REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI

BY

CHO WEI TUCK

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

In partial fulfillment of the requirement

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONS)

COMMUNICATIONS AND NETWORKING

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2020

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI

Academic Session: MAY 2020

I CHO WEI TUCK
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

(Author's signature)

(Supervisor's signature)

Address:
259, KAMPUNG BARU JERAM,
31850 KAMPAR, PERAK.

LAU PHOOI YEE
Supervisor's name

Date: 10th SEPTEMBER 2020

Date: 11 SEPTEMBER 2020

REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI

BY

CHO WEI TUCK

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

In partial fulfillment of the requirement

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONS)

COMMUNICATIONS AND NETWORKING

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2020

DECLARATION OF ORIGINALITY

I declare that this report entitled "**REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : CHO WEI TUCK

Date : 10th September 2020

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere thanks and appreciation to my supervisor, Dr. Lau Phooi Yee who had given me this bright opportunity to engage in an Advanced Driver Assistance System (ADAS) based project. This project would have never been accomplished without her supervision and encouragement as the image processing is not in my degree syllabus. A million thanks for your support and patience over this period.

Besides, I would like to thank my friend, Gai Reen for lending me her vehicle (Proton Axia), acting as a driver and following my instructions to adjust the vehicle's position for analysis purposes. Without her help, I might not be able to do the analysis task or might be take longer time to finish it.

Lastly, many thanks to my sister who helps me to record the demonstration video for presentation purpose and also my grandmother who always care about my project. I would like to thank my parents for their support and love throughout the entire course.

ABSRACT

The Advanced Driver Assistance System (ADAS) is invented and installed to the vehicles to protect drivers and reduce car accident by monitoring the road condition. However, not all of the vehicles can install ADAS system due to the system complexity and the great price of the required sensors. Vision-based ADAS is affordable for most of the driver as it needed only cameras as sensors to monitor the car surrounding.

Car accidents happened frequently among the teenagers. This is because they were not familiar with the road condition and lack of driving skills after they newly get their driving license. Some blind spots might difficult to be noticed by the new drivers during the driving process. Drivers might hit other vehicles and lead to small collision or serious tragic accident. Therefore, a driving monitoring system is needed to keep an eye on the road condition and provides guidance to the drivers.

The proposed system provides a surround view of the vehicle and able to alert the driver when the vehicle is near to other vehicles. Raspberry Pi is the main component of the proposed system and 3 wide angle Pi camera are used to capture the surround view. Besides, Forward Collision Warning (FCW) feature is applied to this system by using a pre-trained TensorFlow model for vehicle detection and an approximate distance algorithm to determine whether or not the distances between vehicles are closer enough to trigger the alert action.

TABLE OF CONTENTS

TITLE PAGE i

DECLARATION OF ORIGINALITY ii

ACKNOWLEDGEMENTS iii

ABSTRACT iv

TABLE OF CONTENTS v

LIST OF FIGURES viii

LIST OF TABLES xiii

LIST OF ABBREVIATIONS xiv

CHAPTER 1 INTRODUCTION 1

1.1 Problem Statement and Motivation 1

1.2 Project Scope 3

1.3 Objectives 3

1.4 Proposed Approach 4

1.5 Background 6

1.6 Project Organization 8

CHAPTER 2 LITERATURE REVIEW 9

2.1 360° Surround View System with Parking Guidance 9

2.1.1 Combing Look-up Tables 9

2.1.2 Seamless Image Stitching 10

2.1.2 Brightness Balance 11

2.2 A Surround View Camera Solution for Embedded Systems 12

2.2.1 Geometric Alignment Algorithm 12

2.2.2 Photometric Alignment Algorithm 13

2.2.3 Surround View Synthesis 13

2.2.4 Embedded Implementation 14

2.3 Three Dimensional Surround View System 14

2.3.1 3D Bowl Geometry Model 15

2.3.2 Camera Calibration 15

2.3.3 Homography Transformation and Extrinsic Parameters 16
2.4 Critical Remarks of Previous Works 17
2.4.1 Strength and Weakness Comparison of Existing Systems 17
2.4.2 Technical Information Comparison of Existing Systems 18

CHAPTER 3 PROPOSED METHOD/APPROACH 19
3.1 Full System Flow 19
3.2 System Setup 20
3.3 System Architecture 21
3.4 Flow Chart of Pre-collision Feature 23
3.5 Flow Chart of Web Streaming Feature 24

CHAPTER 4 METHODOLOGY AND TOOLS 25
4.1 Design Specification 25
4.1.1 Methodology 25
4.2 Tools to use 26
4.2.1 Hardware 26
4.2.2 Software 29
4.2.3 Programming Language 31
4.3 Implementation Issue and Challenges 32
4.4 Timeline 33

CHAPTER 5 IMPLEMENTATION AND TESTING 35
5.1 Design Analysis of Camera Mounting Module 35
5.2 Hardware Setup 42
5.2.1 Raspberry Pi with Buzzer 42
5.2.2 Raspberry Pi with Raspberry Pi Wide Angle Camera 44
5.2.3 Hardware Components with Camera Mounting Module 45

5.2.4 Camera Mounting Module Setup on Vehicle 47
5.2.5 Buzzers Setup inside the Vehicle 48
5.2.6 Monitor Raspberry Pis using VNC viewer 49
5.3 Software Setup 51
5.3.1 Raspbian OS Installation on Raspberry Pi 51
5.3.2 TensorFlow Object Detection API Setup 53
5.4 Pre-collision Feature Setup 66
5.4.1 Using TensorFlow Object Detection API 66
5.4.2 Concept of Bounding Boxes 67
5.4.3 Forward Collision Warning Feature for Frontal Raspberry Pi 68
5.4.4 Pre-collision Feature for SideA and SideB Raspberry Pi 73
5.4.5 Buzzer Selection Analysis for Pre-collision Alert Feature 76
5.5 Live Streaming Feature Setup 77
5.6 Experimental Results 78
5.6.1 Normal State for Frontal Raspberry Pi 78
5.6.2 Warning State for Frontal Raspberry Pi 80
5.6.3 Normal State for SideA Raspberry Pi 81
5.6.4 Warning State for SideA Raspberry Pi 83
5.6.5 Normal State for SideB Raspberry Pi 86
5.6.6 Warning State for SideB Raspberry Pi 87
5.6.7 Pre-collision Detection Result Analysis 89
5.7 Weakness of Proposed System 91

CHAPTER 6 CONCLUSION 94
6.1 Conclusion 94
6.2 Future Work 95

REFERENCES / BIBLIOGRAPHY 96
APPENDIX A – BIWEEKLY REPORT A-1

LIST OF FIGURES

Figure Number	Title	Page
Figure 1.1	Blind spot areas of vehicles.	1
Figure 1.2	Proposed approach of system flow.	4
Figure 1.3	The evolution of vehicles.	6
Figure 2.1	Distortion correction look-up table.	9
Figure 2.2	Project_LUT transformation.	10
Figure 2.3	Merge into one projection.	10
Figure 2.4	Seamless stitching based on weighted average.	11
Figure 2.5	Brightness balance.	11
Figure 2.6	An example geometric calibration chart.	12
Figure 2.7	Views and overlapping regions.	13
Figure 2.8	3D bowl mesh geometry.	15
Figure 2.9	Three dimensional surround view.	16
Figure 3.1	Full system diagram.	19
Figure 3.2	Monitoring of Pi camera and system.	20
Figure 3.3	System architecture.	21
Figure 3.4	How hardware components work.	22
Figure 3.5	Flow chart of pre-collision feature.	23
Figure 3.6	Flow chart of web streaming feature.	23
Figure 4.1	Prototyping model.	25
Figure 4.2	Raspberry Pi model B.	26
Figure 4.3	Raspberry Pi camera module.	27
Figure 4.4	Micro SD card.	28
Figure 4.5	Active buzzer.	28
Figure 4.6	Raspbian logo.	29
Figure 4.7	TensorFlow logo.	29
Figure 4.8	OpenCV logo.	30
Figure 4.9	Python logo.	31
Figure 5.1	Sides view of Perodua Axia.	35

Figure 5.2	First design of camera mounting module.	35
Figure 5.3	Square holes on the carton box.	36
Figure 5.4	Second design of camera mounting module.	38
Figure 5.5	Top-down view of second module.	38
Figure 5.6	Third design of camera mounting module.	39
Figure 5.7	Top-down view of third module.	39
Figure 5.8	A triangle is cut out from a box.	39
Figure 5.9	Third module is plugged into a box.	39
Figure 5.10	Frontal view.	40
Figure 5.11	SideA view.	40
Figure 5.12	SideB view.	40
Figure 5.13	Final version of camera mounting module.	41
Figure 5.14	A small box is issued inside the camera module to place the equipment for extra protection.	41
Figure 5.15	GPIO pinout diagram of Raspberry Pi 3 Model B.	42
Figure 5.16	9 jumper wires connected into 1 extended jumper wire.	43
Figure 5.17	Actual setup of Raspberry Pi with buzzer.	43
Figure 5.18	Actual setup of Raspberry Pi with camera.	44
Figure 5.19	Actual setup of wide angle cameras on the mounting module.	45
Figure 5.20	Raspberry Pi with casing.	45
Figure 5.21	Holes for jumper wires (left) and holes for ribbon cables to pass through (right).	46
Figure 5.22	Raspberry Pis and powerbanks were placed into the small box.	46
Figure 5.23	Actual setup of camera mounting module on the vehicle.	47
Figure 5.24	Position of the module was fixed using duct tape.	47
Figure 5.25	Three buzzers were pasted on the car ceiling.	48
Figure 5.26	One iPhone monitors Frontal Raspberry Pi.	49
Figure 5.27	One iPad monitors SideA Raspberry Pi (left) and another iPad monitors SideB Raspberry Pi (right).	49
Figure 5.28	Raspberry Pi Configuration.	50
Figure 5.29	IP address of Raspberry Pi.	50
Figure 5.30	Download website of Raspbian.	51

Figure 5.31	Micro SD card adapter is used to burn ISO image.	51
Figure 5.32	Raspberry Pi is successfully booted.	52
Figure 5.33	Print screen for update and upgrade Raspberry Pi.	53
Figure 5.34	Print screen for TensorFlow installation (1).	53
Figure 5.35	Print screen for TensorFlow installation (2).	54
Figure 5.36	Print screen for TensorFlow installation (3).	54
Figure 5.37	Print screen for TensorFlow installation (4).	54
Figure 5.38	Print screen for OpenCV installation (1).	55
Figure 5.39	Print screen for OpenCV installation (2).	55
Figure 5.40	Print screen for OpenCV installation (3).	56
Figure 5.41	Print screen for OpenCV installation (4).	56
Figure 5.42	Print screen for OpenCV installation (5).	56
Figure 5.43	Print screen for Protobuf Compilation and Installation.	57
Figure 5.44	Print screen for TensorFlow directory structure setup (1).	58
Figure 5.45	Print screen for TensorFlow directory structure setup (2).	58
Figure 5.46	Print screen for TensorFlow directory structure setup (3).	59
Figure 5.47	Print screen for TensorFlow directory structure setup (4).	59
Figure 5.48	Print screen for TensorFlow directory structure setup (5).	59
Figure 5.49	Print screen for TensorFlow directory structure setup (6).	60
Figure 5.50	Print screen for TensorFlow directory structure setup (7).	60
Figure 5.51	*objectdetection.py* hosted with EdjeElectronics by GitHub (1).	61
Figure 5.52	*objectdetection.py* hosted with EdjeElectronics by GitHub (2).	61
Figure 5.53	*objectdetection.py* hosted with EdjeElectronics by GitHub (3).	61
Figure 5.54	Object detection using Raspberry Pi.	64
Figure 5.55	Basic workflow of object detection.	64
Figure 5.56	Flowchart of *objectdetection.py*	64
Figure 5.57	Coordinates of bounding box.	67
Figure 5.58	Vehicle detection logic.	68
Figure 5.59	Half top-down view of Axia is detected as a vehicle with scores of 0.65.	68

Figure 5.60	Approximate distance algorithm logic for FCW feature.	69
Figure 5.61	Difference between distance in acquire image (left) and actual distance (right) of the green flowerpot**.**	69
Figure 5.62	The details about bounding box.	70
Figure 5.63	Part of python code for buzzer to produce low frequency of beeping sound.	70
Figure 5.64	Part of python code for buzzer to produce medium frequency of beeping sound.	71
Figure 5.65	Part of python code for buzzer to produce high frequency of beeping sound.	71
Figure 5.66	The approximate distance of 0.4 (left) and its actual distance (right).	72
Figure 5.67	Driver vision for Figure 5.66.	72
Figure 5.68	The approximate distance value is keep changing when a vehicle is passing the own car.	73
Figure 5.69	Approximate distance algorithm logic for side detection.	74
Figure 5.70	Part of python code for buzzer to produce beeping sound for side detection.	74
Figure 5.71	The approximate distance of 0.2 for side detection (left) and its actual distance (right).	75
Figure 5.72	The approximate distance of 0.1 for side detection (left) and its actual distance (right).	75
Figure 5.73	Active buzzer (left) and passive buzzer (right).	76
Figure 5.74	Part of python code for streaming feature.	77
Figure 5.75	Live feed of SideA Pi camera in web browser.	77
Figure 5.76	Live feed of SideB Pi camera in web browser.	77
Figure 5.77	Vehicle 1 is detected with a bounding box.	78
Figure 5.78	Vehicle 1 is not detected with a bounding box.	79
Figure 5.79	Vehicle 2 is detected with a bounding box.	79
Figure 5.80	Vehicle 1 is detected with approximate distance value of 0.5 by Frontal Raspberry Pi.	80
Figure 5.81	Vehicle 1 is detected with approximate distance value of 0.4 by Frontal Raspberry Pi.	80

Figure 5.82	Vehicle 1 is detected with approximate distance value of 0.3 by Frontal Raspberry Pi.	81
Figure 5.83	Vehicle 2 is detected with approximate distance value of 0.5 by Frontal Raspberry Pi.	81
Figure 5.84	Vehicle 2 is detected with approximate distance value of 0.4 by Frontal Raspberry Pi.	82
Figure 5.85	Vehicle 2 is detected with approximate distance value of 0.3 by Frontal Raspberry Pi.	82
Figure 5.86	Vehicle 1 is detected with approximate distance value of 0.2 by SideA Raspberry Pi.	83
Figure 5.87	Vehicle 2 is detected with approximate distance value of 0.3 by SideA Raspberry Pi.	83
Figure 5.88	Vehicle 1 is detected with approximate distance value of 0.1 by SideA Raspberry Pi.	84
Figure 5.89	Vehicle 2 is detected with approximate distance value of 0.1 by SideA Raspberry Pi.	85
Figure 5.90	Vehicle 1 is detected with approximate distance value of 0.2 by SideB Raspberry Pi.	86
Figure 5.91	Vehicle 2 is detected with approximate distance value of 0.2 by SideB Raspberry Pi.	86
Figure 5.92	Vehicle 1 is detected with approximate distance value of 0.1 by SideB Raspberry Pi.	87
Figure 5.93	Vehicle 2 is detected with approximate distance value of 0.1 by SideB Raspberry Pi.	88
Figure 5.94	Screenshot from SideB (left) and screenshot from SideA (right).	89
Figure 5.95	Prayer house was detected as a train.	91
Figure 5.96	Proton Axia was detected as a train.	92
Figure 5.97	No object around but still detected as a train.	92
Figure 5.98	Screenshot of false positive result.	93
Figure 5.99	The motorcycle is not be detected.	93

LIST OF TABLES

Table Number	Title	Page
Table 2.1	Strength and weakness comparison of existing systems.	17
Table 2.2	Technical information comparison of existing systems.	18
Table 4.1	Gantt chart of FYP1 project timeline.	33
Table 4.2	Gantt chart of FYP2 project timeline.	34
Table 5.1	Comparison of images captured using first camera mounting module.	37
Table 5.2	Comparison of camera mounting module design.	41
Table 5.3	Existing pre-trained object detection model.	66
Table 5.4	Pre-collision detection results.	89
Table 5.5	Result description for results analysis	90

LIST OF ABBREVIATIONS

ADAS	Advanced Driver Assistance System
FCW	Forward Collision Warning
WHO	World Health Organization
MIROS	Malaysian Institute of Road Safety Research
GPS	Global Positioning System
CV	Computer Vision
VB-DAS	Vision-Based Driver Assistance System
LUT	Look-Up Table
LDC	Lens Distortion Correction
ROI	Regions of Interest
AE	Auto Expose
AWB	Auto White Balance
3D	Three Dimensional
DSP	Digital Signal Processors
FOV	Field of View
2D	Two Dimensional
SDLC	System Development Life Cycle
HDMI	High-Definition Multimedia Interface
USB	Universal Serial Bus
MIPI	Mobile Industry Processor Interface
GPIO	General Purpose Input / Output
SoC	System on a Chip
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
DSI	Display Serial Interface
SD	Secure Digital
MBPS	Megabits Per Second
LAN	Local Area Network
CSI	Camera Serial Interface
OV	Omni Vision

GB	Gigabyte
OS	Operating System
API	Application Program Interface
BSD	Berkely Software Distribution
COCO	Common Objects in Context
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

CHAPTER 1: INTRODUCTION

1.1 Problem Statement and Motivation

Advanced Driver Assistance System (ADAS) are designed to provide drivers a safety and comfort driving condition. Besides, ADAS can also reduce the human errors by providing drivers a guidance about the vehicle surroundings and give some warning signals to a driver about the incoming danger. Nowadays, ADAS systems became more common among latest-generation vehicles. Some existing ADAS system examples like driver drowsiness detection, intelligent speed adaptation, automatic parking, automatic braking, adaptive cruise control and blind spots monitor (Willian, 2017).

Unfortunately, ADAS systems apply only on advanced luxury cars due to the high cost and complexity of these systems because of` several sensors employed. Therefore, vision-based ADAS are the first choice as monitoring system because their cost are lower and their availability is higher compared to other ADAS systems. In addition, vision-based ADAS have the ability to combine with other systems in order to meet different user requirement. Around view system will provide a complete top down view (or bird's eye view) of the vehicle surroundings. This causes the blind spots around the vehicle can be seen by the novice drivers. Several cameras are used to set up 360 degree vision in order to display a stitched image made from the camera data (Liu, Lin & Chen, n.d.). However, there may be more blind spot appeared due to the larger size of the vehicle and traditional four cameras might not be enough to present a clear top down view.

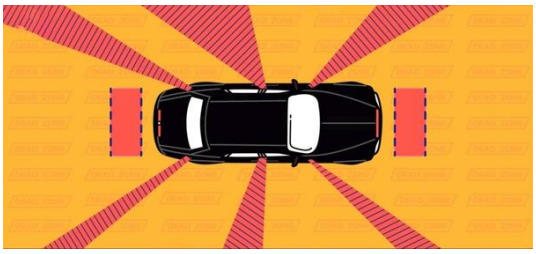


Figure 1.1: Blind spot areas of vehicles.

One of the major causes of road accidents is due to the drivers' inattention and aggressive driving behaviour. According to the reports of World Health Organization (WHO), around 1 to 1.24 million people are killed while 20 to 50 million people are

injured on the road every year across the world (WHO, 2020). In 2018, transport accidents are the 4th main cause of death in Malaysia (Shaheera Aznam Shah, 2019). The integration of ADAS system may have the potential to assist drivers and to reduce traffic accident fatalities, i.e. the system could help to keep an eye on the driving environment. However, due to cost, most of the vehicles nowadays still do come with ADAS system due to its cost and complexity. Some vehicles do not have even a rear camera monitoring system. Thus, recently, many inexpensive driving monitoring systems are being developed, which comes with several features such as accurate, easy pluggable, efficient and user friendly.

Car accidents are the leading cause of death for teenagers in Malaysia. According to a report of Department of Statistics Malaysia, around 22.5% of Malaysia population aged 15 to 40 years died because of the transport accidents in 2017 (Mohn Uzir Mahidin, 2018). In fact, adolescents are more likely to be involved in an accident, compared to adults. The first 18 months after teenagers obtained their driving license are the most dangerous, as during that period, statistics shows that novice drivers are four times as likely as adults to be involved in an accident. The reason is because they are inexperience in driving car and have tendency to be distracted. All novice drivers start off being inexperienced, no matter how careful they are. They will be subjected to many distractions such as notification from smart phones, chatty passengers in the car, or listening to their favourite song playing on the radio. These actions may cause the most common types of car accident which is the rear-end collisions. National Highway Transportation Safety Administration (NHTSA) stated that 87% of the rear-end collision occurred due to these distraction events (Sam C. Mitchell & Associates, 2018).

Moreover, novice drivers with no experience in driving may not noticed these blind spots when driving, as they are inexperience. These novice drivers may encounter minor collisions with other vehicles, especially during heavy traffic conditions. Therefore, cheap novice drivers driving assistant system is needed. These driving assistant system should offer features such as real-time streaming function which could be used to monitor novice drivers' behavior by observing whether or not the drivers drive too close to the front and side vehicle could be the most feature for novice drivers, in order to increase on the road and car safety.

1.2 Project Scope

This project involves developing a real-time 360 degree around view driving monitoring framework for novice drivers. The driving monitoring system consists of three fisheye cameras pointing to the front, the left and the right side of the vehicle, to capture image data of vehicle surrounding. Pre-collision features are being added into this system using TensorFlow model. Besides, this system will be embedded into the Raspberry Pi 3 Model B+ to process the captured data and show the surrounding view of the vehicle in real-time. The dual-band Wifi 802.11ac component in the Raspberry Pi will be used for real-time live streaming of the surrounding view, as guardians or trainer can watch and observe the novice drivers' driving behaviour. This prototyping system will be tested using an economic small passenger cars size vehicle in day light and good weather conditions.

1.3 Objectives

The primary objective in this proposed project is to develop a prototype of real-time 360 degree surrounding view driving monitoring system for new drivers. Here are the sub-objectives to achieve the main objective:

- To monitor the novice drivers' driving behaviour by providing an alarm
- To apply live streaming feature into the system for parents and trainers to monitor novice drivers' driving behaviour
- To apply pre-collision warning feature for alerting novice drivers before collision occurs
- To alert the novice driver about the pre-collision event by using buzzer

This project is proposed to hope that the probability of car collision event occurred will be decreased. Novice driver will be become safer while driving a vehicle by using this proposed system.

1.4 Proposed Approach

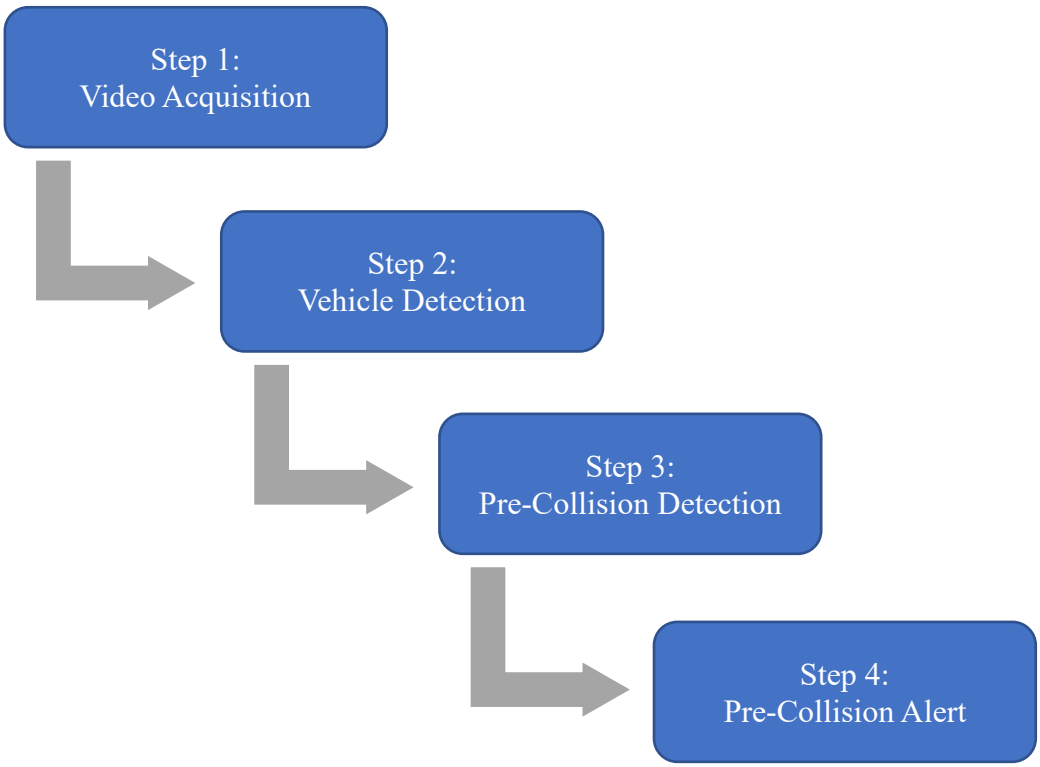


Figure 1.2: Proposed approach of system flow.

Video acquisition is the first step of the proposed framework. Three 175 degree wide angle Pi camera are used and each Pi camera is attached with a Raspberry Pi using a 15-Pin Ribbon Cable to achieve the video acquisition. The videos are captured in real-time and in 640p with 10 frames per second.

The second step is the vehicle detection. In this step, TensorFlow Object Detection API is used to detect different types of vehicles in real-time. Besides, the pre-trained TensorFlow model "ssdlite_mobilenet_v2_coco" is selected for the project due to its efficiency when performing actual detection. Only three categories are selected which are 'car', 'bus' and 'truck' as the proposed system is used in a normal road driving scenario. When an object is detected by the model, it will return a "score" to represent the level of confidence for the detected object. Therefore, an object will be detected as a car, bus and truck when it has a score of above 0.8.

The third step is the Pre-collision Detection. Two different types of distance algorithms are employed to determine whether or not the own car is too near to the other vehicles. Due to the OpenCV will visualise the results of the detection, a bounding box will be drawn around the detection vehicle. In this step, the x-axis of the bounding box is used to detect pre-collision for Frontal side and the y-axis of the bounding box is used to detect pre-collision for SideA and SideB.

The last step is the pre-collision alert. When the detected vehicle is determined as too neat with the own vehicle and almost causes collision, by the distance algorithms, the system will trigger the buzzer to alert the driver. A “Warning!!!” sign will also display on the screen.

1.5 Background

The mechanization of transportation started in the era after 1750 which is the year of industrial revolution began. New engine technologies offer the ability of being used across several modes with specific adjustment. Steam engine is the first most significant engine innovation that enhance the performance of the maritime and railway modes at the end of the eighteenth century (Rodrigue 2020).

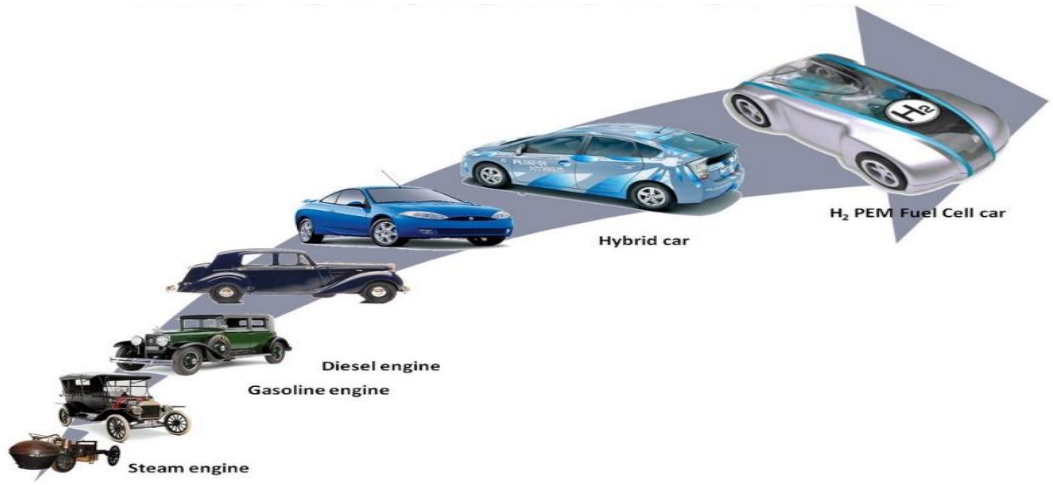


Figure 1.3: The evolution of vehicles.

The automation of transport system was developed in the 21st century. Transportation nowadays plays an important role among people. It brings benefits to the social community such as the economy and quality of life. However, these benefits are not for free. People pay money for vehicles' purchase, operation and maintenance costs, and ecological costs such as resource utilization, air and noise pollution, traffic jams, and even pay with their lives due to fatal or dreadful traffic accidents. Some actions are made to enhance the quality of modern transport system at each level of the community vary from government policies to individual drivers' performance (WHO). Vision Zero is a main objective of such improvement. Vision Zero is a strategy to prevent all traffic fatal and severe injuries. However, its broader spectrum will only focus on the studies and system developed to improve the driver performance and road safety.

A new automotive safety system, known as Advanced Driver Assistance Systems (ADAS) has developed and still improving as human needs to be more safety

on the road. The developers need to add new features to meet consumer requirement and expectation. ADAS make use of GPS, cameras, sensors such as radar, sound or laser range detector to get measurement and detect distances to the objects around the vehicle. Some existing ADAS system examples like parking assistant system, lane departure warning system and traffic assistance system. Parking assistant is an ultrasound system that warns the driver when the rear distance to an obstacle is below a certain limit. Besides, alert function of lane departure warning system will be triggered while any wheel of a car crosses a lane delimiter line. Traffic assistance system provides the ability of detecting and interpreting the traffic signals to the vehicles.

Most of the inventors made use of Computer Vision (CV) and image processing to set up Vision-Based Driver Assistance System (VB-DAS). To improve safety and better comfort of vehicles, one or more multiple cameras is used along with built-in sensors. A blind spots refers to the area surrounding the car that cannot be seen while the driver is looking forward or through either rear-view mirrors or side mirrors. Luckily, this system provides a great help to the drivers as they can see clearly around each corner of the vehicle. Even when some driving issues happen such as driving through narrow areas or driving along roads with bad condition, the drivers also can easily get rid of these problems using vision-based ADAS.

Nowadays, only luxurious cars and jeeps consist of this smart driving system with multiple cameras and built-in sensor. Therefore, an easy pluggable and affordable smart driving system should be developed for those ADAS not supported vehicles.

1.6 Project Organization

This will be 6 chapter in this paper. Chapter 1 includes problem statement and motivation that related to the topic, project scope, objective and background of the proposed project, and the proposed approach. Chapter 2 contains the literature reviews of the project, and the critical remarks of previous work. Besides, Chapter 3 consists of the full system flow, system setup, system architecture and flowchart for both pre-collision and web streaming feature. Next, Chapter 4 explains the methodology of the proposed project, tools to use in this project, implementation issue and challenges, and the timeline to develop this project. Moreover, Chapter 5 contains the details of implementation and testing for this project. Lastly, Chapter 6 consist of the conclusion and future work of this project.

CHAPTER 2: LITERATURE REVIEW

2.1 360° Surround View System with Parking Guidance

Yu and Ma (2014) presented a surround view system that can provides 360 degree surround view and applied parking guidance feature in their study by using four 180 degree fisheye cameras. They used some steps to generate top-down view for their system. All images will be registered on a planar surface after the correction of the distortion of four fisheye image. In order to create a high-quality image result, a flexible image stitching method to seam of adjacent images away was developed. Because of the images are captured with different exposure condition, a special algorithm for the brightness balance was used to compensate the difference of exposure. In this study, three main issues are listed out to solve and improve the achievement of surround view generation.

2.1.1 Combing Look-up Tables

Yu and Ma (2014) applied look-up tables (LUT) for distortion correction and projection transformation. The distortion curve given by lens supplier is used to adjust the radial distortion caused by fisheye lens.

Field Of View Angle (degree)	Real Image Height (mm)	Ref. Image Height (mm)	Distortion (%)
0	0	0	0
0.9	0.02338706	0.02338941	-0.01004311
1.8	0.04677156	0.04679036	-0.0401729
2.7	0.07015096	0.07021443	-0.09039081
3.6	0.0935227	0.09367323	-0.16069923
4.5	0.11688422	0.11717845	-0.25110146

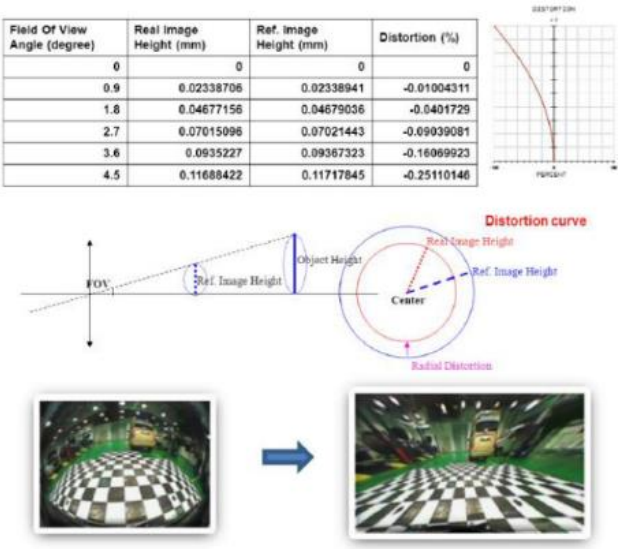


Figure 2.1: Distortion correction look-up table.

Besides, they used the projection transformation algorithm to generate a projection look-up table for the top-down view. A holography matrix is set up and a top-down view is obtained by using the projection transformation.

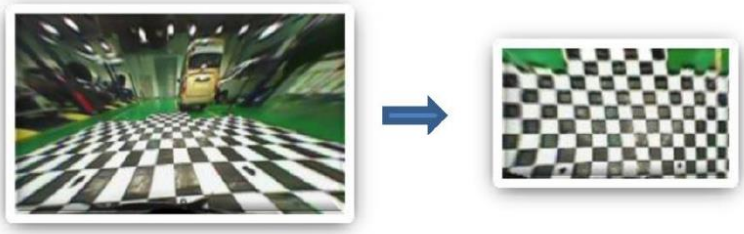


Figure 2.2: Project_LUT transformation.

In this study, they combined both distortion and project transformation LUT into one to achieve lower computational cost by reduce it into half and the running time of the system. As a result, the source fisheye can be direct projected to top-down view image.

Figure 2.3: Merge into one projection.

2.1.2 Seamless Image Stitching

Yu and Ma (2014) proposed the image fusion method to remove those obvious seams of the image to obtain visual consistency. This method can easily to be achieved because it depends on weighted average smooth transition in pixel value which can satisfy the general application requirements. To make the process of camera calibration and image stitching easier, sort of a chess board mat is used by most of the researchers and car makers for pre-processing steps in order to generate surround top-down view. Throughout this method, those errors can be easily recognized by the system installer as it can easy to find out those mismatches of a chess board mat which consists of hundreds of black and white squares.

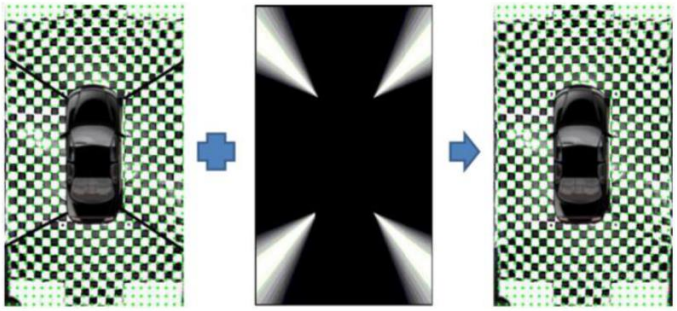


Figure 2.4: Seamless stitching based on weighted average.

2.1.3 Brightness Balance

Brightness compensation is needed in surround view generation as the images were captured in different condition using different cameras and need to be merged into one image. Therefore, Yu and Ma (2014) applied the gain compensation algorithm to reduce the differences of global intensity. By using this algorithm, the images can be tuned into similar exposure.

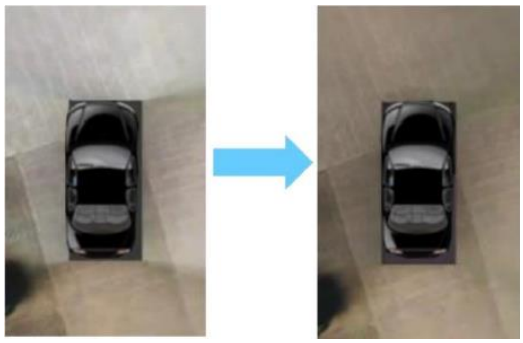


Figure 2.5: Brightness balance.

2.2 A Surround View Camera Solution for Embedded Systems

Zhang et al. (2014) brought out a camera solution for capturing around view including three main algorithm components which are geometric alignment, photometric alignment, and composite view synthesis. Their calibration chart based solution successfully generates a bird's-eye view of the vehicle surrounding by using four cameras. This camera solution is created for embedded system. The embedded system operates real-time on Digital Signal Processors (DSP) C66x and produces an 800 x 1080 high definition (HD) output video at 30fps.

2.2.1 Geometric Alignment Algorithm

Geometric alignment is a calibration-chart-based approach and consists of both lens distortion correction (LDC) for fish-eye cameras and perspective transformation. It is used for fish-eye lens distortion correction from the input videos. It is also used to convert each input frame from the respective perspective to a top-down perspective. Zhang et al. (2014) outline the process of geometric alignment by dividing into several steps. LDC correction is the first step of the algorithm and is applied to each frame. Next, they applied initial perspective transformation to each LDC corrected frame. To find Regions of Interest (ROI), Harris corner detection is executed in the image information in the adjacent views' overlapping area. The filtration of the raw Harris corner is conducted to find the strongest corners and BRIEF descriptor of each corner feature is calculated to match corners from two cameras using BRIEF scores. Next is to find the suitable perspective matrix for each frame that reduces the distances between matched features. Lastly, LUT is created for encoding the information from LDC and perspective information.

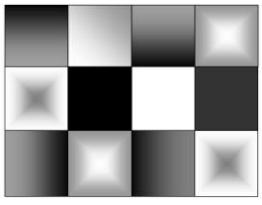


Figure 2.6: An example geometric calibration chart.

2.2.2 Photometric Alignment Algorithm

As for the photometric alignment step, Zhang et al. (2014) successfully made sure of the benefit of using a chess board mat. Using different cameras to capture a same object can be different in terms of the colour and brightness of the same object due to different light condition, Auto Expose (AE), and Auto White Balance (AWB) of the camera. Therefore, Zhang et al. (2014) applied this algorithm on the surround view system to balance the brightness and the colour to achieve seamless stitching. Some overlapping regions occured in the composite around view after geometric alignment process. Therefore, they used the overlapping image data to execute tone mapping functions for photometric correction.

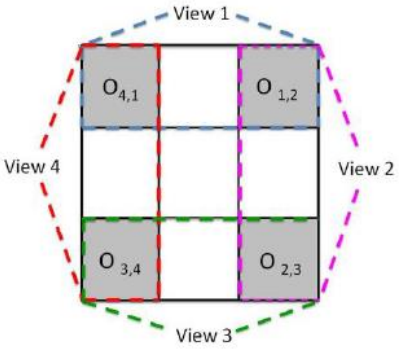


Figure 2.7: Views and overlapping regions.

2.2.3 Surround View Synthesis

This synthesis generates the stitched output by using the mapping encoded in the geometric LUT. Some overlapping areas occurred in the output frames where the image information from both adjacent input frames are needed. Each output pixel matches the pixel locations in two adjacent input images among the overlapping areas. In this paper, they used these overlapping data for surround view synthesis. Moreover, they have used the standard alpha-blending techniques for image stitching.

2.2.4 Embedded Implementation

Zhang et al. (2014) proposed their solution to satisfy the embedded system operation and memory requirements. Moreover, the proposed solution is proven through their prototype demonstration. The geometric function receives input images from the fisheye cameras and generates the output geometric LUT based only on the camera locations. The LUT will not modify remarkably after the beginning of the set up. Therefore, the system framework executed the geometric alignment function only once when the system is triggered. The output geometric LUT is stored in the memory. In the overlapping regions of input frames, there are some statistics required by photometric function which are the block averages of the image data. Hypothetically, the photometric alignment function should be in charge of collecting these statistics. To collect these statistics, input frames are required to be access two times for each output, one time for synthesis and one time for photometric. Besides, accumulate these statistics in synthesis function for the current frame n, and use the statistics for photometric correction in the consecutive frame (n+1). This action reduces the memory bandwidth. However, all the pixel-level will be limited because of this design and each frame requires computationally-intensive operations to the synthesis function block. This will leads to one-frame latency but their demonstration didn’t need for good image quality.

2.3 Three Dimensional Surround View System

Chavan and Shete (2017) proposed a system that provides a 360 degree around view by using four fisheye lens cameras set around the vehicle. The camera calibration method used in this paper is to create inverse prospective mapping of undistorted images onto 3D bowl model, stitched those images on the bowl geometry, and finally a top-down view is produced and can be selected by the driver. The purpose of the proposed system is to eliminate blind spot area during driving section. Moreover, this system operates real time on DSP C66x producing surround view around vehicle.

2.3.1 3D Bowl Geometry Model

The two dimensional view system modified and stitched the images on ground plane. Besides, the two dimensional view system has also hard to find out an obstacle or a presence of pedestrian near vehicle. This is because the images are stitched on a flat, ground mapping plane for analysis. Therefore, this 3D system is proposed in this paper. The 3D mesh bowl geometry can set up using the graphics libraries which is supported by the hardware. In addition, the 3D mesh bowl geometry is used as the mapping plane rather than using a ground plane. The 3D mesh grid geometry includes four fisheye camera image mapping with the help of some essential steps which are camera calibration, LUT and image blending (Chavan & Shete, 2017).

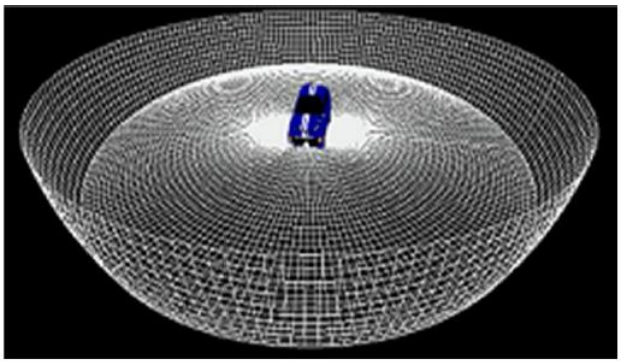


Figure 2.8: 3D bowl mesh geometry.

2.3.2 Camera Calibration

The fisheye camera lens has Field of View (FOV) of 180 degree which has the ability to involve the whole hemispherical field in front of the camera. Besides, it is impossible for the mapping of the hemispherical FOV on a finite image plane by a perspective mapping. The perspective mapping is based on the transformed of 3D world coordinates to the 2D image coordinates. The inverse perspective mapping means that the 2D image ground plan have to apply on 3D bowl geometry plan. This inverse perspective mapping applied on the captured images from four fisheye cameras with the help of camera calibration process.

The purpose of camera alignment is to define a different set of camera alignment parameters, which are extrinsic and intrinsic parameters. These parameters are used for mapping purpose between 3D world coordinates and 2D image coordinates. The focal

length, scale factor and principle point are considered as the intrinsic parameters of fisheye camera. Moreover, they used the camera alignment method that proposed by Zhang. They made use of a chess board as a camera alignment pattern. Several images were captured with the chess board in different positions and angles. Finally, they used the Open CV procedure of Zhang's method to acquire intrinsic parameters of camera (Chavan & Shete 2017).

2.3.3 Homography Transformation and Extrinsic Parameters

Homoghraphy transformation is used to stitch four images onto a reference 3D mesh bowl geometry plane in order to create a vehicle top-down view. The four fisheye camera took images from four directions around the car which are the front, left, right, and the back. The homography transformation matrix helps them to clarify the relationship between 3D plane and each image. Besides, the extrinsic parameters contain translational vector (T) and rotational matrix (R). The extrinsic parameters also correspond to the coordinate system transformation from 3D world coordinate system to 3D camera coordination system (Chavan & Shete 2017).

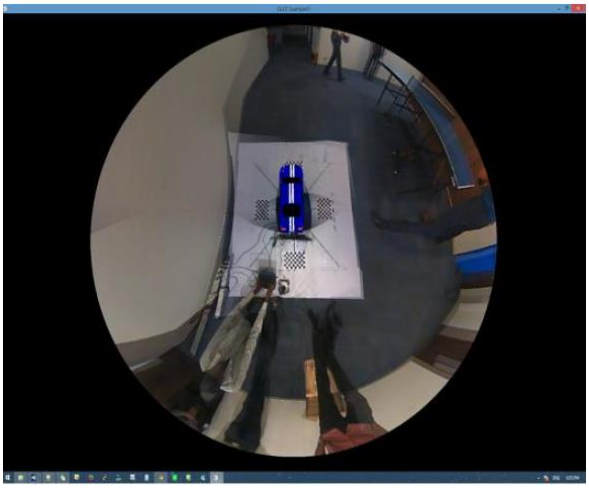


Figure 2.9: Three dimensional surround view.

2.4 Critical Remarks of previous works

2.4.1 Strength and Weakness of Existing Systems

Existing System	Strength	Weakness
360° Surround View System with Parking Guidance (Yu and Ma 2014)	- Reduces computational cost into half by combining of LUT - Extra parking Guidance for this system	- 2D-based around view may consists of undetected blind spot
A Surround View Camera Solution for Embedded Systems (Zhang et al. 2014)	- High performance ratio of image processing application due to DSP-based hardware - Reduced memory bandwidth	- require higher cost for hardware compared to other hardware options - limitation of all pixel-level leads to one frame latency may affect image quality
Three Dimensional Surround View System (Chavan & Shete 2017)	- completely remove blind spot due to the 3D surround view system - High performance ratio of image processing application due to DSP-based hardware	- require higher cost for hardware compared to other hardware options

Table 2.1: Strength and weakness comparison of existing systems.

2.4.2 Technical Information Comparison of Existing Systems

Existing Systems	Methods of Image Registration	Methods of Distortion Correction	Methods of Projection transformation	Methods of Image stitching	Methods of Image Alignment
360° Surround View System with Parking Guidance (Yu and Ma 2014)	Custom pattern registration (Chess broad pattern)	Lens Distortion Curve (LDC) provided	Holography Matrix	Alpha-blending techniques	Gain Compensation Algorithm
A Surround View Camera Solution for Embedded Systems (Zhang et al. 2014)	Custom pattern registration (Chess broad pattern)	Lens Distortion Curve (LDC) in Geometric Alignment Algorithm	Harris corner and BRIEF descriptor in Geometric Alignment Algorithm	Alpha-blending techniques	Photometric Alignment Algorithm
Three Dimensional Surround View System (Chavan & Shete 2017)	Custom pattern registration (Chess broad pattern)	Camera calibration using OpenCV	Homoghraphy Transformation and Extrinsic Parameters	Alpha-blending techniques	Photometric Alignment Algorithm

Table 2.2: Technical information comparison of existing systems.

CHAPTER 3: PROPOSED METHOD / APPROACH

3.1 Full System Flow

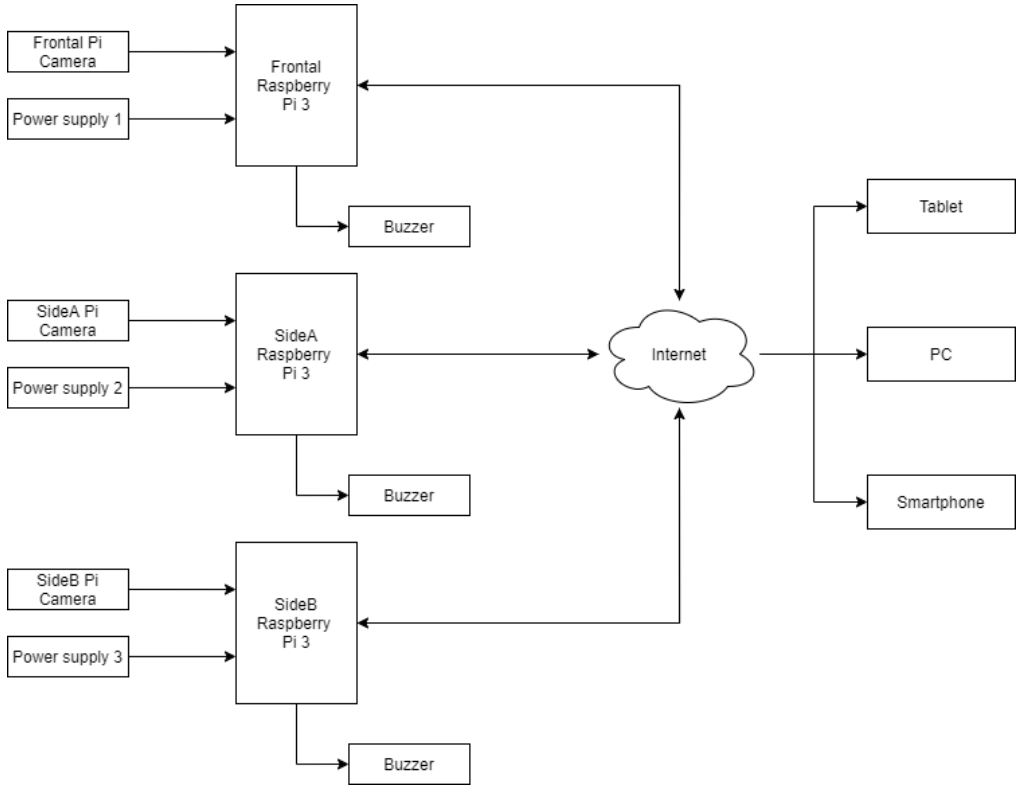


Figure 3.1: Full system diagram.

In this project, the driving monitoring system is embedded into the Raspberry Pi, being the deployment platform. In the prototype, we deployed three Raspberry Pis, each mounted with a Raspberry Pi camera. The power supply is provided to each Raspberry Pi by connecting it to a 20000mAh powerbank, with output 5V 2.1A, as the Raspberry Pi would not be able to be powered on stably below 2.0A. Besides, all the Raspberry Pi are being connected to the same network, to allow the acquired video stream to be delivered to a web browser or VNC viewer. Drivers can monitor the car surrounding through tablet, laptop or smartphone using VNC viewer. Moreover, if parents or trainers want to monitor the new drivers' driving behaviour, they can require the drivers to stream the acquired video stream to the Internet so that they can watch the video stream using web browser. However, the vehicle detection will not work while streaming the car surrounding to the Internet. In the prototype design, three Raspberry Pis are connected to a buzzer to produce an output warning beeping sound as alert.

3.2 System Setup

The hardware components are set up on top of a vehicle. By deploying three Raspberry Pi and three 175 degree wide angle Pi cameras, we could capture 360 degree surrounding view of the vehicle – see Figure 3. Each Pi camera will be mounted as shown in Figure 3 to allow a 360 complete monitoring of the vehicle surrounding, presented in three different video streams.

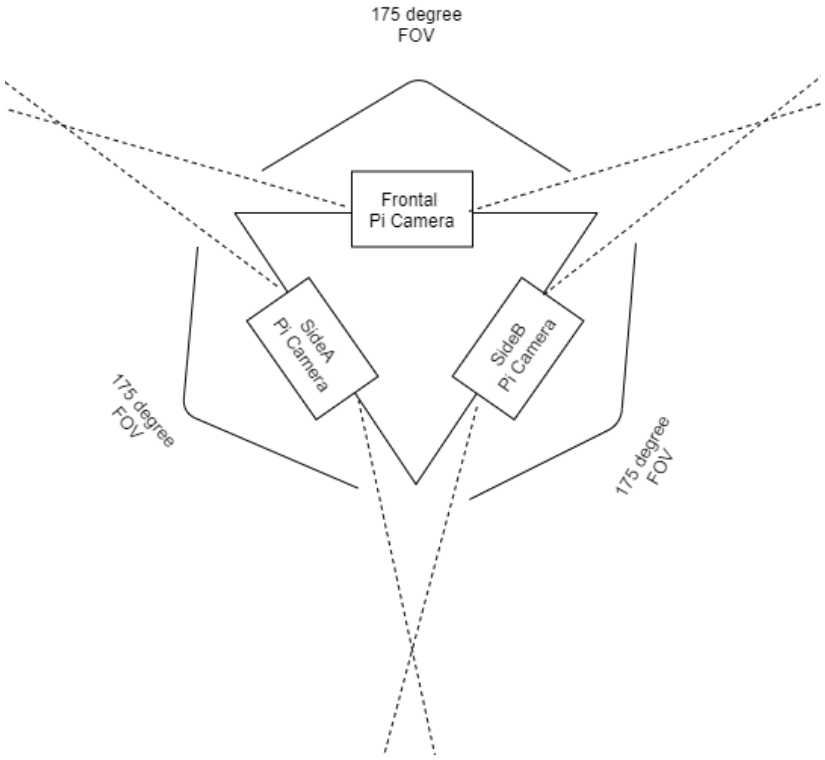


Figure 3.2: Mounting of Pi camera and system.

In the system design, each Raspberry Pi Camera are set into different position which are the Frontal, SideA and SideB, as shown in Figure 3.2. Besides, each Raspberry Pi set handles two functionalities. Pre collision feature is one of the functionalities and it is embedded to detect whether or not the driver drives too close to other vehicles. Another functionality which is the web streaming feature. Each Raspberry Pi able to stream the acquired video of their relevant position in real-time to the Internet and watch it via web browser.

3.3 System Architecture

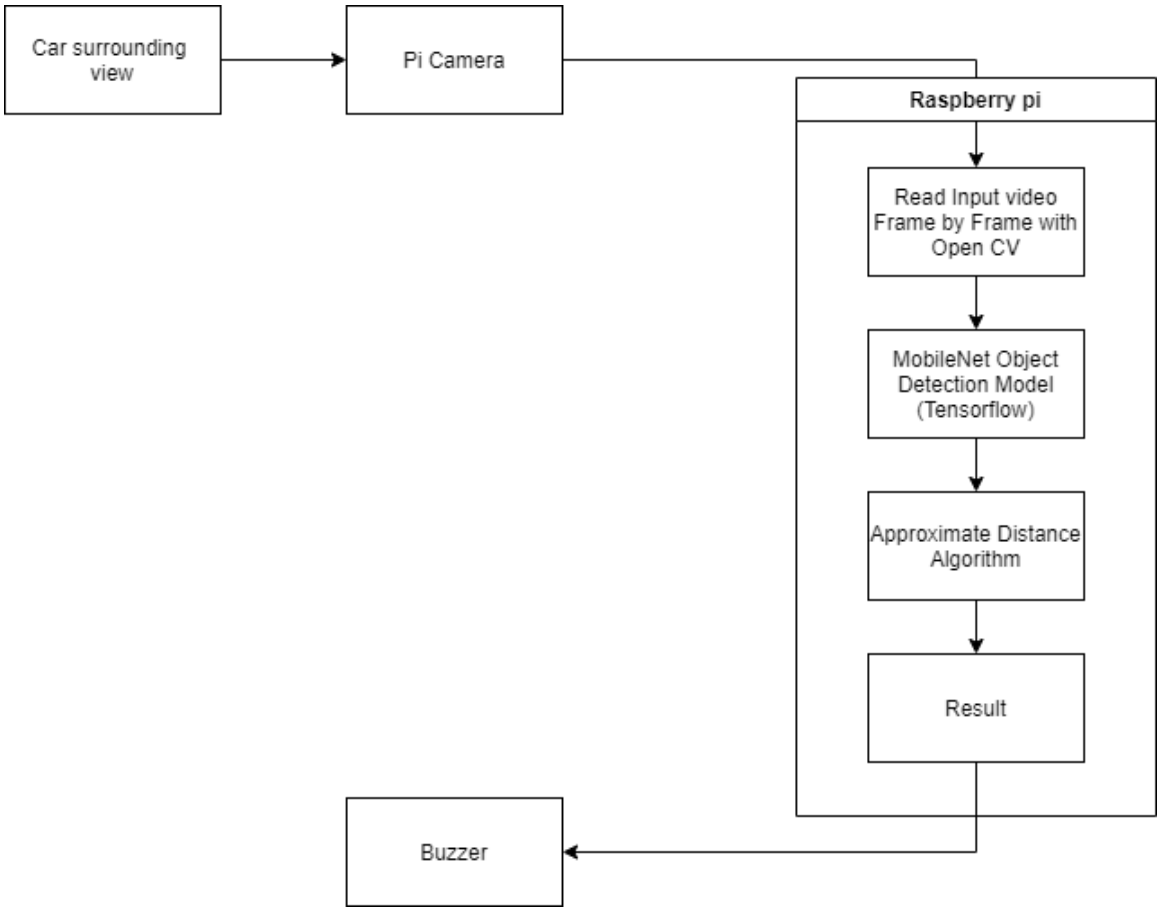


Figure 3.3: System architecture.

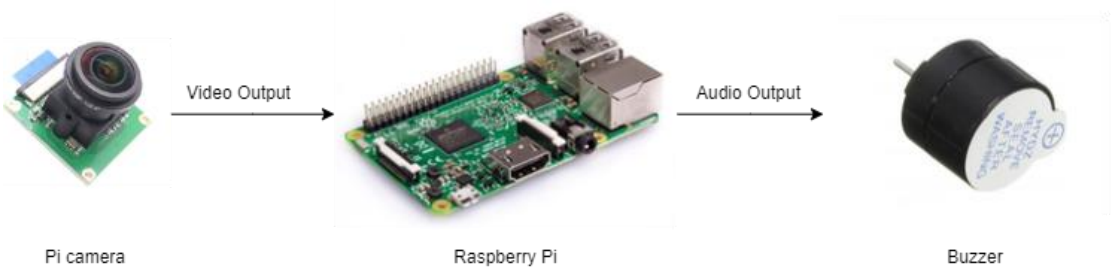


Figure 3.4: How hardware components work.

The system processes the acquired input video stream to determine whether or not the driver is driving dangerously, i.e. if his/her vehicle is too close to the other nearby vehicles. Firstly, each Pi camera will capture the relevant view of vehicle which are the Frontal. SideA and SideB. The captured video stream will be processed in each Raspberry Pi to determine if there are any vehicles nearby, and to notify drivers if they are driving too close to the other nearby vehicles. The MobileNet object detection model by TensorFlow is being embedded into the Raspberry Pi and is being used to detect nearby vehicles. The detection results will be shown on the screen, and an alert beeping sound by the speaker can be heard, by the drivers.

3.4 Flow Chart of Pre-collision Feature

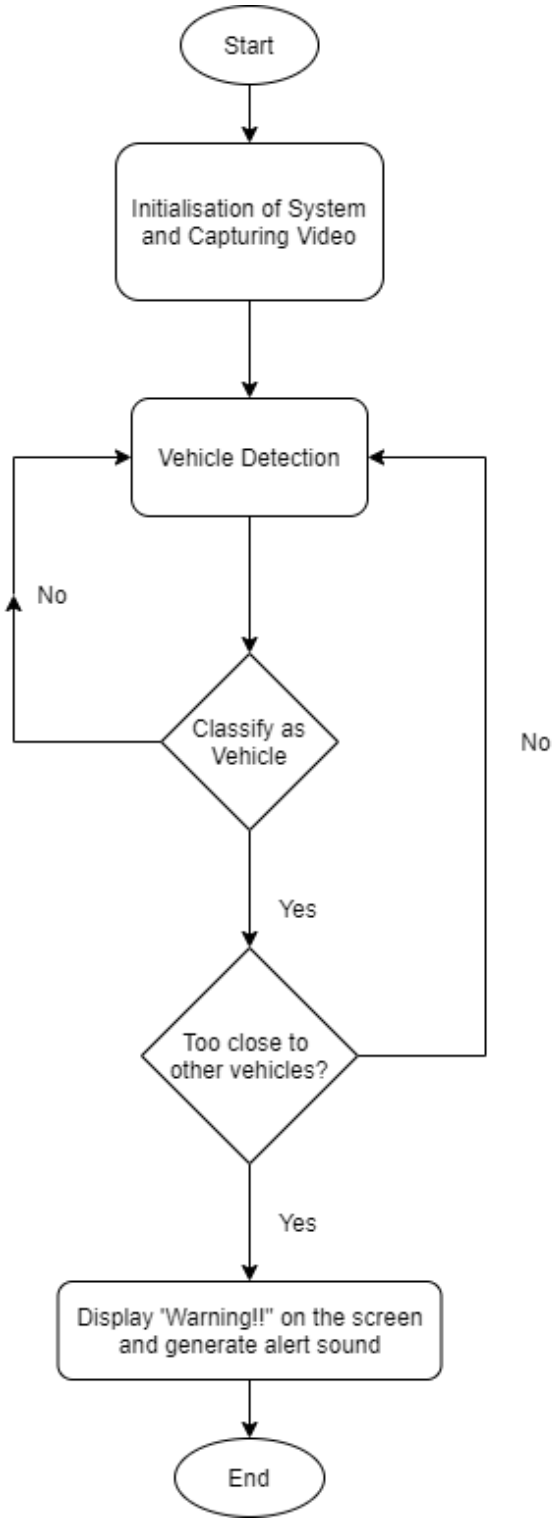


Figure 3.5: Flow chart of pre-collision feature.

3.5 Flow Chart of Web Streaming Feature

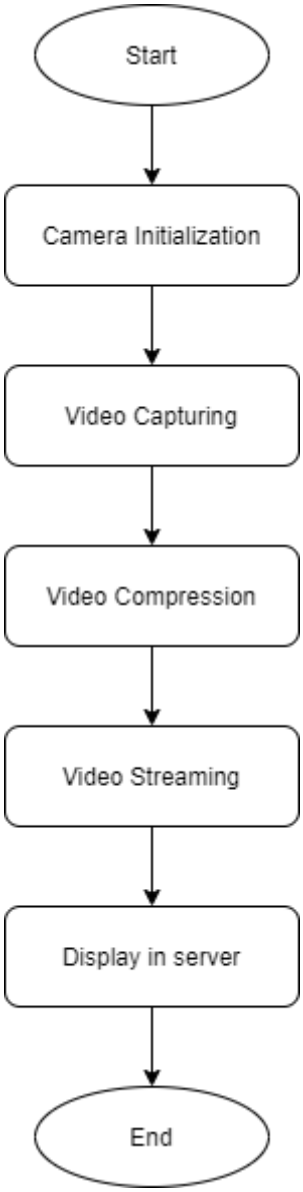


Figure 3.6: Flow chart of web streaming feature.

CHAPTER 4: METHODOLOGY AND TOOLS

4.1 Design Specifications

The methodologies and tools used for the development of this project will be described in this section.

4.1.1 Methodology

The System Development Life Cycle (SDLC) is a systematic process for bringing a proposed project to meet customer expectations and reaches completion. The SDLC has seven distinct phases which are planning, requirements, design, development, testing, deployment, and maintenance. There are many types of SDLC models such as waterfall model, spiral model, prototyping model and so on.

The prototyping model is selected because of the small scale of this project. In the prototyping methodology, the main focus is to develop an early prototyping model of the new system. A system prototype will be developed without full functionality or be completely tested but it will give clients a perception of what's to come for the system. After the prototype is presented, client will provide some feedbacks to the developer. The prototype will be reconstructed due to the user's feedback and suggestion. Finally, a final system is developed based on the finalised prototype as all user requirements are met and user experiences are satisfied with the final prototype. Therefore, prototyping model works best for projects which the requirements are not known.

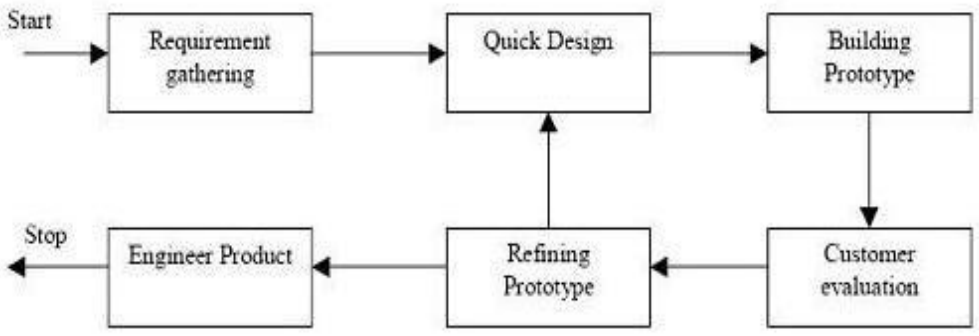


Figure 4.1: Prototyping model.

As this model is used develop this project, any changes or modification can be done easily as this model can get prototype system ready early and test by user for feedback. Developer can make some improvement for the prototype by adding some new features to satisfy user requirement until a prototype is finalized to develop the system.

4.2 Tools to use

This project will be employing three Raspberry Pi and each Raspberry Pi is connected with a 175 degree wide angle camera in order to monitor the car surrounding. A buzzer is implemented to alert the drivers when the own car is too near with other cars. As for software components, TensorFlow, OpenCV, and Python language will be used for this project.

4.2.1 Hardware

Raspberry Pi 3 Model B

Raspberry Pi 3 is the main component of the entire system. It is a low cost mini-computer which helps a lot to digital transformation and signal processing. Besides, it contains standard input/output including HDMI port, USB port, Ethernet port, and audio input/output port. It also has a 15-pin MIPI camera interface which can connect to different camera module for Raspberry Pi. Moreover, it also built in with general purpose input/output (GPIO) which allow the Raspberry Pi to receive input from signal or sensor from another.

Figure 4.2: Raspberry Pi model B.

The specifications of Raspberry Pi 3 Model B are shown below:

- SoC: Broadcom BCM2837
- CPU: 1.2 GHz Quad-Core ARM Cortex-A53
- GPU: Broadcom VideoCore IV @400 MHz64
- RAM: 1 GB SDRAM
- Ports:
    - Micro USB Power Input (5V, can handle up to 2.5A)
    - 4 x USB 2.0 ports
    - 10/100 Ethernet Port
    - CSI Camera Port
    - 3.5mm 4-pole Composite Video and Audio Output Jack
    - Full Size HDMI Video Output
    - DSI Display Port
- Network: 10/100 MBPS Ethernet, 802.11n Wireless LAN, Bluetooth 4.0
- GPIO: 40-pin header (26 GPIOs)
- Storage: Micro-SD

Raspberry Pi Camera Module (Wide Angle Fisheye Webcam)

Raspberry Pi Camera Module is an image sensor module and it is an official product launched by Raspberry Pi Foundation. The camera can be connected to the CSI camera port.

Figure 4.3: Raspberry Pi camera module.

Below are the specifications of the Raspberry Pi Camera Module:

- 2592 x 1944 stills
- 5 megapixels
- OV5647 by Omni Vision
- 175 degrees Field of View (FOV)
- Adjustable focus
- CSI interface with 150mm (15-pin) ribbon cable

16 GB Micro SD

Figure 4.4: Micro SD card.

The Raspberry Pi 3 Model B consists only 1GB of internal storage and it boots only from a SD card. Therefore, microSD card is needed as the storage for operating system and program files. Besides, 16GB storage is enough to store the operating system.

Active Buzzer

Figure 4.5: Active buzzer.

Active buzzer is a kind of audio signaling component to produce sound effect (beeping). It is used to raise an alert when the own vehicle is too near to other vehicle in order to warn the driver about the pre-collision.

4.2.2 Software

Raspbian Operating System

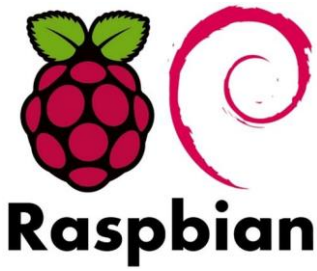


Figure 4.6: Raspbian logo.

Raspbian OS is a free Debian-based operating system for the Raspberry Pi hardware. It consists of a set of basic programs and utilities that can be used by Raspberry Pi for operating purposes. In 2012, it comes with the initial build of over 35,000 Raspbian packages to provide best performance on the Raspberry Pi. It also contains Python, Scratch, Sonic Pi, Java and so on.

TensorFlow

Figure 4.7: TensorFlow logo.

Google TensorFlow is an end-to-end open source platform for numerical computation and large-scale machine learning. It ease the process of collecting data, training models, providing predictions, and refining future results. TensorFlow can be used on many different operating systems including Windows, Linux and Mac OS. Python is used in order to provide a user-friendly API for building application with the framework.

OpenCV

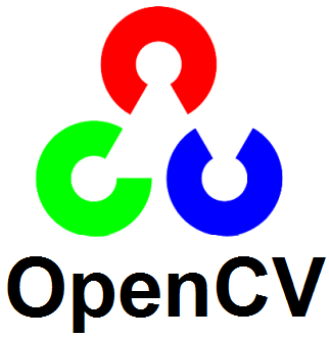


Figure 4.8: OpenCV logo.

Open CV (Open Source Vision Library) is an open-source library that consists of several hundreds of computer vision algorithms. The open-source library is cross-platform and free to use under the open-source BSD license. OpenCV has a modular structure which the packages consist of several shared or static libraries that related to object tracking and image processing that will be used in this system.

4.2.3 Programming Language

Python 3

Figure 4.9: Python logo.

The system will be written in high-level Python programming language as it provides easy syntax for quick coding compared to Java or C++. Besides, Python also consists of several standard libraries which makes the complex functionalities become more easily to be executed.

4.3 Implementation Issue and Challenges

The limited resources of this project caused some implementation issues. As the system is based on Raspberry Pi which has a very less resources compared to a laptop or desktop. Therefore, careful system planning is needed to prevent the lack of resources for the Raspberry Pi.

Besides, there are some difficulties to set up the hardware components. This is because the Raspberry Pi needed to be set on the top of the vehicle which is not easy to do. It also requires an empty field to set up the hardware components and test the prototype of the system. The weather of the environment which is also one of the difficulties as the system cannot be tested if it is rainy day or the weather is too hot that will probably spoilt the hardware components. Therefore, suitable weather condition is considered and shaded environment is chosen for system testing.

Moreover, there are also some difficulties to set up the software environment. This is because the installation of TensorFlow and OpenCV take a lot of time and the installation process is complex. Some instructions and command lines given by some installation guiding websites were outdated which causes the incomplete of TensorFlow and OpenCV installation and failed to use them. Luckily, the latest instructions and command lines were found from the internet and the TensorFlow and OpenCV worked fine after applied the latest installation methods.

4.4 Timeline

Task	Project Week													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Data Collection														
Identify Problem Statements														
Identify Objectives and scope														
Research for literature review														
Determine technologies involved and components used														
Determine system functionalities														
Outline system architecture														
Outline system methodology														
Setup Hardware and Software Environment														
Prototype Testing														
Presentation														
Documentation														

Table 4.1: Gantt chart of FYP1 project timeline.

Task	Project Week													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Finalizing Hardware and Software Environment														
Finalizing system functionalities														
Finalizing system architecture														
Finalizing system methodology														
Finalizing system feature														
Prototype Testing														
Finalizing prototype														
Presentation														
Documentation														

Table 4.2: Gantt chart of FYP2 project timeline.

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1 Design Analysis of Camera Mounting Module

Before start to design the camera mounting module, the vehicle type has to be selected and the length and height of the vehicle has to be defined. Perodua Axia is selected for system testing and analysis purposes. The length of the vehicle is 1.62 meter and the height of the vehicle is 1.51 meter. The camera mounting module will be set on top of the vehicle.

Figure 5.1: Side view of Perodua Axia.

The first camera mounting module is made using a 40cm x 32cm x 29cm carton box and placed on top of the vehicle as diagram:

Figure 5.2: First design of camera mounting module.

By using this design, two pi camera module are mounted on the box, one Pi camera on the front side and one Pi camera on the back side. Besides, some square holes were cut out vertically at the middle of the box to set the Pi camera and the distance between each square hole is 10cm. This action is to determine the suitable height of the camera from the top of the vehicle.

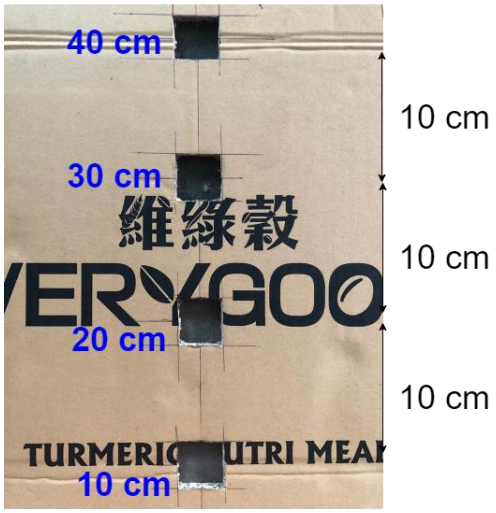
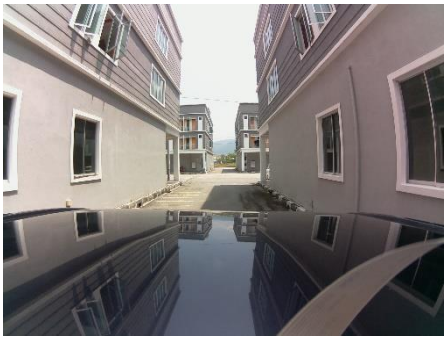


Figure 5.3: Square holes on the carton box.

Below table shows the images captured from different height of the square holes:

	Front	Back
10 cm		
20 cm		

30 cm		
40 cm		

Table 5.1: Comparison of images captured using first camera mounting module.

After comparing the captured images, found that even the highest 40cm height is still not suitable to be used for creating the camera mounting module. Besides, two Pi cameras are not enough to capture the 360 degree surround view of the car. Therefore, 50cm height from the vehicle is picked for camera mounting and three Pi cameras will be used in this system.

The second design of camera mounting module is created based on the analysis results from the first design. A triangular prism shaped module is created and the cross section of the module which is a triangle shape with the length of 29cm for each side. Moreover, a square hole is cut out on 50cm height of the module.

Figure 5.4: Second design of camera mounting module

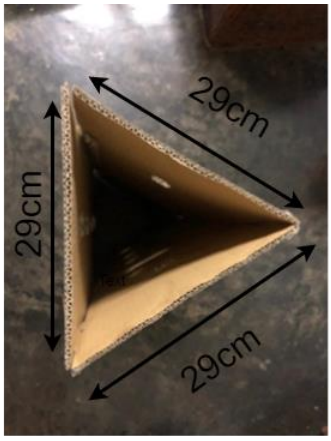


Figure 5.5: Top-down view of second module.

Unfortunately, the second module is still not be able to capture the surround view of the vehicle. This is because there is no overlapping area among the captured images, which means the full 360 degree surround view is not obtained by using this module.

The third design of camera mounting module is created due to the failure of second design. A triangular prism shaped module is created again but with different cross section area which is a triangle with the length of 7cm for each side. A square hole is also cut out on 50cm height of the module.

Figure 5.6: Third design of camera mounting module.

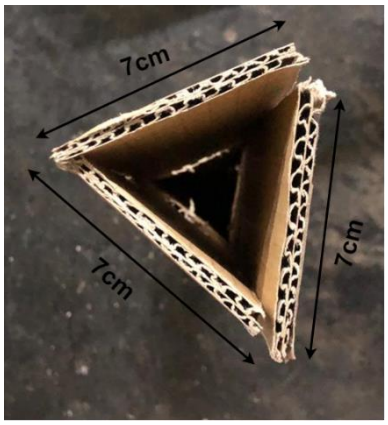


Figure 5.7: Top-down view of third module.

To maintain the stability of the third camera module while capturing the scene, the small box is issued and a hole on the box is being shaped into a triangle, based on the cross section area of the module in order to stick the module into the small box as shown in Figure 5.9:

Figure 5.8: A triangle is cut out from the box.

Figure 5.9: Third module is plugged into a box.

This design allows the vehicle to obtain a 360 view of the vehicle surrounding see Figure 5.10-5.12.

Figure 5.10: Frontal view.

Figure 5.11: SideA view.

Figure 5.12: SideB view.

By observing the captured images, there are some overlapping area among these images. This proves that the surrounding view of the vehicle could be successfully acquired based on the position of these camera in the mounting module. However, the third camera module is not able to withstand the running of the vehicle. Therefore, the final version of camera mounting module is designed as diagrams below:

Figure 5.13: Final version of camera mounting module.

Figure 5.14: A small box is issued inside the camera module to place the equipment for extra protection.

The comparison between the design of camera mounting module is shown as below:

	First version	**Second version**	**Third version**	**Final version**
Number of camera mounted	2	3	3	3
Ability to acquire 360 degree surrounding view	No	Yes	Yes	Yes
Ability to withstand the running vehicle	No	No	No	~~No~~ Yes
Ability to protect hardware components	No	No	No	Yes

Table 5.2: Comparison of camera mounting module design.

5.2 Hardware Setup

5.2.1 Raspberry Pi with Buzzer

The system of this project involves three Raspberry Pi as the main component. Each Raspberry Pi is attached with a buzzer by using jumper wires. The buzzer has 2 pins called positive (+) pin and negative (-) pin and it is connected through Raspberry Pi GPIO pins. The positive pin can be connected to any GPIO pin except the reserved GPIO pin 0 and 1 for advanced use, 5V GPIO pins, 3V3 GPIO pins, and GPIO ground (0V) pins. Besides, the negative pin must be connected to a GPIO ground pin. Below diagram shows the GPIO pinout of Raspberry Pi 3.

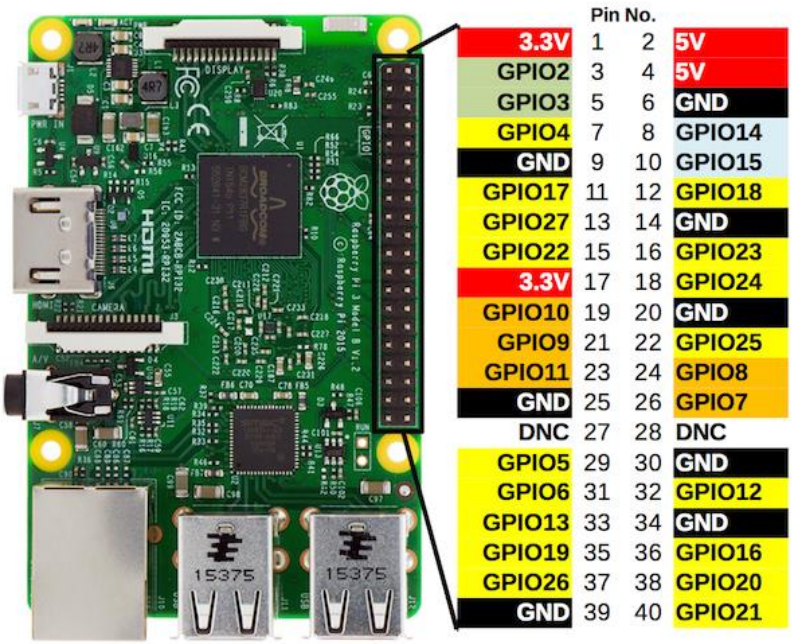


Figure 5.15: GPIO pinout diagram of Raspberry Pi 3 Model B.

As for the wire connection for the buzzer in this project, the positive pin was connected to the GPIO17 (pin no. 11) and the negative pin was connected to the GRD (pin no. 9).

Besides, 9 jumper wires were connected into 1 extended jumper wire by using soldering technique. Each Raspberry Pi needs 2 extended jumper wires to connect with a buzzer and paste the buzzer inside the car. Therefore, total of 6 extended jumper wires were used for this project.

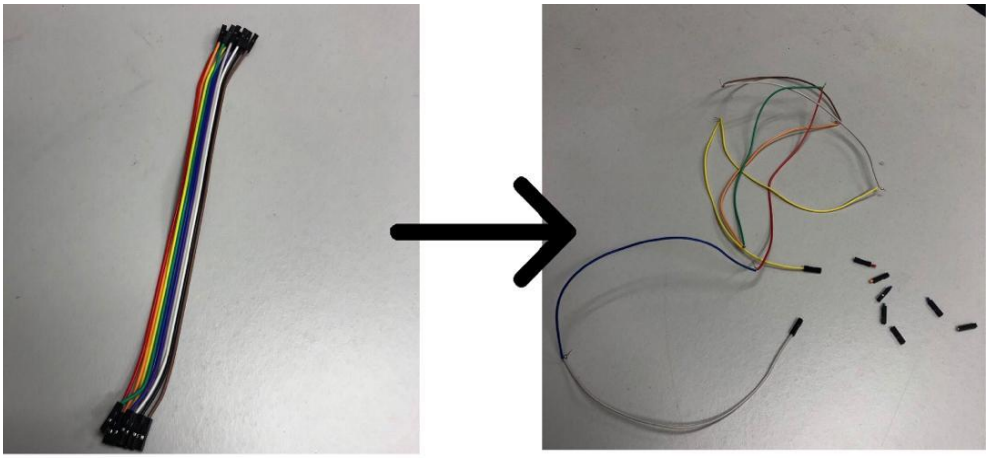


Figure 5.16: 9 jumper wires connected into 1 extended jumper wire.

The actual setup of the Raspberry Pi with buzzer is shown as diagram below.

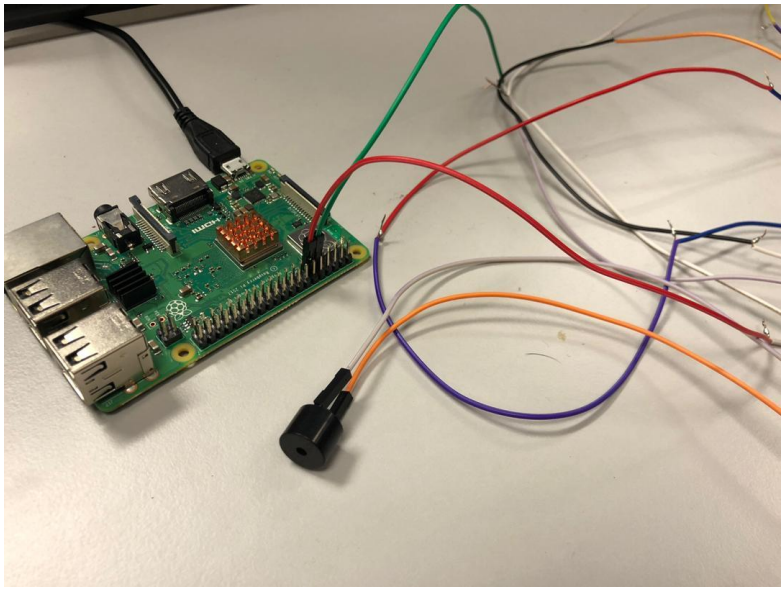


Figure 5.17: Actual setup of Raspberry Pi with buzzer.

5.2.2 Raspberry Pi with Raspberry Pi Wide Angle Camera

In this project, each Raspberry Pi was attached with a 175 degree wide angle camera by using a 1 meter long of 15-pin ribbon cable through the 15-pin CSI camera port. Below diagram shows the actual implementation of wide angle camera module.

Figure 5.18: Actual setup of Raspberry Pi with camera.

5.2.3 Hardware Components with Camera Mounting Module

There are three square holes on the triangular prism part of the camera mounting module. These square holes are used for implementing three wide angle camera to acquire the vehicle surrounding view. Below diagram shows the actual setup of the wide angle camera on the mounting module.

Figure 5.19: Actual setup of wide angle cameras on the mounting module.

Besides, there are a small box inside the camera mounting module to place the Raspberry Pis and the powerbanks. Before putting the Raspberry Pis into the small box, each Raspberry Pi is put into a case for extra protection.

Figure 5.20: Raspberry Pi with casing.

Moreover, some holes were cut on the module for letting the ribbon cable and jumper wires to pass through the holes for making the implementation easier.

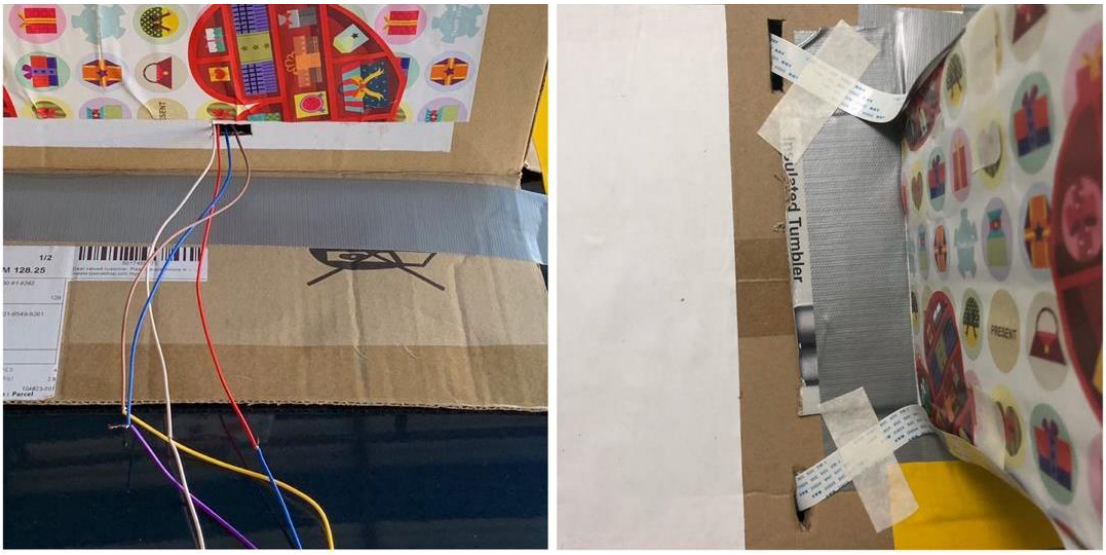


Figure 5.21: Holes for jumper wires (left) and holes for ribbon cables to pass through (right).

After that, the Raspberry Pis and powerbanks were placed into the small box of camera mounting module, as shown in Figure 22.

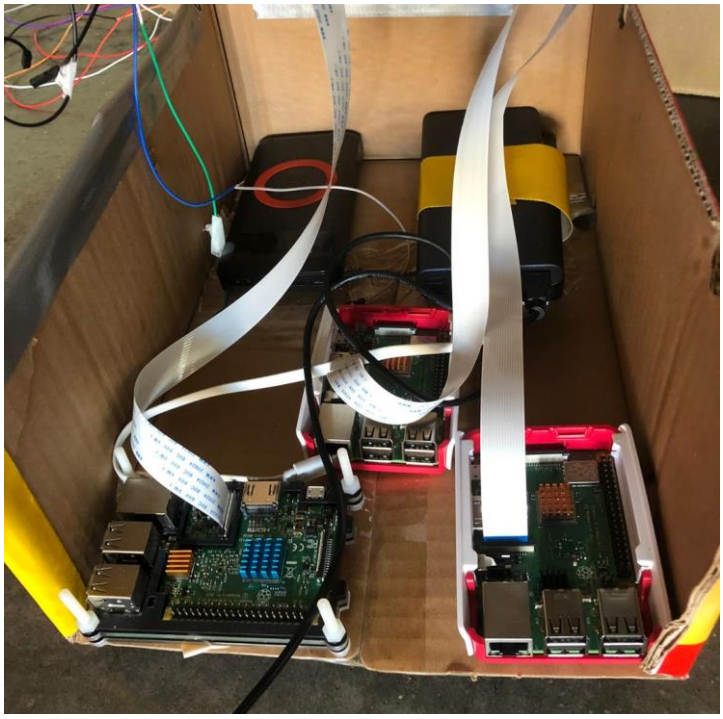


Figure 5.22: Raspberry Pis and powerbanks were placed into the small box.

5.2.4 Camera Mounting Module Setup on Vehicle

The whole set of camera mounting module was set on top the of vehicle. The position of the mounting module was fixed using duct tape. Below diagram shows the actual setup of the camera mounting module on the vehicle.

Figure 5.23: Actual setup of camera mounting module on the vehicle.

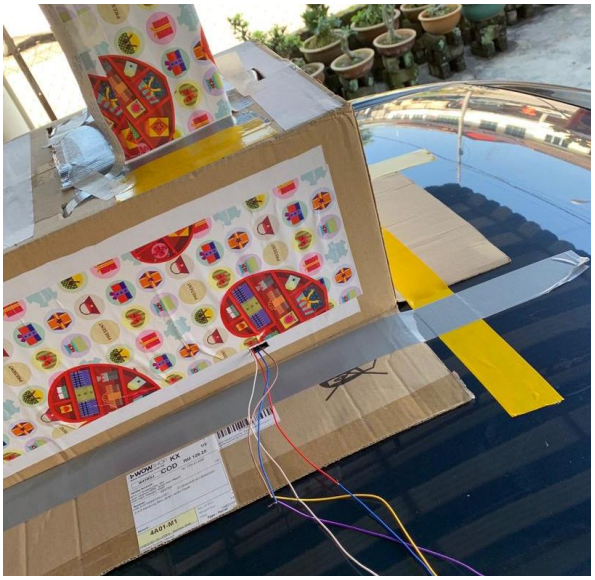


Figure 5.24: Position of the module was fixed using duct tape.

5.2.5 Buzzers Setup inside the Vehicle

The buzzers were connected to the Raspberry Pis using a long soldered jumping wire. Besides, the wires were pass through the holes of the camera mounting module and extended into the vehicle. Three buzzers were pasted respectively on the left, right, and front side of the headliner, also known as the car ceiling – see Figure 5.25.

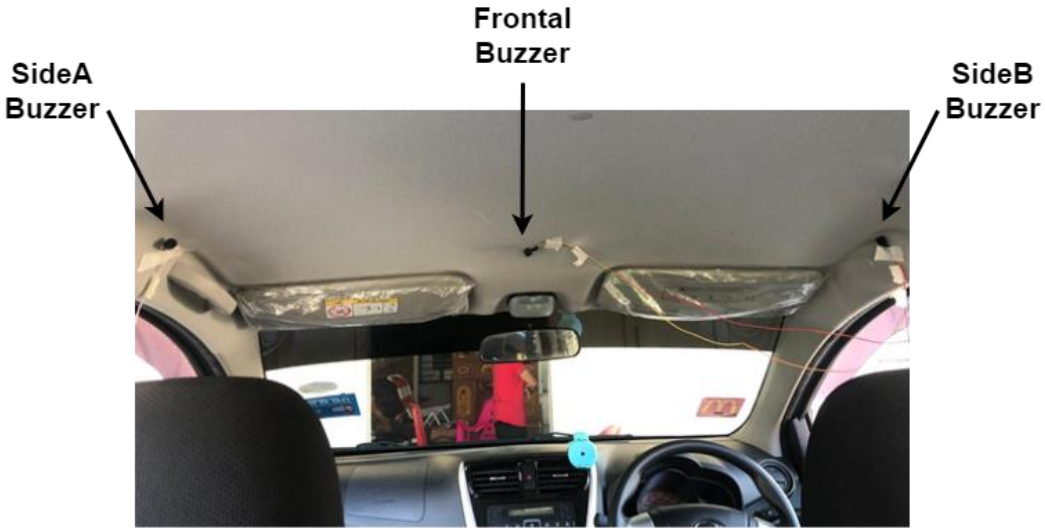


Figure 5.25: Three buzzers were pasted on the car ceiling.

5.2.6 Monitor Raspberry Pis using VNC viewer

During actual system implementation, VNC viewer is used to monitor the Raspberry Pis and run the python in the terminal through two iPad device and one iPhone device. As for iPhone, it is in-charged of monitoring and controlling the Frontal Raspberry Pi. Besides, 2 iPad will be in-charged of monitoring and controlling the SideA and SideB Raspberry Pi.

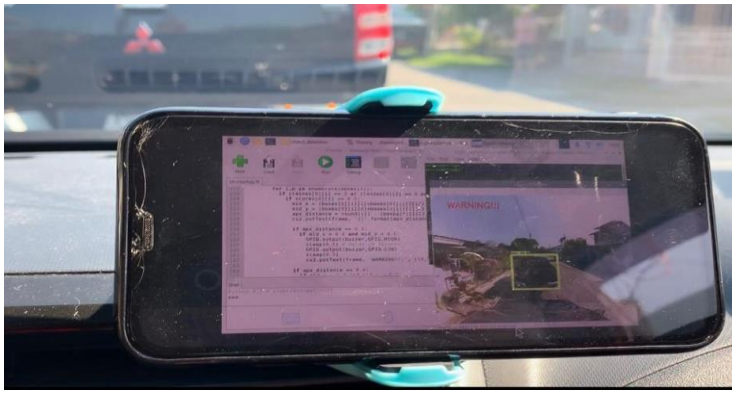


Figure 5.26: iPhone monitors Frontal Raspberry Pi.

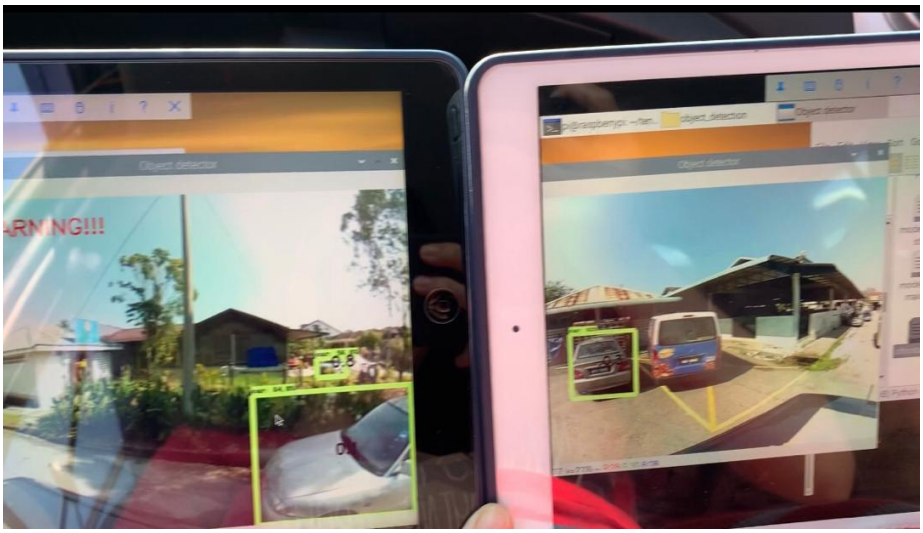


Figure 5.27: One iPad monitors SideA Raspberry Pi (left) and another iPad monitors SideB Raspberry Pi (right).

To control Raspberry Pis using VNC viewer, the VNC server must be enabled through Raspberry Pi Configuration.

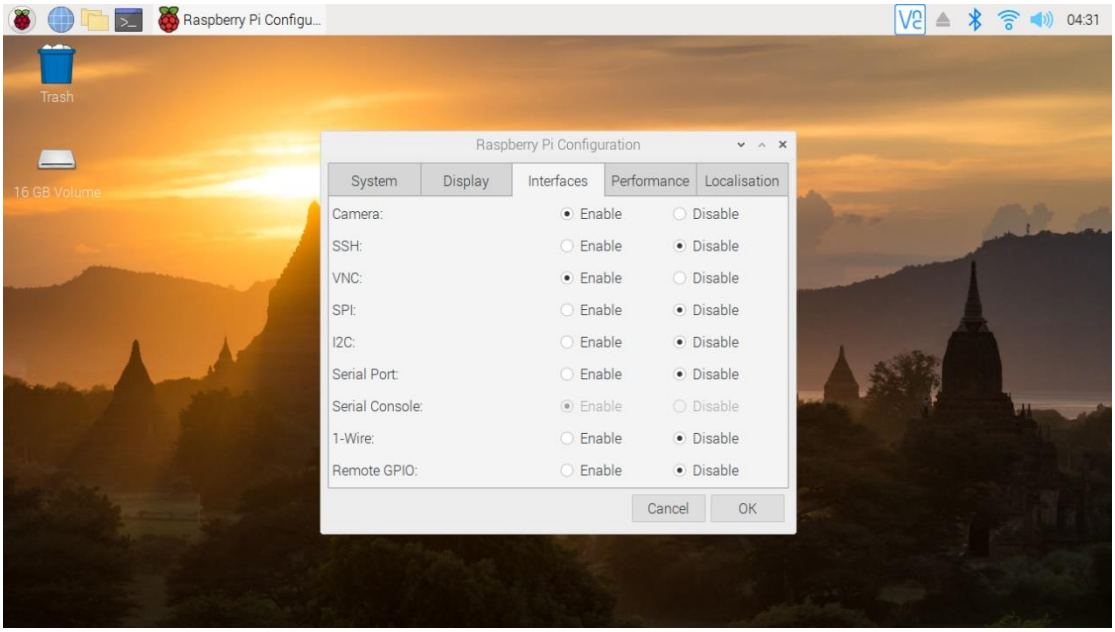


Figure 5.28: Raspberry Pi Configuration.

Next, IP address of the Raspberry Pi is needed in order to connect Raspberry Pi with VNC viewer.

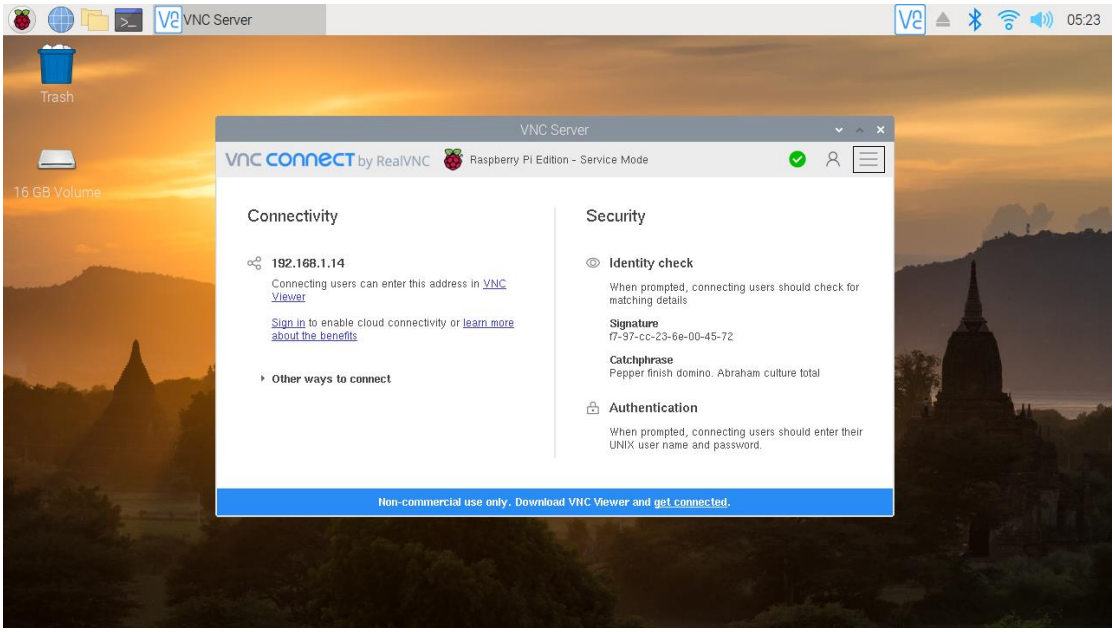


Figure 5.29: IP address of Raspberry Pi.

5.3 Software Setup

5.3.1 Raspbian OS Installation on Raspberry Pi

An operating system is required in order to operate the Raspberry Pi. There are two existing operating system suitable for the Raspberry Pi which is Raspbian and NOOBS and they are available in the website (www.raspberrypi.org). Raspbian OS is selected for this system. The step of installation is stated as below:

1. **Download the Raspbian operating system:** Raspbian is available for download on the Raspberry Pi official website: raspberry.org/downloads which shown as figure 4.9. The zip file is recommended to download compared to torrent file as the user might not familiar with torrenting.

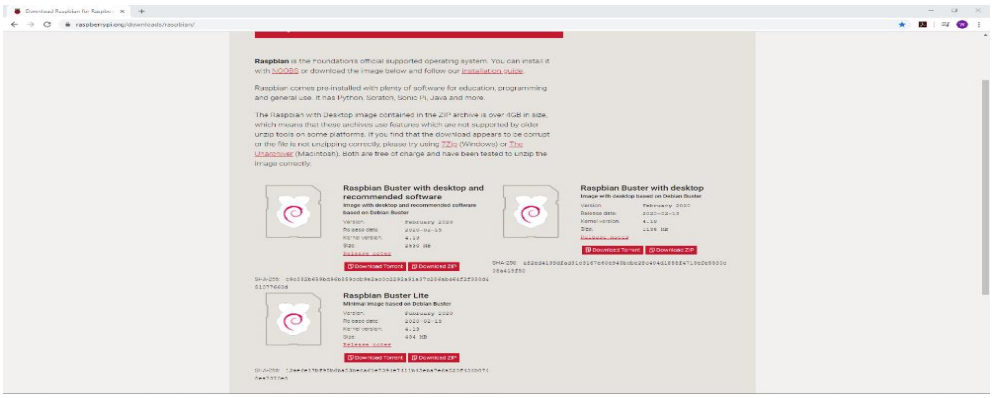


Figure 5.30: Download website of Raspbian.

2. **A formatted Micro SD card required:** SD Formatter 4.0 is used to format the micro SD card.
3. **Burn ISO image into micro SD card:** After download the zip file, the ISO image is extracted and burn into micro SD card by using Rufus 3.5 and a micro SD Card Adapter.

Figure 5.31: Micro SD card adapter is used to burn ISO image.

4. **Booting the Raspberry Pi**: Once the burning process is completed, the micro SD card can now be inserted into Raspberry Pi. After that, the Raspberry Pi will be booted with power supply and able to operate using Raspbian OS.

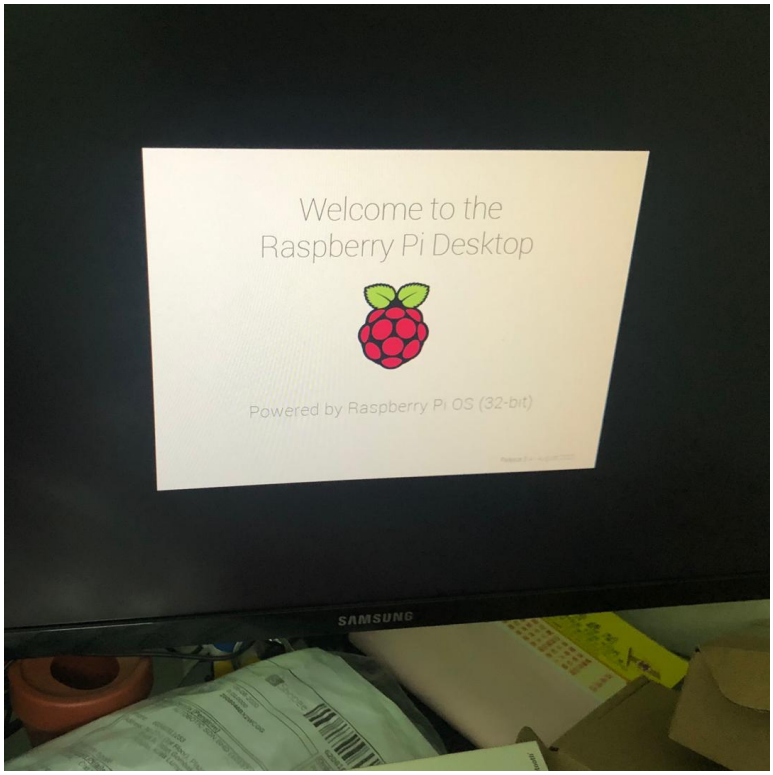


Figure 5.32: Raspberry Pi is successfully booted.

5.3.2 TensorFlow Object Detection API Setup

As the system is required to detect incoming vehicles, TensorFlow is needed to be installed on the Raspberry Pi. There are several steps to set up TensorFlow Object Detection API in Raspberry Pi:

1. **Update and Upgrade Raspberry Pi**: In this step, open terminal in Raspberry Pi and type in *sudo apt-get update* and *sudo apt-get dist-upgrade*

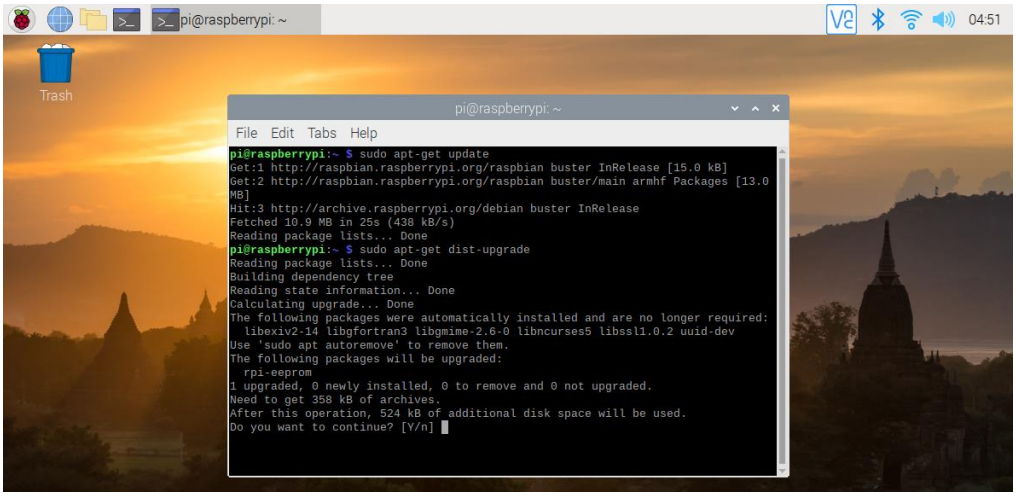


Figure 5.33: Print screen for update and upgrade Raspberry Pi.

2. **Tensorflow Installation**: Rather than downloading and installing the wheel file from the github repository, TensorFlow is now much easier to be installed on the Raspberry pi by typing in *sudo apt-get install libatlas-base-dev* and *pip3 install tensorflow.* After installing TensorFlow, users need to issue *sudo pip3 install pillow lxml jupyter matplotlib cython* and *sudo apt-get install python-tk.*

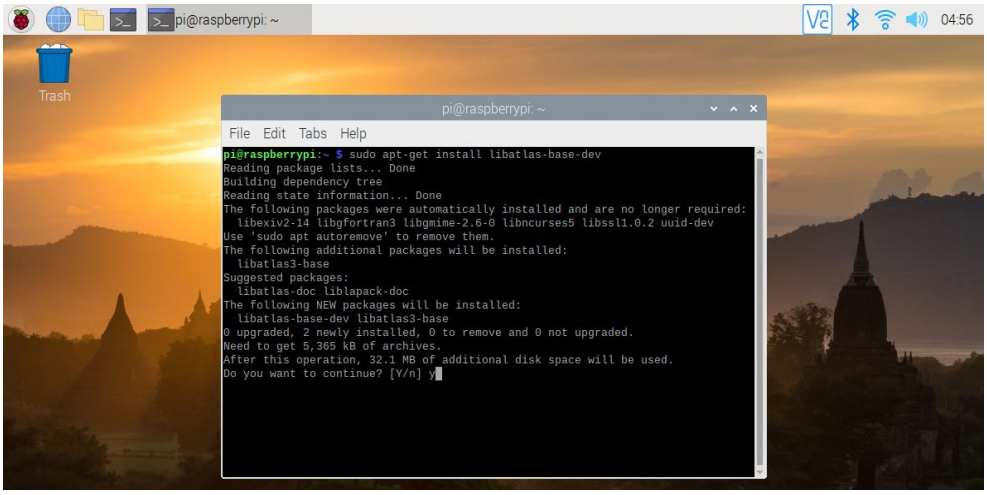


Figure 5.34: Print screen for TensorFlow installation (1).

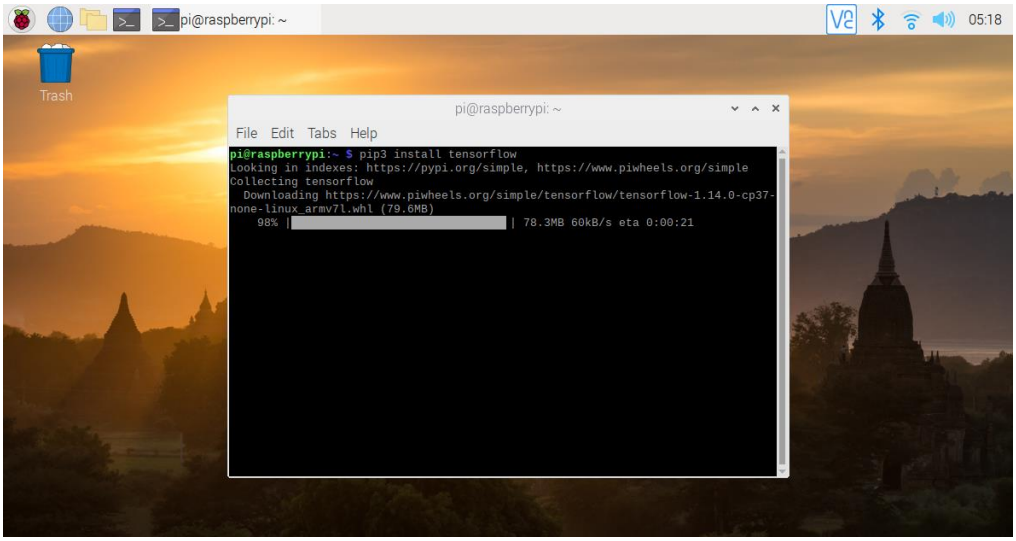


Figure 5.35: Print screen for TensorFlow installation (2).

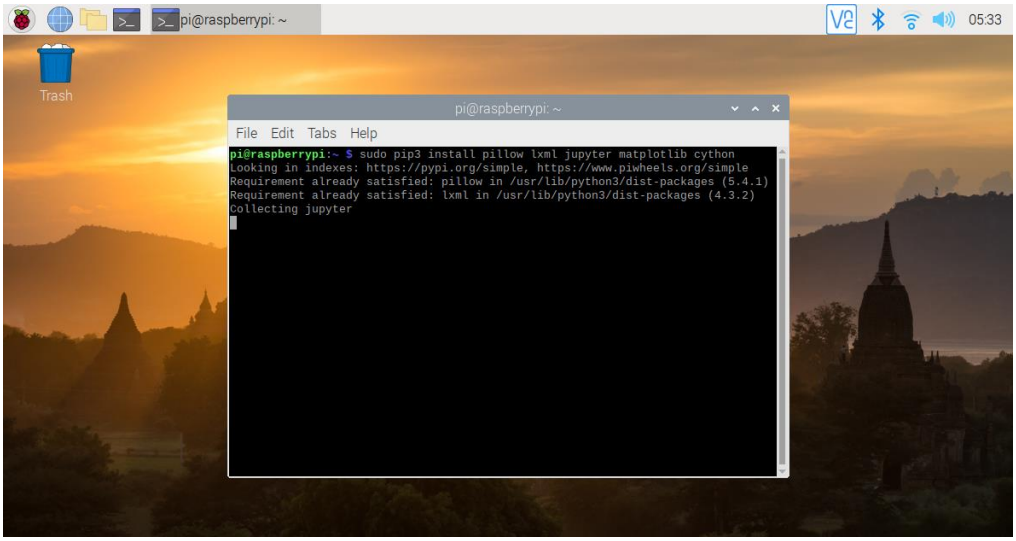


Figure 5.36: Print screen for TensorFlow installation (3).

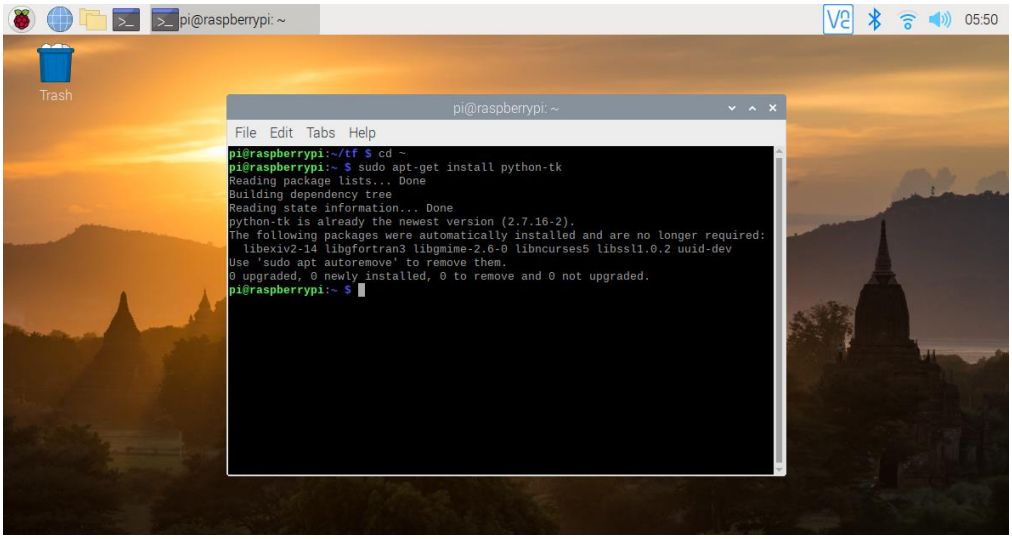


Figure 5.37: Print screen for TensorFlow installation (4).

3. **OpenCV Installation**: The required dependencies and tools are need to be installed before install OpenCV. After installing those dependencies, type in *pip3 install opencv-python.*

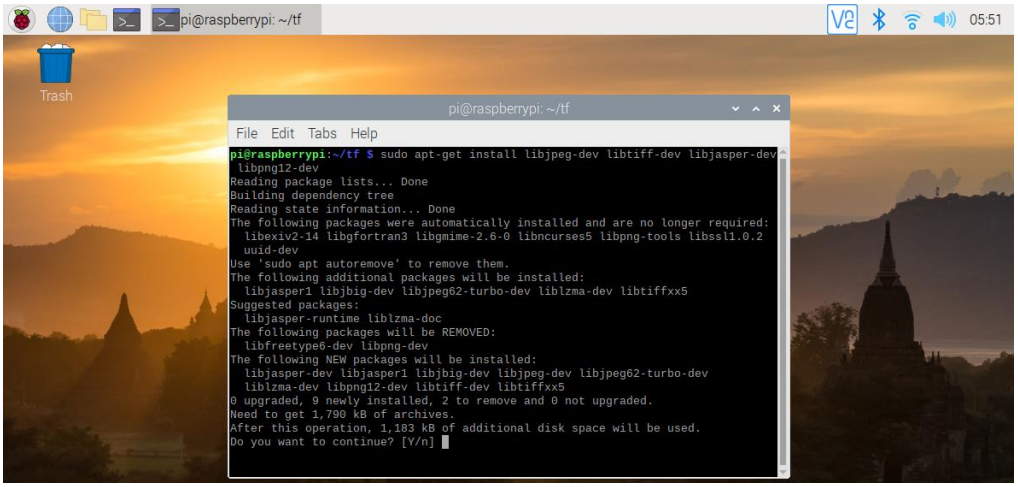


Figure 5.38: Print screen for OpenCV installation (1).

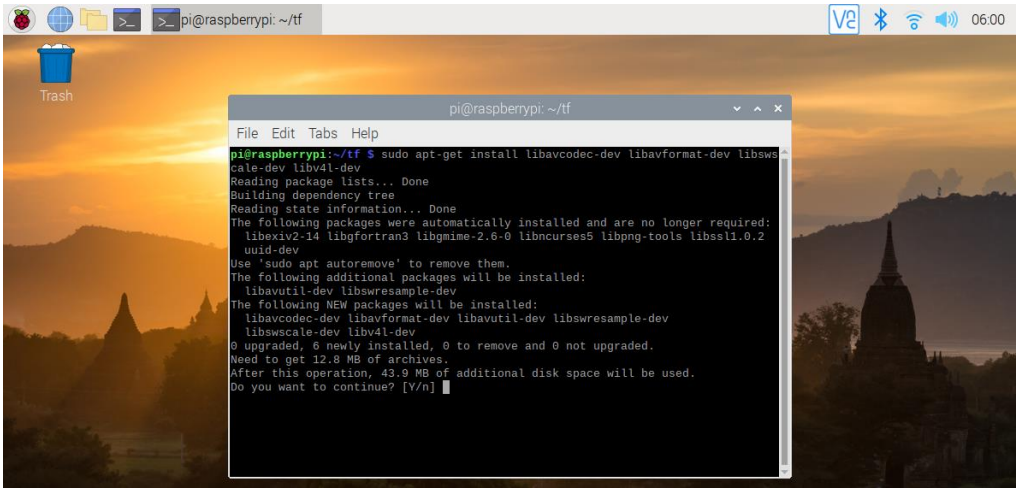


Figure 5.39: Print screen for OpenCV installation (2).

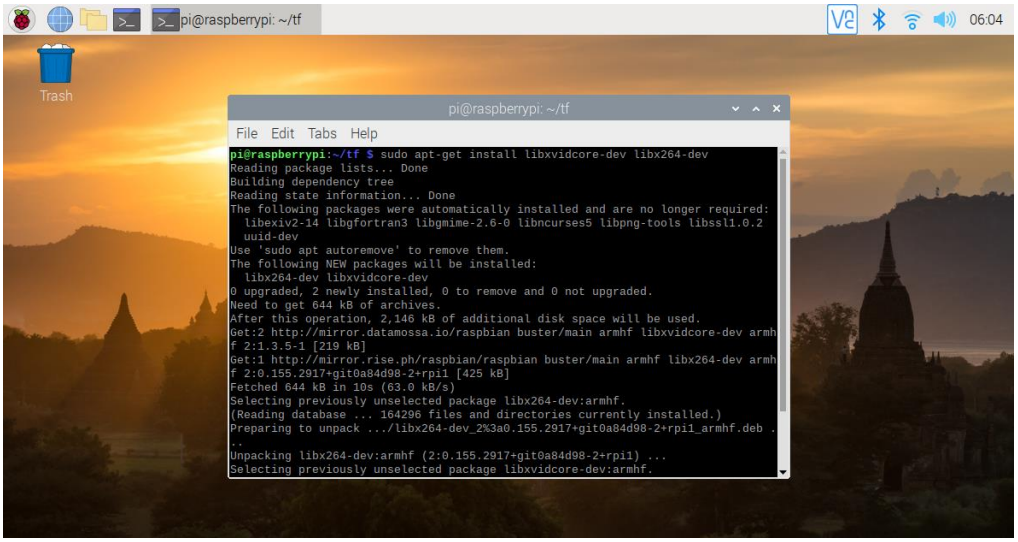


Figure 5.40: Print screen for OpenCV installation (3).

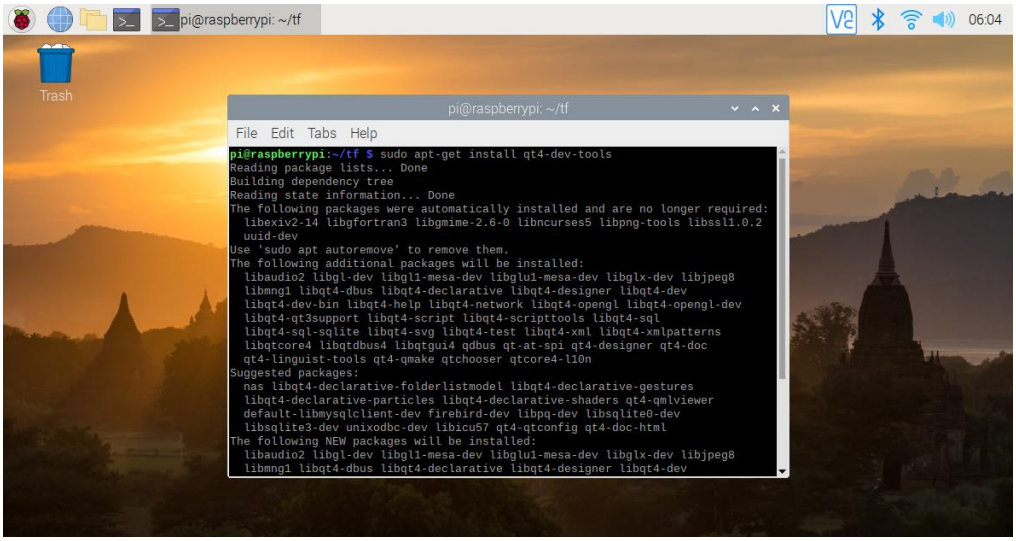


Figure 5.41: Print screen for OpenCV installation (4).

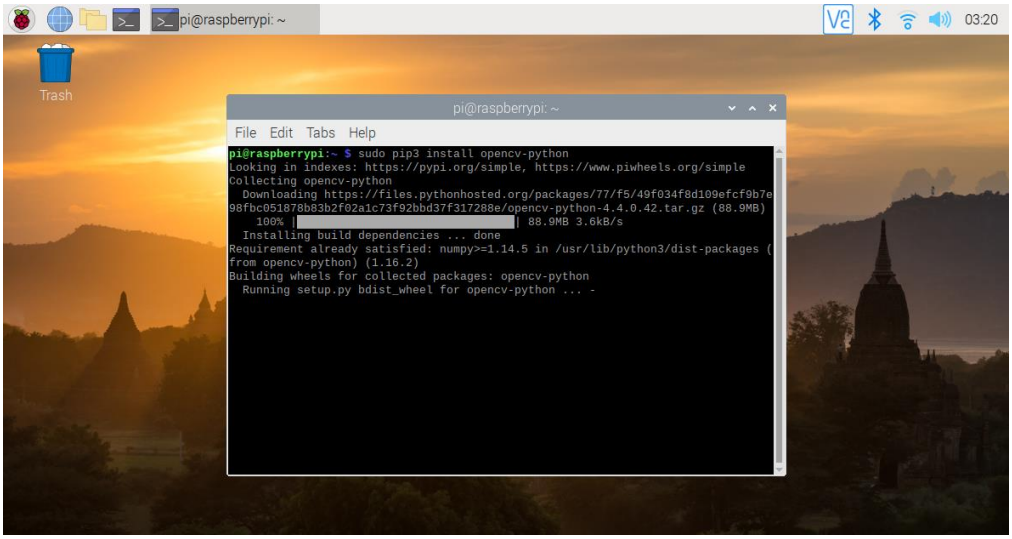


Figure 5.42: Print screen for OpenCV installation (5).

4. **Protobuf Compilation and Installation**: In this steps, simply type in *sudo apt-get install protobuf-compiler*

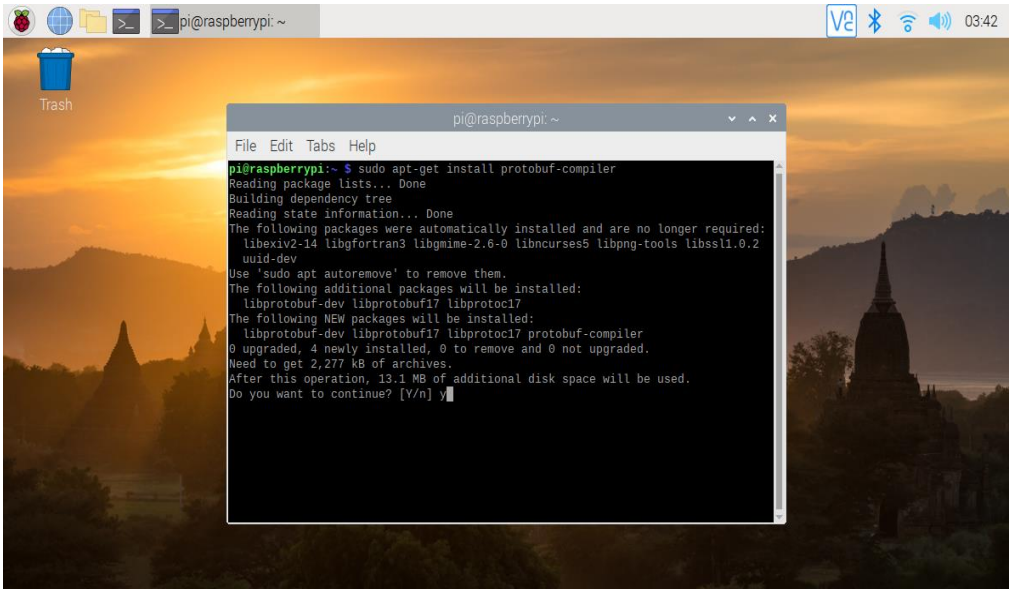


Figure 5.43: Print screen for Protobuf Compilation and Installation.

5. **TensorFlow Directory Structure Setup**: A file is created to get the tensorflow repository from the github by typing *git clone –recurse-submodules https://github.com/tensorflow/models.git*. Next, the Python path environment variable is needed to be modified to point at some directories inside the downloaded TensorFlow repository as the path is needed to be set every time when opening the terminal. */bashrc* file is modified and at end of the file and on the last line, add *export PYTHONPATH=$PYTHONPATH:/.* Furthermore, pre-trained object detection model *ssd_mobilenet_v2_coco* is downloaded from the github for classify the detected object.

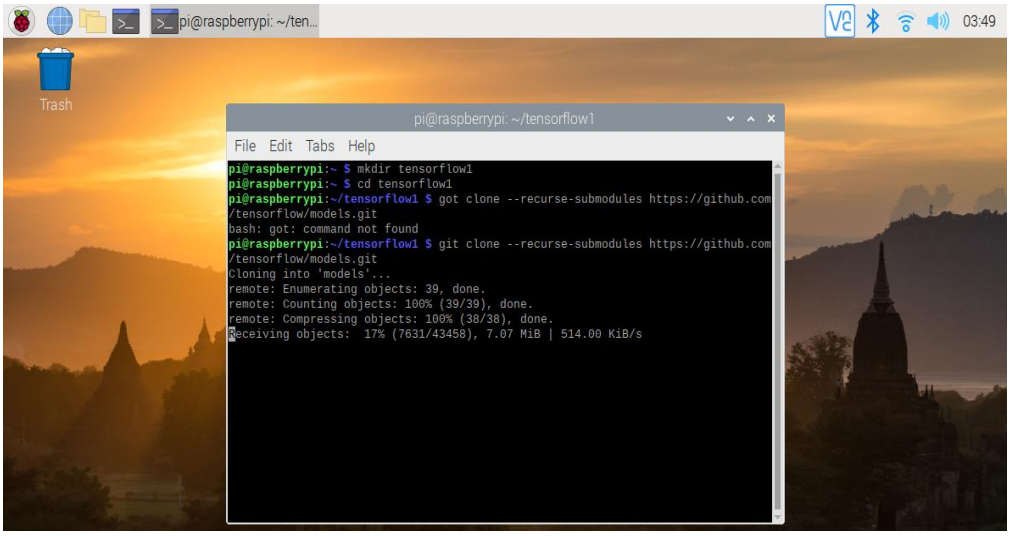


Figure 5.44: Print screen for TensorFlow directory structure setup (1).

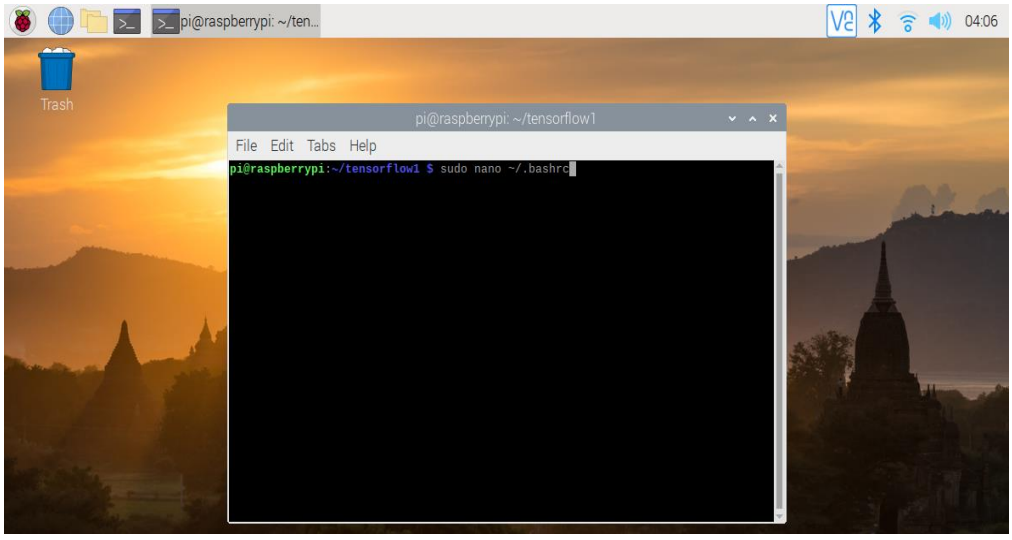


Figure 5.45: Print screen for TensorFlow directory structure setup (2).

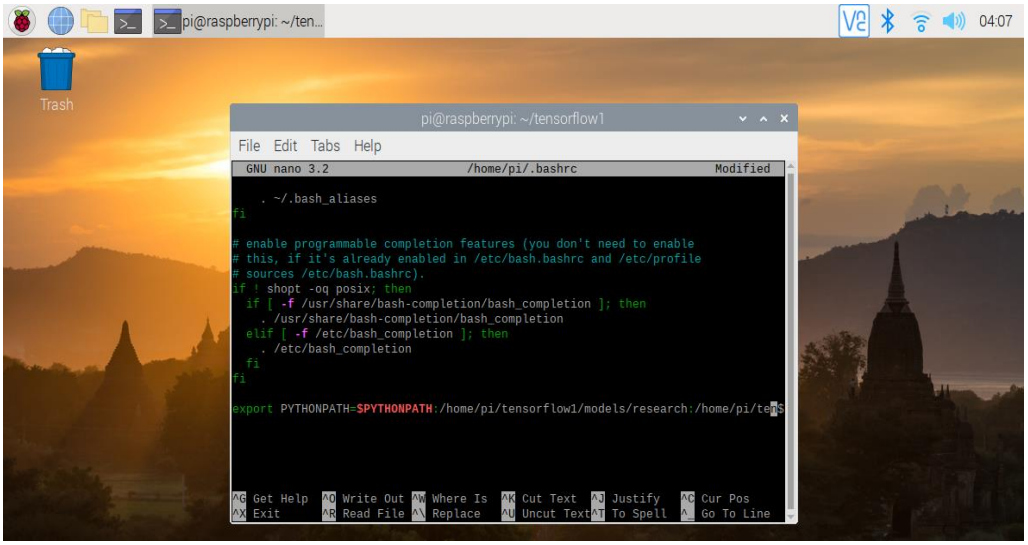


Figure 5.46: Print screen for TensorFlow directory structure setup (3).

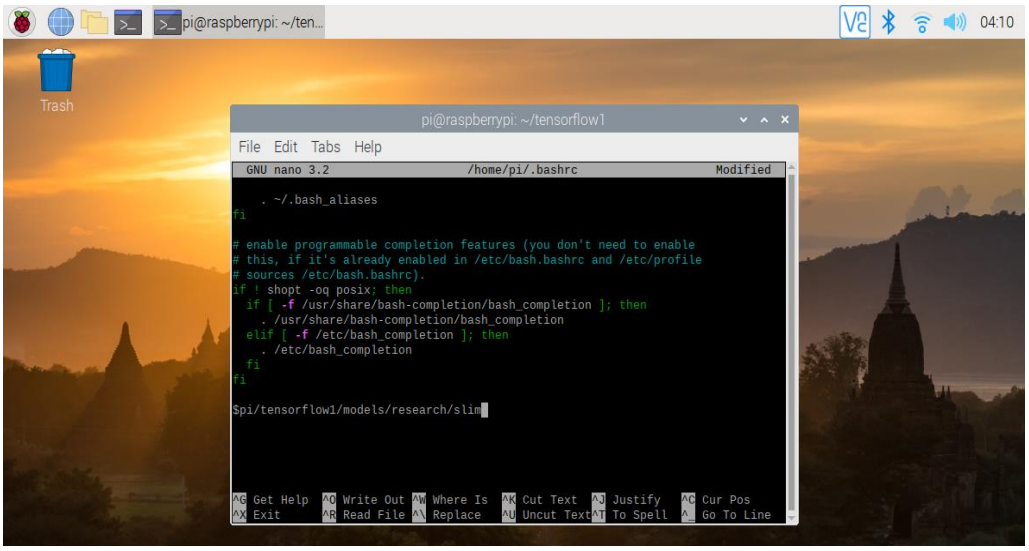


Figure 5.47: Print screen for TensorFlow directory structure setup (4).

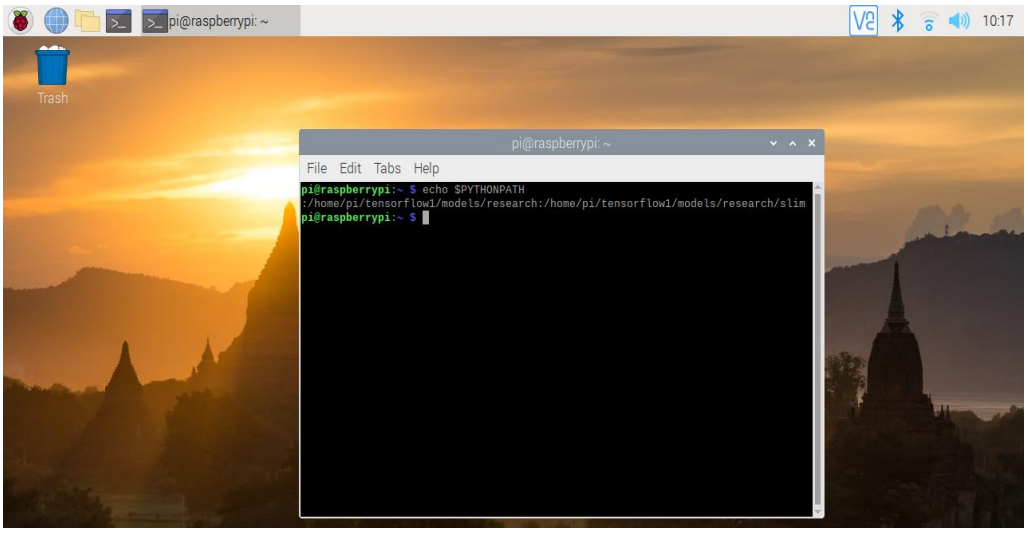


Figure 5.48: Print screen for TensorFlow directory structure setup (5).

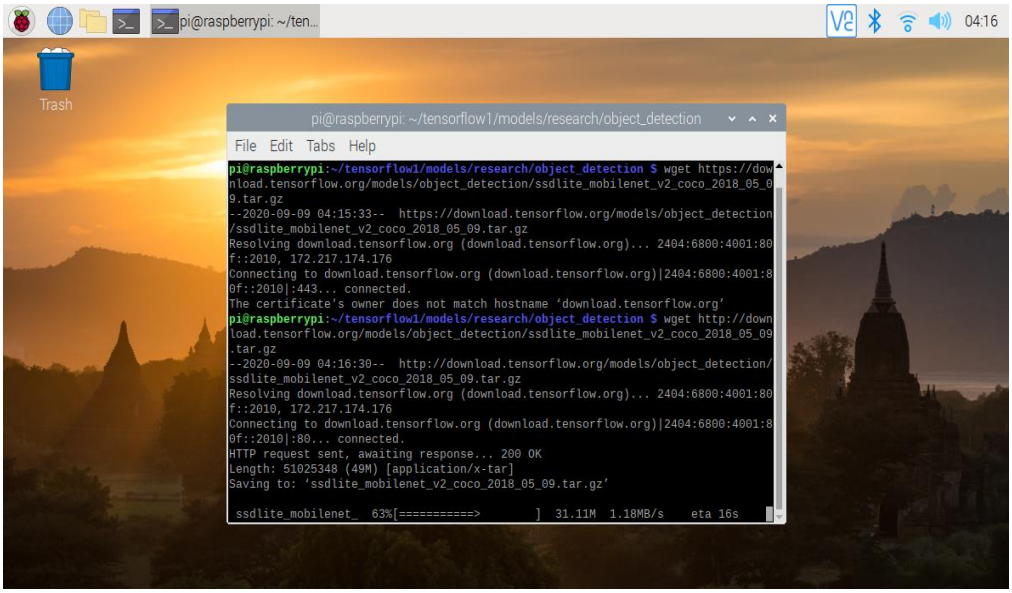


Figure 5.49: Print screen for TensorFlow directory structure setup (6).

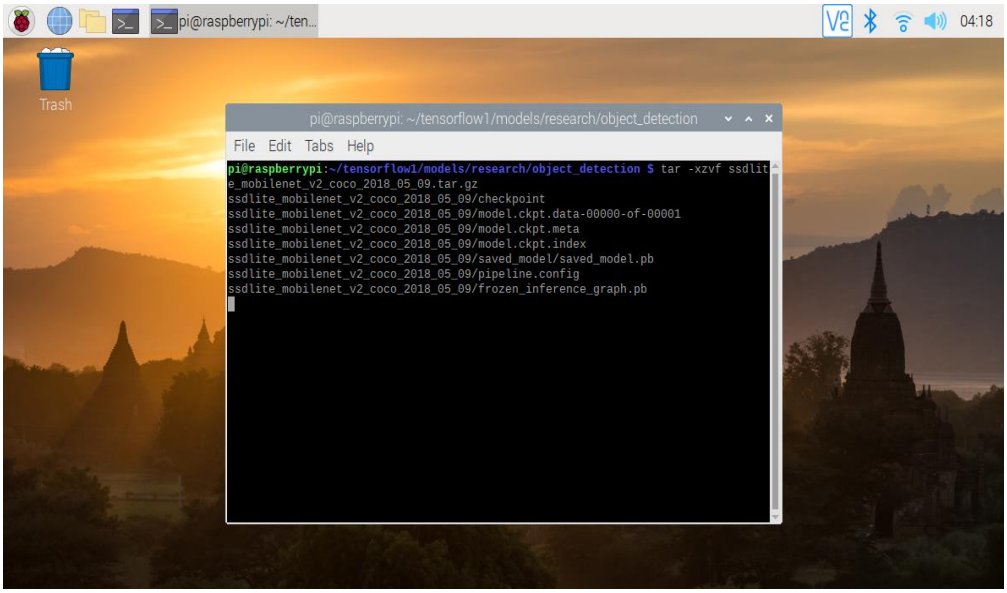


Figure 5.50: Print screen for TensorFlow directory structure setup (7).

To perform object detection in live feeds from Pi camera or USB webcam, a python code *objectdetection.py* is written.

```
import os
import cv2
import numpy as np
from picamera.array import PiRGBArray
from picamera import PiCamera
import tensorflow as tf
import argparse
import sys

# Set up camera constants
IM_WIDTH = 1280
IM_HEIGHT = 720
#IM_WIDTH = 640    Use smaller resolution for
#IM_HEIGHT = 480   slightly faster framerate

# Select camera type (if user enters --usbcam when calling this script,
# a USB webcam will be used)
camera_type = 'picamera'
parser = argparse.ArgumentParser()
parser.add_argument('--usbcam', help='Use a USB webcam instead of picamera',
                    action='store_true')
args = parser.parse_args()
if args.usbcam:
    camera_type = 'usb'

# This is needed since the working directory is the object_detection folder.
sys.path.append('..')

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'ssdlite_mobilenet_v2_coco_2018_05_09'

# Grab path to current working directory
CWD_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')
```

Figure 5.51: *objectdetection.py* hosted with EdjeElectronics by GitHub (1).

```
# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'data','mscoco_label_map.pbtxt')

# Number of classes the object detector can identify
NUM_CLASSES = 90

## Load the label map.
# Label maps map indices to category names, so that when the convolution
# network predicts `5`, we know that this corresponds to `airplane`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.Session(graph=detection_graph)


# Define input and output tensors (i.e. data) for the object detection classifier

# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')

# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()
font = cv2.FONT_HERSHEY_SIMPLEX
```

Figure 5.52: *objectdetection.py* hosted with EdjeElectronics by GitHub (2).

```
### Picamera ###
if camera_type == 'picamera':
    # Initialize Picamera and grab reference to the raw capture
    camera = PiCamera()
    camera.resolution = (IM_WIDTH,IM_HEIGHT)
    camera.framerate = 10
    rawCapture = PiRGBArray(camera, size=(IM_WIDTH,IM_HEIGHT))
    rawCapture.truncate(0)

    for frame1 in camera.capture_continuous(rawCapture, format="bgr",use_video_port=True):

        t1 = cv2.getTickCount()

        # Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]
        # i.e. a single-column array, where each item in the column has the pixel RGB value
        frame = np.copy(frame1.array)
        frame.setflags(write=1)
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame_expanded = np.expand_dims(frame_rgb, axis=0)

        # Perform the actual detection by running the model with the image as input
        (boxes, scores, classes, num) = sess.run(
            [detection_boxes, detection_scores, detection_classes, num_detections],
            feed_dict={image_tensor: frame_expanded})

        # Draw the results of the detection (aka 'visulaize the results')
        vis_util.visualize_boxes_and_labels_on_image_array(
            frame,
            np.squeeze(boxes),
            np.squeeze(classes).astype(np.int32),
            np.squeeze(scores),
            category_index,
            use_normalized_coordinates=True,
            line_thickness=8,
            min_score_thresh=0.40)

        cv2.putText(frame,"FPS: {0:.2f}".format(frame_rate_calc),(30,50),font,1,(255,255,0),2,cv2.LINE_AA)

        # All the results have been drawn on the frame, so it's time to display it.
        cv2.imshow('Object detector', frame)

        t2 = cv2.getTickCount()
        time1 = (t2-t1)/freq
        frame_rate_calc = 1/time1

        # Press 'q' to quit
        if cv2.waitKey(1) == ord('q'):
            break

        rawCapture.truncate(0)

    camera.close()
```

Figure 5.53: *objectdetection.py* hosted with EdjeElectronics by GitHub (3).

The *objectdetection.py* was executed in the terminal. The objects in the live feeds were detected successfully which shown as below.

Figure 5.54: Object detection using Raspberry Pi.

Based on the concept of object detection, the detected objects' features were extracted from the input frames by the algorithm and classify them based on those detected features through MatLab or Open CV (Keshari, 2020). Below diagram shows the basic workflow of object detection.

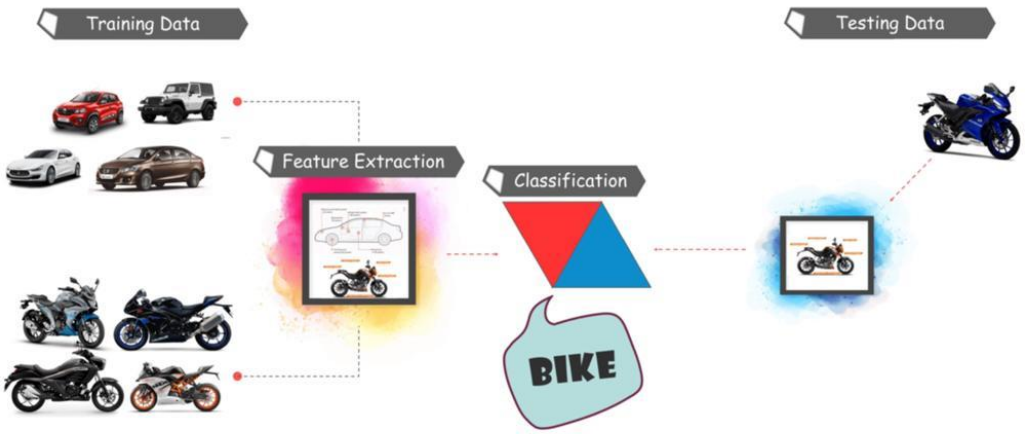


Figure 5.55: Basic workflow of object detection.

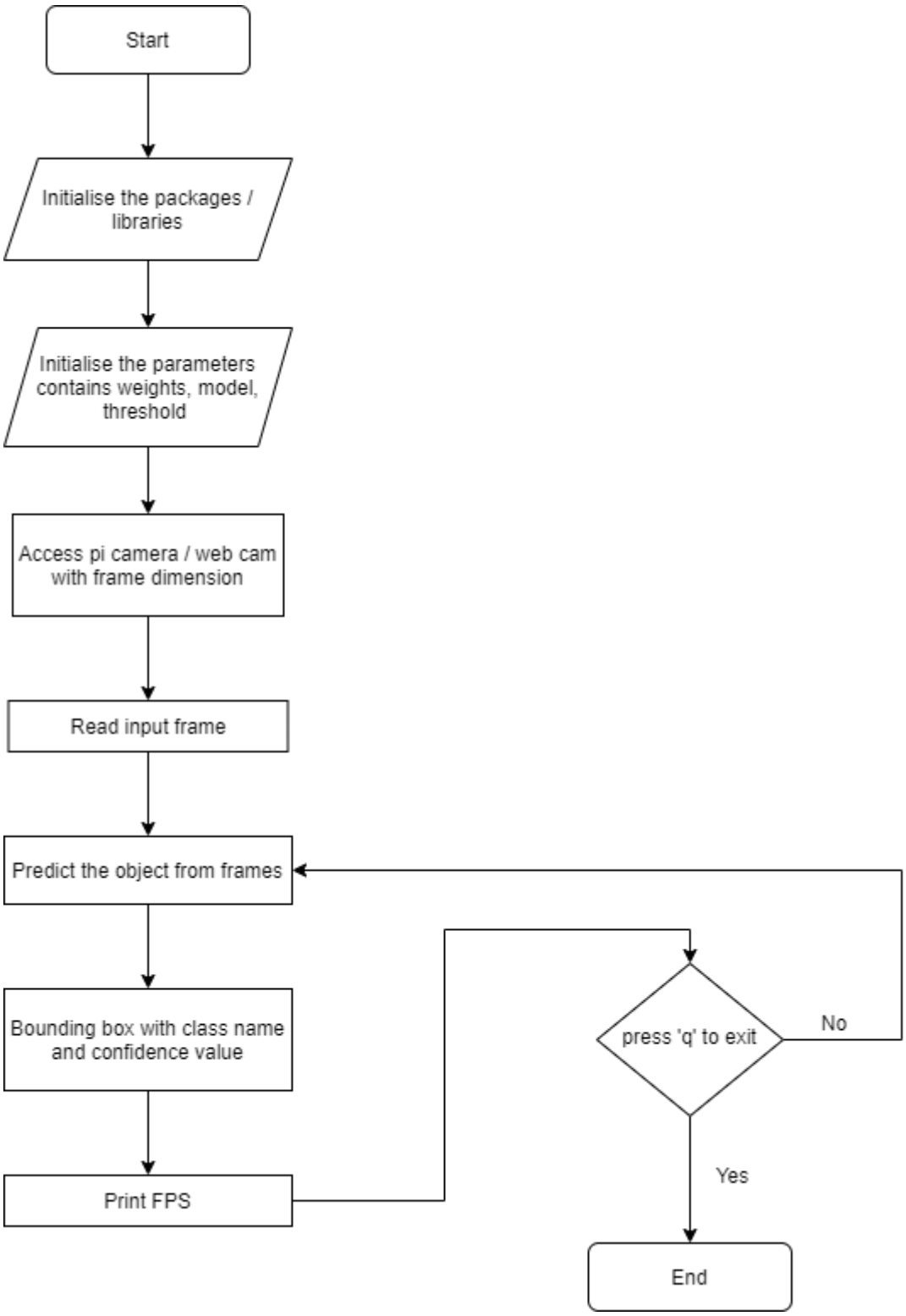


Figure 5.56: Flowchart of *objectdetection.py*

5.4 Pre-collision Feature Setup

5.4.1 Using TensorFlow Object Detection API

TensorFlow Object Detection Application Programming Interface (API) is used for this project as the camera sensor is the main component of the driving monitoring system. The developers are now can save time and no need to write code from the scratch. This is because an API creates a set of common operations for them.

In this proposed system, TensorFlow Object Detection API is used to detect the objects in real-time video along with the object description by using bounding boxes. Besides, the API is pre-trained using a collection of Common Objects in Context (COCO) dataset which consist of 300 different images for 90 different objects. There are some existing pre-trained models which are described in Table 5.3.

Model name	Speed	COCO mean average precision (mAP)	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	31	24	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
fastet_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes

Table 5.3: Existing pre-trained object detection model.

Due to the Raspberry Pi contains a weak and slow processor, a model that use less processing power is needed for this project. The model ‘ssdlite_mobilenet_v2_coco’ is selected due to its efficiency. Although this model will run faster, it still contains a lower detection accuracy compared to other pre-trained models.

5.4.2 Concept of Bounding Boxes

In object detection, a bounding box will be outlined around a detected object in the image and displayed it on the screen. The bounding box consists of the information about the detected object. It also consists of the coordinates of the detected object which able to locate the object in the image. Therefore, the object detection also can be known as the combination of object classification and object localization. TensorFlow Object Detection API is used as this system requires both object classification and localization to detect the vehicles and pedestrians on the road.

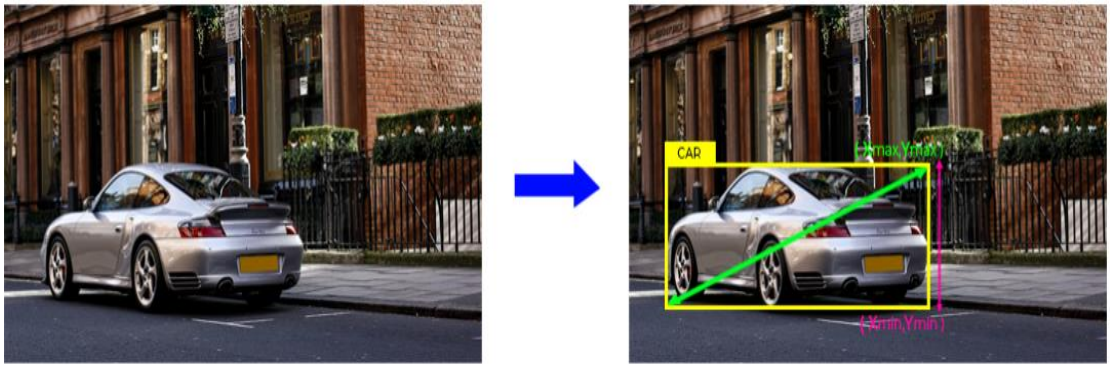


Figure 5.57: Coordinates of bounding box.

5.4.3 Forward Collision Warning Feature for Frontal Raspberry Pi

In the frontal Raspberry Pi 3 set, the forward collision warning (FCW) features is deployed. A pre-trained TensorFlow model is used to detect incoming vehicles. Only the vehicle classes are selected to apply the approximate distance algorithm. Besides, variable *scores* represented the score of an object that detected by TensorFlow model. For example, if a car is detected, it should get a score that above 0.5. Then, the python code, written for the FCW feature, is later deployed.

```
for i,b in enumerate(boxes[0]):
#                       car                   bus                   truck
  if classes[0][i] == 3 or classes[0][i] == 6 or classes[0][i] == 8:
    if scores[0][i] >= 0.8:
```

Figure 5.58: Vehicle detection logic.

The *scores* is set to above or equal to 0.8, after rigorous experiments, because sometimes the half top-down view of the vehicle are being detected as an actual vehicle as shown in Figure 17.

Figure 5.59: Half top-down view of Axia is detected as a vehicle with scores of 0.65.

As the bounding box will be drawn around the detected object in the live feed, the bounding box will become bigger as the object moves nearer to the camera and become smaller when the object moves further. Thus, a distance algorithm, named as *approximate distance algorithm logic* to estimate the distance between the vehicles.

```
mid_x = (boxes[0][i][1]+boxes[0][i][3])/2
mid_y = (boxes[0][i][0]+boxes[0][i][2])/2
apx_distance = round(((1 - (boxes[0][i][3] - boxes[0][i][1]))**4),1)
cv2.putText(frame, '{}'.format(apx_distance), (int(mid_x*640),int(mid_y*480)),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,0), 2)
```

Figure 5.60: Approximate distance algorithm logic for FCW feature.

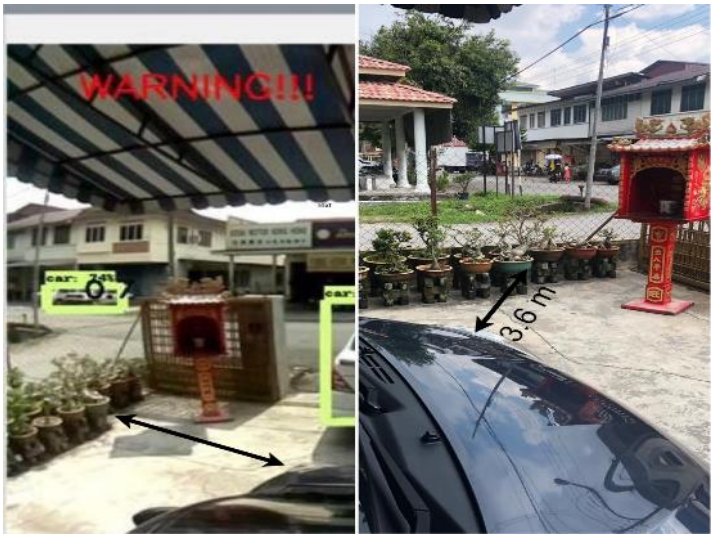


Figure 5.61: Difference between distance in acquire image (left) and actual distance (right) of the green flowerpot**.**

Three vehicle classes are selected which is car, bus, and truck based on the approximate distance algorithm . The bounding boxes in this model return the *x-min, x-max, y-min, y-max* coordinates. *x* coordinate will be used for this algorithm. When a vehicle is detected, a bounding box will be outlined around the detected vehicle. The width of the bounding box (difference between *x-max* and *x-min*) is used to be calculated as the width of a vehicle is easy to be differentiated from the video acquired by the frontal camera. Note: The *apx_distance* return here, is in percentage and it is not the actual distance.

Figure 5.62: The details about bounding box.

In the approximate distance algorithm, *boxes[0][i][3*] represent the *x-max* and *boxes[0][i][1]* represent the *x-min*. The result of *x-max* minus *x-min* will become larger in value as the object moves closer and unpredictable. So, 1 minus the difference of *x-max* and *x-min* will result in a smaller value as the object moves closer. The results to the power of 4 will produce a value more granular and the value is then rounded. The midpoint of *x* and *y* are clarified by adding the *max* and *min* value and then be divided by 2. The midpoints of *x* are then used to determine whether or not the vehicles are shown in a line of sight.

For the pre-collision alert function, if the approximate distance value is equal to 0.5 and the vehicle is in a line of sight, the frontal buzzer will be triggered and generated a low frequency of beeping sound.

```
if apx_distance == 0.5:
  if mid_x > 0.4 and mid_x < 0.6:
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.5) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.5)
    cv2.putText(frame, 'WARNING!!!', (50,50),
    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,0,255), 2)
```

Figure 5.63: Part of python code for buzzer to produce low frequency of beeping sound.

If the approximate distance value is equal to 0.4 and the vehicle is in a line of sight, the frontal buzzer will be triggered and generated a medium frequency of beeping sound.

```
if apx_distance == 0.4:
  if mid_x > 0.4 and mid_x < 0.6:
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.4) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.4)
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.4) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.4)
    cv2.putText(frame, 'WARNING!!!', (50,50),
    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,0,255), 2)
```

Figure 5.64: Part of python code for buzzer to produce medium frequency of beeping sound.

If the approximate distance value is less than or equal to 0.3 and the vehicle is in a line of sight, the frontal buzzer will be triggered and generated a high frequency of beeping sound.

```
if apx_distance <= 0.3:
  if mid_x > 0.4 and mid_x < 0.6:
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.3) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.3)
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.3) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.3)
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.3) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.3)
    cv2.putText(frame, 'WARNING!!!', (50,50),
    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,0,255), 2)
```

Figure 5.65: Part of python code for buzzer to produce high frequency of beeping sound.

Besides, a “Warning!” text will be displayed on the screen using the cv.putText method.

The real distances of the approximate distance 0.5 is about 2.0m from Axia and the real distances of the approximate distance 0.4 is about 1.4m from Axia.

Figure 5.66: The approximate distance of 0.4 (left) and its actual distance (right).

Figure 5.67: Driver vision for Figure 5.66.

5.4.4 Pre-collision Feature for SideA and SideB Raspberry Pi

In the SideA and SideB Raspberry Pi, the pre-collision feature is deployed to prevent . A python code is written and used for both Raspberry Pi as it is in charge of detecting the vehicle for left and right side of the car. This python code is used back the same concept from the python code that wrote for the FCW feature. However, there are some part of the code is different compared to the code for FCW feature.

In this feature, *y* coordinate will be used for the approximate distance algorithm. The height of the bounding box (difference between *y-max* and *y-min*) is used to be calculated by the algorithm. This is because when other vehicles pass own vehicle, the height of bounding boxes outlined around those vehicles remain the same. In other word, the approximate distance value will become constant if *y*-coordinate is used by the approximate distance logic for side detection. If the previous code used by the FCW is applied for the SideA and SideB Raspberry Pi, which is using *x*-coordinate for the algorithm, the approximate distance value will keep changing when other vehicles are passing the own vehicle as shown diagram below.

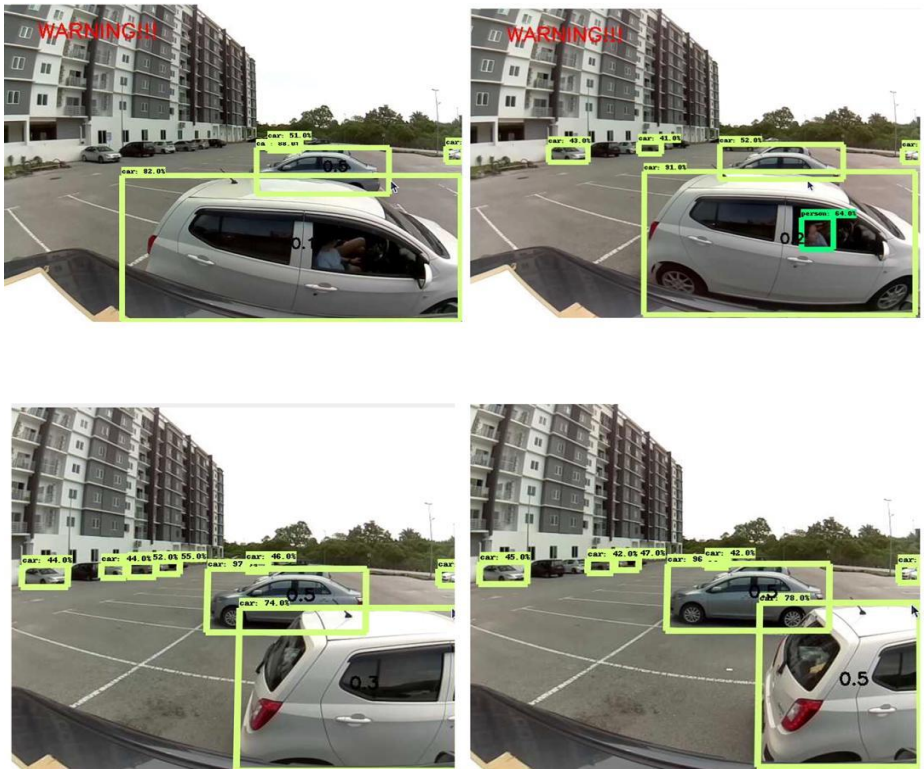


Figure 5.68: The approximate distance value is keep changing when a vehicle is passing the own car.

This is because the width of bounding box will become smaller when the detected vehicle is passing by the side of the own car. Therefore, x- coordinate cannot be used by the algorithm and y-coordinate will be used to detect the approximate distance for side detection. Below diagram shows the approximate distance algorithm for side detection.

```
mid_x = (boxes[0][i][1]+boxes[0][i][3])/2
mid_y = (boxes[0][i][0]+boxes[0][i][2])/2
apx_distance = round(((1 - (boxes[0][i][2] - boxes[0][i][0]))**4),1)
cv2.putText(frame, '{}'.format(apx_distance), (int(mid_x*640),int(mid_y*480)),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,0), 2)
```

Figure 5.69: Approximate distance algorithm logic for side detection.

If the approximate distance value is less than or equal to 0.1, the buzzer will be triggered and generated a beeping sound to alert the novice driver that the other vehicle is passing the own car.

```
if apx_distance <=0.1:
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.3) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.3)
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.3) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.3)
    GPIO.output(buzzer,GPIO.HIGH)
    sleep(0.3) # Delay in seconds
    GPIO.output(buzzer,GPIO.LOW)
    sleep(0.3)
    cv2.putText(frame, 'WARNING!!!', (50,50),
    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,0,255), 2)
```

Figure 5.70: Part of python code for buzzer to produce beeping sound for side detection.

The real distance when the vehicle is detected with an approximate distance of 0.2 for side distance is about 1.6m.

Figure 5.71: The approximate distance of 0.2 for side detection (left) and its actual distance (right).

Besides, the real distance when the vehicle is detected with an approximate distance of 0.1 for side distance is about 0.8m.

Figure 5.72: The approximate distance of 0.1 for side detection (left) and its actual distance (right).

5.4.5 Buzzer Selection Analysis for Pre-collision Alert Feature

Two different types of buzzers are bought which are active and passive buzzer for pre-collision feature. An active buzzer consists of internal oscillator which able generate sound once it is activated. On the other hand, a passive buzzer lacks an internal oscillator which not able to produce a tone automatically. The passive buzzer produces a sound by changing the input signal such as square wave. For passive buzzer, the pitch of sound can be controlled by adjusting the sound frequency.

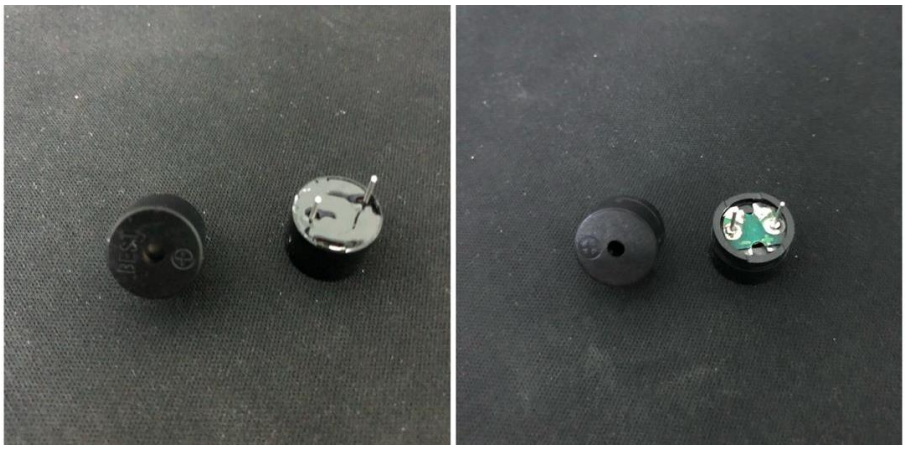


Figure 5.73: Active buzzer (left) and passive buzzer (right).

At first, this project was planned to use 1 active buzzer for frontal Raspberry Pi and 2 passive buzzers for SideA and SideB Raspberry Pi in order to differentiate the warning sound. However, the sound produced by the passive buzzer is much smaller than the active buzzer and may not noticed by the driver. Therefore, 3 active buzzers were used in this project and the buzzers were pasted on the left, right and the frontal side of car ceiling in order to differentiate which Raspberry Pi was triggered the pre-collision feature.

5.5 Live Streaming Feature

A simple streaming feature is created using the source code from the official PiCamera package. This feature is employed in three Raspberry Pi to provide car surrounding view in real-time. Parents or trainers can use this feature to monitor the novice drivers' driving behaviour. They can use their smartphone or laptop to watch the live feeds by entering the IP address of Raspberry Pis. Besides, parents or trainers can determine whether or not the novice drivers are driving dangerously based on their driving experiences and trainers' profession. The live feed is live streamed using Python's built in "http.server" module.

```
if __name__ == '__main__':
    with picamera.PiCamera(resolution=(640, 480), framerate=30) as camera:
        output = StreamingOutput()
        camera.start_recording(output, format='mjpeg')
        try:
            address = ('', 8000)
            server = StreamingServer(address, StreamingHandler)
            server.serve_forever()
        except KeyboardInterrupt:
            pass
        finally:
            camera.stop_recording()
```

Figure 5.74: Part of python code for streaming feature.

The live feeds captured by different Raspberry Pis were successfully live streamed to the internet as below:

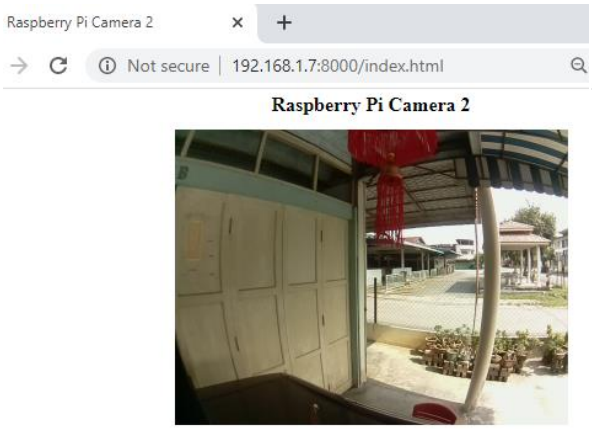


Figure 5.75: Live feed of SideA Pi camera in web browser.

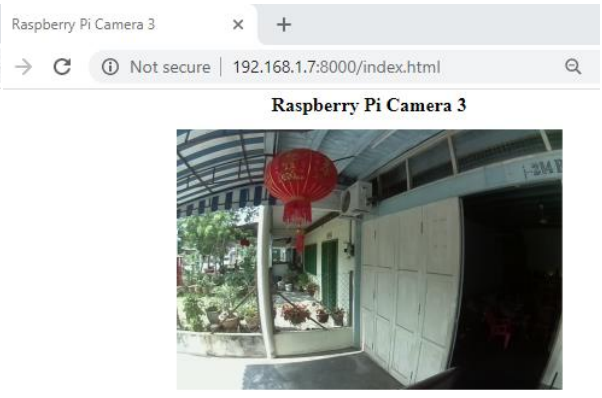


Figure 5.76: Live feed of SideB Pi camera in web browser.

5.6 Experimental Results

The proposed system is analysed and tested on different locations such as the front yard, parking area, and basketball court. Besides, two different sizes of vehicles are used to test out the accuracy of the proposed system, which are Nissan Almera (Vehicle 1) and Perodua Axia (Vehicle 2).

5.6.1 Normal State for Frontal Raspberry Pi

Below diagrams show the screenshots of normal state for Frontal Raspberry Pi, which means the Forward Collision Warning (FCW) feature is not triggered. Figure 5.42 shows that Vehicle 1 was detected with a bounding box but did not triggered the FCW feature. This is because the *scores* of Vehicle 1 as a vehicle is lower than 0.6, means that Vehicle 1 is too far to be detected as a vehicle and will not causes any collision.

Figure 5.77: Vehicle 1 is detected with a bounding box.

Figure 5.78 shows that Vehicle 1 is not detected along with a bounding box. However, this can be considered as Vehicle 1 is far enough and will not causes any collision.

Figure 5.78: Vehicle 1 is not detected with a bounding box.

Below diagrams show that Vehicle 2 is detected with a bounding box but the FCW feature still not be triggered as the *scores* of Vehicle 2 as a vehicle is lower than 0.8, which means Vehicle 2 is too far to be detected and will not triggered the pre-collision alert.

Figure 5.79: Vehicle 2 is detected with a bounding box.

5.6.2 Warning State for Frontal Raspberry Pi

The FCW feature will be triggered when a vehicle is detected with an approximate distance value that lower or equal to 0.5. Below diagrams show that the FCW feature was triggered when Vehicle 1 is near the own vehicle. In Figure 5.80, Vehicle 1 was detected with an approximate distance value 0.5 and it is in a line of sight. Besides, the frontal buzzer is activated and generated a low frequency of beeping sound.

Figure 5.80: Vehicle 1 is detected with approximate distance value of 0.5 by Frontal Raspberry Pi.

In Figure 5.81, Vehicle 1 is detected with approximate distance value of 0.4 and the vehicle is in a line of sight. The frontal buzzer is triggered and generated a medium frequency of beeping sound.

Figure 5.81: Vehicle 1 is detected with approximate distance value of 0.4 by Frontal Raspberry Pi.

In Figure 5.82, Vehicle 1 is detected with approximate distance value of 0.3 and the vehicle is in a line of sight. The frontal buzzer is triggered and generated a high frequency of beeping sound.

Figure 5.82: Vehicle 1 is detected with approximate distance value of 0.3 by Frontal Raspberry Pi.

In Figure 5.83, Vehicle 2 is detected with approximate distance value of 0.5 and the vehicle is in a line of sight. The frontal buzzer is triggered and generated a high frequency of beeping sound.

Figure 5.83: Vehicle 2 is detected with approximate distance value of 0.5 by Frontal Raspberry Pi.

In Figure 5.84, Vehicle 2 is detected with approximate distance value of 0.4 and the vehicle is in a line of sight. The frontal buzzer is triggered and generated a medium frequency of beeping sound.

Figure 5.84: Vehicle 2 is detected with approximate distance value of 0.4 by Frontal Raspberry Pi.

In Figure 5.85, Vehicle 2 is detected with approximate distance value of 0.3 and the vehicle is in a line of sight. Although this screenshot shows that Vehicle 2 is detected as a train with a score of 0.54, but it also detected as a car with a score that above 0.8. This is because Axia car looks like a train for the TensorFlow Object Detection API. As you can see that a yellow bounding box is outlined but covered with a light blue bounding box, and the yellow bounding box appeared means that the object is detected as a car. Therefore, the frontal buzzer is triggered and generated a medium frequency of beeping sound.

Figure 5.85: Vehicle 2 is detected with approximate distance value of 0.3 by Frontal Raspberry Pi.

5.6.3 Normal State for SideA Raspberry Pi

Below diagrams show the normal state for SideA Raspberry Pi, meaning that the detected vehicle has an approximate distance value that higher than 0.1. In Figure 5.86, Vehicle 1 is detected with an approximate distance value of 0.2 and this will not activate the pre-collision feature.

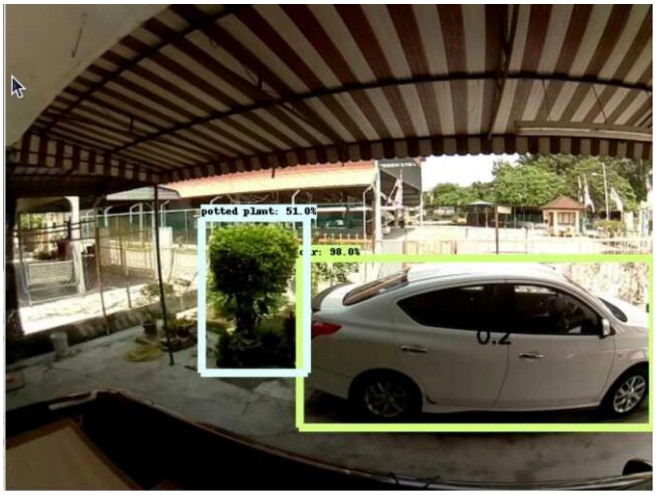


Figure 5.86: Vehicle 1 is detected with approximate distance value of 0.2 by SideA Raspberry Pi.

In Figure 5.87, Vehicle 2 is detected with an approximate distance value of 0.3 and this will not activate the pre-collision feature.

Figure 5.87: Vehicle 2 is detected with approximate distance value of 0.3 by SideA Raspberry Pi 3.

5.6.4 Warning State for SideA Raspberry Pi

Warning state for SideA Raspberry Pi means that a vehicle is detected with an approximate distance value that lower than or equal to 0.1. Below diagrams show that the pre-collision feature was triggered as Vehicle 1 is too near to the own car. In Figure 5.88, Vehicle 1 is detected with an approximate distance value of 0.1. The buzzer is activated and generate a beeping sound to alert the driver.

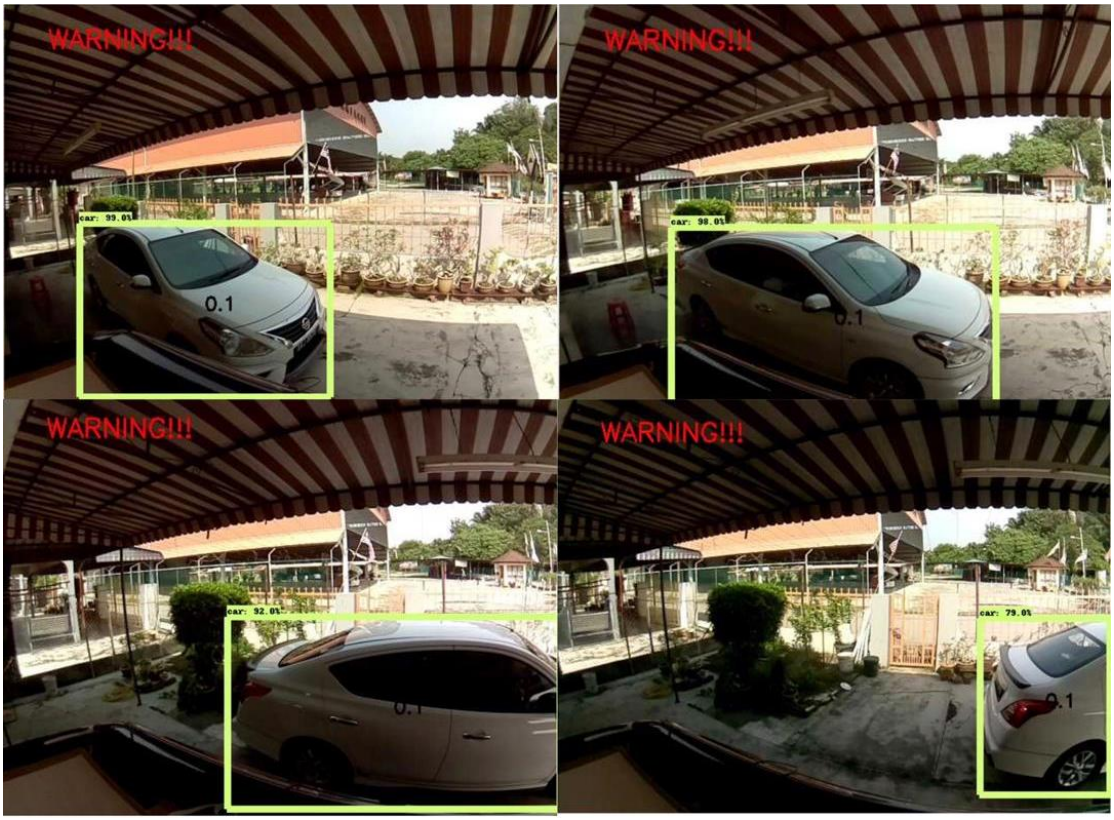


Figure 5.88: Vehicle 1 is detected with approximate distance value of 0.1 by SideA Raspberry Pi.

Below diagrams show that the pre-collision feature was triggered as Vehicle 1 is too near to the own car. In Figure 5.54, Vehicle 2 is detected with an approximate distance value of 0.1. The buzzer is activated and generate a beeping sound to alert the driver.

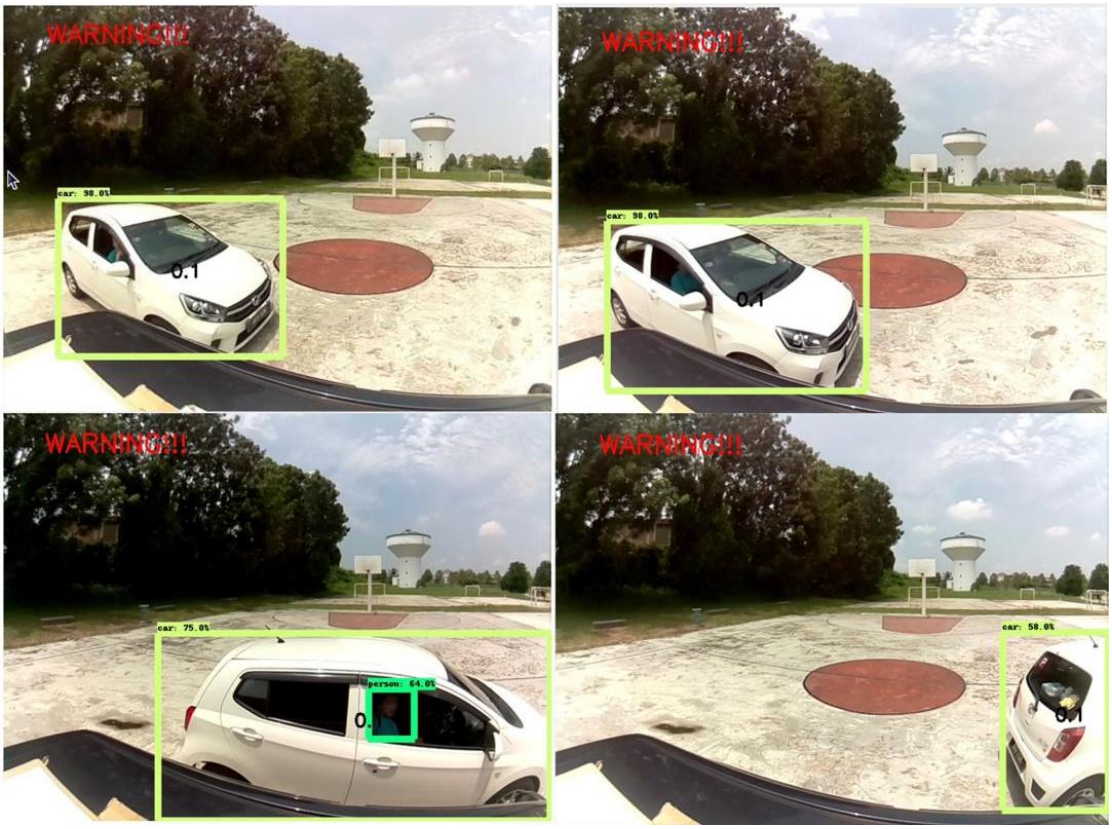


Figure 5.89: Vehicle 2 is detected with approximate distance value of 0.1 by SideA Raspberry Pi.

5.6.5 Normal State for SideB Raspberry Pi

Below diagrams show the normal state for SideB Raspberry Pi, meaning that the detected vehicle has an approximate distance value that higher than 0.1. In Figure 5.90, Vehicle 1 is detected with an approximate distance value of 0.2 and this will not activate the pre-collision feature.

Figure 5.90: Vehicle 1 is detected with approximate distance value of 0.2 by SideB Raspberry Pi.

In Figure 5.91, Vehicle 2 is detected with an approximate distance value of 0.2 and this will not activate the pre-collision feature.

Figure 5.91: Vehicle 2 is detected with approximate distance value of 0.2 by SideB Raspberry Pi.

5.6.6 Warning State for SideB Raspberry Pi

Warning state for SideB Raspberry Pi means that a vehicle is detected with an approximate distance value that lower than or equal to 0.1. Below diagrams show that the pre-collision feature was triggered as Vehicle 1 is too near to the own car. In Figure 5.92, Vehicle 1 is detected with an approximate distance value of 0.1.

Figure 5.92: Vehicle 1 is detected with approximate distance value of 0.1 by SideB Raspberry Pi.

Below diagrams show that the pre-collision feature was triggered as Vehicle 1 is too near to the own car. In Figure 5.93, Vehicle 2 is detected with an approximate distance value of 0.1. The buzzer is activated and generate a beeping sound to alert the driver.

Figure 5.93: Vehicle 2 is detected with approximate distance value of 0.1 by SideB Raspberry Pi.

5.6.7 Pre-collision Detection Result Analysis

Video of Test Case	Test Case Scenario	Total Frame	TP	TN	FP	FN	Accuracy: (TP+TN /Total Frame) *100%
1	Own vehicle drives near to the rear side of Vehicle 1	900	566	258	44	32	91.56%
			Total: 824		Total: 76		
2	Own vehicle drives near to the rear side of Vehicle 2	930	613	203	56	58	87.75%
			Total: 816		Total: 114		
3	Own vehicle drives near to the right side of Vehicle 1	868	512	255	57	44	88.36%
			Total: 767		Total: 101		
4	Own vehicle drives near to the right side of Vehicle 2	886	548	267	50	21	91.99%
			Total: 815		Total: 71		
5	Own vehicle drives near to the left side of Vehicle 1	770	454	238	45	33	89.87%
			Total: 692		Total: 78		
6	Own vehicle drives near to the left side of Vehicle 2	857	526	263	48	20	92.07%
			Total: 789		Total: 69		
Average pre-collision detection accuracy:							**90.27%**

Table 5.4: Pre-collision detection results.

Above table shows the result of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) of the pre-collision detection and its accuracy for 6 different test case scenarios with two different vehicles (Vehicle 1 and 2). The average accuracy of the pre-collision detection is around 90.27%.

		Condition	
		Pre-collision	**No pre-collision**
Test Result	**Alarm On**	TP	FP
	Alarm Off	FN	TN

Table 5.5: Result description for result analysis.

Table 5.5 shows the result description for the result analysis in previous table. True Positive (TP) means that the buzzer is triggered when pre-collision event occurs while False Positive (FP) means that the buzzer is not triggered although there is no pre-collision event. Besides, True Negative (TN) is achieved when there is no pre-collision event and the buzzer is not triggered while the condition will become False Negative (FN) when the buzzer is not triggered when pre-collision event occurs.

5.7 Weakness of Proposed System

There are some areas that the TensorFlow Object Detection API cannot be performed in order to use the approximate distance algorithm. For SideA and SideB Raspberry Pi, the rear side of vehicle can be seen but it is cut into half as shown in Figure 5.94.

Figure 5.94: Screenshot from SideB (left) and screenshot from SideA (right).

Besides, the TensorFlow Object Detection API has a lower accuracy for object detection as sometimes a detected object is not match with the object that described by the API through bounding box. Figure 5.95 shows that the prayer house is detected as a train with a *score* of 0.41 but it does not a train.

Figure 5.95: Prayer house was detected as a train.

Sometimes, Perodua Axia is detected as both car and train at the same time as shown in Figure 5.96.

Figure 5.96: Proton Axia was detected as a train.

Even there is no object around the vehicle but still detected as a train – see Figure 5.97.

Figure 5.97: No object around but still detected as a train.

Moreover, there are some false positive results occurred for this proposed system. When a vehicle is passing through the frontal part of own vehicle, a false alarm was triggered although it is not near to the own car – see Figure 5.98. This is because the width of bounding box that outlined around that detected vehicle is calculated and returned an approximate distance value of 0.4 and the FCW feature is triggered. In other word, the FCW feature for Frontal Raspberry Pi cannot be applied to those vehicles that passing through the own vehicle.

Figure 5.98: Screenshot of false positive result.

In this proposed system, only vehicle categories are selected for pre-collision prediction. This is because the approximate distance algorithm is not suitable for other categories like motorcycle and bicycle. For motorcycle category, it sometimes cannot be detected by the TensorFlow Object Detection API when the tyres of motorcycle was covered. In Figure 5.99, the motorcycle was not detected as a motorcycle but only the motorcyclist was detected.

Figure 5.99: The motorcycle is not be detected.

CHAPTER 6: CONCLUSION

6.1 Conclusion

The purpose of Advanced Driver Assistance System (ADAS) is to obtain a safety and comfort driving condition for the drivers. However, ADAS will only be installed to the advanced luxury cars due to system complexity and the expensive hardware components. Therefore, the vision-based ADAS is the first choice for most of the people as it only requires camera modules.

Nowadays, the novice drivers are more likely be involved in a car accident compared to others as many things can distract these novice drivers such as phone notification and car entertainment systems. They might not familiar with the road condition as they are still a novice driver. In addition, some areas cannot be seen using only the side mirrors and rear-view mirror. The driver might unintentionally hit the other vehicles or any pedestrian because of these existing problems.

In this project, the prototype of real-time driving monitoring system is being developed using three sets of Raspberry Pi 3, each mounted with wide angle Pi camera in order to capture the surround view of the vehicle. Besides, each Raspberry Pi is connected an active buzzer in order to generate beeping sound to alert the novice driver. A pre-trained TensorFlow model is being used for the collision avoidance. Also, Python3 is used as the development language. When the other vehicles are too close to the owner's car, a beeping sound alert will be generated. A triangular prism shaped camera module was designed for mounting three wide angle cameras in order to acquire 360 degree surrounding view. Also, the pre-collision feature is applied to detect whether or not the other vehicles are too close to the owner's car. If the distance between two vehicles are near, a warning beeping sound will be generated and a "Warning!!!" sign will be displayed on the screen.

6.2 Future Work

Some future enhancement and improvement can be achieved for this proposed project. Firstly, the camera module can be changed to night vision wide angle camera. The current camera module used in this project cannot be used to acquire video well in low-light condition. The pre-collision feature may fail to be triggered as the car surrounding cannot be captured well by current camera module. Therefore, night vision camera is used to ensure that the pre-collision feature can be functioned well during night-time.

Besides, the number of cameras can be increased from 3 cameras into 4 cameras. In this project, although the car surrounding view was successfully captured by 3 cameras, the pre-collision feature is still cannot be applied for the rear side of the vehicle. Both SideA and SideB Raspberry Pi were in-charged of capturing the rear view of the vehicle and caused the rear view was cut into half. Therefore, 4 cameras can be used as future enhancement and one of the cameras can be used to apply pre-collision feature for rear side of the vehicle.

Latency is one of the issues to be solved when showing the detection results on the screen from Raspberry Pi. This is because Raspberry Pi is a weak processor for image processing. Therefore, Raspberry Pi can be acted as a client or gateway to pre-process the data and send to a powerful server for processing such as edge computing.

Lastly, current camera mounting module is made up of carton box, which means it cannot withstand bad weather condition such as heavy rain or windy day. Current module also cannot withstand high speed of the vehicle when it is place on the vehicle. Therefore, the materials for making a camera mounting module should be changed to other materials such as metal to build a stronger module that can handle the bad weather condition and high speed of the vehicle.

References / Bibliography

Benchoff, B 2016, *INTRODUCING THE RASPBERRY PI 3*. Available from: <https://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/>. [18 March 2020].

Chavan, S & Shete, VV 2017, 'Three Dimensional Surround View System', *International Journal of Electronics, Electrical and Computational System*, vol.6, no.7.

Edje Electronics 2018, 'How to Set Up TensorFlow Object Detection on the Raspberry Pi' (video file). Available from: <https://www.youtube.com/watch?v=npZ-8Nj1YwY>. [18 March 2020].

Gan, PL 2017, 'Road accidents cost Malaysia RM9.2bil in 2016', *TheStar*. Available from: <https://www.thestar.com.my/news/nation/2017/02/02/road-accidents-cost-malaysia-rm9dot2bil-in-2016/>. [3 March 2020].

Hsieh, C.H., Lin, D.C., Wang, C.J., Chen, Z.T. and Liaw, J.J., 2019, July. Real-Time Car Detection and Driving Safety Alarm System With Google Tensorflow Object Detection API. In *2019 International Conference on Machine Learning and Cybernetics (ICMLC)* (pp. 1-4). IEEE.

Keshari, K 2020, Object Detection Tutorial in TensorFlow: Real-Time Object Detection. Available from: <https://www.edureka.co/blog/tensorflow-object-detection-tutorial/>. [22 August].

Kim 2019, *What's the difference between active buzzers and passive buzzers?*. Available from: <https://www.manorshi.com/What-s-the-difference-between-active-buzzers-and-passive-buzzers-id3333285.html>. [21 August 2020].

Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014, September. Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740-755). Springer, Cham.

Mohn Uzir Mahidin, 2018, *PRESS RELEASE STATISTICS ON CAUSES OF DEATH, MALAYSIA, 2018.* Available from: <https://www.dosm.gov.my/v1/index.php?r=column/pdfPrev&id=aWg2VjJkZHhYcDdEM3JQSGloeTVlZz0930>. [12 March 2020].

OpenCV.org, computer software 2020. Available from: <https://opencv.org/>. [12 March 2020].

Prototyping Model in Software Engineering: Methodology, Process, Approach, 2020. Available from: <https://www.guru99.com/software-engineering-prototyping-model.html>. [23 March 2020].

RaspberryPi.org, computer software 2020. Available from: <https://www.raspberrypi.org/>. [12 March 2020].

Rodrigue, J 2020, *Evolution of Transport Technology since the 18th Century.* Available from: <https://transportgeography.org/?page_id=1599>. [10 March 2020].

Sam C. Mitchell & Associates 2018, Common Types of Car Crashes. Available from: <https://www.scmitchell.com/blog/2018/january/common-types-of-car-crashes/>. [20 August 2020].

Sentdex 2017, 'Determining other vehicle distances & collision warning (object detection) – Self Driving Cars p.18' (video file). Available from: <https://www.youtube.com/watch?v=o3Ky_EdHVrA&feature=emb_title>. [20 March 2020].

Singh, V 2020, *Top 7 SDLC Methodologies.* Available from: <https://hackr.io/blog/sdlc- methodologies>. [23 March 2020].

Solawatz, J 2020*, The TensorFlow 2 Object Detection Library is Here*. Available from: <https://blog.roboflow.com/the-tensorflow2-object-detection-library-is-here/>. [21 August 2020].

Stevens, AP 2018, *Here's what puts teen drivers at greatest risk of a crash*. Available from: <https://www.sciencenewsforstudents.org/article/heres-what-puts-teen-drivers-greatest-risk-crash> [8 March 2020]

Udayan, OP 2019, *How Bounding Box Enables Object Detection?*. Available from: <https://www.infolks.info/blog/how-bounding-box-enable-object-detection/>. [23 August 2020].

Yegulalp, S 2019, *What is TensorFlow? The machine learning library explained*. Available from: <https://www.infoworld.com/article/3278008/what-is-tensorflow-themachine-learning-library-explained.html>. [22 March 2020].

Yu, M. and Ma, G., 2014. 360 surround view system with parking guidance. *SAE International Journal of Commercial Vehicles*, *7*(2014-01-0157), pp.19-24.

Zhang, B., Appia, V., Pekkucuksen, I., Liu, Y., Umit Batur, A., Shastry, P., Liu, S., Sivasankaran, S. and Chitnis, K., 2014. A surround view camera solution for embedded systems. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 662-667).

APPENDIX A – BIWEEKLY REPORT

FINAL YEAR PROJECT WEEKLY REPORT

Project II

Trimester, Year: Year 3 Semester 3	**Study week no.: 2**
Student Name & ID: Cho Wei Tuck – 16ACB03122	
Supervisor: Dr. Lau Phooi Yee	
Project Title: REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI	

1. WORK DONE

- Successful to run Object Detection using TensorFlow for two new arrived Raspberry Pis.

2. WORK TO BE DONE

- Build a camera mounting module that can be used for actual implementation on the running vehicle.

3. PROBLEMS ENCOUNTERED

- None.

4. SELF EVALUATION OF THE PROGRESS

- Knowledge about Image Processing and Image Classification.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

Project II

Trimester, Year: Year 3 Semester 3	**Study week no.: 4**
Student Name & ID: Cho Wei Tuck – 16ACB03122	
Supervisor: Dr. Lau Phooi Yee	
Project Title: REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI	

1. WORK DONE

- Successful built a camera mounting module that can be used for actual implementation on the running vehicle

2. WORK TO BE DONE

- Make the camera mounting module to be more presentable.
- Improve the accuracy of detection.

3. PROBLEMS ENCOUNTERED

- None.

4. SELF EVALUATION OF THE PROGRESS

- Knowledge about bounding box that outlined around the detected object using OpenCV

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

Project II

Trimester, Year: Year 3 Semester 3	**Study week no.: 6**
Student Name & ID: Cho Wei Tuck – 16ACB03122	
Supervisor: Dr. Lau Phooi Yee	
Project Title: REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI	

1. WORK DONE

- Successfully made the camera mounting module to be more presentable by decorating.
- Successfully improve the accuracy of detection by adjusting the current algorithm used.

2. WORK TO BE DONE

- Add pre-collision feature for SideA and SideB of the vehicle.
- Update the report until system design.

3. PROBLEMS ENCOUNTERED

- None.

4. SELF EVALUATION OF THE PROGRESS

- Knowledge about coordinates of the bounding box.

Lau

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

Project II

Trimester, Year: Year 3 Semester 3	**Study week no.: 8**
Student Name & ID: Cho Wei Tuck – 16ACB03122	
Supervisor: Dr. Lau Phooi Yee	
Project Title: REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI	

1. WORK DONE

- Successfully add pre-collision feature for SideA and SideB of the vehicle
- Report was update until system design

2. WORK TO BE DONE

- Do rigorous testing for detection algorithm used in this project.
- Apply buzzer to provide warning feature.

3. PROBLEMS ENCOUNTERED

- None.

4. SELF EVALUATION OF THE PROGRESS

- Report was progressing as scheduled

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

Project II

Trimester, Year: Year 3 Semester 3	**Study week no.: 9**
Student Name & ID: Cho Wei Tuck – 16ACB03122	
Supervisor: Dr. Lau Phooi Yee	
Project Title: REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI	

1. WORK DONE

- Successfully did rigorous testing for detection algorithm used in this project.
- Successfully sound up the buzzer when another vehicle is near to own vehicle.

2. WORK TO BE DONE

- Use different beeping sound for different side detection.

3. PROBLEMS ENCOUNTERED

- The sound produced by the passive buzzer is too small to be heard.

4. SELF EVALUATION OF THE PROGRESS

- Report was progressing as scheduled
- Knowledge about GPIO of Raspberry Pi

Lau

_____
Supervisor's signature

_____
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

Project II

Trimester, Year: Year 3 Semester 3	**Study week no.: 10**
Student Name & ID: Cho Wei Tuck – 16ACB03122	
Supervisor: Dr. Lau Phooi Yee	
Project Title: REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI	

1. WORK DONE

- Successfully use other way to differentiate the alert sound for different side detection.

2. WORK TO BE DONE

- Show different camera detection.

3. PROBLEMS ENCOUNTERED

- None.

4. SELF EVALUATION OF THE PROGRESS

- Report was progressing as scheduled.
- Knowledge about the difference of active buzzer and passive buzzer.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

Project II

Trimester, Year: Year 3 Semester 3	**Study week no.: 12**
Student Name & ID: Cho Wei Tuck – 16ACB03122	
Supervisor: Dr. Lau Phooi Yee	
Project Title: REAL-TIME DRIVING MONITORING SYSTEM FOR NEW DRIVERS USING 360° CAMERA WITH RASPBERRY PI	

1. WORK DONE

- Successfully showed different camera detection by using VNC viewer.

2. WORK TO BE DONE

- Continue with system demonstration
- Finalise the report.
- Prepare presentation slides.

3. PROBLEMS ENCOUNTERED

- None.

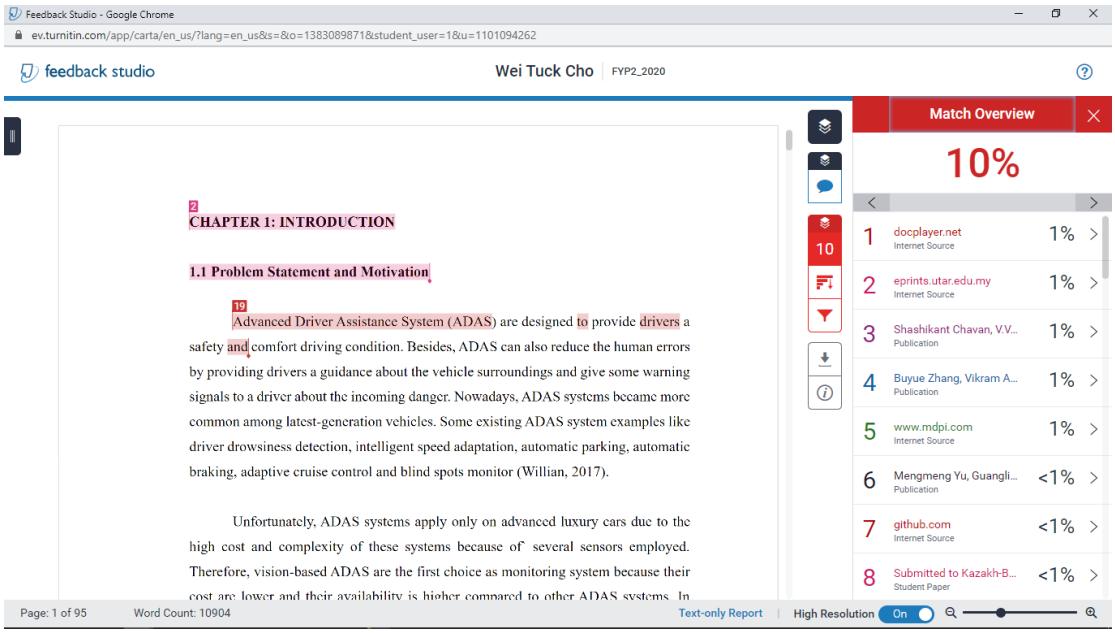
4. SELF EVALUATION OF THE PROGRESS

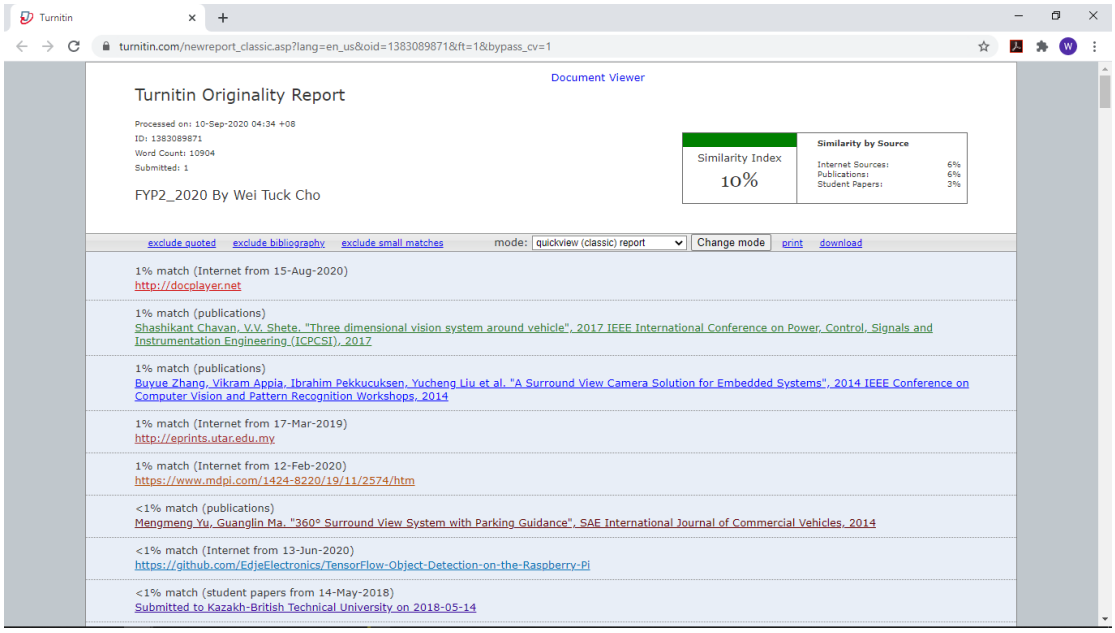
- Confident for demonstration of the final system.

Supervisor's signature

Student's signature

Plagiarism Check Result





FYP2_2020

ORIGINALITY REPORT

10%	6%	6%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	docplayer.net Internet Source	1%
2	eprints.utar.edu.my Internet Source	1%
3	Shashikant Chavan, V.V. Shete. "Three dimensional vision system around vehicle", 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), 2017 Publication	1%
4	Buyue Zhang, Vikram Appia, Ibrahim Pekkucuksen, Yucheng Liu et al. "A Surround View Camera Solution for Embedded Systems", 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014 Publication	1%
5	www.mdpi.com Internet Source	1%
6	Mengmeng Yu, Guanglin Ma. "360° Surround View System with Parking Guidance", SAE	<1%

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Cho Wei Tuck
ID Number(s)	16ACB03122
Programme / Course	Bachelor of Information Technology (Hons) Communications and Networking
Title of Final Year Project	Real-Time Driving Monitoring System for New Drivers using 360° Camera with Raspberry Pi

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 10 % **Similarity by source** Internet Sources: 6 % Publications: 6 % Student Papers: 3 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required and limits approved by UTAR are as follows: (i) Overall similarity index is 20% and below, and **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.\

Lau

Signature of Supervisor

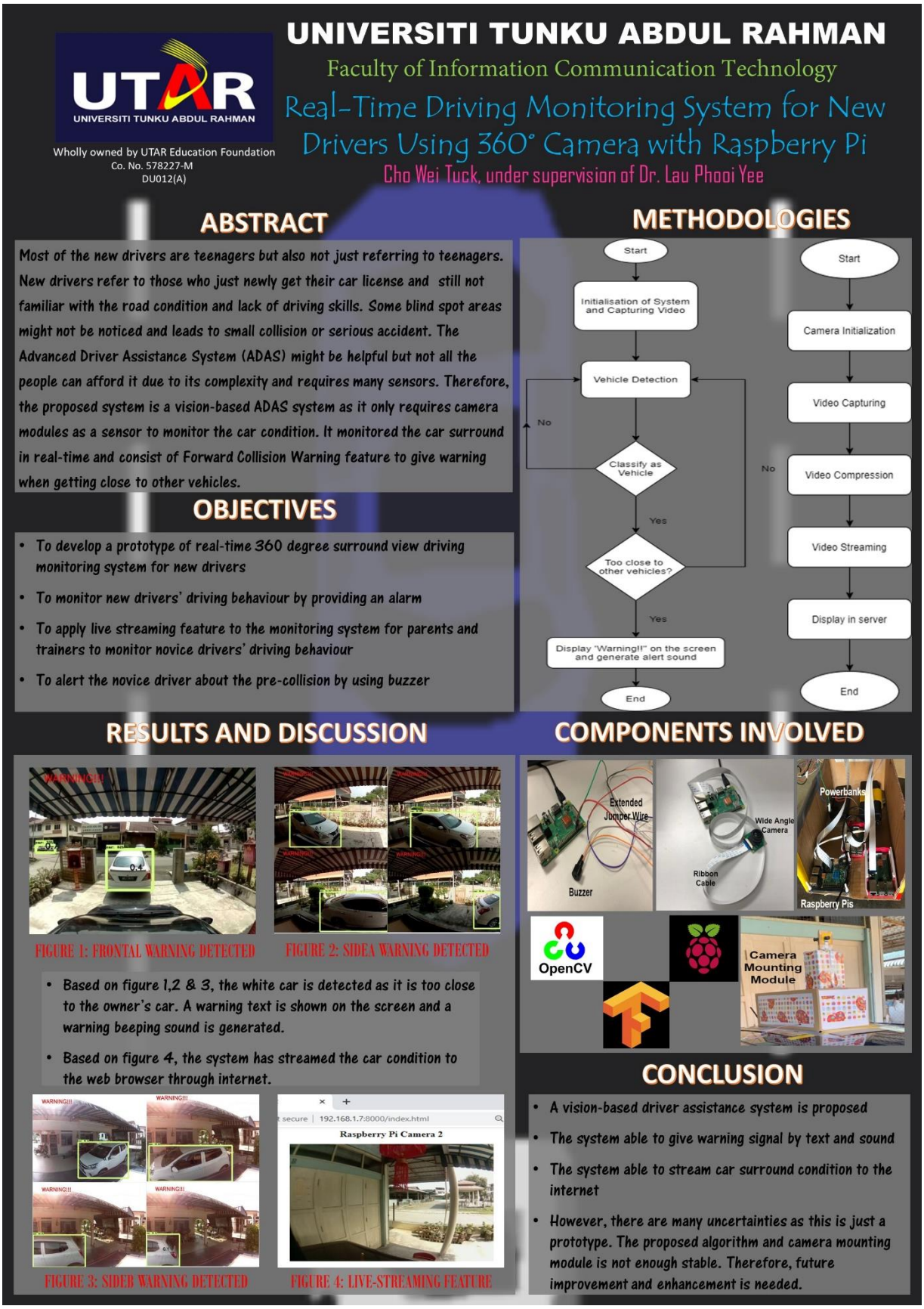
Name: LAU PHOOI YEE

Date: 11 SEPTEMBER 2020

Signature of Co-Supervisor

Name:

Poster


UNIVERSITI TUNKU ABDUL RAHMAN
Wholly owned by UTAR Education Foundation
Co. No. 578227-M
DU012(A)
UNIVERSITI TUNKU ABDUL RAHMAN
Faculty of Information Communication Technology
Real-Time Driving Monitoring System for New Drivers Using 360° Camera with Raspberry Pi
Cho Wei Tuck, under supervision of Dr. Lau Phooi Yee
ABSTRACT
Most of the new drivers are teenagers but also not just referring to teenagers. New drivers refer to those who just newly get their car license and still not familiar with the road condition and lack of driving skills. Some blind spot areas might not be noticed and leads to small collision or serious accident. The Advanced Driver Assistance System (ADAS) might be helpful but not all the people can afford it due to its complexity and requires many sensors. Therefore, the proposed system is a vision-based ADAS system as it only requires camera modules as a sensor to monitor the car condition. It monitored the car surround in real-time and consist of Forward Collision Warning feature to give warning when getting close to other vehicles.
METHODOLOGIES
Start
Initialisation of System and Capturing Video
Vehicle Detection
No
Classify as Vehicle
Yes
Too close to other vehicles?
Yes
Display "Warning!!" on the screen and generate alert sound
End
No
Start
Camera Initialization
Video Capturing
Video Compression
Video Streaming
Display in server
End
OBJECTIVES
To develop a prototype of real-time 360 degree surround view driving monitoring system for new drivers
To monitor new drivers' driving behaviour by providing an alarm
To apply live streaming feature to the monitoring system for parents and trainers to monitor novice drivers' driving behaviour
To alert the novice driver about the pre-collision by using buzzer
RESULTS AND DISCUSSION
FIGURE 1: FRONTAL WARNING DETECTED
FIGURE 2: SIDEA WARNING DETECTED
Based on figure 1,2 & 3, the white car is detected as it is too close to the owner's car. A warning text is shown on the screen and a warning beeping sound is generated.
Based on figure 4, the system has streamed the car condition to the web browser through internet.
FIGURE 3: SIDEB WARNING DETECTED
192.168.1.7:8000/index.html
Raspberry Pi Camera 2
FIGURE 4: LIVE-STREAMING FEATURE
COMPONENTS INVOLVED
Extended Jumper Wire
Buzzer
Wide Angle Camera
Ribbon Cable
Powerbanks
Raspberry Pis
OpenCV
Camera Mounting Module
CONCLUSION
A vision-based driver assistance system is proposed
The system able to give warning signal by text and sound
The system able to stream car surround condition to the internet
However, there are many uncertainties as this is just a prototype. The proposed algorithm and camera mounting module is not enough stable. Therefore, future improvement and enhancement is needed.

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	16ACB03122
Student Name	CHO WEI TUCK
Supervisor Name	DR. LAU PHOOI YEE

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Front Cover
√	Signed Report Status Declaration Form
√	Title Page
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report. _____ (Signature of Student) Date: 10th SEPTEMBER 2020	Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction. _____ (Signature of Supervisor) Date: 11 SEPTEMBER 2020