cm but less than 20 cm, then the Raspberry Pi sends a command to the Arduino to stop the car for 4 seconds, and then move ahead for 1 second to overtake the stop sign and take the next decision.

If a traffic signal with a red light is detected and the distance is more than 5 cm but less than 20 cm, then the Raspberry Pi sends commands to the Arduino to keep on stopping the car for 2 seconds continuously, until the red light changes to green light, or turns off completely.

If a car ahead is detected and the distance is more than 5 cm but less than 30 cm, then the Raspberry Pi sends a command to the Arduino to make the car change its lane. For this, a turn is taken using motor drivers, followed by a forward operation and then a reverse turn to align with the new lane. Then, after continuing forward on the new lane for some time while checking for other detections, the car comes back to its previous lane in a similar way with a different direction, ultimately overtaking the car that it had detected before.

If the histogram value of the region of interest exceeds 4500, then it indicates a lane end, and the Arduino makes our car take a U-Turn with the help of the motor driver. It rotates 90 degrees in one direction, moves forward for 1 second, then rotates 90 degrees in the same direction, before continuing to run on the second lane.

The rest 7 conditions are kept to maintain our car on the lane. If the deviation value is 0, the car moves forward. If the deviation value comes out to be between 0 to 10, then a low right turn is executed by varying the velocities of the left and right wheels. If the deviation value comes out to be between 10 to 20, then a normal right turn is executed. If the deviation value comes out to be more than 20, then a high right turn is executed. Similarly low, normal, and high left turns are executed but for negative deviation values, depending on the magnitude.

5.3 Evaluation of the solution to meet desired needs

5.3.1 Maintaining Lane

As discussed before, for our car to maintain its lane, the deviation value is calculated from the processed image and the car maneuvers accordingly based on that value.

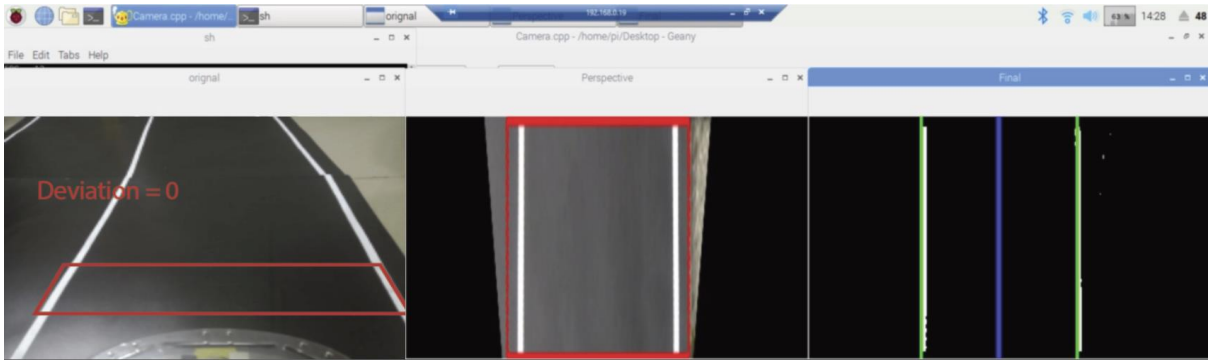

Fig 59: Vehicle having its frame center match with lane center perfectly, with no deviation

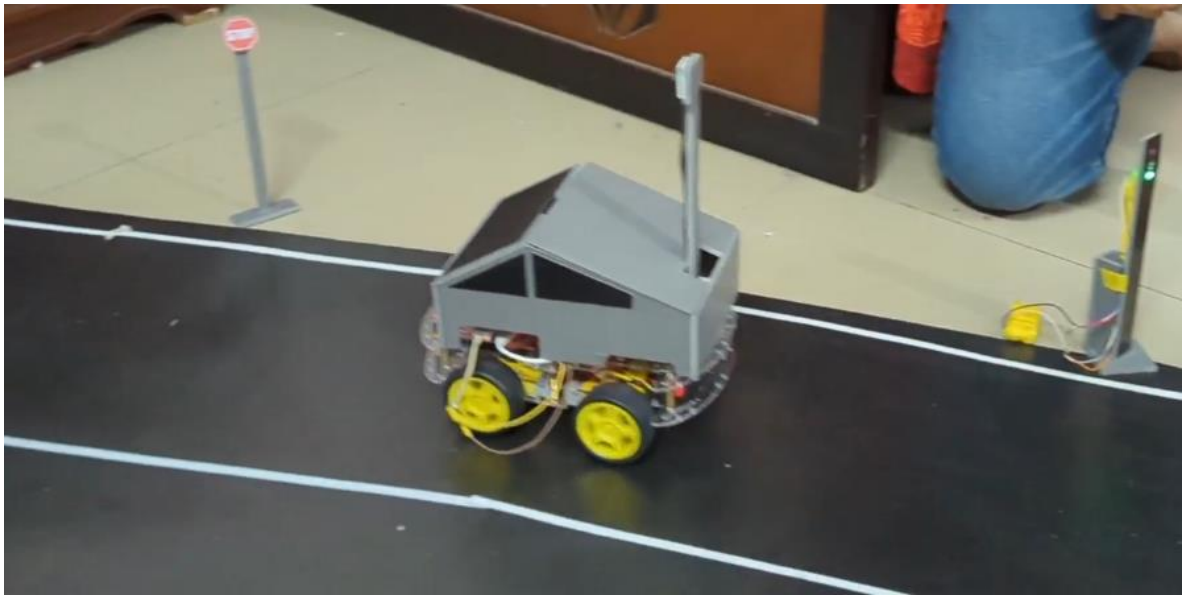

Fig 60: Vehicle moving forward after getting a deviation value of 0

In Fig. 59, our self-driving car is at the center of the lane, with its front frame center matching perfectly with the lane center. So, there is no deviation and our car continues to move forward as shown in Fig. 60.

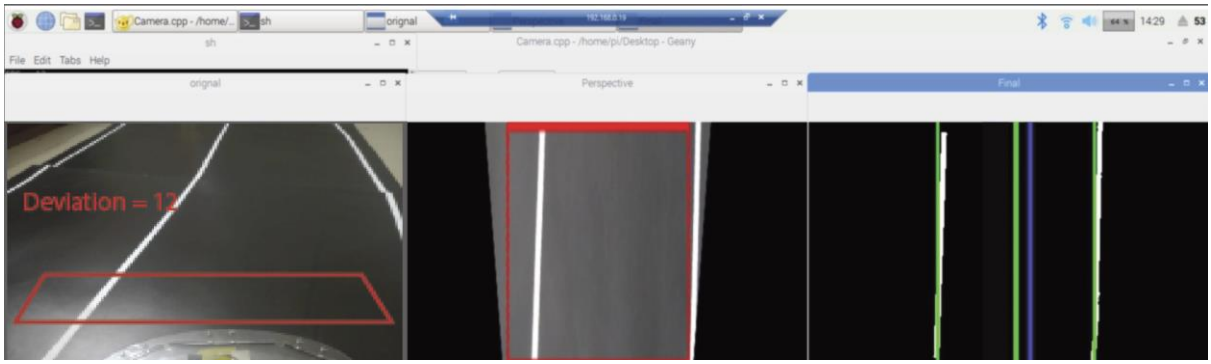

Fig 61: Vehicle having a difference of 12 units between its frame center (blue) and lane center (green)

In Fig. 61, our self-driving car is facing left compared to the lane, with its front frame center having a difference of 12 units with the lane center. So, there is a positive deviation value and our car needs to move normally right according to the decision model until it matches its frame center with the lane center.

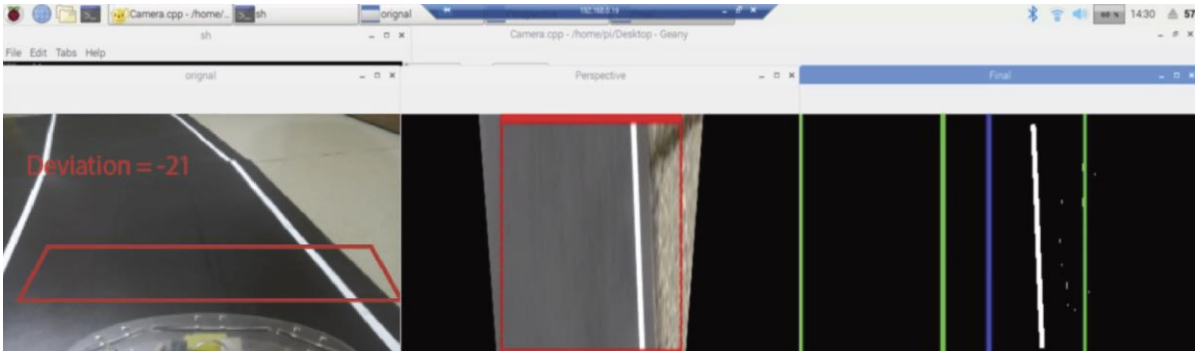

Fig 62: Vehicle having a difference of -21 units between its frame center (blue) and lane center (left green)

In Fig. 62, our self-driving car is facing right compared to the lane, with its front frame center having a difference of -21 units with the lane center. So, there is a deviation value less than -20 and our car needs to move to a high left according to the decision model until it matches its frame center with the lane center.

5.3.2 U-Turn Operation at Lane End

As discussed before, if the histogram value of the region of interest of the processed image exceeds 4500, then our car decides to take a U-Turn

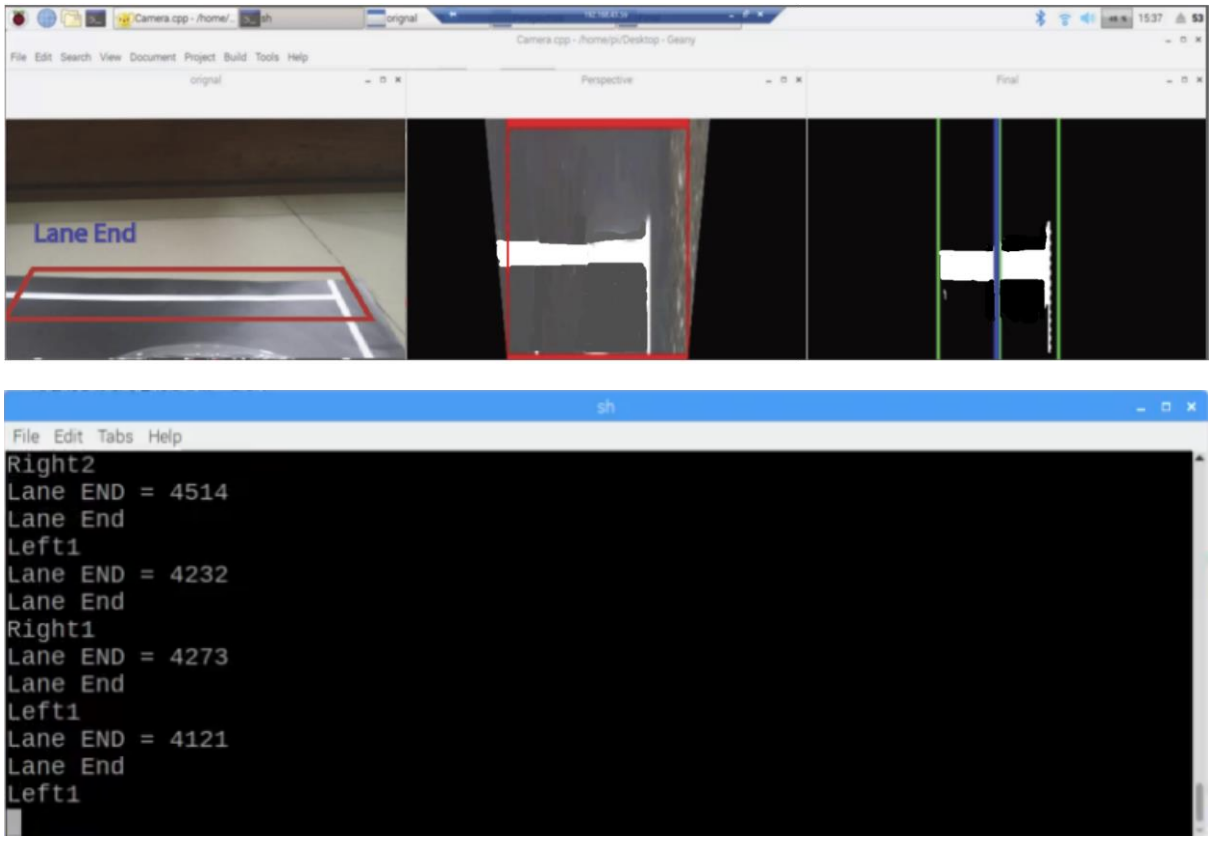

Fig 63: Vehicle detecting Lane End after reaching a value of 4500 and deciding on taking a U-Turn

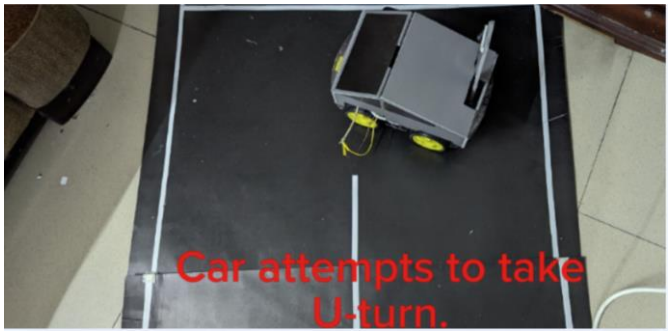

Fig 64: Vehicle taking a U-Turn

5.3.3 Lane Change operation at car detection

As mentioned before, our car decides to change lanes after detecting a car ahead within 5 to 30 cm. Raspberry Pi calculates the width of the bounding box in pixels and finds out the distance the detected car is from our vehicle, before deciding on changing lanes.

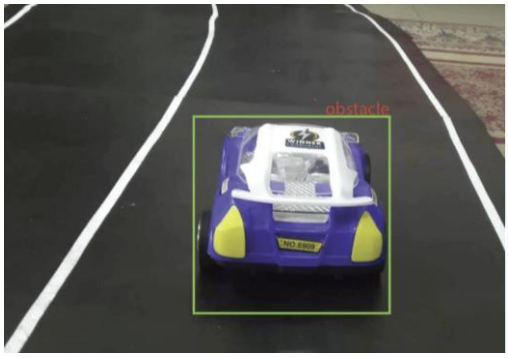

Fig 65: Vehicle detecting car ahead with a bounding box width of 93 pixels

In Fig. 65, the vehicle detects a car ahead having a bounding box width of 93 pixels. To find out the distance using this,

$$\text{Distance} = -0.48 \times 93 + 56.6 = 11.96 \text{ cm}$$

So, this distance falls between 5 to 30 cm and the vehicle decides to change lanes to the left, as shown in Fig. 66

Fig 66: Vehicle detecting car ahead within 5 to 30 cm and changing lanes to the left

5.3.4 Traffic Signal operation

Our car stops if it detects a traffic signal with a red light ahead between 5 to 20 cm. It stops continuously for 2 seconds if it keeps on detecting the red traffic signal between 5 to 20 cm. If green signal is given, then it does not detect it anymore and continues along the lane, as green signals were considered negative samples while training its object detection model. Raspberry Pi calculates the width of the red traffic signal bounding box in pixels and finds out the distance the detected traffic signal is from our vehicle, before making a decision of stopping.

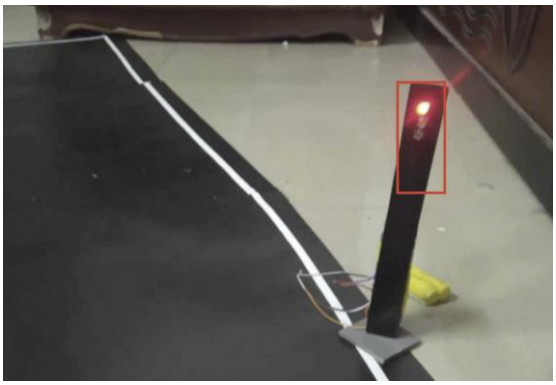

Fig 67: Vehicle detecting traffic signal with red light ahead with bounding box width of 87 pixels

In Fig. 67, the vehicle detects a red traffic signal ahead having a bounding box width of 87 pixels. To find out the distance using this,

Distance = $-1.07 \times 87 + 102.597 = 9.507$ cm

So, this distance falls between 5 to 20 cm and the vehicle decides to stop as shown in Fig. 68, until the traffic turns green and it does not detect a traffic signal anymore and continues forward as shown in Fig. 69.

Fig 68: Vehicle detecting red traffic signal ahead within 5 to 20 cm and decides to stop

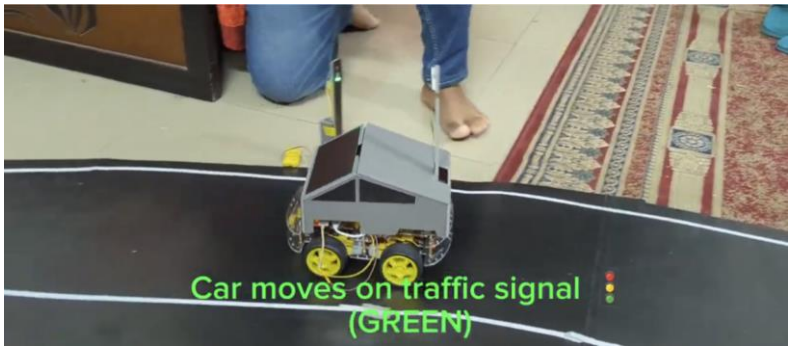

Fig 69: Vehicle no more detecting red traffic signal ahead within 5 to 20 cm and decides to move on

5.3.5 Stop Sign operation

As mentioned before, our car stops for 4 seconds and then moves on for 1 second, if it detects a stop sign ahead within 5 to 20 cm. Raspberry Pi calculates the width of the stop sign bounding box in pixels and finds out the distance the detected stop sign is from our vehicle, before making a decision of stopping for 4 seconds and then moving on for 1 second.

Fig 70: Vehicle detecting stop sign ahead with bounding box width of 90 pixels

In Fig. 70, the vehicle detects a car ahead having a bounding box width of 90 pixels. To find out the distance using this,

Distance = $-1.07 \times 90 + 102.597 = 6.297$ cm

So, this distance falls between 5 to 30 cm and the vehicle decides to stop for 4 seconds and then moves on for 1 second before taking the next decision, as shown in Fig. 71 below.

Fig 71: Vehicle detecting stop sign ahead and decides to stop for 4 seconds before moving on for 1 second

5.3.5 Performance Analysis

We ran several test runs of our prototype car on our track and verified the accuracy of the final design. The accuracy of image processing, object detection, decision model, and vehicle maneuver were all found out collectively by determining the successful path our car takes on Lane lines, Lane End, traffic signal, stop sign, and car detection.

Table 9. Accuracy of our self driving car's operations

Operation	No. of tests	Accurate Operations	Accuracy
Lane Maintaining operation	23	22	95.6%
U-Turn operation at Lane End	16	12	75%
Lane Changing operation at car detection	19	16	84.2%
Traffic Signal operation	16	14	87.5%
Stop Sign operation	18	17	94.4%

From Table 9 above, it can be seen that accuracy for all of the operations are quite acceptable. Our car maintains lane and stops at stop signs with around 95% accuracy, whereas lane changing operation and traffic signal operation had an accuracy of around 85%.

U-Turn operation has relatively lower accuracy (75%) than the others and this is due to the varying speed of our motor. As we run our prototype again and again, the battery that drives the motors depletes continuously. This causes our motors to run slower and this hampers the U-Turn operation. When our car has to take a U-Turn, specific Arduino commands are run with fixed specific speeds to the motor drivers for our car to turn, go forward, turn again, and continue. During this time, our car does not detect any other things so, if speed varies at this time, it may move out of the lane sometimes. We can improve this by using a stable power supply for our motor drivers, instead of batteries that deplete and change speed over time.

5.4 Conclusion

We can conclude that our final design meets all of our objectives and requirements with good accuracy. Our car can accurately maintain its lane, take a U-turn at Lane End, and change lanes at car detection. Moreover, it can stop at red traffic signals and start moving at green traffic signals. It also achieved the objective of stopping at stop signs. All of these operations had an overall accuracy of 88% for 92 test runs that we executed on our hardware design. So, we can safely say that all of our objectives and project requirements were met successfully.

Chapter 6

Impact Analysis and Project Sustainability

6.1 Introduction

Project impact analysis explains how the project will impact the stakeholders, the surroundings like the environment, economy, and even people in the future. Here the aim is to understand the likely impacts of the project both favorably and unfavorably in the future. This part is crucial in understanding the events and consequences that our project holds for the future. Here, the project impact analysis can be described as - safety, societal, environmental, economic and legal.

6.2 Assess the impact of solution

6.2.1 Safety Impact

Implementing this design will help with safe driving and significantly avoiding road accidents and fatalities. Passengers and other passersby on the roads will feel safe knowing about this project implementation.

Many news reports have been heard and observed where drivers get into accidents during changing lanes, especially in highways with cars at high speeds. Accidental issues also appear when drivers feel tired and fall asleep while driving, and even sometimes when under influence of alcohol or other drugs. With our self-driving technology and automatic lane changing system, cars will be able to react very quickly, even quicker than human drivers in accordance with the surrounding environment[41]. This will positively affect the safety of the stakeholders like all passengers and other bystanders, and in turn increase the quality of life.

6.2.2 Societal Impact

With the implementation of our design, the self-driving system greatly affects people elderly people, people unable to drive and even disabled people like amputees, people with vision and hearing impairments[42].

The design we have implemented in cars follows smooth lane maintenance, lane changing, traffic light detection, detection of other traffic on the road, stop sign detection and can even take U-turns. This requires very less supervision from human drivers. Human drivers in long drive need periodic stops to rest and regain energy to drive, whereas the automated system designed requires little supervision and input from drivers which in turn puts less stress on the human drivers and they can drive for a longer period of time. Amputees like people with one hand or one leg will be able to drive using this system smoothly without any trouble, even people with vision and hearing impairments can use this system to drive cars during emergencies since very little supervision is required manually from the drivers. In addition to this, another societal impact will be consumer skepticism. Consumers will not blindly trust this system as this is new in the market and has not been used for a very long time yet.

6.2.3 Environmental Impact

Automobiles, especially cars, have high carbon emissions and contribute to global warming. With an increasing number of vehicles on the road every year, it is vital to analyze this situation and bring the carbon emissions under control somehow.

The system designed here will not have any carbon emissions or environmental inpacts itself. But the system has an automated braking and accelerating process which will smoothly take decisions when to brake and when to accelerate following the surroundings. With maintaining average speed and gradual acceleration and deceleration, the cars with this system can maintain fuel efficiency, thereby burning less fuel and contributing less towards global warming.

6.2.4 Economical Impact

New cars with this self-driving technology will outgrow the old technologies, making them obsolete. This in turn will force the old car manufacturers to close down their factories, dump their machinery and sack their employees. This might greatly impact the economy of many countries as factory workers, Uber drivers and taxi drivers will be out of jobs. On the bright side, with these new technologies, car manufacturers will have to open new factories and replace their old technologies, which will create job displacements. Existing employees will learn new skills and techniques, and many new engineers and technicians specializing in self-driving systems will be hired, which will increase new employment opportunities and create a whole new demand market for workers.

6.2.5 Legal Impact

Like most self-driving cars, our system will collect vast amounts of personal data including real-time location information and many other data from the additional sensors. These data generated are exclusive to each car and their drivers and all the data are sent to cloud servers for better maintenance of safety and road rules. These data are crucial for the driver and passengers and must be protected at all costs [43].

Our system follows every required code and applicable standards linked to the system. The idea of self-driving cars is very delicate when it comes to implementation, as people have trust issues with the performance of self-driving cars from the beginning. So maintaining all the legal rules and regulations while manufacturing the system is a legal necessity which must be addressed.

6.3 Evaluate the sustainability

The main components used for the project are Raspberry Pi, arduino, motor driver and micro SD card which are very durable and cost efficient. These can be made further durable by applying protective layers around them, inside the car systems for long-term usage. These components have very low power consumption compared to other car systems which make them economically sustainable and productive for a long time. The system does not have any environmental impact itself, so it can be said that it is very safe. However regular tests must be done to check the accuracy of the system, and small maintenance will be required from time to time.

6.4 Conclusion

To conclude, this system provides mostly a self-driving system which will require little to no human intervention. Human drivers and passengers can rest at ease knowing their car will safely maintain and change lanes and operate properly. In addition to this, the system offers more safety features than existing car technologies as the system has lower reaction time to operate than human drivers. Moreover the manufacture of the system is very cost efficient and long lasting which makes the system very durable and sustainable. Last but not the least, the legal requirements of the system falls in with all the applicable codes and standards of road rules and regulations.

Chapter 7

Engineering Project Management

7.1 Introduction

A good management plan is necessary to have an orderly process for the successful outcomes and to ensure that all of the project's objectives are completed within the allotted time limit. A study in [26] found that there are five primary groups into which the project's entire flow can be categorized. They are initiation, planning, execution, monitoring and control, and closure.

Fig: 72 Five Key Factors of Engineering Project Management

7.2 Define, plan and manage engineering project

7.2.1 Initiation

Any project should start with a problem that has to be solved. Brainstorming is an essential task to identify the problems that require solution. After choosing an issue, it is crucial to research previous and existing solutions to the problem and conduct a literature review. Finding the research gap is then an essential step in introducing novel solutions or improving ones that already exist.

7.2.2 Planning

Before submitting the project proposal, we had a consultation with an expert in the area to better grasp the functional and non-functional needs of the project. When designing the system, the expert's feedback was carefully taken into account. Different design strategies were suggested, all based on the system's requirements and goals. Using modern engineering and IT tools, each system was simulated, and its performance was assessed. At last, the optimal design

was chosen according to cost, precision, adaptability, knowledge representation, real-time data requirements, complexity, safety and liability, and interpretability and explainability.

7.2.3 Execution

Purchasing the Hardware components was the initial task. Managing all the components together was the biggest difficulty. Some components were totally new to work as we have never worked with them. We had to go through the hardware specifications and learn about their nature and work process from the data sheet. Finally, we tested the hardware and started with our work. There were lots of differences between the theoretical literature and the practical implementation. To make the process more well-organized and efficient we had to troubleshoot thoroughly in every process. Data collection was another tedious work, where a tremendous amount of time and effort was given to collect personalized dataset. Once the dataset was ready, we had to annotate the images specifically in one-by-one nature. Upon completion of data collection we had to recheck the data and then started to work to assure ourselves that the personalized data will let us achieve our functional and non-functional requirements. Finally, once our system was ready we had to troubleshoot multiple times to make sure that our project fulfills four objectives finely.

7.2.4 Monitoring and Control

In order to determine whether the objectives and requirements were satisfied, the system validation was carried out based on the system. Following system validation, the data were examined to assess the system's accuracy and evaluate whether the design still needs to be optimized. Additionally, a backup strategy for risk management was created.

7.2.5 Closure

Through analysis of the data prior to and following the optimization, the system underwent a comprehensive examination. To evaluate the accuracy of the answer even more, the data were compared and the recognition success rates were calculated. Furthermore, the outcomes and development have been consistently tracked and documented.

7.3 Gantt Chart/Project Timeline

The three phases of project management—project planning, project development, and project completion—were systematically divided into phases that were carried out over the course of the year. The project's progress was monitored by assigning deadlines to defined tasks and allocating responsibilities to guarantee that the work was finished on time and with accountability. The Gantt Charts are depicted in the following Figures.

Phase		Details			EEE499P																										
		Month			JAN					FEB					MAR							APR										
		Task	Assigned to	Duration (Days)	1	7	14	21	31	1	7	14	21	28	1	7	14	21	31	1	7	14	21	30
EEE499P	Project Planning	Research & Literature Review	Shehab, Mashook, Tahmid	38						Feb 9 - Mar 18															
		Problem Identification & Planning	Mashook, Mrinmoy, Shehab, Tahmid	23										Feb 24 - Mar 18											
		Concept Note	Mashook, Mrinmoy, Shehab, Tahmid	19										Feb 28 - Mar 18											
		Expert Consultation	Mashook, Mrinmoy, Shehab	12																Apr 3 - Apr 14					
		Justification	Mashook, Mrinmoy, Shehab, Tahmid	52										Feb 28 - Apr 21											
		Project Proposal	Mashook, Mrinmoy, Shehab, Tahmid	37																Mar 23 - Apr 28					

Fig: 73 Gantt Chart/Project Timeline for EEE499P

Phase		Details			EEE499D																										
		Month			MAY					JUNE						JULY						AUG										
		Task	Assigned to	Duration (Day)	1	7	14	21	31	1	7	14	21	30	1	7	14	21	31	1	7	14	21	31
EEE499D	Project Development	Tools Selection	Mashook, Mrinmoy, Shehab, Tahmid	17			May 20 - June 5																		
		Simulation of Multiple Designs	Mashook, Mrinmoy, Shehab, Tahmid	75						May 30 - Aug 12															
		Optimal Solution Development	Mashook, Mrinmoy, Shehab, Tahmid	12												July 28 - Aug 8									
		Evaluation	Mashook, Mrinmoy, Shehab	14																Aug 1 - Aug 14					
		Design Report	Mashook, Mrinmoy, Shehab, Tahmid	37												July 18 - Aug 23									

Fig: 74 Gantt Chart/Project Timeline for EEE499D

Phase		Details			EEE499C																										
		Month			SEPT					OCT					NOV						DEC										
		Task	Assigned to	Duration (Day)	1	7	14	21	30	1	7	14	21	28	1	7	14	21	30	1	7	14	21	31
EEE499C	Project Completion	Design Implementation	Mashook, Mrinmoy, Shehab, Tahmid	52					Sept 30 - Nov 21																
		Performance Analysis	Mashook, Mrinmoy, Shehab, Tahmid	47						Oct 14 - Dec 1															
		Validation	Mashook, Mrinmoy, Shehab, Tahmid	47						Oct 14 - Dec 1															
		Project Reflection	Mashook, Mrinmoy, Shehab	60						Oct 14 - Dec 14															
		Report	Mashook, Mrinmoy, Shehab, Tahmid	50											Nov 1 - Dec 20										
		Project Closing	Mashook, Mrinmoy, Shehab, Tahmid	21																Dec 1 - Dec 22					
		Conference Paper	Mashook, Mrinmoy, Shehab, Tahmid	34	Sept 1 - Oct 4																				

Fig: 75 Gantt Chart/Project Timeline for EEE499C

7.4 Conclusion

By properly prioritizing work, adhering to a project management scheme can assist and ensure that the project is completed on schedule. Arranging the chores according to their right sequence also helps. It also shows the comparison between the estimated and real amount of time needed. Additionally, assigning responsibilities to members guarantees their participation and improves teamwork.

Chapter 8

Economical Analysis

8.1 Introduction

A project's cost and benefit are broken down in an economic analysis. The following are economic analysis's key components:

- Determining and estimating costs directly related to investment
- Determining and estimating benefits obtained from investment
- Comparing cost and benefits to justify investment

8.2 Economic analysis

8.2.1 Budget for working prototype

Table 10. Budget for the Final Design

Sub System	Component	Model	Unit Price (BDT)	Unit	Total Cost (BDT)
Object Detection	Camera	Raspberry Pi Camera Module V2	4,690	1	4,690
	Flex Cable	Adafruit Flex Cable for Raspberry Pi Camera	1227	1	1227
Processing Unit	Raspberry Pi	Raspberry Pi 3B+ Motherboard	11255	1	11255
	Case	Raspberry Pi 4 Acrylic Case	299	1	299
	Storage	SanDisk Ultra 16GB Micro SDHC UHS-I	1200	2	2400
Vehicle Maneuver	Body and Motor	4 Wheel 2 Layer Robot Smart Car Chassis Kits with Speed Encoder	833	1	833
	Arduino	Arduino Uno R3	1100	1	1100
	Motor Driver	L298N H-Bridge Dual Motor Driver	195	1	195

Power Supply	Power Bank	Meko Power Bank 10000mAh	1,699	1	1,699
	Battery	Li-ion Rechargeable Battery	70	16	1120
	Battery Charger		700	1	700
Miscellaneous	USB type B cable, Ethernet cable, LEDs, Toy car, Plastic wood, Jumper wire, Resistors				2805
			Total Cost		28323

Table 8 shows the total cost for the working prototype to be BDT 28323. This however excludes the components that have been used in trial and error scenarios

8.2.2 Estimated Cost

Table 11. Estimated cost for the working prototype, first unit and commercial product

Cost of working prototype with spare parts (BDT)	Estimated Budget for first unit (BDT)	Estimated Budget for commercialized product (BDT)
30000	20000	18000
● Bought extra components for contingency ● Extra cost of having ship components ● Additional implementation of demo/simulated environment and vehicle maneuver sub-system	● Main system consisting of object detection and processing unit subsystems only ● Specific number of components will be bought ● Components bought from local vendors	● Will enjoy economies of scale for bulk component purchase ● Components will be factory assembled ● Development of proprietary components to further reduce cost and size

Table 9 shows the estimated cost of working prototype, first unit and commercialized units. The estimated cost of the first unit is lower as compared to the prototype as there are no trial and error situations. Additionally, the prototype concurred with the cost for the implementation of supportive sub-systems to validate the functionality of the main system. Commercialized units are estimated to cost much less than the first unit due to the development of proprietary components as well as economies of scale due to bulk buying.

8.3 Cost-benefit analysis

Table 12. Profit Estimation with respect to Number of Companies in Contract

Estimated Profit/unit (BDT)	Estimated number of car companies in contract	Estimated number of cars to be fitted with system/contracted companies	Total Estimated Profit (BDT)
7000	2	10	140000
7000	3	15	315000
7000	5	15	525000

In Table 10, we get the projection of the estimated profit. Taking into account that the commercialized product will cost BDT 18000, it is estimated to be sold at BDT 25000, which will result in a profit of BDT 525000 if 5 car companies are contracted, where each company assembles 15 cars with our system.

8.4 Evaluate economic and financial aspects

Table 13. Profit Estimation from Number of Units Sold per Year

		Cost of sales (BDT)	Sale (BDT)	Profit (BDT)	Loss (BDT)
Cost of production/unit (BDT)	18000	XXXXXX	XXXXXX	XXXXX	XXXXX
Selling price/unit (BDT)	25000	XXXXXX	XXXXXX	XXXXX	XXXXX
Production/year /unit	20	360000			
No. of units sold/yer	20		500000	140000	0
Production/year /unit	50	900000			
No. of units sold/yer	40		1000000	100000	0
Production/year /unit	90	1620000			
No. of units sold/yer	75		1875000	255000	0

In Table 11, we have given an arbitrary assumption, estimating the production and selling of units per year and the resulting profit or loss per year. From the table we can observe that selling each unit with a markup of BDT 7000 for a price of BDT 25000 leads to a profitable business. We can observe from the table that even while producing 90 units at a cost of BDT 1620000 and selling only 75 units, we still manage to be profitable with an annual profit of BDT 255000.

8.5 Conclusion

Conducting a thorough economic analysis is necessary in order to forecast the total project expenses. Establishing a budget for the prototype in development is also crucial. Together with the budget, the cost-benefit analysis helps to determine the system's financial prospects and can motivate stakeholders to take an active role in the project.

Chapter 9

Ethics and Professional Responsibilities

9.1 Introduction

Ethical considerations and professional responsibilities must always be considered at all costs while dealing with any engineering projects. For all projects resources must be used efficiently and the project developed must be for the best interest of the people. This helps in acknowledging all the possible risks of the project and minimizing them. Throughout the development of the project, all the ethical considerations and professional responsibilities required and demanded by the problem statement, are recognized and taken into account.

9.2 Identify ethical issues and professional responsibility

9.2.1 Informed Consent

Every consumer of our project shall be informed beforehand about all the technicalities, about what our system does and how it operates, what kinds of real-time data it collects and how it analyzes and uses those data to operate. There must be clear understanding and transparency for both the drivers and the passengers since any discrepancies might lead to severe consequences like accidents and deaths.

9.2.2 Privacy and Confidentiality Protection

Since all data collected and used by the sensors and Raspberry Pi are temporarily stored in cloud servers, data breach and hacking is a major concern, even though the storage is temporary. Moreover major problems might occur when data is hacked during driving, which can lead to severe accidents.

9.2.3 Acknowledgement of Proper Sources

To come up with the solution of our problem statement, many applicable past papers and works have been thoroughly studied and reviewed. Proper credits, citations and references have been given to wherever necessary.

9.2.4 Approval from Respective Bodies

Since our project is a system development of cars which will operate on roads, many approvals had to be taken from Bangladesh Road Transport Authority(BRTA). This respective body controls the registration of all the production and manufacture of systems related to transportations. They have predetermined and set guidelines which strictly follow all the road rules and safety regulations that needs to be maintained for any system manufactured[43].

9.3 Apply ethical issues and professional responsibility

9.3.1 Informed Consent

The users of the Lane changing car system will be accordingly given the knowledge about the self driving car system briefly when used in industry level. As our project focused on prototype level, there were no passengers involved and as a result we did not need any informed consent.

9.3.2 Privacy and Confidentiality Protection

Data hacking and security breaching must be stopped at all costs. Actions must be taken to minimize the chances of security breaches and hacking to protect private data of drivers and the car. The real-time data that is being collected should be encrypted to diminish the chances of hacking at any given time.

9.3.3 Acknowledgement of Proper Sources

Maintaining academic integrity was always a vital part of our project. Therefore we addressed all the resources we used throughout the project by means of intext citation and referencing.

9.3.4 Approval from Respective Bodies

Taking approval from BRTA is a must, however since we are BRACU students and designing this project system while being in BRACU we must take permission from Institutional Review Board (IRB).

9.4 Conclusion

In conclusion, to get approval from the authorities to implement our project, and to gain the trust of all our stakeholders, especially the consumer, we must design and implement our system to properly solve the engineering problem. To do this, we must follow the aforementioned aspects like Consent, Privacy and Confidentiality Protection, Acknowledgement of Proper Sources, and Approval from respective bodies to maintain absolute transparency and the standards of the project work done.

Chapter 10

Conclusion and Future Work

10.1 Project summary

In a nutshell, the main motivation behind choosing the topic, 'Vision Driven Lane Changing System of Self Driving Car' is to solve the problem of impaired and distracted driving, improving the safety on roads which not only includes the passengers and driver in the car but also includes pedestrians and other objects nearby. Our motivation was strengthened to solve the problem of road rule regulation breakage. Building a successful lane-changing autonomous vehicle will ease the maintenance of road rules carefully as it will be trained with past datasets which will include Training, Validation, as well as Testing Datasets.

On the contrary, our system uses image processing which specifically focuses on the creation of region of interest, perspective transformation, as well as threshold operations, canny edge detections, histogram and various other calculations for lane detection, Lane End detection and lane maintenance. Object detection models are trained and used to detect vehicles, stop signs and traffic signals using the Haar cascade Deep learning method, and passed through our decision model to make decisions and maneuver our vehicle accordingly. Furthermore, the output of our design analysis is deployed on a prototype level build with the help of Raspberry Pi 3B+ having integrated connection with Arduino and motor driver system.

Moreover, several parameters were being tested prior to hardware preparation on CARLA Simulator that gave us real world experience [27]. We used both a rule-based approach and a machine learning approach to design our self-driving vehicle and its lane-changing mechanism. We found out that our machine-learning approach gave better results and had more advantages. So, we chose that as our optimal design and implemented it in our hardware design.

10.2 Future work

This project can be later scaled up by using transfer learning to incorporate the entire model on a more delicate microprocessor so that it can be used in real industry-level car systems. The project still has a future scope if it can be developed to manufacture a device in one embedded system that has all the camera sensors, as well as more sensors as its auxiliary parameters so that real-time data collection and decision-making can be done easily and faster with more accuracy. Thus, a plug-and-play system as the end result would help to deliver the decisions from the embedded system to the car maneuvering system for ease of transportation.

Chapter 11

Identification of Complex Engineering Problems and Activities.

11.1 Attributes of Complex Engineering Problems (EP)

Table 14. Attributes of Complex Engineering Problems (EP)

	Attributes	Put tick (√) as appropriate	Justification
A1	Range of resource	√	Required knowledge from various people, equipment support from both EEE and CSE department, and need financial support to make.
A2	Level of interaction		
A3	Innovation	√	Autonomous car with automatic lane changing.
A4	Consequences for society and the environment	√	Will positively impact mainly people who are unable to drive
A5	Familiarity	√	Unique design solution revolving around lane changing.

11.2 Attributes of Complex Engineering Activities (EA)

Table 15. Attributes of Complex Engineering Activities (EA)

	Attributes	Put tick (√) as appropriate	Justification
P1	Depth of knowledge required	√	Use of machine-learning, image processing, sensor interface, simulation tools and computer language.
P2	Range of conflicting requirements		
P3	Depth of analysis required	√	Analysis of two design solutions using various simulation tools.
P4	Familiarity of issues	√	Unique design solution revolving around lane changing.
P5	Extent of applicable codes	√	Solutions require the modification and creation of new robust codes and standards due to uniqueness of solution.
P6	Extent of stakeholder involvement and needs		
P7	Interdependence	√	Contains various sub-systems that work together to give the desired outcome.

References

[1] US Department of Transportation, National Highway Traffic Safety Administration, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey", Traffic Safety Facts, No. DOT HS 812 115, Feb 2015

[2] J-P Skeete, "The obscure link between motorsport and energy efficient, low-carbon innovation: Evidence from the UK and European Union", .Journal of Cleaner Production, Volume 214, pp. 674-684, 2019

[3] A Mihalea ; R Samoilescu ; A Cristian Nica ; M Trăscău ; A Sorici ; A Magda Florea, "End-to-end models for self-driving cars on UPB campus roads", IEEE 15th International Conference on Intelligent Computer Communication and Processing (ICCP), Cluj-Napoca, Romania, Romania, 5-7 Sept. 2019

[4] J.P. Giacalone, L. Bourgeois, and A. Ancora, "Challenges in aggregation of heterogeneous sensors for Autonomous Driving Systems", IEEE Sensors Applications Symposium (SAS), Sophia Antipolis, France, 1-13 March 2019

[5] Tian et al., "Autonomous Driving System Design for Formula Student Driverless Racecar", IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26-30 June 2018

[6] L.Andresen et al., "Fast and Accurate Mapping for Autonomous Racing", arXiv:2003.05266 [cs.RO], March 2020 A Mihalea ; R Samoilescu ; A Cristian

[7] J. Kabzan et al "AMZ Driverless: The Full Autonomous Racing System", arXiv.org, arXiv:1905.05150, May 2019

[8] "A Real-Time Dynamic Trajectory Planning for Autonomous Driving Vehicles," *IEEE Conference Publication | IEEE Xplore*, Sep. 01, 2019. https://ieeexplore.ieee.org/document/8951735

[9] Claussmann, L., Revilloud, M., Gruyer, D. and Glaser, S., "A review of motion planning for highway autonomous driving", IEEE Transactions on Intelligent Transportation Systems, 21(5), pp.1826-1848, May 2019.

[10] C. Vallon, Ziya Ercan, A. Carvalho, and F. Borrelli, "A machine learning approach forpersonalized autonomous lane change initiation and control," IEEE Intelligent Vehicles Symposium, Jun. 2017, doi: https://doi.org/10.1109/ivs.2017.7995936.

[11] What is ADAS (Advanced Driver Assistance Systems)? – Overview of ADAS Applications | Synopsys, www.synopsys.com. https://www.synopsys.com/automotive/what-is-adas.html#:~:text=Definition

[12] H. A. Research, "What Is Adaptive Cruise Control?," Car and Driver, Jun. 10, 2020. https://www.caranddriver.com/research/a32813983/adaptive-cruise-control/

[13] Connected Vehicles - IEEE Connected Vehicles, site.ieee.org. https://site.ieee.org/connected-vehicles/ieee-connected-vechicles/connected-vehicles/

[14] "SAE Levels of Driving AutomationTM Refined for Clarity and International Audience," SAE Levels of Driving AutomationTM Refined for Clarity and International Audience. https://www.sae.org/site/blog/sae-j3016-update

[15] "Raspberry Pi Documentation - Camera." https://www.raspberrypi.com/documentation/accessories/camera.html
[16] R. P. Ltd, "Buy a Raspberry Pi 3 Model B+ – Raspberry Pi," *Raspberry Pi*. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/
[17] "SanDisk Ultra 16GB Micro SDHC UHS-I Card With Adapter (SDSQUAR-016G-GN6MA)," *Star Tech Ltd*. https://www.startech.com.bd/sandisk-ultra-16gb-micro-sdhc-uhs-i-card
[18] "4 Wheel 2 Layer Robot Smart Car Chassis Kits with Speed Encoder for Arduino DIY (Yellow) - Robotonbd," *Robotonbd*, Aug. 18, 2022. https://www.robotonbd.com/4-wheel-2-layer-robot-smart-car-chassis-kits-with-speed-encoder-for-arduino-diy-yellow/
[19] "4 Wheel 2 Layer Robot Smart Car Chassis Kits with Speed Encoder for Arduino DIY (Yellow) - Robotonbd," *Robotonbd*, Aug. 18, 2022. https://www.robotonbd.com/4-wheel-2-layer-robot-smart-car-chassis-kits-with-speed-encoder-for-arduino-diy-yellow/
[20] "L298N Motor Driver Module," *Components101*. https://components101.com/modules/l293n-motor-driver-module
[21] "Meko Power Bank 10000mAh." https://gadgetandgear.com/product/meko-power-bank-10000mah
[22] "3.7V 18650 Rechargeable Battery Solderable," *RoboticsBD*. https://store.roboticsbd.com/battery-charger/1687-37v-18650-rechargeable-battery-solderable-robotics-bangladesh.html
[23] "Universal Dual Battery Charger for Li-ion Battery," *RoboticsBD*. https://store.roboticsbd.com/battery-charger/1371-universal-dual-battery-charger-for-li-ion-battery-robotics-bangladesh.html
[24] "J3016_202104: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International." https://www.sae.org/standards/content/j3016_202104/
[25] "IEEE Standards Association," *IEEE Standards Association*. https://standards.ieee.org/ieee/2846/10831/
[26] "ISO/PAS 21448:2019," *ISO*. https://www.iso.org/standard/70939.html
[27] "IEEE Standards Association," *IEEE Standards Association*. https://standards.ieee.org/ieee/1451.99/10355/
[28] C. Team, "CARLA," *CARLA Simulator*. https://carla.org/
[29] "IEEE Standards Association," *IEEE Standards Association*. https://standards.ieee.org/ieee/2700/6770/
[30] "ISO/IEC TR 24029-1:2021," *ISO*. https://www.iso.org/standard/77609.html
[31] "ISO 26262-1:2011," *ISO*. https://www.iso.org/standard/43464.html
[32] S. Shin, S. K. Tirukkovalluri, J. Tuck, and Y. Solihin, "Proteus: A Flexible and Fast Software Supported Hardware Logging Approach for NVM," IEEE Xplore, Oct. 01, 2017. https://ieeexplore.ieee.org/document/8686655/figures (accessed Sep. 03, 2023).
[33] "Software," *Arduino*. https://www.arduino.cc/en/software
[34] "Cascade Trainer GUI - Amin," *Amin*, May 30, 2020. https://amin-ahmadi.com/cascade-trainer-gui/
[35] "Code::Blocks," *Code::Blocks*. https://www.codeblocks.org/
[36] "Standard C++." https://isocpp.org/

[37] "RealVNC® - Remote access software for desktop and mobile | RealVNC," *RealVNC®*, Nov. 30, 2023. https://www.realvnc.com/en/
[38] "PuTTY: a free SSH and Telnet client." https://www.chiark.greenend.org.uk/~sgtatham/putty/
[39] G. D. Team, "Home | Geany." https://www.geany.org/
[40] "OpenCV - Open Computer Vision Library," *OpenCV*, Dec. 28, 2023. https://opencv.org/

[41] M. Abdel-Basset, A. Gamal, N. Moustafa, A. Abdel-Monem, and N. El-Saber, "A Security-by-Design Decision-Making Model for Risk Management in Autonomous Vehicles," IEEE Access, vol. 9, pp. 107657–107679, 2021, doi: 10.1109/access.2021.3098675. [Online]. Available: http://dx.doi.org/10.1109/access.2021.3098675

[42] "Autonomous vehicles: theoretical and practical challenges," Autonomous vehicles: theoretical and practical challenges - ScienceDirect, Oct. 30, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352146518302606

[43] M. Abdel-Basset, A. Gamal, N. Moustafa, A. Abdel-Monem, and N. El-Saber, "A Security-by-Design Decision-Making Model for Risk Management in Autonomous Vehicles," IEEE Access, vol. 9, pp. 107657–107679, 2021, doi: 10.1109/access.2021.3098675. [Online]. Available: http://dx.doi.org/10.1109/access.2021.3098675

[44] R. B. Issa, Md. Saferi Rahman, M. Das, M. Barua, and Md. G. Rabiul Alam, "Reinforcement Learning based Autonomous Vehicle for Exploration and Exploitation of Undiscovered Track," 2020 International Conference on Information Networking (ICOIN), Jan. 2020, doi: 10.1109/icoin48656.2020.9016539. [Online]. Available: http://dx.doi.org/10.1109/icoin48656.2020.9016539

Appendix

Software Codes for Optimum Design (Design Approach-2: Machine Learning):

Python Code in CARLA for Design approach-2 Machine learning:

```
import glob
import os
import sys
try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass
import carla

import random
import time
import numpy as np
import cv2

IM_WIDTH = 640
IM_HEIGHT = 480

from roboflow import Roboflow
rf = Roboflow(api_key="WzH6em9wsEIJVOEl9UtD")
project = rf.workspace().project("design-and-implementation-of-self-driving-cars")
model = project.version(5).model
```

```
image_counter = 0

output_directory1 = 'C:\\Users\\20121006\\Desktop\\Camera images\\'


def process_img1(image):
    global image_counter
    i = np.array(image.raw_data)
    i2 = i.reshape((IM_HEIGHT, IM_WIDTH, 4))
    i3 = i2[:, :, :3]


    #Incrementing counter and generate filename
    image_counter += 1
    filename = f'image_{image_counter}.PNG'

    # # Saving image
    cv2.imwrite(output_directory1 + filename, i3)
    model.predict(f"C:\\Users\\20121006\\Desktop\\Camera images\\{filename}", confidence=40, overlap=30).save(f"C:\\Users\\20121006\\Documents\\Trained Images\\{filename}")

    image = cv2.imread(f"C:\\Users\\20121006\\Documents\\Trained Images\\{filename}")
    # cv2.namedWindow('image_window', cv2.WINDOW_NORMAL)
    cv2.imshow('image_window', image)
    cv2.waitKey(1)

    # time.sleep(2)
    return i3/255.0
```

```
actor_list = []
try:
    client = carla.Client('localhost', 2000)
    world = client.get_world()

    blueprint_library = world.get_blueprint_library()

    bp1 = blueprint_library.filter('model3')[0]
    bp2 = blueprint_library.filter('mini')[0]
    print(bp1)
    print(bp2)

    spawn_point = world.get_map().get_spawn_points()

    vehicle1 = world.spawn_actor(bp1, spawn_point[230])
    vehicle1.set_autopilot(True)

    actor_list.append(vehicle1)

    # getting the blueprint for this sensor
    blueprint1 = blueprint_library.find('sensor.camera.rgb')
    # changing the dimensions of the image
    blueprint1.set_attribute('image_size_x', f'{IM_WIDTH}')
    blueprint1.set_attribute('image_size_y', f'{IM_HEIGHT}')
    blueprint1.set_attribute('fov', '110')

    # Adjusting sensor relative to vehicle
    spawn1 = carla.Transform(carla.Location(x=2.5, z=0.7))
```

```
    # spawning the sensor and attach it to the vehicle.
    sensor1 = world.spawn_actor(blueprint1, spawn1, attach_to=vehicle1)

    # adding sensor to list of actors
    actor_list.append(sensor1)

    sensor1.listen(lambda data: process_img1(data))

    time.sleep(1000)

finally:
    print('destroying actors')
    for actor in actor_list:
        actor.destroy()
    print('done.')
```

Proteus Simulation Code for Wheel Drive with Ultrasonic Sensor:

```
int M1I1=11;
int M1I2=10;
int M2I1=9;
int M2I2=8;

int M3I1=1;
int M3I2=2;
int M4I1=3;
int M4I2=4;
```

```
int LInd=12;
int RInd=13;
int FInd=6;

int trig_f = A0;
int echo_f = A1;
long duration_f;
float distance_cm_f;

int trig_r = A2;
int echo_r = A3;
long duration_r;
float distance_cm_r;

int trig_l = A4;
int echo_l = A5;
long duration_l;
float distance_cm_l;

void setup() {
  // put your setup code here, to run once:
pinMode(M1I1,OUTPUT);
pinMode(M1I2,OUTPUT);
pinMode(M2I1,OUTPUT);
pinMode(M2I2,OUTPUT);

pinMode(M3I1,OUTPUT);
pinMode(M3I2,OUTPUT);
pinMode(M4I1,OUTPUT);
```

```
pinMode(M4I2,OUTPUT);

pinMode(LInd,OUTPUT);
pinMode(RInd,OUTPUT);
pinMode(FInd,OUTPUT);

pinMode(trig_f,OUTPUT);
pinMode(echo_f,INPUT);

pinMode(trig_r,OUTPUT);
pinMode(echo_r,INPUT);

pinMode(trig_f,OUTPUT);
pinMode(echo_f,INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
digitalWrite(trig_f,LOW);
delayMicroseconds(2);
digitalWrite(trig_f,HIGH);
delayMicroseconds(10);
digitalWrite(trig_f,LOW);
duration_f = pulseIn(echo_f,HIGH);
distance_cm_f = duration_f*.034/2;

digitalWrite(trig_r,LOW);
delayMicroseconds(2);
digitalWrite(trig_r,HIGH);
```

```
delayMicroseconds(10);
digitalWrite(trig_r,LOW);
duration_r = pulseIn(echo_r,HIGH);
distance_cm_r = duration_r*.034/2;

digitalWrite(trig_l,LOW);
delayMicroseconds(2);
digitalWrite(trig_l,HIGH);
delayMicroseconds(10);
digitalWrite(trig_l,LOW);
duration_l = pulseIn(echo_l,HIGH);
distance_cm_l = duration_l*.034/2;

//FORWARD
if(distance_cm_f>1000){
  digitalWrite(M1I1,HIGH);
  digitalWrite(M1I2,LOW);
  digitalWrite(M2I1,LOW);
  digitalWrite(M2I2,HIGH);

  digitalWrite(M3I1,LOW);
  digitalWrite(M3I2,HIGH);
  digitalWrite(M4I1,HIGH);
  digitalWrite(M4I2,LOW);

  digitalWrite(FInd,HIGH);
  digitalWrite(LInd,LOW);
  digitalWrite(RInd,LOW);
  }
```

```
//LEFT
else if(distance_cm_l>500){
  digitalWrite(M1I1,LOW);
  digitalWrite(M1I2,LOW);
  digitalWrite(M2I1,LOW);
  digitalWrite(M2I2,HIGH);

  digitalWrite(M3I1,LOW);
  digitalWrite(M3I2,HIGH);
  digitalWrite(M4I1,HIGH);
  digitalWrite(M4I2,LOW);

  digitalWrite(LInd,HIGH);
  digitalWrite(FInd,LOW);
  digitalWrite(RInd,LOW);
  }
//RIGHT
else if(distance_cm_r>500){
  digitalWrite(M1I1,HIGH);
  digitalWrite(M1I2,LOW);
  digitalWrite(M2I1,LOW);
  digitalWrite(M2I2,LOW);

  digitalWrite(M3I1,LOW);
  digitalWrite(M3I2,HIGH);
  digitalWrite(M4I1,HIGH);
  digitalWrite(M4I2,LOW);

  digitalWrite(RInd,HIGH);
```

```
  digitalWrite(LInd,LOW);
  digitalWrite(FInd,LOW);
  }
else{
  digitalWrite(M1I1,LOW);
  digitalWrite(M1I2,LOW);
  digitalWrite(M2I1,LOW);
  digitalWrite(M2I2,LOW);

  digitalWrite(M3I1,LOW);
  digitalWrite(M3I2,LOW);
  digitalWrite(M4I1,LOW);
  digitalWrite(M4I2,LOW);

  digitalWrite(RInd,LOW);
  digitalWrite(LInd,LOW);
  digitalWrite(FInd,LOW);
}
}
```

Final Hardware Codes for Optimum Design (Design Approach-2: Machine Learning)

Final C++ Code in Raspberry Pi:

```
#include <opencv2/opencv.hpp>
#include <raspicam_cv.h>
#include <iostream>
#include <chrono>
#include <ctime>
#include <wiringPi.h>
```

```
using namespace std;
using namespace cv;
using namespace raspicam;

// Image Processing variables
Mat frame, Matrix, framePers, frameGray, frameThresh, frameEdge, frameFinal, frameFinalDuplicate, frameFinalDuplicate1;
Mat ROILane, ROILaneEnd;
int LeftLanePos, RightLanePos, frameCenter, laneCenter, Result, laneEnd;

RaspiCam_Cv Camera;
stringstream ss;

vector<int> histrogramLane;
vector<int> histrogramLaneEnd;

Point2f Source[] = {Point2f(40,135),Point2f(360,135),Point2f(0,185), Point2f(400,185)};
Point2f Destination[] = {Point2f(100,0),Point2f(280,0),Point2f(100,240), Point2f(280,240)};

//Machine Learning variables
CascadeClassifier Stop_Cascade, Object_Cascade, Traffic_Cascade;
Mat frame_Stop, RoI_Stop, gray_Stop, frame_Object, RoI_Object, gray_Object, frame_Traffic, RoI_Traffic, gray_Traffic;
vector<Rect> Stop, Object, Traffic;
int dist_Stop, dist_Object, dist_Traffic;

 void Setup ( int argc,char **argv, RaspiCam_Cv &Camera )
  {
    Camera.set ( CAP_PROP_FRAME_WIDTH,  ( "-w",argc,argv,400 ) );
```

```
   Camera.set ( CAP_PROP_FRAME_HEIGHT,  ( "-h",argc,argv,240 ) );
   Camera.set ( CAP_PROP_BRIGHTNESS, ( "-br",argc,argv,50 ) );
   Camera.set ( CAP_PROP_CONTRAST ,( "-co",argc,argv,50 ) );
   Camera.set ( CAP_PROP_SATURATION,  ( "-sa",argc,argv,50 ) );
   Camera.set ( CAP_PROP_GAIN,  ( "-g",argc,argv ,50 ) );
   Camera.set ( CAP_PROP_FPS,  ( "-fps",argc,argv,0));
}

void Capture()
{
   Camera.grab();
   Camera.retrieve( frame);
   cvtColor(frame, frame_Stop, COLOR_BGR2RGB);
   cvtColor(frame, frame_Object, COLOR_BGR2RGB);
   cvtColor(frame, frame_Traffic, COLOR_BGR2RGB);
   cvtColor(frame, frame, COLOR_BGR2RGB);
}

void Perspective()
{
        line(frame,Source[0], Source[1], Scalar(0,0,255), 2);
        line(frame,Source[1], Source[3], Scalar(0,0,255), 2);
        line(frame,Source[3], Source[2], Scalar(0,0,255), 2);
        line(frame,Source[2], Source[0], Scalar(0,0,255), 2);

        Matrix = getPerspectiveTransform(Source, Destination);
        warpPerspective(frame, framePers, Matrix, Size(400,240));
}
```

```
void Threshold()
{
	cvtColor(framePers, frameGray, COLOR_RGB2GRAY);
	inRange(frameGray, 230, 255, frameThresh);
	Canny(frameGray,frameEdge, 900, 900, 3, false);
	add(frameThresh, frameEdge, frameFinal);
	cvtColor(frameFinal, frameFinal, COLOR_GRAY2RGB);
	cvtColor(frameFinal, frameFinalDuplicate, COLOR_RGB2BGR);	//used in histrogram function only
	cvtColor(frameFinal, frameFinalDuplicate1, COLOR_RGB2BGR);	//used in histrogram function only
}

void Histrogram()
{
   histrogramLane.resize(400);
   histrogramLane.clear();

   for(int i=0; i<400; i++)	//frame.size().width = 400
   {
	ROILane = frameFinalDuplicate(Rect(i,140,1,100));
	divide(255, ROILane, ROILane);
	histrogramLane.push_back((int)(sum(ROILane)[0]));
   }

   histrogramLaneEnd.resize(400);
   histrogramLaneEnd.clear();
	for (int i = 0; i < 400; i++)
	{
		ROILaneEnd = frameFinalDuplicate1(Rect(i, 0, 1, 240));
```

```
                divide(255, ROILaneEnd, ROILaneEnd);
                histrogramLaneEnd.push_back((int)(sum(ROILaneEnd)[0]));
        }

          laneEnd = sum(histrogramLaneEnd)[0];
          cout<<"Lane END = "<<laneEnd<<endl;
}

void LaneFinder()
{
   vector<int>:: iterator LeftPtr;
   LeftPtr = max_element(histrogramLane.begin(), histrogramLane.begin() + 150);
   LeftLanePos = distance(histrogramLane.begin(), LeftPtr);

   vector<int>:: iterator RightPtr;
   RightPtr = max_element(histrogramLane.begin() +250, histrogramLane.end());
   RightLanePos = distance(histrogramLane.begin(), RightPtr);

   line(frameFinal, Point2f(LeftLanePos, 0), Point2f(LeftLanePos, 240), Scalar(0, 255,0), 2);
   line(frameFinal, Point2f(RightLanePos, 0), Point2f(RightLanePos, 240), Scalar(0,255,0),
2);
}

void LaneCenter()
{
   laneCenter = (RightLanePos-LeftLanePos)/2 +LeftLanePos;
   frameCenter = 188;

   line(frameFinal, Point2f(laneCenter,0), Point2f(laneCenter,240), Scalar(0,255,0), 3);
```

```
   line(frameFinal, Point2f(frameCenter,0), Point2f(frameCenter,240), Scalar(255,0,0), 3);


   Result = laneCenter-frameCenter;
}


void Stop_detection()
{
   if(!Stop_Cascade.load("//home//pi//Desktop//MACHINE
LEARNING//Stop_cascade.xml"))
   {
        printf("Unable to open stop cascade file");
   }


   RoI_Stop = frame_Stop(Rect(200,0,200,140));
   cvtColor(RoI_Stop, gray_Stop, COLOR_RGB2GRAY);
   equalizeHist(gray_Stop, gray_Stop);
   Stop_Cascade.detectMultiScale(gray_Stop, Stop);


   for(int i=0; i<Stop.size(); i++)
   {
        Point P1(Stop[i].x, Stop[i].y);
        Point P2(Stop[i].x + Stop[i].width, Stop[i].y + Stop[i].height);


        rectangle(RoI_Stop, P1, P2, Scalar(0, 0, 255), 2);
        putText(RoI_Stop, "Stop Sign", P1, FONT_HERSHEY_PLAIN, 1,  Scalar(0, 0, 255,
255), 2);
        dist_Stop = (-1.07)*(P2.x-P1.x) + 102.597;


     ss.str(" ");
     ss.clear();
```

```
      ss<<"D = "<<dist_Stop<<"cm";
      putText(RoI_Stop, ss.str(), Point2f(1,130), 0,1, Scalar(0,0,255), 2);
   }
}

void Traffic_detection()
{
   if(!Traffic_Cascade.load("//home//pi//Desktop//MACHINE
LEARNING//Trafficc_cascade.xml"))
   {
         printf("Unable to open traffic cascade file");
   }

   RoI_Traffic = frame_Traffic(Rect(200,0,200,140));
   cvtColor(RoI_Traffic, gray_Traffic, COLOR_RGB2GRAY);
   equalizeHist(gray_Traffic, gray_Traffic);
   Traffic_Cascade.detectMultiScale(gray_Traffic, Traffic);

   for(int i=0; i<Traffic.size(); i++)
   {
         Point P1(Traffic[i].x, Traffic[i].y);
         Point P2(Traffic[i].x + Traffic[i].width, Traffic[i].y + Traffic[i].height);

         rectangle(RoI_Traffic, P1, P2, Scalar(0, 0, 255), 2);
         putText(RoI_Traffic, "Traffic Light", P1, FONT_HERSHEY_PLAIN, 1,  Scalar(0, 0,
255, 255), 2);
         dist_Traffic = (-1.07)*(P2.x-P1.x) + 102.597;

      ss.str(" ");
      ss.clear();
```

```
      ss<<"D = "<<P2.x-P1.x<<"cm";
      putText(RoI_Traffic, ss.str(), Point2f(1,130), 0,1, Scalar(0,0,255), 2);
   }
}

void Object_detection()
{
   if(!Object_Cascade.load("//home//pi//Desktop//MACHINE
LEARNING//Object_cascade.xml"))
   {
         printf("Unable to open Object cascade file");
   }

   RoI_Object = frame_Object(Rect(100,50,200,190));
   cvtColor(RoI_Object, gray_Object, COLOR_RGB2GRAY);
   equalizeHist(gray_Object, gray_Object);
   Object_Cascade.detectMultiScale(gray_Object, Object);

   for(int i=0; i<Object.size(); i++)
   {
         Point P1(Object[i].x, Object[i].y);
         Point P2(Object[i].x + Object[i].width, Object[i].y + Object[i].height);

         rectangle(RoI_Object, P1, P2, Scalar(0, 0, 255), 2);
         putText(RoI_Object, "Object", P1, FONT_HERSHEY_PLAIN, 1,  Scalar(0, 0, 255,
255), 2);
         dist_Object = (-0.48)*(P2.x-P1.x) + 56.6;

      ss.str(" ");
      ss.clear();
```

```
        ss<<"D = "<<dist_Object<<"cm";
        putText(RoI_Object, ss.str(), Point2f(1,130), 0,1, Scalar(0,0,255), 2);
    }
}

int main(int argc,char **argv)
{
    wiringPiSetup();
    pinMode(21, OUTPUT);
    pinMode(22, OUTPUT);
    pinMode(23, OUTPUT);
    pinMode(24, OUTPUT);

    Setup(argc, argv, Camera);
    cout<<"Connecting to camera"<<endl;
    if (!Camera.open())
    {
        cout<<"Failed to Connect"<<endl;
    }
        cout<<"Camera Id = "<<Camera.getId()<<endl;

    while(1)
    {
    auto start = std::chrono::system_clock::now();

    Capture();
    Perspective();
    Threshold();
    Histrogram();
```

```
LaneFinder();
LaneCenter();
Stop_detection();
Object_detection();
Traffic_detection();

if (dist_Stop > 5 && dist_Stop < 20)
{
      digitalWrite(21, 0);
      digitalWrite(22, 0);    //decimal = 8
      digitalWrite(23, 0);
      digitalWrite(24, 1);
      cout<<"Stop Sign"<<endl;
      dist_Stop = 0;

      goto Stop_Sign;
}

   if (dist_Object > 5 && dist_Object < 30)
{
      digitalWrite(21, 1);
      digitalWrite(22, 0);    //decimal = 9
      digitalWrite(23, 0);
      digitalWrite(24, 1);
      cout<<"Object"<<endl;
      dist_Object = 0;

      goto Object;
}
```

```
   if (dist_Traffic > 5 && dist_Traffic < 20)
{
      digitalWrite(21, 0);
      digitalWrite(22, 1);    //decimal = 10
      digitalWrite(23, 0);
      digitalWrite(24, 1);
      cout<<"Traffic Light"<<endl;
      dist_Traffic = 0;

      goto Traffic;
}

if (laneEnd > 4500)
{
      digitalWrite(21, 1);
      digitalWrite(22, 1);    //decimal = 7
      digitalWrite(23, 1);
      digitalWrite(24, 0);
      cout<<"Lane End"<<endl;
}

if (Result == 0)
{
      digitalWrite(21, 0);
      digitalWrite(22, 0);    //decimal = 0
      digitalWrite(23, 0);
      digitalWrite(24, 0);
      cout<<"Forward"<<endl;
```

```
}

else if (Result >0 && Result <10)
{
      digitalWrite(21, 1);
      digitalWrite(22, 0);    //decimal = 1
      digitalWrite(23, 0);
      digitalWrite(24, 0);
      cout<<"Right1"<<endl;
}

   else if (Result >=10 && Result <20)
{
      digitalWrite(21, 0);
      digitalWrite(22, 1);    //decimal = 2
      digitalWrite(23, 0);
      digitalWrite(24, 0);
      cout<<"Right2"<<endl;
}

   else if (Result >20)
{
      digitalWrite(21, 1);
      digitalWrite(22, 1);    //decimal = 3
      digitalWrite(23, 0);
      digitalWrite(24, 0);
      cout<<"Right3"<<endl;
}
```

```
   else if (Result <0 && Result >-10)
{
      digitalWrite(21, 0);
      digitalWrite(22, 0);    //decimal = 4
      digitalWrite(23, 1);
      digitalWrite(24, 0);
      cout<<"Left1"<<endl;
}

   else if (Result <=-10 && Result >-20)
{
      digitalWrite(21, 1);
      digitalWrite(22, 0);    //decimal = 5
      digitalWrite(23, 1);
      digitalWrite(24, 0);
      cout<<"Left2"<<endl;
}

   else if (Result <-20)
{
      digitalWrite(21, 0);
      digitalWrite(22, 1);    //decimal = 6
      digitalWrite(23, 1);
      digitalWrite(24, 0);
      cout<<"Left3"<<endl;
}

Stop_Sign:
Object:
```

Traffic:

```
if (laneEnd > 4500)
 {
   ss.str(" ");
   ss.clear();
   ss<<" Lane End";
   putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(255,0,0), 2);
  }

 else if (Result == 0)
 {
   ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (Move Forward)";
   putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(0,0,255), 2);
  }

 else if (Result > 0)
 {
   ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (Move Right)";
   putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(0,0,255), 2);
  }

  else if (Result < 0)
 {
   ss.str(" ");
```

```
  ss.clear();
  ss<<"Result = "<<Result<<" (Move Left)";
  putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(0,0,255), 2);
 }

namedWindow("orignal", WINDOW_KEEPRATIO);
moveWindow("orignal", 0, 100);
resizeWindow("orignal", 640, 480);
imshow("orignal", frame);

namedWindow("Perspective", WINDOW_KEEPRATIO);
moveWindow("Perspective", 640, 100);
resizeWindow("Perspective", 640, 480);
imshow("Perspective", framePers);

namedWindow("Final", WINDOW_KEEPRATIO);
moveWindow("Final", 1280, 100);
resizeWindow("Final", 640, 480);
imshow("Final", frameFinal);

namedWindow("Stop Sign", WINDOW_KEEPRATIO);
moveWindow("Stop Sign", 1280, 580);
resizeWindow("Stop Sign", 640, 480);
imshow("Stop Sign", RoI_Stop);

namedWindow("Object", WINDOW_KEEPRATIO);
moveWindow("Object", 640, 580);
resizeWindow("Object", 640, 480);
imshow("Object", RoI_Object);
```

```
    namedWindow("Traffic", WINDOW_KEEPRATIO);
    moveWindow("Traffic", 0, 580);
    resizeWindow("Traffic", 640, 480);
    imshow("Traffic", RoI_Traffic);

    waitKey(1);
    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end-start;

    float t = elapsed_seconds.count();
    int FPS = 1/t;
    //cout<<"FPS = "<<FPS<<endl;
    }
    return 0;
}
```

C++ Code in Raspberry Pi to capture images from Raspberry Pi Camera:

```
#include <opencv2/opencv.hpp>
#include <raspicam_cv.h>
#include <iostream>
#include <chrono>
#include <ctime>

using namespace std;
using namespace cv;
using namespace raspicam;
```

```
Mat frame;



 void Setup ( int argc,char **argv, RaspiCam_Cv &Camera )
  {
    Camera.set ( CAP_PROP_FRAME_WIDTH,  ( "-w",argc,argv,400 ) );
    Camera.set ( CAP_PROP_FRAME_HEIGHT,  ( "-h",argc,argv,240 ) );
    Camera.set ( CAP_PROP_BRIGHTNESS, ( "-br",argc,argv,50 ) );
    Camera.set ( CAP_PROP_CONTRAST ,( "-co",argc,argv,50 ) );
    Camera.set ( CAP_PROP_SATURATION,  ( "-sa",argc,argv,50 ) );
    Camera.set ( CAP_PROP_GAIN,  ( "-g",argc,argv ,50 ) );
    Camera.set ( CAP_PROP_FPS,  ( "-fps",argc,argv,100));
}

int main(int argc,char **argv)
{
        RaspiCam_Cv Camera;
            Setup(argc, argv, Camera);
            cout<<"Connecting to camera"<<endl;
            if (!Camera.open())
            {
            cout<<"Failed to Connect"<<endl;
     }
     cout<<"Camera Id = "<<Camera.getId()<<endl;

    while(1)
    {
    auto start = std::chrono::system_clock::now();
```

```
    Camera.grab();
    Camera.retrieve( frame);
    auto end = std::chrono::system_clock::now();

    std::chrono::duration<double> elapsed_seconds = end-start;

    float t = elapsed_seconds.count();
    int FPS = 1/t;
    cout<<"FPS = "<<FPS<<endl;
    imshow("orignal", frame);

    waitKey(1);
    }
    return 0;
}
```

Code in Arduino IDE for Arduino:

```
int i =0;
unsigned long int j =0;

const int EnableL = 5;
const int HighL = 6;       // LEFT SIDE MOTOR
const int LowL =7;

const int EnableR = 10;
const int HighR = 8;       //RIGHT SIDE MOTOR
const int LowR =9;
```

```
const int D0 = 0;       //Raspberry pin 21    LSB
const int D1 = 1;       //Raspberry pin 22
const int D2 = 2;       //Raspberry pin 23
const int D3 = 3;       //Raspberry pin 24    MSB

int a,b,c,d,data;



void setup() {

pinMode(EnableL, OUTPUT);
pinMode(HighL, OUTPUT);
pinMode(LowL, OUTPUT);

pinMode(EnableR, OUTPUT);
pinMode(HighR, OUTPUT);
pinMode(LowR, OUTPUT);

pinMode(D0, INPUT_PULLUP);
pinMode(D1, INPUT_PULLUP);
pinMode(D2, INPUT_PULLUP);
pinMode(D3, INPUT_PULLUP);
}

void Data()
{
  a = digitalRead(D0);
  b = digitalRead(D1);
  c = digitalRead(D2);
```

```
   d = digitalRead(D3);

   data = 8*d+4*c+2*b+a;
}

void Forward()
{
  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL,255);

  digitalWrite(HighR, LOW);
  digitalWrite(LowR, HIGH);
  analogWrite(EnableR,255);
}

void Backward()
{
  digitalWrite(HighL, HIGH);
  digitalWrite(LowL, LOW);
  analogWrite(EnableL,255);

  digitalWrite(HighR, HIGH);
  digitalWrite(LowR, LOW);
  analogWrite(EnableR,255);
}

void Stop()
{
```

```
  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL,0);

  digitalWrite(HighR, LOW);
  digitalWrite(LowR, HIGH);
  analogWrite(EnableR,0);
}

void Left1()
{
  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL,160);

  digitalWrite(HighR, LOW);
  digitalWrite(LowR, HIGH);
  analogWrite(EnableR,255);
}

void Left2()
{
  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL,90);

  digitalWrite(HighR, LOW);
  digitalWrite(LowR, HIGH);
  analogWrite(EnableR,255);
```

```
}

void Left3()
{
  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL,50);

  digitalWrite(HighR, LOW);
  digitalWrite(LowR, HIGH);
  analogWrite(EnableR,255);
}

void Right1()
{
  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL,255);

  digitalWrite(HighR, LOW);
  digitalWrite(LowR, HIGH);
  analogWrite(EnableR,160);  //200
}

void Right2()
{
  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL,255);
```

```
  digitalWrite(HighR, LOW);
  digitalWrite(LowR, HIGH);
  analogWrite(EnableR,90);   //160
}

void Right3()
{
  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL,255);

  digitalWrite(HighR, LOW);
  digitalWrite(LowR, HIGH);
  analogWrite(EnableR,50);   //100
}

void UTurn()
{
  analogWrite(EnableL, 0);
  analogWrite(EnableR, 0);
  delay(400);

  analogWrite(EnableL, 250);
  analogWrite(EnableR, 250);    //forward
  delay(1000);

  analogWrite(EnableL, 0);
  analogWrite(EnableR, 0);
```

```
delay(400);

digitalWrite(HighL, HIGH);
digitalWrite(LowL, LOW);
digitalWrite(HighR, LOW);   //   left
digitalWrite(LowR, HIGH);
analogWrite(EnableL, 255);
analogWrite(EnableR, 255);
delay(700);

analogWrite(EnableL, 0);
analogWrite(EnableR, 0);
delay(400);

digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
digitalWrite(HighR, LOW);   // forward
digitalWrite(LowR, HIGH);
analogWrite(EnableL, 255);
analogWrite(EnableR, 255);
delay(900);

analogWrite(EnableL, 0);
analogWrite(EnableR, 0);
delay(400);

digitalWrite(HighL, HIGH);
digitalWrite(LowL, LOW);
digitalWrite(HighR, LOW);    //left
```

```
  digitalWrite(LowR, HIGH);
  analogWrite(EnableL, 255);
  analogWrite(EnableR, 255);
  delay(700);

  analogWrite(EnableL, 0);
  analogWrite(EnableR, 0);
  delay(1000);

  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  digitalWrite(HighR, LOW);
  digitalWrite(LowL, HIGH);
  analogWrite(EnableL, 150);
  analogWrite(EnableR, 150);
  delay(300);
}


void Object()
{

  analogWrite(EnableL, 0);
  analogWrite(EnableR, 0);          //stop
  delay(1000);

  digitalWrite(HighL, HIGH);
  digitalWrite(LowL, LOW);
  digitalWrite(HighR, LOW);
```

```
digitalWrite(LowR, HIGH);       //left
analogWrite(EnableL, 250);
analogWrite(EnableR, 250);
delay(500);

analogWrite(EnableL, 0);
analogWrite(EnableR, 0);          //stop
delay(200);

digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);          //forward
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableL, 255);
analogWrite(EnableR, 255);
delay(1000);

analogWrite(EnableL, 0);          //stop
analogWrite(EnableR, 0);
delay(200);

digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
digitalWrite(HighR, HIGH);        //right
digitalWrite(LowR, LOW);
analogWrite(EnableL, 255);
analogWrite(EnableR, 255);
delay(500);
```

```
  analogWrite(EnableL, 0);              //stop
  analogWrite(EnableR, 0);
  delay(1000);

  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  digitalWrite(HighR, LOW);       // forward
  digitalWrite(LowR, HIGH);
  analogWrite(EnableL, 150);
  analogWrite(EnableR, 150);
  delay(500);

   i  = i+1;
}

void Lane_Change()
{
  analogWrite(EnableL, 0);
  analogWrite(EnableR, 0);            //stop
  delay(1000);

  digitalWrite(HighL, LOW);
  digitalWrite(LowL, HIGH);
  digitalWrite(HighR, HIGH);
  digitalWrite(LowR, LOW);        //Right
  analogWrite(EnableL, 250);
  analogWrite(EnableR, 250);
  delay(500);
```

```
analogWrite(EnableL, 0);
analogWrite(EnableR, 0);          //stop
delay(200);

digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);          //forward
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableL, 255);
analogWrite(EnableR, 255);
delay(800);

analogWrite(EnableL, 0);          //stop
analogWrite(EnableR, 0);
delay(200);

digitalWrite(HighL, HIGH);
digitalWrite(LowL, LOW);
digitalWrite(HighR, LOW);         //LEFT
digitalWrite(LowR, HIGH);
analogWrite(EnableL, 255);
analogWrite(EnableR, 255);
delay(500);

analogWrite(EnableL, 0);            //stop
analogWrite(EnableR, 0);
delay(1000);

digitalWrite(HighL, LOW);
```

```
  digitalWrite(LowL, HIGH);
  digitalWrite(HighR, LOW);       // forward
  digitalWrite(LowR, HIGH);
  analogWrite(EnableL, 150);
  analogWrite(EnableR, 150);
  delay(500);

}

void loop()
{
    if (j > 25000)
    {
      Lane_Change();
      i = 0;
      j = 0;
    }

  Data();
  if(data==0)
   {
     Forward();
     if (i>0)
     {
      j = j+1;
     }
   }

  else if(data==1)
```

```
 {
  Right1();
      if (i>0)
  {
   j = j+1;
  }
 }

else if(data==2)
 {
  Right2();
      if (i>0)
  {
   j = j+1;
  }
 }

else if(data==3)
 {
  Right3();
      if (i>0)
  {
   j = j+1;
  }
 }

else if(data==4)
 {
  Left1();
```

```
        if (i>0)
   {
    j = j+1;
   }
 }

else if(data==5)
 {
   Left2();
        if (i>0)
   {
    j = j+1;
   }
 }

else if(data==6)
 {
   Left3();
        if (i>0)
   {
    j = j+1;
   }
 }

else if(data==7)
 {
   UTurn();
 }
```

```
 else if (data==8)
  {
    analogWrite(EnableL, 0);
    analogWrite(EnableR, 0);
    delay(4000);

    analogWrite(EnableL, 150);
    analogWrite(EnableR, 150);
    delay(1000);
  }

    else if(data==9)
  {
   Object();
  }
       else if(data==10)
  {
    analogWrite(EnableL, 0);
    analogWrite(EnableR, 0);
    delay(2000);
  }

   else if(data>10)
  {
   Stop();
  }

}
```