

Nemobile: The Self Driving Robot



Group Members

Muhammad Murtaza Khan	16I-0478
Tayyab Rehman	16I-0528

Project Supervisor

Dr. Ata-ul-Aziz Ikram

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Islamabad
2020

Developer's Submission

"This report is being submitted to the Department of Electrical Engineering of the National University of Computer and Emerging Sciences in partial fulfillment of the requirements for the degree of BS in Electrical Engineering.

Developer's Declaration

"We take full responsibility for the project work conducted during the Final Year Project (FYP) titled **"Nemobile: The Self Driving Robot"**. We solemnly declare that the project work presented in the FYP report is done solely by us with no significant help from any other person; however, small help wherever taken is duly acknowledged. We have also written the complete FYP report by ourselves. Moreover, we have not presented this FYP (or substantially similar project work) or any part of the thesis previously to any other degree awarding institution within Pakistan or abroad.

We understand that the management of the Department of Electrical Engineering of National University of Computer and Emerging Sciences has a zero-tolerance policy towards plagiarism. Therefore, we as an author of the above-mentioned FYP report solemnly declare that no portion of our report has been plagiarized and any material used in the report from other sources is properly referenced. Moreover, the report does not contain any literal citing of more than 70 words (total) even by giving a reference unless we have obtained the written permission of the publisher to do so. Furthermore, the work presented in the report is our own work and we have positively cited the related work of the other projects by clearly differentiating our work from their relevant work.

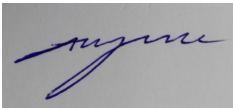
We further understand that if we are found guilty of any form of plagiarism in our FYP report even after our graduation, the University reserves the right to withdraw our BS degree. Moreover, the University will also have the right to publish our names on its website that keeps a record of the students who committed plagiarism in their FYP reports."



Muhammad Murtaza Khan
BS(EE) 2016-0478



Tayyab Rehman
BS(EE) 2016-0528



Certified by Supervisor



Verified by Plagiarism Cell Officer

Dated: 27 July 2020

Abstract

Our world is evolving into a new phase where machines are used to make our life easier and provide faster and safer services to humans without the mistakes caused by human errors. USA NPR Surveys report that over 90% of fatal car crashes are caused by mistakes made by humans. The only way to stop this is to make cars which are fully autonomous, unless we do this there will still be mistakes made by humans which will cause thousands of automobile deaths every year. That is what we wish to achieve by this project, doing research and starting to make a fully autonomous robot, which can be further made into a car in the future. The robot we made uses convolutional neural networks and data from ultrasonic sensors to perfectly detect objects in its way and detects the lane it is in and steers to stay within its lane. The project does its processing from an external server where it sends its data to and waits for the data to be processed. After processing the server sends instructions back to the robot and it executes them. In conclusion the project was successfully completed with all simulations providing 87% accuracy.

Acknowledgements

As a matter of first importance, we are thankful to Allah for the great wellbeing and prosperity that was important to finish this book.

We must thank our supervisor, Dr. Ata-ul-Aziz Ikram. Without his devoted help and guidance, this project would be incomplete. We want to express gratitude toward him especially for his help and comprehension over the previous year.

Traversing our Final Year Project needed something other than scholastic help, and we have many, numerous individuals to thank for tuning in through the span of this program. We can't start to offer our thanks and thankfulness for their fellowship and backing.

Above all, none of this would have been conceivable without our family. This paper remains as a demonstration of your genuine love, backing and consolation

Table of Contents

DEVELOPER'S SUBMISSION	II
DEVELOPER'S DECLARATION	VI
ABSTRACT	VII
ACKNOWLEDGEMENTS	VIII
TABLE OF CONTENTS	VIII
LIST OF FIGURES	VIII
LIST OF TABLES	VIII
CHAPTER 1 INTRODUCTION	1
1.1 PROBLEM STATEMENT	1
1.2 MOTIVATION	1
1.3 LITERATURE REVIEW	2
1.4 REPORT OUTLINE	3
CHAPTER 2 SOLUTION DESIGN & IMPLEMENTATION	4
2.1 BLOCK DIAGRAM	4
2.2 FLOW CHART	8
2.4 SOFTWARE IMPLEMENTATION	12
2.5 HARDWARE IMPLEMENTATION	16
CHAPTER 3 RESULT AND RECOMMENDATIONS	21
3.1 RESULTS	21
3.2 BUDGET	27
3.3 FUTURE ASPIRATIONS.....	29
APPENDIX: PROJECT CODES	30
A-1: PYTHON CODE (LANE DETECTION).....	30
A-2: PYTHON CODE (TRAFFIC SIGN DETECTION)	37
A-3: PYTHON CODE (OBJECT DETECTION):	42

List of Figures

Figure 2.1 Block diagram of the project.....	4
Figure 2.2 Flow chart of project.....	9
Figure 2.3 Flow chart of Obstacle Avoidance.....	10
Figure 2.4 Flow chart of Lane Detection.....	10
Figure 2.5 Flow chart of Lane Detection.....	11
Figure 2.6 Dataset for Lane Detection.....	12
Figure 2.7 Dataset for Lane Detection.....	13
Figure 2.8 Graphical Analysis of Data.....	13
Figure 2.9 Sample Training And Validation Steering Angle Distributions.....	14
Figure 2.10 Processed data for Lane Detection.....	14
Figure 2.11 Graphical Analysis of Processed Data.....	15
Figure 2.12 Circuit Diagram.....	16
Figure 2.13 Diagram of PCB.....	17
Figure 2.14 Diagram of PCB.....	17
Figure 2.15 Diagram of Robot Chassis.....	18
Figure 2.16 Image of Completed Robot.....	19
Figure 2.17 Raspberry Pi Camera Configuration.....	Error!
Bookmark not defined.19	
Figure 2.18 Robot Track.....	20
Figure 3.1 Model Accuracy.....	22
Figure 3.2 Diagram of Udacity Simulator.....	23
Figure 3.3 Diagram of Udacity Simulator.....	24
Figure 3.4 Diagram of Lane Detection Simulation.....	25
Figure 3.5 Diagram of Lane Detection Simulation.....	25
Figure 3.6 Diagram of Object Detection.....	26

Chapter 1 Introduction

Self-driving vehicles are automobiles in which human help isn't required to work the vehicle. Also called "driverless" vehicles, they use sensors and image processing to control, navigate and drive the vehicle.

Presently there are no completely automated self-driving cars in our market. However, there are some semi-automated self-driving vehicles in production, from traditional vehicles with brake and path assist to fully autonomous, self-driving models which are not accessible for the market because of absence of testing.

Even though the technology is still in the beginning phases of its development, AI automated self-driving algorithm development is getting dynamically popular and could revolutionize our transportation ways (and by future improvement, our economy and our planet). The motivation behind our idea was to improve car safety and efficiency.

We wish to make a smaller scale model of a car which will be autonomous enough to self-drive to a set location while following basic traffic rules like lane assist and watching out for obstacles for our FYP by implementing self-drive functions on a small toy car to see if it works. If completed, we wish to upgrade it into a full-scale model in the future.

1.1 Problem statement

The world is evolving into a new phase where machines are used to make our life easier and provide faster and safer services to humans without the mistakes caused by human errors. USA NPR Surveys report that over 90% of fatal car crashes are caused by mistakes made by humans. The only way to stop this is to make cars which are fully autonomous, unless we do this there will still be mistakes made by humans which will cause thousands of automobile deaths every year.

1.2 Motivation

By looking at the factor of human error in critical accidents over the world which takes the life of thousands of people every year we knew we had to come up with a solution to that problem, we have used this as motivation to complete our project. Using our project, we will design a car which can be fully automated in the future which will decrease the amount and severity of accidents in the world saving multiple lives in the process.

1.3 Literature Review

This paper consisted of knowledge about a fully automated working model of a self-driving vehicle which was equipped for transportation from one area to the next or to state on various sorts of road conditions, for example, bend roads, normal straight roads and a combination of straight and bend roads. A camera system is placed on the head of the vehicle alongside a Raspberry Pi3 which helps send the pictures from the device to the designed Convolutional Neural Network(CNN) which at that point predicts the scenario accompanying bearings. for example moving right, moving left, keep going forward or stopping altogether which is then trailed by imparting a sign from the Arduino to the vehicle controller and because of it the vehicle travels in the ideal course with no human intercession. [1]

The reason for this report was to manufacture an independent Remote-Controlled Car that utilizes ANN or Artificial Neural Network algorithms for controlling the vehicle. It depicts the hypothesis behind the entire neural system and independent driving vehicles, and how a model with a camera module as its sole information gathering area is intended to test and assess the calculation capacities. The ANN is a decent calculation that assists in perceiving designs in a picture, it can with a preparation set, containing 2000 pictures, group a picture with 90% of exactness rate. The primary commitment of this paper comprises in utilizing a solitary camera for route, potentially for snag evasion. Yilmaz Kaya took a shot at two phases of hepatitis, by executing outrageous learning machines (ELM) and harsh sets (RS). Early finding was finished utilizing RS, while ELM was utilized for grouping purposes. The exactness acquired by utilizing this methodology was 96 percent.[2]

Rong-Ho Lin proposed a model using regression tree and case-based reasoning. First stages are diagnosed using a regress tree (CART) that determines if the patient is undergoing any liver disease or not, if the results are positive then CBR is implemented to classify the type of disease. Accuracy level of these two methods was 92 percent using CART and 87 percent using CBR. [3]

1.4 Report Outline

This report is further divided into multiple chapters as listed below.

In chapter 2, the proposed solution is discussed in detail. It includes details of the block diagram, Flow chart of the process. It also includes all the hardware use and their data inputs and outputs.

Chapter 3 discusses the results acquired by testing the device using real time simulations. It further discusses the deductions obtained from the results and the recommendations/future work that is proposed for further enhancements.

Chapter 2 Solution Design & Implementation

This chapter discusses the complete design and implementation of the proposed device. Section 2.1 discusses the block diagram and module specifications. The details of flow charts are presented in section 2.2 whereas, Section 2.3 discusses the circuit model simulated on Fritzing. Section 2.4 discusses in detail Software implementation of the project. The hardware implementation is presented in Section 2.5.

2.1 Block Diagram

Figure 2.1 shows the complete block diagram of the project. The details of each block with related technical specifications are discussed below.

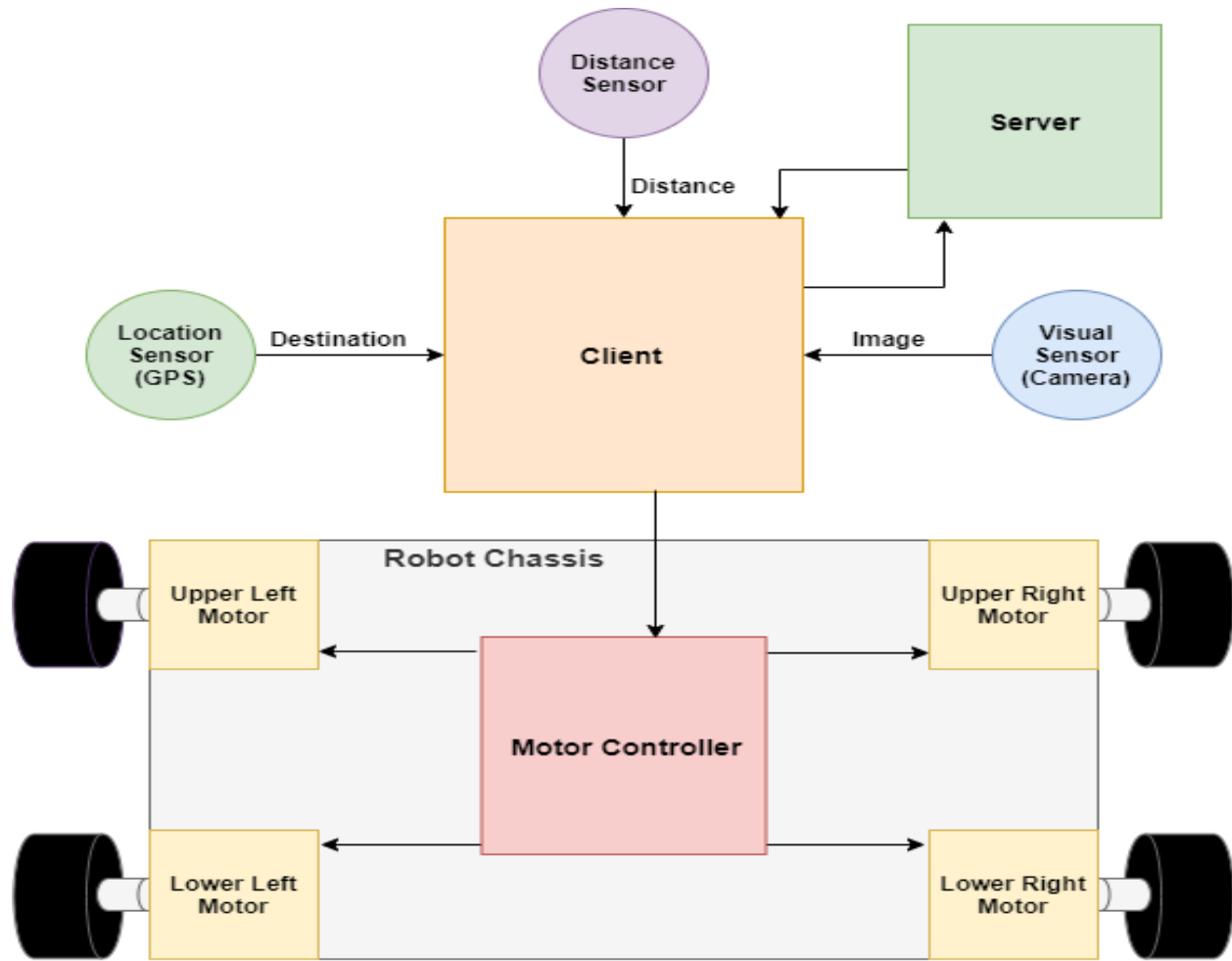


Figure 2.1 Block diagram of the project.

Module Specifications:

1.Server

This unit will take data of the location, obstacle distance and camera footage from the client and read all its data and process it. After processing it will send data back to the client.

Data Input:

1. Live video from Visual Sensor
2. Location coordinates from GPS Sensor
3. Distance in centimeters from Ultra Sonic sensor.

Data Output:

1. Voltage signal to motor controller for steering and moving

Specs:

- 1 Processing speed 2.10 GHz
- 2 Random access memory 2GB

2. Visual Sensor (Camera)

The Camera used is a 16 MegaPixel software enhanced camera, with night vision, and built in microphone. It provides us with excellent sound quality and photo results. You can use live video processing to detect where to go on the road.

Data Input:

1. Live Video from its lens.

Date Output:

1. Live Video from its lens onto the main processing unit.

1. Client

This unit will take data from the Distance, GPS and Visual sensor and will send all the data to the server. After receiving processed data from the server, it will accordingly send signals to the motor controller which will further move or steer the robot in the direction of control units choosing to avoid obstacles.

2. GPS Sensor

It gets signals from circling Global Positioning System (GPS) satellites and afterward utilizes the signs to compute position and speed. This sensor additionally gives a profoundly exact one-beat per-second (PPS) yield for exact planning estimations.

Data Input:

1. 12 parallel channel signals from 12 satellites.

Data Output:

1. Location Coordinates
Accuracy: Less than 15 meters

3. Motor Controller

The motor controller will help take data from the main processing unit and use it to know which wheels to activate for moving forward or steering.

Data input:

1. Voltage signal from Main processing unit

Data output:

- 1 Continuous voltage signal to motor driver

Specs:

1. Voltage: 5.5 to 16 V. It works with 5 V logic levels
2. Current: can deliver up to continuous 14 A

4. Distance Sensor (HC-SR04 ultrasonic sensor)

This Ultrasonic sensor uses sonar to find the distance of the obstacle in front of it. It offers high accuracy and is very small in size so it can be installed anywhere. Its distance is from 2cm to 400cm which makes it very versatile. Its values are not affected by any type of metals or surfaces so it always gives the proper value.

Data Input:

1. Time of UltraSonic burst waves of 40 KHz to hit obstacles and come back to the sensor.

Range: 0.116 milliseconds – 23.2 milliseconds

Data Output:

1. Float value of distance in centimeters from Sensor to obstacle. Sent to the main processing Unit.

Range: 2cm – 400cm

5. 4WD Robot Chassis

This chassis includes a plastic body, along with 4 wheels which will be used as the body for our robot.

Dimensions:

1. Length: about 25.3cm
2. Width: about 14.8cm
3. Tire Diameter: 6.6cm

6. Small DC Motor

These motors will be used to drive the robot forward or backward, depending on the requirement of the road.

Data Input:

1. Signal from motor controller

Data Output:

1. Rotation of wheels for up to 8000 rpm

Specs:

- 1 Rated voltage: DC12V
- 2 No-load speed: $8000 \pm 10\%$ rpm
- 3 No-Speed Current: 0.22A
- 4 Weight: 200g

2.2 Flow Chart

Figure 2.2 shows the complete flow chart of the project. The details are discussed below.

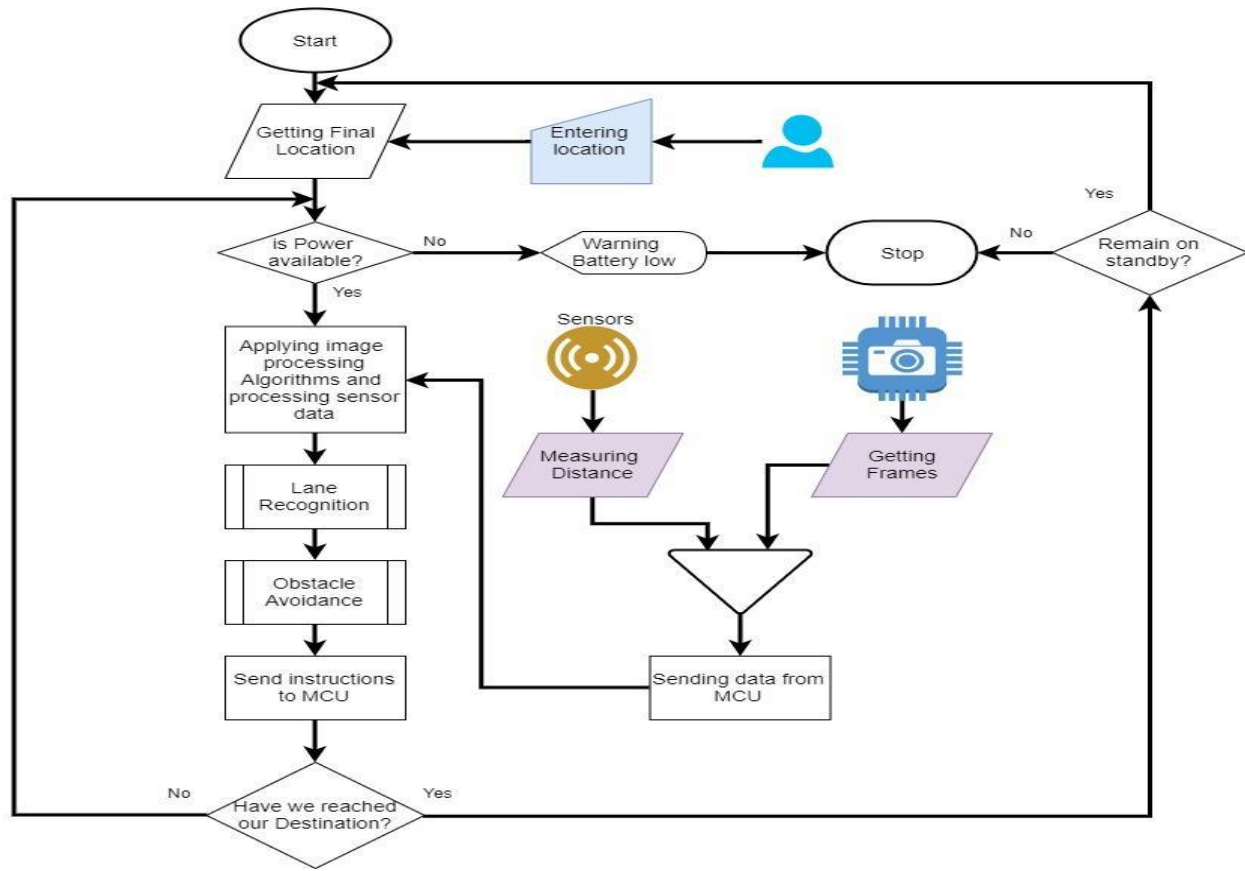


Figure 2.2 Flow chart of project.

This is our main flow chart. It shows the main processes of our project and those processes can further be divided and shown on other flow charts. So, we start by getting our final location by the user. This could be a direct input in the program, or some GUI will be provided. The first thing we check in a system is the power, if there is no power available, then no work can be done. After everything is turned and running, we will start getting frames and sensor data from the MCU to the laptop. The laptop will start processing the frames using image processing, which will recognize obstacles and lanes on the road. Lane recognition will help us find the correct lane to follow and help us change lanes if needed be. Obstacle avoidance will help us recognize obstacles such as cars and pedestrians and will give the right commands to avoid them. The instructions from these two processes will be sent back to MCU , which will execute these commands on the motors.

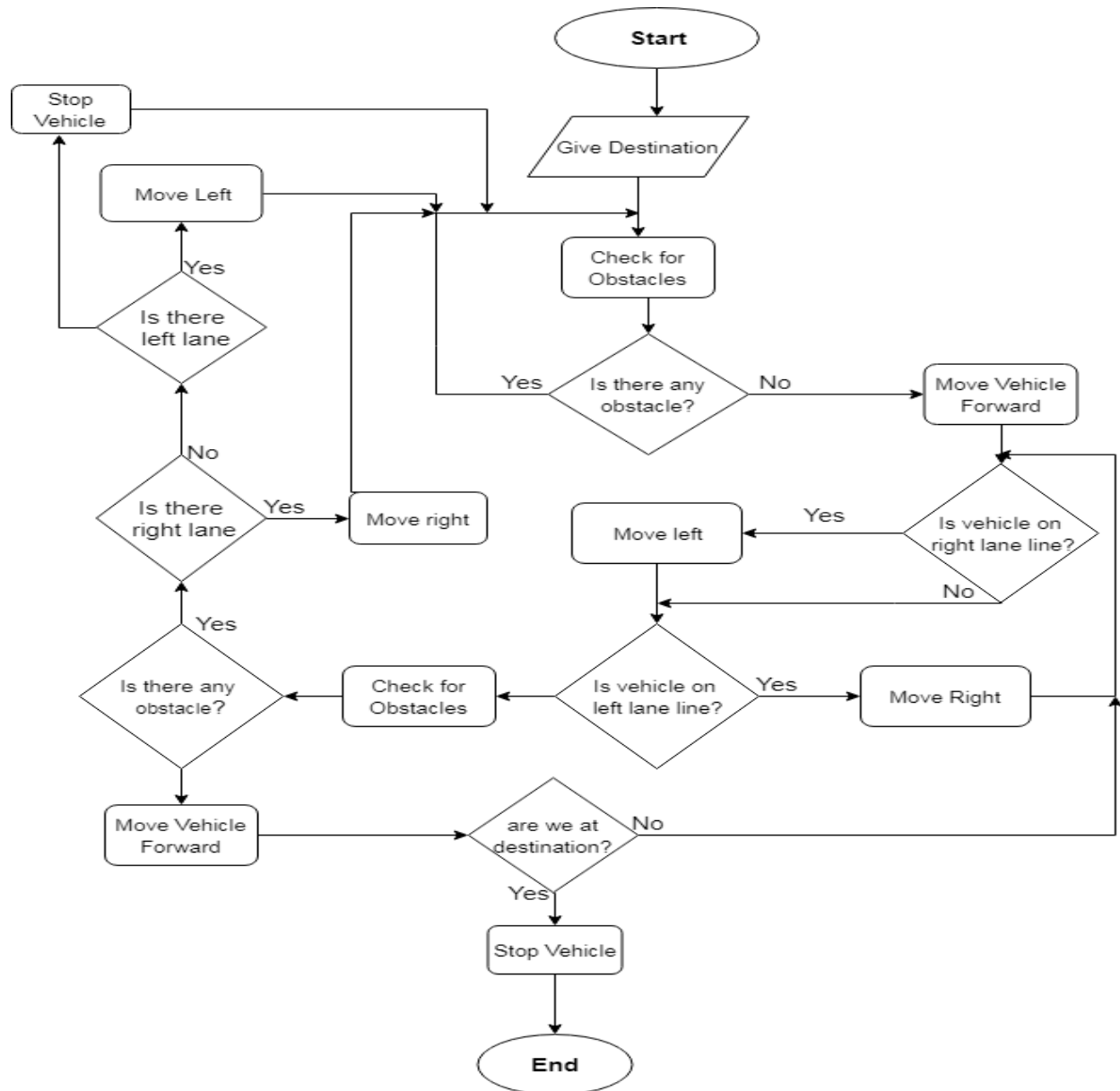


Figure 2.3 Flow chart of Obstacle Avoidance.

Another sub process that needs more explaining is obstacle avoidance. First, we get data from our distance sensors, to see if an obstacle is nearby. This is so we can avoid unnecessary processing in case there is no object nearby. If there is an object inside the warning range, we will start our image recognition to recognize where obstacles exist. If we have an obstacle on right, we turn on motors to move left, if we have the obstacle on left, we turn the motors to move right. If we have obstacles on straight ahead, we check the right side first ,if it is clear. If it is clear we move right otherwise we check the left side. If we even have obstacles on the left side as well, we come to a stop. Until it is clear for us to move again.

Lane Detection Sub Flow Chart:

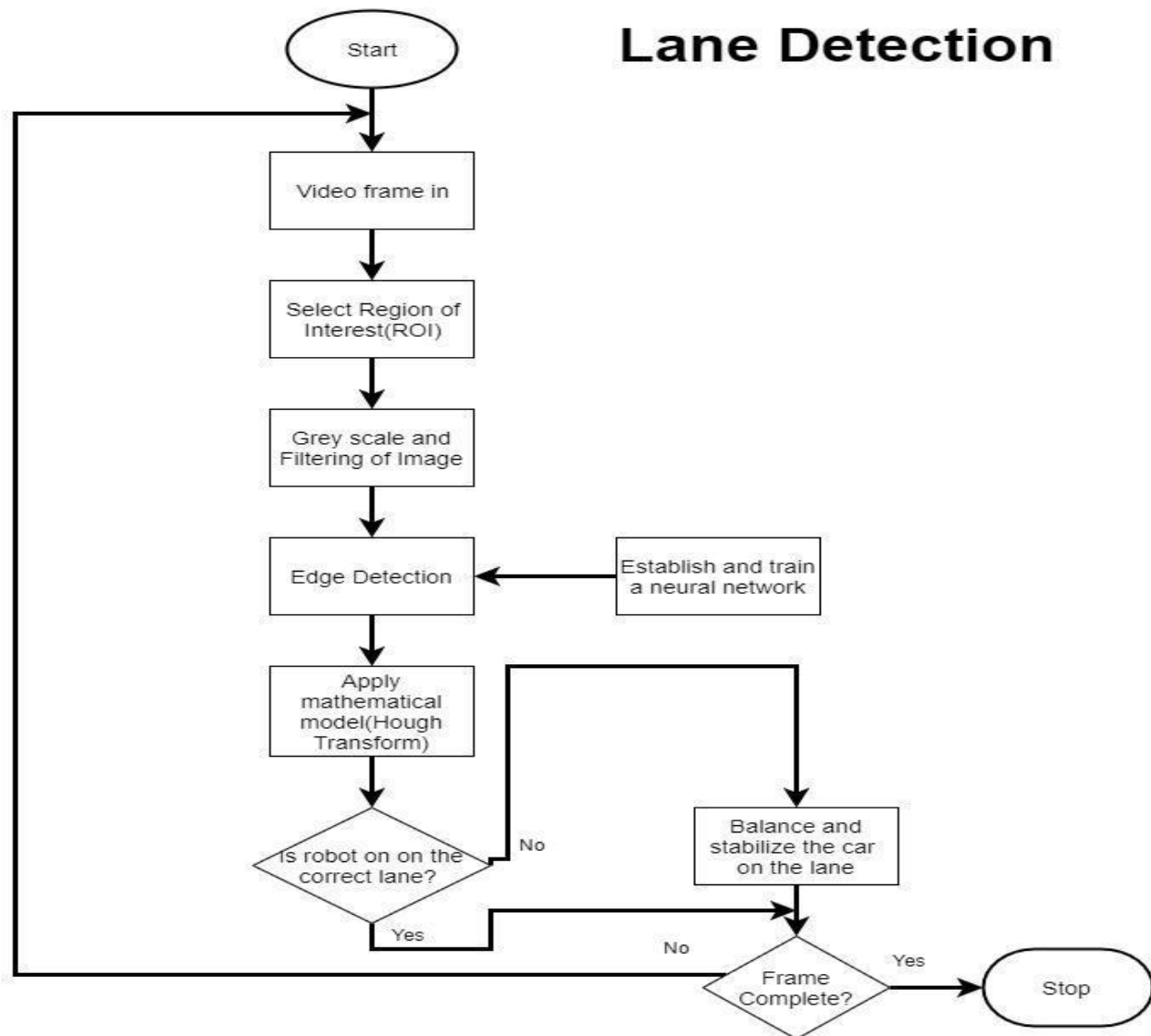


Figure 2.4 Flow chart of Lane Detection.

From the main body, a sub process needs further explanation. So, we decided to have a different flow chart for lane detection. When frames come in from our MCU, we need from these frames to find our lane. First, we select our region of interest(ROI). The process of ROI is needed to concentrate on only the area that we need. Since we are working in real time and lane recognition needs time to process, we cannot waste time on processing unnecessary data. Next step is grey scaling and image filtering. This is also done to make the process more simple and faster. Grey scaling is just making the image in shades of grey, so we don't have to process colors, which is also unnecessary in lane detection. These two processes are part of pre-processing steps as they are preparing the frame for the actual processing. Then comes edge detection, which is the main heavy task of lane detection. For edge detection we are using neural networks as they are efficient, accurate, faster and easier to implement. We will take the help of tensorflow and kesar

to establish our neural network. From lane detection we get some points which lie on the lane. Then Hough transform is used to extend those points to lines which can be formed to display our lane. After we have identified our lane, we must balance our robot to stay inside the correct lane, most likely using a PID controller.

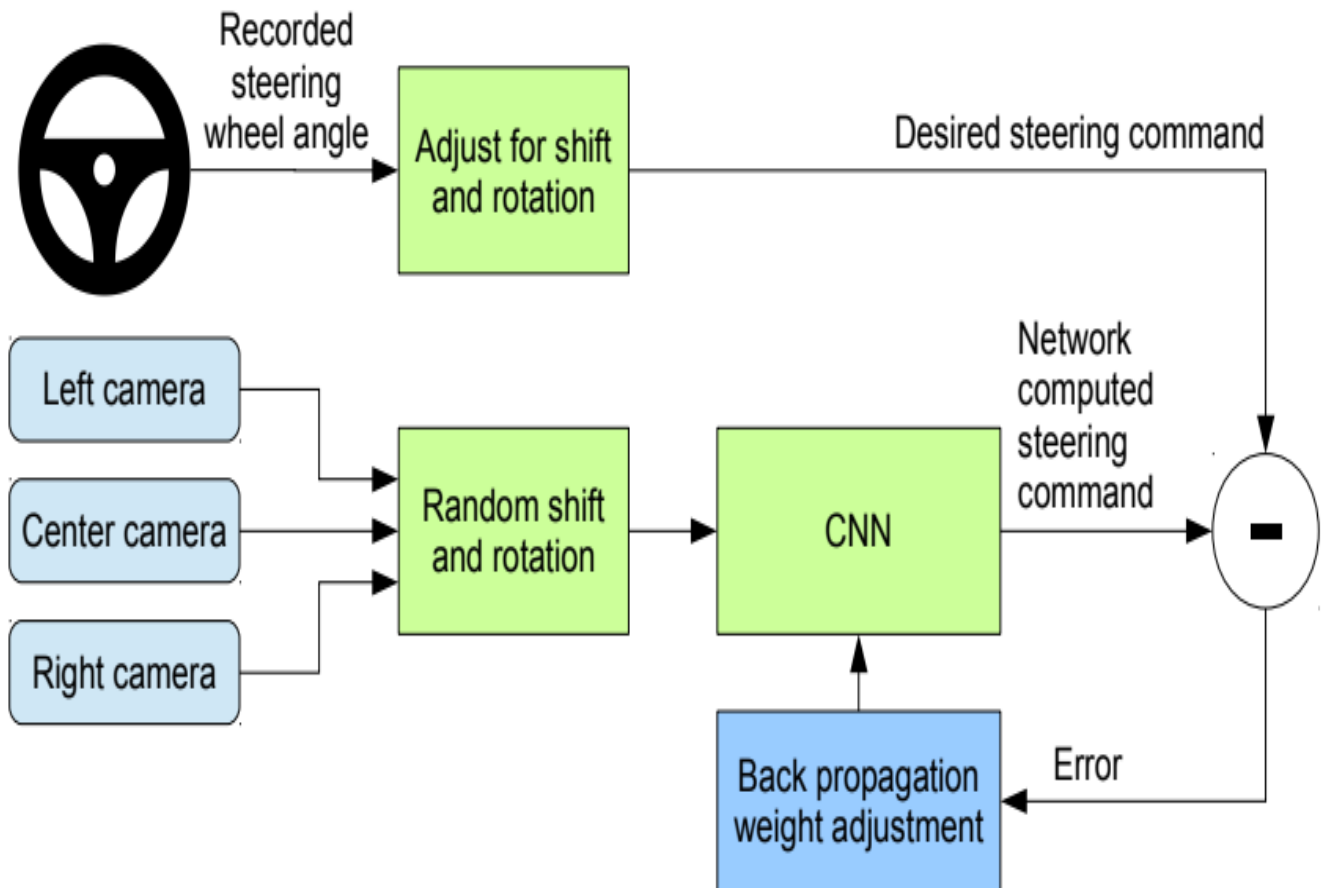


Figure 2.5 Flow chart of Lane Detection.

This flowchart is also a follow up to lane detection where we use 3 cameras to detect if we are going out of our lane. The left side camera is used to detect left side lanes while the right-side camera is used to detect right side lanes. By using the front camera, it does both the activities that the other cameras do while also detecting for other corners which are present on the track. Using these cameras this sets a specific angle for rotation which it does using convolutional neural network algorithms which is run in the server. After running this, it removes any error in the calculations and sends instructions back to the robot which then adjusts its steering and rotation to avoid going off the lane.

2.3 Software Implementation

First, the dataset. The dataset has— center, left, right (camera image paths), steering, throttle, reverse, speed (values) columns which help in its process. We used panda's data frame to sort and store our data.

right	steering	throttle	reverse	speed
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_382.jpg	0.0	0.0	0	0.000079
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_526.jpg	0.0	0.0	0	0.000082
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_669.jpg	0.0	0.0	0	0.000078
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_802.jpg	0.0	0.0	0	0.000078
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_937.jpg	0.0	0.0	0	0.000080

Figure 2.6 Dataset for Lane Detection

To clean the data a bit, we removed the path of images as they were all in the same folder.

right	steering	throttle	reverse	speed
right_2019_07_22_20_38_15_382.jpg	0.0	0.0	0	0.000079
right_2019_07_22_20_38_15_526.jpg	0.0	0.0	0	0.000082
right_2019_07_22_20_38_15_669.jpg	0.0	0.0	0	0.000078
right_2019_07_22_20_38_15_802.jpg	0.0	0.0	0	0.000078
right_2019_07_22_20_38_15_937.jpg	0.0	0.0	0	0.000080

Figure 2.7 Dataset for Lane Detection

Chapter2: Solution Design and Implementation

After that we sorted the data accordingly to graphical analysis. Graphical analysis of the data is necessary to make sense out of it and to have a better idea of what preprocessing the data needs. So, we took steering wheel angle values and plotted their distribution. As you can see in our data that almost all the data is in the middle which is indicating that mostly the car is moving in a straight line on a straight road.

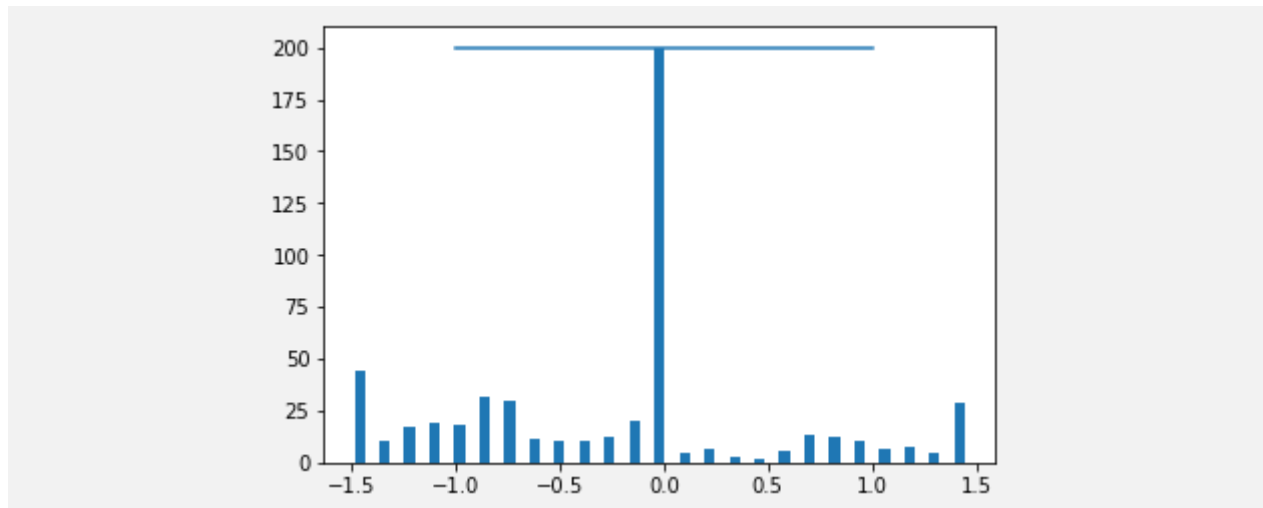


Figure 2.8 Graphical Analysis of Data

Then we loaded all the images and steering wheel angles into NumPy arrays for later processing. Then we divided the data in ratios of 80% for data training and the 20% for testing on unseen scenarios and images. We also plotted its validation steering angle and sample training distributions on the curves below.

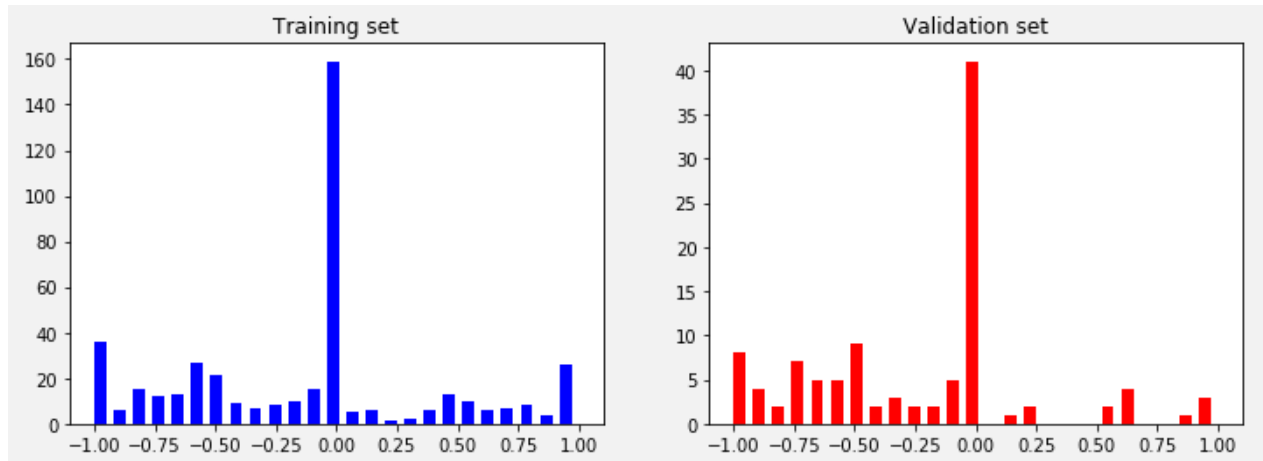


Figure 2.9 Sample Training And Validation Steering Angle Distributions

Next step was to preprocess the data. This included cropping the images. Removing unnecessary features and changing from RGB to YUV format. We also used the gaussian blur and reduced the size of images. We can see the difference in the picture below.

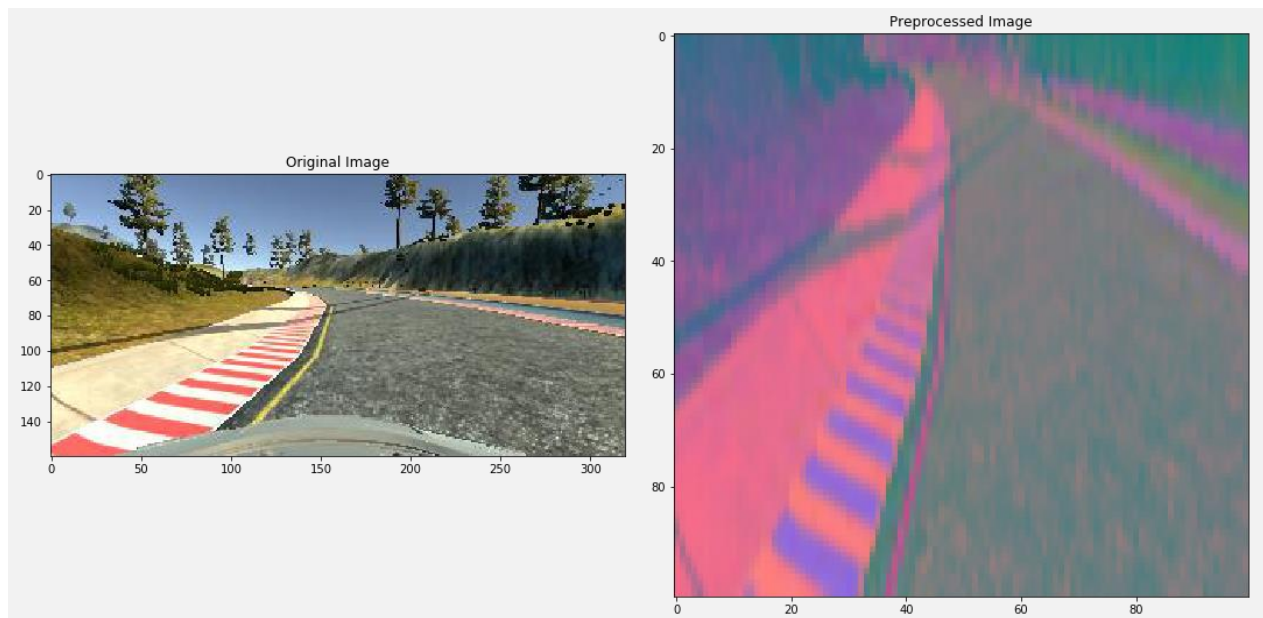


Figure 2.10 Processed data for Lane Detection

After that we found graphical analysis of the data. Graphical analysis of the data is required to make sense out of the data we have and is used to have a better idea of what preprocessing the

Chapter2: Solution Design and Implementation

data needs. So, we used the sheeting wheel values and plotted their distribution. The data shows that the car is on a straight road and that is shown by all the peaks in the middle.

After that we convert all image data to NumPy arrays. Now it was the time to build our CNN model. We used a previously trained model available in Keras library named ResNet. We removed the last 4 layers as they were not needed. So now we had 3 layers in our model. In the first layer we had 100 neurons and 50% dropout rate. The dropout rate was consistent in all layers. The second layer had 50 neurons and the last layer had 10. Before training we normalized our data for better processing. Finally, I trained my network for 25 epochs. The picture below shows us our models' accuracy on paper. As we can see, as we train our model for more epochs, it seems to improve sharply, until near 20 epochs where it starts to saturate. That's why 25 epochs were enough as beyond that, the model would start overfitting.

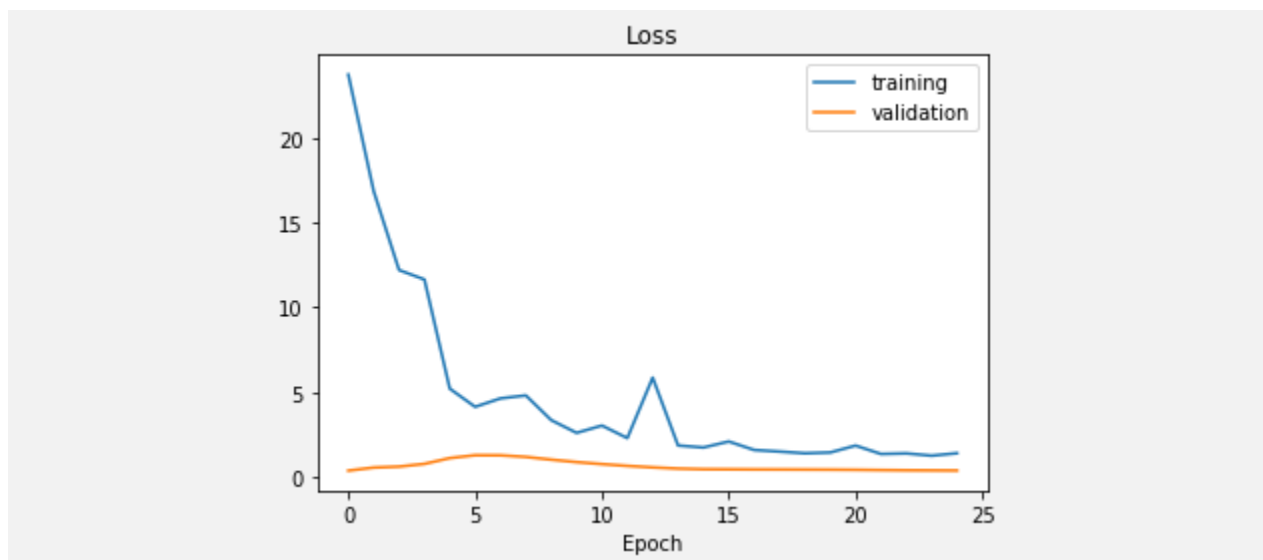


Figure 2.11 Graphical Analysis of Processed Data

2.4 Hardware Implementation

Hardware implementation was carried out in following two stages

1. Complete motor control system on Fritzing.
2. Internal circuit.

Complete motor control system on Fritzing

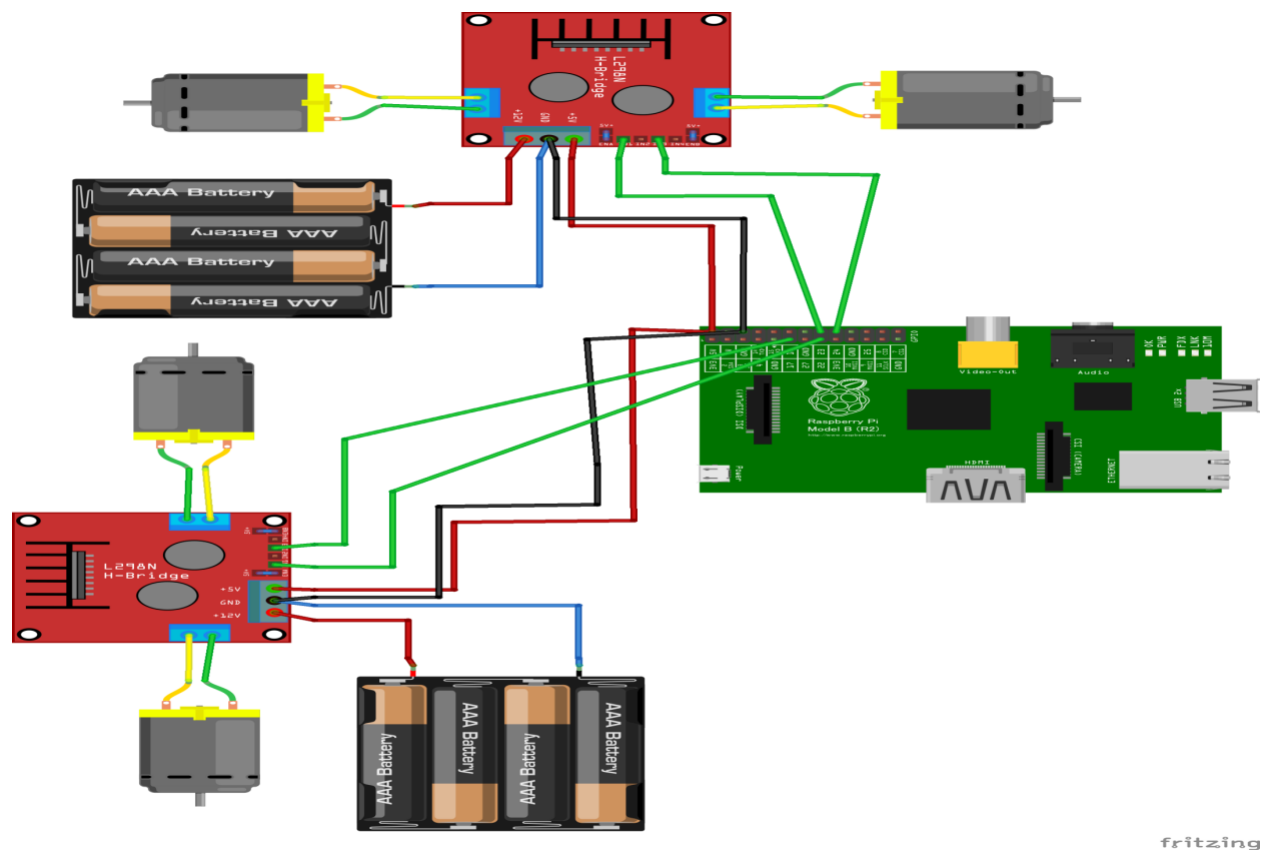


Figure 2.12 Circuit Diagram for Configuring Motor drivers with Raspberry Pi and Motors.

This circuit shows how raspberry pi sends the instructions from its pins to the motor controllers. The motor controller is powered by a high current 7 Volt Lipo battery. Each motor controller controls one side of the robot, the left side and the right side. By running the left side motors, the robot rotates towards right side, by running right side motors the robot rotates towards left side. By running all 4 motors at once the robot moves forward.

Pin 16 and 18 of raspberry pi controls the right-side motors of the robot.

Pin 11 and 13 of raspberry pi controls the left side motors of the robot.

Chapter2: Solution Design and Implementation

In the motor controllers pin 1 and 2 are used to control first motor while pin 3 and 4 are used to control the other motor.

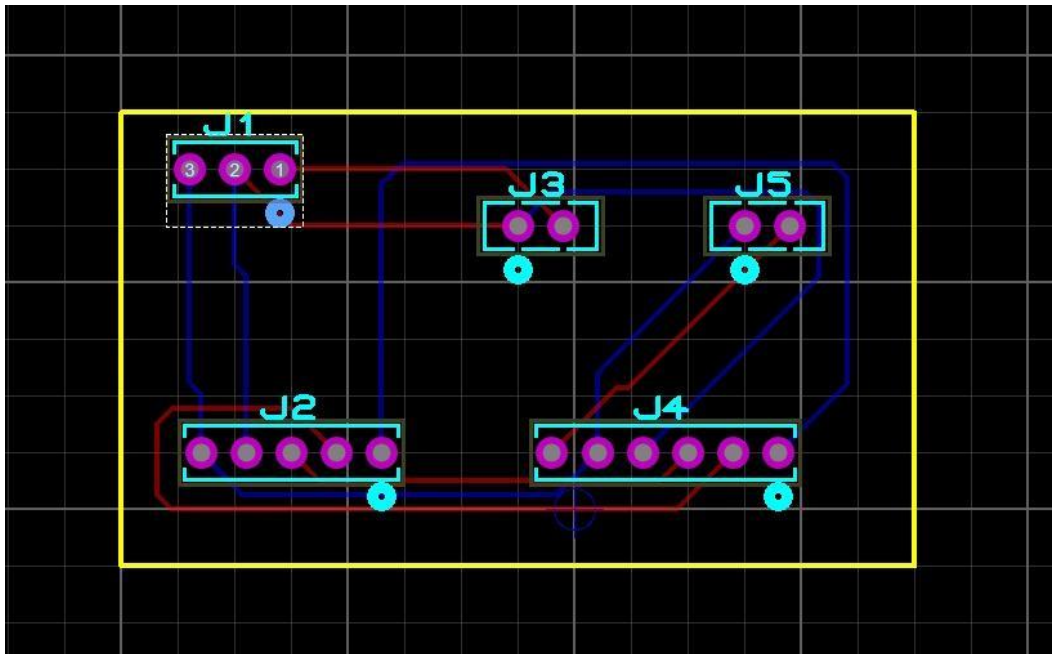


Figure 2.13 Diagram of PCB.

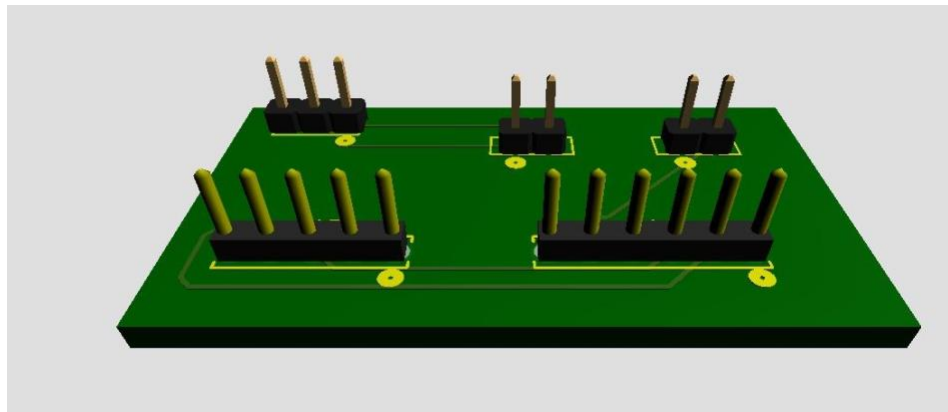


Figure 2.14 Diagram of PCB.

The PCB was made to make the circuit easier and cleaner to navigate across. Without them there would be a lot of wires in the circuit which would cause problems if any wire was damaged.

J1 is the Raspberry Pi while J4 and J2 are Motor Drivers. While J3 and J5 are the motors which are connected to the motor drivers.

1. 4WD Robot Chassis:



Figure 2.15 Diagram of 4WD Robot Chassis.

This Robot was ordered in parts and assembled on university grounds. This model has 4 pre attached motors and wheels which made it easier to configure for our project.

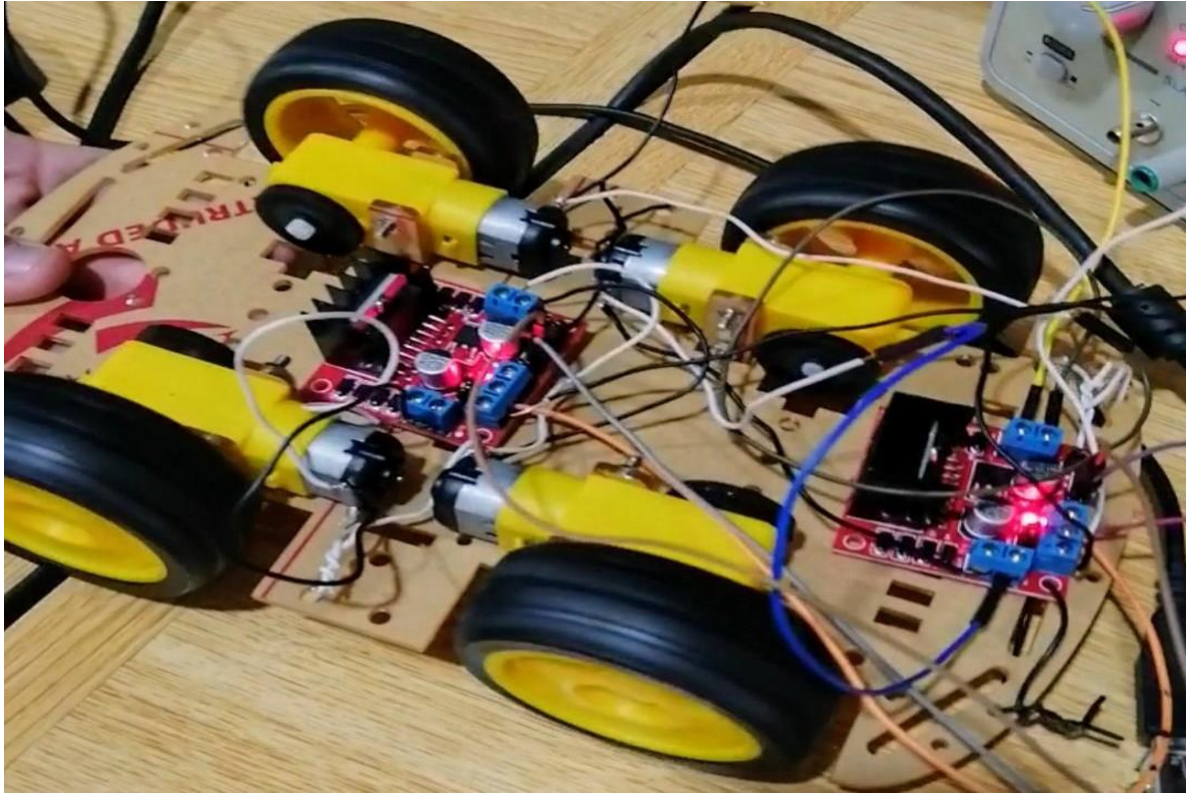


Figure 2.16 Diagram of Completed Robot.

This Robot was tested and could take instructions from the laptop and run wirelessly without any human control. This could be run remotely using Wi-Fi.



Figure 2.17 Diagram of Raspberry Pi camera configured to robot.

The Raspberry Pi camera was configured to the Hardware model.

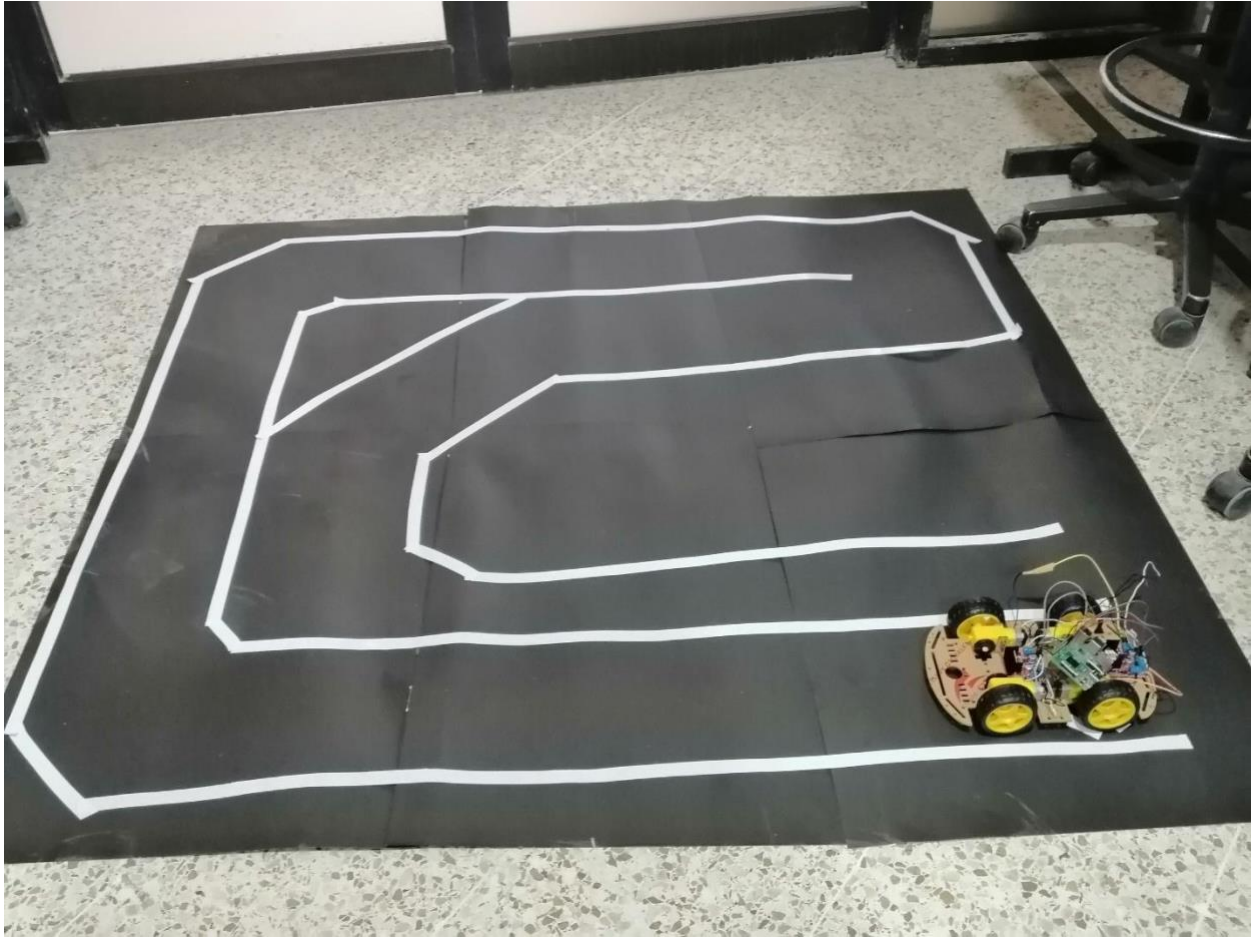


Figure 2.18 Diagram of Robot Track

This track was used in testing of the AI so that it can detect the lanes Infront of it and stay within it. The design was given a proper road so that it will not be confused if we change its track in the future. Objects could also be placed on the track which would help train its object avoidance.

Chapter 3 Result and Recommendations

3.1 Results:

We have made our CNN lane detection model to drive our autonomous car. To test the model, we must use simulation. One very popular and accurate simulation was the Udacity autonomous car simulator. We chose the Udacity autonomous car simulator for a few reasons. First it was source code independent. As in, it only needed the AI model to run, which meant that there would be no compatibility issues. The model could be made in thonny or Jupiter or google colab, if you have the trained model, the simulation could run. This was extremely advantageous to us as we were using google colab for our model training. Since our laptops didn't have powerful GPUs, cloud computing was our main option. Through google collab we would get our model and the model would be run on a simulated environment in Udacity.

The second reason for choosing this simulator was that data collection was possible within the simulator. This meant that we could gather the simulators own data and later train the model with it. This made our model more accurate and simulated real life situations better. As the process was very similar to how you would run an autonomous car in real life. You would have to run it first by yourself to collect data. Through that data your model would be trained.

The final reason was that the Udacity simulator was not as resource demanding as other simulators. For example, we also used another simulator called Carla simulator. Which also was like Udacity. The difference was that Carla was very computation intensive. When we ran the simulator, it ran extremely slow. Whereas Udacity runs okay and hardly lags.

The picture below shows us our models' accuracy on paper. As we can see, as we train our model for more epochs, it seems to improve sharply, until near 20 epochs where it starts to saturate. That's why 25 epochs were enough as beyond that, the model would start overfitting.

Chapter 3 Results and Recommendations.

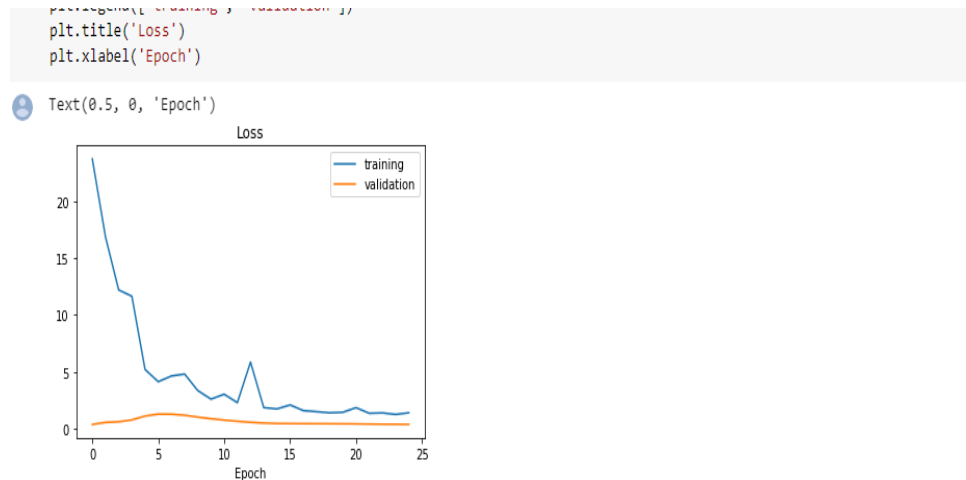


Figure 3.1 Diagram of Model Accuracy.

To run the simulator, first download the simulator, also you need to create an environment in anaconda. Anaconda or some other source of python engine is needed to run the simulations. These were the libraries needed in the environment:

dependencies:

- h5py
- pillow
- scikit-image
- python==3.5.2
- scikit-learn
- matplotlib
- Jupiter
- pandas
- opencv3
- SciPy
- eventlet
- NumPy
- flask-socketio

Chapter 3 Results and Recommendations.

- seaborn
- imageio

You would also need to download the repo of car behavioral simulator as it would be needed to run the simulator. We named our environment FYP. Now we activated our environment by first opening anaconda prompt and typing, conda activate FYP.

This opened our environment. After that we changed directory towards the behavioral car simulator folder. We put our model in that folder. Before we run the code, we open the Udacity simulator and set it to autonomous mode. Then we run the code driver.py with our model in anaconda prompt. This would result in our car in the simulator to run autonomously.

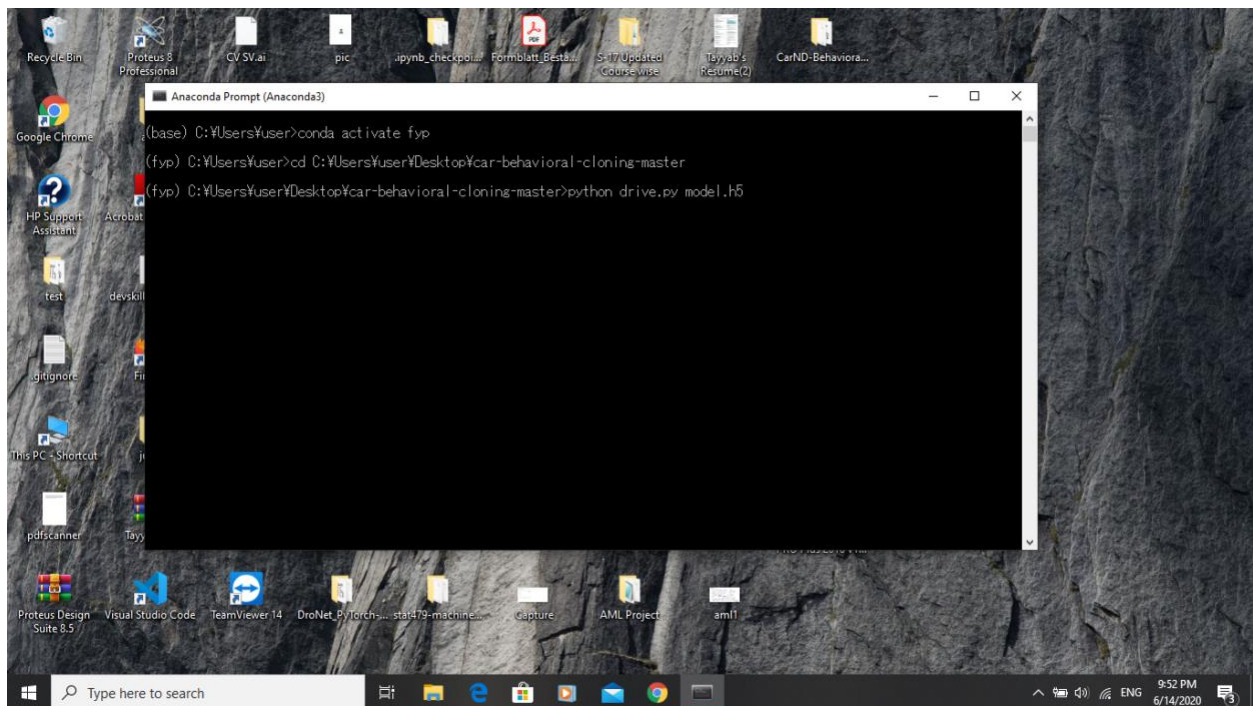


Figure 3.2 Diagram of Udacity Simulator



Figure 3.3 Diagram of Udacity Simulator



Figure 3.4 Diagram of Lane Detection Simulation

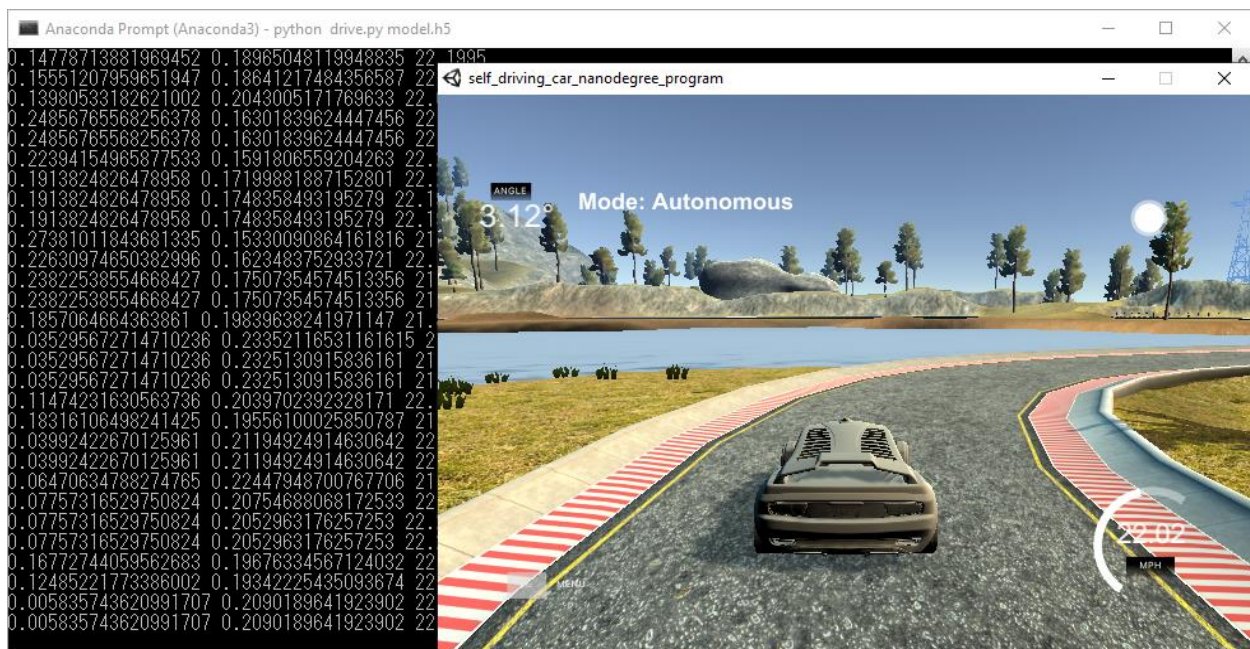


Figure 3.5 Diagram of Lane Detection Simulation

Chapter 3 Results and Recommendations.

The Object Detection was also implemented using the simulator.

It was done in 2 steps:

1. Using Convolutional Neural Network
2. Using Ultrasonic sensors

By combining both these 2 aspects we designed a system which detected objects and avoided them.

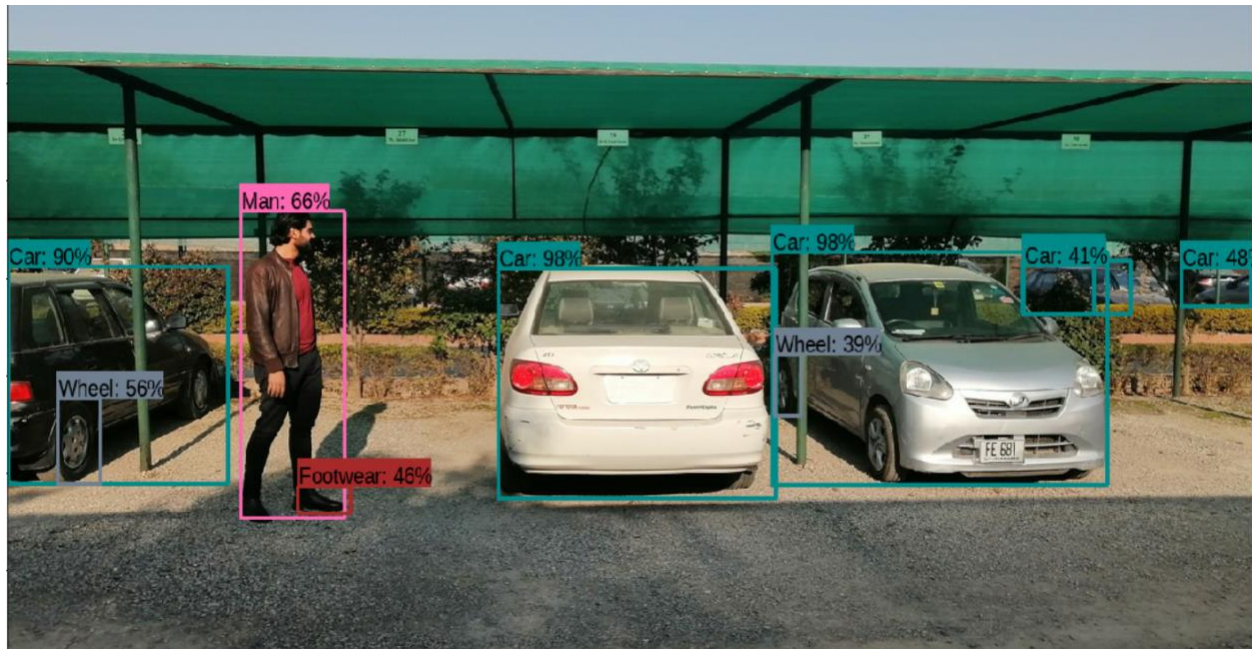


Figure 3.6 Diagram of Object Detection

3.2 Budget

The total expenses that occurred during the course of our project are mentioned below

S No.	Items	Price/Unit	Units	Price (Rps)
1	Raspberry Pi 3B	6,800	1	6,800
2	5MP Raspberry Pi Camera Module v1.3	1,100	1	1,100
3	LM298 MOTOR DRIVER MODULE	270	2	540
4	4WD Robot Chassis + 4 DC Motors	1,800	1	1,800
5	HC SR04 HC-SR04 Ultrasonic Sensor	380	1	380
3	Car Track	200	1	200
4	Webcam	1,300	1	1,300
5	Power Bank	1,000	1	1,000
6	Vga to HDMI Cable	700	1	700
7	L293 L293D Driver	280	1	280
8	7 Volt Lipo Battery	1,395	1	1,395
	Total Budget for Robot:			15,495 Rps

3.3 Future Aspirations

For this project we had 2 stages of future aspirations:

Short Term:

To implement this on Public Transport Services such as Metro Bus which will make their vehicles fully autonomous on their predetermined tracks. This project can be implemented on any vehicle with a predetermined track currently. The issue with this is getting permission and funds to implement this on the vehicle. This will decrease the costs of drivers for the company.

Long Term:

To further develop these algorithms so that a self-driving kit is made which can be installed onto any car compatible with a sheeting motor so that the car will become autonomous. This will solve the main issue we had at the start of the report which was to take out the human element from driving. This will be a much cheaper solution for self-driving vehicles and the cost of the kit would be a fraction of getting an electric car. This will revolutionize the market where it is launched.

Appendix: Project Codes

A-1 Python Code (Lane Detection) :

```
import keras
from sklearn.model_selection import train_test_split
import numpy as np
from keras.optimizers import Adam
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
from keras.models import Sequential
import ntpath
import matplotlib.image as npimg
import pandas as pd
from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense
import cv2
import os
import random

datadirect = 'Self-Driving-Car'
col = ['center1', 'left1', 'right1', 'steering1', 'throttle1', 'reverse1', 'speed1']
car_info = pd.read_csv(os.path.join(datadirect, 'file.csv'), names = col)
pd.set_option('display.max_colwidth', -1)
car_info.head()

def leaf_path(path):
    head1, tail1 = ntpath.split(path)
    return tail1

car_info['center1'] = car_info['center1'].apply(leaf_path)
car_info['left1'] = car_info['left1'].apply(leaf_path)
car_info['right1'] = car_info['right1'].apply(leaf_path)
car_info.head()

number_of_cbin = 25
cbin_sample = 200
histo, cbin = np.histogram(car_info['steering1'], number_of_cbin)
center1 = cbin[:-1] + cbin[1:] * 0.5

plt.bar(center1, histo, width=0.05)
```

```

plt.plot((np.min(car_info['steering1']),      np.max(car_info['steering1'])),      (cbin_sample,
cbin_sample))

rem_list = []
for j in range(number_of_cbin):
    list1_ = []
    for i in range(len(car_info['steering1'])):
        angle_steer = car_info['steering1'][i]
        if angle_steer >= cbin[j] and angle_steer <= cbin[j+1]:
            list1_.append(i)
    list1_ = shuffle(list1_)
    list1_ = list1_[cbin_sample:]
    rem_list.extend(list1_)

print('Total car_info: {0}'.format(len(car_info)))

car_info.drop(car_info.index[rem_list], inplace=True)

histo, _ = np.histogram(car_info['steering1'], (number_of_cbin))
plt.bar(center1, histo, width=0.05)
plt.plot((np.min(car_info['steering1']),      np.max(car_info['steering1'])),      (cbin_sample,
cbin_sample))

print('Removed: {0}'.format(len(rem_list)))
print('Remaining: {0}'.format(len(car_info)))

def image_steering1(datadirect, df):
    path_to_image = []
    steering1 = []
    for i in range(len(car_info)):
        car_info_ind = car_info.iloc[i]
        center1, left1, right1 = car_info_ind[0], car_info_ind[1], car_info_ind[2]
        path_to_image.append(os.path.join(datadirect, center1.strip()))
        steering1.append(float(car_info_ind[3]))
    paths_to_image = np.asarray(path_to_image)
    steering1s = np.asarray(steering1)
    return paths_to_image, steering1s

paths_to_image, steering1s = image_steering1(datadirect + '/IMG', car_info)

train_data_X, x_val, train_data_Y, y_val = train_test_split(paths_to_image, steering1s,
test_size=0.2, random_state=0)

```



```
print("Training Samples: {}\nValid Samples: {}".format(len(train_data_X), len(x_val)))
fig, axes1 = plt.subplots(1, 2, figsize=(12, 4))
axes1[0].hist(train_data_Y, bins=number_of_cbin, width=0.05, color='green')
axes1[0].set_title('Training set')
axes1[1].hist(y_val, bins=number_of_cbin, width=0.05, color='yellow')
axes1[1].set_title('Validation set')

def image_preprocess(image):

    image = npimg.imread(image)

    image = image[60:135, :, :]

    image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)

    image = cv2.GaussianBlur(image, (3, 3), 0)

    image = cv2.resize(image, (100, 100))

    image = image / 255
    return image

images1 = paths_to_image[100]
origin_images1 = npimg.imread(images1)
processed_images1 = image_preprocess(images1)

fig, axes1 = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()
axes1[0].imshow(origin_images1)
axes1[0].set_title('Before Preprocess')
axes1[1].imshow(processed_images1)
axes1[1].set_title('After Preprocess')

train_data_X = np.array(list(map(image_preprocess, train_data_X)))
x_val = np.array(list(map(image_preprocess, x_val)))
```

```
plt.imshow(train_data_X[random.randint(0, len(train_data_X)-1)])
print(train_data_X.shape)

from keras.applications import ResNet50

resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(100, 100, 3))

for layer in resnet.layers[:-4]:
    layer.trainable = False

for layer in resnet.layers:
    print(layer, layer.trainable)

def nvidia_model():
    mod_classification_lane = Sequential()
    mod_classification_lane.add(resnet)
    mod_classification_lane.add(Dropout(0.5))

    mod_classification_lane.add(Flatten())

    mod_classification_lane.add(Dense(100, activation='elu'))
    mod_classification_lane.add(Dropout(0.5))

    mod_classification_lane.add(Dense(50, activation='elu'))
    mod_classification_lane.add(Dropout(0.5))

    mod_classification_lane.add(Dense(10, activation='elu'))
    mod_classification_lane.add(Dropout(0.5))

    mod_classification_lane.add(Dense(1))

    optimizer = Adam(lr=1e-3)
    mod_classification_lane.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
    return mod_classification_lane

mod_classification_lane = nvidia_model()
print(mod_classification_lane.summary())

history_mod_classification_lane = mod_classification_lane.fit(train_data_X, train_data_Y,
epochs=25, validation_data=(x_val, y_val), batch_size=128, verbose=1, shuffle=1)

plt.plot(history_mod_classification_lane.history['loss'])
plt.plot(history_mod_classification_lane.history['val_loss'])
plt.legend(['training', 'validation'])
```

Chapter 3 Results and Recommendations.

```
plt.title('Loss')  
plt.xlabel('Epoch')
```

A-2 Python Code (Traffic sign detection):

```
## Importing

# Import the necessary modules for information control and visual portrayal

import pandas as pd
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
import matplotlib as matplot
from sklearn.model_selection import train_test_split
import cv2
import os
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

df_meta = pd.read_csv('Meta.csv', index_col=None)

df_data=pd.read_csv('Train.csv', index_col=None)

# We are bringing in the two information and its meta to do a superior information examination

# Data Inspection

# Check to check for any missing qualities in our informational index

df_data.isnull().any()

print(df_meta.head())

# The information is by all accounts neatly designed so no requirement for control. All the
shapes and hues are given numbers.

#But the way isn't required for information examination so we will evacuate it

df_meta= df_meta.drop(columns="Path")

print(df_data.head())

# The information is by all accounts neatly designed so no requirement for control

#But the way isn't required for information examination so we will evacuate it
```

```
df_data= df_data.drop(columns="Path")

print(df_meta.shape)

print(df_meta.dtypes)

print(df_data.shape)

print(df_data.dtypes)

# Data Analysis

corr1 = df_meta.corr()

corr1 = (corr1)

sns.heatmap(corr1,

xticklabels=corr1.columns.values,

yticklabels=corr1.columns.values)

corr1

# Drawing distribution graphs between classid. We can see how all the classes are distributed
in the data.
fig,ax = plt.subplots(figsize=(15,5))
ax = sns.countplot(df_data['ClassId'])
plt.show()

# From the figure we can tell what number of various signs are there in database and what
number of signs are a similar sign however with various classes.Sign ID (by Ukrainian Traffic
Rule)

fig,ax = plt.subplots(figsize=(15,5))

hatchet = sns.countplot(df_meta['SignId'])

plt.show()

f,ax = plt.subplots(figsize=(15,6))
```

```
hatchet = sns.scatterplot(x='SignId',y='ClassId',data=df_meta)

plt.show()

# We can perceive how tallness and width shifts between classes

hatchet = sns.lineplot(x="ClassId", y="Height", data=df_data)

hatchet = sns.lineplot(x="ClassId", y="Width", data=df_data)

sns.pairplot(df_data)

df_meta.describe()

df_data.describe()

# Data Preprocessing and KNN Implementation

#reading the quantity of classes from meta record

meta_data = pd.read_csv('Meta.csv')

meta_shape = meta_data.shape

no_classes = meta_shape[0]

# resizing picture to 20x20 size. likewise grayscaling the picture

# changing over all pictures to np exhibit and including marks

# took some assistance from web on the most proficient method to change over picture
informational collection into numpy cluster. As it was something new for me

# Take in note that you need to give the way in which your preparation information lies on way
factor underneath

# My information was in AML venture record in work area that is the reason I have given that
way

import cv2 # going to utilize cv2 as its quicker to resize and dim scale picture with it.

import os
```

```
train_data=[]

train_labels=[]

side = 20

for c in range(no_classes) :

way = "/Users/client/Desktop/AML Project/train/{0}/".format(c) # ensure you give the right
root way

documents = os.listdir(path)

for document in records:

train_image = cv2.imread(path+file)

image_resized = cv2.resize(train_image, (side, side), interpolation = cv2.INTER_AREA) #resizing
picture to 20x20 here

dim = cv2.cvtColor(image_resized, cv2.COLOR_BGR2GRAY) #Grayscale picture

train_data.append(np.array(gray))

train_labels.append(c)

information = np.array(train_data)

information = data.reshape((data.shape[0], 20*20))

data_scaled = data.astype(float)/255 # Normalizing information by partitioning with 255. so
information is somewhere in the range of 0 and 1

names = np.array(train_labels)

le = LabelEncoder()

names = le.fit_transform(labels) #adding marks to information

X_train, X_val, y_train, y_val = train_test_split(data_scaled, names, test_size=0.25,
random_state=42) # splitting information into test and preparing. 25% test information

from sklearn.neighbors import KNeighborsClassifier
```

```
model = KNeighborsClassifier(n_neighbors=3) #just default settings

model.fit(X_train, y_train)

# Since we need to process a huge number of pictures, this progression takes some time.
Around 4-5 minutes

y_pred = model.predict(X_val)

# Same for this step,it requires a significant stretch of time to finish because of huge dataset.
Around 4-5 mins.

model.score(X_val, y_val)

# we can see we got 87% accuracy,atleast in my PC.

#I tried it with n_neighbors=5 and the exactness appeared to drop to 83%

# ### All models were tested Comparision between Models

# Accuracies are as follows:
# 1. Decition trees:
# entropy:0.7870039783739672
# gini: 0.7749668468836071
# with max depth=20:0.7428338263796797
# 2. Logistic Regression:
# Accuracy: 0.8894783377541998
# 3. Support Vector Machines
# Mean SVM Cross-validation Score: 0.9304895438545735
# 4. Random Forest:
# Mean RF Cross-validation Score: 0.9072640638631736
# 5. Ensemble Learning:
# KNearest Neighbours : 0.839
# Classification Tree : 0.775
# Voting Classifier: 0.911
# Logistic Regression : 0.865
# 6. Bagging Method:
# Accuracy of Bagging Classifier: 0.137
# 7. Boosting Method:
# Test set accuracy: 0.130
# OOB accuracy: 0.130
# 8. KNN:
# Accuracy score:0.8695297357951648
```


From the above results it is evident that SVM has the most accurate results on this data set on second ensembling on the third random forest and so on. The least accurate result with a lot of error is boosting and bagging methods.

A-3 Python Code (Object Detection):

```
# Object Detection

## Setup
"""

import tensorflow as tf
import matplotlib.pyplot as plt
import tensorflow_hub as hub

import tempfile
from six.moves.urllib.request import urlopen
from six import BytesIO
import time

import numpy as np
from PIL import Image
from PIL import ImageColor
from PIL import ImageDraw
from PIL import ImageFont
from PIL import ImageOps

def show_image(image1):
    fig = plt.figure(figsize=(20, 15))
    plt.grid(False)
    plt.imshow(image1)

def resize_download_image(web_image, new_width=256, new_height=256,
                           display=False):
    _, filename = tempfile.mkstemp(suffix=".jpg")
    response = urlopen(web_image)
```

```
data_of_image = response.read()
data_of_image = BytesIO(data_of_image)
image_pils = Image.open(data_of_image)
image_pils = ImageOps.fit(image_pils, (new_width, new_height), Image.ANTIALIAS)
image_pils_rgb = image_pils.convert("RGB")
image_pils_rgb.save(filename, format="JPEG", quality=90)
print("Image downloaded to %s." % filename)
if display:
    show_image(image_pils)
return filename

def image_bounding_box(image2,
                        ymin1,
                        xmin1,
                        ymax1,
                        xmax1,
                        color,
                        font,
                        thickness=4,
                        display_str_list1=()):
    draw = ImageDraw.Draw(image2)
    im_width1, im_height1 = image2.size
    (left_side, right_side, top_side, bottom_side) = (xmin1 * im_width1, xmax1 * im_width1,
                                                       ymin1 * im_height1, ymax1 * im_height1)
    draw.line([(left_side, top_side), (left_side, bottom_side), (right_side, bottom_side),
               (right_side, top_side),
               (left_side, top_side)],
              width=thickness,
              fill=color)

    display_str_heights12 = [font.getsize(ds)[1] for ds in display_str_list1]

    total_display_str_height = (1 + 2 * 0.05) * sum(display_str_heights12)

    if top_side > total_display_str_height:
        text_bottom_side = top_side
    else:
        text_bottom_side = top_side + total_display_str_height

    for display_str in display_str_list1[::-1]:
        text_width, text_height = font.getsize(display_str)
        margin = np.ceil(0.05 * text_height)
```

```
draw.rectangle([(left_side, text_bottom_side - text_height - 2 * margin),
               (left_side + text_width, text_bottom_side)],
               fill=color)
draw.text((left_side + margin, text_bottom_side - text_height - margin),
         display_str,
         fill="black",
         font=font)
text_bottom_side -= text_height - 2 * margin

def box_draw(image1, boxes1, class_names1, scores1, max_boxes1=10, min_score1=0.1):

    colr = list(ImageColor.colormap.values())

    try:
        font = ImageFont.truetype("/usr/share/fonts/truetype/liberation/LiberationSansNarrow-
Regular.ttf",
                                25)
    except IOError:
        print("Font not found, using default font.")
        font = ImageFont.load_default()

    for i in range(min(boxes1.shape[0], max_boxes1)):
        if scores1[i] >= min_score1:
            ymin1, xmin1, ymax1, xmax1 = tuple(boxes1[i])
            display_str = "{}: {}".format(class_names1[i].decode("ascii"),
                                         int(100 * scores1[i]))
            color1 = colr[hash(class_names1[i]) % len(colr)]
            image_pil = Image.fromarray(np.uint8(image1)).convert("RGB")
            image_bounding_box(
                image_pil,
                ymin1,
                xmin1,
                ymax1,
                xmax1,
                color1,
                font,
                display_str_list1=[display_str])
            np.copyto(image1, np.array(image_pil))
    return image1

image_web_image = "https://imagevars.gulfnews.com/2020/01/11/191101-sheikh-zayed-
road_16f944d7361_original-ratio.jpg" #@param
downloaded_image_path = resize_download_image(image_web_image, 1280, 856, True)
```

```
module_transfer =
"https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1"  #@param
["https://tfhub.dev/google/openimages_v4/ssd/mobilenet_v2/1",
"https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1"]

Image_classifier_detect = hub.load(module_transfer).signatures['default']

def image_load(path1):
    img1 = tf.io.read_file(path1)
    img1 = tf.image.decode_jpeg(img1, channels=3)
    return img1

def detector_func(Image_classifier_detect, path1):
    img = image_load(path1)

    converted_img = tf.image.convert_image_dtype(img, tf.float32)[tf.newaxis, ...]
    start_time1 = time.time()
    res = Image_classifier_detect(converted_img)
    end_time = time.time()

    res = {key:value.numpy() for key,value in res.items()}

    print("Found %d objects." % len(res["detection_scores"]))
    print("Inference time: ", end_time-start_time1)

    image_with_boxes = box_draw(
        img.numpy(), res["detection_boxes"],
        res["detection_class_entities"], res["detection_scores"])

    show_image(image_with_boxes)

detector_func(Image_classifier_detect, downloaded_image_path)
```

Bibliography

- [1] A. Mogaveera, R. Giri, M. Mahadik and A. Patil, "Self Driving Robot using Neural Network," *Self Driving Robot using Neural Network*, 2018.
- [2] A. K. Jain, "Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino," *Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino*, 2018.
- [3] M. S. M. Hajer Omrane, "Neural controller of autonomous driving mobile robot by an embedded camera," *Neural controller of autonomous driving mobile robot by an embedded camera*, 2018.