TABLE OF CONTENTS

TABLES

FIGURES

Chapter 1

INTRODUCTION

According to the statistics provided by ASIRT nearly 1.3 million people die in road crashes each year caused by recklessness, distractions or speeding, on average 3,287 deaths a day and an additional 20-50 million are injured or disabled. Apart from this, there are millions being spent on vehicle repairs, all the while paying for the large volumes of fuel due to high consumption and ending up with more pollution. One solution to taper this global problem by a substantial amount is self-driving cars. They have the potential to significantly reduce the number of vehicle fatalities. Additionally, wasted time spent commuting could be reduced, increasing overall productivity. Fuel efficiency could be increased by calculating the best possible route between destinations, thus truncating the level of pollution, however, these stand as only a subset of the multitude of benefits provided by self-driving cars. Due to these advantages, it is a field being highly researched on; spending immense time, intellect and resources.

For self-driving cars to come to realization require both hardware components and software packages to be designed and made compatible with one another. Nonetheless, in this project, we deal with the software aspects necessary to build a model that can learn to drive a car in a very diverse set of virtual environment. Though the basic definition of a self-driving car: a robotic device that can travel between destinations without a human operator, sounds quite simple and straightforward in reality, it isn't so. This is the very reason for us to come down in favour of the utilization of Convolutional Neural Networks (CNN). Our approach consists of the concept of regression at its core that is trained with prodigious datasets of images that have been taken up from diverse conditions generated by manually driving a car in a computer video game. Based on the input given, the model is drilled to determine the appropriate action to be taken without the need of any human intellect. The accuracy of its decision will be based on its trained data-sets from an image that it receives, the quality of the image and the number of course changing factors taken into account.

The aim of our approach is to convert frames per second into appropriate steering angles. The input data frames obtained by screen capturing utilizing OpenCV while manually driving a car on a GTA 5 game console are labeled. This dataset is then scaled down to an optimal resolution suitable for training. The scaled dataset is given as an input to a neural network consisting of multiple hidden layers. Each layer applies a function to transform the input that has been broadcast to it into an output. When an input arrives each of it is purveyed with a specific weight, a bias value is added to this to obtain an output. The output is in the form of a zero or one. If the output predicted stands to be true then the respective input value is rewarded by incrementing its weight otherwise, it is punished by decrementing its weight. We train our neural network using TensorFlow that works on regression.

Chapter 2

LITERATURE SURVEY

- Jiman Kim [1], proposed a project that deals with a sequential end-to-end transfer learning method to estimate left and right ego lanes directly and separately without any post processing. They redefined a point-detection problem as a region-segmentation problem; as a result, the proposed method is insensitive to occlusions and variations of environmental conditions, because it considers the entire content of an input image during training. An extensive dataset that is suitable for a deep neural network training by collecting a variety of road conditions, annotating ego lanes, and augmenting them systematically was constructed. This proposed method compared to few recent methods in deep learning demonstrated improved accuracy and stability on input variations. Their approach does not involve post processing, and is therefore flexible to change of target domain.

- Zhilu Chen and Xinming Huang [2], proposed an end-to-end learning approach to obtain a proper steering angle to maintain the car in the lane. The convolutional neural network (CNN) model here takes raw image frames as input and outputs the steering angles accordingly. The model is trained and evaluated using the comma.ai dataset, which contains the front view image frames and the steering angle data captured when driving on the road. Unlike the traditional approach that manually decomposes the autonomous driving problem into technical components such as lane detection, path planning and steering control, the end-to-end model can directly steer the vehicle from the front view camera data after training. Thus their approach learns how to keep in lane from human driving data. It also presents the discussion of the end-to-end approach employed and its limitations.

- Mariusz Bojarski et al [3], proposed information regarding the training of a Convolutional Neural Network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. Their end-to-end approach with minimum training data from humans provides a system that learns to drive in traffic on local roads with or without lane markings and on highways. It also operates in areas with unclear visual guidance such as in parking lots and on unpaved roads. Their system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. Compared to explicit decomposition of the problem, such as lane marking detection, path planning, and control, this end-to-end system optimizes all processing steps simultaneously. Thus the proposed system will lead to better performance and smaller systems because the internal components self-optimize to maximize overall system performance, instead of optimizing human-selected intermediate criteria, e. g., lane detection.

- Lin Li, Yanheng Liu, et al [4], proposed a new driver model based on human behaviour dynamics for autonomous cars, which allows driverless cars to move appropriately in accordance to the behavioural features of driver owners. This model is established through analysing drivers' various properties, e.g. gender, age, driving experience, personality, and emotion. These attributes collectively determine all the actions occurred during the driving process. Their proposed model is validated by the hardware-in-loop simulator and real driving experiment. From the perspective of human dynamics, this approach introduces the Theory of Planned Behaviour (TPB) into modelling driver for autonomous cars. The driving process is divided into four discrete actions, i.e. acceleration/ deceleration, direction turning, lane change, and light change and the differences of these four actions during the navigation are realized.

- Chenyi Chen et al [5], proposed a direct perception approach to estimate the affordance for driving. The two major paradigms for vision-based autonomous driving systems are mediated perception approaches that parse an entire scene to make a driving decision, and behaviour reflex approaches that directly map an input image to a driving action by a

regressor while the one paper proposed by then is the third paradigm. The idea put forward is to map an input image to a small number of key perception indicators that directly relate to the affordance of a road/traffic state for driving. This representation provides a set of compact yet complete descriptions of the scene to enable a simple controller to drive autonomously. Falling in between the two extremes of mediated perception and behaviour reflexes, the direct perception representation proposed in this paper provides the right level of abstraction. The demonstration is shown by training a deep Convolutional Neural Network thus proving that this model can work well to drive a car in a very diverse set of virtual environments.

Chapter 3

SYSTEM ANALYSIS

3.1 Proposed System

We have proposed a system to handle the software aspect of the self-driving cars. We use Convolutional Neural Network to train our model using the concept of regression. We simulate the car in a computer video game for training and testing of the model. The input to the system is generated by manually playing the game for several hours to build a data set. This data set is used to train our CNN. The network relates the steering angle of the car with its surroundings. This knowledge is further used to take decisions when the car is run on its own.

Advantages of proposed system

- Convolutional Neural Networks make use of filters to assess the frames. This improves the efficiency of the system by removing the unwanted parts of the frame which are irrelevant for making driving decisions.
- This system does not require the presence of exact road markings and sign boards.
- Does not require decomposition of the process into several parts such as lane detection, path planning and steering control. It can directly steer the vehicle from the front view camera data after training. It learns how to keep in lane from human driving data.

Chapter 4

SYSTEM REQUIREMENTS

4.1 Hardware Requirements

Processor	Intel core 2 Quad CPU Q6600 @ 2.40Ghz (4CPUs) / AMD Phenom 9850 Quad core processors (4CPUs) @ 2.50Ghz
RAM	4GB

Table 4.1 Hardware requirements

4.2 Software Requirements

- Operating System: Windows 8.1(64 bit) or above. It's easy to run and simulate high graphics game in Windows OS.
- Drivers: NVIDIA/ AMD graphic Drivers
- Programming Language: Python
- Open CV
- TensorFlow
- Pywin32
- Numpy
- Pandas

Chapter 5

SYSTEM DESIGN

5.1 Introduction

Purpose

System design is a blueprint of the solution for a system. Design of the system can be defined as the process of applying various techniques and principles for the purpose of defining a process are a system in sufficient details to permit its physical realization. System design is concerned with how the system functionalities must be provided by the different components of the system. Thus, system design is a “how to” approach to the creation of a new system.

Scope

The scope of this design document is to achieve the features of the system such as pre-processing the image, feature extraction and prediction of steering angle.

5.2 Network architecture

Architecture focuses on looking at the system as a combination of many different components, and how they interact with each other to produce the desired result. The focus is on identifying components or subsystems and how they connect. In other words, focus is on what major components are needed.

The network architecture consists of the following phases.

Pre-Processing: The pre-processing is a series of operations performed on the generated input frames. It essentially resizes and enhances the frame making it suitable for training. It primarily includes converting the RGB image frames into greyscale and resizing it to a lower resolution

(80x60). The other operations included in this phase are noise filtering, smoothing and sharpening. This phase is important for making the training and prediction process faster.

Normalization: The second phase in the architecture performs image normalization. Normalization done prior to the training process, is crucial to obtain good results as well as fasten significantly the calculations. This phase ensures that all the inputs are at a comparable range.

Convolution: This phase includes a number of convolutional layers which are designed to perform feature extraction. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. A weight matrix called filter slides over the input image to produce a feature map. The convolution of another filter (with the green outline), over the same image gives a different feature map. CNN learnsthe values of these filters on its own during the training process

Pooling: Spatial Pooling (also called subsampling or down sampling) reduces the dimensionality of each feature map but retains the most important information. To rescale a large image, one natural approach is to aggregate statistics of these features at various locations. Spatial Pooling can be of different types: Max, Average, Sum etc. These summary statistics are much lower in dimension (compared to using all extracted features) and improves results.
Max pooling: It computes the max value of a particular feature over a region of the image.
Mean pooling: It computes the mean value of a particular feature over a region of the image.

Fully connected layers: The frames then pass through a number of fully connected layers, leading to a final output control value which is the inverse-turning radius. The fully connected layers are designed to function as a controller for steering.

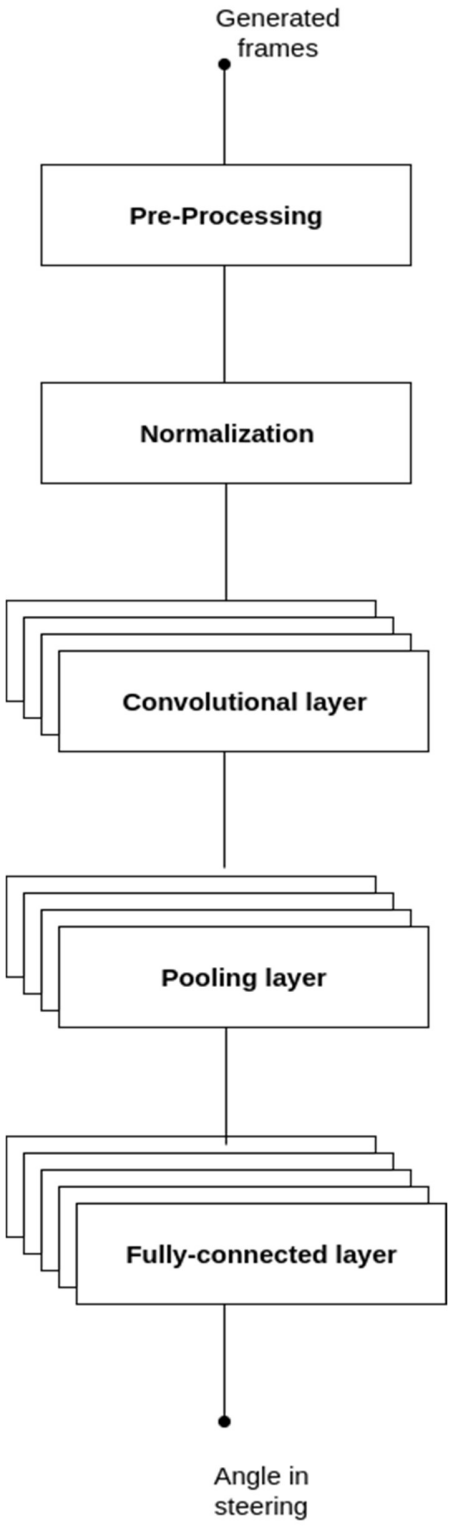

Fig 5.1 Network Architecture

Chapter 6

DETAILED SYSTEM DESIGN

6.1 Use case diagram

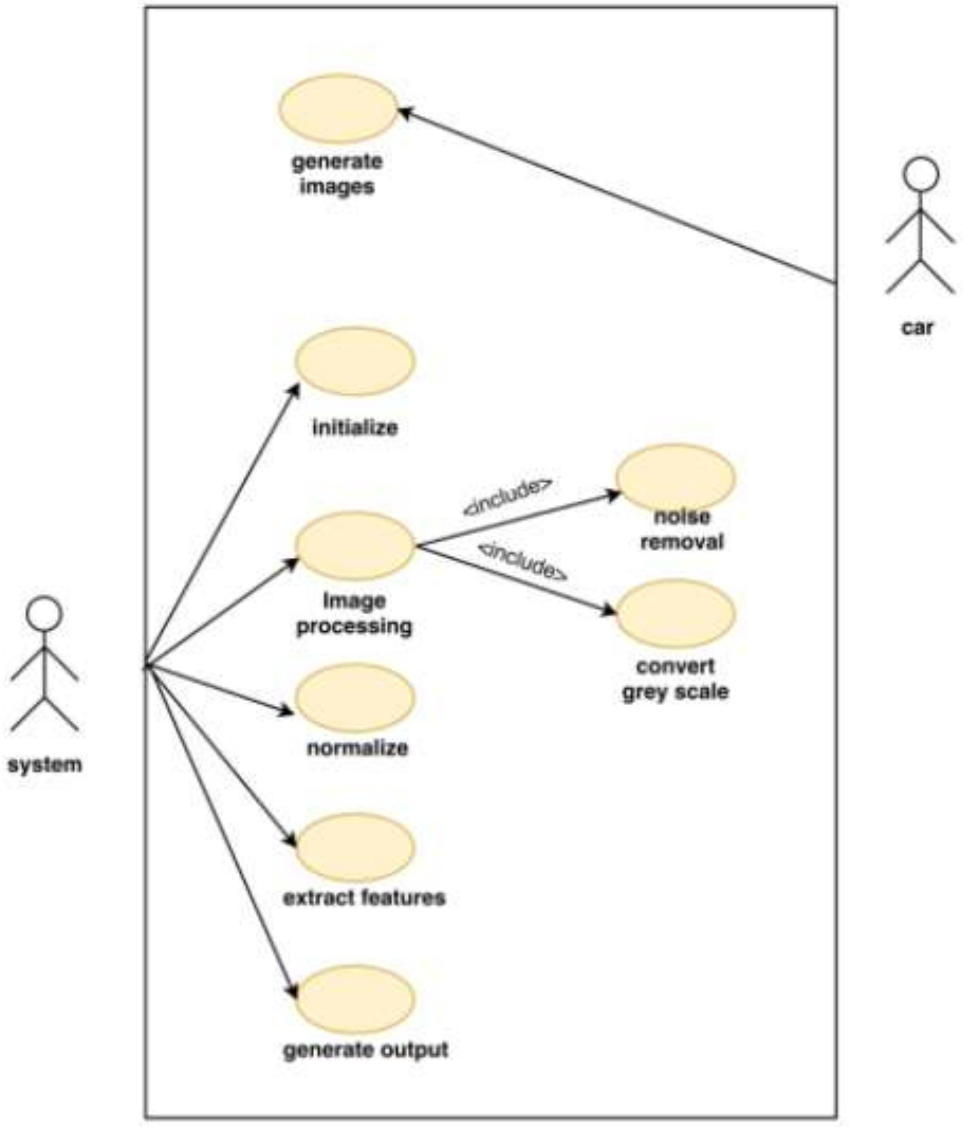

Fig 6.1 Use case diagram

A use case diagram in the Unified Modelling Language (UML) is a type of behavioral diagram defined by and created from a use case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals, and any dependencies between those use cases.

The above figure 6.1 depicts the actions performed by the system and the car. Most of the activities are performed by the system. Initialization, image processing, normalization, extraction of features and generation of output are performed by the system. The car generates the images and the screenshots of the path in front of it which are used to train the model.

6.2 Swimlane diagram

A swim lane (or swim lane diagram) is a visual element used in process flow diagrams, or flowcharts that visually distinguishes job sharing and responsibilities for sub-processes of a business process. Swim lanes may be arranged either horizontally or vertically.

The jobs in our system are majorly shared among three components, car, pre-processing and prediction. The car generates the frames which depict the situation in front of it. In the pre-processing section, filtering, noise-removal, grayscale conversion and normalization are applied to the images sent by the car. The prediction component extracts features from this data and predicts the steering angle.

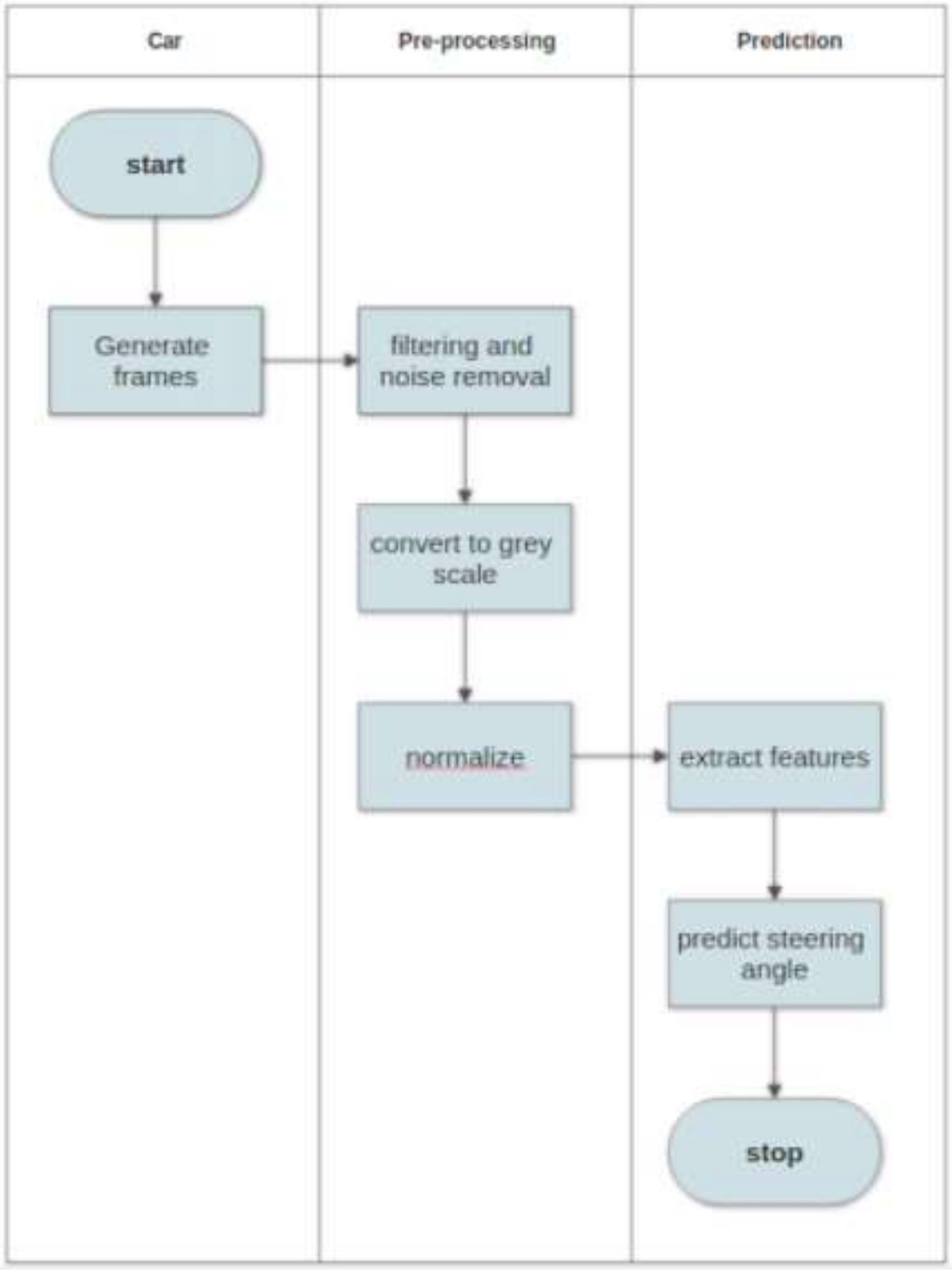

Fig 6.2 Swimlane diagram

6.3 Activity diagram

Activity diagram describes the flow of control in a system. So, it consists of activities and links. The flow can be sequential, concurrent or branched.

Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.

Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.

Fig 6.3 Activity diagram

Chapter 7

IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system. The most critical stage is achieving a successful system and in giving confidence on new system for the users, that it will work efficient and effectively.

It involves careful planning, investigating the current system, and its constraints on implementation, design of methods to achieve the changeover, an evaluation of changeover methods.

The implementation process started with preparing a plan for the implementation of the system. According to this plan, discussion has been made regarding the equipment, resources and test activities to be performed. Thus, a clear plan was prepared for activities.

7.1 Module Implementation and related concepts

7.1.1 Python

Python is a widely used high-level programming language for general-purpose programming. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly braces or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems.

7.1.2 NumPy

NumPy is a fundamental package for scientific computing in python programming language. It is a Python library that provides multidimensional array object, various derived objects such as masked arrays and matrices, and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences.NumPy fully supports an object-oriented approach, starting, once again, with ndarray. For example, ndarray is a class, possessing numerous methods and attributes. Many of its methods mirror functions in the outer-most NumPy namespace, giving the programmer complete freedom to code in whichever paradigm she prefers and/or which seems most appropriate to the task at hand.

7.1.3 Win32 API

The Windows API, informally WinAPI, is Microsoft's core set of application programming interfaces (APIs) available in the Microsoft Windows operating systems. The name Windows API collectively refers to several different platform implementations that are often referred to by their own names (for example, Win32 API). Almost all Windows programs interact with the Windows API.Developer support is available in the form of a software development kit, Microsoft Windows SDK, providing documentation and tools needed to build software based on the Windows API and associated Windows interfaces. The functions provided by Windows API can be grouped into eight categories namely 1) Base services 2) Advance services 3) Graphic device interface 4) User Interface 5)Common Dialogue box library 6) Common control library 7)Windows shell 8) NetworK services.

Win32 is the 32-bit application programming interface (API) for modern versions of Windows. The API consists of functions implemented, as with Win16, in system Dynamic Link Library (DLL). The core DLLs of Win32 are kernel32.dll, user32.dll, and gdi32.dll. Win32 was introduced with Windows NT. The version of Win32 shipped with Windows 95 was initially referred to as Win32c, with c meaning compatibility. This term was later abandoned by Microsoft in favour of Win32.

7.1.4 Win32 GUI

"Win32::GUI is a Win32-platform native graphical user interface toolkit for Perl. Basically, it's an XS implementation of most of the functions found in user32.dll and gdi32.dll, with an object oriented Perl interface and an event-based dialog model that mimic the functionality of visual basic.

7.1.5 Win32ui

Win32ui is a Windows only IDE and GUI framework for Python. It has an integrated debugger, and a rich Python editing environment.

Win32ui is implemented as a 'wrapper' for the Microsoft Foundation Class library. With it, you can use MFC in an interactive, interpreted environment, or write full blown stand-alone applications tightly coupled with the Windows environment. Over 30 MFC objects are exposed, including Common Controls, Property Pages/Sheets, Control/Toolbars, Threads, etc.

Win32ui could almost be considered a sample program for the MFC UI environment. This Python UI environment can be embedded in almost any other application - such as OLE clients/servers, Netscape plugins, as a Macro language etc.

7.1.6 Ctypes

Ctypes is a foreign function library for Python 2.3 and higher versions. It provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python.It is even possible to implement C callback functions in pure Python.ctypes also includes a code generator tool chain which allows automatic creation of library wrappers from C header files. ctypes works on Windows, Mac OS X, Linux, Solaris, FreeBSD and other systems.

7.1.7 Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. The name is derived from “panel data”,an econometrics term for multi-dimensional structured datasets.

Pandas is a game changer when it comes to data analysis with python and it is the most preferred tool in data wrangling. Pandas take data in CSV or TSV or SQL database format and creates a python object with rows and columns called “dataframes” . It is open source and free to use is another advantage of pandas. We can find the maximum, minimum, mean, sort the values, select the values, replace a value, find a particular value in the dataframe using pandas commands.

7.1.8 Tensor Flow

Tensor Flow is an open source software library for machine learning across a range of tasks, and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. It is currently used for both research and production at Google products, often replacing the role of its closed-source predecessor, DistBelief. TensorFlow was originally developed by the Google Brain team for internal Google use before being released under the Apache 2.0 open source license on November 9, 2015.

While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, and mobile computing platforms including Android and iOS.

7.1.9 Tflearn

TF.Learn is a high-level Python module for distributed machine learning inside TensorFlow. It provides an easy-to-use interface to simplify the process of creating, configuring, training, evaluating, and experimenting a machine learning model. TF.Learn integrates a wide range of state-of art machine learning algorithms built on top of TensorFlow’s low level APIs for small to large-scale supervised and unsupervised problems. This module focuses on bringing

machine learning to non-specialists using a general-purpose high-level language as well as researchers who want to implement, benchmark, and compare their new methods in a structured environment.

Tflearn fully utilizes python's object oriented characteristics. Similar to libraries like Pandas, a high-level DataFrame module is included in TFLearn to facilitate many common data reading/parsing tasks from various resources such as tensorflow. It also includes functions like FeedingQueueRunner to fetch data batches and put them in a queue so training and data feeding can be performed asynchronously in different threads to avoid wasting a lot of time waiting for data batches to get fetched. This is very useful especially in case of using virtual GPU.

7.2 Algorithms Used

7.2.1 Canny Edge Detection Algorithm

The Canny Edge detector was developed by John F. Canny in 1986. Also known to many as the optimal detector, Canny algorithm aims to satisfy three main criteria:

- **Low error rate:** Meaning a good detection of only existent edges.
- **Good localization:** The distance between edge pixels detected and real edge pixels have to be minimized.
- **Minimal response:** Only one detector response per edge.

Steps

1. Filter out any noise. The Gaussian filter is used for this purpose. An example of a Gaussian kernel of size = 5 that might be used is shown below:

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2. Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:

    a. Apply a pair of convolution masks (in x and y directions):

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

    b. Find the gradient strength and direction with:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

    The direction is rounded to one of four possible angles (namely 0, 45, 90 or 135)

3. Non-maximum suppression is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.

4. Double Thresholding:

    - Noise and color are distinguished using a certain threshold
    - Edge pixels stronger than the high threshold are marked strong and those edge pixels weaker than the low threshold are marked weak.

5. Hysteresis: The final step. Canny does use two thresholds (upper and lower):

    a. If a pixel gradient is higher than the upper threshold, the pixel is accepted as an edge

    b. If a pixel gradient value is below the lower threshold, then it is rejected.

c. If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the upper threshold.

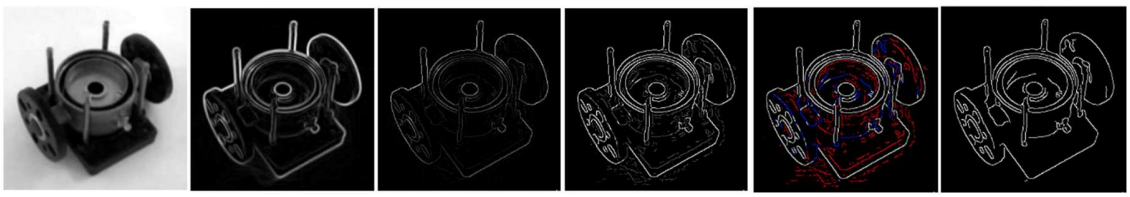

Fig 7.1 An image as it undergoes the steps in Canny edge detection algorithm

7.2.1 Alexnet

This architecture was one of the first deep networks to push ImageNet Classification accuracy by a significant stride in comparison to traditional methodologies. It is composed of 5 convolutional layers followed by 3 fully connected layers.

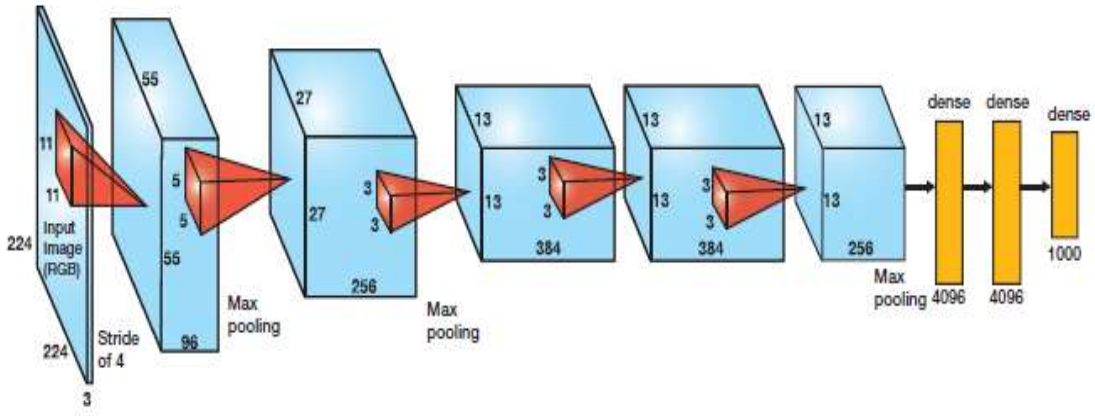

Fig 7.2 Alexnet architecture

AlexNet, proposed by Alex Krizhevsky, uses ReLu(Rectified Linear Unit) for the non-linear part, instead of a Tanh or Sigmoid function which was the earlier standard for traditional neural networks. ReLu is given by:

$f(x) = \max(0,x)$

The advantage of the ReLu over sigmoid is that it trains much faster than the latter because the derivative of sigmoid becomes very small in the saturating region and therefore the updates to the weights almost vanish. This is called vanishing gradient problem.

In the network, ReLu layer is put after each and every convolutional and fully-connected layers(FC).

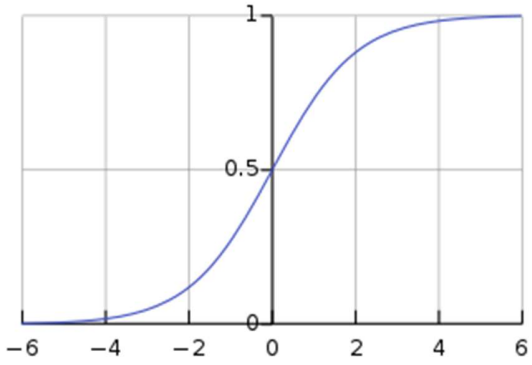

Fig 7.3 ReLU

Another problem that this architecture solved was reducing the over-fitting by using a Dropout layer after every FC layer. Dropout layer has a probability,(p), associated with it and is applied at every neuron of the response map separately. It randomly switches off the activation with the probability p, as can be seen in figure 5.

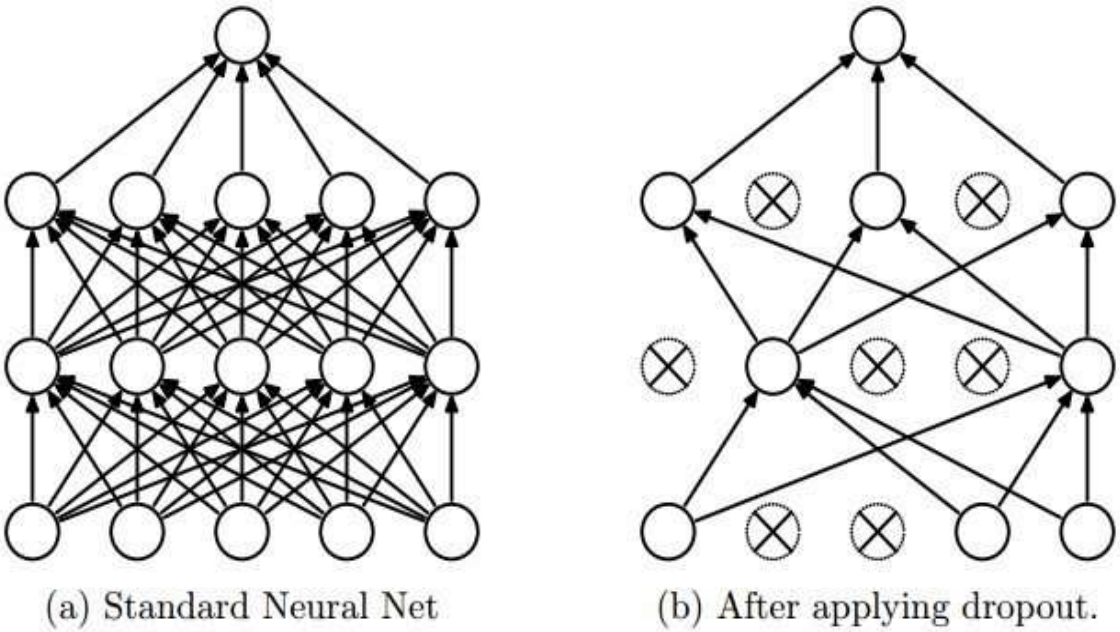

Fig 7.4 Droupout in Alexnet

Why does dropout work?

The idea behind the dropout is similar to the model ensembles. Due to the dropout layer, different sets of neurons which are switched off, represent a different architecture and all these different architectures are trained in parallel with weight given to each subset and the summation of weights being one. For n neurons attached to DropOut, the number of subset architectures formed is 2^n. So it amounts to prediction being averaged over these ensembles of models. This provides a structured model regularization which helps in avoiding the over-fitting. Another view of DropOut being helpful is that since neurons are randomly chosen, they tend to avoid developing co-adaptations among themselves thereby enabling them to develop meaningful features, independent of others.

7.3Steps involved in implementation

7.3.1 Grabscreen

We use this function to obtain a screenshot of the screen.

This function uses the various methods available in the win32ui, win32gui, win32con and win32api.These methods provide calls to various APIs which take the value of each pixel on the screen and convert into an image format.

This function does not take any argument and returns an image.

```
import cv2
import numpy as np
import win32gui, win32ui, win32con, win32api

def grab_screen(region=None):

    hwin = win32gui.GetDesktopWindow()

    if region:
            left,top,x2,y2 = region
            width = x2 - left + 1
            height = y2 - top + 1
    else:
        width = win32api.GetSystemMetrics(win32con.SM_CXVIRTUALSCREEN)
        height = win32api.GetSystemMetrics(win32con.SM_CYVIRTUALSCREEN)
        left = win32api.GetSystemMetrics(win32con.SM_XVIRTUALSCREEN)
        top = win32api.GetSystemMetrics(win32con.SM_YVIRTUALSCREEN)

    hwindc = win32gui.GetWindowDC(hwin)
    srcdc = win32ui.CreateDCFromHandle(hwindc)
    memdc = srcdc.CreateCompatibleDC()
    bmp = win32ui.CreateBitmap()
    bmp.CreateCompatibleBitmap(srcdc, width, height)
    memdc.SelectObject(bmp)
    memdc.BitBlt((0, 0), (width, height), srcdc, (left, top), win32con.SRCCOPY)

    signedIntsArray = bmp.GetBitmapBits(True)
    img = np.fromstring(signedIntsArray, dtype='uint8')
    img.shape = (height,width,4)

    srcdc.DeleteDC()
    memdc.DeleteDC()
    win32gui.ReleaseDC(hwin, hwindc)
    win32gui.DeleteObject(bmp.GetHandle())

    return cv2.cvtColor(img, cv2.COLOR_BGRA2RGB)
```

7.3.2 Processing image

The image that is obtained by taking the screenshot of the screen is a colored image.In a colored image, each pixel is represented by an array of three numbers each ranging from 0 to 255.Hence, each pixel occupies 32 bits of memory. But, for our analysis, the distance of each object from our vehicle is necessary.

The color of the object does not matter.Hence, we convert the image to grayscale so that each pixel can be represented using just 8 bits.The size of our data set will reduce three times by performing this step.

```
def process_img(original_image):
    processed_img = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
    processed_img = cv2.Canny(processed_img, threshold1=200, threshold2=300)
    vertices = np.array([[10,500],[10,300], [300,200],
                         [500,200], [800,300], [800,500]], np.int32)
    processed_img = roi(processed_img, [vertices])
    return processed_img
```

Fig 7.5 Colour image

Fig 7.6 Image after converting to grayscale

After converting the image to grayscale, we use the canny edge detection algorithm to detect the edges from the image.

This step removes all the noise from the image and returns only the edges that are necessary for our processing.

Fig 7.7 Image before applying canny edge detection

Fig 7.8 Image after applying canny edge detection

7.3.3 Region of Interest

The image that is obtained by capturing the screen contains many elements.

To take a driving decision, we only require the road in front of the vehicle and other obstacles on the road.Elements which are present outside the road and in the sky are not required for our analysis.

To eliminate these objects from increasing the size of our data set and adulterating our decision making process, we define a function, region of interest.

This function basically cuts off the image into two parts and eliminates the unnecessary objects.It takes an image and a set of vertices as input, performs the necessary processing and returns the cropped image as output.

```
def roi(img, vertices):
    mask = np.zeros_like(img)
    cv2.fillPoly(mask, vertices, 255)
    masked = cv2.bitwise_and(img, mask)
    return masked
```

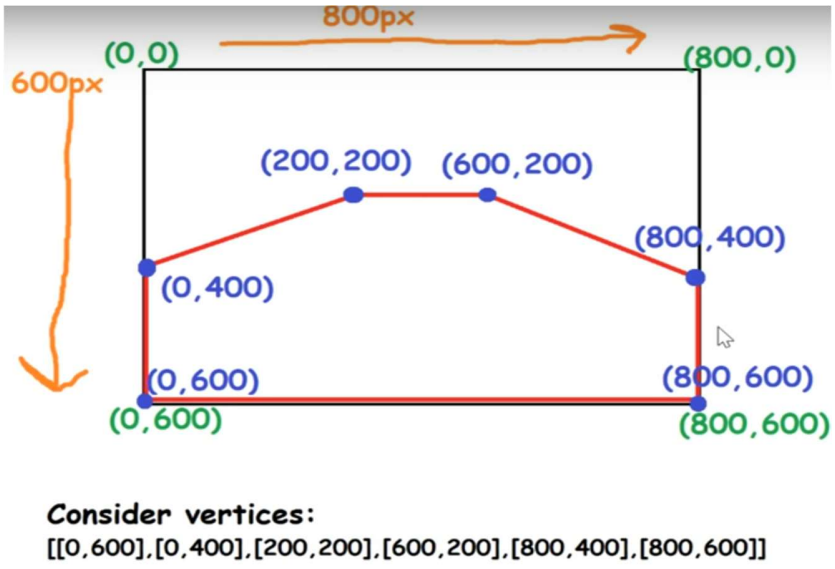

Fig 7.9 Region of Interest on the screen

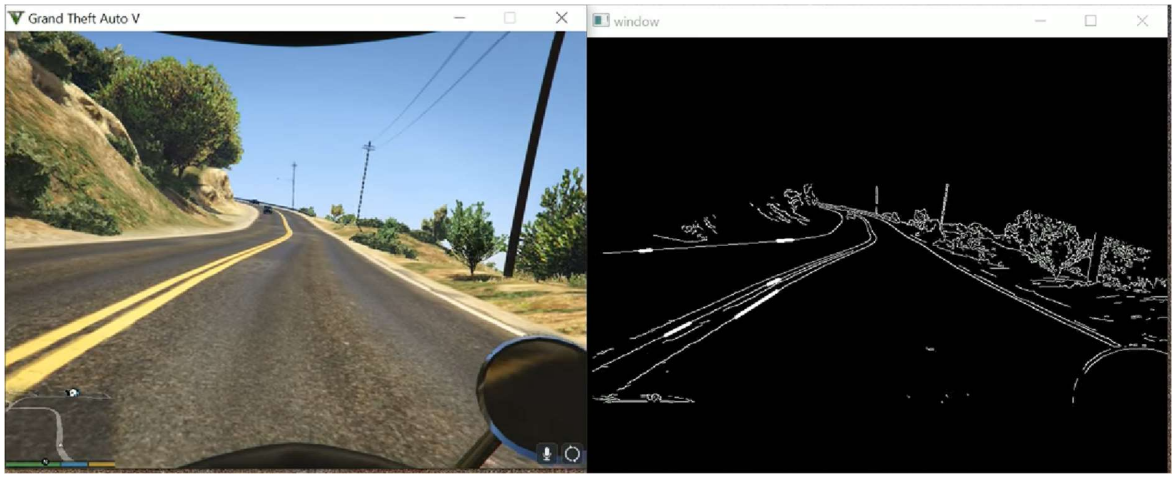

Fig 7.10 Image before and after applying ROI filter

7.3.4 Line Detection

cv2.HoughLinesP(pro_img, w, np.pi/180, x, y, z)

- The lines are detected from the given image using Houghline transform mechanism
- Parameters :
    - Pro_img - image on which the function is to be applied
    - w - Rho value
    - x - Threshold
    - y - Minimum length
    - z - Maximum gap

Hough Line transformation is used to detect any shape. This method represents any shape in mathematical form. It has two parameters associated with it:

- Minimum line length - This is used to specify that value below which line segments are rejected
- Maximum line gap - Maximum gap that is permissible between lines to treat them as a single line

Fig 7.11 Image before detecting lines

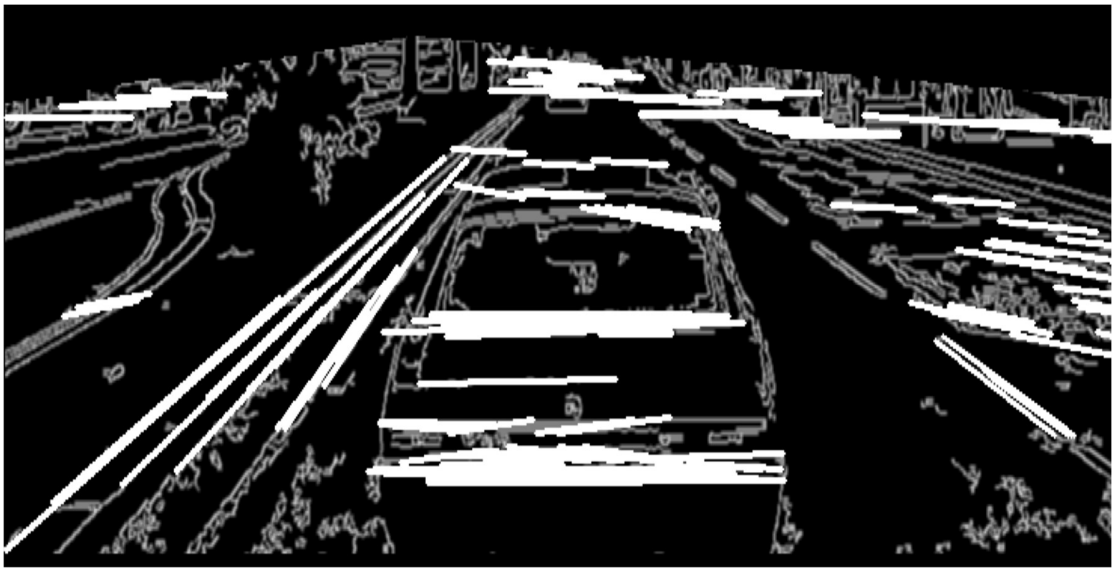

Fig 7.12 Image after detecting lines

```
lines = cv2.HoughLinesP(processed_img, 1,
                        np.pi/180, 180, 20, 15)
draw_lines(processed_img,lines)
return processed_img
```

The houghlines function returns an array in which each element is a set of two geometrical coordinates representing the end points of each line. This array is then passed to the cv2.line function in the draw_lines() method which draws these lines as shown on the screen.

```
def draw_lines(img, lines):
    try:
        for line in lines:
            coords = line[0]
            cv2.line(img, (coords[0],coords[1]), (
                coords[2],coords[3]), [255,255,255], 3)
    except:
        pass
```

7.3.5 Lane Detection

Not all the lines that are detected from the draw_lines() function are lanes. To specifically find the lanes present in the image, we use the function draw_lanes().

We take the lines obtained from the draw_lines() function as the input to this function. Here, we filter out the unwanted lines by first determining the length of each line. The lines which are smaller than a minimum value are neglected.

From the remaining lines, we calculate the slopes of each line by using the formula (y1-y2)/(x1-x2). Only the pairs of lines whose slopes are almost equal and are of opposite signs represent a lane. All such pairs are taken into consideration. We then draw these lines over the image using its coordinates to show the lanes.

```
def draw_lanes(img, lines, color=[0, 255, 255], thickness=3):
    try:
        ys = []
        for i in lines:
            for ii in i:
                ys += [ii[1],ii[3]]
        min_y = min(ys)
        max_y = 600
        new_lines = []
        line_dict = {}
        for idx,i in enumerate(lines):
            for xyxy in i:
                x_coords = (xyxy[0],xyxy[2])
                y_coords = (xyxy[1],xyxy[3])
                A = vstack([x_coords,ones(len(x_coords))]).T
                m, b = lstsq(A, y_coords)[0]
                x1 = (min_y-b) / m
                x2 = (max_y-b) / m
                line_dict[idx] = [m,b,[int(x1), min_y, int(x2), max_y]]
                new_lines.append([int(x1), min_y, int(x2), max_y])
        final_lanes = {}
        for idx in line_dict:
            final_lanes_copy = final_lanes.copy()
            m = line_dict[idx][0]
            b = line_dict[idx][1]
            line = line_dict[idx][2]
            if len(final_lanes) == 0:
                final_lanes[m] = [ [m,b,line] ]
            else:
                found_copy = False
                for other_ms in final_lanes_copy:
                    if not found_copy:
                        if abs(other_ms*1.2) > abs(m) > abs(other_ms*0.8):
                            if abs(final_lanes_copy[other_ms][0][1]*1.2) > abs(b) >
                            abs(final_lanes_copy[other_ms][0][1]*0.8):
                                final_lanes[other_ms].append([m,b,line])
                                found_copy = True
                                break

                    else:
                        final_lanes[m] = [ [m,b,line] ]
    line_counter = {}
    for lanes in final_lanes:
        line_counter[lanes] = len(final_lanes[lanes])
    top_lanes = sorted(line_counter.items(), key=lambda item: item[1])[::-1][:2]
    lane1_id = top_lanes[0][0]
    lane2_id = top_lanes[1][0]
    def average_lane(lane_data):
        x1s = []
        y1s = []
        x2s = []
        y2s = []
        for data in lane_data:
            x1s.append(data[2][0])
            y1s.append(data[2][1])
            x2s.append(data[2][2])
            y2s.append(data[2][3])
        return int(mean(x1s)), int(mean(y1s)), int(mean(x2s)), int(mean(y2s))
    l1_x1, l1_y1, l1_x2, l1_y2 = average_lane(final_lanes[lane1_id])
    l2_x1, l2_y1, l2_x2, l2_y2 = average_lane(final_lanes[lane2_id])
    return [l1_x1, l1_y1, l1_x2, l1_y2], [l2_x1, l2_y1, l2_x2, l2_y2]
except Exception as e:
    print(str(e))
```

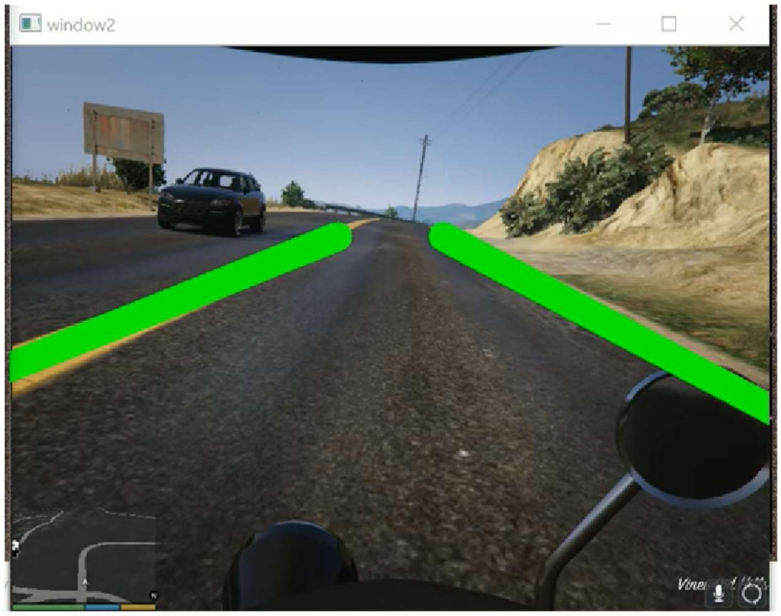

Fig 7.13 Detecting lanes on the road

7.3.6 Creating training data

The data set required for the training of our model is generated using this function. To generate this data, we manually drive the car in the game. While we are manually driving the car, screenshots of the screen are taken continuously. We do this until we generate 1,00,000 images.

The data set is in the form of a numpy array in which each element is a tuple. The first element of the tuple is the image. The second element is an array which denotes the key that was being pressed when that image was taken. All these tuples are stored in a file named training_data.

```
while(True):

    if not paused:
        # 800x600 windowed mode
        screen = grab_screen(region=(0,40,800,640))
        last_time = time.time()
        screen = cv2.cvtColor(screen, cv2.COLOR_BGR2GRAY)
        screen = cv2.resize(screen, (160,120))
        # resize to something a bit more acceptable for a CNN
        keys = key_check()
        output = keys_to_output(keys)
        training_data.append([screen,output])
```

7.3.6 Extracting the list of keys that are pressed

To extract the list of keys that are pressed when a particular screenshot is taken, we use this function named key_check(). We make use of the GetAsyncKeyState() method available in the win32api package to accomplish this.

```
def key_check():
    keys = []
    for key in keyList:
        if win32api.GetAsyncKeyState(ord(key)):
            keys.append(key)
    return keys
```

The keys_to_output() function returns the list of keys in the format required for creating the dataset. As we use only the ‘A’, ‘W’, and ‘D’ keys to control the car in the game, we consider the state of only these keys.

```
def keys_to_output(keys):
    '''
    Convert keys to a ...multi-hot... array
    [A,W,D] boolean values.
    '''
    output = [0,0,0]

    if 'A' in keys:
        output[0] = 1
    elif 'D' in keys:
        output[2] = 1
    else:
        output[1] = 1
    return output
```

7.3.7 Alexnet

We employ Tensorflow TFlearn package to create an Alexnet, a type of CNN which contains five Convolutional layers and three fully connected layers as shown in fig.(2). The design of the convolutional layers is to perform feature extraction. It preserves the spatial relationship between the pixels by learning image features using small squares of input data. A weight matrix called filter slides over the input image produces a feature map. The network learns values of these filters on its own during the training process. The design of the fully connected layers is to function as a controller for steering where the image frames pass through these layers, leading to a final output control value which is an inverse turning radius. Rectified Linear Unit(ReLU) [6],[7] is applied after every convolutional and fully connected layer. ReLU is a function first introduced by Hahnloser et al. It was then stated by Nair et al that ReLU is an effective activation for use in neural network as well. ReLU function is given by:

$f(x)=\max(0,x)$

Dropout is applied before the first and the second fully connected layer. Dropout helps in removing complex co-adaption. Removal of complex co-adaption implies training node in a

neural network with a randomly selected sample of other nodes. This makes the node more robust and drive it towards creating useful features, without relying much on other nodes. Overlap pooling is used to reduce the size of the network.

Further, we use TensorFlow object detection API to detect other vehicles on road and to determine the distance from them. They use pixels from the input matrix data set as predictors and predict which operation the vehicle has to perform (turn left/ turn right or / straight).

```
def alexnet(width, height, lr):
    network = input_data(shape=[None, width, height, 1], name='input')
    network = conv_2d(network, 96, 11, strides=4, activation='relu')
    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)
    network = conv_2d(network, 256, 5, activation='relu')
    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)
    network = conv_2d(network, 384, 3, activation='relu')
    network = conv_2d(network, 384, 3, activation='relu')
    network = conv_2d(network, 256, 3, activation='relu')
    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)
    network = fully_connected(network, 4096, activation='tanh')
    network = dropout(network, 0.5)
    network = fully_connected(network, 4096, activation='tanh')
    network = dropout(network, 0.5)
    network = fully_connected(network, 3, activation='softmax')
    network = regression(network, optimizer='momentum',
                         loss='categorical_crossentropy',
                         learning_rate=lr, name='targets')

    model = tflearn.DNN(network, checkpoint_path='model_alexnet',
                        max_checkpoints=1, tensorboard_verbose=0,
                        tensorboard_dir='log')

    return model
```

7.3.8 Training the model

TensorFlow is an open source programming library made by Google which is utilized to configure, construct and train profound learning models. The library of TensorFlow contains

various powerful algorithms to do numerical computations, which in itself doesn't seem all too special, but achieve these computations with data flow graphs. In these graphs, edges depict the data while the nodes illustrate the mathematical operations, usually are multidimensional tensors and/or data arrays, that are conveyed between these edges. The operations which neural networks perform on multidimensional data arrays or tensors is literally a flow of tensors, hence the name "TensorFlow".

We utilize TensorFlow in our model to learn how to automatically spot a complex pattern or image. Depending on the images recognised the system takes the best possible decision independently. Further, we construct a computational graph that consists of nodes represents an operation and edges which represents multi-dimensional data arrays using TensorFlow. Subsequent to defining the operations we set up a TensorFlow session in order to perform calculations on the defined graph.

We divide the dataset that was generated into parts, the training data and testing data. We then train the neural network using the training data.

We use the model.fit() function from the tflearn package to train the data. The syntax of the function is:

fit (X_inputs, Y_targets, n_epoch=10, validation_set=None, show_metric=False, batch_size=None, shuffle=None, snapshot_epoch=True, snapshot_step=None, excl_trainops=None, validation_batch_size=None, run_id=None, callbacks=[])

This function takes many arguments:

- **X_inputs**: array, list of array (if multiple inputs) or dict (with inputs layer name as keys). Data to feed to train model.
- **Y_targets**: array, list of array (if multiple inputs) or dict (with estimators layer name as keys). Targets (Labels) to feed to train model.
- **n_epoch**: int. Number of epoch to run. Default: None.

- **validation_set**: tuple. Represents data used for validation. tuple holds data and targets (provided as same type as X_inputs and Y_targets). Additionally, it also accepts float (<1) to performs a data split over training data.
- **show_metric**: bool. Display or not accuracy at every step.
- **batch_size**: int or None. If int, overrides all network estimators 'batch_size' by this value. Also overrides validation_batch_size if int, and if validation_batch_size is None.
- **validation_batch_size**: int or None. If int, overrides all network estimators 'validation_batch_size' by this value.
- **shuffle**: bool or None. If bool, overrides all network estimators 'shuffle' by this value.
- **snapshot_epoch**: bool. If True, it will snapshot model at the end of every epoch. (Snapshot a model will evaluate this model on validation set, as well as create a checkpoint if 'checkpoint_path' specified).
- **snapshot_step**: int or None. If int, it will snapshot model every 'snapshot_step' steps.
- **excl_trainops**: list of TrainOp. A list of train ops to exclude from training process (TrainOps can be retrieve through tf.get_collection_ref(tf.GraphKeys.TRAIN_OPS)).
- **run_id**: str. Give a name for this run. (Useful for Tensorboard).
- **callbacks**: Callback or list. Custom callbacks to use in the training life cycle

```
WIDTH = 160
HEIGHT = 120
LR = 1e-3
EPOCHS = 10
MODEL_NAME = 'pygta5-car-fast-{}-{}-{}-epochs-300K-data.model'.format
             (LR, 'alexnetv2',EPOCHS)

model = alexnet(WIDTH, HEIGHT, LR)

hm_data = 22
for i in range(EPOCHS):
    for i in range(1,hm_data+1):
        train_data = np.load('training_data-{}-balanced.npy'.format(i))

        train = train_data[:-100]
        test = train_data[-100:]

        X = np.array([i[0] for i in train]).reshape(-1,WIDTH,HEIGHT,1)
        Y = [i[1] for i in train]

        test_x = np.array([i[0] for i in test]).reshape(-1,WIDTH,HEIGHT,1)
        test_y = [i[1] for i in test]

        model.fit({'input': X}, {'targets': Y}, n_epoch=1, validation_set=(
            {'input': test_x}, {'targets': test_y}),
            snapshot_step=500, show_metric=True, run_id=MODEL_NAME)

        model.save(MODEL_NAME)
```

7.3.9 Test model

Once the model is trained, the model will be ready to be tested. We open the game in one window and then run the script. The program continuously takes screenshots and analyses the image. From its knowledge from the training, the neural networks now takes the appropriate driving decision.

In this function, we send the image as an argument to the model.predict() function. This function returns the set of keys to be pressed for that particular image. This returned value is an array of three elements, each representing the probability of the respective key to be pressed. If the value corresponding to a key crosses a threshold, then that prediction is taken into confidence and the appropriate key is pressed. The threshold for going straight is kept as 0.7 while the threshold for left and right keys is set to 0.75.

```
if not paused:
    screen = grab_screen(region=(0,40,800,640))
    print('loop took {} seconds'.format(time.time()-last_time))
    last_time = time.time()
    screen = cv2.cvtColor(screen, cv2.COLOR_BGR2GRAY)
    screen = cv2.resize(screen, (160,120))

    prediction = model.predict([screen.reshape(160,120,1)])[0]
    print(prediction)

    turn_thresh = .75
    fwd_thresh = 0.70

    if prediction[1] > fwd_thresh:
        straight()
    elif prediction[0] > turn_thresh:
        left()
    elif prediction[2] > turn_thresh:
        right()
    else:
        straight()

keys = key_check()
```

Once a decision is taken on which key is pressed, we call the appropriate functions to implement that decision.

```
def straight():
    PressKey(W)
    ReleaseKey(A)
    ReleaseKey(D)

def left():
    PressKey(W)
    PressKey(A)
    ReleaseKey(D)
    time.sleep(t_time)
    ReleaseKey(A)

def right():
    PressKey(W)
    PressKey(D)
    ReleaseKey(A)
    time.sleep(t_time)
    ReleaseKey(D)
```

We use the functions PressKey() and ReleaseKey() which in turn use the methods defined in the ctypes package to make system calls which press the corresponding keys from the keyboard.

```
def PressKey(hexKeyCode):
    extra = ctypes.c_ulong(0)
    ii_ = Input_I()
    ii_.ki = KeyBdInput( 0, hexKeyCode, 0x0008, 0, ctypes.pointer(extra) )
    x = Input( ctypes.c_ulong(1), ii_ )
    ctypes.windll.user32.SendInput(1, ctypes.pointer(x), ctypes.sizeof(x))

def ReleaseKey(hexKeyCode):
    extra = ctypes.c_ulong(0)
    ii_ = Input_I()
    ii_.ki = KeyBdInput( 0, hexKeyCode, 0x0008 | 0x0002, 0, ctypes.pointer(extra) )
    x = Input( ctypes.c_ulong(1), ii_ )
    ctypes.windll.user32.SendInput(1, ctypes.pointer(x), ctypes.sizeof(x))
```

Chapter 8

TESTING

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects).

8.1 Purpose of testing

There are two fundamental purposes of testing: Verifying procurement specifications and Managing Risk. First, testing is about verifying that what was specified is what was delivered: it verifies that the product (system) meets the functional, performance, design and implementation requirements identified in the procurement specifications. Second, testing is about, managing risks for both the acquiring agency and the system vendor/ developer/ integrator. The testing programme is used to identify when the work has been completed so that the contract can be closed, the vendor paid, and the system shifted by the agency into the warranty and maintenance phase of the project.

8.2 Black Box testing

The technique of testing without having any knowledge of interior working of the application is Black box testing. The tester is obvious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

8.3 White Box testing

White box testing is the detailed investigation of the internal logic and structure of the code. White box testing is also called glass testing or open box testing. In order to performwhite box testing on an application, the tester needs to possess knowledge of the internal working of the code. The tester needs to have a look at the source code and find out which unit of the code is behaving inappropriately.

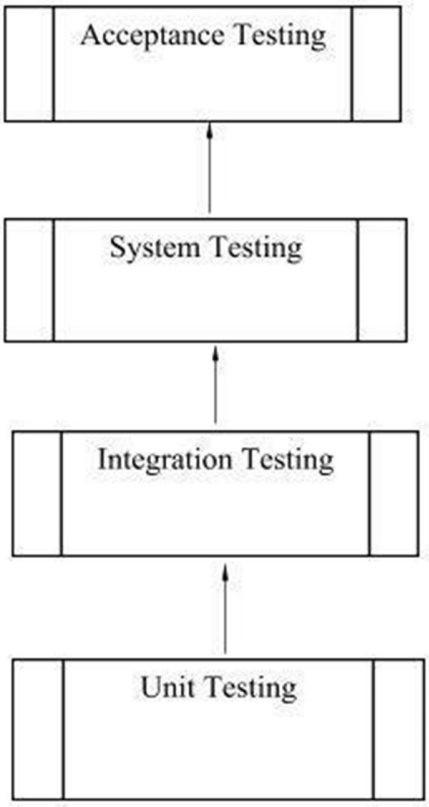

Fig 8.1 Different levels of testing

8.4 Unit Testing

Unit testing focuses on the smallest unit of software design. Smallest unit include the modules which are integrated to produce the final product. The unit testing focuses on the internal logic and the data structures within the boundaries of the component. Test considerations

can be the interface, the local data structures, boundary conditions, independent paths, error handling paths etc.

The modules of this project were tested to verify its working using unit testing and suitable corrections were made to achieve the desired functionality.

Unit testing done on verifying if the lanes are being detected on the screen.

Image type	Result with accuracy
Good quality image with clear lines	Correct label with accuracy >0.9
Low quality image with overlapping lines	Correct label with accuracy <0.6
Good quality image of untrained lanes looking similar to trained lanes	Wrong label with accuracy <0.6
Good quality image of untrained lanes looking distinct from trained lanes	Wrong label with accuracy <0.4
Low quality image untrained lanes	Wrong label with accuracy <0.4

Table 8.1 Unit testing

Similarly, all the modules were tested with various test cases and the results were verified.

8.5 Integration Testing

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined

in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Test Case ID	Objective	Expected result
1	Check whether the screenshots are being grabbed continuously	The number of frames generated per second must be at least 10
2	Check whether the parameters to alexnet are appropriate	The neural network should make predictions with a high confidence
3	Check whether the program is able to give keyboard inputs to the game	The game should accept inputs from the code instead of the keyboard

Table 8.2 Integration testing

Similarly, all the interface links were tested on integration with various scenarios and the results were verified. Various strategies that are used to execute Integration testing are:

- Big Bang approach
- Incremental approach: which is further divided into
    - Top down approach
    - Bottom up approach
    - Sandwich approach

8.6 System testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have passed testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between

the software units that are integrated together (called *assemblages*) or between any of the *assemblages* and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

In our project, we make diligent alterations to the Alexnet parameters to obtain the most efficient model that results in the relatively high accuracy. We have created proliferates as the size of the data frame input increases and on modification of parameters in the Alexnet. Thus, we can state with confidence that the prediction model has a high accuracy when the input frames are comparable to the size of the data set when the parameters are tuned to specific values.

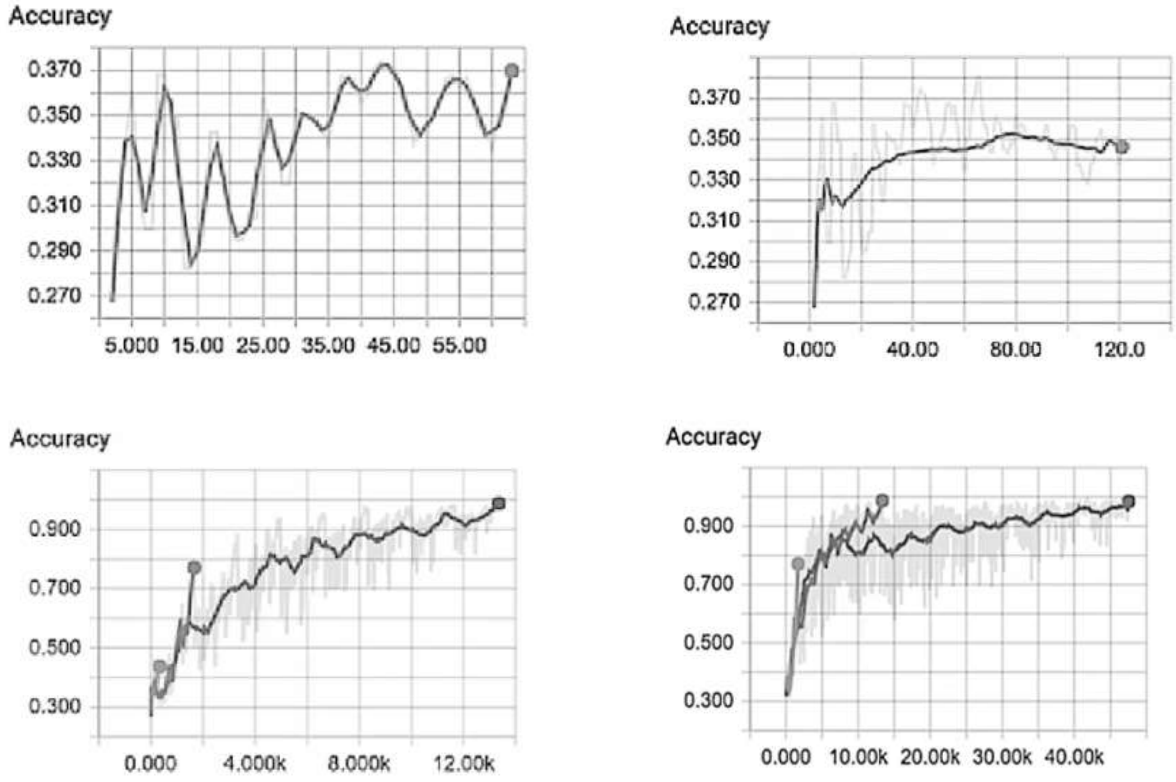

Fig 8.2 Accuracy plot with x-axis representing number of data frames

As shown in the graphs, the accuracy of the graphs increases as we increase the amount of training data. With around 37% accuracy using 60 frames to more than 90% accuracy using 40,000 frames, we continuously increase the accuracy of the system by generating more and more data for the training data set.

8.7 Acceptance testing

In engineering and its various sub disciplines, acceptance testing is a test conducted to determine if the requirements of a specification or contract are met. It may involve chemical tests physical tests, or performance.

In systems engineering it may involve black-box testing performed on system (for example: a piece of software, lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery.

Software developers often distinguish acceptance testing by the system provider from acceptance testing by the customer (the user or client) prior to accepting transfer of ownership. In the case of software, acceptance testing performed by the customer is known as user acceptance testing (UAT), end-user testing, site (acceptance) testing, or field (acceptance) testing.

This system of simulating a self-driving car using a PC Game amusement accomplishes all the functionalities that are proposed. It operates efficiently with high accuracy and decisions taken with a high level of confidence by the neural network.

8.8 Test cases

TC #	Description	Expected Input	Expected Output	Status of execution Pass/fail
1	Processing of captured images	Continuous screenshots captured while the game is running	Grayscale images containing elements only in the region of interest	Pass
2	Detect lines using canny algorithm	Screenshots of the game	The lines in the corresponding image	Pass

3	Lane detection by calculating slopes	Lines detected using canny algorithm	Differentiate lanes which represent the road from the remaining line in the image	Pass
4	Recording the keys to be pressed	Manual input to the keyboard	List of keys being pressed when the corresponding screenshot has been taken	Pass
5	Creating data set	Images and corresponding list of keys	A balanced numpy array containing the data in a format which can be supplied to the neural network	Pass
6	Training the data using alexnet	Parameters to the neural network	High accuracy	Pass
7	Testing the model	The knowledge developed from the training	The car is able to drive itself following the lanes and avoiding obstacles	Pass

Table 8.3 Test cases

CONCLUSION

Through this project, we have introduced the concept of lane switching in self-driving automobiles. We have clarified in detail Alexnet, a type of convolutional Neural Network which is most appropriate for this application. Put into words, the generation of input data sets using OpenCV, the system architecture and the methodologies employed for simulation. Further, a demonstration of the transition in system accuracy with the varying volumes of training data is portrayed through graphical depictions. Thus, we have successfully met with the aim of that project, which is lane switching in self driving cars using convolutional neural networks.

FUTURE ENHANCEMENTS

Robotization in automobiles is a fast developing field in the present world today. To stay aware of the changing scenario and requirements, the system must be updated frequently. Aside from the re-enactment of lane switching in self-driving we have delineated through this paper, there are couple of more areas under this vast field that can be enhanced in future. One such feature would be the capacity of a self-driving vehicle to have the capacity to discover the way of slightest harm if there should arise an occurrence of unavoidable conditions. This would diminish the dangers caused by an extraordinary degree.

Yet, another healthy addition to this project would be the incorporation of maps and GPS which are to be integrated with the current system to enable voice over control. This would stand to provide further automisation in the field of self-driving cars.