A Report submitted in partial fulfilment
of the regulations governing the award of
the Degree of
BSc (Honours) Computer Science
at KDU Penang University College

Project Report

Automove – End-to-End Deep Learning system for Self-Driving Vehicles

Name: **SRIRAM A/L NARAYANAN RAMASAMY**

Student ID: **0188677**

2019

Application Project

Declarations

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and reference to **ALL** sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is **17507**

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertations to being place on the eLearning Portal (Canvas), if deemed appropriate, to allow future students the opportunity to see examples of pas dissertations. I understand that if displayed on eLearning Portal, it would be made available for no longer than five years and that students would be able to print off copies or download it.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other School or from other institutions using this service.

In the event of the service detecting a high degree of similarity between content within the service, this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

SIGNED:

SRIRAM A/L NARAYANAN RAMASAMY

Date: 6th December 2019

Acknowledgement

First and foremost, I would like to thank my project supervisor, Dr J. Joshua Thomas who has given me unconditional support and motivation to develop this system. He has helped me by providing ample amount of knowledge on developing this deep learning system. Besides that, as he has a lot of knowledge in Artificial Intelligence, it has helped me to understand the core structure of the project before developing it. Furthermore, he also assisted me during my research on implementation of this system and provided many useful feedbacks to enhance this system. Moreover, he also follows up regularly on my report documentation which helped me to write the report with in depth information.

I would like to thank Mr. Danny Chen, the lecturer of CCP3024 individual Project for guiding us through the subject on how to prepare documentation and also to schedule the time properly in order to complete the project on time. He had given us some important tricks and suggestions at the end of the semester which helped us in big time during the development stage. Other than that, I would like to thank IT Department of KDU Penang, who gave me the opportunity to use the Predator Orion PC to develop my system. I would like to express my deepest thanks to my mother and grandparents who motivated me from the start to end. They have been my biggest strength throughout my study life. Finally, I would like to thank Hong Leong Foundation who awarded me full scholarship to undergo my undergraduate studies in KDU Penang.

Abstract

Self-Driving Vehicles are one of the rising solutions to improve road safety, passengers' comfort and traffic issues. The United Nations has categorised Malaysia as 30th in the ranking of highest number of fatal accidents with an average of 4.5 causalities per 10,000 vehicles. To develop applications such as Self-Driving Vehicles, advanced computer vision algorithm that demand powerful computers with high-speed processing capabilities are required. In this project, an End-to-End Deep Learning System is developed to navigate the vehicle autonomously to its destination.

There are five important aspects in this system which enables the vehicle to detect the lanes, detect the speed of the vehicle, detect the angle of the road, recognise the objects on the road and predict the steering angle of the vehicle. For lane detection, colour thresholding technique is applied and noises around the lanes are removed on the road. A blue mask is attached on the lane to see the difference between the lanes and the other images on the road. The speed of the vehicle will be calculated using distance / time formula where distance is the distance of the white line and time is the time taken of the vehicle to reach the end point of white line. The speed will be showed in km/h. The angle of the road is measured by using polynomial curve fitting model which find the best-fit function (Curve) along a set of range. The average angle is calculated by using exponential moving average formula. From this, the average angle of each frame is saved into a .txt file to create a new dataset.

For this system, YOLO algorithm is used for object detection. Behavioural cloning is done by training the neural network model using Tensorflow and Keras library. The video dataset is split into images frame by frame. The model is trained on 2937 samples and validated on 1259 samples with 30 epochs. Then the best model is saved as Autopilot.h5. The trained model (Autopilot.h5) is loaded and predicts the processed image which is the image that was fed into the training model. Next, a steering image is read from the local disk to visualise the steering angle difference on the output window. Finally, the video of the local road was set as the output to predict the steering angle which will show the difference between actual steering angle and predicted steering angle.

This system was developed mainly to reduce accidents in Malaysia and also to help elderly people who are too old to drive safely. This is because as driver age increases, changes in flexibility, visual acuity, reaction time, memory and strength will affect the ability to drive.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Overview

This chapter discusses about the background and motive behind this project. It includes aim and objective of the proposed system as well as the problem statement to develop this system. Besides, the methodology and software tools used during the development stage will be discussed too.

1.2 Background of the project

AutoMove, an End-to-End Deep Learning for Self-Driving Vehicles have five main features which is to detect the lanes of the road, detect the speed of the vehicle from white-dashed lines, detect the angle of the road, object detection and predicting the steering angle of the vehicle.

This project involves five phases to complete. Firstly, dataset is taken using smartphone in a Malaysian local road. Then, the dataset is analysed and added as an input to detect the lanes in order for the vehicle to navigate inside a particular lane. After detecting the lanes, white-dashed lines on the middle of the road are taken to calculate the speed of the vehicle. The speed is determined by calculating the distance of starting and ending point of white-dashed lines and the time taken to reach the end line. Next, the angle of the road is calculated. Then, the objects on the road are detected. Finally, the steering angle of the vehicle is predicted based on the curvature of the road. If the angle is around zero to five degrees, the steering angle will be straight whereas if the angle goes under zero degree, the steering angle will turn left and if the angle is more than five degree the steering angle will turn right.

1.3 Project Motivation

The motivation for doing this End-to-End Deep Learning for Self-Driving Vehicle system was primarily an interest in undertaking a challenging project in an interesting area of research which is Artificial Intelligence. Furthermore, another reason why this project was chosen is because the opportunity to learn a new area of computing not covered in lecturer was appealing. Besides, autonomous vehicle is still new to the technology industry, which also one of the factors to develop something new that can help the future generations.

1.4 Aim and Objectives

The aim of the proposed system is to build an End-to-End Deep Learning for Self-Driving Vehicles that enable the system to *detect road boundaries and lanes, detect the speed of the vehicle,* and *predicting the steering angle for autonomous navigation using neural network.*

To achieve this aim, there are ten objectives which need to be considered. These ten objectives can be split into four categories. One of the objectives to investigate OpenCV library. This objective is to help in identifying the roads boundaries and lanes. Next, is to research the formula and physics to identify the speed of the vehicle. With these important calculations, the speed of the vehicle can be identified from the white-dashed lines. This objective will be carried out with the support of another objective which is to take dataset of the local roads. Besides, another category in these objectives is to investigate on machine learning framework and Convolutional Neural Network. This objective is to help in predicting the steering angle for navigation of the vehicle using neural network. The final category is on the project documentation. The documentation includes the analysis, synthesis, implementation, conclusion of the proposed system and to test the proposed system using black box and white box testing methodology. The final objective is to evaluate the end product and collect feedback on the further improvement on the product. This is to identify the strength and weakness of the product.

1.5 Problem Statement

The background story on how the project idea was found and the major reason to develop self-driving vehicle is because according to Assidiq et al., (2008), the United Nations has categorised Malaysia as 30th in the ranking of highest number of fatal accidents with an average of 4.5 causalities per 10,000 vehicles. Therefore, a system that can drive autonomously will reduce the number of accidents. Besides, in the same year in the United States, 32,885 people were killed from motor vehicle crashes and more than 2.2 million were injured (2010 Motor Vehicle Crashes: Overview, 2012). A research was carried out by KPMG, Center for Automotive Research, where they predicted more than 30,000 lives can be saved from this crashless future (Self-driving cars: The next revolution, 2012). Self-driving vehicles likely can reduce all driver error and most auto crashes.

1.6 Methodology

To develop this system, two methodologies were used. For software development, waterfall model is used. The main reason why waterfall is selected is because this project had a fixed start and end date. Besides, for the prototype implementation, spiral methodology will be used. This is because, in this entire application implementation, there are times where the proposed system's framework need to be refined where tools that are being implemented might need to be changed during the implementation period due to its availability or incompatibility.

1.7 Approach and Software Tools

The End-to-End Deep Learning for Self-Driving Vehicles system was developed using Python programming. PyCharm was used as the integrated development environment (IDE) to develop this system. Furthermore, OpenCV was used to detect the road

boundaries and white-dashed lines. It also helped to calculate the speed of the vehicle and the road angle in real time. Moreover, Adobe Premiere was used to change the frame rate of the video dataset and enhance the quality of the video. For the database, local database was used to store the video dataset.

1.8 Summary

After identifying the aims and objectives for this proposed system, it is concluded that this system requires more research before implementation. All the other areas of research will be discussed in next chapters with proper justifications and examples.

Chapter 2: Literature Review

2.0 Overview

In this chapter, three areas of research will be discussed that includes image recognition, machine learning and object detection. This analysis was aimed to help in studying all the methodology and complexity in the implementation of the proposed system. Each of them will be supported by at least two existing system literature.

2.1 Image Recognition

The purpose of this study is to analyse all the existing image recognition projects to get some understanding and insights about their approach and methodology.

2.1.1 Traffic Sign Recognition

Rubén Laguna and his group mates developed a traffic sign recognition system using image processing techniques (Laguna et al., 2014). There are four steps in this application. Firstly, image pre-processing step was done and also the detection of regions of interest (ROIs), that involves image transformation to grayscale and implementing edge detection by the Laplacian of Gaussian (LOG) filter. Next step is to identify the potential traffic signs detection, where comparison between ROIs and each shape pattern will be done (Laguna et al., 2014). Third step is recognition stage using a cross-correlation algorithm, where if each potential traffic sign is validated, it will be classified based on the traffic sign database. Finally, previous stages are managed and controlled by a graphical user interface. The results obtained showed a good accuracy and performance of the developed application, acceptable conditions of size and contrast of the input image were taken in consideration (Laguna et al., 2014).

2.1.2 Face Recognition

Another important area of research in image recognition is face detection system. The threshold function for a system will extract the information from the human face. The first existing system in this field was by Fares Jalled in 2017 who build a face recognition system using EigenFace which discriminating image input into multiple classes for person identification. The researcher's proposed system structure starts from the image input to the pre-processing process of the image. In the image pre-processing stage, the image resolution, environment elimination, image rotation as well as illumination to the image were taken care of before the next step, feature extraction (Jalled, 2017). In the feature extraction, the system will retrieve all the distinctive elements from a person's face for classification purpose. The classification purpose will classify the person by their name and features on their face through the database. Another face recognition project considered for this area was from

Priyanka Wagh and her colleague in 2017. This proposed system was aimed to use face recognition to help in taking attendance. This system achieves face recognition by setting up a static camera where the entire class is covered to periodically take a photo. The taken photos were converted to a grayscale image with histogram equalization to improve the contrast in the image (Wagh et al., 2015). For the student recognition in the classroom, Viola and Jones framework were used. Each recorded face was tested through EigenFace and Principal Component Analysis (PCA) for random value reduction as EigenFace often maximize variation so with PCA it helps the system to combine the variations based on a specific principle (Wagh et al., 2015).

2.1.3 Intelligent Parking Space Detection System

A research was conducted by the Research and Innovation Department of University Malaysia Pahang that capture and process the brown rounded image which was drawn on a parking lot and show the information of the empty car parking spaces (Yusnita et al., 2012). The information will be displayed at a display unit which contains seven segments in real time. The segments will show the number of available parking lots in the parking area. There are five main modules in this project. The first module is system initialisation where the location of every parking lot will be identified in the image. Next, is image acquisition module, which captures and stores digital images taken from camera. Third module is image segmentation, which split the objects from the background and differentiate pixels to improve the contrast (Yusnita et al., 2012). Another technique used in this module is thresholding. The next module is image enhancement. In this module, morphology functions are used to remove noise. The final module is the image detection where the boundaries of the object in the images will be traced. According to Yusnita (2012), image detection is used to find which objects are round by estimating the area and parameter of each object. In this project, a threshold value of 0.9 is used for the rounded image. Therefore, if the detected image is more than 0.9 threshold value, then the image is considered as available parking lot (Yusnita et al., 2012).

2.1.4 Number plate recognition

Tella Pavani and DVR Mohan, students from SRKR Engineering College, AP, India, developed a computer system that recognise any digital image automatically on the number plate. There are four processes to develop this system which include taking pictures, localising the number pad, truncating characters and OCR from alphanumeric characters (Pavani and Mohan, 2019). The main aim of this system is to develop algorithms to localise the license plate in the captured image, divide the characters from the number plate and to identify each character of the segment using Open Computer Vision Library

(OpenCV). K-NN algorithm was used to develop this system (Pavani and Mohan, 2019).

2.2 Deep Learning Neural Network

Three types of neural network architectures that are relevant to this proposed system will be discussed throughout this sub-topic.

2.2.1 Convolutional Neural Network (CNN)

In 2016, NVIDIA Corporation implemented End to End Learning for Self-Driving Cars using a convolutional neural network (CNN). Raw pixels from a single front-facing camera was map directly to steering commands (Bojarski et al., 2016, 3-6). According to the authors, this end-too-end approach proved powerful. The system learns to drive in traffic with less training data on local roads with or without lane markings. It also worked in places with blurry visual guidance such on impervious surface roads and in parking lots. The system automatically learns internal model of the needed processing steps like as detecting suitable road signs with only the human steering angle as the training signal (Bojarski et al., 2016, 3-6). Images are fed into CNN, which then measure a proposed steering command. After the training is done, the steering from the video images will be generated from the network. For data collection, training data taken by driving on a different type of roads and in a diverse set of weather conditions and lighting. Mostly, the road data was collected in central New Jersey. Data was collected in cloudy, clear, snowy, foggy, and rainy weather, both day and night. 72 hours of driving data was collected as of March 28, 2016 (Bojarski et al., 2016, 3-6). The weights of network were trained to reduce the mean squared error between the adjusted steering command and steering command output by the network. There are 9 layers for the network including 5 convolutional layers, a normalization layer and 3 fully connected layer. The input image is separated into YUV planes and send to the network (Bojarski et al., 2016, 3-6).

Next is the training. Selecting the frames to use is the first step to train a neural network. The collected data is labelled with weather condition, road type, and driver's activity (staying in lane, turning and switching lanes. The data that driver was staying in a lane was selected and others were discarded. Then the video sampled at 10 FPS (Bojarski et al., 2016, 3-6). After that is data augmentation. The data will be augmented after selecting the final set of frames by adding artificial rotations and shifts to teach the network how to restore from a poor orientation or position. The final stage is simulation. Pre-recorder videos will be taken by the simulator from a forward-facing on board camera and images that relatively what would appear if the CNN were, instead, steering the vehicle will be generated. In conclusion, during training, the system learns to detect the outline of a road without the explicit labels.

2.2.2 Recurrent Neural Network (RNN)

Recurrent neural networks have become the choice of neural networks when it comes to handling sequential data. Examples of these includes text, sound and video. RNNs has proven useful for handwriting recognition, speech recognition, language modelling and video classification. Graves et al used RNN with a Bidirectional Long Short Term Memory (BLSTM) to recognise handwriting and argued it to be superior to the then state-of-the-art Hidden Markov Model-based systems. Mohamed et al tackled speech recognition using RNN with Long Short Term Memory (LSTM) and argues they achieve the best recorded result on the dataset. Mikolov et al improved an RNN language model. What all these issues have in common is that a single datapoint does not hold enough information to define the data. The basic algorithm of the RNN is much like a normal neural network layer but it has a separate set of weights to evaluate the result of the previous element in the sequence's output. There are many different variations of it, but the one used by the framework Keras used in this thesis is the Elman network. This hold a hidden state calculated by Equation 1 where x_t is the input and h_t is the hidden state. W_h and U_h is the correspondent weight matrices. b_h is a bias. σ is the sigmoid activation function. According to Jeffrey L Elman (1990), the hidden state is then used to calculate the output in Equation 2 where y_t is the output for time t, h_t is the hidden state from Equation 1, W_y is the weight matrix for the output and by is a bias.

$$ht = \sigma h\ (\mathcal{W}h\mathrm{X}t + UhHt - 1 + bh \quad (1)$$

$$yt = \sigma y(WyHt + by) \quad (2)$$

2.2.3 Reinforcement Learning

This project was proposed by Matt Vitelli and Aran Nayebi under the title "CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving". A deep Q-network (DQN) agent was created to undergo the task of autonomous car driving from raw inputs that were taken using sensors. The agent's performance was evaluated against several standard agents in a racing simulation environment. The experiment was done by comparing final hybrid CNN-RNN DQN agent against discrete Q-learning agent. All agents used the same reward function. The agents were compared based on three criteria: 1) The average rewards attained by each agent during the course of its run, 2) the average speed attained by each agent during the course of its run, and 3) the highest speed attained by each agent during the course of its run. All tests were done on the Vdrift Ruddskogen track (Vitelli and Nayebi, 2016) and were trained before calculating the three criteria. Table 1 shows the comparison between different agents (Vitelli and Nayebi, 2016).

Table 1: Performance comparison

	Average Reward	Average Speed	Max Speed
Hand-Crafted Controller	0.542	17.65 m/s	18.77 m/s
Greedy Agent	1.416	17.53 m/s	18.16 m/s
Discrete Agent	0.635	18.71 m/s	20.21 m/s
DQN Agent	0.915	17.57 m/s	24.71 m/s

After comparing the criteria, the DQN agent was the second highest rewards and achieved the highest speed. However, it has the second lowest average speed. The DQN controller mostly did not perform braking actions. According to Vitelli and Nayebi (2016), this helps to improve the performance in terms of speed, but causes the agent to be more unstable compared to other agents. The strongest agent was the discrete Q-learning agent. This is because the average speed was higher and the maximum speed was still near to DQN's maximum speed. Besides, the discrete Q-learning agent maintain the stability of the car and only use the brakes during sharp turns. Vitelli (2016) argued that despite limitation in roads selection, reinforcement learning agent able to successfully navigate a car without any explicit notion of the car's basic dynamics.

2.3 Object Detection

There are two types of algorithms for object detection that will be discussed in this sub topic.

2.3.1 Faster R-CNN

The R-CNN technique trains CNN end-to-end to do classification for proposal regions into object background or categories. R-CNN mostly act as a classifier, and the object bounds cannot not be predicted. The performance of the region proposal module defines the accuracy. Pierre Sermanet and his team proposed a paper in 2013 under the title of "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks" which shows the ways of using deep networks for predicting object bounding boxes (Sermanet et al., 2013). In the OverFeat method, a fully-connected layer is trained to predict the coordinates of the box for the localisation task that consider a single object. Then, the fully-connected layer is turned into a convolutional layer for multiple class object detection (Sermanet et al., 2013). Figure 1 shows the architecture of Faster R-CNN which is a single, unified network for object detection.

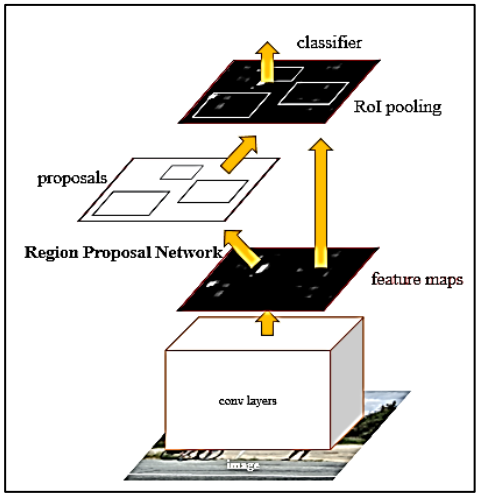


Figure 1: architecture of Faster R-CNN (Sermanet et al., 2013)

Fast R-CNN implement end-to-end detector training on shared convolutional features and display decent accuracy and speed. On the other hand, R-CNN fails to do real time detection because of its two step architecture.

2.3.2 YOLO

YOLO stands for "You only look once". According to Redmon (2016), it is an object detection algorithm that runs quicker than R-CNN because of its simpler architecture. Classification and bounding box regression will be done at the same time. Figure 2 shows how YOLO detection system works.

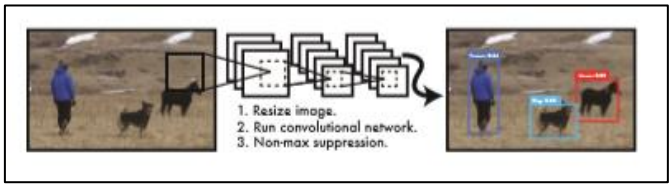


Figure 2: YOLO Detection

A single convolutional network predicts several bounding boxes and class probabilities for those boxes simultaneously. YOLO trains on full images and the detection performance will be optimised directly. This ensure YOLO to run extremely fast, hence make real time prediction.

2.4 Summary

After getting all the information from three areas or research, the approach and tools required to develop the system can be identified. For neural network, convolutional neural network is chosen because it is more suitable for images and video processing. Furthermore, for object detection YOLO is chosen as is more efficient and can work in real time compared to faster R-CNN. In the next chapter, functional and non-function requirements will be discussed.

Chapter 3: Commentary on Key Points in the Requirements Specification

3.0 Overview

After completing all the research on relevant literatures, main functionalities of the system will be discussed in this chapter though a conceptual model. After discussing the conceptual model, commentary on chosen methods will be done in terms of functional and non-functional requirements to further the research.

3.1 Functional Requirements

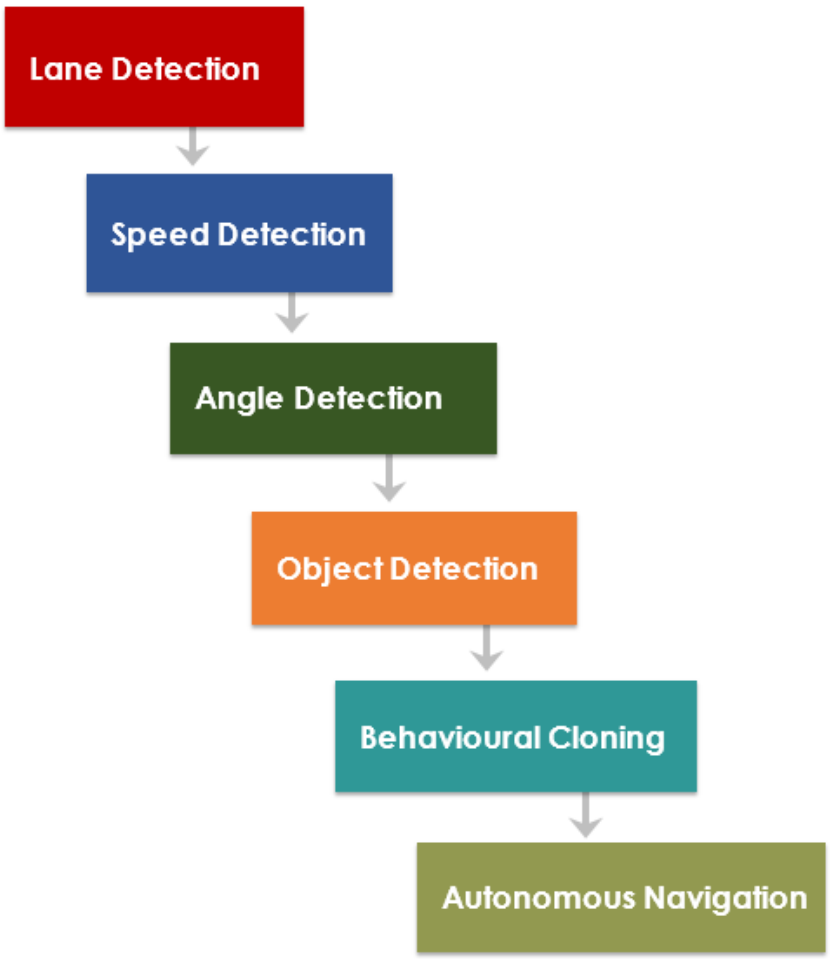


Figure 3: Automove Functional Requirements

3.1.1 Lane Detection

There are six functional requirements in Automove based on the Figure 3. The first one is called lane detection. This accepts road data and uses OpenCV image processing technique to identify the lanes. The first step is applying colour thresholding. Then the image will be converted to grayscale to identify the region of interest. After applying threshold, noise is removed so that the white lanes can be identified. Then higher order polynomial is needed to incorporate

to detect curved roads. For that, the algorithm must have an awareness of previous detection and a value of confidence. In order to fit a higher order polynomial, the image is sliced in horizontal strips. On each strip, straight line algorithm was applied and pixels were identified corresponding to lane detections. Then, the pixels will combine all the stripes and a new polynomial function will be created that fits the best. Finally, a blue mask is attached on the lane to see the difference between the lanes and the other images on the road.

3.1.2 Speed Detection

The second functional requirement is detecting the speed of vehicle. Speed detection is an important aspect for autonomous vehicle to help the driver to prevent the vehicle from overtaking the speed limit. In this system, to detect the speed, centre white-lane markers are used. When a lane-marker is present, number of frames will be tracked until the next lane marker appears in the same window. Then, the number of frame rate of the video is taken which will be extracted from the video. From this, the speed of the vehicle will be calculated using distance / time formula where distance is the distance of the white-lane and time is the time taken of the vehicle to reach the end point of white-lane. The detailed formula is explained in Chapter 5 under implementation. Then, the speed will be showed in km/h.

3.1.3 Angle Detection

Detecting the angle of the road is crucial to predict the vehicle's steering angle. Polynomial curve fitting model is used to find the best-fit function (Curve) along a set of range. Firstly, the difference angle for each frame is measured by subtracting top angle and bottom angle of the rectangle which is placed on the white-dashed lines. Then, the average angle is calculated by using exponential moving average formula. The formula explanation can be seen in Chapter 5 under implementation. From this, the average angle of each frame is saved into a .txt file to create a new dataset.

3.1.4 Object Detection

Object detection is used to avoid the obstacle around the vehicle during autonomous driving. For this system, algorithm based on regressions which is the YOLO method is used. YOLO stands for "You Only Look Once". According to Geethapriya (2019), the classes and bounding boxes of the whole image will be predicted at a single run of the algorithm and multiple objects will be detected using a single neural network. To make use of YOLO algorithm, an image is divided as grids of 3 x 3 matrixes. After the image is divided, classification and localisation of the object for each grid will be done. From this, the confidence score of each grid is determined. If no proper object is found in the grid, then the confidence and value of bounding box will be zero or if there is an object found in the grid, then the confidence will be 1 and the value of

bounding box will be its corresponding bounding values of the object that is found. The in-depth details of how the objects are detected can be seen in chapter 5.

3.1.5 Behavioural Cloning

The next functional requirement is behavioural cloning. In this part, the model will be trained using Tensorflow and Keras library. The video dataset is split into images frame by frame. Then pre-process is done. Data pre-processing is a data mining technique which transforms raw data into understandable format. Then, a pickle file is created to store all the images to train. After that, a CNN model is built in Keras. After training the model, compilation takes place. It takes three parameters which are optimizer, loss and metrics. The optimizer controls the learning rate. For optimiser, 'adam' is used as adam optimizer as it adapts the learning rate during training. The learning fate will determine how fast the best weight is calculated for the model. Mean square error (mse) is used for the loss function. The model was train on 2937 samples and validated on 1259 samples with 30 epochs. Then the best model is saved as Autopilot.h5. The full details of training can be seen in chapter 5.

3.1.6 Autonomous Navigation

To predict the steering angle, the trained model (Autopilot.h5) is loaded. Then, the model is predicted using a Keras model. After that the model predicts the processed image which is the image that was fed into the training model. Next, a steering image is read from the local disk to visualise the steering angle difference on the output window. Then, the video of the local road were set as the output to predict the steering angle which will show the difference between actual steering angle and predicted steering angle. The accuracy of the steering angle is explained in detail in chapter 5.

3.2 Non-functional Requirements

3.2.1 Security

Security is an important factor in any application. Hence, in this system, none of the data’s information is on any external online server. The algorithm stored in local disk where administrator access is required before accessing the codes.

3.2.1 Performance

Performance plays a crucial role in sustaining a system in the market. To maintain a good performance for AutoMove, heavy tasks are done externally. The heavy tasks include searching and retrieving road data that consists of thousands of rows, and traffic signs recognition model training.

3.2.2 Usability

There is no login or sign out process in AutoMove which will help the users directly to use the system. Furthermore, there is no complex interface to run this system as user can just run the output from PyCharm IDE.

3.2.3 Availability

The system should be available at all times without any errors or bugs as most of the function in the system runs in real-time. The only necessity for users to use this system is a computer with high processing power.

3.3 Justification of Requirements

Firstly, the road dataset was collected by driving on a Malaysian local road. The road has two lanes with white dashed lines on the middle. Data was collected in clear and sunny weather, so that the algorithm can identify the road clearly. The road data was collected using a phone camera that was placed on the car dashboard. The reason behind is this to get a proper view of the road for the system to train. Secondly, the video was taken with 30 frames/second so that video will be stable. The frame width and height are 1920 and 1080 pixels. Thirdly, the data collected will be not stored in cloud is because data that stored in the server is usually more prone to cyber-attacks or hacking compared to data that is being stored on the local storage (Mario Ballano , et al., 2014). For the technique, the architecture chosen was CNN as it provides more customizability and assist in reducing the complexity requirements in computation with high performance. Moreover, based on the previous literature, CNN is able to learn feature from images adaptively that makes the architecture selection more appropriate to this proposed system (Bojarski et al., 2016).

3.4 Justification of Software Resources

For this proposed system, the main resource required for this system would be a laptop with minimum 8GB RAM and NVIDIA GPU. To build this system, an appropriate platform is required. For this, PyCharm is chosen as the IDE. There are few reasons behind this. One of the reasons is PyCharm provide code completion. This will assist the developer during coding. Next, is PyCharm provide debugging and breakpoints that helps the program to debug and check for errors easily. It also provide code refactoring. Pycharm will understand the syntax tree and has the ability to make changes consistently (Marsja, 2017). To train the model, TensorFlow framework was chosen. This is because of the available resources and documentation for the framework (Martín Abadi, 2016). This huge amount of resources will reduce the chances of failing to implement the features compared to frameworks that has fewer resources or projects for developers. Keras, an open source neural-network library, will be used on top of TensorFlow as an interface because autonomous vehicles project researches as well as developers stated that Keras is able to help in creating layers in CNN architecture with less number of codes, which is easy to understand (Zhang, 2017).

3.5 Methodology

In this proposed system, the software development methodology used is waterfall. The main reason why waterfall is selected is because this project had a fixed start and end date and since waterfall methodology is popular for its milestone focused development. Therefore, this approach can assist in building the system within the time given. Besides, for the prototype implementation, spiral methodology will be used. This is because, in this entire application implementation, there are times where the proposed system's framework need to be refined where tools that are being implemented might be changed during the implementation period due to its availability or incompatibility. These changes more suited in spiral methodology, since there will be not be any negative domino impact to modify periods of the model (Boehm, 1988). Furthermore, another reason why spiral model is selected is because of spiral model point by point yet hazard free framework improvement process, which means there will be some increment that can be used to text and use (Boehm, 1988). With all these reasons, it is evident that waterfall methodology is the most suitable for system development methodology while spiral as the proposed system's prototype implementation approach. The time taken to develop this system was only confined to two semesters therefore proper planning and analysis were done to avoid any failure later on. The subchapters below explain the phases of project management.

3.5.1 Planning

During planning phase, the appropriate project title was discussed with project supervisor. Then, the required tools and methods of choosing the project was planned and chosen. After that, Project Initiation Document (PID) was submitted before starting the project proposal. After the PID was reviewed by the project supervisor, few weeks were spent on writing the project proposal. Project proposal contains background of the project, aim and objectives, resources and also the sources of information. The proposal then was submitted to be reviewed by the project supervisor and second marker. The changes were made from the feedbacks during the meeting. After submitting the revised proposal, the project was approved and given green mark in the ethics form. The detailed description of planning can be found in Chapter 7.

3.5.2 Analysis

After completing the planning phase, analysis is started by writing report which consist of analysis chapter. The literature review was written by researching through the required neural network, deep learning techniques, dataset, and also image processing techniques. Then, the functional and non-functional requirements were classified. Furthermore, the tools that are needed were analysed and chosen with proper justifications. Then, to get in-depth details of the project's outline, Software Requirement Specification (SRS) were written that includes user characteristics which would be a guideline for developers.

SRS is added to the appendix B of the report. After submitting the analysis chapter, a short presentation was done to project supervisor to show what have been found and researched which will be helpful to develop the system.

3.5.3 Design

After analysis phase, the design of the system will be created. The required diagrams are drawn such as use case diagram, sequence diagram and class diagram in Chapter 4. This design phase gave an outline on all the important classes that should be implemented.

3.5.4 Implementation

The development of End to End Deep Learning for self-driving vehicles began in this phase. The system is developed using PyCharm IDE that was chosen earlier in the planning phase. Convolutional Neural Network was used to train the model. Then, for object detection. Tensorflow Object Detection API is used to recognise the objects in the video. The functional requirements were implemented to the system to complete the development of this project.

3.5.5 Testing

Testing is done which can be viewed in Chapter 6 after implementation phase to check for bugs and errors. Test cases were created. After completing this phase, the system will be demonstrated.

3.6 Existing system

In this section, existing systems that are related to this proposed system will be discussed to determine whether the proposed system requirements are appropriate before implementing the system.

3.6.1 End to End Learning for Self-Driving Cars

For this project, Mariusz Bojarski and his team proposed a method that adapts Convolutional Neural Network (CNN). The network consist of 9 layers, 5 convolutional layers, 3 fully connected layers, and also a normalization layer. The input images are split into YUV planes and passed to the network (Bojarski et al., 2016). For the first layer, image normalisation is performed. The normaliser is hard-coded and is not modified in the learning process. Normalisation scheme will be altered with the network architecture when performing normalisation in the network. The convolutional layers were designed to do feature extraction and were chosen experimentally (Bojarski et al., 2016). A series of experiments were done for layer configurations. The team used strided convolutions in the first three convolutional layers with a 2x2 stride and a 5x5 kernel and a non-strided convolutional with a 3x3 kernel size in the final two convolutional layers (Bojarski et al., 2016). They followed five

convolutional layers with three fully connected layers heading to an output control value which is the inverse turning radius (Bojarski et al., 2016). For steering, the fully connected layers are designed to act as a controller. However, the team noticed that training the system end-to-end, a clean break between which parts of the network function primarily as controller and which part serve as a feature extractor is not possible (Bojarski et al., 2016). The team trained the CNN to map raw pixels from a single front-facing camera that is directed to steering commands. The training data was minimum from humans, nevertheless the system learns to drive in traffic on local roads with or without lane markings and also on highways (Bojarski et al., 2016). It also could work in areas with blurry visual guidance such as on incomplete roads and parking lots.

3.6.2 Traffic Signs Recognition and Classification based on Deep Feature Learning

A novel traffic signs recognition and classification method was presented by Yan Lai and team based on Convolutional Neural Network and Support Vector Machine (CNN-SVM). In this method, the YCbCr color space is introduced in CNN to divide the color channels for feature extraction (Lai et al., 2018). A SVM classifier is used for classification based on the extracted features. The experiments are conducted on a real world data set with images and videos captured from ordinary car driving. The experimental results showed that compared with the state-of-the-art methods, the method achieves the best performance on traffic signs recognition and classification, with a highest 98.6% accuracy rate. The basic model follows the classic LeNet-5 network, which consists of three convolutional layers (C1, C3 and C5) and two sub-sampling layers (S2 and S4). The input of each layer represents a small group of local units from the upper layer (Lai et al., 2018). Each convolution layer contains multiple convolution kernels. These convolution kernels are able to scan the image features via different expressions, based on this, we can acquire various feature maps in different locations. The subsampling layer, following the convolution layer, is mainly used to reduce the resolution of the feature map, to extract the existing image features and to determine the features' relative location (Lai et al., 2018).

3.7 Summary

After reviewing all the existing systems, it is evident that self-driving vehicle is still in research development. Furthermore, most of the systems are done with only one objective. Nevertheless, in this system, there are more than four functions which will be implemented together. In next chapter, the design for proposed system will be discussed.

Chapter 4: Design

4.0 Overview

In this chapter, system design of this proposed system will be discussed. Furthermore, this chapter will also cover the design models such as Unified Modelling Language (UML) diagrams such as use case diagram, class diagram, sequence diagram and storyboard of the application.

4.1 System Architecture

Figure 4 shows the overview of AutoMove system architecture. The entire process was done after collecting the road dataset using a smartphone. A smartphone camera was mounted in front of the steering of a car. The video of the road approximately 1.5km was taken while driving on the road. Then, the dataset was fed into the code. The first step to produce a self-driving vehicle is detecting the lanes. Then, the speed of the vehicle is identified by using the white line markers on the road. After that, objects that are surrounded on the road are detected using object detection. Finally, is the behavioural cloning which predicts the steering angle from the trained model.

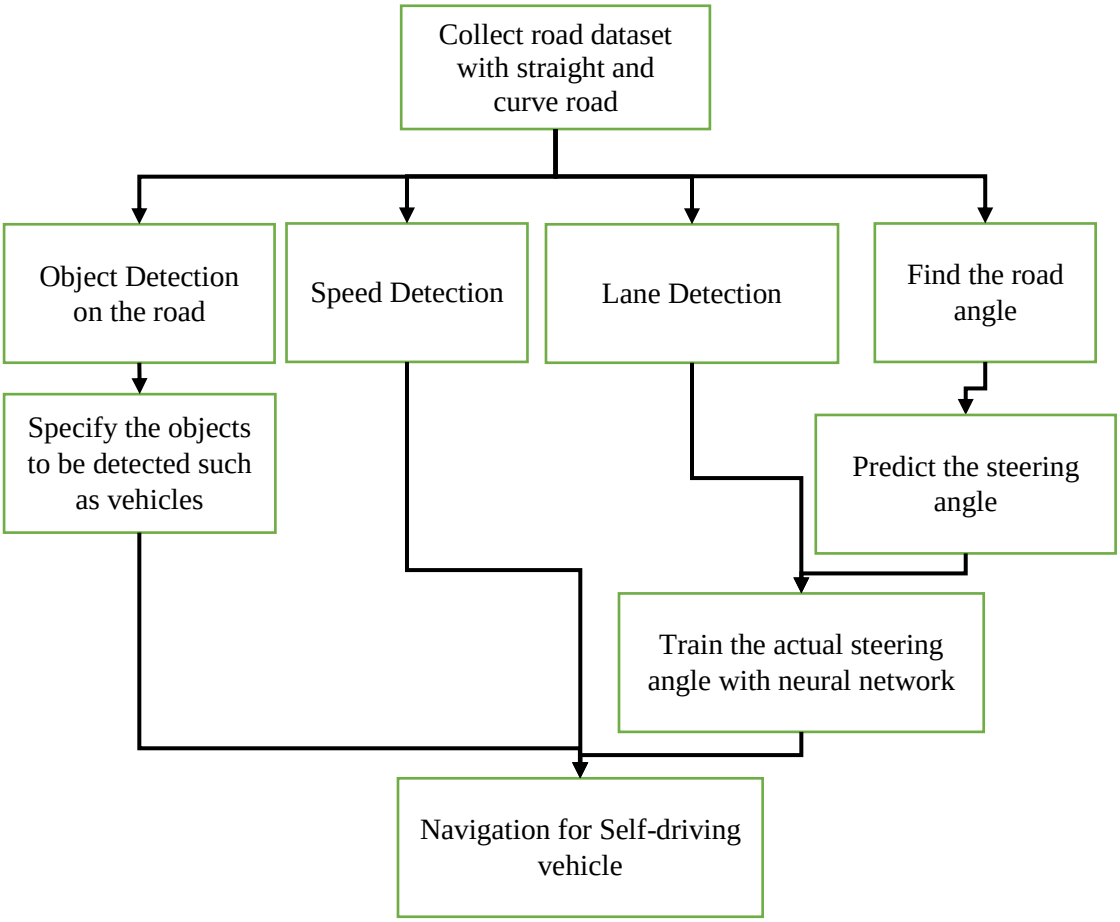


Figure 4: Overview of AutoMove

4.2 System Design

In this section, three main components of system design will be discussed which are 4.2.1: Functional Design, 4.2.2: Structural Design and 4.2.3: Interaction Design.

4.2.1 Functional Design

The use case diagram in Figure 5 shows the functional design of the End-to-End Deep Learning for Self-Driving Vehicles. There is one kind of user which is the driver who uses AutoMove system. In order for the self-driving vehicle to navigate autonomously, the first process is identifying the lanes of the vehicle. Then, the speed of the vehicle will detected based on white line markers. Besides that, to avoid the obstacles on the road, object detection will be implemented so that the vehicle can recognise the objects on the video dataset. Finally, the steering angle of the vehicle will be predicted based on straight and curve roads by using convolutional neural network.

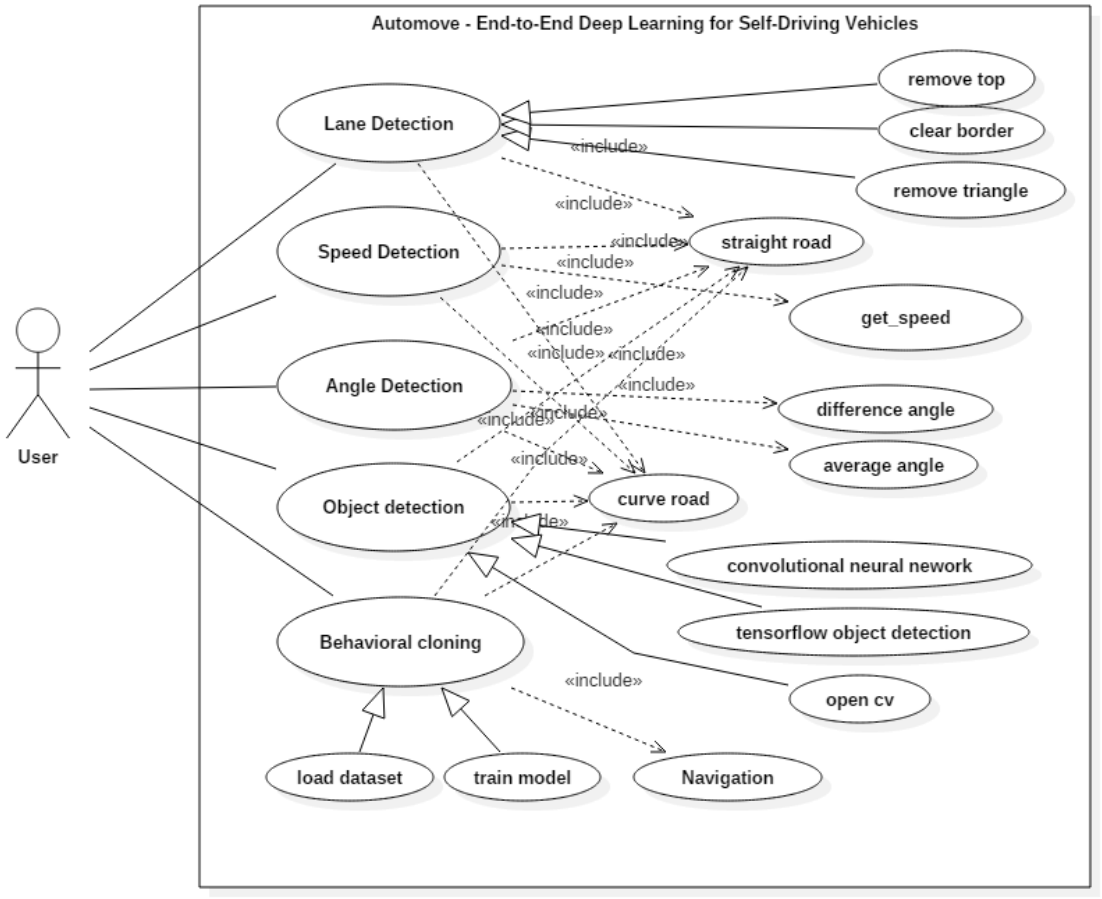


Figure 5: Use Case Diagram for AutoMove

4.2.2 Structural Design

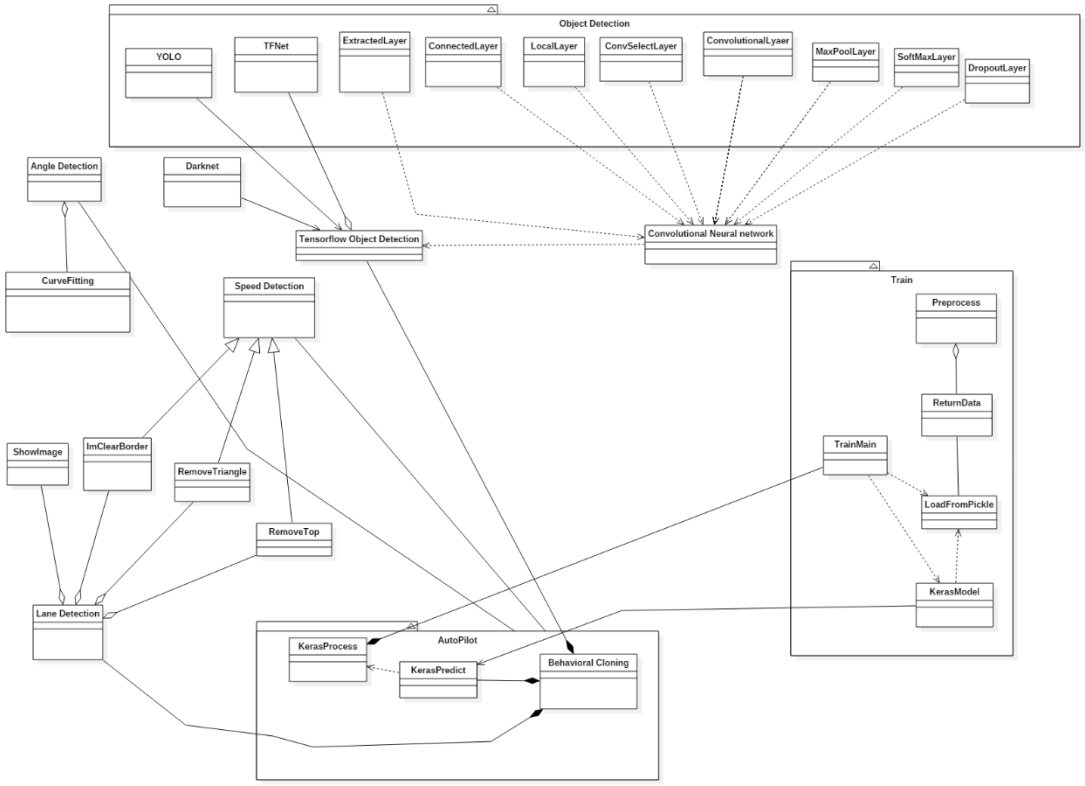


Figure 6: Class Diagram of Automove

Figure 6 shows the class diagram of AutoMove. There were several classes and three packages were used for this system. Object detection package has the necessary class files that connects classes for YOLO, TFNet, Extracted layer, Connected layer, Local layer, ConvSelect layer, Convolutional layer, Maxpool layer, Softmax layer and Dropout layer to form a convolutional neural network. Then, train package contains classes like preprocess. ReturnData, LoadFromPickle, KerasModel and TrainModel which trains the model of the steering angle. Then, Autopilot package which represents classes' files such as KerasProcess, KerasPredict and Behavioural Cloning takes care of the navigation of predicted steering angle.

Furthermore, a common public class is used to keep several public data that will be used multiple times by other classes. The lane detection class is associated with the common class and will be used by other classes such as showImage, imClearborder, removeTriangle and removeTop. For speed detection, generalisation is used to connect the same classes which are being used for other functions such as lane detection. Angle detection class consist of curve fitting which will measure the top angle and bottom angle of the road

which then will calculate the average angle of the road. The second layer class diagram with all the attributes and methods are shown in Appendix C which contain details in depth and purposes of the classes.

4.2.3 Interaction of System Design

In this section, six sequence diagrams will be discussed. Sequence diagrams are created to visualise the interactions that represents the objects and classes involved in the situation to execute a functionality. Figure 7 shows the sequence diagram of lane detection. It contains messages message 1 to 8. Message 1 shows, if the image is loaded then the image will be converted to grayscale and threshold will be applied in message 2. If the image does not have a proper quality, then threshold will not be applied in message 3 which is the reply message and shows an invalid image error in message 4. Then, in message 5, border will be cleared. After that, a triangle is drawn to remove the unnecessary parts of the road in message 6. Top of the road then will be removed in message 7. Finally, after detecting the lanes on the road, mask will be applied in message 8.

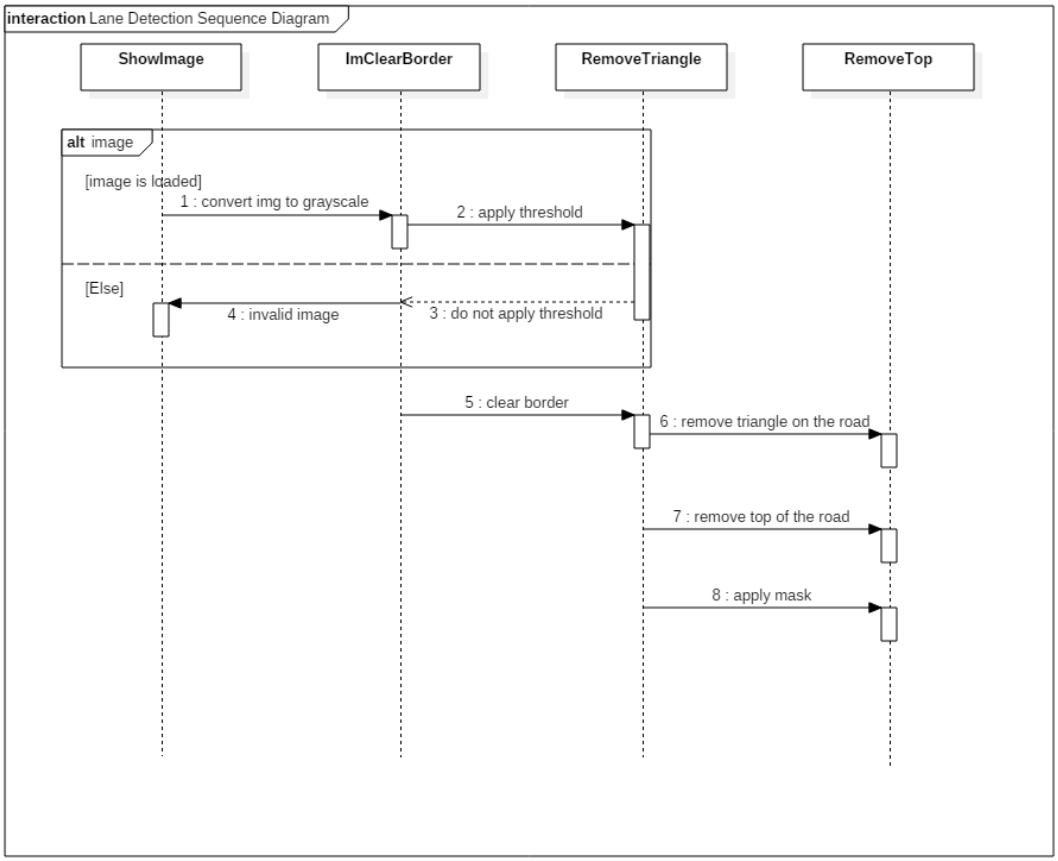


Figure 7: Lane Detection Sequence Diagram

Figure 8 shows the sequence diagram to detect speed of the vehicle. Messages from 1 to 7 is shown. Message 1 shows if the video from the dataset is loaded,

then threshold will be applied. Then, clear border function is used in message 2. If the video path is different, message 3 will be shown which is video is invalid. Message 4 will be shown if rectangle detect the while lines. It will set the speed for each frame. If rectangle does not detect white lines, then frames are not counted which the 5th message is and speed of the previous will be shown in message 6. After that, message 7 will be shown which is the draw text function that shows the speed in km/h.

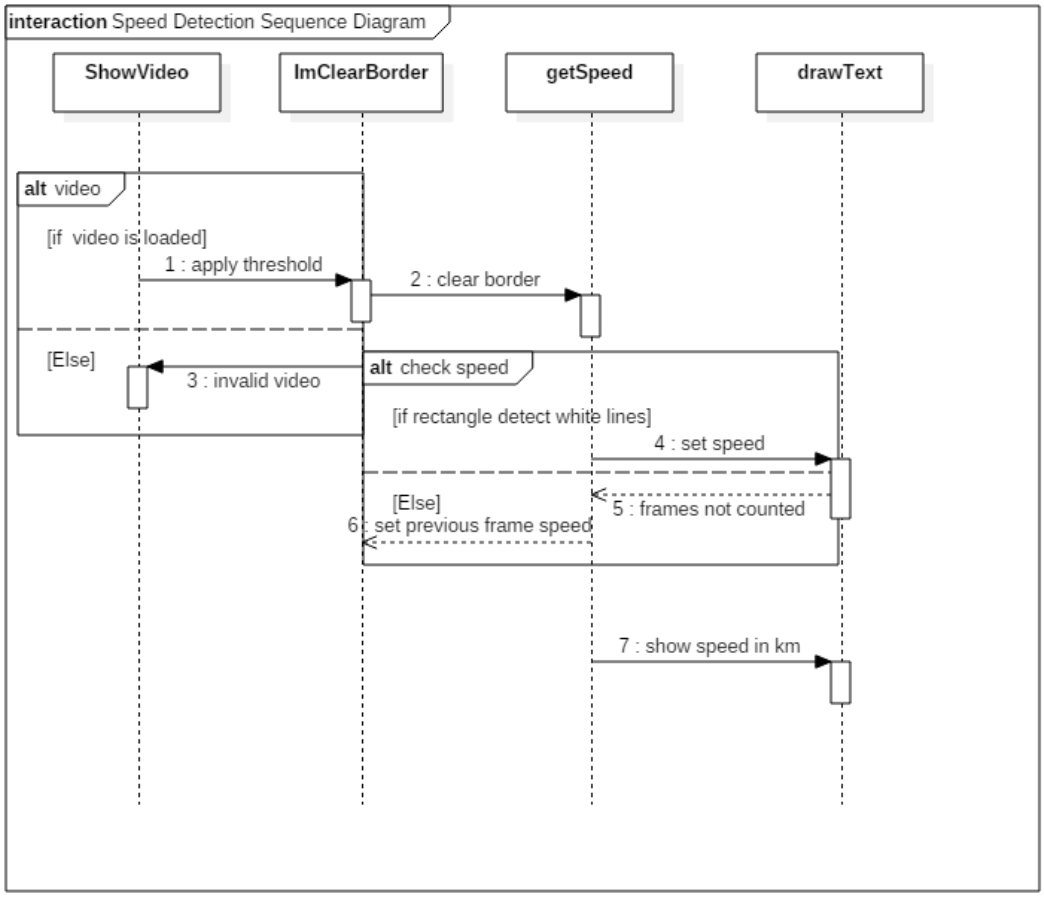


Figure 8: Speed Detection Sequence Diagram

Figure 9 shows the sequence diagram to detect angle of the road for each frame. This diagram also contains messages from 1 to 7. The first message is shown if the video if loaded, then the threshold will be applied. Message 2 will clear the border of the road. If the video is not loaded from the dataset folder, then message 3 will be shown which is invalid video. Then, if pixelCounts > 0, message 4 will be shown which is calculating the curvature of road. After getting the curvature, difference of angle will be measured in message 5. Then, with the difference angle average angle of the road will be shown in message 6. Finally, top angle, bottom angle, difference angle and average angle will be drawn on the output in message 7.

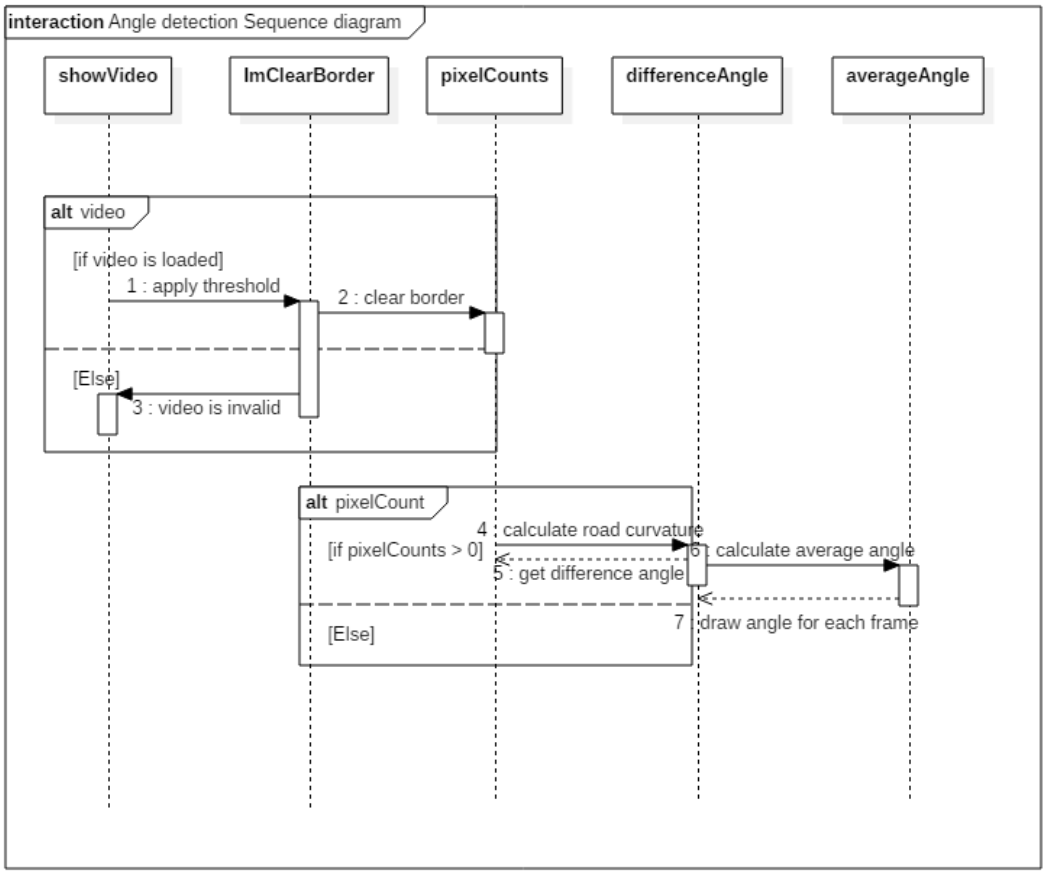


Figure 9: Angle Detection Sequence Diagram

Figure 10 shows the sequence diagram for object detection using YOLO algorithm. This diagram contains 7 messages. The first message will build the TFNet model if yolo pre-trained model is loaded. After that, the video dataset will be captured. Then the prediction of the objects will take place. If no model is found, message 4 will be shown which is no model found. Therefore, the objects in the video will not be detected in message 5. Then, predict function will be returned in the message 6. Finally, results will be label on the bounding box.

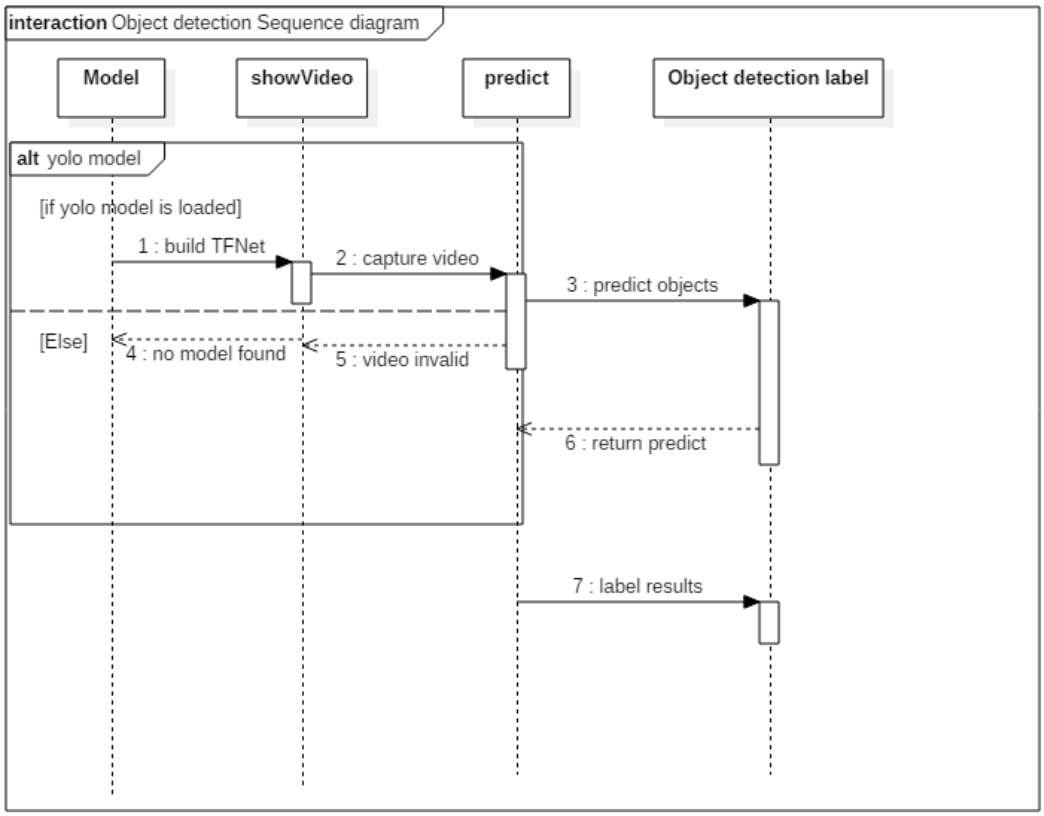


Figure 10: Object Detection Sequence Diagram

The final sequence diagram is the autopilot which is shown in Figure 11. There are six functions and 12 messages. First message is the pre-process. Then, the data will be returned in message 2 which will help to create a pickle file that gather all the images from dataset folder. If the dataset is invalid, message 3 will be shown. Then, in message 4, pickle file is created and then images will be stored in pickle file. After that, it will be split into features and labels. Message 6 will train the model of actual steering angle. Then the model will be saved in Autopilot.h5 which is shown in message 7. For the behavioural cloning, the model is loaded and images are processed using keras.process. Finally, in message 9, the steering angle will be predicted using keras.predict function.

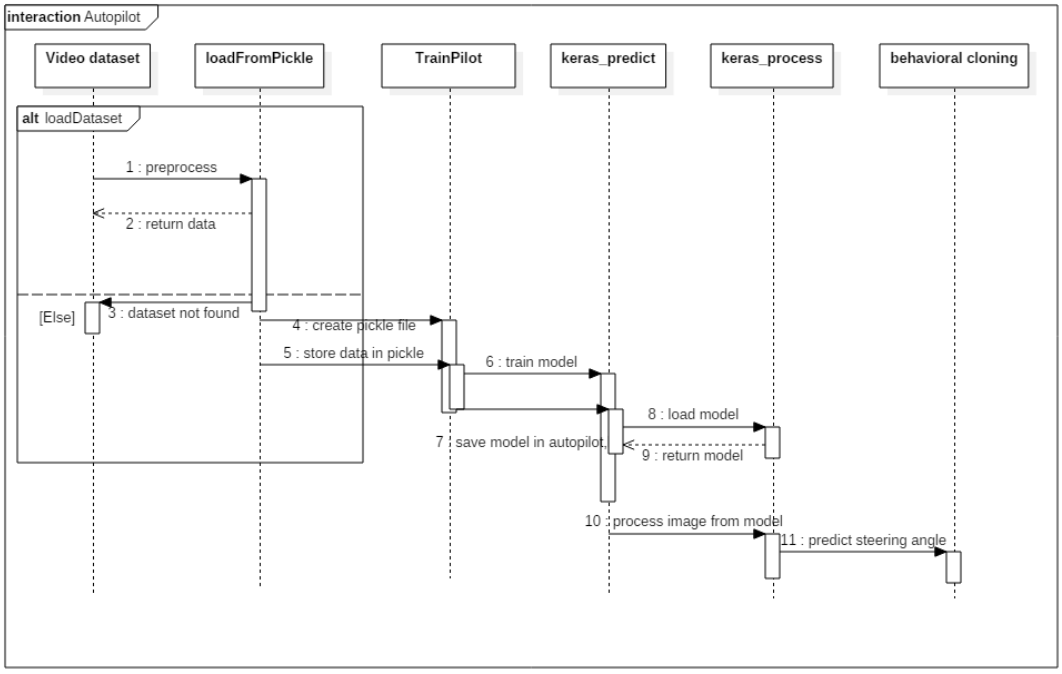


Figure 11: Autopilot Sequence Diagram

4.3 Summary

After designing all the UML diagrams, a clearer picture can be seen to develop the proposed system. In next chapter, the implementation of the system will be discussed with proper justifications.

Chapter 5: Implementation

5.0 Overview

In this chapter, the implementations that were implemented in this project will be discussed which covers the methods and techniques used for the implementation and also the justification on the main functionalities along with code snippets.

There are four functions which were required to implement this system. The first one was to identify lanes of the road from the given dataset. According to Laurent Desegur (2017), lane detection is an important component for self-driving vehicles. This is because it provides lateral bounds on the movement of vehicle which gives a perspective about road's curvature and how much the vehicle has been deviated from the center of the lane (Desegur, 2017). Lane detection depends only on camera images. For this project, computer vision techniques has been applied to process the image and provide lane markings as the output. Figure 12 shows the flowchart of lane detection.

5.1 Lane Detection

Lane detection is implemented by colour thresholding. The first step is to convert the image to grayscale identifying the region of interest. The pixels with a higher brightness value will be highlighted in this step. Figure 12 shows the code snippet to convert the image to grayscale and Figure 13 shows the image that is converted to grayscale.

```
#read image
img = cv2.imread('road png.PNG')
#convert image to grayscale
gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY)
```

Figure 12: Code snippet to convert image to grayscale

Figure 13: Image that are converted to grayscale

Next, threshold is set according to the values of colour channels. For example, white lanes are detected by applying a specific threshold for RGB colour channel which is [gray, 170, 255] followed by cv2.THRESH_BINARY. According to OpenCV official documentation, THRESH_BINARY function is used to transform a grayscale image to a binary image which will process the image in-place (OpenCV: Miscellaneous Image Transformations, 2019). Equation 3 shows the function for THRESH_BINARY.

$$dst(x,y) = \begin{cases} maxValue & if\ src(x,y) > T(x,y) \\ 0 & otherwise \end{cases} \quad (3)$$

Based on the formula above, (x,y) represent threshold calculation for each pixel. Dst is the image destination of the same size and type whereas maxValue is the non-zero value which is assigned to the pixels. Therefore, pixels that have channel values more than the threshold correlate to white objects in the image. Figure 14 shows the image after applying threshold.

Figure 14: Image after applying threshold

After applying threshold, it is required to remove as much of the noise as possible in a frame (Lane detection for self driving vehicles, 2018). According to the orientation and position of the camera, the lanes are located in the bottom half of the image. Therefore, a function called imclearborder is created to remove the unnecessary noise at the top of the image. Figure 15 shows the code snippet to clear the border of the image.

```
def imclearborder(bw):
    numRows = bw.shape[0]
    numCols = bw.shape[1]
    for row in [0]:
        for col in range(numCols):
            if bw[row, col] == 255:
                mask = numpy.zeros([numRows + 2, numCols + 2],
                numpy.uint8)
                cv2.floodFill(bw, mask, (col, row), 0)
    for row in range(numRows):
        for col in [0, numCols - 1]:
            if bw[row, col] == 255:
                mask = numpy.zeros([numRows + 2, numCols + 2],
                .uint8)
                cv2.floodFill(bw, mask, (col, row), 0)
    return bw
```

Figure 15: clear border function

According to the code snippet in Figure 15, a function is created to remove the noise that is filled with white colour at the top of the image. The numpy.zeros is called

which will return a new array of given shape, filled with zeros. Then, a mask will be added with black colour. Figure 16 shows the image after clearing the border.

Figure 16: Image after clearing border

Then, a function to remove top of the image is created. The area can be segregated using a trapezoid shape (Lane detection for self driving vehicles, 2018). When driving, the lines will keep changing position, therefore a simple ratio is used to create a polygonal shape. Figure 17 shows the code snippet of the functions and Figure 18 shows the image after masking the region of interest.

```
def RemoveTop(bw, height):
    for row in range(height):
        for col in range(bw.shape[1]):
            bw[row, col] = 0


def RemoveTriangle(bw):
    for col in range(bw.shape[1]):
        value1 = -9 / 8 * col + 775

        for row in range(100, bw.shape[0]):
            if row < value1:
                bw[row, col] = 0
            else:
                break

        value2 = 9 / 8 * col - 575

        for row in range(100, bw.shape[0]):
            if row < value2:
                bw[row, col] = 0
            else:
                break
```

Figure 17: Remove top and triangle function

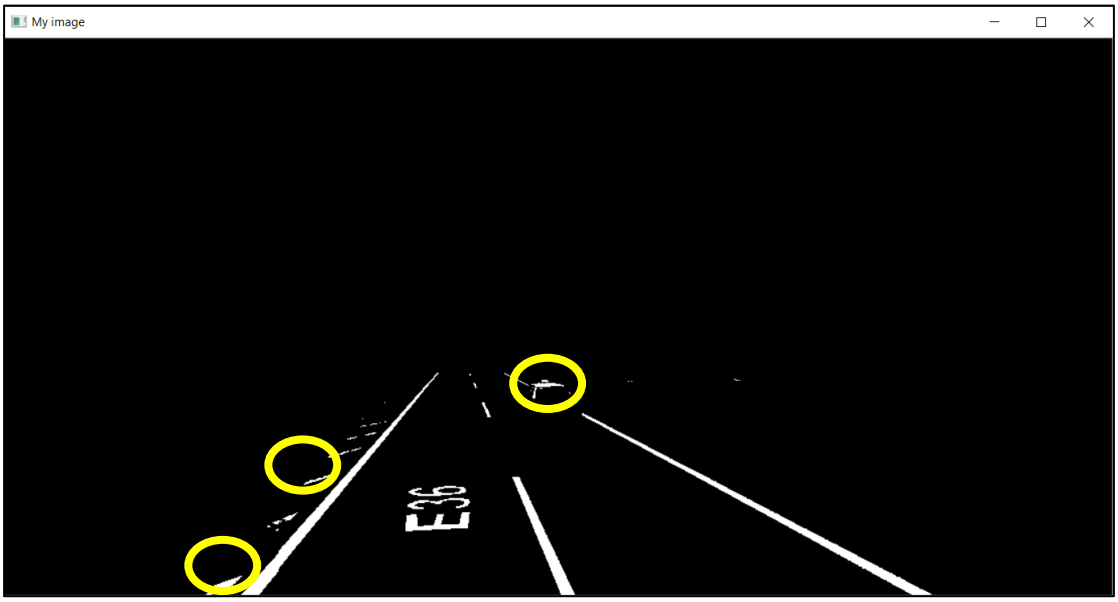


Figure 18: Masking of region of interest

After adding mask to the region of interest, next step is to remove the unwanted noise near the lanes so that the lanes can be detected clearly. Based on the Figure 18, it can be seen that there is some noise which might affect the lanes. Therefore, another

function was created to find the area of the white lines. Figure 19 shows the code snippet to find the area of white lines.

```
def bwareaopen(bw, minArea):
    output = cv2.connectedComponentsWithStats(bw) #end
points of bw
    numObjects = output[0]
    lb = output[1]

    for obj in range(1, numObjects):
        area = output[2][obj, 4]

        if area < minArea: #find area of white lines
            left = output[2][obj, 0]
            top = output[2][obj, 1]
            width = output[2][obj, 2]
            height = output[2][obj, 3]

            isFound = False

            for row in range(top, top + height):
                for col in range(left, left + width):
                    if lb[row, col] == obj:
                        mask = numpy.zeros((bw.shape[0] + 2,
                        bw.shape[1] + 2), numpy.uint8)
                        cv2.floodFill(bw, mask, (col, row),
                        0) #set the mask to black
                        isFound = True
                        break

                if isFound:
                    break

    return bw
```

Figure 19: Find minimum area function

Based on the function, if the area is smaller than the minimum area, then a mask will be placed which will remove the unnecessary white lines. Figure 20 shows the image after removing the unwanted lines.

Figure 20: Image after removing noises

After removing all the noises, higher order polynomial is needed to incorporate to detect curved roads. For that, the algorithm must have an awareness of previous detection and a value of confidence. In order to fit a higher order polynomial, the image is sliced in many horizontal strips. On each strip, straight line algorithm was applied and pixels were identified corresponding to lane detections. Then, the pixels will be combined all the stripes and a new polynomial function will be created that fits the best (Lane detection for self driving vehicles, 2018). Finally, a blue mask is attached on the lane to see the difference between the lanes and the other images on the road. Figure 21 shows the lane detection on an image.

Figure 21: Lane Detection

5.2 Speed Detection

Speed detection in autonomous vehicles is very crucial to reduce number of accidents on the road. According to Salvatore Trubia (2017), one of the countermeasure is the use of in-vehicle technology to help drivers to maintain the speed limits and also prevent the vehicle from overtaking the speed limit. Therefore, determining speed of the vehicle is very important to keep the autonomous vehicle from crashing or exceeding the speed limits (Trubia et al., 2017).

In this project, white line markers are used to measure the speed of the vehicle. When a line marker is present, number of frames will be tracked until the next lane marker appears in the same window. Then, the number of frame rate of the video is also taken which will be extracted from the video to determine the speed. Figure 22 shows the architecture to calculate the linear speed of the vehicle.

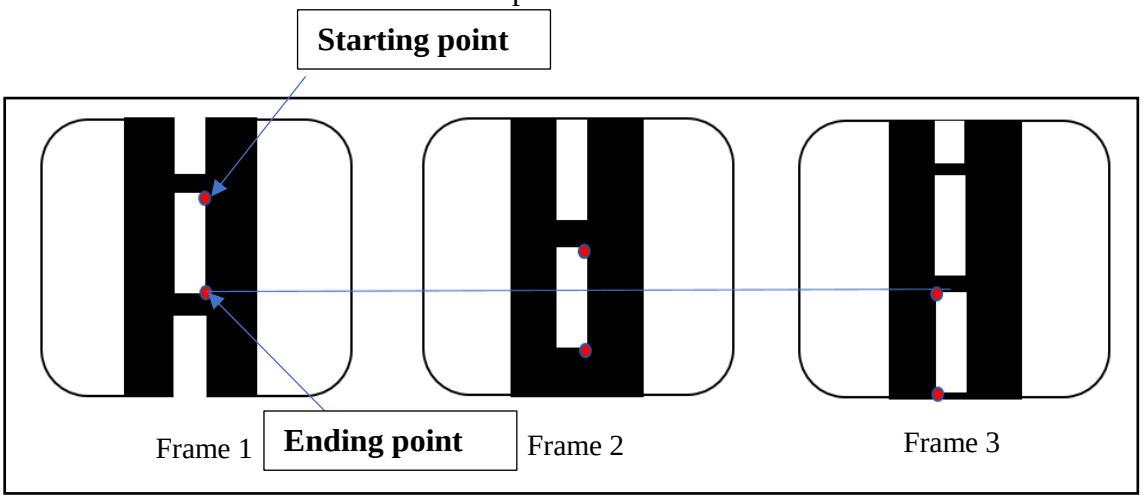


Figure 22: Architecture to calculate linear speed of the vehicle

Based on the Figure 24, on a straight road, the far white lines will be ignored because the distance between the dash cam and the white line is very far from the video. Therefore, nearest white straight line which can be seen fully from the dash cam will be taken. Starting and endpoint of the white dashed line will be taken to measure the speed. When it goes from one frame to another, the position of the white dashed lines will be measured (mcCormick, 2017). Then, when the white dashed lines reach the end point of it, the number of frames took to reach the point will be calculated. Equation 4 is the calculation to calculate the speed:

$$Speed = \frac{distance}{time}, where\ time\ is\ number\ of\ frames \quad (4)$$

To get the position of the white dashed lines, a cv2.rectangle is used. A rectangle is drawn in between of two lines. Then, using matplotlib, the output is plotted to see the position of rectangle. It is plotted to the nearest white line. Figure 23 shows the output of the drawn rectangle.

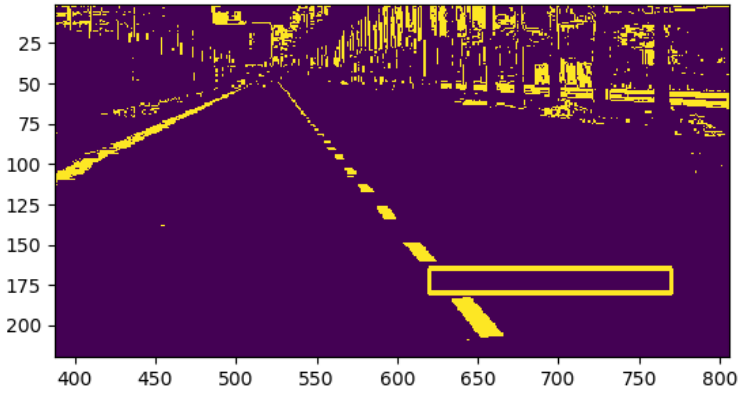


Figure 23: Output of the drawn rectangle

Next, when the rectangle touches the white line, it will start to count the number of frames to reach the end point of the white lines. If the number of frames more than zero, the speed will be measured using the distance of white lines, number of frames from the video (scaling factor) divided by frame counts. Figure 24 shows the code snippet to measure the speed of vehicle.

```
if pixelCounts > 70:
frameCounts = frameCounts + 1
elif frameCounts > 0:
speed = round ( 10*50*30/100 / frameCounts, 1)
print ('{0} - Frame count: {1}, Speed: {2}
km/h'.format ( speedCount, frameCounts, speed ))
frameCounts = 0
```

Figure 24: Measure speed

For instance, if the number of frame counts is 14, then the speed of the vehicle will be 10.7km/h. This is calculated using the equation 4. Figure 25 shows the output using matplotlib and cv2 and Figure 26 shows the speed of the vehicle on a straight road.

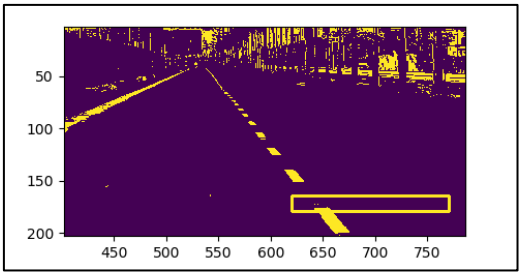


Figure 25: Output of speed detection using matplotlib

Figure 26: Speed detection output

To measure the speed on the curve, the same method is used. However, the position of rectangle will be changed according to the position of the white lines. Figure 27 shows the speed of the vehicle on a curve road.

Figure 27: Speed Detection output for curved road

5.3 Road Angle Detection

According to Cuenca (2019), vehicle speed and steering angle are required to develop autonomous driving. Supervised learning algorithms such as linear regression, polynomial regression and deep learning are used to develop the predictive models. In this project, due to lack of hardware support, the road angle is taken to predict the steering angle of the vehicle. The video dataset is used to measure real-time road angle.

Firstly, curve fitting model is used to find the best-fit function (curve) along a set of range. This will capture the trend in data and predictions can be done in the future. In this project, polynomial curve-fitting which is known as polynomial regression is used. Polynomial curve-fitting is used because the dataset was correlated but the relationship is not linear (Pant, 2019). Figure 28 shows the difference between linear regression and polynomial regression.

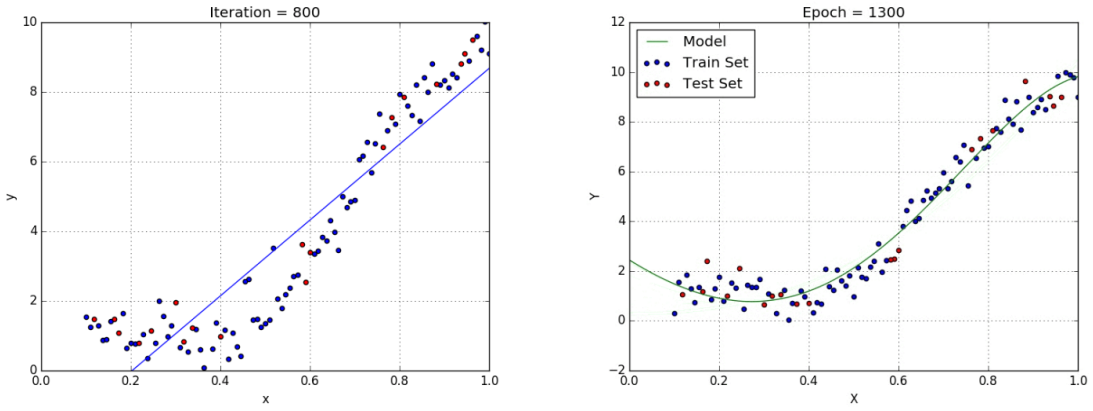


Figure 28: The difference between linear regression and polynomial regression

Based on the Figure 28, it is evident that if a simple linear regression is used to find the road angle, then the linear regression line will not fit properly because of the low value error. Therefore, a polynomial regression is used to fit a polynomial line to get a minimum cost or minimum error. One of the advantages using this model is it fits a wide range of curvature. The equation of a polynomial regression (Curve Fitting and Interpolation, 2019) is:

$$y = a_o + a_1 x + a_2 x^2 + e \tag{5}$$

Then, Rosenbrock function (Croucher, 2011) is created to optimize the algorithms. The mathematical equation is stated in Equation 6:

$$f(x, y) = \mathrm{n} \sum i = 1[b(x_i + x_1^2)^2 + (a - x_i)^2] \tag{6}$$

In this formula, a and b parameters represent the constants and usually set to a=1 and b=100. Figure 29 shows the code snippet for Rosenbrock function.

```
def curve ( x, t, y ):
    return x[0] * t * t + x[1] * t + x[2] - y
```

Figure 29: Rosenbrock function

Then, a rectangle is drawn on the white-dashed line. Then, a for loop is created to loop through every row of white lines. Pixel counts and column count is set to zero while an empty is created for row, column and median. After that the centre of white pixels are calculated in a loop where each column will loop through in a range. If white is identified, pixel count will be incremented by one and column will be added into column count. Based on this loop, the centre of white pixels can be identified and added into the least square equation. Least square regression line is then found which makes the vertical distance from the data points to the line of regression as minimum as possible. This is called “least squares” because the best line of fit the one that reduces the variance (Genn, 2014). Figure 30 shows the code snippet to find least squares.

```
if len ( arrRow ) > 0:
x0 = np.ones ( 3 )
res_robust = least_squares(curve, x0, loss='soft_l1',
             f_scale=0.1, args=(np.array(arrRow),
             np.array(arrCol))) #find the a0, a1, a2 which
             wil be stored in coeff
coeff = res_robust.x
```

Figure 30: Least squares

Based on the code snippet, f_scala is set to 0.1, this means the inlier residuals must not exceed 0.1 (noise level used). For the loss, soft_|1 is used because it is a smooth estimation of |1 (absolute value) loss. This line of code will find a0, a1, a2 which will be stored in a coefficient. After this, the difference angle is taken. Figure 31 shows the method how the difference angle is taken in real-time.

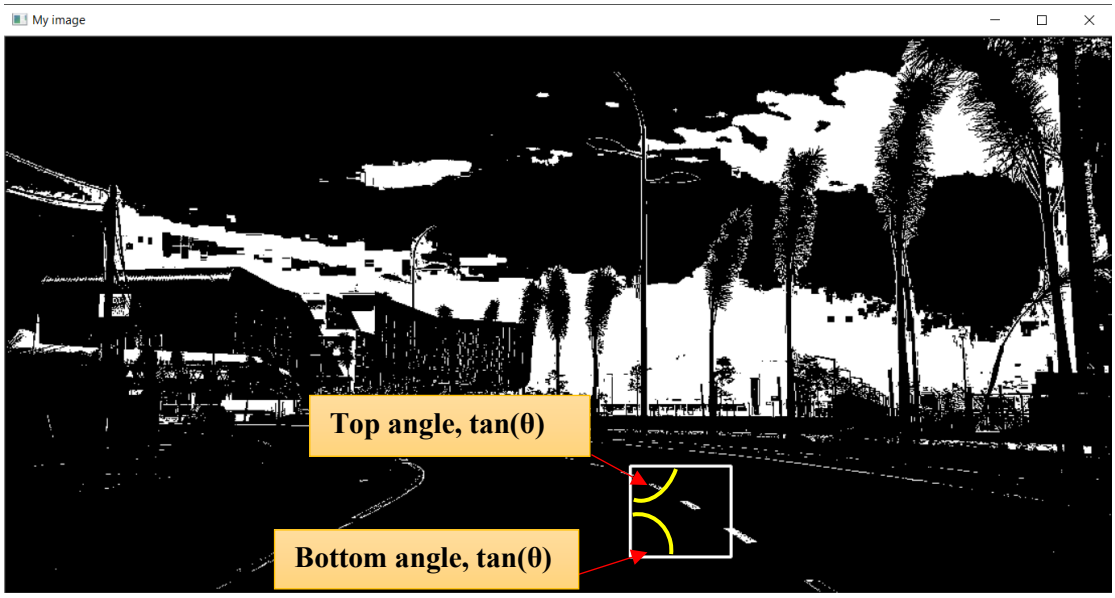


Figure 31: Method to find the angle difference of a road

Based on the Figure 31, the difference angle for each frame is taken by subtracting top angle with bottom angle of the rectangle which is being placed on the white-dashed lines. However, the average angle is calculated using a formula called exponential moving average, (EMA). The formula was obtained from simple moving average, (SMA). The difference between EMA and SMA is in SMA the average of the road angle is calculated whereas in EMA, more weight is given to the current data (Majaski, 2019). This means, EMA will have more impact on the angle compared to SMA which will have a lesser impact. More precisely, EMA have a higher weighting to the angle and SMA have equal weighting to all values. Equation 7 splits down the elements of the calculation which provides a clear visualisation (Venketas, 2019).

$$EMA = Difference\ Angle\ (t)\ \times k + EMA\ (y)\ \times\ (1 - k) \qquad (7)$$

Where:

t = difference angle

y = average angle

k = alpha

After getting the EMA, the top angle, bottom angle, difference angle and average angle is showed in the output video for every frame. Figure 32 shows the code snippet and output to get the angle of the road.

```
startRow = arrRow [ 0 ]
middleRow = arrRow [ len ( arrRow ) // 2 ]
endRow = arrRow [ len ( arrRow ) - 1 ]

startCol = coeff [ 0 ] * startRow * startRow + coeff [ 1 ] *
startRow + coeff [ 2 ]
middleCol = coeff [ 0 ] * middleRow * middleRow + coeff [ 1 ] *
middleRow + coeff [ 2 ]
endCol = coeff [ 0 ] * endRow * endRow + coeff [ 1 ] * endRow +
coeff [ 2 ]

#mTop = ( middleRow - startRow ) / ( middleCol- startCol )
#mBottom = ( endRow - middleRow ) / ( endCol - middleCol )

angleTop = math.degrees ( math.atan2 ( middleRow - startRow,
middleCol- startCol ) )
angleBottom = math.degrees ( math.atan2 ( endRow - middleRow,
endCol - middleCol ) )

diffAngle = angleTop – angleBottom

#exponential moving average
averageAngle = averageAngle * ( 1 - ALPHA ) + ALPHA * diffAngle
```

Figure 32: Find angle of the road

Figure 33: Average angle for straight road

Figure 34: Average angle for curved road

Based on the images in Figure 33 and 34 above, it is proven that when the road is straight, the difference and average angle is minimum whereas when there is a curve, the difference and average angle increases. For the straight road, the average angle is 1.52 degrees whereas when it reaches to a curve road, the average angle is -12.61 degrees. After collecting the angle for each frame, the images were split to each frame in a folder and labelled them in a text file. Then, the angle of the road is matched to each frame in the text file. Figure 35 shows the dataset of the video taken.

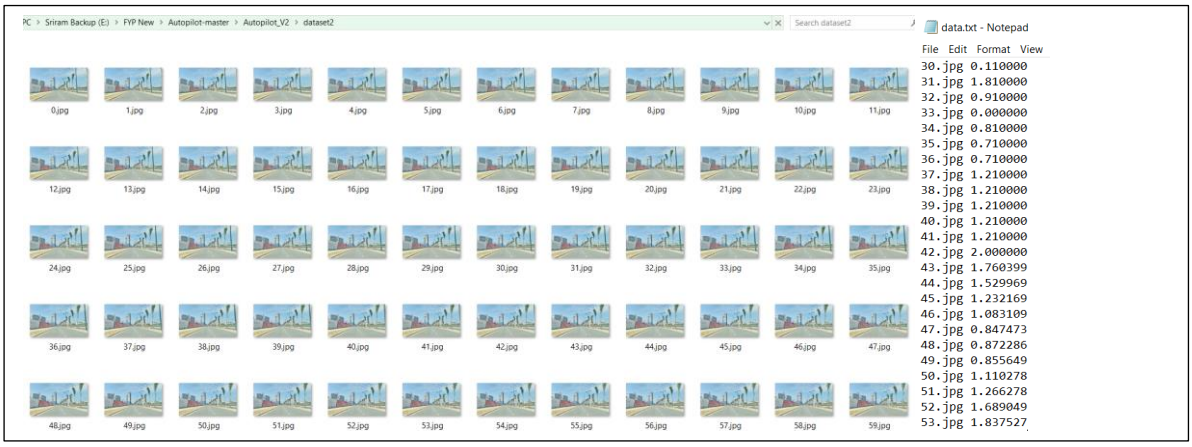


Figure 35: Dataset of the video

5.4 Object Detection

Object detection is a technology that recognise the objects of a class which are semantic in digital image and videos (Geethapriya et al., 2019). For autonomous vehicle, object detection is very important as real time detection is required to identify the objects surrounding the vehicle to avoid obstacles. In this project, multiple objects will be detected from video dataset taken using local roads. Most common object to recognise in this application is the sign boards, surrounding vehicles and traffic signs. Object localisation is used to locate the objects in the video and to locate more than one object in real-time (Redmon et al., 2016). There are several techniques for object detection, which can be split into two categories. The first one is based on classification algorithms such as Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). CNN is used to classify the regions from the image. This method will consume a lot of time because prediction must be run for each region. The second category is by using algorithms based on regressions which is the YOLO method. YOLO stands for "You Only Look Once". For this algorithm, the interested regions will not be selected from the image (Redmon and Farhadi, 2016). According to Geethapriya (2019), the classes and bounding boxes of the whole image will be predicted at a single run of the algorithm and multiple objects will be detected using a single neural network. Bounding boxes are the boxes which are drawn around images and confidence will be shown for each of the regions. Confidence here stating to the probability of an object exist in the bounding box. The Equation 8 is defined below (Huang et al., 2018):

$$C = \Pr(Object) * IOU_{pred}^{truth} \quad (8)$$

Where IOU, intersection over union, describes a fraction between 0 and 1. Overlapping area between the predicted bounding box and ground truth is called as intersection. According to Racheal (2018), he stated that IOU must be close to 1 which indicates the predicted bounding box is close to the ground truth. Next, YOLO uses the Equation 9 to calculate loss (Huang et al., 2018) and optimising confidence:

Loss =

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{A} \mathbb{1}_{ij}^{obj} [(b_{x_i} - b_{\hat{x}_i})^2 + (b_{y_i} - b_{\hat{y}_i})^2] \\
&+ \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{A} \mathbb{1}_{ij}^{obj} [(\sqrt{b_{w_i}} - \sqrt{b_{\hat{w}_i}})^2 + (\sqrt{b_{h_i}} - \sqrt{b_{\hat{h}_i}})^2] \\
&+ \sum_{i=0}^{s^2} \sum_{j=0}^{A} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
&+ \lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{A} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{s^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2.
\end{aligned}
\tag{9}
$$

The loss function is used to correct the centre and the bounding box when making each prediction. Each image is split into an S x S grid, with A bounding boxes for each grid. The b_x and b_y variables describe the center of each prediction, meanwhile b_w and b_h are the dimensions of bounding box. The λcoord and λcoorj variables will increase emphasis on boxes with objects, and decrease the emphasis on boxes with no objects. C describe the confidence, while p(c) is the classification prediction. The loss describes the model's performance. If the loss is lower which means the performance is higher. Besides, the accuracy of predictions made by the models can be calculated via average precision equation which is show in Equation 10 (Huang et al., 2018):

$$
avgPrecision = \sum_{k=1}^{n} P(k) \Delta r(k). \tag{10}
$$

Where P(k) is the precision at threshold k and Δr(k) is the change in recall. YOLO neural network architecture have 24 convolutional layers and 2 fully connected layers.

In this project, YOLO algorithm is used for object detection as it is faster compared to other classification algorithms. In real time, YOLO algorithm process 45 frames per second (Redmon and Farhadi, 2016). Furthermore, even though it makes localisation error, less false positives will be predicted in the background. To make use of YOLO algorithm, an image is divided as grids of 3 x 3 matrixes. After the image is divided, classification and localisation of the object for each grid will be done (Redmon and Farhadi, 2016). From this, the confidence score of each grid is determined. If no proper object is found in the grid, then the confidence and value of bounding box will

be zero or if there is an object found in the grid, then the confidence will be 1 and the value of bounding box will be its corresponding bounding values of the object that is found. Figure 36 shows the how prediction of bounding box is done.

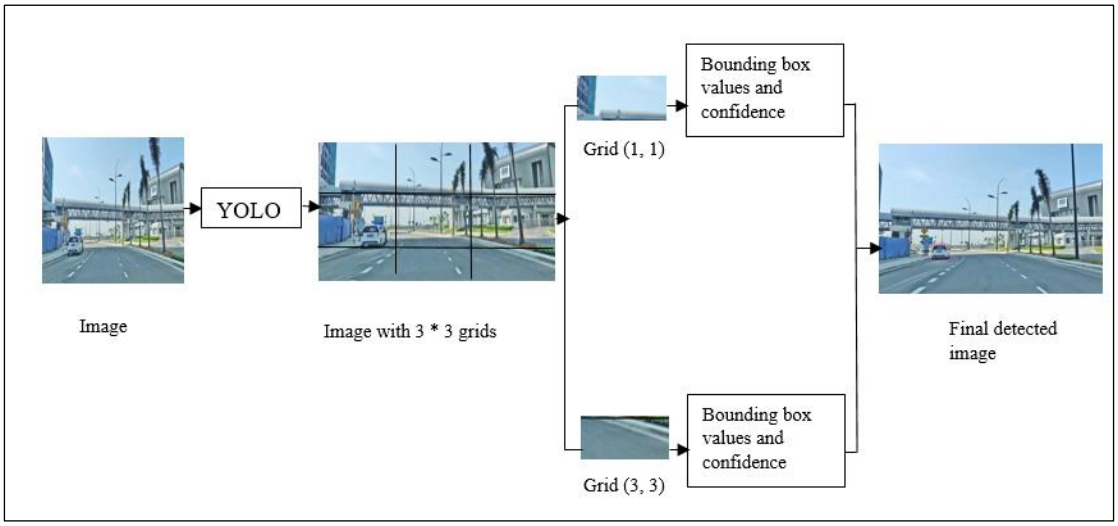


Figure 36: Bounding box

YOLO algorithm is used to predict the bounding boxes accuracy from the image (Geethapriya et al., 2019). The image is divided into S x S grids by predicting the bounding boxes for each grid and class probabilities. Each grid is labelled with a label. After that, each grid is checked separately and marks the label that has an object in it while the bounding boxes will be marked. If the object is not found, then the labels will be marked as zero (Geethapriya et al., 2019). Figure 37 shows an image with 3x3 grids.

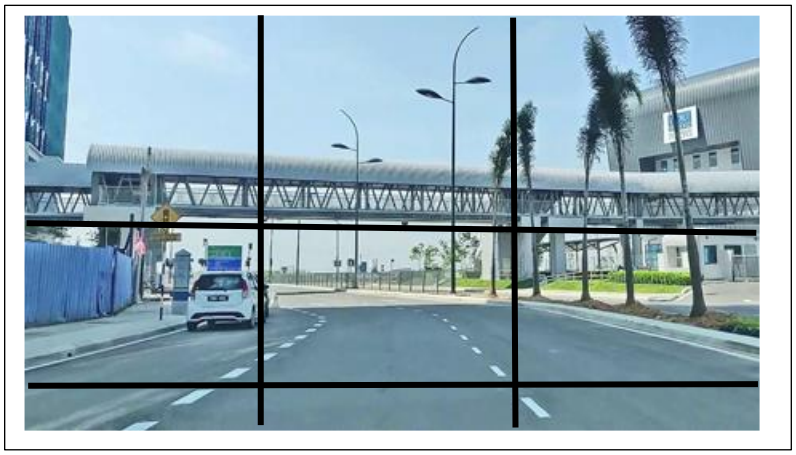


Figure 37: Image with 3 x 3 grids

Based on the Figure 38, an image from a video dataset is taken and it is split into 3 x 3 matrixes. Labelling has done for each grid and both classification and localisation have been done. Y is considered as the label which consist of 8 values.

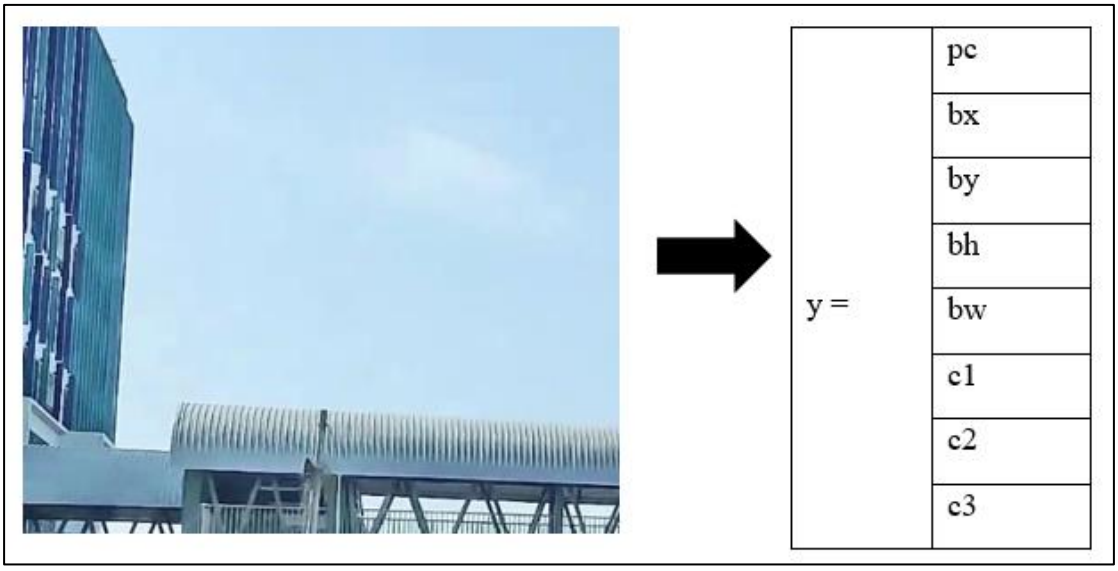


Figure 38: Grid 1 of the image

Figure 38 shows the elements of label Y. Pc represents whether an object is found on the grid or not. If found pc = 1 else 0. Bx, by, bh and bw represent the bounding boxes (if found) meanwhile c1, c2 and c3 are the classes. If the object is a car, then c2 will be 1 whereas c1 and c3 will be 0.

y=	0
	?
	?
	?
	?
	?
	?
	?

Figure 39: Y elements of Grid 1

In this grid, there are no proper objects, therefore the pc value will be 0. For other values, they are represented as ?, because there are no existence of an object. Figure 40 shows an object exist on the 4th grid of the image.

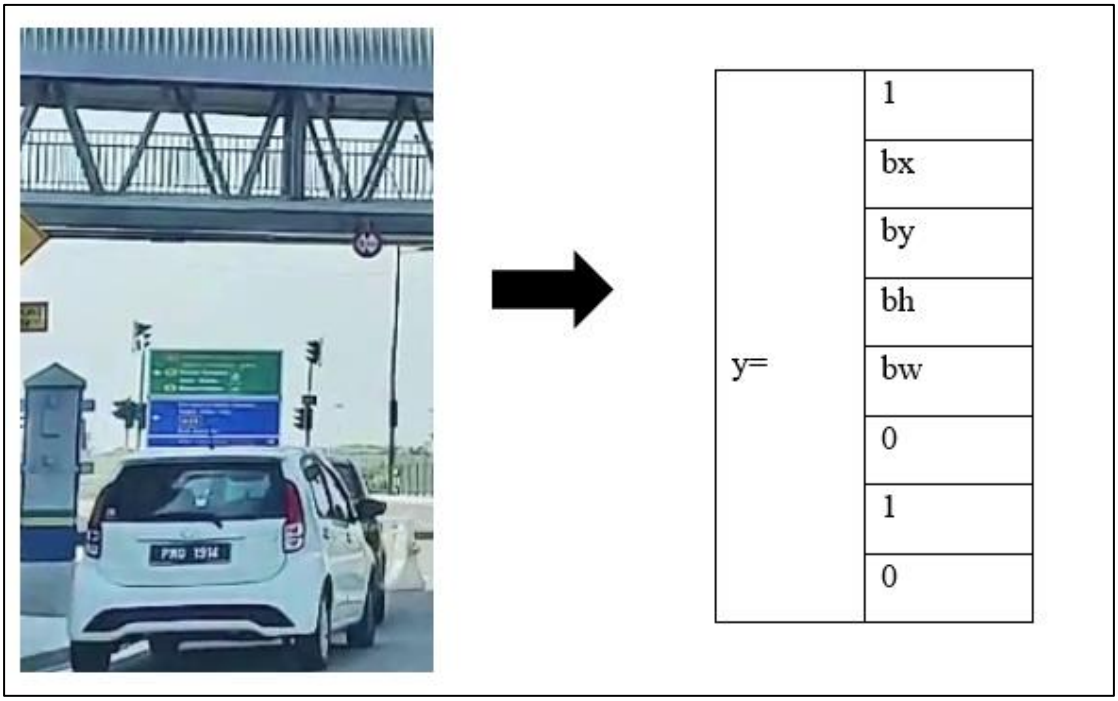


Figure 40: Grid 4 of the image

Based on the Figure 40, 1 represents the presence of an object. Meanwhile, bx, by, bh and bw are the bounding boxes of the object in the 4th grid. The object in this grid is a car. Hence the classes are (0, 1, 0). The matrix formed of Y is Y=3x3x8. If two or more grids contain the same image, then the centre point of the object is taken. IOU will be used for this case. If the value of IOU is more than or equal to threshold value of 0.5, then it is a reasonable prediction. Geethapriya (2019) stated that, higher threshold can be taken to increase the accuracy as the threshold is just an assuming value.

For this project, darknet, the framework to develop YOLO was used to train and test the models. The training and testing were done in Predator GTX 1080 GPU. The model was trained in two datasets, combination of PASCAL VOC 2007 and 2012. The model which had the highest performance trained on PASCAL VOC and then it was retrained on the second dataset, COCO 2014 which contains 80 classes with around 40,000 training images. According to Table 2, it is evident that YOLOv2 performs on par with other detectors such as SSD512 and Faster R-CNN and is 2 – 10x faster.

Table 2: Comparison of YOLO with other algorithms (Huang et al., 2018)

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Figure 41 shows the output of the object detection when run in video dataset.

Figure 41: Object Detection Output

5.5 Autonomous Navigation Training

To develop navigation for autonomous vehicle, convolutional neural network (CNN) was trained to map raw pixels from smartphone's front-facing camera to steering commands. This end to end deep learning system learns to drive on local roads with or straight or curve roads. Increase in computational power allowed us to train deep neural network to act as a "brain" for these vehicles which then understands the vehicle's surroundings and make navigation in real time. Figure 42 shows the block diagram of AutoMove's training system. Images are fed into a CNN which then comes up with a proposed steering angle. The proposed steering angle is then compared with the actual steering angle and the weights of CNN are modified to make the output more accurate to the expected output.

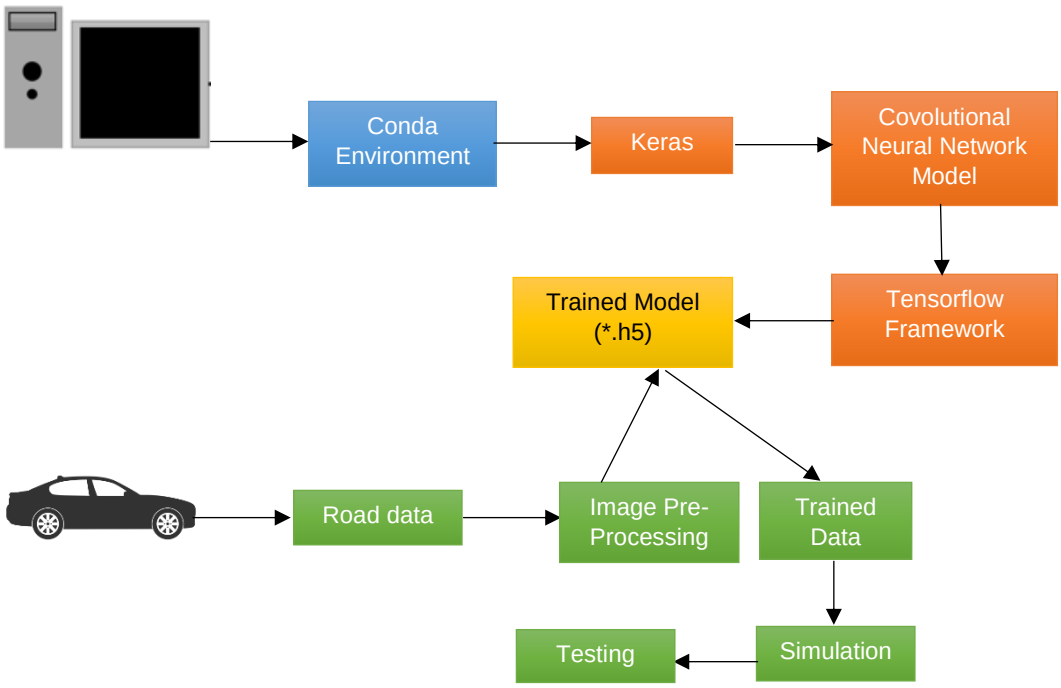


Figure 42: Deep Learning Neural Network model on the proposed system

Once the network is trained, it is used to generate steering angle of the local roads from a single front-facing camera.

5.6 Dataset

To develop this project, a new dataset was created. A One Plus 6 model smartphone was placed on a car's dashboard. A video of the driving was taken at Jalan Barat Cassia 2, Batu Kawan, Penang which consist of straight and curve roads. The video size is 1920 x 1080 and the frame rate is 30fps per second.

5.7 Data Augmentation

A classic convolutional neural network can have many parameters and to tune this, millions of training instances of uncorrelated data are needed. However, this may not always be possible. Deep neural networks might over fit the data when there are limited dataset. Data augmentation is done to avoid over-fitting (Yadav and Mody, 2017). For instance, for a more generalised dataset, driving under different weather is required. For this project, thousands of new instances of the image will be generated in real time, it is not practical to keep these images on the disk. Hence, pickle is used to read the data from the folder, augment and use it to train the model. One of the augmentation techniques used in this project is brightness augmentation. The aim behind brightness augmentation is to detect the lanes on a darken image (Yadav and Mody, 2017). Figure 43 shows the original and brightened image.

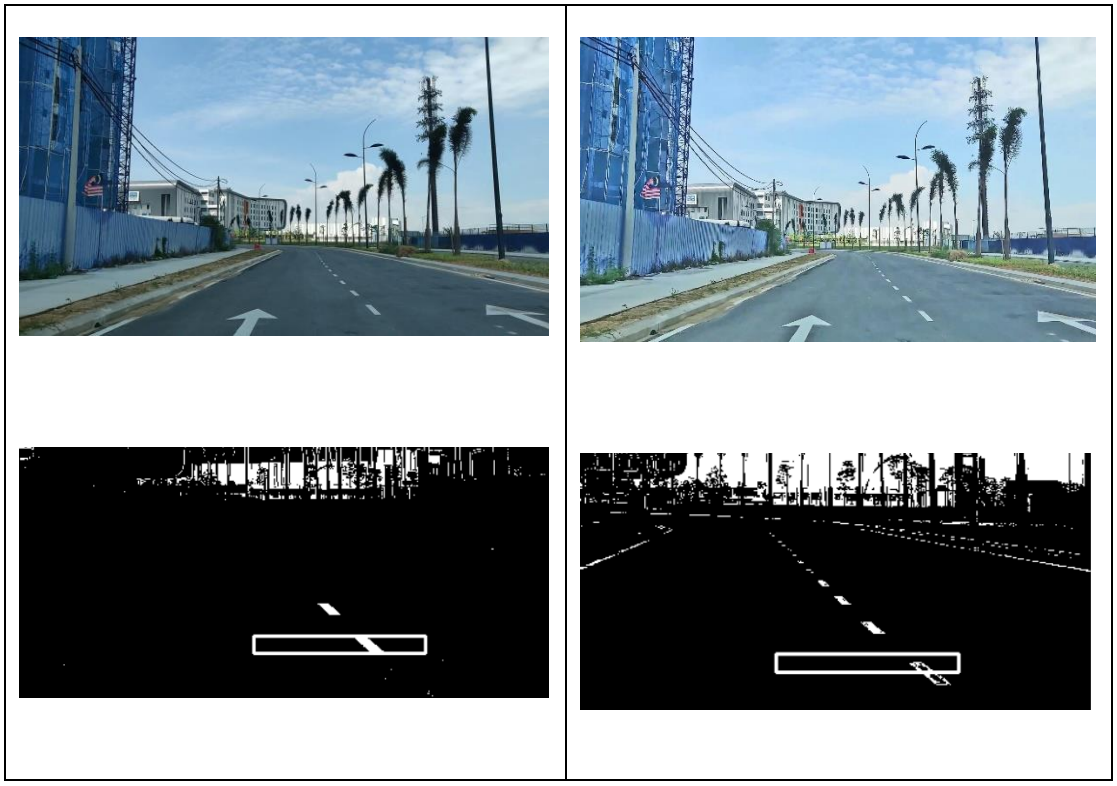


Figure 43: Difference between original and brightened image

Figure 43 shows the difference between the original image and brightened image. It is evident that in the brightened image, white lanes can be seen more clearly compared to the one in original image. This is the reason why, brightness augmentation is being done so that, the algorithm can understand the markings. After this process, pre-processing take place. Figure 44 shows the code for pre-processing.

```
DATA_FOLDER = 'dataset2'
TRAIN_FILE = os.path.join(DATA_FOLDER, 'data.txt')

def preprocess(img):
    resized = cv2.resize((cv2.cvtColor(img,
    cv2.COLOR_RGB2HSV))[:, :, 1], (100, 100))
    return resized
```

Figure 44: Preprocessing function

Based on the code snippet, img is set as a parameter for preprocess function. Then, by using cv2, the image is resized and the image is converted from RGB to HSV. According to Robot (2015), image descriptions in RGB makes object discrimination difficult. This is because, RGB components of an object's colour are all correlated with the intensity of light hitting the object whereas for HSV, it abstracts colour (hue) by splitting it from pseudo-illumination and saturation which makes the algorithm to identify the image easily. Then images are then resized to (100,100). Next, a pickle file

is created to gather all the images from a folder called 'dataset2'. Figure 45 shows the code snippet to create a pickle file to store the images.

```
# This function help to create a pickle file gathering all
the image from a folder
def return_data():

    X = []
    y = []
    features = []

    with open(TRAIN_FILE) as fp:
        for line in islice(fp, LIMIT):
            path, angle = line.strip().split()
            full_path = os.path.join(DATA_FOLDER, path)
            X.append(full_path)
           # using angles from -pi to pi to avoid rescaling
           the atan in the network
            y.append(float(angle) * scipy.pi / 180)

    for i in range(len(X)):
        img = plt.imread(X[i])
        features.append(preprocess(img))

    features = np.array(features).astype('float32')
    labels = np.array(y).astype('float32')
    # we save the newly created data base
    with open("features", "wb") as f:
        pickle.dump(features, f, protocol=4)
    with open("labels", "wb") as f:
        pickle.dump(labels, f, protocol=4)

return_data()
```

Figure 45: Creating pickle file

According to the return_data function, first a for loop is created to take the data from TRAIN_File which is data.txt. Line.strip.().split() is used to separate the path(image name) and angle of the road. Then the images are added to a list by using .append. After that, scipy.pi is used to avoid the angles from rescaling the atan. Next, the files are saved to a pickle file in features and labels using pickle.dump. This will store the images and angle data in a pickle file.

5.8 Deep ConvNet Model

In this project, the model is inspired from NVIDIA'S CNN architecture End to End learning from self-driving cars (Bojarski et al., 2016, 3-6). The model contains three 5x5 convolutional layers with 2x2 stride. Furthermore, the depth of the network is 24, 36 and 48 respectively. After that, it is followed by two 3x3 convolutional layers with 1x1 stride and depth of 64. Batch Normalisation is done after each convolutional layer. The output of the final convolutional layers is flattened before reaching the fully connected phase (Zuccolo, 2017). Series of fully connected layers are used which consists of 1152, 200, 50 and 10, 1 respectively. The steering angle will be predicted in the last layer. The activation function used for all layers are Rectified Linear Activation (ReLU). The reason why ReLU is picked rather than sigmoid as the activation function is because, according to (Krizhevsky et al.), ReLU is more computationaly fater to compute compared to sigmoid functions. This is because ReLU must only pick max (0,x) and expensive exponential operations will not be performed as in Sigmoid. Furthermore, ReLU also show better convergence performance compared to sigmoid. This model uses RMSE as evaluation metrics with adam as optimizer for the loss function. Adam optimiser is used so that the learning rate was ot tuned manually. In general, this model consists of 1,229,193 parameters. Batch Normalisation after each convolutional layer will help to use higher learning rates and assist the model to be more independent from the initialisation parameters. Moreover, it also act as a regulariser in some cases to eliminate the need for Dropout. Figure 46 shows the NVIDIA architecture and in Figure 47 the visualisation of the implemented architecture can be seen. (Zuccolo, 2017)

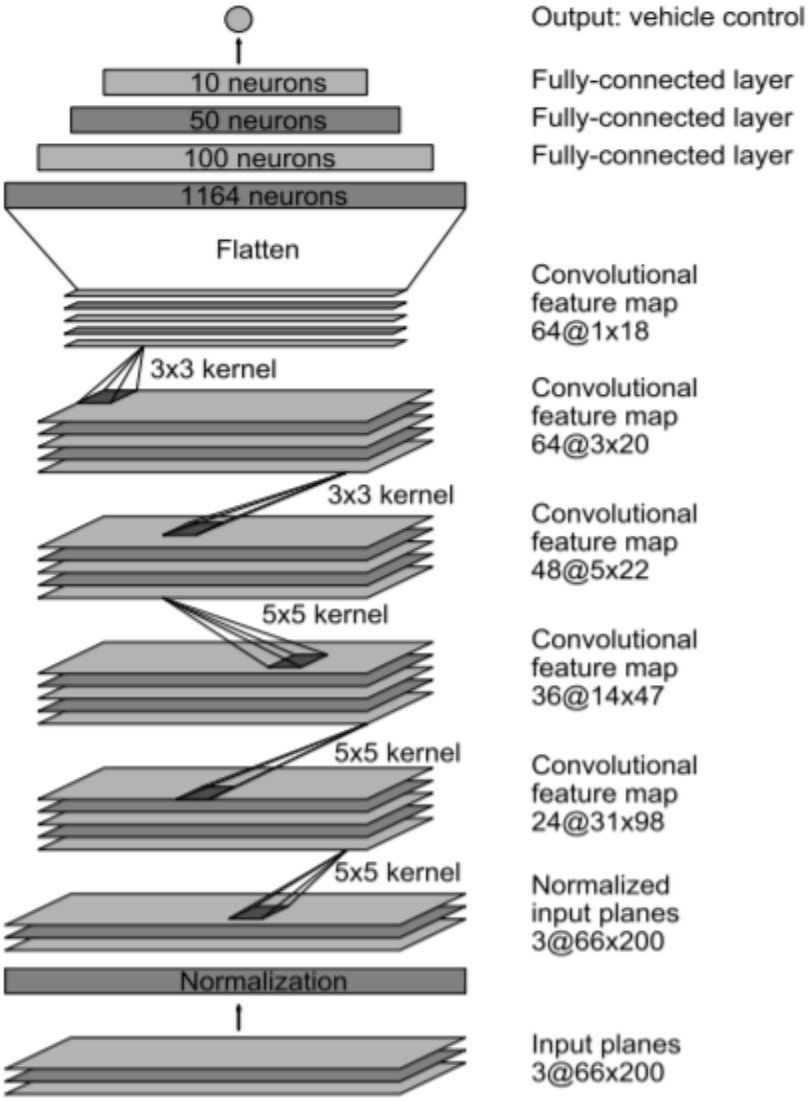


Figure 46: NVIDIA architecture

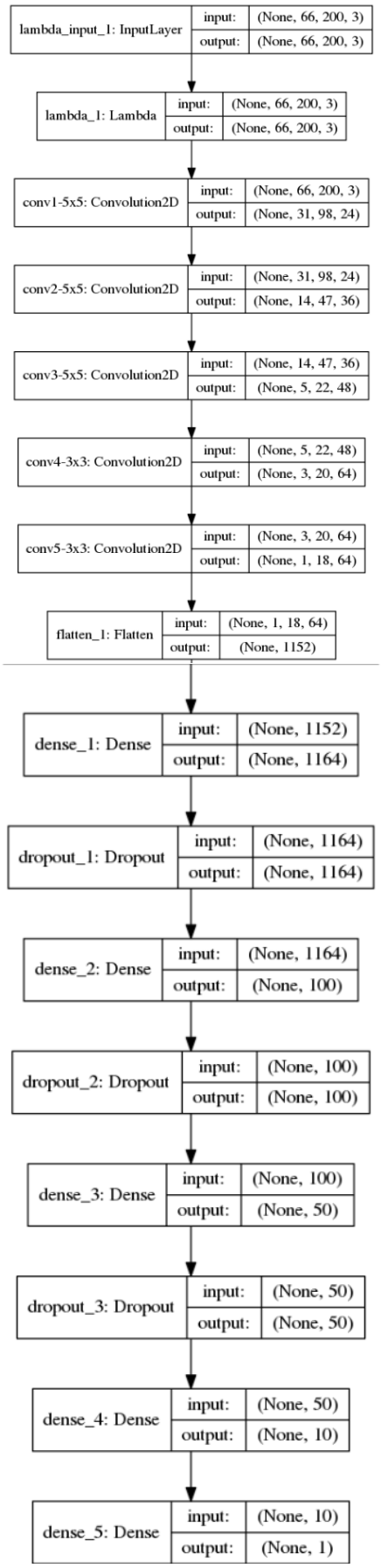


Figure 47: Layers overview

Figure 48 shows the code snippet to build a CNN in keras to train the model. According to the official documentation of keras, keras is a high-level neural network API, which is written in Python and is capable to run on top of Tensorflow (Home - Keras Documentation, 2019). In this project, keras is used to speed up the experimentation.

```
def keras_model(image_x, image_y):
    model = Sequential()
    model.add(Lambda(lambda x: x / 127.5 - 1.,
    input_shape=(image_x, image_y, 1)))
    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(128, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(128, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(1024))
    model.add(Dense(256))
    model.add(Dense(64))
    model.add(Dense(1))
```

Figure 48: keras model function

Figure 48 shows the function for keras_model. Firstly, Sequential is used as the model type. This is because, sequential is the simplest method to build a model in Keras. It allows to build a model layer by layer. Next, 'add()' function is used to append layers to the model. The model includes data normalisation/zero mean by 255/127.5-1 using a Keras lamda layer, 5x5 and 3x3 convolutions layers using Keras Convolution2Dm, introducing nonlinearity using RELU layers, fully connected layers using Keras Flatten and Dense, and for overfitting function, Keras Dropout is used. The loss is calculated using mean square error (mse) and adam optimiser. The first layer is Conv2d layer. This layer will deal with the input image that are seen as 2-dimensional matrices. The number of nodes in first layer is 32. This number can be set to be higher or lower, depending on the dataset's size. Therefore, for this project it set from 32, 64 and 128. Kernel size is the size of filter matrix. In that case, a size of 3 means, the filter matrix

will be 3x3. Next, ReLU is used as the activation function of the layer. ReLU has been proven to work well when handling neural networks. Furthermore, the first layer takes an input shape which consist of image_x, image_y, 1, with the 1 converts the images to grayscale. In between the layers, MaxPooling2D is used with 2x2 filter. This is to avoid over-fitting by providing an abstracted form of the representation. Next, 'Flatten' layer is used as it connects the convolution and dense layers. Then, there is a 'Dense' layer which is used for output layer. It is a standard layer type which is used commonly for neural networks.

Next, the model is compiled. It takes three parameters which are optimizer, loss and metrics. The optimizer controls the learning rate. For optimiser, 'adam' is used as adam optimizer as it adapts the learning rate during training. The learning fate will determine how fast the best weight is calculated for the model. If the learning rate is smaller, then the weights might be more accurate but the time taken to compute will be longer. Mean square error (mse) is used for the loss function. The model was train on 2937 samples and validated on 1259 samples with 30 epochs. Then the best model is saved as Autopilot.h5. Figure 49 shows the code snippet of compiling the model.

```
model.compile(optimizer='adam', loss="mse")
filepath = "Autopilot.h5"
checkpoint = ModelCheckpoint(filepath, verbose=1,
save_best_only=True)
callbacks_list = [checkpoint]

return model, callbacks_list
```

Figure 49: model compilation

Figure 50 shows the model summary after trained. It contains dropout layers and validated to ensure the model did not over fit. The model is then tested by in a simulator to ensure the vehicle stay on the track.

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 100, 100, 1)	0
conv2d_1 (Conv2D)	(None, 100, 100, 32)	320
activation_1 (Activation)	(None, 100, 100, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 50, 50, 32)	0
conv2d_2 (Conv2D)	(None, 50, 50, 32)	9248
activation_2 (Activation)	(None, 50, 50, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 25, 25, 32)	0
conv2d_3 (Conv2D)	(None, 25, 25, 64)	18496
activation_3 (Activation)	(None, 25, 25, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 64)	0
conv2d_4 (Conv2D)	(None, 12, 12, 64)	36928
activation_4 (Activation)	(None, 12, 12, 64)	0
max_pooling2d_4 (MaxPooling2	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 6, 6, 128)	73856
activation_5 (Activation)	(None, 6, 6, 128)	0
max_pooling2d_5 (MaxPooling2	(None, 3, 3, 128)	0
conv2d_6 (Conv2D)	(None, 3, 3, 128)	147584
activation_6 (Activation)	(None, 3, 3, 128)	0
max_pooling2d_6 (MaxPooling2	(None, 1, 1, 128)	0
flatten_1 (Flatten)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1024)	132096
dense_2 (Dense)	(None, 256)	262400
dense_3 (Dense)	(None, 64)	16448
dense_4 (Dense)	(None, 1)	65

Total params: 697,441
Trainable params: 697,441
Non-trainable params: 0

Figure 50: Model summary

Figure 51 shows the model loss of the model in a line graph. As we can see, the training loss for train and test data move closer to each which says the accuracy is quite high for the steering angle.

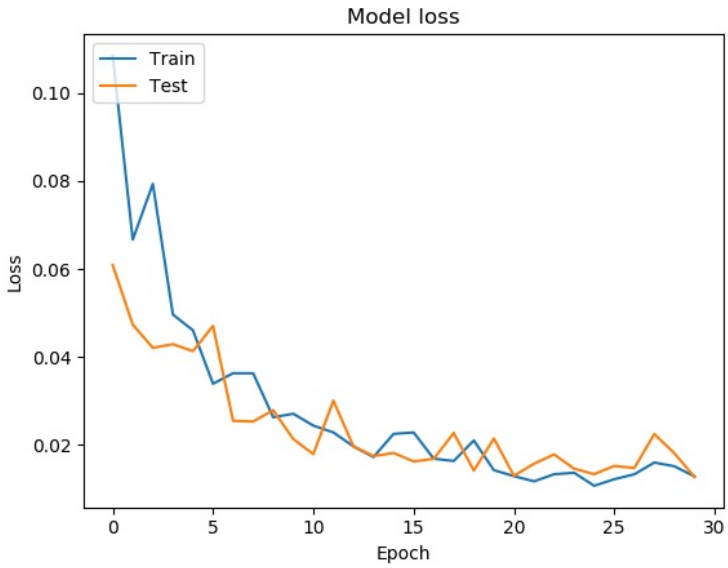


Figure 51: Model loss graph

Table 3 shows the result for model loss. For training data, the loss is 0.0156 whereas for testing data the val_loss is 0.100. Hence, the difference between them is less, so accuracy is high.

Table 3: Model loss result

Loss (train)	**Val_loss (test)**
0.0156	0.0100

5.9 Autonomous Navigation

In this section, we will discuss about the methods to navigate the autonomous vehicle using the trained model. Firstly, a the trained model (Autopilot.h5) is loaded. Then, the model is predicted using a Keras model. According to the official Tensorflow documentation (tf.keras.Model | TensorFlow Core r2.0, 2019), model groups layers into an object with training and inference features. The model then predicts the processed image which is the image that was fed into the training model. After that, a steering image is read from the local disk to visualise the steering angle difference on the output window. Then, the video of the local road were set as the output to predict the steering angle.

To determine the accuracy of the steering angle, the difference between actual and predicted steering angle were measured. At the last frame of the video, the accuracy for the whole model is 91.3784%. Table 4 shows the actual and predicted angle for straight road.

Table 4: Actual and predicted angle for straight road

Actual steering angle (°)	Predicted steering angle (°)	Accuracy (%)
0.4912	3.1006	91.3743
0.5635	2.7528	91.3754
0.3902	-0.2838	91.3771
-0.3001	-1.2578	91.3784

Figure 52 shows the output of autonomous navigation on a straight road. Based on the output, it can be seen that, the steering of the vehicle is quite straight as the steering wheel does not turn on a straight road. This is because, the steering angle is in between zero to five degrees which was set to drive straight.

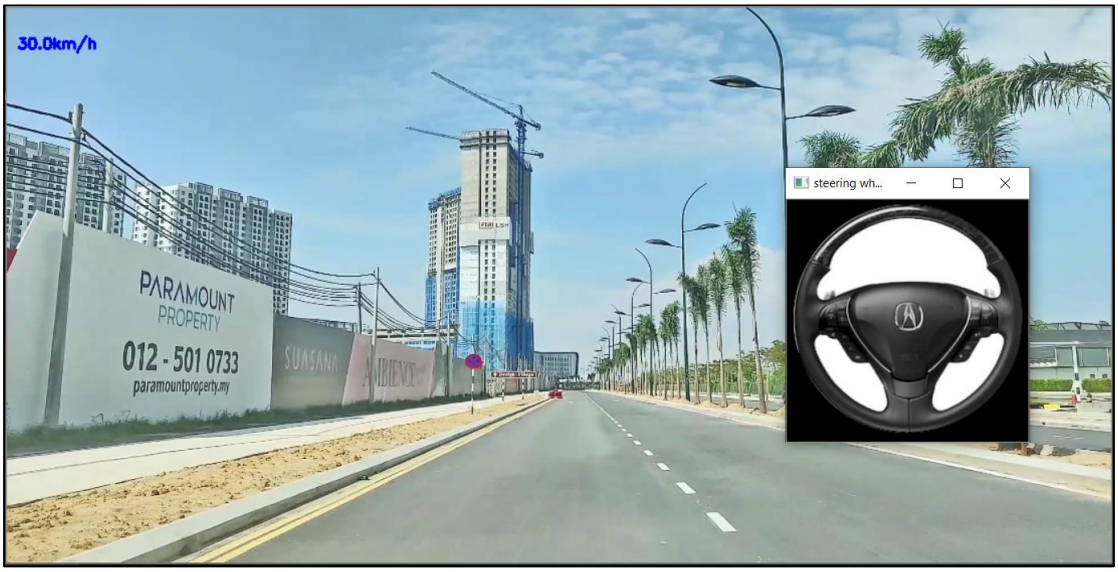


Figure 52: Output of Autonomous Driving on a straight road

However, when the road is curved, the accuracy reduces to 85% which can be seen in Table 5. Furthermore, the steering wheel is set according to few parameters, for instance, if the steering angle is less than zero degrees, then the steering wheel will turn left whereas if the steering angle is more than five degrees, the steering wheel will shift to right. Table 5 shows the actual and predicted angle for curved road.

Table 5: Actual and predicted steering angle for curved road

Actual steering angle (°)	Predicted steering angle (°)	Accuracy (%)
-10.8001	-9.5169	85.8128
-10.8005	-9.8750	85.8192
-10.3392	-7.8328	85.8265
-10.7338	-10.0125	85.8334

Figure 53 shows the output of autonomous navigation on a curved road. Based on the output, it can be seen that, the steering of the vehicle is shifting to left. This is because, the steering angle is lower than zero degree which was set to turn the steering wheel to left.

Figure 53: Output of Autonomous Driving on a curved road

5.10 Summary

In this chapter, implementation of every function are clearly discussed and justified with code snippets and the methods to develop them. In next chapter, types of testing done for this system will be discussed.

Chapter 6: Testing

6.0 Overview

In this chapter, the overall testing done for End to End Deep Learning for Self-Driving Vehicles will be discussed. This is to evaluate the application based on the specified requirements. In this project, several testing methods were used to justify the quality of the product for end user to use it without any issues. Moreover, test plans and test results will also be discussed under this chapter.

6.1 Testing methods

Testing methods that are used for this system are black-box testing and white-box testing. Details regarding these testing are discussed in 6.1.1 (Black Box Testing) and 6.1.2 (White Box Testing). Besides that, test cases and the results are attached in Appendix D: Test Plans.

6.1.1 Black Box Testing

Black Box Testing is a testing technique that test the functionality of the Application Under Test (AUT) without seeing the internal code structure, knowledge of internal paths of the software and also the implementation details. This type of testing is based on software specifications and requirements. In general, this testing will only focus on input and output of the software system without the internal knowledge of the software. Firstly, the specifications and requirements of the system are checked. Then, a tester chooses input that is valid and also few invalid inputs to check whether the system can detect them. Then tester writes down the expected outputs for all those inputs. Next, test cases are constructed by software tester with the selected inputs. Then the test cases are executed. Actual outputs and the expected outputs will be compared. If any defects are detected, it will be fixed and re-tested.

One of the black box testing done for this system is testing the lane detection using different set of images. Next, object detection is tested using road video dataset. Then, speed detection is tested using the white-lane marker on the video dataset. The detailed black box test cases are attached in Appendix D.

6.1.2 White Box Testing

White box testing is a testing technique that test a software solution's internal structure, coding and design. For this kind of testing, the code can be seen by the tester. It is usually tested by a developer who have knowledge with the system. There are two important steps in this testing. First is to understand the functionality of a system via its source code. The next step is by creating the test cases and execute them. In this project, the white box test cases are only shown for the important aspects as it would result in too many rest cases if it is

tested for every function. The detailed white box testing test cases are attached in Appendix D.

Unit testing is done for the components in white box testing. Unit testing is where tests which interact directly with the application is created. Usually functions will be checked in unit testing. It ensures each function works well without any errors. Table 6 shows a test case done in white box testing. This test case is important as it will clear the unnecessary noise that are found on the road image so that, the lanes will be detected clearly.

Table 6: Test case 1

Test Case ID: 1	**Function Name:** imclearborder()
Parameters taken: bw	
Test Case Description: Clear the border of the road	
Expected Result: Return bw	
Pre-Condition: Load road.png	
Test Execution Steps: 1. Convert image to grayscale 2. Apply threshold	
Post Conditions: Removes the border outside the lanes of the road	
Test Result: **Passed**	

Table 7 shows another important test case for white box testing. This function will check the speed of the vehicle by calculating the distance between starting and end point.

Table 7: Test case 6

Test Case ID: 6	**Function Name:** checkSpeed()
Parameters taken: -	
Test Case Description: Measure the speed of the vehicle	
Expected Result: Speed will be showed in km/h	
Pre-Condition: Load 30fps.mp4	
Test Execution Steps: 1. User must run the video of the road	
Post Conditions: The speed of the vehicle will be showed in km/h	
Test Result: **Passed**	

6.2 Summary

In general, there were two types of testing done for this system which includes black box testing and white box testing. From these testing, bugs and errors were found out and fixed. In next chapter, product evaluation will be discussed.

Chapter 7: Evaluation

7.0 Overview

This chapter discusses about the functionalities of the application by evaluating the features and also the process to complete the product. This chapter will also explain the idea on the application's overall description and the process went through by the developer to develop this end to end deep learning for self-driving vehicles.

7.1 Product Evaluation

The detailed evaluation of product was done after the implementation and testing phase of the system. Strengths and weaknesses of the system based on the functionality will be discussed in product evaluation. 7.1.1 Product Functionality would explain the evaluation of technical features of the end to end deep learning for self-driving vehicles that contains the fitness of purpose and build quality.

7.1.1 Product Functionality

The main features of this end to end deep learning system are to predict the steering angle of the vehicle. AutoMove has also successfully met the main aim of the project which are detecting the lanes, detecting the speed of the vehicle, detecting the angle of the road and also detecting the object using Tensorflow API object detection. Firstly, to detect the lane, an image of a road is loaded. Then the image is converted to grayscale using cv2.COLOR_BGR2GRAY. This is because grayscale only contains two dimensions compared to RGB which consist of three-dimensional matrix. Smaller data helps the image to do more complex operations in a short period of time. After that, thresholding is applied using cv2.THRESH_BINARY. The threshold value is set to 170 and 255, which enable the image to detect the region of interest clearly. After applying threshold, it is required to remove as much of the noise as possible in a frame. According to the orientation and position of the camera, the lanes are located in the bottom half of the image. Therefore, unnecessary noise at the top of the image was removed. After removing all the noises, higher order polynomial is needed to incorporate to detect curved roads. For that, the algorithm must have an awareness of previous detection and a value of confidence. In order to fit a higher order polynomial, the image is sliced in many horizontal strips. On each strip, straight line algorithm was applied and pixels were identified corresponding to lane detections. Then, the pixels combine all the stripes and a new polynomial function will be created that fits the best. Finally, a blue mask is attached on the lane to see the difference between the lanes and the other images on the road.

Once the lanes are detected, speed of vehicle are measured. In this system, to detect the speed, centre white-lane markers are used. When a lane-marker is present, number of frames will be tracked until the next lane marker appears in

the same window. Then, the number of frame rate of the video is taken which will be extracted from the video. From this, the speed of the vehicle will be calculated using distance / time formula where distance is the distance of the white-lane and time is the time taken of the vehicle to reach the end point of white-lane. Then, the speed will be showed in km/h.

After speed detection, angle of the road is calculated. Polynomial curve fitting model is used to find the best-fit function (Curve) along a set of range. Firstly, the difference angle for each frame is measured by subtracting top angle and bottom angle of the rectangle which is placed on the white-dashed lines. Then, the average angle is calculated by using exponential moving average formula. From this, the average angle of each frame is saved into a .txt file to create a new dataset. Then, object detection is used to avoid the obstacle around the vehicle during autonomous driving. For this system, algorithm based on regressions which is the YOLO method is used. To make use of YOLO algorithm, an image is divided as grids of 3 x 3 matrixes. After the image is divided, classification and localisation of the object for each grid will be done. From this, the confidence score of each grid is determined. If no proper object is found in the grid, then the confidence and value of bounding box will be zero or if there is an object found in the grid, then the confidence will be 1 and the value of bounding box will be its corresponding bounding values of the object that is found.

After that, the model will be trained using Tensorflow and Keras library. The video dataset is split into images frame by frame. Then pre-process is done. A pickle file is created to store all the images to train. A CNN model is built in Keras. After training the model, compilation takes place. It takes three parameters which are optimizer, loss and metrics. The optimizer controls the learning rate. The model was train on 2937 samples and validated on 1259 samples with 30 epochs. Then the best model is saved as Autopilot.h5. To predict the steering angle, the trained model (Autopilot.h5) is loaded. Then, the model is predicted using a Keras model. After that, the model predicts the processed image which is the image that was fed into the training model. Next, a steering image is read from the local disk to visualise the steering angle difference on the output window. Finally, the video of the local road was set as the output to predict the steering angle which will navigate the vehicle autonomously.

7.1.2 Strengths and Weaknesses of Product

There are few strengths and weaknesses for this end to end deep learning for self-driving vehicle system. One of the strengths is the system can be run offline. The user does not need internet connection to run this system as all the data is stored in local disk. Furthermore, there are four functions in this system such as lane detection, speed detection, angle detection and object detection which can

be done when different images and videos are inputted. For example, when a road image is loaded into lane detection code, the algorithm would recognise the lanes on the road. Besides that, objects can also be detected on different dataset as the model is trained with huge amount of data.

This system also has weaknesses. One of them is, high processing power computer is needed to run this system. For example, Predator Orion with GTX graphics card is used to run this system. This is because, the video is processed in real time which may affect the output's speed. Next, the accuracy of the steering angle will be not accurate when different road video is set as an input. Moreover, the speed of the vehicle cannot be detected if the road does not have white-dashed lines. This is because the speed is measured using the distance from the starting point to end point white-dashed line.

7.1.3 Summary

Overall, the system has easy interface with proper functions which users would not find any difficulty to use this system. The functionalities of the system are developed in a way where users can understand the flow of the system. As self-driving vehicles projects are still under development, this system have integrated many functions together that is required for a self-driving vehicle such as lane detection, speed detection and object detection.

7.2 Process Evaluation

In this sub chapter, process evaluation will be discussed which explains the time taken for the developer to develop AutoMove. 7.2.1 explains about the development progress of the project.

7.2.1 Development Progress

The development process of AutoMove will be discussed based on the system development cycle phases and with the project plan that was planned during the project proposal. Then, it will be reviewed how well it have been executed. The duration days might not be matched with the start date and end date which can be referred in Appendix A: Proposal.

7.2.1.1 Initiation Phase

Table 8: Initiation Phase

Initiation Phase	Start Date	End Date	Duration (Days)
Project Plan	15-01-2019	30-01-2019	15
Actual Progress	15-01-2019	28-01-2019	13

In this phase, project title was discussed with project supervisor and appropriate title was chosen. Then, research papers were searched and project initiation document (PID) was prepared. After that, PID is submitted to be reviewed.

7.2.1.2 Planning Phase

Table 9: Planning Phase

Planning Phase	Start Date	End Date	Duration (Days)
Project Plan	31-01-2019	08-03-2019	36
Actual Progress	29-01-2019	05-03-2019	35

Planning phase is one of the most important phases in the project development. Neural network and machine learning were new to the developer back then. Hence, many articles, research papers and online videos were used to understand the concept to develop this system before writing the project proposal draft. Then, the project proposal draft was written and submitted to supervisor and second marker. Then, it was reviewed by second marker and changed were made. Finally, the revised proposal was submitted. The proposal was completed one day earlier as they were only few changes need to be done.

7.2.1.3 Analysis Phase

Table 10: Analysis Phase

Planning Phase	Start Date	End Date	Duration (Days)
Project Plan	07-03-2019	20-04-2019	44
Actual Progress	06-03-2019	24-04-2019	49

Analysis phase is another important phase during project development. Waterfall methodology and spiral methodology are used to develop this system. It is important to analyse the methods and techniques to develop the project before implementation. To start off, few research journals on how tow Tensorflow works were read. Then, types of neural networks either CNN, RNN or ANN were investigated and researched. Furthermore, journals for lane detection and object detection were also studied to determine which function to implement at first. After finding all the readings, literature review was written and submitted to project supervisor for review. CNN was chosen as the neural network to develop this system and YOLO algorithm was chosen for object detection. As the project was relatively new for the developer, the actual time taken to research the journals took extra five days compared to the planned progress. Then, UML diagram was drawn but after discussing with the project

supervisor. The first layer class diagram was sketched to have a rough idea on how the classes from other functions would interact with each other. Use case diagram was also designed based on the functional requirements that were discussed earlier. User interface was skipped as the time taken to develop the core of the system may take longer time.

7.2.1.4 Implementation Phase

Table 11: Implementation Phase

Planning Phase	Start Date	End Date	Duration (Days)
Project Plan	09-09-2019	01-11-2019	53
Actual Progress	02-09-2019	25-10-2019	54

The implementation phase was started in September first week as the developer undergone internship for three months from June to August. It was done according to the functional, non-functional requirements and UML diagrams which were discussed earlier in analysis phase. There was only one change in the functionality part where traffic sign detection was replaced by speed detection. Other than that, most of the functions were implemented during the implementation phase. However, the time taken to complete the implementation of the product took a longer time but it was started one week earlier. As the time taken for to train the model for object detection took some time, project supervisor suggested the developer to use pre-trained YOLO model.

7.2.1.5 Testing phase

Table 12: Testing Phase

Planning Phase	Start Date	End Date	Duration (Days)
Project Plan	02-11-2019	23-11-2019	21
Actual Progress	26-10-2019	15-11-2019	21

Initially, the planned phase for testing was from 2nd November until 23rd November. However, the final semester was cut short from 14 weeks to 13 weeks. Therefore, the developer had to start the testing process earlier than planned. AutoMove system was tested using black box testing and white box testing to ensure the system is working efficiently without any errors or bugs.

7.2.1.6 Report Documentation

Table 13: Report Documentation

Planning Phase	Start Date	End Date	Duration (Days)
Project Plan	24-11-2019	23-12-2019	29
Actual Progress	16-11-2019	29-11-2019	20

Since the official submission for documentation was on 6th December 2019, therefore the developer had only 20 days to complete the documentation. Since the diagram and testing phase were completed in time, it was easier to focus on writing the report in a short period of time. The product and process evaluation were done which explains the final objective that was stated in the proposal. Proposal can be referred in Appendix A. Finally, the report was submitted to the project supervisor, Dr Joshua to get feedback. Then, the changed were made accordingly.

7.3 Summary

In short, the product and process evaluation are discussed in this chapter. Due to the changes on submission date, only nine out of ten objectives were achieved. The user interface of the system could not be developed because of the time constraint. However, a new objective was created and done which was to measure the speed of the vehicle.

Chapter 8: Conclusion and Recommendation

8.1 Conclusion

This system was developed mainly to reduce accidents in Malaysia. According to Othman O.Khalifa (2008), the United Nations has categorised Malaysia as 30th in the ranking of highest number of fatal accidents with an average of 4.5 causalities per 10,000 vehicles. Furthermore, this system was also developed to help elderly people who are too old to drive safely. This is because as driver age increases, changes in flexibility, visual acuity, reaction time, memory and strength will affect the ability to drive (Lutin et al., 2013). After reviewing all these problems, End-to-End Deep Learning system for Self-Driving Vehicle was developed which enables the vehicle to detect the lanes, detect the speed of the vehicle, recognise the objects on the road and predict the steering angle of the vehicle.

The lanes are detected by creating a mask on the lanes and also by removing unnecessary noises by applying threshold on the road background. This will help the algorithm to recognise the lanes clearly. After removing all the noises, higher order polynomial is needed to incorporate to detect curved roads. For that, the algorithm must have an awareness of previous detection and a value of confidence. In order to fit a higher order polynomial, the image is sliced in many horizontal strips. On each strip, straight line algorithm was applied and pixels were identified corresponding to lane detections.

After detecting the lanes, the speed or the vehicle is measured. In this system, to detect the speed, centre white-lane markers are used. When a lane-marker is present, number of frames will be tracked until the next lane marker appears in the same window. Then, the number of frame rate of the video is taken which will be extracted from the video. From this, the speed of the vehicle will be calculated using distance / time formula where distance is the distance of the white-lane and time is the time taken of the vehicle to reach the end point of white-lane. Then, the speed will be showed in km/h.

Once speed detection is calculated, angle of the road is calculated. Polynomial curve fitting model is used to find the best-fit function (Curve) along a set of range. The average angle is calculated by using exponential moving average formula. From this, the average angle of each frame is saved into a .txt file to create a new dataset. Then, object detection is used to avoid the obstacle around the vehicle during autonomous driving. For this system, algorithm based on regressions which is the YOLO method is used. After the image is divided, classification and localisation of the object for each grid will be done. From this, the confidence score of each grid is determined.

Then, the model is trained using Tensorflow and Keras library for behavioural cloning. The video dataset is split into images frame by frame. Then pre-process is done. A pickle file is created to store all the images to train. After training the model,

compilation takes place. It takes three parameters which are optimizer, loss and metrics. The optimizer controls the learning rate. The model was train on 2937 samples and validated on 1259 samples with 30 epochs. Then the best model is saved as Autopilot.h5. To predict the steering angle, the trained model is predicted using a Keras model. After that, the model predicts the processed image which is the image that was fed into the training model. Next, a steering image is read from the local disk to visualise the steering angle difference on the output window. Finally, the video of the local road was set as the output to predict the steering angle which will navigate the vehicle autonomously.

In conclusion, AutoMove is an End-to-End system for self-driving vehicles that was developed to achieve the aim of the project in order to reduce accidents and help elderly people to drive safely. AutoMove followed the usual development process which includes planning, analysis, design, implementation and testing phases. Every objective was met during the development of this project. This system would be helpful for researchers and developer who is trying to implement self-driving vehicle without hardware components.

8.2 Recommendation

AutoMove can be improved in many aspects for future development by implementing more useful and interesting features. Deep learning is widely used in many industries nowadays to improve their performance. For example, Twitter is using deep learning to combat inappropriate and racist content on its platform. According to Marr (2017), Twitter blocked almost 300,000 terrorist accounts in the first six months by using their AI tools. Hence, it is proven that deep learning is the new trend in tech industry. Through AutoMove, the lanes can be detected easily for straight and curved roads without using any of the external algorithms such as Canny Detection and Hough Transform. However, the lanes detected are not able to tell the self-driving vehicle in which lane it should drive. Therefore, implementing an algorithm to give instruction for the self-driving vehicle to drive in the right lane is necessary so that the vehicle can reach the destination in correct time and also avoid accidents.

Furthermore, for the speed detection of vehicle, the current system uses the white-dashed lane marker to calculate the distance between starting point to end point of the line. After getting the distance, the time taken to reach the end point will be taken. With this, the speed of the vehicle is measured. However, the speed of the vehicle would not be detected if the lane does not have split white lines. Hence, by adding sensors, the speed of the vehicle can be measured accurately for all types of roads. This can control the speed limit of the vehicle while driving. According to Star Online (2018), Malaysian Transport Minister Anthony Loke stated that, there were 3.1 million unpaid AES summonses in Malaysia. By implementing this autonomous vehicle, the speed of the vehicle will be controlled. Apart from that, objects on the road are detected by pre-trained YOLO model which does not provide high accuracy. Hence, to improve the accuracy, the objects must be trained with objects from the roads.

Besides, for the behavioural cloning, only one dataset is used to train, therefore when different roads are inputted, the accuracy for the steering angle will go down. To improve this, large datasets with different weathers, lighting and roads can be trained to improve the steering angle prediction. Finally, sensors such LiDAR can be used to view 3D vision around the car. (Mehta, 2018)

References

2010 Motor Vehicle Crashes: Overview (2012) [ebook]. Washington, DC: NHTSA's National Center for Statistics and Analysis. Available from https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/811552 [accessed 4 March 2019].

Assidiq, A., Khalifa, O., Islam, M. and Khan, S. (2008) Real time lane detection for autonomous vehicles. *2008 International Conference on Computer and Communication Engineering*.

Boehm, B. (1988) A spiral model of software development and enhancement. *Computer*, 21(5) 61-72.

Bojarski, M., Del Testa, D. and Dworakowski, D. (2016) *End to End Learning for Self-Driving Cars*. Available from https://arxiv.org/abs/1604.07316 [accessed 10 March 2019].

Break the rules, pay the price (2018). Available from https://www.thestar.com.my/news/nation/2018/09/04/break-the-rules-pay-the-price-no-more-mercy-for-those-aes-offenders-from-now-on-says-loke [accessed 2 December 2019].

Croucher, M. (2011) Minimizing the Rosenbrock Function. Available from https://demonstrations.wolfram.com/MinimizingTheRosenbrockFunction/ [accessed 11 October 2019].

Curve Fitting and Interpolation (2019) [ebook]. Available from http://www.engineering.uco.edu/~aaitmoussa/Courses/ENGR3703/Chapter5/ch5.pdf [accessed 6 October 2019].

Desegur, L. (2017) A Lane Detection Approach for Self-Driving Vehicles. Available from https://medium.com/@ldesegur/a-lane-detection-approach-for-self-driving-vehicles-c5ae1679f7ee [accessed 11 October 2019].

García Cuenca, L., Sanchez-Soriano, J., Puertas, E., Fernandez Andrés, J. and Aliane, N. (2019) Machine Learning Techniques for Undertaking Roundabouts in Autonomous Driving. *Sensors*, 19(10) 2386.

Geethapriya, S., Duraimurugan, N. and Chokkalingam, S. (2019) Real-Time Object Detection with Yolo. *International Journal of Engineering and Advanced Technology (IJEAT)*, 8(38) 578-581. Available from https://www.ijeat.org/wp-content/uploads/papers/v8i3S/C11240283S19.pdf [accessed 13 November 2019].

Genn, S. (2014) Least Squares Regression Line: Ordinary and Partial. Available from https://www.statisticshowto.datasciencecentral.com/least-squares-regression-line/ [accessed 3 November 2019].

Home - Keras Documentation (2019). Available from https://keras.io/ [accessed 20 November 2019].

Huang, R., Pedoeem, J. and Chen, C. (2018) *YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers*. Available from https://arxiv.org/pdf/1811.05588.pdf [accessed 13 November 2019].

Jalled, F. (2017) *Face Recognition Machine Vision System Using Eigenfaces*. Available from https://arxiv.org/abs/1705.02782 [accessed 3 March 2019].

Krizhevsky, A., Sutskever, I. and Hinton, G. (2017) ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6) 84-90.

Laguna, R., Barrientos, R., Blázquez, L. and Miguel, L. (2014) Traffic sign recognition application based on image processing techniques. *IFAC Proceedings Volumes*, 47(3) 104-109.

Lai, Y., Wang, N., Yang, Y. and Lin, L. (2018) Traffic Signs Recognition and Classification based on Deep FeatureLearning. *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods*.

Lane detection for self driving vehicles (2018). Available from https://mc.ai/lane-detection-for-self-driving-vehicles/ [accessed 8 October 2019].

Lutin, J., L. Kornhauser, A. and Lerner-Lam, E. (2013) The Revolutionary Development of Self-Driving Vehicles and Implications for the Transportation Engineering Profession. *Ite Journal*. Available from https://www.researchgate.net/publication/292622907_The_Revolutionary_Development_of_Self-Driving_Vehicles_and_Implications_for_the_Transportation_Engineering_Profession [accessed 23 November 2019].

Majaski, C. (2019) Comparing Simple Moving Average and Exponential Moving Average. Available from https://www.investopedia.com/ask/answers/difference-between-simple-exponential-moving-average/ [accessed 9 November 2019].

Marr, B. (2019) *How Twitter Uses Big Data And Artificial Intelligence (AI)*. Available from https://www.bernardmarr.com/default.asp?contentID=1373 [accessed 26 November 2019].

Marsja, E. (2017) PyCharm vs Spyder: a quick comparison of two Python IDEs - Erik Marsja %. Available from https://www.marsja.se/pycharm-vs-spyder-comparing-ides/ [accessed 4 March 2019].

mcCormick, C. (2017) CarND Advanced Lane Lines. Available from https://github.com/colinmccormick/CarND-Advanced-Lane-Lines [accessed 12 October 2019].

Mehta, R. (2018) Sensors in Autonomous Vehicles. Available from https://medium.com/swlh/sensors-in-autonomous-vehicles-5c8929d460e2 [accessed 29 November 2019].

OpenCV: Miscellaneous Image Transformations (2019). Available from https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef70a9b1115576a147222a96556ebc1d948b372bcd7ac59 [accessed 4 October 2019].

Pant, A. (2019) Introduction to Linear Regression and Polynomial Regression. Available from https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb [accessed 1 November 2019].

Pavani, T. and Mohan, D. (2019) Number Plate Recognition by using open CV-Python. *irjet*, 6(3). Available from https://www.irjet.net/archives/V6/i3/IRJET-V6I31275.pdf [accessed 3 December 2019].

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016) *You Only Look Once: Unified, Real-Time Object Detection* 1-4.

Redmon, J. and Farhadi, A. (2016) *YOLO9000: Better, Faster, Stronger*. Available from https://pjreddie.com/media/files/papers/YOLO9000.pdf [accessed 11 November 2019].

Robot, B. (2015) *Converting from RGB to HSV*. Available from http://coecsl.ece.illinois.edu/ge423/spring05/group8/FinalProject/HSV_writeup.pdf [accessed 15 November 2019].

Self-driving cars: The next revolution (2012) [ebook]. KPMG. Available from https://staff.washington.edu/jbs/itrans/self_driving_cars%5b1%5d.pdf [accessed 16 March 2019].

Sermanet, P., Eigen, D., Zhang, X. and Mathieu, M. (2013) *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. Available from https://arxiv.org/abs/1312.6229 [accessed 7 November 2019].

tf.keras.Model | TensorFlow Core r2.0 (2019). Available from https://www.tensorflow.org/api_docs/python/tf/keras/Model [accessed 23 November 2019].

Trubia, S., Canale, A., Giuffrè, T. and Severino, A. (2017) *Automated Vehicle: a Review of Road Safety Implications as Driver of Change*.

Venketas, W. (2019) Exponential Moving Average (EMA) Defined and Explained. Available from https://www.dailyfx.com/forex/education/trading_tips/daily_trading_lesson/2019/07/29/exponential-moving-average.html [accessed 12 November 2019].

Vitelli, M. and Nayebi, A. (2016) *CARMA : A Deep Reinforcement Learning Approach to Autonomous Driving*. Available from https://www.semanticscholar.org/paper/CARMA-%3A-A-Deep-Reinforcement-Learning-Approach-to-Vitelli-Nayebi/b694e83a07535a21c1ee0920d47950b4800b08bc [accessed 16 November 2019].

Wagh, P., Thakare, R., Chaudhari, J. and Patil, S. (2015) Attendance system based on face recognition using eigen face and PCA algorithms. *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*.

Yadav, N. and Mody, R. (2017) *Predict Steering Angles in Self-Driving Cars*. Available from https://rmmody.github.io/pdf/682Project.pdf [accessed 15 November 2019].

Yusnita, R., Norbaya, F. and Basharuddin, N. (2012) Intelligent Parking Space Detection System Based on Image Processing. *International Journal of Innovation, Management and Technology*, 3(3) 232-235.

Appendix A – Project Proposal

a) Project Title

AutoMove (End-to-End Deep Learning system for Self-Driving Vehicles)

b) Background of the project

End users all around the globe are very keen about the rise of self-driving vehicles for public. Self-driving vehicles also known as a driverless vehicle or autonomous vehicle. A self-driving vehicle can work without human conduction and does not require any human intervention. According to World Health Organisation (2013), there were 1.25 million people were killed on roads in 2013. Besides, in the same year in the United States, 32,885 people were killed from motor vehicle crashes and more than 2.2 million were injured (2010 Motor Vehicle Crashes: Overview, 2012). A research was carried out by KPMG, Center for Automotive Research, where they predicted more than 30,000 lives can be saved from this crashless future (Self-driving cars: The next revolution, 2012). Self-driving vehicles likely can reduce all driver error and most auto crashes.

The background story on how the project idea was found and the major reason mobility for seniors' scope is the focus of this project because of a news from an international website called "CNN Business". According to Subodh Mhaisalkar, the professor in charge of Nanyang's Energy Research Institute said that Singapore is aging quickly and [at] a faster rate than anywhere else is in the world. Therefore, in order to help the seniors, it would be beneficial to have such technology in the market (Jamshed, 2019). By 2030, Singapore's population size is expected to reach 6.34 million according on projections from the United Nations (UN) released in 2017 (Ming En, 2017). By that time, there will be 1.8 million people who are aged 65 years or older and 806,000 people under the age of 15. By then, the numbers will stretch to 3.08 million and 722,000, respectively, out of a total population of 6.58 million by 2050. Based on this figures, about half (47 per cent) of Singapore's population will be at least 65 years old in three decades. After discovering this fact, researchers started to find more details about this (Ming En, 2017).

Firstly, the problem found in the scope was elderly people are too old to drive safely. This is because as driver age increases, changes in flexibility, visual acuity, reaction time, memory and strength will affect the ability to drive (Lutin et al., 2013). According to Rosenbloom (2014), "people who is more than 75 have higher motor vehicle deaths per 1000 compared to any other cohort of the population except those under age 25. When older drivers become conscious of their decrease capacity to drive, they voluntarily give up their licenses. Some self-limit their driving to avoid heavy traffic or to daylight hours only. AutoMove will act as a cure to this problem by providing end-to-end self-driving vehicle system. Based on an online article research from MIT Autonomous Vehicle Technology Study, the 2007 DARPA Urban Challenge

was a milestone achievement in robotics, when six of the eleven self-driving vehicles in the finals successfully reach the finish line of the navigation in an urban environment (Fridman, 2018). The first place finisher travelled at an average speed of 15 mph. Many declared the fully autonomous driving task a "solved problem" because of the success of the competition. Even today, over 12 years later, the problems of mapping, localization, vehicle control, and scene perceptions with self-driving vehicle development still go on with open challenges which are not be fully solved by systems incorporated into a production platform, for example offered a sale for even a limited operational space. A human supervisor is responsible for controlling during situations where the AI system is not sure of not able to safely proceeds remains the norm when testing of prototype vehicles (Fridman, 2018). The assumption from the MIT Autonomous Vehicle Technology (MIT-AVT) study is that the DARPA Urban Challenge was only a beginning down a long process towards developing self-driving vehicle systems. According to the authors, they believe that the current challenge faced by the world is one that has the human being as an important part of every aspect of the system (Fridman, 2018).

Moreover, a research journal on End to End Learning for Self-Driving Cars by NVIDIA Corporation (2016) were investigated for their approach to implement self-driving car using a convolutional neural network (CNN). Raw pixels from a single front-facing camera was mapped directly to steering commands. According to the authors, this end-too-end approach proved powerful. The system learns to drive in traffic with less training data on local roads with or without lane markings. It also worked in places with blurry visual guidance such on impervious surface roads and in parking lots. The system automatically learns internal model of the needed processing steps like as detecting suitable road signs with only the human steering angle as the training signal (Bojarski et al., 2016, 3-6).

In this project, Automove will solve some of the problems in autonomous vehicle systems such as detecting road boundaries and lanes using vision system on the vehicle, detecting Malaysian traffic signs / signboards and developing a vision-based navigation system for self-driving vehicles

c) Proposed Work

The proposed work to be produced at the end of the project is Automove End-to-End Deep Learning for Self-Driving Vehicles using python programming. The process starts from identifying the main purpose of this project. After identifying the purpose, it is easier to stay on the same track and avoid running out from the proposed system scope. The next process phase is to identify the technical part of the project. There are four important decisions that needed to be finalised to keep the flow of implementation which are:

1. Detecting road boundaries and lanes using vision system on the vehicle

I. Edges detection. Canny edges' detector will be used to determine frames' edges, and essentially the road boundaries. The frame's edges will be classified to hyperboles or to straight lines according to the relative vehicle's position and direction — to predict the future road lanes parameters and to the road geometry (Assidiq et al., 2008).

II. Hough transform and line detection. To detect lines, standard Hough transform (HT) will be used with a restricted search space. The main advantage of HT is its robustness to noise and occlusions that means the algorithm is still valid to estimate noisy and dashed edges (Bounini et al., 2015). It will transform a set of frame pixels, points, in the Cartesian space to another space known as Hough space over some parameter space.

III. Lane boundary detection. The lane boundary scan phase uses the edge image the Hough lines and the horizon line as input. The edge image is what is scanned and the edges are the data points it collects. The scan begins where the projected Hough lines intersect the image border at the bottom of the image (Bounini et al., 2015). Once that intersection is found, it is considered the starting point for the left or right search, depending upon which intersection is at hand. From the starting point, the search begins a certain number of pixels towards the center of the lane and then proceeds to look for the first edge pixel until reaching a specified number of pixels after the maximum range. The range will be set to the location of the previously located edge pixel plus a buffer number of pixels further (Bounini et al., 2015).

2. Detecting Malaysian traffic signs / signboards using edge detection and image recognition

I. Deep Convolutional Neural Network. Traffic sign images will be trained by LeNet-5 convolutional neural network architecture.

II. Edge Detection using Zhang’s Method. Zhang’s method of edge detection follows the principle of linear prediction, a filtering method that used to predict the future values of a signal using the past and the present values. The main idea behind this linear prediction method is to optimize filter coefficients in order to minimize the prediction errors (Jung et al., 2015).

III. Hough Transform Algorithm. In the detection phase, light-weight color-based segmentation algorithm and Hough transform algorithm are applied to extract candidate regions of traffic signs (Vishwanathan et al., 2017).

IV. Image Processing. Captured image of traffic sign will be converted to a series of image frames based on the number of seconds using OpenCV. An image is converted to a matrix and then various operation is performed to get desired results and values, for instance the captured image is converted to a grey scale from RGB color after filtering and based on the coding the eight edges (octagon shape) of a STOP sign are located and the edges of the STOP sign are mapped and sent to the microprocessor to stop the vehicle (Jung et al., 2015).

V. Tensorflow. It is a method which can be used to obtain image recognition. This approach need more implementation time as it is very complex. The process of image recognition begin from training a model to recognise objects which are the sign boards of Malaysian roads. The model is trained in a computer because it requires high processing power. Once high accuracy is reached, the application will run to perform the prediction of the traffic signs.

3. Developing a vision-based navigation system for self-driving vehicles

I. There are four steps to develop this. In the first step an image is obtained and the road is identified using ANNs classification (R. Souza et al., 2017).

II. A template matching algorithm is used to identify the geometry of the road.

III. Third step is to filter noisy inputs and any classification error.

IV. Finally, a template memory is used in order to define the action that the vehicle should take to keep on road (R. Souza et al., 2017).

V. The basic network structure used is a feedforward MLP, the activation function of the hidden neurons is the sigmoid function and the ANN learning is the resilient backpropagation (RPROP). The inputs are represented by templates memory and the outputs are the steer angle and speed (R. Souza et al., 2017).

- Methodology

  Software development life cycle method is an important part of the whole process. Without the proper selection of methodology, the time taken to do the project might be longer than expected, therefore a right methodology should be chosen.

i. Spiral methodology. This model contains four phases, which starts from planning that includes understanding of the system requirements. The next phase is identifying possible risks and resolving it. Then, development and finally is testing to the planning of next phase (Boehm, 1988). When a spiral cycle occurs each time, objectives that are finalised will be taken to next phase for checking risks and resolving phase (Boehm, 1988). After this phase is completed, the prototype will be developed and tested for any problems. Lastly, next cycle preparation will be conducted. According to Boehm (1988), in spiral model, the model will provide a more precised view for each cycle. It will help the developer in terms of unknown risks, the probability of a function being produced on time and others (Boehm, 1988).

The only reason spiral methodology was chosen is because in this whole application development, there are times where the system requirements need to be refined, adding and removing of requirements. Such ways are more suited to be done in spiral methodology because there will not be any negative domino impact to other phases of the model.

With the proposed work is done, the next question will be whether there is any existing systems available for this project. By researching and discovering how to do the project, it also helps to understand the difficulties in this proposed system and the estimation time to complete. A Gantt chart draft was drawn and according to that the proposed system can be completed in 320 hours.

d) Aim of Project

The main Aim of Automove is:

1. To develop an End-to-End Deep Learning for Self-Driving Vehicles to detect road boundaries and lanes using vision system on the vehicle, detect Malaysian traffic signs / signboards and developing a vision-based navigation system.

e) Objectives

Objectives are very important for every project or system. The proposed system will have minimal chance of going out of the track with objectives.

Computational Objectives

1. To research on Convolutional Neural Networks (CNN) for detecting road boundaries and lanes using vision system on the vehicle.
2. To research on machine learning framework TensorFlow to achieve image recognition.
3. To detect Malaysian traffic signs / signboards such as stop sign, right and left signs using image recognition
4. To develop a vision-based navigation system for self-driving vehicles
5. To do research in Keras, to build complex neural network.
6. To design the user interface of the system.
7. To test the proposed system using black box and white box testing methodology after completion before submission.
8. To take dataset of local roads for the system inputs.
9. To research about the strength and weakness of the system.
10. To complete the documentation for the main report.

f) Skills

- Familiar skills and knowledge

Skills	How it was acquired
Artificial Intelligence	The skill was acquired from the subject "Introduction to Artificial Intelligence". This course taught me the basics and understanding of artificial intelligence such as methodology, how it works and others. With these fundamentals, it is possible to implement the AI related features for this system. Furthermore, there was another subject "Intelligent Systems" which also taught about supervised, unsupervised and reinforcement learning which is the core understanding to implement this system. It also taught me how machine learning and neural network works. Cleaning techniques acquired from this subject was also helpful in this project.
Security	This is an autonomous application, so security is very important In this project. Hence, there was a subject in diploma which taught about the ways to protect the system from threats, encryption algorithms like Caesar cipher which helped to protect user's data from being accessed by others.
	In any project, referencing and in citation is very important. Without proper referencing,

Referencing	the documentation for the proposed system will be not good. The skills acquired for referencing was from the subject called "Writing and referencing" where all the proper references using Harvard Referencing style were taught. With this skill, the documentation will be supported with proper references and in citations.

- Skills or knowledge for research

Skill	How will it be acquired
Image Recognition / Processing using Tensorflow	This set of skills will be taught in the final semester under "Image Processing" subject. In that subject, methods for retrieving information from an image file will be taught. From this module, it is possible to produce a proper image recognition to identify sign boards or traffic signs on the roads.

g) Sources of information / bibliography

2010 Motor Vehicle Crashes: Overview (2012) [ebook]. Washington, DC: NHTSA's National Center for Statistics and Analysis. Available from https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/811552 [accessed 4 March 2019].

Assidiq, A., Khalifa, O., Islam, M. and Khan, S. (2008) Real time lane detection for autonomous vehicles. *2008 International Conference on Computer and Communication Engineering* 2-5. Available from https://www.researchgate.net/publication/4356226_Real_time_lane_detection_for_autonomous_vehicles [accessed 16 March 2019].

Boehm, B. (1988) A spiral model of software development and enhancement. *Computer*, 21(5) 61-72.

Bojarski, M., Del Testa, D., Dworakowski, D. and Firner, B. (2016) End to End Learning for Self-Driving Cars. *NVIDIA Corporation* 3-6. Available from https://arxiv.org/abs/1604.07316 [accessed 8 March 2019].

Bounini, F., Gingras, D., Lapointe, V. and Pollart, H. (2015) Autonomous Vehicle and Real Time Road Lanes Detection and Tracking. *2015 IEEE Vehicle Power and Propulsion Conference (VPPC)* 1-3. Available from

https://www.researchgate.net/publication/308852371_Autonomous_Vehicle_and_Real_Time_Road_Lanes_Detection_and_Tracking [accessed 16 March 2019].

Fridman, L. (2018) MIT Autonomous Vehicle Technology Study. *Massachusetts Institute of Technology (MIT).* Available from https://arxiv.org/pdf/1711.06976.pdf [accessed 7 March 2019].

Jamshed, Z. (2019) Singapore wants self-driving cars to help its aging society. *CNN*. Available from https://edition.cnn.com/2019/02/25/tech/self-driving-cars-singapore/index.html [accessed 5 March 2019].

Jung, S., Lee, U., Jung, J. and Shim, D. (2016) Real-time Traffic Sign Recognition system with deep convolutional neural network. *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI).*

Lutin, J., L. Kornhauser, A. and Lerner-Lam, E. (2013) The Revolutionary Development of SelfDriving Vehicles and Implications for the Transportation Engineering Profession. *Ite Journal* [accessed 6 March 2019].

Ming En, S. (2017) Elderly to make up almost half of S'pore population by 2050: United Nations. *Today Online*. Available from https://www.todayonline.com/singapore/elderly-make-almost-half-spore-population-2050-united-nations [accessed 6 March 2019].

Number of road traffic deaths (2015). World Health Organization. Available from https://www.who.int/gho/road_safety/mortality/traffic_deaths_number/en/ [accessed 16 March 2019].

Rosenbloom, S. (2004) *Mobility of the elderly: Good news and bad news.*

R. Souza, J., Pessin, G., Y. Shinzato, P., S. Osorio, F. and F. Wolf, D. (2014) *Vision-based Autonomous Navigation Using Neural Networks and Templates in Urban Environments* 1-3. Available from https://www.researchgate.net/publication/228399154_Vision-based_Autonomous_Navigation_Using_Neural_Networks_and_Templates_in_Urban_Environments [accessed 16 March 2019].

Self-driving cars: The next revolution (2012) [ebook]. KPMG. Available from https://staff.washington.edu/jbs/itrans/self_driving_cars%5b1%5d.pdf [accessed 16 March 2019].

Vishwanathan, H., Peters, D. and Zhang, J. (2017) Traffic Sign Recognition in Autonomous Vehicles Using Edge Detection. *Volume 1: Aerospace Applications.*

h) Resources - statement of hardware / software required

1. Hardware

Resources	Specification
Laptop	Minimum 4GB ram to run the IDE
Internet Connection	Minimum of 4mbps speed to receive data from online

2. Software

Resources	Specification
Google Colab	Able to train the model from datasets
Tensorflow	Able to assist in training models to do image, lane and pattern recognition
Python IDE	Able to build the application

i) Structure and contents of project report

Report Structure

1. Title Page
2. Authorship Declaration
3. Acknowledgments
4. Abstract
5. List of Contents
6. Introduction
7. Analysis
8. Synthesis
9. Evaluation and Conclusions
10. References
11. Bibliography
12. Appendices

a. Terms of Reference

b. Test Case

c. User Manual (Basic)

j) Marking Scheme

1. Project Type

- Software Engineering Project

2. Project Report

- Introduction
  - Brief description of the problem.
  - Background of the Study.
  - Brief description about the proposed system.
- Analysis
  - Literature Review
    - Description of the Existing System
    - Proposed System
    - Methodology
    - Chosen Methodology
    - Conceptual Model
- Synthesis
  - System Design and Implementation
  - Design
    - User Interface
    - Database
    - System Design
  - Implementation
  - Testing
- Evaluation, Conclusions
- Future Recommendations

3. Product

1. Fitness for Purpose (40%)
   i. Meeting of requirements as identified during the project (15%)
   ii. Quality of Functionality (15%)
   iii. Human Computer Interaction (10%)

2. Build Quality (60%)
   i. Requirement specification and analysis (15%)
   ii. Design specification (15%)

iii. Code quality (15%)
iv. Test plan and results (15%)

3. Deliverables

i. An Neural Network Based Application
ii. Complete Report
iii. CD-ROM with Source Code and Report Softcopy
iv. Human Computer Interaction (HSI)
v. Test Cases for Main Features
vi. Simple User Manual

k) Project Plan - Schedule of activities

Semester 1

Task	Start Date	End Date
Initiation Phase	15-Jan-19	30-Jan-19
Select project title	15-Jan-19	18-Jan-19
Search supporting research pape	18-Jan-19	24-Jan-19
Prepare for Project Initiation Doc	24-Jan-19	27-Jan-19
Submit PID	28-Jan-19	28-Jan-19
Receive PID Review	29-Jan-19	30-Jan-19
Planning Phase	31-Jan-19	8-Mar-19
Write Project Proposal Draft	31-Jan-19	20-Feb-19
Project Proposal review with se	21-Feb-19	28-Feb-19
Make changes	1-Mar-19	4-Mar-19
Submit project proposal	5-Mar-19	6-Mar-19
Analyse Phase	7-Mar-19	20-Apr-19
Research on Tensorflow	7-Mar-19	12-Mar-19
Research on CNN and ANN	13-Mar-19	20-Mar-19
Research on Tensorflow serving	21-Mar-19	27-Mar-19
Write Literature Review	7-Mar-19	9-Apr-19
Submit Analysis Chapter	10-Apr-19	17-Apr-19
Design Phase	18-Apr-19	10-May-19
Design UML Diagram	18-Apr-19	26-Apr-19
Design user interface	27-Apr-19	5-May-19
Design database architecture	6-May-19	10-May-19

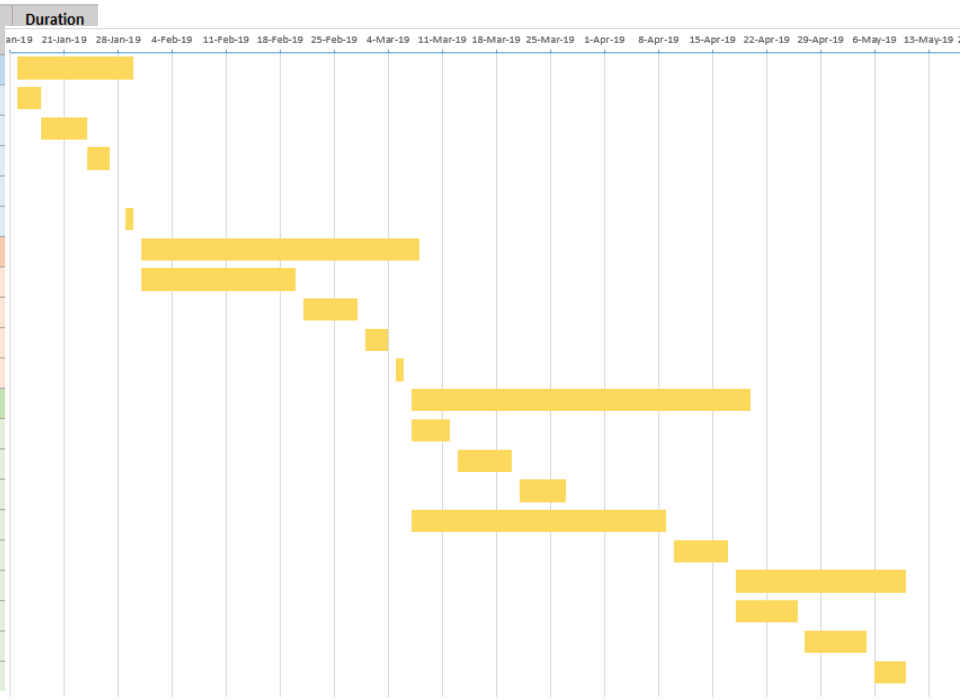


Figure 1: Semester 1 Gantt Chart

Semester 2

Task	Start Date	End Date	Duration (days)
Implementation phase	9-Sep-19	1-Nov-19	
Collecting data	9-Sep-19	18-Sep-19	
Preprocess trafic signs dataset	19-Sep-19	20-Sep-19	
Train CNN model and paramete	21-Sep-19	28-Sep-19	
Testing road signs	29-Sep-19	1-Oct-19	
Preprocess road dataset	2-Oct-19	4-Oct-19	
Train CNN model and paramete	5-Oct-19	15-Oct-19	
Build model	16-Oct-19	19-Oct-19	
Data Augmentation	20-Oct-19	23-Oct-19	
Fit Generator	24-Oct-19	25-Oct-19	
Refine details	26-Oct-19	1-Nov-19	
Testing phase	2-Nov-19	23-Nov-19	
Black box testing	2-Nov-19	8-Nov-19	
Final Product testing	9-Nov-19	13-Nov-19	
Writing test case document	14-Nov-19	16-Nov-19	
Bug fixing	17-Nov-19	23-Nov-19	
Report documentation	24-Nov-19	23-Dec-19	
Introduction	24-Nov-19	27-Nov-19	
Chapter 3: Synthesis	28-Nov-19	8-Dec-19	
Product Evaluation	8-Dec-19	13-Dec-19	
Process Evaluation	14-Dec-19	17-Dec-19	
Conclusion and Recommendati	18-Dec-19	20-Dec-19	
Report Printing	21-Dec-19	23-Dec-19	

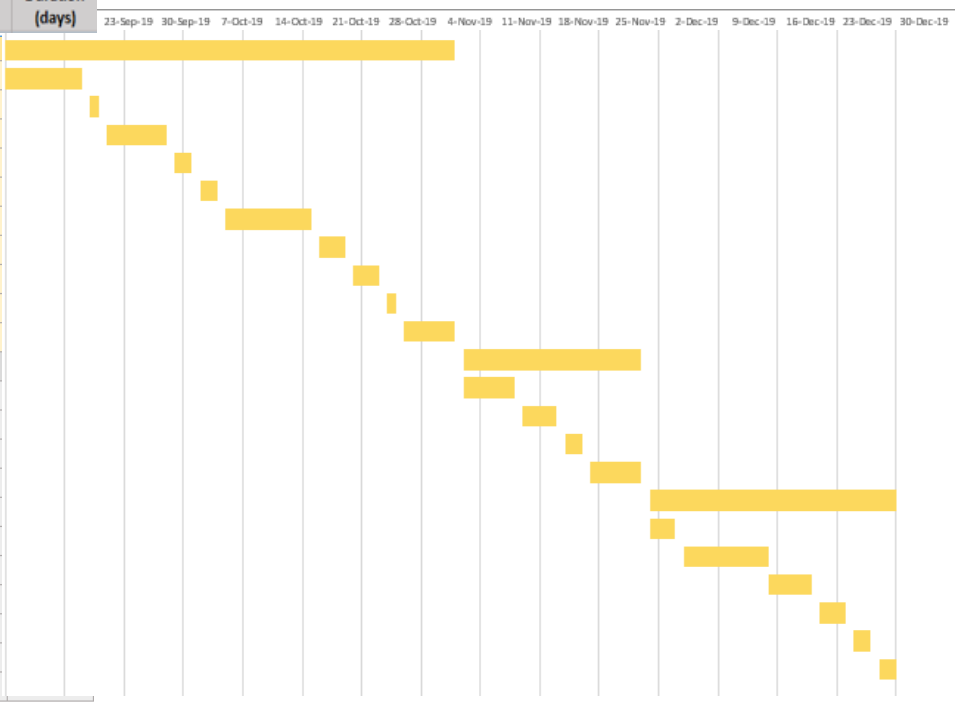


Figure 2: Semester 2 Gantt Chart

Appendix B – Software Requirement Specifications

1.0 Introduction

This chapter will brief through about the software requirement specifications of this final year project. There are sub chapters which will be discussed such as purpose, scope, definitions, abbreviations, acronyms and the overview of the product.

1.1 Purpose

This chapter is written to show the purpose of this end-to-end deep learning system for self-driving vehicles with detailed description. An outline of the application will be provided along with the functional and non-functional requirements. This document will be the key for developer during the development process of this application.

1.2 Scope

The main functionality of this self-driving vehicle application is to act as an end-to-end deep learning system which predicts the steering angle of the vehicle. At first, the system will detect the lanes of the vehicle. Then, it will measure the speed of the vehicle to maintain the speed limit. After that, the angle of the road will be calculated by getting the difference of top and bottom angle of the road for each frame. From this, the angles are saved into .txt file. Then, the objects on the road will be detected to avoid obstacles. Finally, behavioural cloning will be done by predicting the steering angle of the vehicle using convolutional neural network.

1.3 Definitions, Acronyms and Abbreviations

Term	Definition
PID	Project Initiation Document
UML	Unified Modified Language
SRS	Software Requirement Specification
IDE	Integrated Development Environment
CNN	Convolutional Neural Network
Tensorflow	TF
MSE	Mean Square Error
YOLO	You Only Look Once
RNN	Recurrent Neural Network

1.4 Reference

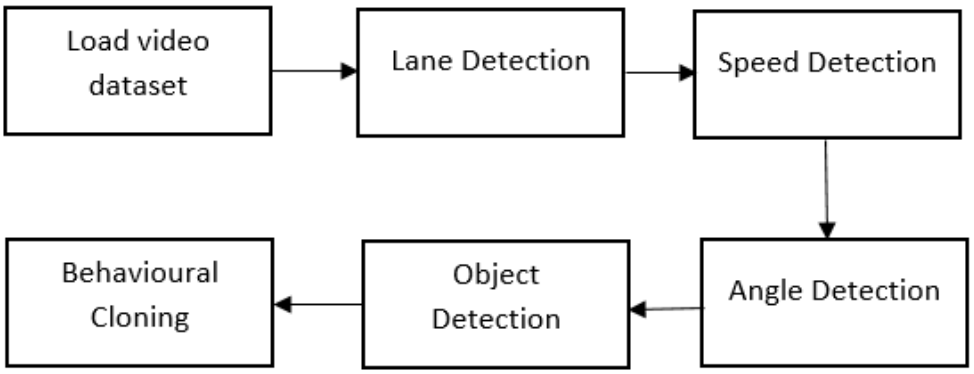
IEEE, 2011. *ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes --Requirements engineering.* 1 ed. New York: IEEE.

1.5 Overview

In this section, overall description of the application will be discussed. It also provides an overview on overall features of the application and system requirements. Besides, constraints of the software will be discussed too.

2.0 Product Perspective

This product focused on developing an end-to-end deep learning system for self-driving vehicle on PyCharm IDE using Python programming language and Tensorflow with Keras to train the neural network.



The system will be able to detect the lanes of the vehicle. Then, the speed of the vehicle will be calculated using the distance of the white-line markers. After that, angle of the road will be measured for each frame. Then, the objects on the road will be recognised. Finally, the steering angle of the vehicle will be predicted using behavioural cloning.

2.1 Product Functionality

The end-to-end deep learning system for self-driving vehicle assist the users to autonomously drive their vehicle, therefore users do not have to drive their vehicle. Furthermore, with autonomous vehicle, users will be able to reduce fuel consumption. Next, autonomous vehicles can also reduce accidents on the roads as there will be less human errors.

2.2 User Characteristics

The targeted users for this system are drivers, old aged people and also people with disability. This system will help them to travel around easily as they do not have to depend on others.

2.3 Constraints

As this system only focused on software components, there are few constraints. Firstly, there is only one dataset which was trained, therefore the steering angle accuracy will be less on other roads. Next, to run this system, high processing power PC is required as it contains real time processing.

2.4 Dependencies

This application has three dependencies. One of the dependencies is, in order to run the system, minimum of 8GB RAM and NVIDIA Graphics card are required. Next, Anaconda GPU version need to be installed to run this application. Furthermore, libraries such as OpenCV, Tensorflow and Keras are required to train the model and also for video processing.

2.5 Recommendations

One of the future enhancements for this system is to implement an algorithm to give instruction for the self-driving vehicle to drive in the right lane. Furthermore, for the speed detection of vehicle, the current system uses the white-dashed lane marker to calculate the distance between starting point to end point of the line. Hence, by adding sensors, the speed of the vehicle can be measured accurately for all types of roads.

3.0 Specific Requirements

In this section, functionalities of this end-to-end deep learning for self-driving vehicles will be explained.

3.1 External Interface Requirements

This section describes the outline of the inputs and outputs from this application which includes software interface and hardware interface of the application.

3.1.1 Software Interface

The software that was used to implement this application is PyCharm IDE. The implementation of the software can also be done using Visual Studio Code or Python Idle. The neural network was trained using Tensorflow and Keras library while for the video processing, OpenCV is used.

3.1.3 Hardware Interface

The hardware which is required to run this application is a laptop with GPU version graphics card.

3.2 Functional Requirements

3.2.1 Lane Detection

The first functional requirement is lane detection. This accepts road data and uses OpenCV image processing technique to identify the lanes. A blue mask is attached on the lane to see the difference between the lanes and the other images on the road.

3.2.2 Speed Detection

The second functional requirement is detecting the speed of vehicle. The speed of the vehicle will be calculated using distance / time formula where distance is the distance of the white-lane and time is the time taken of the vehicle to reach the end point of white-lane. The speed will be showed in km/h.

3.2.3 Angle Detection

Detecting the angle of the road is crucial to predict the vehicle's steering angle. Polynomial curve fitting model is used to find the best-fit function (Curve) along a set of range. The average angle is calculated by using exponential moving average formula. From this, the average angle of each frame is saved into a .txt file to create a new dataset.

3.2.4 Object Detection

Object detection is used to avoid the obstacles around the vehicle during autonomous driving. For this system, algorithm based on regressions which is the YOLO method is used.

3.2.5 Behavioural Cloning

The next functional requirement is behavioural cloning. In this part, the model will be trained using Tensorflow and Keras library. The video dataset is split into images frame by frame. The model is trained on 2937 samples and validated on 1259 samples with 30 epochs. Then the best model is saved as Autopilot.h5.

3.2.6 Autonomous Navigation

To predict the steering angle, the trained model (Autopilot.h5) is loaded. The model is predicted using a Keras model. After that the model predicts the processed image which is the image that was fed into the training model. Next,

a steering image is read from the local disk to visualise the steering angle difference on the output window. Finally, the video of the local road were set as the output to predict the steering angle which will show the difference between actual steering angle and predicted steering angle.

3.3 Performance Requirements

The minimum requirement to run this application is to have a GPU powered laptop with 8GB RAM and NVIDIA Graphics card.

3.4 Design Constraints

The design of this application is to predict steering angle of a straight and curved road which was trained using neural network. However, in order to view the autonomous navigation, the proper dataset must be inputted.

3.5 Software System Attributes

In this section, the attributes that are available on the application will be discussed.

3.5.1 Security

Only the developer of this system will be able to use the functionalities of this system.

3.5.2 Compatibility

In order to run this system, a minimum of GPU powered laptop with 8GB RAM, NVIDIA Graphics card and Tensorflow 1.5 is required.

3.5.3 Usability

There are no login or sign out process in AutoMove which will help the users directly to use the system. Furthermore, there is no complex interface to run this system as user can just run the output from PyCharm IDE.

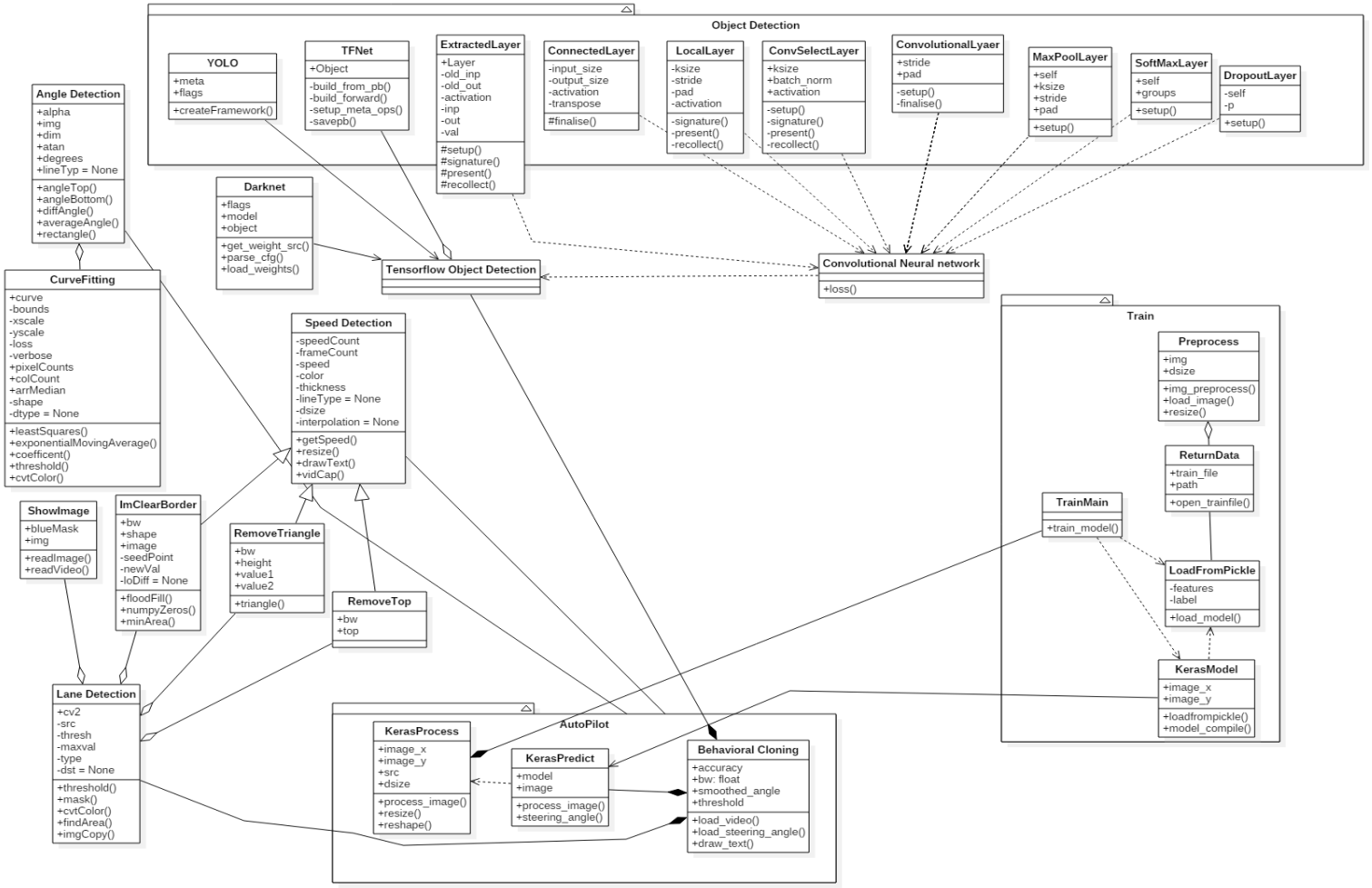
3.5.4 Availability

The system should be available at all times without any errors or bugs as most of the function in the system runs in real-time. The only necessity for users to use this system is a computer with high processing power.

3.5.5 Portability

The system can be run in any laptop with high graphics card.

Appendix C – Class Diagram (2nd Level)

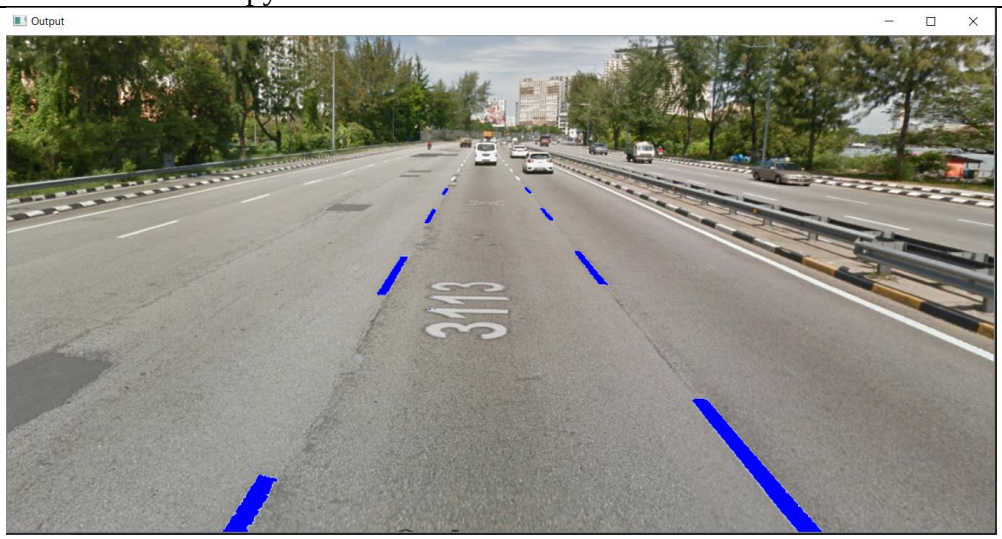


Appendix D – Test Plans

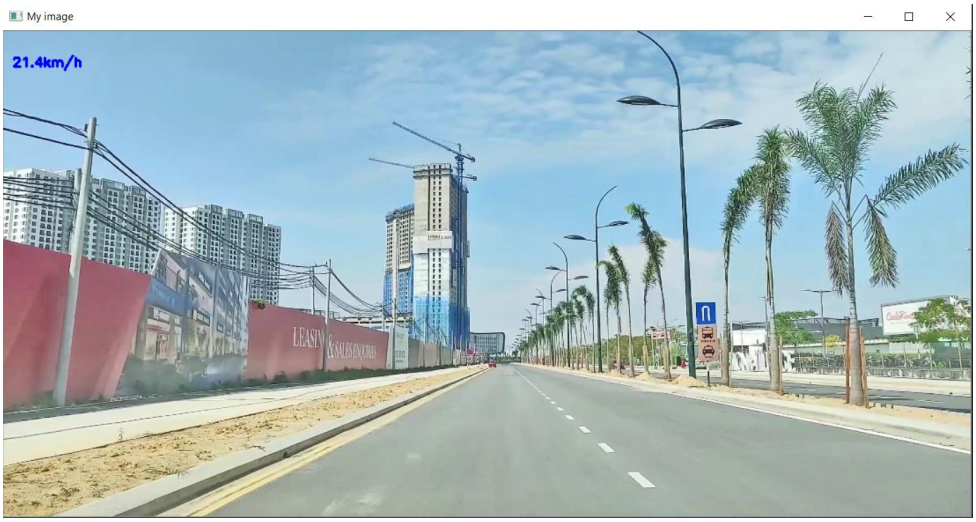
i. Black Box Testing

The test plans and test cases are as below:

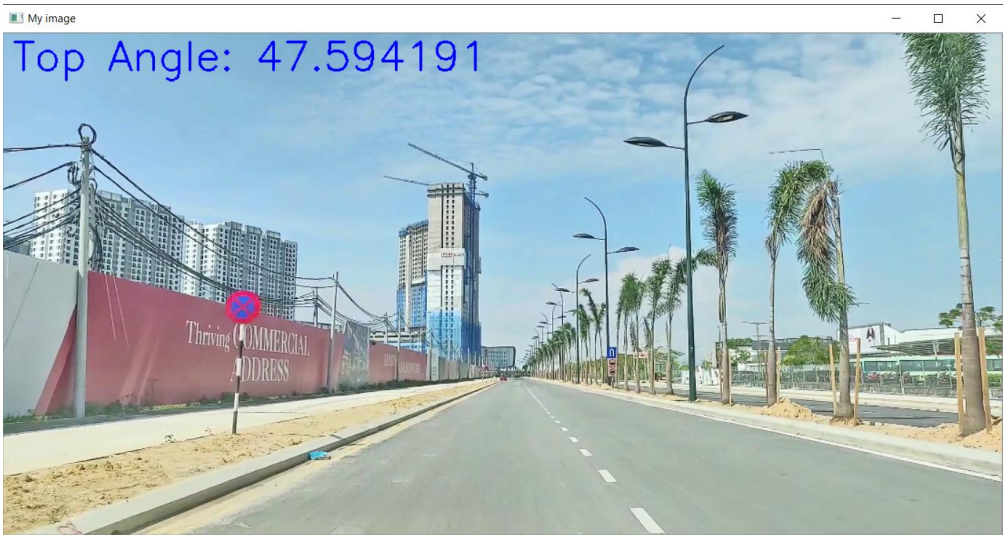
Test Case Number	**1**
Test Case Description	Display the lanes on a straight road captured in Penang Highway
Input Data	Road.png
Pre-condition	-
Expected output	The output will detect the lanes by adding a mask on the lanes
Procedures	Run lanedetection.py
Screenshot of actual result	
Post-condition	The output will detect the lanes
Test result	**Passed**

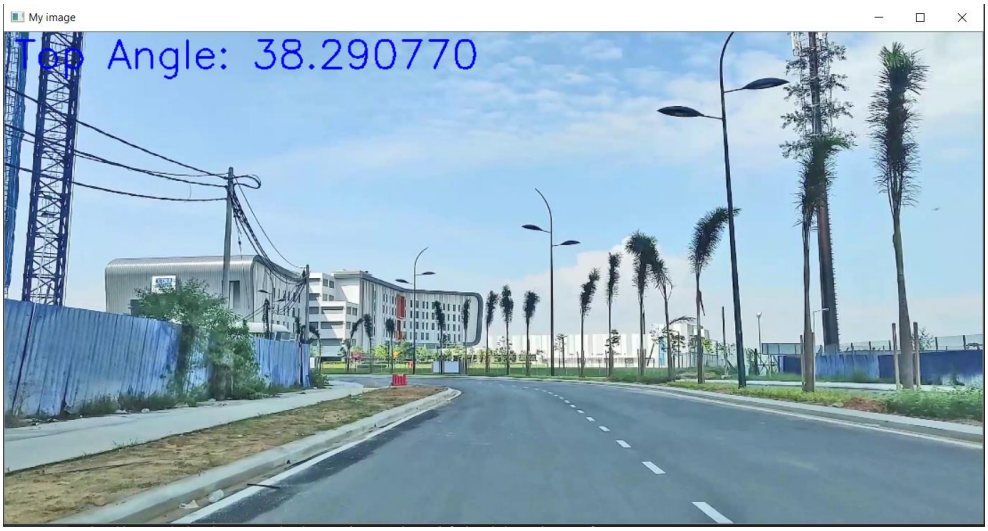
Test Case Number	**2**
Test Case Description	Display the lanes on a straight road captured in Lebuhraya Tun Dr Lim Chong Eu
Input Data	bridge.PNG
Pre-condition	-
Expected output	The output will detect the lanes by adding a mask on the lanes
Procedures	Run lanedetection.py
Screenshot of actual result	
Post-condition	The output will detect the lanes
Test result	**Passed**

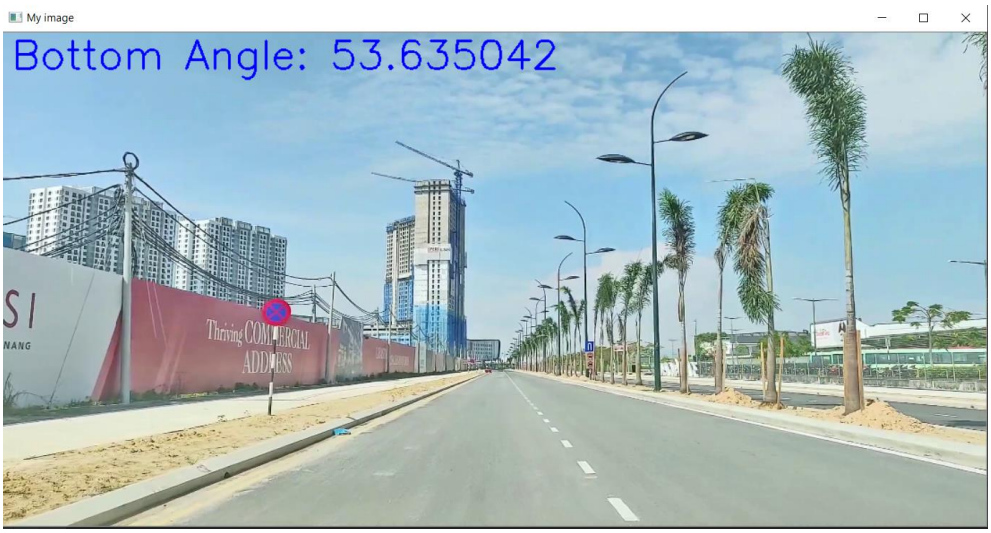
Test Case Number	**3**
Test Case Description	Display the lanes on a curved road captured in front of UOW KDU, Batu Kawan
Input Data	2226.PNG
Pre-condition	-
Expected output	The output will detect the lanes by adding a mask on the lanes
Procedures	Run lanedetection.py
Screenshot of actual result	
Post-condition	The output will detect the lanes
Test result	**Passed**

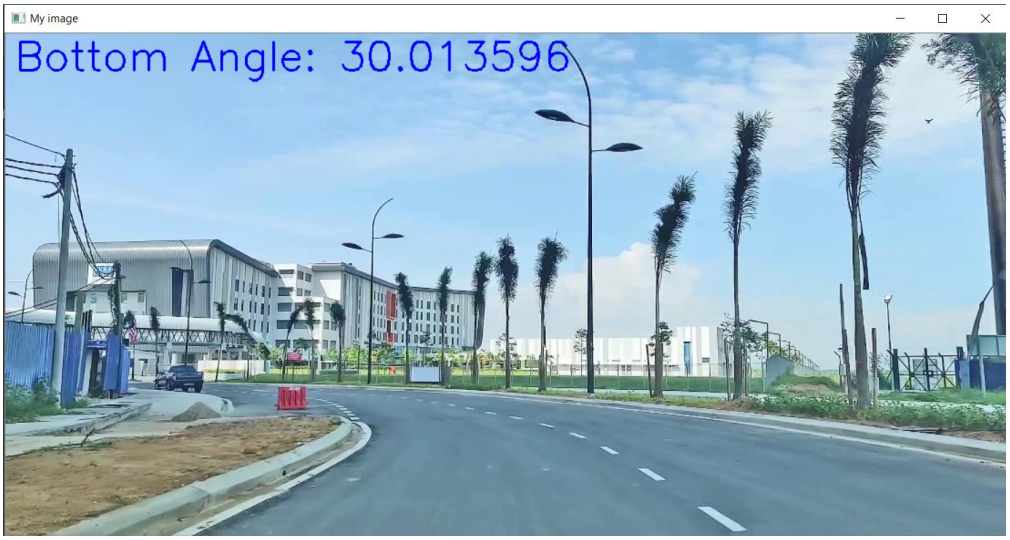
Test Case Number	**4**
Test Case Description	Display the speed of the vehicle on a straight road
Input Data	30fps.mp4
Pre-condition	-
Expected output	The output will show the speed of the vehicle in km/h on left top corner of the output window
Procedures	road_detection_speed.py
Screenshot of actual result	
Post-condition	The output will show the speed of the vehicle in km/h
Test result	**Passed**

Test Case Number	**5**
Test Case Description	Display the speed of the vehicle on a curved road
Input Data	30fps.mp4
Pre-condition	-
Expected output	The output will show the speed of the vehicle in km/h on left top corner of the output window
Procedures	road_detection_speed.py
Screenshot of actual result	
Post-condition	The output will show the speed of the vehicle on a curved road in km/h
Test result	**Passed**

Test Case Number	**6**
Test Case Description	Display the top angle of the road while driving on the straight road
Input Data	30fps.mp4
Pre-condition	-
Expected output	The output will show the top angle of the road
Procedures	road_detection_speed.py
Screenshot of actual result	
Post-condition	The output will show the top angle of the road in degrees
Test result	**Passed**

Test Case Number	**7**
Test Case Description	Display the top angle of the road while driving on the curved road
Input Data	30fps.mp4
Pre-condition	-
Expected output	The output will show the top angle of the road
Procedures	road_detection_speed.py
Screenshot of actual result	
Post-condition	The output will show the top angle of the road in degrees
Test result	**Passed**

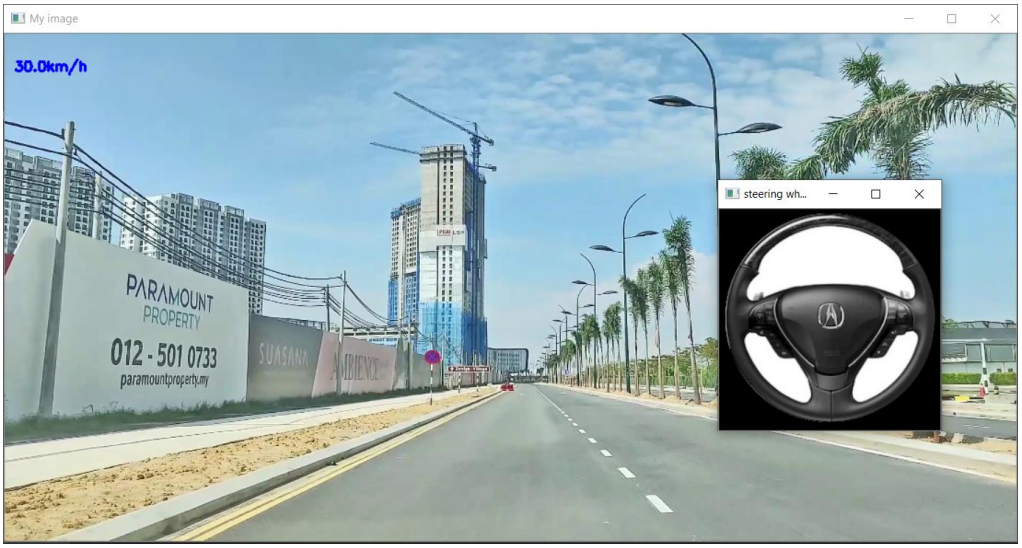
Test Case Number	**8**
Test Case Description	Display the bottom angle of the road while driving on the straight road
Input Data	30fps.mp4
Pre-condition	-
Expected output	The output will show the bottom angle of the road
Procedures	road_detection_speed.py
Screenshot of actual result	
Post-condition	The output will show the bottom angle of the road in degrees
Test result	**Passed**

Test Case Number	**9**
Test Case Description	Display the bottom angle of the road while driving on the curved road
Input Data	30fps.mp4
Pre-condition	-
Expected output	The output will show the bottom angle of the road
Procedures	road_detection_speed.py
Screenshot of actual result	
Post-condition	The output will show the bottom angle of the road in degrees
Test result	**Passed**

Test Case Number	**10**
Test Case Description	Display the difference and average angle of the road while driving on the straight road
Input Data	30fps.mp4
Pre-condition	Must consist of top and bottom angle of the road
Expected output	The output will show the difference and average angle of the road
Procedures	road_detection_speed.py
Screenshot of actual result	
Post-condition	The output will show the difference and average angle of the road in degrees
Test result	**Passed**

Test Case Number	**11**
Test Case Description	Display the difference and average angle of the road while driving on the curved road
Input Data	30fps.mp4
Pre-condition	Must consist of top and bottom angle of the road
Expected output	The output will show the difference and average angle of the road
Procedures	road_detection_speed.py
Screenshot of actual result	
Post-condition	The output will show the difference and average angle of the road in degrees
Test result	**Passed**

Test Case Number	**12**
Test Case Description	Display the bounding box and object detected labels
Input Data	30fps.mp4
Pre-condition	Must consist of YOLO algorithm model
Expected output	The output will show the labels of objects detected with bounding box which will be cover the objects
Procedures	object_detection.py
Screenshot of actual result	
Post-condition	The output will show objects detected on the road
Test result	**Passed**

Test Case Number	**13**
Test Case Description	Display the steering movement on the video dataset
Input Data	30fps.mp4
Pre-condition	Must consist of Autopilot.h5 model
Expected output	The output will show how much turn the steering of the vehicle make on a straight road
Procedures	AutopilotV2.py
Screenshot of actual result	
Post-condition	The output will show the position of the vehicle's steering angle
Test result	**Passed**

Test Case Number	**14**
Test Case Description	Display the steering movement on the video dataset
Input Data	30fps.mp4
Pre-condition	Must consist of Autopilot.h5 model
Expected output	The output will show how much turn the steering of the vehicle make on a curved road
Procedures	AutopilotV2.py
Screenshot of actual result	
Post-condition	The output will show the position of the vehicle's steering angle
Test result	**Passed**

ii. White box Testing

Unit Testing

Test Case ID: 1	**Function Name:** imclearborder()
Parameters taken: bw	
Test Case Description: Clear the border of the road	
Expected Result: Return bw	
Pre-Condition: Load road.png	
Test Execution Steps: 3. Convert image to grayscale 4. Apply threshold	
Post Conditions: Removes the border outside the lanes of the road	
Test Result: **Passed**	

Test Case ID: 2	**Function Name:** RemoveTop()
Parameters taken: bw, height	
Test Case Description: Clear the noise at the top of the road	
Expected Result: Top of the road will be cleared	
Pre-Condition: Load road.png	
Test Execution Steps: 1. Convert image to grayscale 2. Apply threshold 3. Clear border	
Post Conditions: Removes all the noise at top of the road	
Test Result: **Passed**	

Test Case ID: 3	**Function Name:** RemoveTriangle()
Parameters taken: bw	
Test Case Description: Apply a triangle on the road to detect the lanes	
Expected Result: Removes all the items outside the triangle shape	
Pre-Condition: Load road.png	
Test Execution Steps: 1. Convert image to grayscale 2. Apply threshold 3. Clear border	
Post Conditions: Removes all the items outside the triangle	
Test Result: **Passed**	

Test Case ID: 4	**Function Name:** RemoveTriangle()
Parameters taken: bw	
Test Case Description: Apply a triangle on the road to detect the lanes	
Expected Result: Removes all the items outside the triangle shape	
Pre-Condition: Load road.png	
Test Execution Steps: 1. Convert image to grayscale 2. Apply threshold 3. Clear border	
Post Conditions: Removes all the items outside the triangle	
Test Result: **Passed**	

Test Case ID: 5	**Function Name:** bwareaopen()
Parameters taken: bw, minArea	
Test Case Description: Measure minimum area of the white lines present on the road	
Expected Result: Removes all items that are lesser than area size	
Pre-Condition: Load road.png	
Test Execution Steps: 1. Convert image to grayscale 2. Apply threshold 3. Clear border 4. Remove top and triangle	
Post Conditions: Removes all the items except for the white lanes	
Test Result: **Passed**	

Test Case ID: 6	**Function Name:** checkSpeed()
Parameters taken: -	
Test Case Description: Measure the speed of the vehicle	
Expected Result: Speed will be showed in km/h	
Pre-Condition: Load 30fps.mp4	
Test Execution Steps: IV. User must run the video of the road	
Post Conditions: The speed of the vehicle will be showed in km/h	
Test Result: **Passed**	

Test Case ID: 7	**Function Name:** curve()
Parameters taken: x, t, y	
Test Case Description: get the curvature of the road	
Expected Result: return x[0] * t * t + x[1] * t + x[2] - y	
Pre-Condition: Load 30fps.mp4	
Test Execution Steps: -	

Post Conditions: It optimize the algorithms
Test Result: **Passed**

Test Case ID: 8	**Function Name:** diffAngle()
Parameters taken: -	
Test Case Description: Measure the angle difference of the road	
Expected Result: Difference between top and bottom angle	
Pre-Condition: Load 30fps.mp4	
Test Execution Steps: 1. User must run the video of the road	
Post Conditions: Difference angle for each frame will be showed in degrees	
Test Result: **Passed**	

Test Case ID: 9	**Function Name:** averageAngle()
Parameters taken: -	
Test Case Description: Measure the average angle of the road	
Expected Result: Average angle of the road will be showed	
Pre-Condition: Load 30fps.mp4	
Test Execution Steps: 1. User must run the difference angle function	
Post Conditions: Average angle for each frame will be showed in degrees	
Test Result: **Passed**	

Test Case ID: 10	**Function Name:** drawText()
Parameters taken: text	
Test Case Description: Draw the average angle on the output window	
Expected Result: Average angle will be showed on top left in output window	
Pre-Condition: Load 30fps.mp4	
Test Execution Steps: -	
Post Conditions: The results will be shown as blue text on output window	
Test Result: **Passed**	

Test Case ID: 11	**Function Name:** preprocess()
Parameters taken: img	
Test Case Description: Preprocess the data	
Expected Result: return resizez	
Pre-Condition: -	
Test Execution Steps: -	
Post Conditions: The image will be resized to (100, 100)	
Test Result: **Passed**	

Test Case ID: 12	**Function Name:** return_data()
Parameters taken: img	
Test Case Description: creates a pickle file gathering all the image from a folder	
Expected Result: The data will be stored in a pickle file	
Pre-Condition: -	
Test Execution Steps: 1. Open train file	
Post Conditions: The dataset will be stored in a pickle file and split into features and labels	
Test Result: **Passed**	

Test Case ID: 13	**Function Name:** keras_model()
Parameters taken: image_x, image_y	
Test Case Description: creates a group of layers into an object with training and inference features	
Expected Result: Returns model, calbacks_list	
Pre-Condition: -	
Test Execution Steps: -	
Post Conditions: The model will be compiled	
Test Result: **Passed**	

Test Case ID: 14	**Function Name:** main()
Parameters taken:	
Test Case Description: The training of the model will take place	
Expected Result: Model will be trained and save into .h file	
Pre-Condition: Load labels and features	
Test Execution Steps: 1. Load data from pickle 2. Set epoch to 30, batch_size = 32	
Post Conditions: The model will be saved as Autopilot.h5	
Test Result: **Passed**	

Test Case ID: 15	**Function Name:** keras_predict()
Parameters taken: model, image	
Test Case Description: Predict the steering angle	
Expected Result: Return steering_angle	
Pre-Condition: The images are processed	
Test Execution Steps: 1. Load Autopilot.h5 2. Load 30fps.mp4	
Post Conditions: The steering angle will be shown for each frame	
Test Result: **Passed**	

Test Case ID: 16	**Function Name:** keras_process_image()
Parameters taken: img	
Test Case Description: Process the images from dataset	
Expected Result: Return img	
Pre-Condition: -	
Test Execution Steps: -	
Post Conditions: The images are processed and resized to (100,100)	
Test Result: **Passed**	

Appendix E – User Manual

KDU PENANG UNIVERSITY COLLEGE

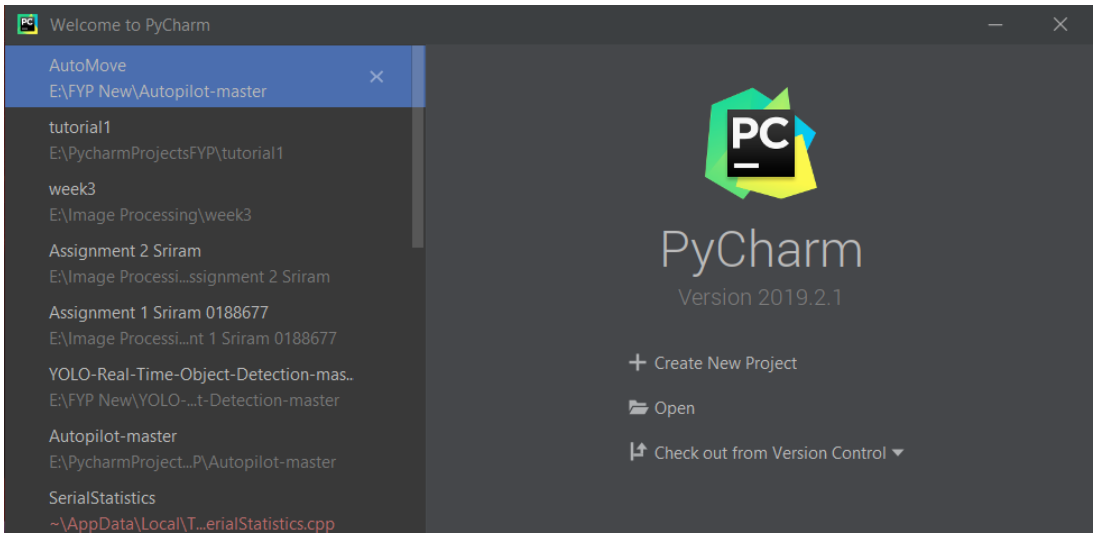
BACHELOR (HONS) COMPUTER SCIENCE

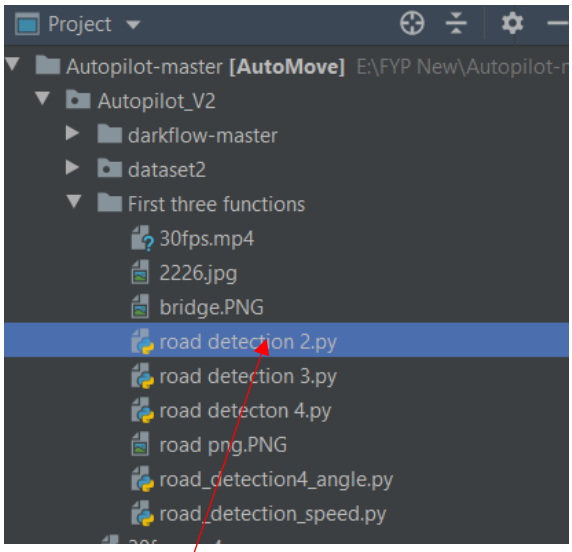
AutoMove User Manual

Chapter 1

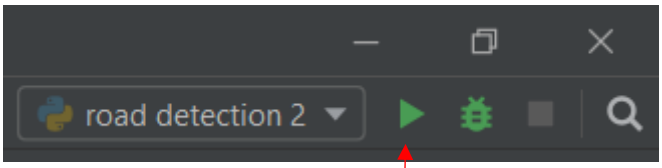
1. Lane Detection

1. Open AutoMove file in Pycharm IDE





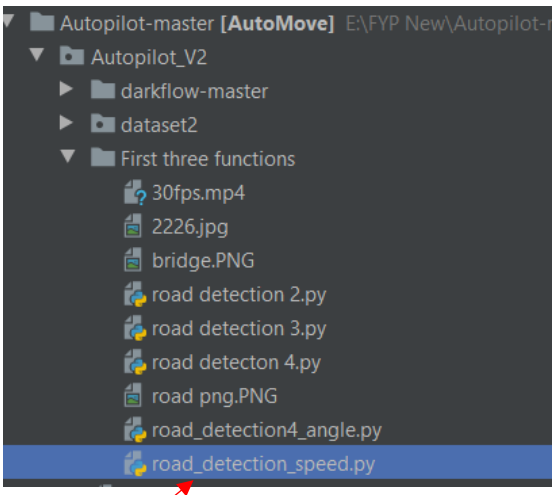
2. Open the folder “First three function” and click road detection2.py



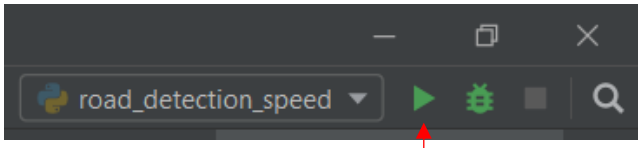
3. Click the run icon to view the lane detection

Chapter 2

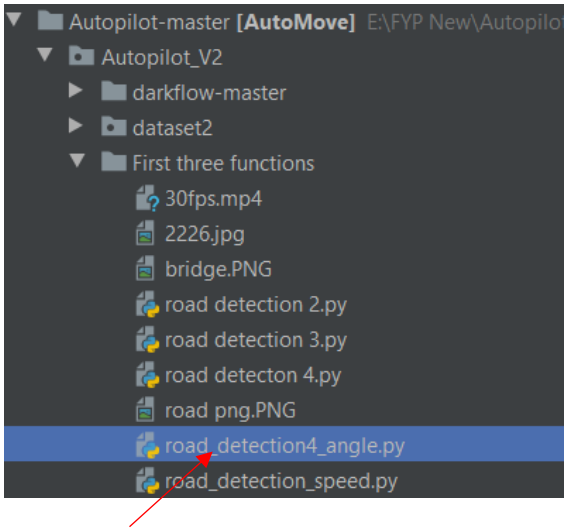
2. Speed Detection and Angle Detection



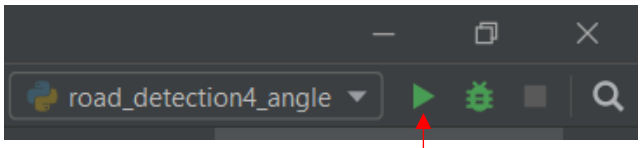
1. Open the folder “First three function” and click road_detection_speed.py



2. Click the run icon to calculate the speed of the vehicle



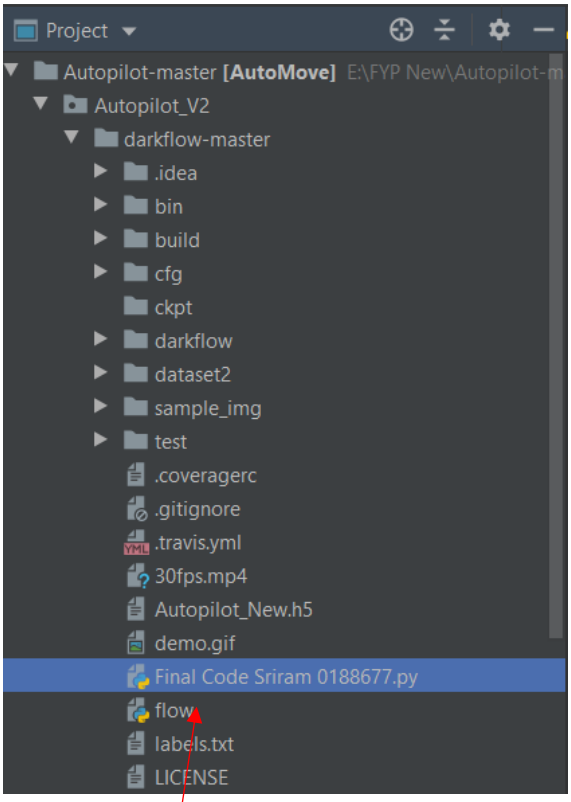
3. Open the folder “First three function” and click road_detection_angle.py



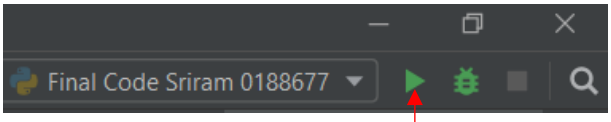
4. Click the run icon to view the angle of the road for each frame

Chapter 3

3. Autonomous Navigation



1. Open the folder “darkflow-master” and click Final Code Sriram 0188677.py



2. Click the run icon to navigate the vehicle autonomously

AutoMove - FYP Documentation Sriram 0188677

ORIGINALITY REPORT

15%	4%	2%	12%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to KDU College Sdn Bhd Student Paper	8%
2	www.ijeat.org Internet Source	2%
3	Rachel Huang, Jonathan Pedoeem, Cuixian Chen. "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers", 2018 IEEE International Conference on Big Data (Big Data), 2018 Publication	1%
4	Submitted to CSU, San Jose State University Student Paper	<1%
5	cienciaencanoa.blogspot.com.co Internet Source	<1%
6	Submitted to Monash University Student Paper	<1%
7	Submitted to Birla Institute of Technology and Science Pilani Student Paper	<1%

8	Submitted to University of Lincoln Student Paper	<1%
9	Submitted to University of Bath Student Paper	<1%
10	Submitted to VHS Virtual High School Student Paper	<1%
11	Submitted to Coventry University Student Paper	<1%
12	Submitted to MCAST Student Paper	<1%
13	medium.com Internet Source	<1%
14	Submitted to Middle East College of Information Technology Student Paper	<1%
15	Submitted to University of Northumbria at Newcastle Student Paper	<1%
16	aojp.lamost.org Internet Source	<1%
17	Submitted to University of Leeds Student Paper	<1%
18	www.mbbm.com.pl Internet Source	<1%

19	Submitted to Heriot-Watt University Student Paper	<1%
20	www.alpari.co.uk Internet Source	<1%
21	Submitted to Brunel University Student Paper	<1%
22	Submitted to University of Hong Kong Student Paper	<1%
23	Yongzheng Xu, Guizhen Yu, Yunpeng Wang, Xinkai Wu, Yalong Ma. "Car Detection from Low-Altitude UAV Imagery with the Faster R-CNN", Journal of Advanced Transportation, 2017 Publication	<1%
24	paduaresearch.cab.unipd.it Internet Source	<1%
25	Submitted to Loughborough University Student Paper	<1%
26	Submitted to National University of Ireland, Galway Student Paper	<1%
27	Submitted to AUT University Student Paper	<1%
28	Ce Li, Baochang Zhang, Hanwen Hu, Jing Dai. "Enhanced Bird Detection from Low-Resolution	<1%

Aerial Image Using Deep Neural Networks", Neural Processing Letters, 2018
Publication

29	Submitted to Carnegie Mellon University Student Paper	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		