AUTONOMOUS LANE FOLLOWER

Project report submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Bachelor of Technology
In
Electrical and Electronics Engineering

by
Shubhanshu Gupta (BT15EEE094)
Pranay Pourkar (BT15EEE061)
Piyush Dudurkar (BT15EEE059)
Tushar Umredkar (BT15EEE087)
Ailneni Rakshitha (BT15EEE007)

under the guidance of
Dr. Pradyumn Chaturvedi

Department of Electrical and Electronics Engineering
Visvesvaraya National Institute of Technology
Nagpur 440010 (India)

2019

AUTONOMOUS LANE FOLLOWER

Project report submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Bachelor of Technology
In
Electrical and Electronics Engineering

by
Shubhanshu Gupta (BT15EEE094)
Pranay Pourkar (BT15EEE061)
Piyush Dudurkar (BT15EEE059)
Tushar Umredkar (BT15EEE087)
Ailneni Rakshitha (BT15EEE007)

under the guidance of
Dr. Pradyumn Chaturvedi

Department of Electrical and Electronics Engineering
Visvesvaraya National Institute of Technology
Nagpur 440010 (India)

2019

Department of Electrical Engineering

Visvesvaraya National Institute of Technology, Nagpur

Declaration

We, **Shubhanshu Gupta, Pranay Pourkar, Piyush Dudurkar, Tushar Umredkar, Ailneni Rakshitha** hereby declare that this project work titled **"Autonomous Lane Follower"** is carried out by us in the Department of Electrical and Electronics Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier whole or in part for the award of any degree/diploma at this or any other Institution University.

S. No.	Enrollment No.	Names	Signature
1	BT15EEE094	Shubhanshu Gupta	
2	BT15EEE061	Pranay Pourkar	
3	BT15EEE059	Piyush Dudurkar	
4	BT15EEE087	Tushar Umredkar	
5	BT15EEE007	Ailneni Rakshitha	

Date:

Certificate

This is to certify that the project titled **"Autonomous Lane Follower"**, submitted by Name of the students in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Engineering/Bachelor of Architecture, VNIT Nagpur. The work is comprehensive, complete and fit for final evaluation.

Dr. Pradyumn Chaturvedi
Assistant Professor
Electrical and Electronics Engg.
VNIT, Nagpur

Dr. Madhuri A. Chaudhari
Head
Dept. of Electrical and Electronics Engg.
VNIT, Nagpur

Date:

ACKNOWLEDGEMENTS

First and foremost, we would like to express our deepest appreciation to our project guide, **Dr. Pradyumn Chaturvedi**, Department of Electrical Engineering, VNIT, Nagpur for his constant guidance and motivation throughout the project.

We would also like to thank **Dr. M. A. Chaudhari**, Head of the Department, Electrical Engineering for giving us this opportunity and providing us with proper resources.

Furthur, we would like to thank **Dr. Shital S. Chiddarwar**, faculty in-charge of **IvLabs**, for providing us all the necessary resources.

We finally wish to thank our families for their constant support and encouragement which has been a motivating factor throughout the project.

Shubhanshu Gupta
Pranay Pourkar
Piyush Dudurkar
Tushar Umredkar
Ailneni Rakshitha

ABSTRACT

Self driving car is one of the most trending and innovative technology that is becoming prominent. The automotive industry is adopting self-driving features as the next evolutionary step in automobile technology. It is expected that this technology, after fully developed, will improve traffic management, increase safety on roads and reduce pollution. We wish to develop a small prototype of the Self-Driving car which will introduce the Level-2 type Self-Driving technology to us. In this report, we introduce and analyze the application of image processing and Artificial Intelligent based techniques on practical problems in detection of road lanes for self driving vehicle. Our main goal is to study and develop an algorithm using two different approaches, i.e. Image Processing and Learning based technique to detect road lanes in simulation as well as make an RC car to follow it in real-time.

List of Figures

List of Tables

List of Acronyms

1. AI - Artificial Intelligence
2. BLDC - Brushless DC Motors
3. CMU - Carnegie Mellon University
4. CNN - Convolutional Neural Network
5. Conv - Convolution
6. csv - Comma-separated values(File format)
7. DC - Direct Current
8. ELU - Exponential linear unit
9. ESC - Electronic Speed Controller
10. FC - Fully Connected
11. FoV - Field of View
12. fps - frames per second
13. GPIO - General Purpose Input Output
14. GPS - Global Positioning System
15. GPU - Graphics Processing Unit
16. HSV - Hue, Saturation, Value
17. LiPo - Lithium polymer
18. MSE - Mean Squared Error
19. OpenCV - Open Source Computer Vision
20. PWM - Pulse Width Modulation
21. RC - Radio-Controlled Car
22. RGB - Red, Green, and Blue
23. R-pi - Raspberry Pi
24. SUV - Sport Utility Vehicle

Contents

Chapter 1

INTRODUCTION

1.1 What is an Autonomous Lane Follower?

A self-driving car (also referred as an autonomous car or driver-less car) is a vehicle that uses a combination of sensors, cameras, RADAR and artificial intelligence (AI) to travel between locations without a human operator. To qualify as fully autonomous, a vehicle must be able to navigate without human intervention to a predetermined destination over roads that have not been adapted for its use.

1.1.1 Levels of Automation

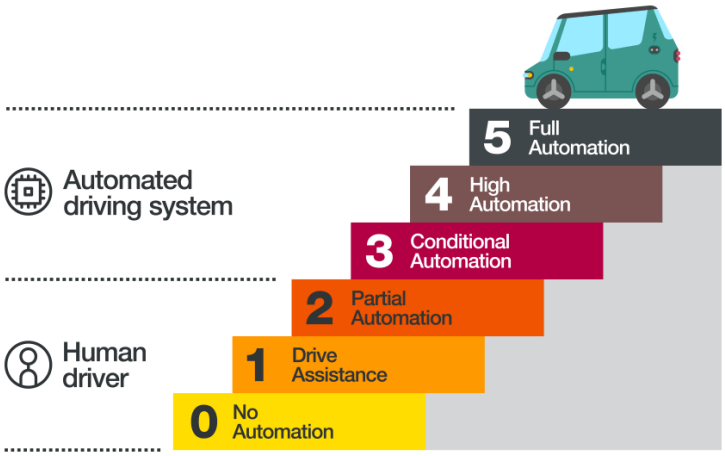


Fig. 1.1: Levels of Autonomy

There are certain levels defined in building a Self-Driving Car depending on the automated driving task in different driving scenarios[1]. These levels are :
Level 0: It involves detecting unusual performance and warning the driver about the same but has no control over the vehicle.
Level 1: Cars of this level provide at least one automated task for the driver assistance and improve safety but it cannot take over the steering. Ex: Automatic application of brakes if a pedestrian is seen crossing the road.
Level 2: At this level, cars are able to take over the steering and control multiple

tasks such as acceleration, braking, parking, etc.
Level 3: Cars at this level are capable of taking the control over the vehicle in certain conditions but the driver needs to be ready to take over the control at any time if needed.
Level 4: At this level cars become fully autonomous and driver need not to be attendant. Warnings and alarms are taken care of by the cars itself. But, the vehicle may have the geographical and speed constraints.
Level 5: At this level true autonomy is granted to the vehicle and the presence of driver is not needed.
As of 2018, many of the technologists are struggling to complete level 2 while some have made it to the level 3. The fully autonomous car is still the future and need many development.

1.2 Previous Work

Experiments have been conducted on automated driving systems since at least the 1920s;trials began in the 1950s.The first truly automated car was developed in 1977 by Japan's Tsukuba Mechanical Engineering Laboratory. The vehicle tracked white street markers, which were interpreted by two cameras on the vehicle, using an analog computer for signal processing. The vehicle reached speeds up to 30 kilometres per hour with the support of an elevated rail.

Autonomous prototype cars appeared in the 1980s, with Carnegie Mellon University's Navlab and ALV projects funded by DARPA starting in 1984, Mercedes-Benz and Bundeswehr University Munich's EUREKA Prometheus Project in 1987. By 1985, the ALV had demonstrated self-driving speeds on two-lane roads of 31 kilometres per hour with obstacle avoidance added in 1986 and off-road driving in day and night time conditions by 1987. From the 1960s, through the second DARPA Grand Challenge in 2005, automated vehicle research in the U.S. was primarily funded by DARPA, the US Army and the U.S. Navy, yielding incremental advances in speeds, driving competence in more complex conditions, controls and sensor systems.Companies and research organizations have developed prototypes. Partly funded by the National Automated Highway System and DARPA, the Carnegie Mellon University's Navlab drove 4,584 kilometres across America in 1995, 98% of it autonomously. Navlab's record achievement stood unmatched for two decades until 2015 when Delphi improved it by piloting an Audi, augmented with Delphi technology, over 5,472 kilometres through 15 states while remaining in self-driving mode 99% of the time. In 2015, the US states of Nevada, Florida, California, Virginia and Michigan together with Washington allowed the testing of automated cars on public roads.

In 2017, Audi stated that its latest A8 would be automated at speeds of up to 60 kilometres per hour using its "Audi AI." The driver would not have to do safety checks such as frequently gripping the steering wheel. The Audi A8 was claimed to be the first production car to reach level 3 automated driving and Audi would be the first manufacturer to use laser scanners in addition to cameras and ultrasonic sensors for their system. In November 2017, Waymo announced that it had begun testing driver-less cars without a safety driver in the driver position;however, there was still an employee in the car. In October 2018, Waymo announced that its test vehicles had traveled in automated mode for over 10,000,000 miles, increasing by about 1,000,000 miles per month. In December 2018, Waymo was the first to commercialize a fully autonomous taxi service in the U.S [2] [3].

1.2.1 Companies Working on Autonomous Vehicles

CMU Navlab

CMU Navlab group builds computer-controlled systems for automated and assisted driving. Since 1984, they have built a series of robo cars, vans, SUVs and buses. The latest is the Navlab 11,a robot jeep Wrangler equipped with wide variety of sensors.

Google Cars

Google cars have sensors designed to sense objects as far as two football fields away in all directions, including pedestrians, cyclists and vehicles.

Tesla

Tesla Model S with hardware to allow for the incremental introduction of self-driving technology a forward RADAR, forward-looking camera, 12 long-range ultrasonic sensors positioned to sense 16 feet around the car in every direction at all speeds and a high precision digitally-controlled electric assist braking system. This combined suite of features represents the only fully integrated autopilot system involving four different feedback modules:camera, RADAR, ultrasonics and GPS.

Mercedes-Benz

The 2014 Mercedes S-class has options for autonomous steering, lane keeping, acceleration/braking, parking, accident avoidance and driver fatigue detection. Partially automated driving is available to drivers of new Mercedes-Benz E and S-class models. The DISTRONIC PLUS with steering assist and StopNGo pilot is capable of steering the vehicle mainly autonomously through traffic jams.

Flux Auto

It aims to facilitate features like cruise control, lane keeping, collision avoidance, keeping a check on accidents, higher efficiency, amongst others in a much cost effective manner. The 16-member startup is developing the autonomous system without using LIDAR—an incredibly must-have component of most autonomous vehicles. Instead their vision algorithms can be used with any cheap cameras with a much quicker response rate.

Swaayatt Robots

Swaayatt robots is developing on-and off roads self driving technology that works in extremely difficult traffic scenarios and in unstructured environmental conditions such as in India. The technology by this startup enables self driving vehicles to perceive their environments using off-the-shelf cameras. The navigation module by Swaayatt is designed in such a way so as to handle previously unseen environments with much ease. Working on GPS denied environments, it can produce high speed trajectories even in cluttered environments [4].

1.3 Applications

Automated Trucks

Several companies are said to be testing automated technology in semi trucks. Otto, a self-driving trucking company that was acquired by Uber in August 2016, demonstrated their trucks on the highway before being acquired. Waymo has also said to be testing automated technology in trucks. In 2016, Anheuser-Busch Inc. and Uber Technologies Inc. joined together and successfully made the first commercial delivery of beer using a self-driving truck. No human action had involved in the driving process and the truck travelled 120 miles.

Transport Systems

In Europe, cities in Belgium, France, Italy and the UK are planning to operate transport systems for automated cars and Germany, the Netherlands, and Spain have allowed public testing in traffic. In China, Baidu and King Long produce automated minibus, a vehicle with 14 seats, but without driving seat. With 100 vehicles produced, 2018 is the first year with commercial automated service in China. Those minibuses should be at level 4, that is driver-less in closed roads [5].

1.3.1 Potential Benefits

- **Safety**:Driving safety experts predict that once driver-less technology has been fully developed, traffic collisions caused by human error, such as delayed reaction time, tailgating, rubbernecking, and other forms of distracted or aggressive driving should be substantially reduced.
- **Welfare**: Automated cars could reduce labor costs; relieve travelers from driving and navigation chores, thereby replacing behind-the-wheel commuting hours with more time for leisure or work. For the young, the elderly, people with disabilities and low-income citizens, automated cars could provide enhanced mobility.
- **Traffic**: Additional advantages could include higher speed limits,smoother rides, increased roadway capacity and minimized traffic congestion,particularly in urban areas, due to decreased need for safety gaps and higher speeds.
- **Energy and environmental impacts**: Vehicle automation can improve fuel economy of the car by optimizing the drive cycle. Reduced traffic congestion and the improvements in traffic flow due to widespread use of automated cars will translate into higher fuel efficiency.Additionally, self-driving cars will be able to accelerate and brake more efficiently.
- **Parking space**: Self-driving cars can park very, very close together. There's no need to leave room for a person exiting the vehicle, thereby reducing the parking space required. Moreover, autonomous vehicles could be used continuously after it has reached its destination. This could dramatically reduce the need for parking space [5].

1.4 Motivation

Autonomous vehicles have made significant advances in recent years due to rapid development of various sensors including RADAR, LIDAR, camera systems and wireless communications. Both car manufacturing and IT companies are competitively investing in self-driving field. The main requirement of a self driving car is to cover long distances in a safer way and at the same time decreasing the rate of accidents and traffic jams, all without human interaction. Studies say that 95% of all the traffic fatalities are caused by human error. Saying that 41% of those traffic fatalities are caused by recognition errors of drivers, what stays for a driver's inattention, distraction either external or internal and inadequate surveillance. Autonomous vehicles don't encounter this kind of problems, so fort they could prevent this kind of fatalities. Also, best part of autonomous cars is to serve the applications for military purpose where the situation is inconvenient, dangerous or impossible to have human operator present. Our project consists of deriving the steering angle of a remote control car using two cutting-edge technologies like image processing and machine learning. This process consists of collecting data, training and testing of our CNN model on collected data. The accuracy could have been increased by using the other sensors, providing a better training dataset and pre-processing the images, say for instance using OpenCV to reduce light reflection and colouring the lanes. This prototype at least achieved its objective of showing the strength and applications of convolutional neural networks. In a real life situation, an autonomous car needs a lot more sensors and a way to store and transport the massive amount of data generated. Collecting enough training data for safe driving is also an issue. Whether or not today's technology is enough to find cheap solutions to these problems while increasing safety is a topic for debate. The only way to find out though is by trying to solve them. So, this our first step towards making a full fledged autonomous vehicle with various features like object detection, passenger mood detection, which could run in complex traffic situations and in extreme environment conditions like rain, fog, etc [5].

1.5 Thesis Outline

This paper explains in detail various steps involved in the process of making an autonomous lane follower. Firstly, the hardware requirements for building the prototype like raspberry-pi, test vehicle and other electronic components are described along with their specifications. Several parts like camera stand are designed to accommodate components like camera in the existing test vehicle, so the design aspects of such parts are specified.

Also, design and dimensions of lane are specified along with the images of different shapes of the lane. Control of BLDC motor and servo control are explained. Two Different approaches used are as follows:

- **Approach 1 (using learning technique):**
  Once the assembly of parts and installation of software and libraries like OpenCV (Open Source Computer Vision), tensorflow (open source machine learning library), GPIO(General Purpose Input/Output library) of raspberry-pi is done, the learning part is implemented. A neural network is formed and trained on the dataset of lane images labelled with corresponding steering angle. There are different dataset available, provided by different laboratories and companies, which can be used or we can manually collect the dataset, which is the case here. This dataset is split into three parts:
  Training Set: Part of the dataset used to train the model
  Validation Set: Part of the dateset to evaluate the model
  Test Set: This provides the final evaluation of the trained model

  After dataset collection, dataset cleaning is done where blurred images etc are deleted and the image is cropped appropriately. Then the model is trained and tested accordingly.

- **Approach 2 (using image processing):**
  In this process several image processing techniques are used to extract useful information from the image. Firstly, different types of distortions are explained in detail. Camera calibration algorithm is implemented and the distortion coefficients are obtained. Perspective transformation is then applied on the image to obtain top view perspective of the image. Color space of the image is changed from RGB to LAB and then image segmentation is used to isolate yellow lanes from the background. Canny edge detection algorithm is used on the obtained binary image and the output of this algorithm is given as input to hough line transform for detecting straight lines. Histogram calculations are then done. Sliding window method is used for finding lane position in a image in which next position of window is calculated by the mean of non-zero pixel intensity coordinates found in the last window region.From the last step, we have a vector having non-zero pixel intensity coordinates for both left and right half separately. This vector acts as input to the polynomial fit method in OpenCV which helps in obtaining 2-Degree polynomial coefficients. From this, the radius of curvature of the road will be estimated.

Chapter 2

COMPONENTS AND HARDWARE ASSEMBLY

2.1 Components and Specification

2.1.1 Raspberry Pi 3

Fig. 2.1: Raspberry Pi 3

Raspberry Pi 3 is a small sized computer which has input ports for the connection of the peripheral devices such as mouse, keyboard, monitor and by inserting an SD card with the installed OS, it can be operated as computer[6][7][8].

Specification:

- It has 1.4Ghz 64-bit quad-core Broadcom processor.
- It has extended 40 pin GPIO pin header.
- It needs 5V/2.5A DC supply.
- 0–50° C operating temperature.

2.1.2 BLDC Motor

Fig. 2.2: BLDC Motor

Brushless DC motor(BLDC) is a compact sized motor which provides high speed and better speed control. It is used as an actuator for the test vehicle and a controlled PWM signal is provided for the working [9].

Specification:

- High speed upto 1000Kv (1000RPM/V).
- Maximum Efficiency of 80%.
- No load current of 0.5A at 10V.
- It requires an on-board ESC for speed control.

2.1.3 LiPo Battery

LiPo battery is a rechargeable Lithium Polymer battery which provides high storing capacity with small compact size. LiPo batteries are more reliable, can provide high value current for the components and are cheaper. It is used to provide power to whole setup[10].

Fig. 2.3: LiPo battery

Specification:

- It has maximum storage capacity of 2200mAh.
- It has 3 cells in series with nominal voltage of 3.7V each which makes nominal voltage of the battery equal to 11.1V.
- It has 25C rating which determines the maximum constant current for which the battery is used. In this case, maximum current is 55A.

2.1.4 Electronic Speed Controller(ESC)

Fig. 2.4: ESC

An electronic speed control or ESC is an electronic circuit that controls and regulates the speed of an electric motor especially BLDC motor.It has 3 sets of wires, one set is used to connect to battery, second set is for getting the signal and third set is used to connect to the motor. ESC receives signal from the controller, in this case R-Pi, and generates pwm for the motor to achieve desired speed[11].

Specification

- It takes input from 6-16.8V DC.
- It has 30A current rating.
- It comes with BEC(Battery Eliminator Circuit) which provides 5V output for the servo motor.

2.1.5 Servo Motor

Servo motor is a motor which rotates to a certain position or angle with great precision. Since the motor used is DC powered, it is known as a DC servo motor. It provides high torque with small and light package which makes it more desirable. In this project, it is used for the control of steering angle of the test vehicle and 30° rotational moment of steering wheels on either side is obtained[12].

Fig. 2.5: Servo Motor

Specification

- Its operating voltage is 4.8-7.2V.
- Its working speed is 0.2s/60° at 4.8V, 0.16s/60° at 6V.
- It has a stall torque of 8.5 kgf.cm at 4.8V, 10kgf.cm at 6V.

2.1.6 Buck Converter

Fig. 2.6: Buck converter

DC-DC buck converter converts a high value of the DC voltage to the lower valued DC voltage. In this project, the battery used provides 11.1V but the R-Pi needs only 5V supply otherwise it will burn out. Hence, the buck converter is used. The buck converter is provided with adjustable potentio-meter through which we can adjust the output voltage[13].

Specification

- It can take input voltage of 3.2-40V DC and generates output of the range 1.25V-35V DC.
- It provides 3A of maximum current.
- It comes with thermal shutdown and current limit protection.

2.1.7 Camera

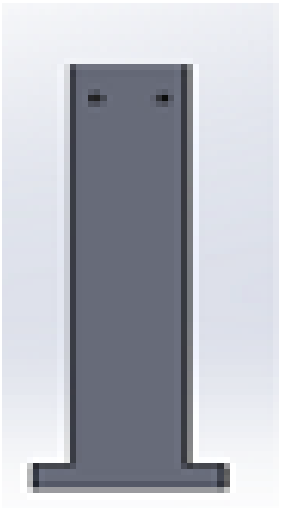
The camera behaves as the eyes of the vehicle. It takes the images of the lane that the test vehicle has to follow and provide them to the trained model to decide the perfect angle for steering[14].

Fig. 2.7: Camera

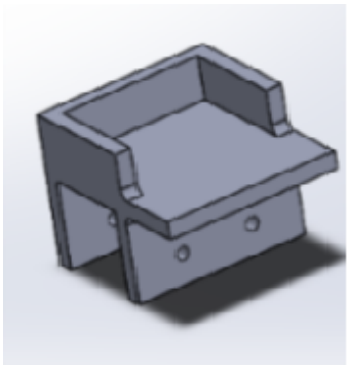
Specification

- It has maximum resolution of 720p/30fps.
- It has 60° field of vision(FoV).

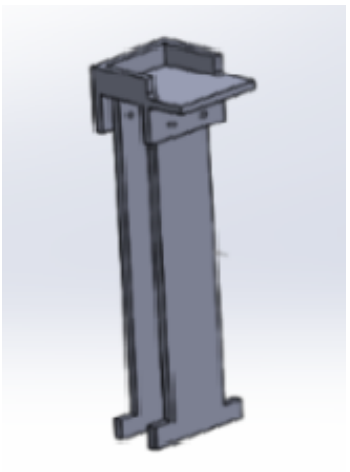
2.1.8 Camera Stand



(a) Strip



(b) Camera base



(c) Final assembly

Fig. 2.8: Camera Stand

To get the proper field of view of the lane, the position at which the camera should be placed was decided and to keep the camera at calculated height, this stand is designed. As shown in fig 2.8b, a slot for the camera is designed so that the camera will fit into it and will not vibrate during the motion of vehicle. Fig 2.8c shows the final assembly which is attached on the vehicle.

2.1.9 Lane Design

As shown in fig 2.9, a chart paper is designed to mimic the real street and number of such papers are connected together to make a desired shape of the street. The lane is designed with the width of 46 cm and the width of the border line (yellow lines) is 3 cm.

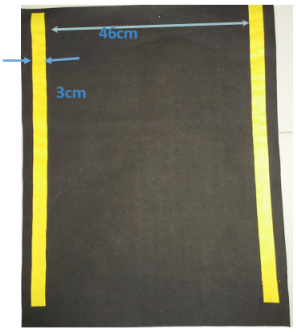


Fig. 2.9: Chart Paper Designed as Lane

By connecting the papers in different fashion, different shapes of the lane can be formed to train the test vehicle. Some of the possible lanes are shown in fig 2.10.

(a) Lane 1

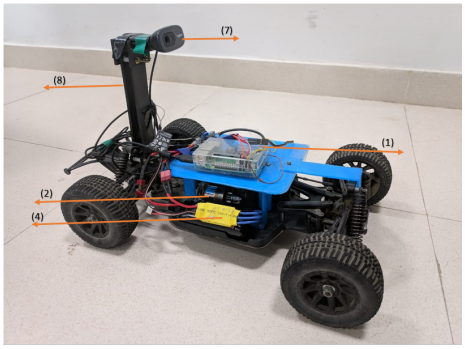
(b) Lane 2

(c) Lane 3

(d) Lane 4

Fig. 2.10: Different Road Lanes Generated

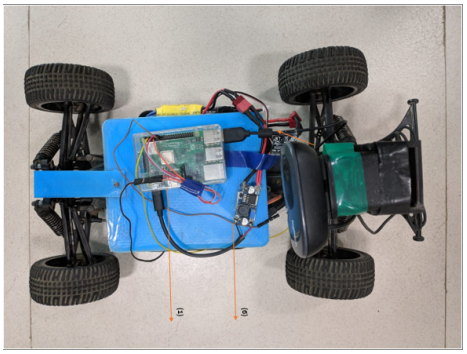
2.2 Assembly



(a) Right Side View 1

(b) Right Side View 2

(c) Left Side View



(d) Top View

Fig. 2.11: Assembly of the Test Vehicle

1. Raspberry Pi
2. BLDC motor
3. LiPo battery
4. ESC
5. Servo Motor
6. Buck Converter
7. Camera
8. Camera Stand

2.3 Control of Components

2.3.1 BLDC Control

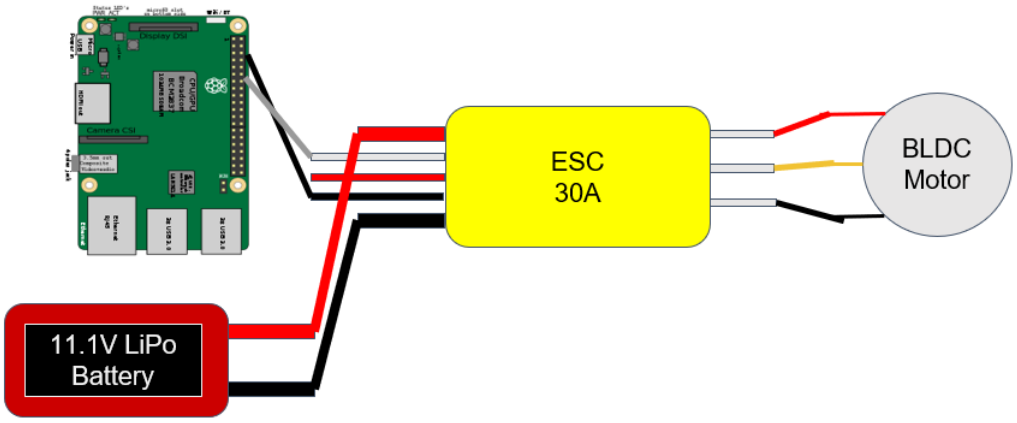


Fig. 2.12: BLDC control

Need of BLDC Motor

In our project BLDC motor is used as forward motion driver. Torque developed by BLDC motors is very high which is important to drive heavy load. Also this motors are available as out runners which increases the thrust of the motors. BLDC motors have smooth vibration less operation which is ideal for less jerking vehicle.

The power to weight ratio of a BLDC motor is very high. This is very important because the motors used driving vehicles should be of high power (high speed and high torque) but should also be of less weight (for max efficiency). A DC motor which could provide the same torque and speed of that of a BLDC motor will be twice as heavy as the BLDC motor[9].

Need of ESC

An electronic speed control or ESC is an electronic circuit that controls and regulates the speed of an electric motor especially BLDC motor. BLDC motors require some kind of controller which covert DC voltage from battery into pulses to power the phase wires of motor. The main task of controller is to energize the phase wires of BLDC motors in an order so that motor rotates. When magnet crosses the coil, coil is energized by controller and back EMF is also sensed simultaneously [11].

PWM based Speed Control

ESC regulates the speed of BLDC motor by sensing the PWM input from microcontroller through its input terminal. As a standard ESC need PWM input of 50Hz with signal ranging from 0.5 to 10. Min speed vehicle need to overcome the load is obtained at signal of 4.7. For our testing and driving purpose we are driving BLDC at signal of 7.35 [11].

2.3.2 Servo Control

Here, servo motor is used for steering mechanism. Control of servo motor is similar as control of BLDC. Operating frequency, voltage and duty cycle at which servo will operate are all given in data sheet. For a particular servo motor duty ratio ranges form 2.5% to 12.5%. As all servo motors can rotate from 0° to 180° [12].

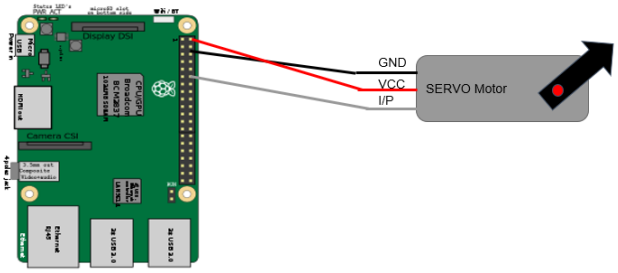


Fig. 2.13: Servo control

Input voltage(+5V) and ground(0V) signal are given to servo directly from R-Pi. Also, PWM signal given to servo motor from R-Pi is of 50Hz with range 5.5 to 8.75 as given vehicle can take max turn of angle 30° (i.e. vehicle can take max turn of 30° in left and right direction) [15].

Transformation of Servo Angle and Duty Ratio

$$\theta = (D - 5.5)\frac{40}{7} + 60 \tag{2.1}$$

Steering Angle= Servo angle - 90

S. No.	Duty Ratio(D)	Servo Angle(θ)	Steering Angle
1	5.5	60	-30
2	5.75	60	-30
3	6	65	-25
4	6.25	70	-20
5	6.5	76	-14
6	6.75	80	-10
7	7	85	-5
8	7.25	90	0
9	7.5	92	2
10	7.75	96	6
11	8	100	10
12	8.25	105	15
13	8.5	110	20
14	8.75	114	24
15	9	118	28

Table 2.1: Relation between Duty Ratio and Steering Angle

2.4 Overall Working of Test Vehicle

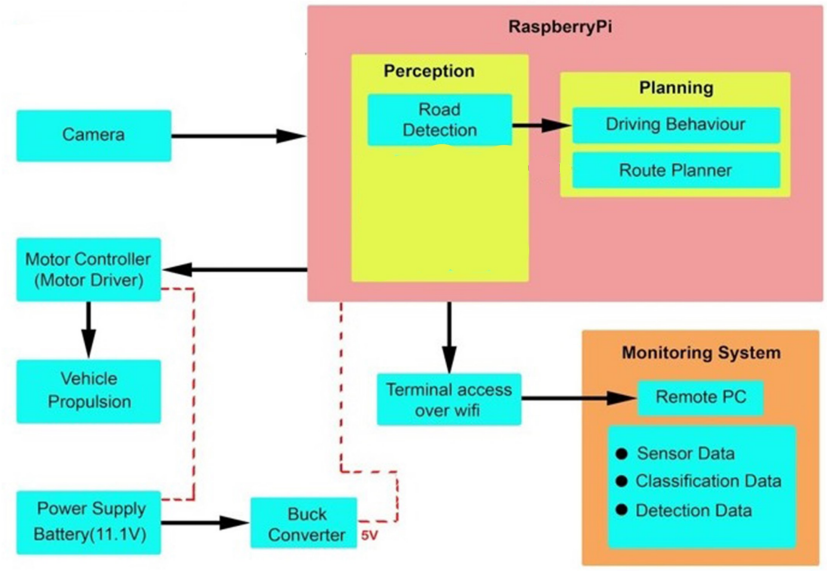


Fig. 2.14: Cyber Physical Architecture

Fig 2.14 shows the overall working of the project. Raspberry Pi, the controller, acts as brain of the vehicle. Camera and other sensors(e.g. ultrasonic sensor) can be attached to the R-Pi to take input of the surrounding of vehicle. This input data is processed in the trained model code built in the R-Pi. During this process, road, obstacles are detected and control planning is done accordingly. Then the generated signals are sent to the BLDC and servo motor which cause the desired movement of the vehicle on the lane. Whole system is powered by the LiPo battery.

Chapter 3

IMAGE PROCESSING BASED APPROACH

Image processing is a technique to convert an image into digital form and perform some operations and manipulate it, in order to get an improved image or to extract some useful information from it. Image Processing finds many application in medical, industrial, Machine/Robot field, etc. Cameras are present on every self-driving car and they are often present in large numbers to get a 360 degree visibility. Cameras are one of the primary tool that vehicle uses to perceive its surrounding environment. They are a key to a variety of essential tasks: lane detection, road curvature estimation, obstacle detection and many more.

Block diagram given below shows a detailed method for finding road lanes and estimation of steering angle and radius of curvature.

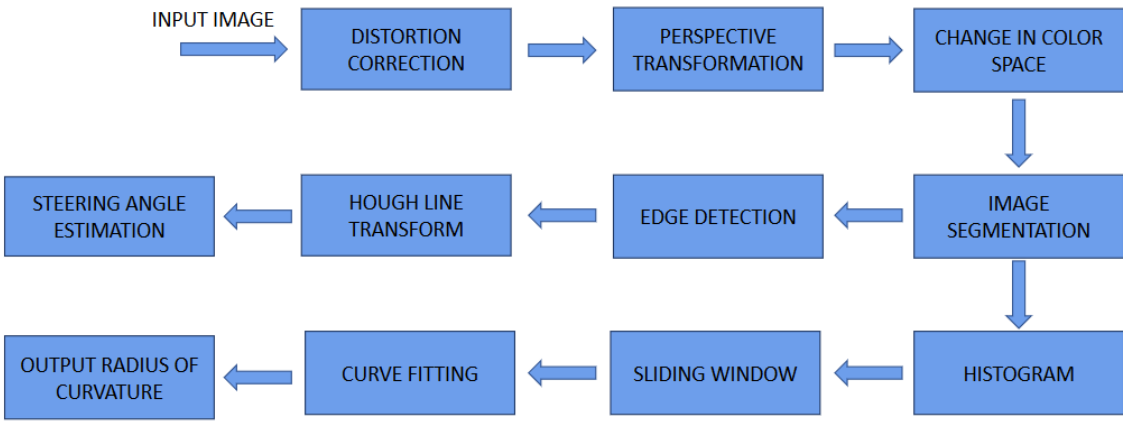


Fig. 3.1: Image Processing Based Approach

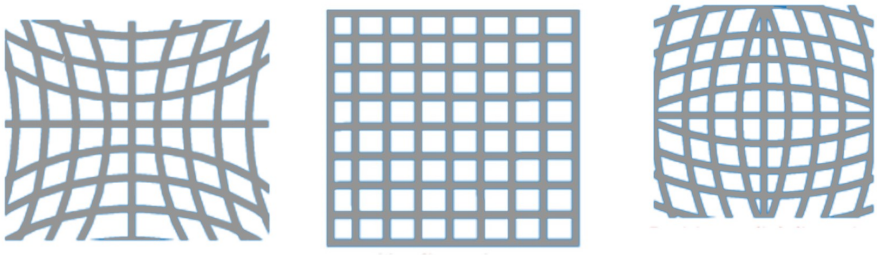
3.1 Image Correction and Preprocessing

3.1.1 Camera Calibration and Distortion Correction

Common pinhole cameras are cheap but they come with a significant distortion. The pinhole camera model does not account for lens distortion because an ideal pinhole camera does not have a lens. However these are constants and by calibrating the camera and by some remapping these distortions can be corrected.
There are two types of major distortion:

1) Radial Distortion

This type of distortion occurs when light rays bend more near the edge of a lens than they bend at its optical center. Because of this straight lines in a image will appear curved.



(a) Negative Radial Distortion (b) No Distortion (c) Positive Radial Distortion

Fig. 3.2: Radial Distortion

The radial distortion coefficients model accounts for this type of distortion and are denoted as

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.1}$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2}$$

$$r^2 = x^2 + y^2 \tag{3.3}$$

where,
x, y - Undistorted points
k_1, k_2, k_3 - Radial distortion coefficients

2) Tangential Distortion

This type of distortion occurs when image taking lens is not aligned perfectly parallel to the imaging plane. So some areas in image may look nearer than expected.

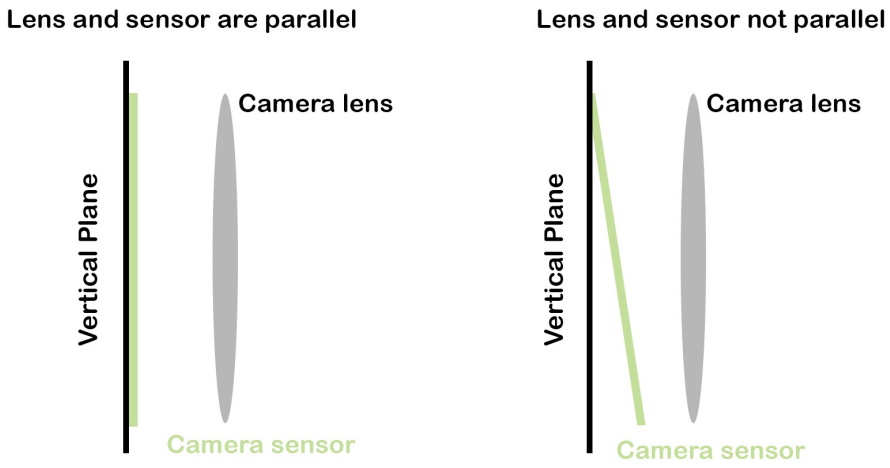


Fig. 3.3: Tangential Distortion

The Tangential distortion coefficients model accounts for this type of distortion and are denoted as

$$x_{distorted} = x + [2p_1xy + p_2(r^2 + 2x^2)] \tag{3.4}$$

$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2xy] \tag{3.5}$$

where,
$x_{distorted}, y_{distorted}$ - Distorted points
x, y - Undistorted points
p_1, p_2 - Tangential distortion coefficients
Distortion coefficients = $(k_1\ k_2\ p_1\ p_2\ k_3)$

Camera Calibration Parameters:
The calibration algorithm calculates the camera matrix. It requires extrinsic and intrinsic parameters for calculating camera matrix. The extrinsic parameters represent a rigid transformation from 3-D world coordinate system to the 3-D camera coordinate system. The intrinsic parameters represent a projective transformation from the 3-D camera coordinates into the 2-D image coordinates.

Extrinsic Parameters:
The extrinsic parameters consist of a rotation, R, and a translation, t. The camera's coordinate system origin is at its optical center and its x-axis and y-axis define the image plane.

Intrinsic Parameters:
The intrinsic parameters include the focal length, the optical center, also known as the principal point, and the skew coefficient. The camera intrinsic matrix, k is defined as,

$$k = \begin{vmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{vmatrix}$$

where,
c_x, c_y - Optical center (the principal point) in pixels
f_x, f_y - Focal length in pixels
The unknown parameters are f_x, f_y, c_x, c_y and the distortion coefficients. The process of determining these unknown parameters is known as calibration. Calculation of these parameters is done through basic geometrical equations. The equations used depend on the chosen calibrating objects. We used classical black-white chessboard, which is supported by OpenCV library, for calibration. Source code for calibration is provided by OpenCV [16]. After calibration we got the following coefficients as

$$k = \begin{vmatrix} 819.752611 & 0.000000 & 329.461908 \\ 0.000000 & 819.011546 & 245.006332 \\ 0.000000 & 0.000000 & 1.000000 \end{vmatrix}$$

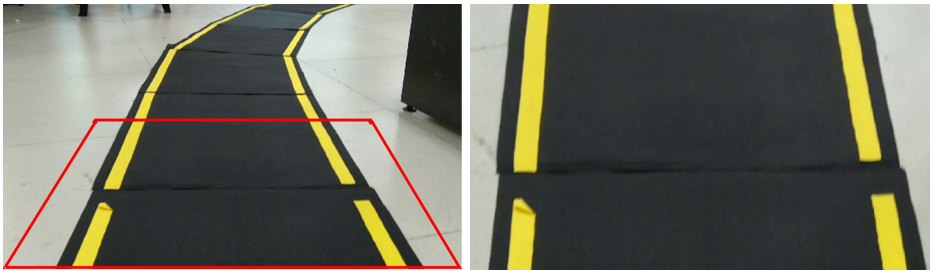
Distortion coefficients = (0.066489 0.117170 0.00383 0.018518 0.000000)

3.1.2 Birds Eye View or Perspective Transformation

Camera fitted on an vehicle are generally front facing. So we are able to get a front view from a vehicle. But this view is not suitable for estimation of steering angle as well as radius of curvature because it does not depict exact angle form by road lanes with the vehicles horizontal axis. Parallel lines appear to converge on image from the front facing camera due to perspective. Birds Eye View transformation technique is a method of generating top view perspective of an image. We choose a suitable trapezoidal area by selecting 4 points from an image. This area is then stretched to form a new image which is rectangular in shape using perspective transformation [17].
In our application, we selected 4 points as,
(0.25*image_width,0.45*image_height),
(0.75*image_width,0.45*image_height),
(0.1*image_width,1*image_height),
(0.9*image_width,1*image_height)

Transformation matrix obtained is

$$k = \begin{vmatrix} 3.928571428571425 & 1.904761904761903 & -2811.428571428569 \\ -3.885780586188048e-16 & 5.71428571428571 & -2777.142857142856 \\ -1.084202172485504e-19 & 0.001984126984126981 & 1 \end{vmatrix}$$

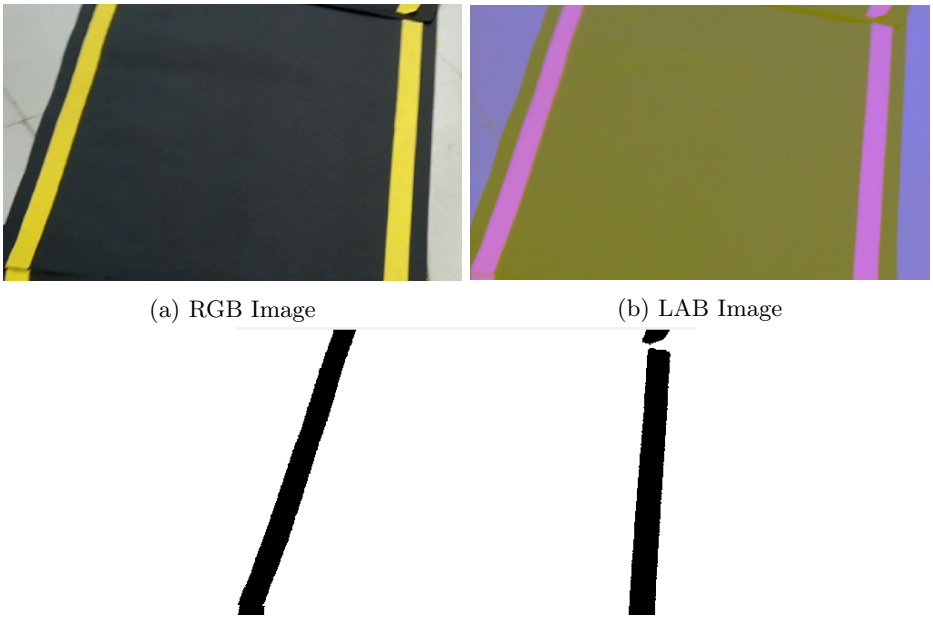


(a) Selection of Trapezoidal region (b) Perspective Transform of image

Fig. 3.4: Perspective Transformation

3.1.3 Color Space and Image Segmentation

Color spaces are different types of color modes used in image processing for various purposes. Some of the common color spaces are: RGB, HSV, LAB, etc. RGB color space is most widely used and RGB stands for red green blue. That is, it consists of 3 channels wherein each channel is 8 bit and has a value in the range (0,255). RGB color space has a non-uniformity in different light condition and hence it make color based segmentation difficult. Therefore it was needed to convert original RGB color space into other. We found LAB color space suitable for our application because it is perceptually uniform color space which approximates how we perceive color. LAB color space has three components, L stands for Lightness(Intensity), A for color component ranging from Green to Magenta and B for color component ranging from Blue to Yellow. Because of this separating yellow lanes from other area becomes more accurate. Image segmentation is a process of Isolating required part from a image by converting, image into binary image. We convert the pixels having values between (0,180,0) and (255,255,154) to 255 and remaining pixel value are set to 0 in a image to get yellow lanes separated from background.



(a) RGB Image (b) LAB Image

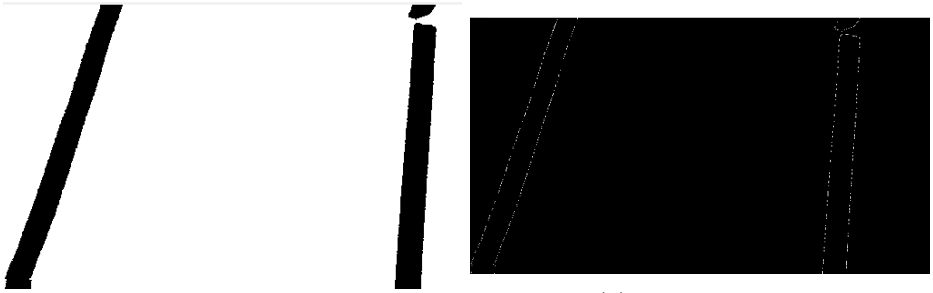
(c) Segmented Image

Fig. 3.5: Color Space and Image Segmentation

3.2 Estimation of Steering Angle

3.2.1 Canny Edge Detection

The canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. This algorithm can be applied on single channel grey image or binary image and it composed of 5 steps: Noise reduction, Gradient calculation, Non-maximum suppression, Double threshold, Edge Tracking by Hysteresis. OpenCV has inbuilt method for applying this algorithm on an image. The output of this method is the edges detected which is a input to the next step which is hough transform [18].



(a) Segmented Image

(b) Edges Detected

Fig. 3.6: Canny Edge Detection

3.2.2 Hough Line Transform

The Hough line transform is a feature extraction technique used in image analysis, image processing and computer vision to detect straight lines. To apply this transform it is required to get edge detection preprocessing done on image before. A lines is a collection of points. A line in the image space can be represented with two variables(r, θ) in Polar coordinate system as

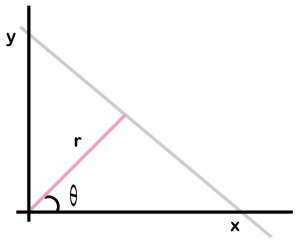
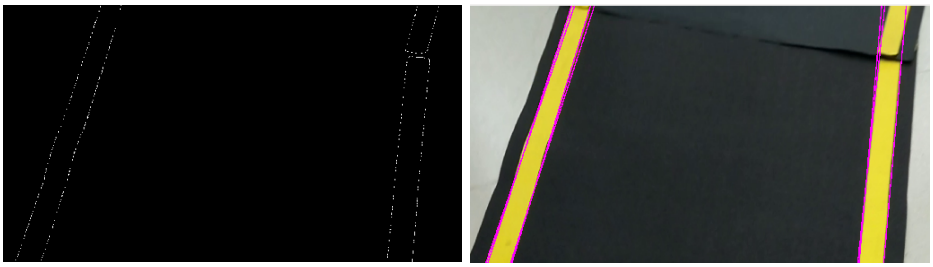


Fig. 3.7: Straight Line Graph

For Hough Transforms, lines are expressed in the Polar system. Hence, a line equation can be written as:

$$r = x\cos\theta + y\sin\theta \tag{3.6}$$

We plot the family of lines that goes through a given (x,y) point and if the curves of two different points intersect in the plane θ - r, that means that both points belong to a same line. Parameters of line detected (r,θ) are stored in a vector [19].



(a) Edges Detected (b) Hough Lines (Pink in color)

Fig. 3.8: Hough Line Transform

When the lanes are present in 1^{st} quadrant, we get the angle in range 0° to 90°. When the lanes are present in 2^{nd} quadrant, we get the angle in range 90° to 180°. We convert the angle obtained in such a way to get values in between -90° to 90° and then took the average of all the angle of lines detected in a single image to get steering angle.

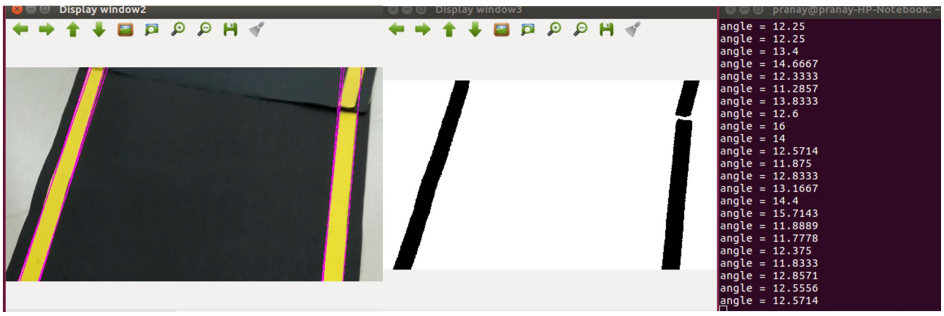


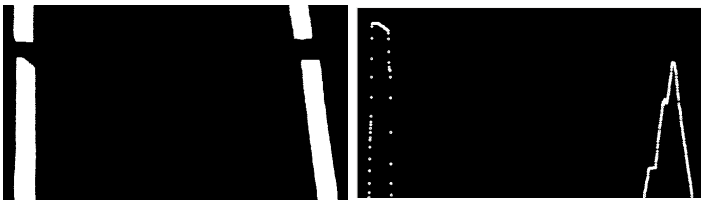
Fig. 3.9: Lanes in First Quadrant

Fig. 3.10: Lanes in Second Quadrant

3.3 Estimation of Radius of Curvature

3.3.1 Histogram Calculation

A histogram is a plot that show the underlying frequency distribution of a set of data. We took a vector of size equal to image width and store in it the sum of pixel intensity values column-wise i.e. we carry-out the sum of all pixel values in a column and store it in a vector having index values between 0 to (image_width - 1). We also note down the index position in the vector which has highest frequency in left half as well as right half of image [20].



(a) Thresholded Image (b) Histogram Plot

Fig. 3.11: Histogram Calculation

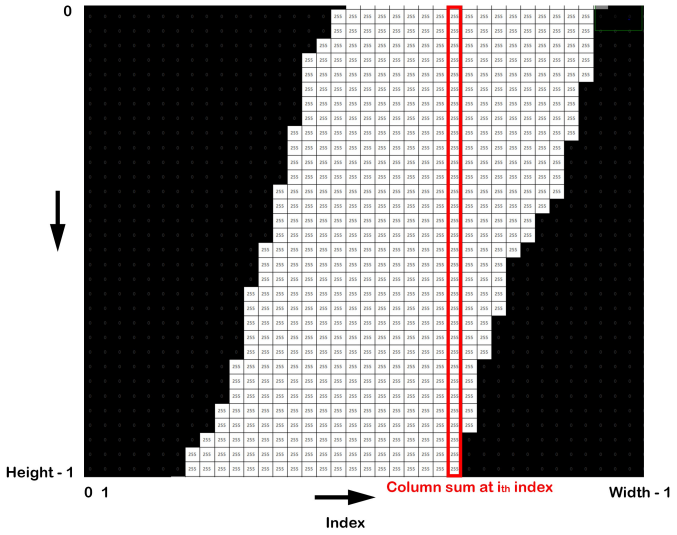


Fig. 3.12: Maximize view of a region in Thresholded Image

3.3.2 Sliding Window Method for Finding Lane Position in an Image

A sliding window is a rectangular region of fixed width and height that slides across an image and extract useful information from it. For each of these windows, we would normally take the window region and search for an object that interests us. We assume the number of windows to be 9 i.e. the image can accommodate 9 windows vertically. So the window height can be calculated by dividing image height by number of windows. Width of the window is assumed to be 300px wide depending upon the resolution of camera. In our case the resolution of camera is 1280 by 720 pixels. Sliding window method is applied on left half of image as well as right of image separately. Starting position coordinate of left window and right window is decided by the index position of vector which has highest frequency in left and right half respectively during histogram calculation. Coordinates which are obtained are checked whether they are in valid range (resolution). Positional coordinates of non-zero pixel in left and right window region are stored in the vector separately and these coordinates are mapped with the original image because size of window and size of original image are different. Next position(x,y) of window is calculated by the mean of non-zero pixel intensity coordinates found in the last window region. This process continues till the window slides across entire height [21].

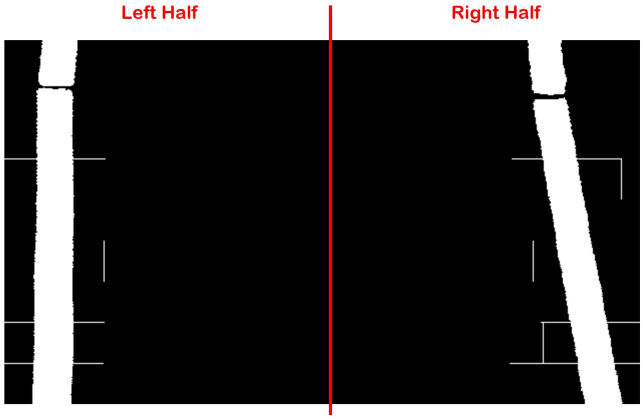


Fig. 3.13: Lane Tracking using left and right window

3.3.3 Curve Fitting and Calculating Radius of Curvature

From the last step, we have a vector having non-zero pixel intensity coordinates for both left and right half separately. This vector act as input to the polynomial fit method in OpenCV which helps in getting 2-Degree polynomial coefficients. Since the lane width is known, centimeter per pixel ratio can be estimated. By multiplying the (x,y) coordinates with the centimeter per pixel ratio we get coordinates having units in centimeter. Now we have 2-Degree curve equation having coefficients a, b and c known [22][23][24].

$$x = ay^2 + by + c \tag{3.7}$$

Radius of curvature can be found out by using formula

$$R = \frac{(1 + (\frac{dx}{dy})^2)^{\frac{3}{2}}}{\frac{d^2x}{dy^2}} \tag{3.8}$$

$$R = \frac{(1 + (2ay + b)^2)^{\frac{3}{2}}}{2a} \tag{3.9}$$

Considering vehicle position is at center, therefore its y coordinate becomes

$$y = (image\_height - 1) \tag{3.10}$$

$$R = \frac{(1 + (2a(image\_height - 1) + b)^2)^{\frac{3}{2}}}{2a} \tag{3.11}$$

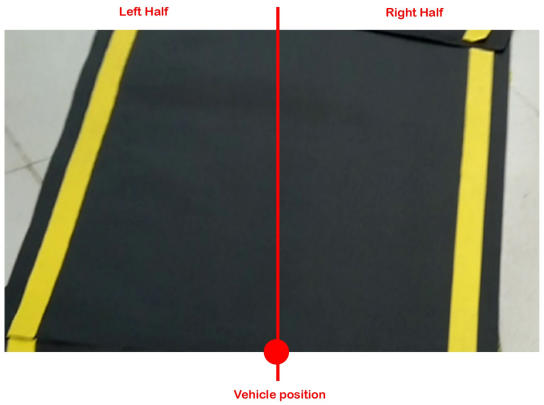


Fig. 3.14: Vehicle Position

Chapter 4

LEARNING BASED APPROACH

Machine learning is a sub-area of artificial intelligence that provides a software system the ability to learn and improve its prediction for an event, without being explicitly programmed. The learning process is initiated with observation of data, and letting the system predict a random output of events. Iterations are done to make this random output converge to true output, and reducing the error between the true output and predicted output for the provided example of events. The iterative process makes the system look for patterns in the data and adjust the program actions accordingly, to make better predictions or decisions in future [25].

4.1 Basics of Artificial Neural Network

An Artificial Neural Network is again a subset of machine learning, that involves higher dimensional input data and where the relation between an event and its output is complex.
The idea of neural network working is similar to the biological neuron of nervous system of a human body. At a cellular level, the nervous system consists of a special type of cell called "neuron". Each single neuron is connected to multiple neurons, which again are connected to next multiple neurons making a complex connected structure.
Same way, artificial neuron in a Neural Network is connected to multiple neurons. The structure of a single neuron unit by comparison to its biological counterpart can be seen in fig.4.1.

(a) Biological Neuron (b) Artificial Neuron

Fig. 4.1: Comparison between a biological neuron and a mathematical model of an artificial neuron

Output of a single neuron unit is calculated as shown in equation 4.1, where x_i is

the i^{th} input, w_i is its weight, b refers to the bias/threshold of the neuron and σ is the activation function.

$$y = \sigma\Big(\sum w_i x_i + b\Big) \tag{4.1}$$

A neural network is a collection of neurons organized in layers. The layers are divided in 3 types. There is an input layer, output layer and the remaining layers are called hidden layers. If there is more than one hidden layer, the network is called a deep neural network. Every neuron in each layer is input to neuron in the following layer. See fig. 4.2.

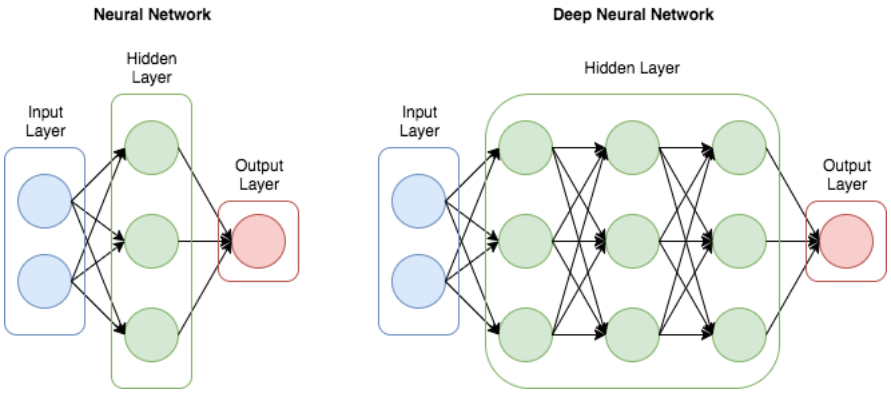


Fig. 4.2: Structure of a simple neural network

We denote by L, the number of layers and by$N^{(l)}$, the number of neurons in layer l. Let l = 0 be the input layer and l = L - 1 the output layer, we'll use w^l_{jk} to denote the weight of connection from k^{th} neuron in $(l-1)^{th}$ layer to j^{th} neuron in l^{th} layer. Similarly, b^l_j is the bias of j^{th} neuron in layer l. With this notation we can compute the activation a^l_j for the j^{th} neuron in l^{th} layer with the following formula

$$a^l_j = \sigma\Big(\sum_k w^l_{jk} a^{l-1}_j + b^l_j\Big) \tag{4.2}$$

In each layer, a non-linear transformation of the output is calculated using previous layer activations.

4.1.1 Learning and Cost Functions

Our goal is to find the parameters θ = (W, b) based on a dataset of input and output vectors (x_n, t_n), which would minimize the difference between the output of the trained network and the real function we are trying to approximate. In other words, we need to minimize a cost function $C(\theta)$, that can be defined in several ways. Equation 4.3 displays one example, the mean squared error loss function.

$$C_{MSE}(\theta) = \frac{1}{2N}\sum_{n=1}^{N} \mid\mid y(x_n, \theta) - t_n \mid\mid^2 \tag{4.3}$$

Now, in order to minimize the cost function and find the optimal weights W, we need to compute the gradient of the cost function with respect to the weights W, i.e.

$$\frac{\partial C_{MSE}}{\partial W} \tag{4.4}$$

4.2 Convolutional Neural Network

Convolutional neural networks are a type of neural network, mostly used while dealing with datasets containing images. In traditional neural networks, each neuron is connected to every node in its previous and forward layer. Due to this, the system has to perform costly matrix calculations while training a network. Also, it can be easily inferred that not every pixel needs to be involved in calculation.

Convolution is a mathematical operation defined on two functions. Convolution of a function f with a function g in continuous (4.5) and discrete (4.6) cases, is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x).g(t-x)dx \tag{4.5}$$

$$(f * g)(t) = \sum_{x=-\infty}^{\infty} f(x).g(t-x) \tag{4.6}$$

We use convolution in CNN on the set of input images. An image is a 3-dimensional input, 2 dimensions are width and height of image and the third is color channels. The equation 4.7 shows the method to calculate convolution of n^{th} output for an image.

$$(I * K)(n, x, y) = \sum_{c=0}^{C} \sum_{p=\frac{-k}{2}}^{\frac{k}{2}} \sum_{q=\frac{-k}{2}}^{\frac{k}{2}} I(c, x+p, y+q).K(c, n, p, q) \tag{4.7}$$

Here, I(c,x,y) is the pixel value of c^{th} channel at *(x,y)* position. K is the kernel with K(c,n,p,q) represents the weights of the network, and k is the kernel size.

Convolutional neural networks have some important properties due to which they are more suitable when dealing with images as compared to simple neural networks. These are sparse connectivity, parameter sharing and equivariant representations. Convolutional neural networks includes the correlation of nearby pixels of an image. It is done by using a kernel of size k which is smaller than the image size. Due to this property of sparse connectivity, the parameters to be learnt are reduced. Hence, the model's memory requirement and time taken for a computation is reduced. Additionally, the weights of a filter is shared when applied to different location of an image. This allows the network equivalent to translation, i.e., if we move an object in the input to a little left, the model can identify it, and modify the output accordingly.

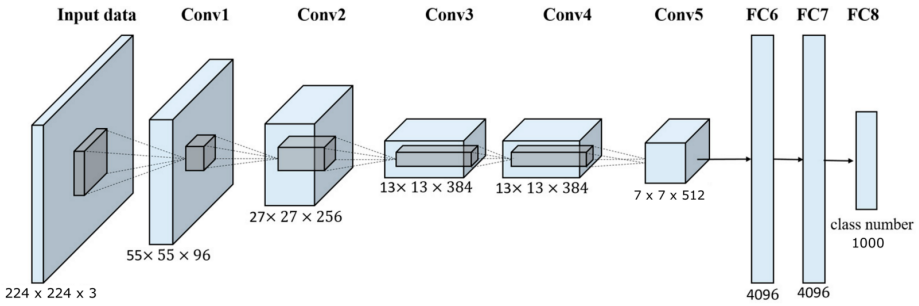


Fig. 4.3: A typical CNN Architecture

The fig. 4.3 shows a generalized architecture of a convolutional neural network. As we can see it mainly consist of 2 types of layers, convolutional layers and fully connected layers. The convolutional layer is responsible for the feature detection. Continuous convolutions on an input image extracts the image features such as color, edges, curves and boundaries. These feaures are then provided to the fully connected layer is used for classification and regression tasks.

4.3 Behavioral Cloning

Behavioral cloning is a method in which human behavior is observed and reproduced by a computer program. The event to be learnt is performed by a human, it is recorded along with the environment conditions related the event. The recorded log is provided as an input to the computer program, which is supposed to learn the behavior and reproduce it[26].

4.4 Network Architecture

The table 4.1 provides the complete architecture of the convolutional neural network for prediction of steering angles from input image.

S. No.	Name	Kernel	Activation
	Input Image		
1	Conv(24)	5 x 5	ELU
2	Conv(36)	5 x 5	ELU
3	Conv(48)	5 x 5	ELU
4	Conv(64)	3 x 3	ELU
5	Conv(64)	3 x 3	ELU
6	Dropout(0.5)		
7	FC(100)		ELU
8	FC(50)		ELU
9	FC(10)		ELU
10	FC(1)		1

Table 4.1: CNN architecture

Here, Conv corresponds to a layer performing convolutions. The number indicated with Conv is the number of output filters. As it can be seen, there are 5 convolutional layers, each with ELU(Exponential linear unit) activation, and 3 fully connected layer. The final layer is the output layer providing the predicted steering angle.

4.5 Method

We used the method of behavioral cloning to make the program learn to predict the steering angles on real time. A large amount of labelled dataset is required for model to learn. The process of data generation is done by two methods.

4.5.1 Simulation

A simulator built for Udacity's Self-Driving Car Nanodegree program is used[27]. The simulator car is manually driven in the simulator environment, and the training data is recorded. Each sample includes the images of the road as viewed from cameras mounted on the car, along with the corresponding steering angles. 3 cameras, left, center and right are mounted on the car, storing image from three perspective as shown in the fig 4.4

(a) left (b) center (c) right

Fig. 4.4: Simulator images

A total of 14727 images are recorded along with other controls like steering angle, throttle, brakes, acceleration etc. A csv file is created containing the location of 3 set of images along with respective steering angle, throttle value, brake and speed. This data is then used to train the neural network. The trained network is supposed to recognize the lanes in the road image and predict appropriate steering angle. The model is then used to drive the car in real-time in the simulator.

4.5.2 Hardware Level

For the hardware implementation of the method of behavioral cloning, an RC car is modified for data generation as explained in chapter 2. Black and yellow chart papers are used to make a single lane road, as shown in the image 4.5.

Fig. 4.5: Chart lane

These charts are assembled one by one to generate different road arenas for dataset generation. Few examples of road lanes prepared is shown in figure 2.10.

The test vehicle is manually driven on the created arena, trying to keep the vehicle at the center of the lane. A digital camera mounted on the vehicle captures the lanes image from the *first person view* and the image is stored in the microcontroller planted on the test vehicle. A total of 2000 events are generated by this process. A csv file is created containing the image address along with the steering angle corresponding to the image.

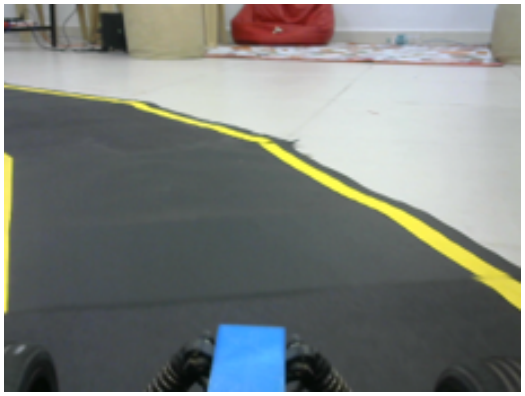
4.6 Data Cleansing

Data cleansing or data cleaning is the process of identifying and removing the data that is not accurate in a dataset, tables. Since the dataset is created on a real world environment, it contains a lot of noisy images which is not suitable to provide for training, which is needed to be cleaned. Some of the examples of inaccurate images are shown in fig. 4.6.

(a) Blurred images

(b) Image with obstacles/no lanes

(c) Off track images



(d) Mislabelled images
Angle=+15

Fig. 4.6: Types of Inaccurate data

4.7 Network Training

Once the data is prepared by either of the method, it is transferred to a GPU (we have used NVIDIA 1080 TI) for training of the network (see table 4.1). Network is trained for 10 epochs, each involving 20000 iterations. Value to be monitored is the mean squared loss between the predicted steering angle and true value of steering angle. Once trained, the model is transferred to the test vehicle, and is used to predict steering angles for the input camera image. The vehicle is made to run at a constant speed and steering is provided to front wheels from the value predicted by the trained model.

Chapter 5

RESULTS

Autonomous lane follower prototype was successfully designed, assembled and some parts of it were fabricated. Image processing based method and Artificial Intelligence based method were implemented. The team was able to design an algorithm for obtaining steering angle using image processing. Artificial Intelligence based method, also known as Behavioral Cloning, model was developed and trained using several sets of images. Trained model was first tested on simulator and after getting satisfying results in simulator, it was transferred to hardware with some modification in algorithm. The team was able to make an RC car follow the road lanes autonomously. The results are shown below.

1. The Image Processing algorithm was tested on a curvy road track, designed by us, in the video. The steering angle predicted by the algorithm and the theoretical angle calculated by drawing a tangent to the lane with the vertical line is found to be nearly same.

Fig. 5.1: Estimation of Steering angle

2. Algorithm to track right and left half road lanes were also tested in video and it was able to track road lanes with good accuracy.

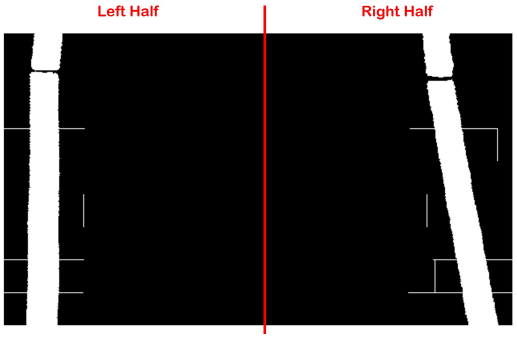


Fig. 5.2: Tracking of lane

3. The data set generated for model learning was divided into two parts, i.e., Training set (80% of total set) and validation set (20% of total set). The training set is used to train the model, and the validation set is used to check if the model predicts correct angles for unseen images. The model is trained for 10 epochs, and its loss is calculated for both training set and validation set after each epoch. The mean square error is used to calculate the loss. The difference between actual steering angle and predicted steering angle is squared and the mean is taken. Training of model is done to minimize the error. The following graph has been obtained (See fig. 5.3).

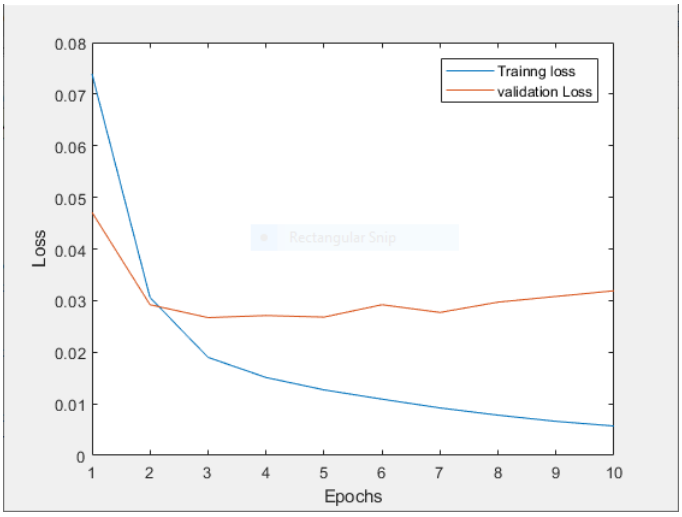


Fig. 5.3: Training and Validation Loss after each Epoch

As it can be seen, that with more and more training, the training loss reduces with each epoch, i.e., the model fits the training set. But the loss for validation set starts to increase after few epochs, which means its prediction accuracy decreases on unseen data set. Epoch 3 being the minimum of validation loss, the model obtained after 3^{rd} epoch is finally used for real time vehicle steering prediction.

Chapter 6

FUTURE WORK

The current model, we have built, is just a prototype of the self driving car. Researchers have shown great interest in Machine Learning and this field is growing exponentially. As technology advances, accuracy in the working of self driving car increases which in fact boosts its applications in real life.

Following things can be implemented for better accuracy and safety:

1. Installation of more sensitive sensors, such as a LIDAR can increase the processing speed of the vehicle, thus increasing detection range and improving safety.
2. Multiple sensors, such as RADAR, wheel encoders, GPS, ultrasonic sensors, can be used.
3. Inter-vehicle communication can be implemented so that the information regarding traffic or any ongoing construction can be effectively communicated and time can be saved. This is divided into four types of communication,
    - Vehicle to other vehicle(V2V)
    - Vehicle to the road-side infrastructure(V2I)
    - Vehicle to Pedestrians(V2P)
    - Vehicle to the cellular networks(V2N)

Chapter 7

APPENDIX

7.1 Software Packages Used

- OpenCV
- Tensorflow
- RPi.GPIO
- Pandas
- Numpy
- Keras
- Jupyter

7.2 Software Source Code

7.2.1 Estimation of steering angle

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

//intrinsic and extrinsic parametrs
float camera_matrix[9] = {819.752611, 0.000000, 329.461908, 0.000000, 819.011546,
    245.006332, 0.000000, 0.000000, 1.000000};
Mat intrinsic = Mat(3, 3, CV_32FC1, camera_matrix);
float dist_matrix[5] = {0.066489, 0.117170, 0.00383, 0.018518, 0.000000};
Mat distCoeffs = Mat(1, 5, CV_32FC1, dist_matrix);
float projection_matrix[12] = {836.864685, 0.000000, 336.647743, 0.000000, 0.000000,
    845.784973, 246.055352, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000};
Mat proj_matrix = Mat(3, 4, CV_32FC1, projection_matrix);
// Input Quadilateral or Image plane coordinates
Point2f InputQuad[4];
// Output Quadilateral or World plane coordinates
Point2f OutputQuad[4];
// Lambda Matrix
```

```
Mat lambda( 2, 4, CV_32FC1 );


int value;
int threshold_type=3 , threshold_count=179;
Mat src_rgb,src_gray,src_thresh, detected_edges, detected_edges_rgb, src_gray_blur;
int ratio = 3;
int kernel_size = 3;
int canny_lowThreshold=100;


int main(int argc, char* argv[])
{
     // VideoCapture cap(1);
      VideoCapture cap("/home/pranay/Final_Year_Project/video/test4.mp4"); // open the
          default camera
    if(!cap.isOpened()) // check if we succeeded
      return -1;

    cap.set(CV_CAP_PROP_FRAME_WIDTH,1280);//640
    cap.set(CV_CAP_PROP_FRAME_HEIGHT,720);//480

    Mat input, output, output_gray , threshold_lab, image_lab;

    //for initialization
    cap >> input;
    int height = input.size().height, width = input.size().width;
    cout<<"height = "<<height<<" width = "<<width;

    //get the perspective transform matrix
    lambda = Mat::zeros( input.rows, input.cols, input.type() );
    // The 4 points that select quadilateral on the input , from top-left in clockwise
        order
    // These four pts are the sides of the rect box used as input
    InputQuad[0] = Point2f( 0.2 * input.size().width,0.45 * input.size().height);
    InputQuad[1] = Point2f( 0.80 * input.size().width,0.45 * input.size().height);
    InputQuad[2] = Point2f( 0.1 * input.size().width,0.9 * input.size().height);
    InputQuad[3] = Point2f( 0.9 * input.size().width,0.9 * input.size().height);
    // The 4 points where the mapping is to be done , from top-left in clockwise order
    OutputQuad[0] = Point2f( 0,0 );
    OutputQuad[1] = Point2f( input.cols,0);
    OutputQuad[2] = Point2f( 0,input.rows);
    OutputQuad[3] = Point2f( input.cols,input.rows );

    // Get the Perspective Transform Matrix i.e. lambda
    lambda = getPerspectiveTransform( InputQuad, OutputQuad );
    while (true)
    {
        cap >> input;
        //if frame is empty then break loop
        if (input.empty())
        {
            break;
        }

        warpPerspective(input,output,lambda,output.size() );

        //put circle on input image
        circle(input, Point2f(0.20 * input.size().width,0.45 * input.size().height), 5,
            cv::Scalar(0, 0, 255), 10 , 8, 0);
        circle(input, Point2f(0.80 * input.size().width,0.45 * input.size().height), 5,
            cv::Scalar(0, 0, 255), 10 , 8, 0);
```

```
        circle(input, Point2f( 0 * input.size().width,1 * input.size().height), 5,
            cv::Scalar(0, 0, 255), 10 , 8, 0);
        circle(input, Point2f(1 * input.size().width,1 * input.size().height), 5,
            cv::Scalar(0, 0, 255), 10 , 8, 0);
       //convert to LAB space
        cvtColor(output, image_lab, CV_BGR2Lab);
        inRange(image_lab, cv::Scalar(0, 108, 0), cv::Scalar(255,255,154), threshold_lab);
        // Canny detector
      Canny( threshold_lab, detected_edges, canny_lowThreshold, canny_lowThreshold*ratio,
          kernel_size );
      // Using Canny's output as a mask, we display our result
     detected_edges_rgb = Scalar::all(0);
     output.copyTo( detected_edges_rgb, detected_edges); // here detected_edges is an
         binary image. It is mapped to pixels in source rgb image

     //hough lines
    vector<Vec2f> lines;
    HoughLines(detected_edges, lines, 1, CV_PI/180, 140, 0, 0 );
      float angle_degree=0 ,theta_degree;
      for( size_t i = 0; i < lines.size(); i++ )
      {
      float rho = lines[i][0], theta = lines[i][1];
      Point pt1, pt2;
      double a = cos(theta), b = sin(theta);
      double x0 = a*rho, y0 = b*rho;
      pt1.x = cvRound(x0 + 1000*(-b));
      pt1.y = cvRound(y0 + 1000*(a));
      pt2.x = cvRound(x0 - 1000*(-b));
      pt2.y = cvRound(y0 - 1000*(a));
      line( output, pt1, pt2, Scalar(255,0,255), 3);
      theta_degree = theta*(180/CV_PI);
      if (theta_degree>90)
      {
       theta_degree = theta_degree - 180;
      }
      angle_degree = angle_degree + theta_degree;
      }
      angle_degree = angle_degree/lines.size();
      cout<<"angle = "<<angle_degree<<endl;
      int count=0;
      namedWindow( "Display window1", WINDOW_NORMAL );
      imshow( "Display window1", input );
      namedWindow( "Display window2", WINDOW_NORMAL );
      imshow( "Display window2", output );
      namedWindow( "Display window3", WINDOW_NORMAL );
      imshow( "Display window3", threshold_lab );
      namedWindow( "window4", WINDOW_NORMAL );
      imshow( "window4",detected_edges);
      namedWindow("window5", WINDOW_NORMAL);
      imshow( "window5",image_lab);
      int iKey = waitKey(50);
        //if user press 'ESC' key
        if (iKey == 27)
        {
        break;
        }
   }
return 0;
}
```

7.2.2 Estimation of radius of curvature

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

//intrinsic and extrinsic parametrs
float camera_matrix[9] = {819.752611, 0.000000, 329.461908, 0.000000, 819.011546,
    245.006332, 0.000000, 0.000000, 1.000000};
Mat intrinsic = Mat(3, 3, CV_32FC1, camera_matrix);

float dist_matrix[5] = {0.066489, 0.117170, 0.00383, 0.018518, 0.000000};
Mat distCoeffs = Mat(1, 5, CV_32FC1, dist_matrix);

float projection_matrix[12] = {836.864685, 0.000000, 336.647743, 0.000000, 0.000000,
    845.784973, 246.055352, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000};
Mat proj_matrix = Mat(3, 4, CV_32FC1, projection_matrix);

//*********************************

// Input Quadilateral or Image plane coordinates
Point2f InputQuad[4];
// Output Quadilateral or World plane coordinates
Point2f OutputQuad[4];
// Lambda Matrix
Mat lambda( 2, 4, CV_32FC1 );


//*********************************

void fitPoly(const vector <Point> &src,Mat &dst, int order){

try
{

    Mat src_x = Mat(src.size(), 1, CV_32FC1);
    Mat src_y = Mat(src.size(), 1, CV_32FC1);

    for (int i = 0; i < src.size(); i++){
        src_x.at<float>(i, 0) = (float)src[i].x;
        src_y.at<float>(i, 0) = (float)src[i].y;
    }

Mat X;
X = Mat::zeros(src_x.rows, order+1,CV_32FC1);
Mat copy;

//change type of copy
copy.convertTo(copy,CV_32FC1);

for(int i = 0; i <=order;i++)
{
copy = src_x.clone();
pow(copy,i,copy);
```

```
Mat M1;

//change type
M1.convertTo(M1,CV_32FC1);

M1 = X.col(i);
copy.col(0).copyTo(M1);
}
Mat X_t, X_inv;
X_inv.convertTo(X_inv,CV_32FC1);
X_t.convertTo(X_t,CV_32FC1);

transpose(X,X_t);

Mat temp;
temp.convertTo(temp,CV_32FC1);
temp = X_t*X;
Mat temp2;
temp2.convertTo(temp2,CV_32FC1);
invert(temp,temp2);

Mat temp3;
temp3.convertTo(temp3,CV_32FC1);

temp3 = temp2*X_t;

Mat W;
W.convertTo(W,CV_32FC1);
W = temp3*src_y;

dst = W.clone();

}
catch (cv::Exception const & e)
{
  //std::cerr<<"OpenCV exception: "<<e.what()<<std::endl;
}


 }

int main(int argc, char* argv[])
{

    VideoCapture cap("/home/pranay/Final_Year_Project/video/test4.mp4"); // open the
        default camera
    //VideoCapture cap(1); // open the default camera
    if(!cap.isOpened()) // check if we succeeded
       return -1;

    cap.set(CV_CAP_PROP_FRAME_WIDTH,1280);//640
    cap.set(CV_CAP_PROP_FRAME_HEIGHT,720);//480

    Mat input, output, output_gray , threshold_lab, image_lab;

    //for initialization
    cap >> input;
    int height = input.size().height, width = input.size().width;
    cout<<"height = "<<height<<" width = "<<width;

    //get the perspective transform matrix
    lambda = Mat::zeros( input.rows, input.cols, input.type() );
```

```
  // The 4 points that select quadilateral on the input , from top-left in clockwise
      order
  // These four pts are the sides of the rect box used as input
  InputQuad[0] = Point2f( 0.25 * input.size().width,0.45 * input.size().height);
  InputQuad[1] = Point2f( 0.75 * input.size().width,0.45 * input.size().height);
  InputQuad[2] = Point2f( 0.1 * input.size().width,1 * input.size().height);
  InputQuad[3] = Point2f( 0.9 * input.size().width,1 * input.size().height);
  // The 4 points where the mapping is to be done , from top-left in clockwise order
  OutputQuad[0] = Point2f( 0,0 );
  OutputQuad[1] = Point2f( input.cols,0);
  OutputQuad[2] = Point2f( 0,input.rows);
  OutputQuad[3] = Point2f( input.cols,input.rows );

  // Get the Perspective Transform Matrix i.e. lambda
  lambda = getPerspectiveTransform( InputQuad, OutputQuad );

  cout<<"\n"<<lambda<<"\n";

  int loop=0;
while (true)
  {

      cap >> input;

      //cout<<"\n"<<loop++<<"\n";

      //if frame is empty then break loop
      if (input.empty())
      {
          break;
      }
      warpPerspective(input,output,lambda,output.size() );

      //put circle on input image
      circle(input, Point2f(0.25 * input.size().width,0.45 * input.size().height), 5,
          cv::Scalar(0, 0, 255), 1 , 8, 0);
      circle(input, Point2f(0.75 * input.size().width,0.45 * input.size().height), 5,
          cv::Scalar(0, 0, 255), 1 , 8, 0);
      circle(input, Point2f( 0.1 * input.size().width,1 * input.size().height), 5,
          cv::Scalar(0, 0, 255), 1 , 8, 0);
      circle(input, Point2f(0.9 * input.size().width,1 * input.size().height), 5,
          cv::Scalar(0, 0, 255), 1 , 8, 0);

      //convert to LAB space
      cvtColor(output, image_lab, CV_BGR2Lab);

      inRange(image_lab, cv::Scalar(0, 108, 0), cv::Scalar(255,255,154), threshold_lab);


      Mat noob;
      cvtColor( input, noob,CV_BGR2Lab);

      inRange(noob, cv::Scalar(0, 108, 0), cv::Scalar(255,255,154), noob);

      bitwise_not ( threshold_lab, threshold_lab);

      int height = threshold_lab.size().height, width = threshold_lab.size().width;

      //declare without size
      vector<int> ColumnSum;
      vector<int> sumCols;
      Mat histogram(height,width, CV_8UC1, Scalar(0));
```

```
reduce(threshold_lab, ColumnSum, 0, CV_REDUCE_SUM, CV_32S);

int lefty_max=0,
    righty_max=0,midpoint,leftx_index,rightx_index,nwindows=9,margin=150,minpix=1;
midpoint = width/2;


for(int i=0;i<ColumnSum.size();i++)
 {

    ColumnSum[i] = (ColumnSum[i]/255);
    //cout<<"ColumnSum[i] "<<ColumnSum[i]<<" i= "<<i<<"\n";
   if(ColumnSum[i] <1080 && i < 1920)
  {


      histogram.at<uchar>(ColumnSum[i], i, 0) = 255;
      circle(histogram, Point2f(i,ColumnSum[i]), 5, cv::Scalar(255, 255, 255), 5
          , 8, 0);

      if(i<midpoint && ColumnSum[i]>lefty_max)
    {
      lefty_max = ColumnSum[i];
      leftx_index = i;
    }
    else if(i>=midpoint && ColumnSum[i]>righty_max)
    {
      righty_max = ColumnSum[i];
      rightx_index = i;
    }
    //cout<<ColumnSum[i]<<" ";

  }

 }

int window_height = height/nwindows;

flip(histogram, histogram, 0);



vector<Point> nonZeroCoordinates(height*width);

// calculates position values x,y of non zero points in threshold_lab binary image
findNonZero(threshold_lab, nonZeroCoordinates);

int leftx_current,rightx_current;

leftx_current = leftx_index;
rightx_current = rightx_index;

vector<Point> left_lane_inds;
vector<Point> right_lane_inds;

int win_y_low,
    win_y_high,win_xleft_low,win_xleft_high,win_xright_low,win_xright_high,
    last_mean_right=rightx_index,last_mean_left=leftx_index,test;
bool draw_windows = true;
Mat window_left,window_right;
```

```
      Rect roi_left,roi_right;


for(int window=0;window<nwindows;window++)
  {
    // Identify window boundaries in x and y (and right and left)
    win_y_low = height - (window+1)*window_height;
    win_y_high = height - window*window_height;
    win_xleft_low = leftx_current - margin;
    win_xleft_high = leftx_current + margin;
    win_xright_low = rightx_current - margin;
    win_xright_high = rightx_current + margin;


    //window left side non zero pixel
    roi_left.x = win_xleft_low;
    roi_left.y = win_y_low;
    if(roi_left.x < 0)
    {
      roi_left.x = 0;
    }



    roi_left.width = win_xleft_high - win_xleft_low; //= 2*margin
    roi_left.height = win_y_high - win_y_low; //window height


    window_left = threshold_lab(roi_left); //select left window image in window_left


    //for storing non zero coordinates in left window
    vector<Point> nonZero_leftwindow(roi_left.width*roi_left.height);

    //get count of searched non zero coordinates in left window
    int count_left = countNonZero(window_left);


    if(count_left > 0)
     {
       //draw window when required and when count_left > 0
       if(draw_windows == true )
        {
        rectangle(threshold_lab, Point(win_xleft_low,win_y_low),
            Point(win_xleft_high,win_y_high), Scalar(255,255,255), 1);
        }

       //finds non zero points in left window
       findNonZero(window_left, nonZero_leftwindow);

       //add initial position because window and image dimension are different
       for (int i = 0; i < nonZero_leftwindow.size(); ++i)
       {
        nonZero_leftwindow[i].x = nonZero_leftwindow[i].x + roi_left.x;
        nonZero_leftwindow[i].y = nonZero_leftwindow[i].y + roi_left.y;
       }

       left_lane_inds.insert(left_lane_inds.end(),nonZero_leftwindow.begin(),nonZero_leftwindow.end()
     }
```

```
  //window right side non zero
  roi_right.x = win_xright_low;
  roi_right.y = win_y_low;
  if(roi_right.x > width)
  {
    roi_right.x = width ;
  }

  roi_right.width = win_xright_high - win_xright_low;
  roi_right.height = win_y_high - win_y_low;


  //********************
  //change roi_right.x if required

  if(roi_right.x + roi_right.width > width)
  {
    roi_right.x = width - roi_right.width;
  }

//cout<<"roi_right.x "<< roi_right.x <<"roi_right.y" <<roi_right.y <<"\n";
//cout<<"roi_right.width"<< roi_right.width <<"roi_right.height "
    <<roi_right.height<<"\n";



//******************


window_right = threshold_lab(roi_right);

vector<Point> nonZero_rightwindow(roi_right.width * roi_right.height);

int count_right = countNonZero(window_right);
//cout<<"count_left "<<count_right;
if(count_right > 0)
 {
  //draw window when required and when count_right > 0
  if(draw_windows == true)
    {
    rectangle(threshold_lab, Point(win_xright_low,win_y_low),
        Point(win_xright_high,win_y_high), Scalar(255,255,255), 1);
    }

  findNonZero(window_right, nonZero_rightwindow);
  //*****************

  for (int i = 0; i < nonZero_rightwindow.size(); ++i)
   {
    nonZero_rightwindow[i].x = nonZero_rightwindow[i].x + roi_right.x;
    nonZero_rightwindow[i].y = nonZero_rightwindow[i].y + roi_right.y;

   }


  right_lane_inds.insert(right_lane_inds.end(),nonZero_rightwindow.begin(),nonZero_rightwindow.end
 }


if(left_lane_inds.size() > minpix)
 {
  leftx_current = int(mean( nonZero_leftwindow)[0]);
```

```
      if (leftx_current==0)
      {
        leftx_current = last_mean_left;
      }
      else
      {
        last_mean_left = leftx_current;
      }
      //cout<<"mean left "<< int(mean( nonZero_leftwindow)[0])<<"\n";

     }
   if(right_lane_inds.size() > minpix)
     {
      rightx_current = int(mean( nonZero_rightwindow)[0]);
      if (rightx_current==0)
      {
        rightx_current = last_mean_right;
      }
      else
      {
        last_mean_right = rightx_current;
      }
      //cout<<"mean right "<< int(mean( nonZero_rightwindow)[0])<<"\n";
      //cout<<"mean right "<< rightx_current<<"\n";

     }

    }
//generate y values from 0 to height-1
 vector<int> y(input.size().height);
 for (int i = 0; i < input.size().height; i++)
 {
   y[i] = i;
 }

    // order of data matrix
int fit_order = 2;


// matrices to store computed coefficients
Mat fit_weights_left(fit_order+1,1,CV_32FC1);

fitPoly(left_lane_inds,fit_weights_left,fit_order);

// matrices to store computed coefficients
Mat fit_weights_right(fit_order+1,1,CV_32FC1);


//cout<<"right_lane_inds "<<right_lane_inds.size()<<"\n";
fitPoly(right_lane_inds,fit_weights_right,fit_order);

double ym_per_pix = 0.74/286;//30.5/720; // meters per pixel in y dimension
double xm_per_pix = 0.03/65;//3.7/720; // meters per pixel in x dimension

ym_per_pix = 74/286;
xm_per_pix = 03/65;


//xm_per_pix = 0.74/286;
double y_set = height-1;

vector<Point> left_fitx(height);
```

```
vector<Point> right_fitx(height);


//generate x values using coefficients of left region
// x = (a*y^2 + b*y + c)xm_per_pix

//right curve
for (int i = 0; i < height; ++i)
{
 right_fitx[i].x = ( (fit_weights_right.at<float>(0,000)*pow(y[i],2)) +
     (fit_weights_right.at<float>(1,000)*y[i]) + fit_weights_right.at<float>(2,000)
     )*xm_per_pix;
}

//left curve
for (int i = 0; i < height; ++i)
{
 left_fitx[i].x= ( (fit_weights_left.at<float>(0,000)*pow(y[i],2)) +
     (fit_weights_left.at<float>(1,000)*y[i]) + fit_weights_left.at<float>(2,000)
     )*xm_per_pix;
}

//already computed
//**************************
//generate y values from 0 to height-1
   for (int i = 0; i < height; ++i)
  {
    right_fitx[i].y = y[i]*ym_per_pix;
    left_fitx[i].y = y[i]*ym_per_pix;
  }

//******************************


 //again apply polyfit with real data

 //************************************************************************

 // matrices to store computed coefficients
 Mat fit_weights_left_real(fit_order+1,1,CV_32FC1);

 fitPoly(left_fitx,fit_weights_left_real,fit_order);

 // matrices to store computed coefficients
 Mat fit_weights_right_real(fit_order+1,1,CV_32FC1);

 //right_fitx size

 fitPoly(right_fitx,fit_weights_right_real,fit_order);

 //************************************************************************

  // Calculate the new radii of curvature
 double left_curverad,right_curverad;
 //left_curverad = ((1 + )**1.5) / abs(2*fit_weights_left_real.at<float>(0,000));
 //right_curverad = ((1 + (2*fit_weights_right_real.at<float>(0,000)*y_set*ym_per_pix +
     fit_weights_right_real.at<float>(1,000))**2)**1.5) /
     abs(2*fit_weights_right_real.at<float>(0,000));
 left_curverad = pow((1 + pow((2*fit_weights_left_real.at<float>(0,000)*y_set*ym_per_pix
     + fit_weights_left_real.at<float>(1,000)) , 2)) ,
     1.5)/abs(2*fit_weights_left_real.at<float>(0,000));
```

```
   right_curverad = pow((1 +
       pow((2*fit_weights_right_real.at<float>(0,000)*y_set*ym_per_pix +
       fit_weights_right_real.at<float>(1,000)) , 2)) ,
       1.5)/abs(2*fit_weights_right_real.at<float>(0,000));

   cout<<"***\n";
   cout<<"real radius of cur left : "<<left_curverad <<"\n";
   cout<<"a = "<<fit_weights_left_real.at<float>(0,000)<<" "<<"b =
       "<<fit_weights_left_real.at<float>(1,000)<<"c =
       "<<fit_weights_left_real.at<float>(2,000);
   cout<<"***\n";

   cout<<"////\n";
   cout<<"real radius of cur right : "<<right_curverad <<"\n";
   cout<<"a = "<<fit_weights_right_real.at<float>(0,000)<<" "<<"b =
       "<<fit_weights_right_real.at<float>(1,000)<<"c =
       "<<fit_weights_right_real.at<float>(2,000);
   cout<<"///\n";

        namedWindow( "Display window1", WINDOW_NORMAL );// Create a window for display.
        imshow( "Display window1", input );// Show our image inside it. //WINDOW_AUTOSIZE

        namedWindow( "Display window2", WINDOW_NORMAL );// Create a window for display.
        imshow( "Display window2", noob );// Show our image inside it.

        namedWindow( "Display window3", WINDOW_NORMAL );// Create a window for display.
        imshow( "Display window3", threshold_lab );// Show our image inside it.

        namedWindow( "Display window4", WINDOW_NORMAL );// Create a window for display.
        imshow( "Display window4", histogram);// Show our image inside it.

         int iKey = waitKey(50);

         //if user press 'ESC' key
         if (iKey == 27)
         {
         break;
         }

    }

   return 0;
}
```

7.2.3 Training of Model

```
   #To train a pretrained model on real world image sets
#Date : 2/3/2019



import pandas as pd # data analysis toolkit - create, read, update, delete datasets
import numpy as np #matrix math
from sklearn.model_selection import train_test_split #to split out training and testing
    data
#keras is a high level wrapper on top of tensorflow (machine learning library)
#The Sequential container is a linear stack of layers
from keras.models import load_model
#popular optimization strategy that uses gradient descent
```

```
from keras.optimizers import Adam
#to save our model periodically as checkpoints for loading later
from keras.callbacks import ModelCheckpoint
#what types of layers do we want our model to have?
from keras.layers import Lambda, Conv2D, MaxPooling2D, Dropout, Dense, Flatten
#helper class to define input shape and generate training images given image paths &
    steering angles
from utils_2 import INPUT_SHAPE, batch_generator
#for command line arguments
import argparse
#for reading files
import os

#for debugging, allows for reproducible (deterministic) results
np.random.seed(0)


def load_data(args):
    """
    Load training data and split it into training and validation set
    """
    #reads CSV file into a single dataframe variable
    data_df = pd.read_csv(os.path.join(os.getcwd(), args.data_dir, 'new_data.csv'),
        names=['image', 'steering'])

    #Using dataframes, we can select rows and columns by their names
    #we'll store the camera images as our input data
    X = data_df['image'].values
    #and our steering commands as our output data
    y = data_df['steering'].values

    #now we can split the data into a training (80), testing(20), and validation set
    #thanks scikit learn
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=args.test_size,
        random_state=0)

    return X_train, X_valid, y_train, y_valid


def build_model(args):
    model = load_model(args.model)
    model.summary()
    return model


def train_model(model, args, X_train, X_valid, y_train, y_valid):
    """
    Train the model
    """
    #Saves the model after every epoch.
    #quantity to monitor, verbosity i.e logging mode (0 or 1),
    #if save_best_only is true the latest best model according to the quantity monitored
        will not be overwritten.
    #mode: one of {auto, min, max}. If save_best_only=True, the decision to overwrite the
        current save file is
    # made based on either the maximization or the minimization of the monitored quantity.
        For val_acc,
    #this should be max, for val_loss this should be min, etc. In auto mode, the direction
        is automatically
    # inferred from the name of the monitored quantity.
    checkpoint = ModelCheckpoint('model-{epoch:03d}.h5',
                                 monitor='val_loss',
```

```
                                verbose=0,
                                save_best_only=args.save_best_only,
                                mode='auto')

    #calculate the difference between expected steering angle and actual steering angle
    #square the difference
    #add up all those differences for as many data points as we have
    #divide by the number of them
    #that value is our mean squared error! this is what we want to minimize via
    #gradient descent
    model.compile(loss='mean_squared_error', optimizer=Adam(lr=args.learning_rate))

    #Fits the model on data generated batch-by-batch by a Python generator.

    #The generator is run in parallel to the model, for efficiency.
    #For instance, this allows you to do real-time data augmentation on images on CPU in
    #parallel to training your model on GPU.
    #so we reshape our data into their appropriate batches and train our model
        simulatenously
    model.fit_generator(batch_generator(args.data_dir, X_train, y_train, args.batch_size,
        True),
                        args.samples_per_epoch,
                        args.nb_epoch,
                        max_q_size=1,
                        validation_data=batch_generator(args.data_dir, X_valid, y_valid,
                            args.batch_size, False),
                        nb_val_samples=len(X_valid),
                        callbacks=[checkpoint],
                        verbose=1)

#for command line args
def s2b(s):
    """
    Converts a string to boolean value
    """
    s = s.lower()
    return s == 'true' or s == 'yes' or s == 'y' or s == '1'


def main():
    """
    Load train/validation data set and train the model
    """
    parser = argparse.ArgumentParser(description='Behavioral Cloning Training Program')
    parser.add_argument('model', type=str, help='Path to model h5 file. Model should be on
        the same path.')
    parser.add_argument('-d', help='data directory',  dest='data_dir',       type=str,
        default='data')
    parser.add_argument('-t', help='test size fraction', dest='test_size',    type=float,
        default=0.2)
    parser.add_argument('-k', help='drop out probability', dest='keep_prob',  type=float,
        default=0.2)
    parser.add_argument('-n', help='number of epochs', dest='nb_epoch',       type=int,
        default=10)
    parser.add_argument('-s', help='samples per epoch', dest='samples_per_epoch',
        type=int, default=30000)
    parser.add_argument('-b', help='batch size',      dest='batch_size',      type=int,
        default=40)
    parser.add_argument('-o', help='save best models only', dest='save_best_only',
        type=s2b, default='true')
    parser.add_argument('-l', help='learning rate',   dest='learning_rate',   type=float,
        default=1.0e-5)
```

```
    args = parser.parse_args()

    #print parameters
    print('-' * 30)
    print('Parameters')
    print('-' * 30)
    for key, value in vars(args).items():
        print('{:<20} := {}'.format(key, value))
    print('-' * 30)

    #load data
    data = load_data(args)
    #build model
    model = build_model(args)
    #train model on data, it saves as model.h5
    train_model(model, args, *data)


if __name__ == '__main__':
    main()
```

7.2.4 Functions Used While Training

```
import cv2, os
import numpy as np
import matplotlib.image as mpimg


IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 66, 200, 3
INPUT_SHAPE = (IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)


def load_image(data_dir, image_file):
    """
    Load RGB images from a file
    """
    return mpimg.imread(os.path.join(data_dir, image_file.strip()))


def crop(image):
    """
    Crop the image (removing the sky at the top and the car front at the bottom)
    """
    return image[120:-60, :, :] # remove the sky and the car front


def resize(image):
    """
    Resize the image to the input shape used by the network model
    """
    return cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)


def rgb2yuv(image):
    """
    Convert the image from RGB to YUV (This is what the NVIDIA model does)
    """
    return cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
```

```
def preprocess(image):
    """
    Combine all preprocess functions into one
    """
    image = crop(image)
    image = resize(image)
    image = rgb2yuv(image)
    return image


def choose_image(data_dir, center, steering_angle):
    """
    load image
    """
    return load_image(data_dir, center), steering_angle



def random_flip(image, steering_angle):
    """
    Randomly flipt the image left <-> right, and adjust the steering angle.
    """
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)
        steering_angle = -steering_angle
    return image, steering_angle



def random_brightness(image):
    """
    Randomly adjust brightness of the image.
    """
    # HSV (Hue, Saturation, Value) is also called HSB ('B' for Brightness).
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    ratio = 1.0 + 0.4 * (np.random.rand() - 0.5)
    hsv[:,:,2] = hsv[:,:,2] * ratio
    return cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)


def augument(data_dir, center, steering_angle, range_x=100, range_y=10):
    """
    Generate an augumented image and adjust steering angle.
    (The steering angle is associated with the center image)
    """
    image, steering_angle = choose_image(data_dir, center, steering_angle)
    image, steering_angle = random_flip(image, steering_angle)
    image = random_brightness(image)
    return image, steering_angle


def batch_generator(data_dir, image_paths, steering_angles, batch_size, is_training):
    """
    Generate training image give image paths and associated steering angles
    """
    images = np.empty([batch_size, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS])
    steers = np.empty(batch_size)
    while True:
        i = 0
        for index in np.random.permutation(image_paths.shape[0]):
            center = image_paths[index]
```

```
            steering_angle = steering_angles[index]
            # argumentation
            if is_training and np.random.rand() < 0.6:
                image, steering_angle = augument(data_dir, center, steering_angle)
            else:
            image = load_image(data_dir, center)
            # add the image and steering angle to the batch
            images[i] = preprocess(image)
            steers[i] = steering_angle
            i += 1
            if i == batch_size:
                break
        yield images, steers
```

7.2.5 Dataset Generation Code

```
import RPi.GPIO as GPIO
import time
import readchar
import signal
import sys
import xlsxwriter
import cv2
# import load_workbook
from openpyxl import load_workbook
import time

#*************

GPIO.setmode(GPIO.BOARD)

#Set pins 12 & 14 as output PWM
GPIO.setup(12, GPIO.OUT)
GPIO.setup(11, GPIO.OUT)

q = GPIO.PWM(12, 50)
p = GPIO.PWM(11, 50)

#Define values to variables
stop = 4.7
run = 7.33
steer0 = 7.25
steer_max = 9
steer_min = 5.5
steer = steer0

p.start(stop) #Initial speed of vehicle = 0
q.start(steer) #Initial steering of vehicle = 0
bldc = stop
wb = load_workbook('sample-test.xlsx')
ws = wb.active

# write the header in row 0, which is Excel row 1
#ws.write_row(0, 0, ['angle'])
i = ws.max_row
print("Last saved set", i)
vidcap = cv2.VideoCapture(-1);
success,image = vidcap.read()
success = True
```

```
#Set video resolution
vidcap.set(3,1280);
vidcap.set(4,720);

#For steering, PWM to servo
while success:
    i=i+1
    x = readchar.readkey()
    #for servo
    if x=="a":
        steer = steer - 0.25
        #bldc = run + abs(steer0-steer)/100
    elif x=="d":
        steer = steer + 0.25
        #bldc = run + abs(steer0-steer)/100
    elif x=="s":
        steer = steer0
 #To exit
    if x=="e":
        wb.save('sample-test.xlsx')
        wb.close()
        sys.exit(0)
        print("Total saved set", i)

 #For BLDC
    if x=="w":
        bldc = run
    elif x==" ":
        bldc = stop

    #If steer limit is crossed
    if steer > steer_max:
        steer = steer_max
    if steer < steer_min:
        steer = steer_min

    q.ChangeDutyCycle(steer) #Chanhge steering angle
    p.ChangeDutyCycle(bldc) #Change forward speed

    #Duty ratio to steering angle conversion
    steering_angle = (60*steer/3.5) - (60*7.25/3.5)

    #Print data on terminal
    print("Data No: ", i, "Servo = ", steer, "BLDC = ", bldc)
    success,image = vidcap.read()
    print('Read a new frame: ', success)

    #Store image and image name in excel file with date
    timestr = time.strftime("%Y%m%d-%H%M%S")
    img_loc = "image/" + timestr + ".png"
    cv2.imwrite(img_loc, image)
    #ws.write_row(i, 0, [img_loc])
    #ws.write_row(i, 1, [duty1])
    ws.cell(row = i, column=1).value = img_loc
    ws.cell(row = i, column=2).value = steering_angle
```

Bibliography

[1] *The 5 autonomous driving levels.* [Online]. Available: https://www.iotforall.com/5-autonomous-driving-levels-explained/.

[2] *History-of-self-driving-cars-milestones.* [Online]. Available: https://www.digitaltrends.com/cars/history-of-self-driving-cars-milestones/.

[3] *History of the autonomous car.* [Online]. Available: https://www.titlemax.com/resources/history-of-the-autonomous-car/.

[4] *Self driving car companies.* [Online]. Available: https://www.predictiveanalyticstoday.com/top-companies-autonomous-cars-self-driving-car/.

[5] *Self-driving car.* [Online]. Available: https://en.wikipedia.org/wiki/Self-driving_car.

[6] *Raspberry pi.* [Online]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi.

[7] *Embedded computer vision: Which device should you choose?* [Online]. Available: https://www.learnopencv.com/embedded-computer-vision-which-device-should-you-choose/.

[8] *What are the best raspberry pi alternatives?* [Online]. Available: https://www.zdnet.com/article/what-are-the-best-raspberry-pi-alternatives-everything-you-need-to-know-about-pi-rivals/.

[9] *Brushless dc motor (bldc).* [Online]. Available: https://circuitdigest.com/microcontroller-projects/what-is-bldc-motor-and-arduino-bldc-motor-control.

[10] *Lipo battery.* [Online]. Available: https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjUqInwidLhAhWHuI8KHW7LCpUQjRx6BAgBEAU&url=https%3A%2F%2Fhacktronics.co.in%2Fcells-and-batteries%2Frechargeable-li-polymer-lipo-battery-111v-2200mah-25c-3s-t-plug-power-for-rc-quadcopter&psig=AOvVaw3xzO_ghEq31Y_WSGYqaZfo&ust=1555416894834386.

[11] *Driving an esc/brushless-motor using raspberry pi.* [Online]. Available: https://www.instructables.com/id/Driving-an-ESCBrushless-Motor-Using-Raspberry-Pi/.

[12] *Controlling a servo from the raspberry pi.* [Online]. Available: https://rpi.science.uoit.ca/lab/servo/.

[13] *Buck converter.* [Online]. Available: https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwi5hJ2yttLhAhULLVAKHYfvBUYQjRx6BAgBEAU&url=https%3A%2F%2Fwww.amazon.com%2FRioRand-3-01-0076-Converter-Module-1-23V-30V%2Fdp%2FB008BHAOQO&psig=AOvVaw0sShu3WbvwpchXIl_Iz3lv&ust=1555428898440084.

[14] *Camera.* [Online]. Available: https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjW5cWHt9LhAhVLXK0KHTZPDDUQjRx6BAgBEAU&url=https%3A%2F%2Fwww.bestbuy.ca%2Fen-ca%2Fproduct%2Flogitech-logitech-hd-webcam-c270-960-000621%2F10146689.aspx&psig=AOvVaw3728DB2fQsL7eS6kOX-eg8&ust=1555429088485972.

[15] *Raspberry pi servo motor control.* [Online]. Available: https://tutorials-raspberrypi.com/raspberry-pi-servo-motor-control/.

[16] *Opencv.* [Online]. Available: https://opencv.org.

[17] *Bird's eye view transformation.* [Online]. Available: https://nikolasent.github.io/opencv/2017/05/07/Bird27s-Eye-View-Transformation.html.

[18] *Canny edge detection*. [Online]. Available: https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html.

[19] *Hough line transform*. [Online]. Available: https://docs.opencv.org/3.4.0/d9/db0/tutorial_hough_lines.html.

[20] *Curved lane detection*. [Online]. Available: https://www.hackster.io/kemfic/curved-lane-detection-34f771.

[21] *Sliding window method*. [Online]. Available: https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/.

[22] *Radius of curvature*. [Online]. Available: https://en.wikipedia.org/wiki/Radius_of_curvature.

[23] *Curvature and radius of curvature*. [Online]. Available: https://www.math24.net/curvature-radius/.

[24] *Calibrating undistorting with opencv in c++*. [Online]. Available: http://www.aishack.in/tutorials/calibrating-undistorting-opencv-oh-yeah/.

[25] *What is machine learning? a definition*. [Online]. Available: https://www.expertsystem.com/machine-learning-definition/.

[26] *Behavioral cloning—make a car drive like yourself*. [Online]. Available: https://medium.com/@ksakmann/behavioral-cloning-make-a-car-drive-like-yourself-dc6021152713.

[27] *A self-driving car simulator built with unity*. [Online]. Available: https://github.com/udacity/self-driving-car-sim.