

Vehicle Speed Detection Using Machine Vision on a Single-Board Computer

By

Goh Jin Yu

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2025

COPYRIGHT STATEMENT

© 2025 Goh Jin Yu. All rights reserved.

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to my supervisor, Ts. Wong Chee Siang, for providing me with the opportunity to work on the single-board computer (SBC) project. This experience marks the beginning of my journey towards a career in SBC development, and I am truly thankful for your guidance and support. Additionally, I want to express my appreciation to my parents and family for their unwavering love, support, and encouragement throughout this journey. Your belief in me has been invaluable. Thank you all immensely.

ABSTRACT

This proposal project is about to develop a real time speed tracking system in UTAR. In this era, more and more students drive to school, and many of them drive very fast. Although their speed cap and road bumps in school area, but many of them still unrealise that they exceed the speed limit. One of the methods to remind student how fast they drive, is to make a sign board where it can detect how fast they drive and show it on the sign board with a large 7segment LCD. This not only reminds the driver how fast they drive, but also other students around that area could also see the sign board.

This project leverages a single-board computer such as Raspberry Pi 4B and Star Five Vision Five v2 to monitor vehicle speed using a camera-based detection system. The captured data is processed to determine the speed, which is subsequently displayed on a 2-digit 7-segment display. The integration of real-time image processing with the simplicity of the display offers an efficient solution for speed detection applications. The system's design, implementation, and performance evaluation are discussed, highlighting its potential use in traffic monitoring and management systems. The platform architecture will be based on ARM and RISC-V.

Area of Study (Minimum 1 and Maximum 2): Single Board Computer, Computer Vision

Keywords (Minimum 5 and Maximum 10): RISC-V architecture, ARM architecture, Python, OpenCV, YouOnlyLookOnce(YOLO)

TABLE OF CONTENTS

TITLE PAGE	i
COPYRIGHT STATEMENT	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	viii
LIST OF TABLES	ix
LIST OF SYMBOLS	x
LIST OF ABBREVIATIONS	xi
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement and Motivation	1-2
1.2 Objectives	2
1.3 Project Scope and Direction	2-3
1.4 Contributions	3-4
1.5 Report Organization	4
CHAPTER 2 LITERATURE REVIEW	5
2.1 Previous Works on Speed Detection System	5-7
2.1.1 Radar and LIDAR-Based Systems	
2.1.2 Inductive Loop Sensors	
2.1.3 Video Based Speed Detection System	
2.2 Limitation of Existing Systems	8
2.2.1 Limitation of Radar and LIDAR-Based Systems	
2.2.2 Limitation of Inductive Loop Sensors	
2.2.3 Limitation of Video Based Speed Detection System	
2.3 Summary of Strength and Limitation	9

CHAPTER 3 SYSTEM METHODOLOGY/APPROACH	10-11
3.1 System Design Diagram/Equation	11
3.1.1 System Architecture Diagram	11-14
3.1.2 Use Case Diagram and Description	15-17
3.1.3 Activity Diagram	18-19
3.1.4 Timeline	19-20
CHAPTER 4 SYSTEM DESIGN	21
4.1 System Block Diagram	21
4.2 System Components Specifications	22-23
4.3 Circuits and Components Design	23-25
4.4 System Components Interaction Operations	26-27
CHAPTER 5 SYSTEM IMPLEMENTATION	28
5.1 Hardware Setup	28-30
5.2 Software Setup	30-41
5.3 Setting and Configuration	41
5.4 System Operation (with Screenshot)	41-50
5.5 Implementation Issues and Challenges	50-53
5.6 Concluding Remark	54-55
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION	56
6.1 System Testing and Performance Metrics	56-57
6.2 Testing Setup and Result	58-72
6.3 Project Challenges	72-75
6.4 Objectives Evaluation	75-76
6.5 Concluding Remark	77

CHAPTER 7 CONCLUSION AND RECOMMENDATION	78
7.1 Conclusion	78-79
7.2 Recommendation	79-80
REFERENCES	81-82
APPENDIX	
POSTER	83

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1	LIDAR speed detection	5
Figure 2.2	Inductive loop sensor	6
Figure 2.3	Video based speed detection system	7
Figure 3.1	Waterfall Model	10
Figure 3.2	System Block Diagram	11
Figure 3.3	Camera Mounting Point 1	13
Figure 3.7	Use case Diagram for Vehicle Speed Detection System	15
Figure 3.8	Activity Diagram for Speed Detection System	18
Figure 3.9	Gantt Chart FYP1	19
Figure 3.9.1	Gantt Chart FYP2	20
Figure 4.1	Block Diagram of Speed Detection System	21
Figure 4.2	General Work Procedure	22
Figure 4.3	Circuit and Component design for Raspi 4B	23
Figure 4.4	GPIO diagram for Raspi 4B	24
Figure 4.5	Circuit and Component design for VFv2	24
Figure 4.6	GPIO diagram for Vision Five v2	25
Figure 4.7	System Components Interaction Operation for Raspi 4B	26
Figure 4.8	System Components Interaction Operation for VFv2	27
Figure 5.1	Example of System Setup	29
Figure 5.2	Example of System Setup 2	30
Figure 5.3	Official download page of Raspi	30
Figure 5.4	Main page of Raspi Imager	31
Figure 5.5	Selecting OS	31
Figure 5.6	Select Storage device	32
Figure 5.7	Installation Complete	32
Figure 5.8	Inserting microSD card into Raspi 4B	33
Figure 5.9	PI connect	34
Figure 5.9.1	VFv2 OS download page	34

Figure 5.9.2	Disk Imager	35
Figure 5.9.3	Connecting all the peripherals to VFv2	36
Figure 5.9.4	Attached heatsink on the CPU of VFv2	36
Figure 5.9.5	TP-link router setup user interface	37
Figure 5.9.6	Selecting wireless network	37
Figure 5.9.7	Setting up password	38
Figure 5.9.8	Git Hub repository	39
Figure 5.9.9	Download coconames from Git Hub	39
Figure 5.9.9.1	Terminal of Debian OS	41
Figure 5.9.9.2	The system is detecting speed of a toy car	42
Figure 5.9.9.3	The marking on the table	42
Figure 5.9.9.4	Measure the marking on the table	43
Figure 5.9.9.5	Shows the coding to edit meter per pixel	43
Figure 5.9.9.6	Shows the coding to edit width and height	43
Figure 5.9.9.7	Shows the coding to edit FPS	44
Figure 5.9.9.8	Shows the coding to edit source of camera	44
Figure 5.9.9.9	Shows the coding to edit smoothing factor	45
Figure 5.9.9.9.1	Shows the coding to edit vehicle counting timer	45
Figure 5.9.9.9.2	Shows the terminal of VFv2	46
Figure 5.9.9.9.3	Shows the OpenCV windows in VFv2	46
Figure 5.9.9.9.4	Shows the table with marking	47
Figure 5.9.9.9.5	Measure the marking distance	47
Figure 5.9.9.9.6	Show code to modify meter per pixel	48
Figure 5.9.9.9.7	Show code to modify resolution of the camera	48
Figure 5.9.9.9.8	Show code to modify FPS of the camera	49
Figure 5.9.9.9.9	Show code to modify resolution of the camera	49
Figure 5.9.9.9.9.1	Show code to modify smoothing factor	50
Figure 5.9.9.9.9.2	Show high cpu usage and program not responding	52
Figure 5.9.9.9.9.3	Vision Five v2 running Yolo with low fps	55
Figure 6.1	Testing location is at the housing area	58
Figure 6.2	Example of the testing setup	58
Figure 6.3	Shows the capture image of Raspi 4B	60
Figure 6.4	Shows the capture image of Raspi 4B detecting vehicle	60

Figure 6.5	Shows VFv2 detects the autogate and show speed	61
Figure 6.6	Shows VFv2 detects dry leaves and show speed	61
Figure 6.7	Shows VFv2 detects human movement and show speed	61
Figure 6.8	Shows VFv2 detects ant movement and show speed	61
Figure 6.9	False speed detected of the moving vehicle when auto gate moving	61
Figure 6.9.1	FPS captured in bright environment on Raspi	62
Figure 6.9.2	FPS captured in bright environment on VFv2	63
Figure 6.9.3	FPS captured in dark environment on Raspi	63
Figure 6.9.4	FPS captured in dark environment on VFv2	63
Figure 6.9.5	Shows the system tested in daytime with Raspi 4B	65
Figure 6.9.6	Shows the system tested in during night time with Raspi 4B	65
Figure 6.9.7	Shows the system tested in during day time with VFv2	66
Figure 6.9.8	Shows the system tested in during night time with VFv2	67
Figure 6.9.9	Shows the system unable to detect vehicle if obstacle blocking using Raspi 4B	68
Figure 6.9.9.1	Shows the system able to detect different type of vehicle using Raspi 4B	68
Figure 6.9.9.2	Shows the system able to detect two vehicle while calculate speed using Raspi 4B	69
Figure 6.9.9.3	Shows the system unable to detect speed of truck and van using Raspi4B	70
Figure 6.9.9.4	Shows the system overheating and skip frame on Raspi 4B	70
Figure 6.9.9.5	Shows any vehicle speed using VFv2 without Yolo	71
Figure 6.9.9.6	Shows the system unable to detect speed when 2 vehicle go different direction	71
Figure 6.9.9.7	Price tag for both SBCs	72
Figure 6.9.9.7	Image distortion when perform recording on VFv2	74
Figure 7.1	Raspberry PI AI HAT+	79

LIST OF TABLES

Table Number	Title	Page
Table 2.1	Strengths and limitations of the existing speed detection systems	9
Table 3.1	Use case Description	16-17
Table 5.1	Specification for Raspi 4B system	28
Table 5.2	Specification for VFv2 system	28
Table 6.1	Verification for Procedure 1	56
Table 6.2	Verification for Procedure 2	57
Table 6.3	Verification for Procedure 3	57
Table 6.4	Evaluation table between two SBCs	76

LIST OF SYMBOLS

LIST OF ABBREVIATIONS

<i>API</i>	Application Programming Interface
<i>CPU</i>	Central Processing Unit
<i>GPIO</i>	General Purpose Input Output
<i>IOT</i>	Internet of Things
<i>RAM</i>	Random Access Memory
<i>RaspPi</i>	Raspberry Pi
<i>SBC</i>	Single board computer
<i>VFv2</i>	Vision Five v2
<i>LIDAR</i>	Light Detection and Ranging
<i>YOLO</i>	You Only Look Once
<i>Webcam</i>	Web camera

Chapter 1

Introduction

Nowadays, more and more university students prefer driving to school or riding a bike instead of using public transportation. With the increased use of personal vehicles, over speeding has also become a common issue. To address this, a real-time speed tracking system has been developed to remind drivers when they exceed the speed limit. Using a camera equipped with an algorithm to calculate and predict vehicle speed, along with a large 7-segment LCD display on a signboard, the system informs drivers of their speed as they pass through a specific road section. The main objective of this project is to design and develop a vehicle speed tracking system that operates along the roadside.

1.1 Problem Statement and Motivation

Students exceeding speed limits, both inside and outside school zones, pose significant safety risks to pedestrians, cyclists, bike riders, and other road users. Despite existing speed regulations and law enforcement efforts, many students continue to drive over the limit, leading to an increased number of accidents and injuries. Some incidents occur within school areas where students drive too fast and fail to stop in time. Therefore, an effective monitoring system is needed to accurately detect and display vehicle speeds in these zones to reduce over speeding and enhance safety.

This project addresses the issue by developing a camera-based speed detection system that utilizes a single-board computer (SBC) to monitor and display vehicle speeds in real-time on a large 2-digit 7-segment display. The motivation behind this project is to create a cost-effective speed tracking system using image recognition technology with SBCs. Compared to existing traffic speed detection methods such as LIDAR-based systems (such as AES/AWAS speed cameras used on highways), SBCs offer a more affordable alternative. Given the large size of school zones, implementing AES/AWAS cameras across multiple areas would be costly. Additionally, school zones typically have multiple speed bumps and a speed limit of 30 km/h, while most over speeding vehicles in these areas travel between 30 km/h and 60 km/h. This

system serves as a practical alternative for detecting moderate vehicle speeds in school zones.

1.2 Objectives

The objective of this project is to analyze the current state of vehicle speed detection systems and explore the use of computer vision on single-board computers. The project will progress through the design and implementation phases, where the system will be developed using OpenCV and deployed on the Vision Five v2 and Raspberry Pi 4B platforms.

A significant part of the project will focus on optimizing system performance to ensure efficient operation within the resource constraints of the selected single-board computers. Additionally, a comparative analysis of the RISC-V and ARM architectures will be conducted to evaluate their respective strengths and weaknesses in handling machine vision tasks.

Extensive real-world testing will be carried out to validate the system's reliability and accuracy. The project will conclude with an assessment of its feasibility for broader deployment, considering factors such as cost, scalability, and integration with existing traffic monitoring infrastructure.

1.3 Project Scope and Direction

This project aims to design, develop, and evaluate a vehicle speed detection system using computer vision on two types of single-board computers: the VisionFive (RISC-V architecture) and Raspberry Pi (ARM architecture). The system will utilize OpenCV, an open-source computer vision library, for image detection and processing. The key objectives of the project are as follows:

1. **System Development** – Implement a vehicle speed detection algorithm using OpenCV, deploy and optimize the algorithm for both single-board computers, and integrate cameras to capture real-time video footage of vehicles.
2. **Performance Optimization** – Enhance machine vision algorithms to operate efficiently within the hardware constraints of the VisionFive and Raspberry Pi. Additionally, compare the processing capabilities and speed detection accuracy between the RISC-V and ARM architectures.

3. **Validation and Testing** – Conduct extensive testing under various environmental conditions, including different lighting, weather, and traffic densities. Validate the accuracy of the speed detection system by comparing its results with standard speed measurement tools. Furthermore, analyze the system's real-time performance, including frame rates, detection accuracy, and processing latency.
4. **System Evaluation** – Assess the effectiveness, cost-efficiency, and scalability of the system for potential deployment in traffic monitoring. Evaluate the advantages and limitations of using Vision Five (RISC-V) versus Raspberry Pi (ARM) for this application.

1.4 Contributions

The development of a vehicle speed detection system using machine vision on single-board computers like Vision Five v2 (RISC-V) and Raspberry Pi (ARM) has the potential to revolutionize traffic monitoring. This low-cost, portable, and efficient solution for speed detection could significantly enhance road safety, reduce speeding-related accidents, and improve traffic law enforcement in both urban and rural areas. The affordability and accessibility of single-board computers make this system highly scalable and deployable, even in regions with limited resources such as electricity. As a result, it offers a practical and cost-effective alternative to traditional speed detection systems, which are often expensive to implement and maintain.

This project holds substantial value in the fields of computer vision, embedded systems, and transportation technology. It contributes to ongoing research on machine vision applications in resource-limited environments, particularly on emerging architectures like RISC-V. Additionally, by utilizing open-source tools such as OpenCV in real-time applications, the project showcases the capability of affordable hardware platforms to handle complex tasks like vehicle speed detection. Furthermore, by comparing the performance of Vision Five v2 and Raspberry Pi 4B, this study provides valuable insights into the suitability of different architectures for machine vision tasks, guiding future developments in this area.

A key contribution of this project is the comparison of RISC-V and ARM architectures in the context of vehicle speed detection. Benchmarking their performance will generate useful data for future research and development efforts in

embedded machine vision. The project's findings and system prototype could serve as a foundation for the deployment of low-cost, scalable speed detection systems in real-world environments.

Additionally, this project serves as an important case study for students and researchers in computer vision, embedded systems, and transportation technology. By demonstrating the practical application of theoretical concepts in a real-world scenario, it provides valuable insights for researchers and developers looking to optimize their systems for specific hardware platforms.

1.5 Report Organization

The information about Vehicle Speed Detection System are shown in the following chapters. In Chapter 2, previous work and project backgrounds are reviewed. Then, method and approach are presented in Chapter 3, which also include the calculation and formula for this project. Next, Chapter 4 describes system block diagram and the component hardware use for the project. Chapter 5 shows the proper setup for the speed detection system and software configuration. Chapter 6 shows the overall system evaluation and result discussion. Finally, Chapter 7 reports the conclusion and summary of this project.

Chapter 2

Literature Review

2.1 Previous works on Speed Detection System

In the research of speed detection system, numerous methods have been found. This section shows review of three current systems that has been implement commonly at the highway such as LIDAR detection system, Inductive loop sensors system, and video-based speed detection system.

2.1.1 Radar and LIDAR-Based Systems

One of the earliest methods for speed detection involves the use of radar and LIDAR (Light Detection and Ranging) technology use laser beams to precisely measure distances and determine vehicle speeds [4]. It has been widely used due to their ability to accurately measure the speed of vehicles from a long distance without direct contact. It can even target specific vehicles in dense traffic. However, the systems are often expensive and require manual operation, making them less suitable for continuous monitoring in school zones where most of the car driving with medium speed.

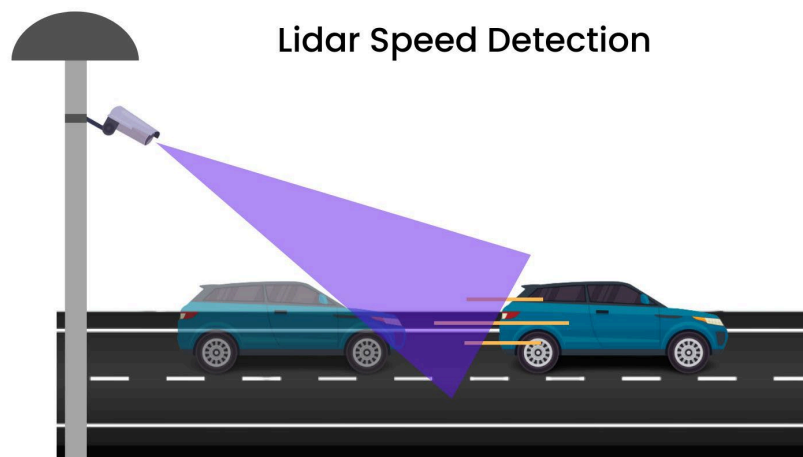


Figure 2.1 LIDAR speed detection

2.1.2 Inductive Loop Sensors

Inductive loop sensors, this systems consist of wire loops embedded in the road surfaces, are another traditional method used for vehicle speed detection. These sensors measure the speed of vehicles by detecting the time it takes for a vehicle to pass over two or more loops placed at known distances apart [6]. Although inductive loop sensors provide accurate speed measurements and are reliable, their installation and maintenance costs are high, Moreover, this method also face drawbacks including complex installation, high maintenance expenses, and potential damage to road surfaces over time. Additionally, they are typically restricted to monitoring vehicles at fixed locations, which may not provide comprehensive traffic data. They are not suitable to be deploy in temporary or changing environments like school zones.

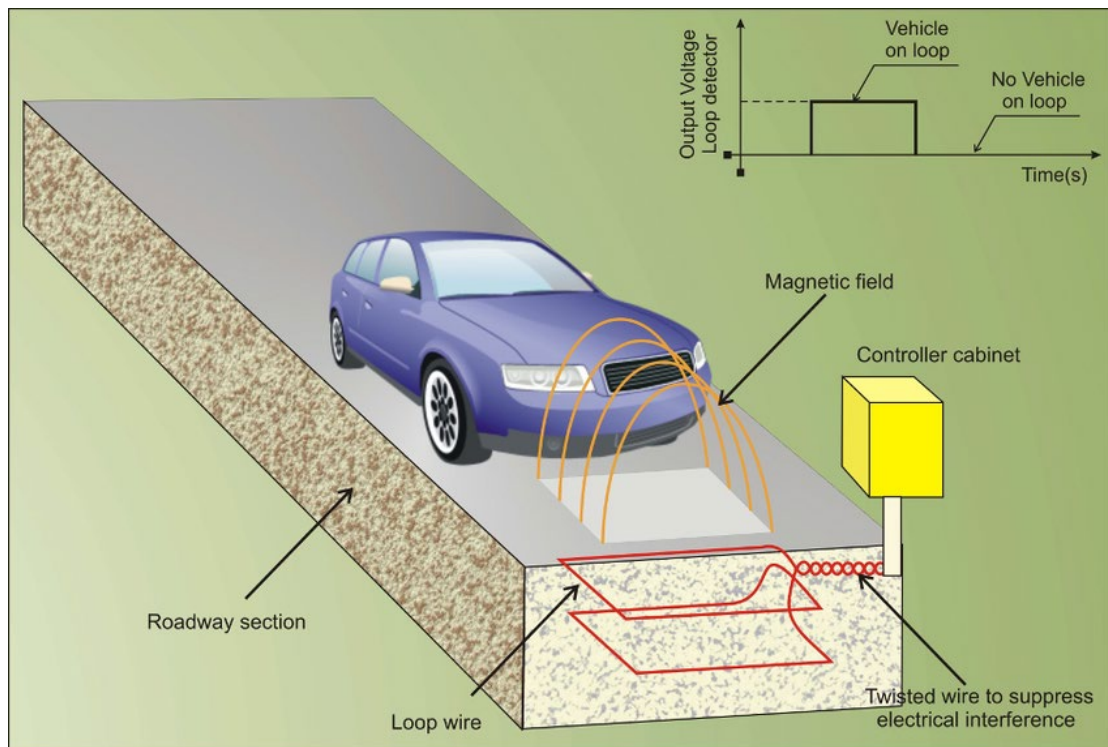


Figure 2.2 Inductive loop sensor

2.1.3 Video-Based Speed Detection Systems

Video-based speed detection systems use camera and image processing techniques to monitor vehicle speed. These systems capture video footage and analyze it to identify vehicles, track their movement, and calculate speed based on the distance traveled over time [7].

Video-based systems are advantageous because they can monitor multiple lanes of traffic simultaneously and provide visual evidence of speeding violations. This method is also more flexible than inductive loops and can be easily integrated with existing traffic cameras. However, challenges such as varying lighting conditions and the need for accurate calibration can impact their performance.

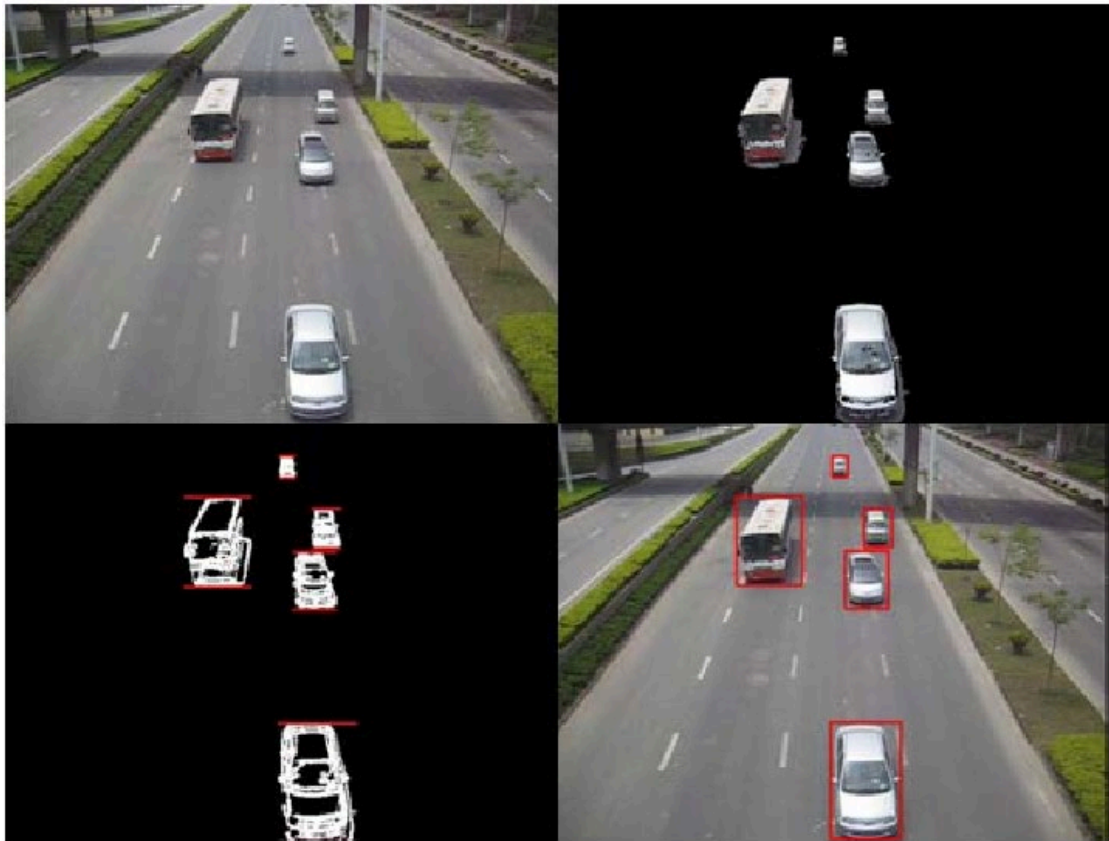


Figure 2.3 Video based speed detection system

2.2 Limitation of Existing Systems

2.2.1 Limitation on Radar and LIDAR-Based Systems

High cost, the purchase and maintenance of these systems are quite expensive, which is not possible to massively implement it in school zones. Radar guns need manual operation by law enforcement, limiting their utility for continuous monitoring [6].

Coverage is limited, the systems can only monitor one lane or vehicle at a time, which may not be sufficient for multi-lane roads and during crowded sessions.

2.2.2 Limitation on Inductive Loop Sensors

The installation requires roadwork, and maintenance can be challenging, especially in areas with frequent traffic or environmental wear. Permanent location, once installed, these sensors are fixed in place and cannot be easily relocated or adjusted, reducing their flexibility. The installation also requires significant amount of time, it will cause traffic jam especially in school area during installation.

The sensor are ineffective for off-road surface, such as bicycle riding zones and the sideway of the road.

2.2.3 Limitation on Video Based Speed Detection System

Video-based systems can be affected by different light conditions, weather, and camera positioning, which may reduce accuracy. For example, lowlight or raining conditions which might cause false positive detection. To ensure accurate speed detection, careful calibration of the camera angle, frame rate, and distance is needed, which can be technically challenging. Real-time video processing requires great amount of computational resources, weaker hardware might not be able to process the video in real time and cause delay.

2.3 Comparison Of Previous Works

Application	Strengths	Limitations
Radar and LIDAR-Based Systems	<ul style="list-style-type: none"> - High accuracy - Proven technology - Real-time Monitoring - Long range 	<ul style="list-style-type: none"> - High Cost - Manual Operation - Limited Coverage
Inductive Loop Sensors	<ul style="list-style-type: none"> - Accurate Measurements - Low False Positives - Reliable 	<ul style="list-style-type: none"> - High Installation and Maintenance Costs - Long installation time - Damage road surface when deploy - Fixed Location - Limited to Road Surface
Video-Based Speed Detection Systems	<ul style="list-style-type: none"> - Flexible Deployment - Multiple Lane Monitoring - Visual Evidence - Utilize existing camera infrastructure 	<ul style="list-style-type: none"> - Environmental Sensitivity - Complex Calibration - Processing Power Requirements

Table 2.1 Strengths and limitations of the existing speed detection systems

Table 2.1 provides summary of the strengths and limitations of three existing speed detection systems. In conclusion, each method of speed detection has its own strengths and weaknesses, making the choice of method dependent on the specific requirements of the application. For school zones, where cost, flexibility, and real-time monitoring are an important factor, video-based systems offers a more promising balance of features.

Chapter 3 System Methodology/Approach

System Development Methodology

For this project, the Waterfall Model was used as the system development methodology. The Waterfall Model follows a linear and sequential approach, where each phase is completed before moving to the next. This method was chosen due to its structured approach, which ensures that all requirements, design, and implementation steps are carefully planned and executed. Waterfall model was chosen because I have a clear scope and fixed requirements which is to detect the vehicle's speed.

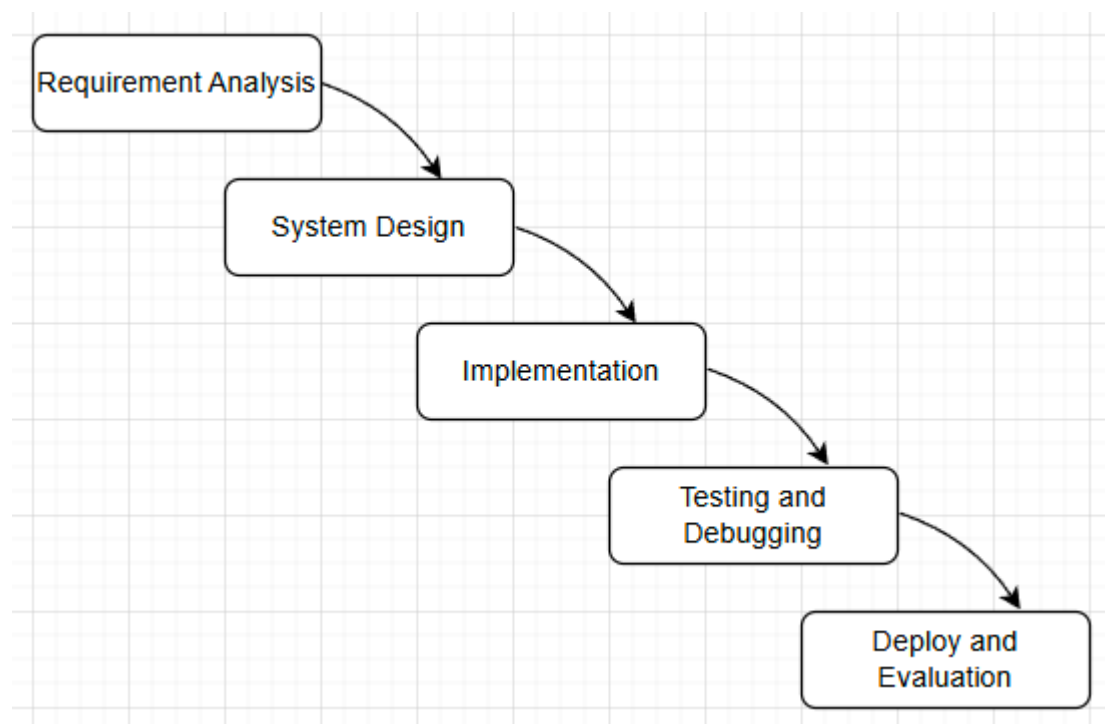


Figure 3.1 Waterfall model

In the first phase, the requirement is analyzed by deciding how the system should be built. The system needs to detect vehicles, estimate the speed and then display results on LCD display. SBCs which is cost effective had been chosen. Challenges such as low processing power, software limitations were considered. An open-source Speed Detection program is downloaded online to test the effectiveness and analyze the pro's and con's of the system. On the system design phase, because the result given by the open-source Speed detection system is not satisfying due to lack of object detection, I have decided to make a Speed Detection System based on YOLO tiny, where it can be configured to only capture vehicle object. Multithreaded is also being implemented to improve FPS and real time performance.

GPIO integration was designed for TM1637 7-segment display to output speed. The system uses pixel movement and time difference to do calculations for the vehicle speed.

During the implementation phase, the video processing pipeline was developed. I integrate some functions such as background subtraction to increase accuracy, multithreaded video capture class was implemented to optimize real time performance. The TM1637 display module was integrated using RPI GPIO to show detected speed. Speed estimation function was fine tuned to convert pixel movement into real world speed.

For Vision Five v2, the TM1637 display module is integrated with GPIOB because RPI GPIO only supports Raspi SBC.

Next, the testing and debugging phase where I test the system under different lighting conditions, vehicle speeds and distances. Issue such as false detection when vehicle is idling in the detection range caused by noise, low FPS issues were fixed.

On Vision Five v2, due to CPU performance issue, the YOLO object detection is being removed to get an acceptable frame rate. Background subtraction and motion tracking is used so that the SBCs could at least give some working results.

Finally, the last phase which is deployment and evaluation. The final system was deployed and real world tests were conducted. The system will be tested with toy car and real car. Limitation of the SBCs were analyzed.

3.1 System Design Diagram/Equation

3.1.1 System Architecture Diagram

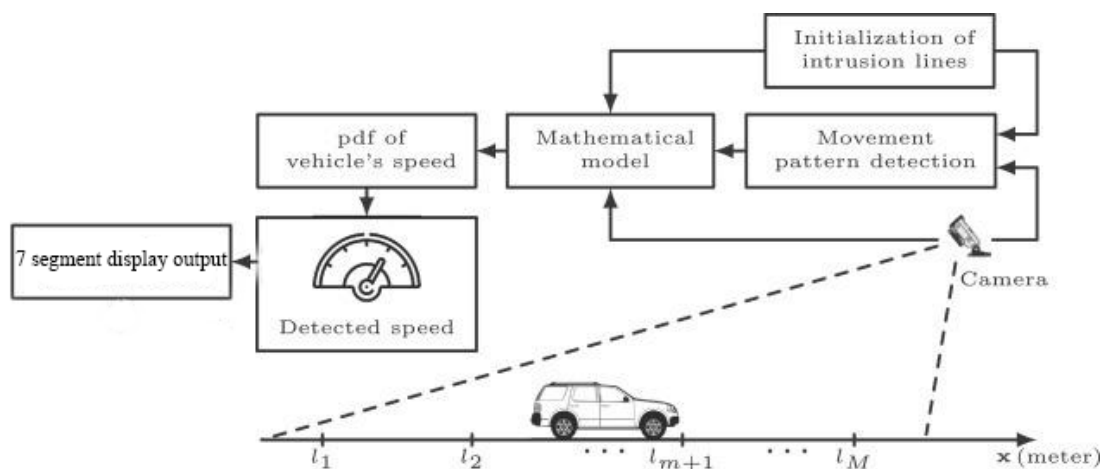


Figure 3.2 System Block Diagram

The overall system developed consists of two parts, hardware and software. The hardware used in this system will be Raspberry Pi that connects USB webcam and 7segment LCD to support the solution. In this report, the detailed system design will only cover the system deployment. For the first step we will need to set up the system by connecting the peripheral to the Raspberry Pi and switch it on. After entering the OS, we would need to tune the intrusion lines, users would need to enter distance per pixel in meter. The speed will be calculated when the vehicle passes through.

The camera will capture the video data at resolution of 320x240 at 30fps to match the load of the CPU. Higher resolution will reduce the framerate which will increase processing time and reduce the accuracy, it can also reduce the detection rate.

Then, we launch OpenCV to perform movement pattern detection, and by using mathematical model implements inside the code, the Raspberry PI should be able to calculate the actual speed of the vehicle when the vehicle pass through the intrusion lines.

Finally, the result of the speed calculated will be output to the 7segment display LCD to show the speed of the driver.

Input : the real time raw video data collect from the webcam

Output : the speed of the vehicle data

Camera mounting point is an important factor which needs to be configured carefully in order to get an accurate speed reading of the vehicle.

Camera Mounting Position

Below is the scientific equation for the vehicle Speed Calculation with OpenCV when the camera is mounted at the side of the road and is 90degree towards the incoming vehicle. This method of mounting provides an easy calculation to the system. But due to the hardware we use which only support up to 30fps and only 78degree FOV, if the vehicle passing through the detection range is travelling at a very fast speed, the system could miss the object detection. For example, if the vehicle travels too fast, the camera manages to capture a single frame of the vehicle, but unable to capture the second frame of the vehicle, it will not output a speed result. The calculation is shown below :

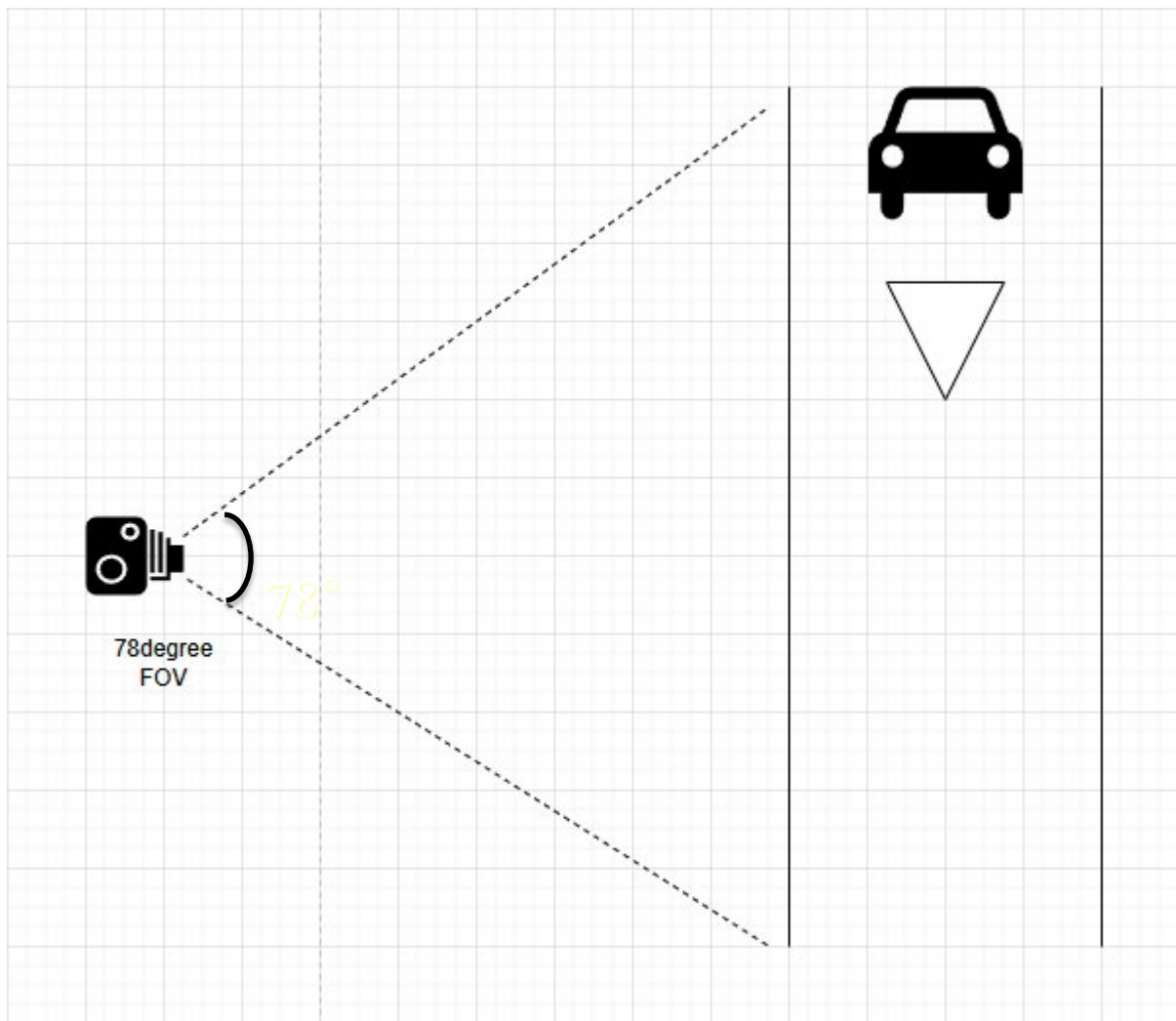


Figure 3.3 Camera Mounting Point 1

Variables and Assumptions:

- **Vehicle Length (L):** 4366 mm (4.366 m) real car, toy car 13.5cm
- **Webcam Field of View (FOV):** 78 degrees
- **Frame Rate per second (fps):** 30fps
- **Pixel width = 800**
- **Pixel height = 240**

First, we need to divide meter by pixel to know how long it is for each pixel. We get the scaling result.

Meter per pixel :

Meter / pixel = scale

CHAPTER 3

Next, we calculate distance by calculating the coordinate of the object every 2frame, and multiply the scale of the pixel.

Distance calculation :

(x1,y1) and (x2,y2) are the coordinates of the object's position in pixels at two different frames.

Scale is the conversion factor of pixel distance to meter (meter per pixel)

Formula is distance (m) = $\sqrt{(x2-x1)^2 + (y2-y1)^2} * \text{scale}$

Formula is shown below :

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2} * \text{scale}$$

Next, we calculate time taken for the system to capture the object. We find the different between the 2 frame and divide by frame captured per second.

Time calculation :

n1, n2 = number of frame between two position

f = fps

time = $|n2-n1| / \text{fps}$

formula is shown below :

$$t = \frac{|n2 - n1|}{f}$$

Finally, we calculate predicted speed using formula

Speed = distance / time

V = speed velocity

$V = (\sqrt{(x2-x1)^2 + (y2-y1)^2} * \text{scale}) / (|n2-n1| / \text{fps})$

$$v = \frac{\sqrt{(x2 - x1)^2 + (y2 - y1)^2} * \text{scale}}{\frac{|n2 - n1|}{f}}$$

Lastly, we convert the meter per second to kilometer per hour :

$$\text{Speed km/h} = \text{speed m/s} * 3.6$$

3.1.2 Use Case Diagram and Description

The use case diagram for this system include 3 actor which is user, system and vehicle that pass through. User can make configuration for the system by setting the meter per pixel of the detection range, the fps of the detection system, resolution of the camera for video input. User can adjust the smoothing factor to match lighting conditions to improve accuracy. User can input the source of the camera to capture video data. User can exit the program by pressing q. The SBCs is responsible to detect vehicle movement and calculate speed, then output the result to the 7-segment display. The vehicle that travels through the detection range will be detected.

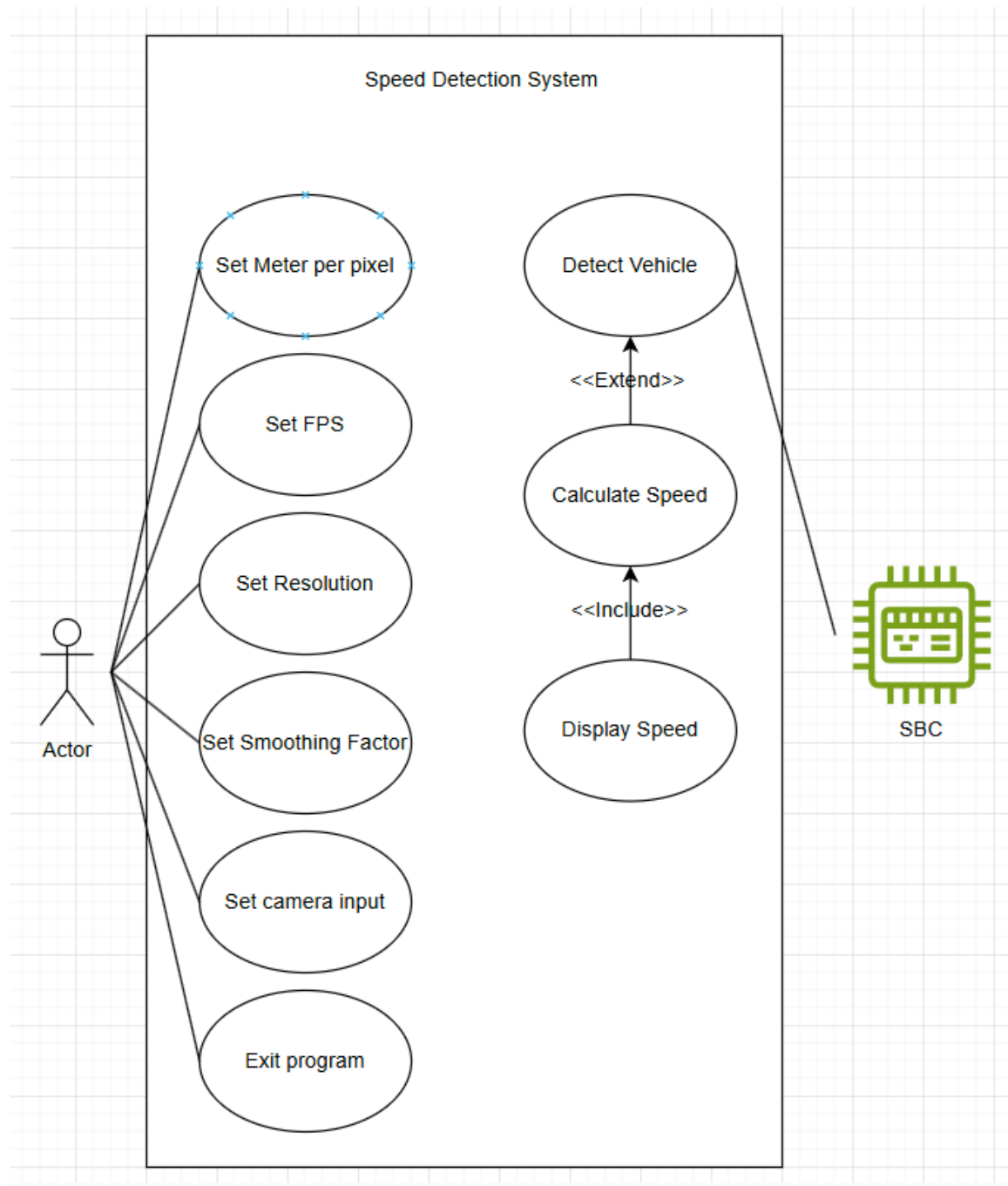


Figure 3.7 Use case diagram for Vehicle Speed Detection System

Use case description

Use case name	Vehicle Speed Detection System Using SBC
Use case description	How the system detects and calculates the speed of vehicles passing through a monitored area (e.g., school zones) using a camera mounted on different angle. The system captures real-time video, processes vehicle movement, calculates speed, and displays the detected speed on a 2-digit 7-segment display.
Actor	Operator
Precondition	<p>The camera and SBC must be properly set up and calibrated.</p> <p>The system must be powered on and running the detection software.</p> <p>The camera's field of view should cover the target road area.</p>
Main Flow	<ol style="list-style-type: none"> 1 Start session <ol style="list-style-type: none"> 1.1 Operator power up the SBC. 1.2 Operator calibrates the system by entering the meter per pixel of the detection range. 1.3 Operator enter the command into terminal to launch the program. 2 Operation process, image capturing <ol style="list-style-type: none"> 2.1 The camera captures a continuous video feed of the road. 2.2 The system applies motion detection and object tracking to identify moving vehicles. 2.3 The system detects a vehicle and tracks its displacement between two frames. 2.4 The system will record video of the vehicle that passes through. (VFv2 only) 2.5 The system will count the vehicle that passes through the detection (Raspi 4B only) 3 Calculation process, data processing <ol style="list-style-type: none"> 3.1 Using pixel displacement and time interval, the system calculates the real-world speed. 3.2 The computed speed is converted into km/h. 4 Display Result

	<p>4.1 The speed is displayed on the 2-digit 7-segment display for driver awareness.</p> <p>5 Reset Display Result</p> <p>5.1 The system will reset the display result after the vehicle leaves the detection range.</p> <p>5.2 The system will stop recording after the vehicle leaves the detection range.</p>
Alternative Flow (Edge cases & error)	<ol style="list-style-type: none"> 1. Vehicle partially visible - The system waits for the vehicle to be fully in view before measuring speed. 2. Multiple vehicles detected - The system applies object tracking to focus on the primary moving vehicle in the lane. 3. Low light conditions - Image enhancement techniques are applied to improve detection. 4. Camera shake or misalignment - Operator has to perform periodic recalibration for the camera using reference markers.
Post Conditions	<ul style="list-style-type: none"> - The vehicle's speed is successfully measured and displayed. - The system will automatically perform recording when it detects vehicle that pass through. - The system will stop recording after the car leaves the detection zone and continues monitoring the next vehicle. - The system will count the vehicle that pass through
Exception	<ul style="list-style-type: none"> - If the system fails to detect moving objects, it will not display any speed and will not perform recording. - If the system fails to detect moving objects, it will not count it as a vehicle. - If the camera is removed from the system, the program will not launch and will give error. - The operator should use Q to terminate the program.
Author	Goh Jin Yu

Table 3.1 Use Case Description

3.1.3 Activity Diagram

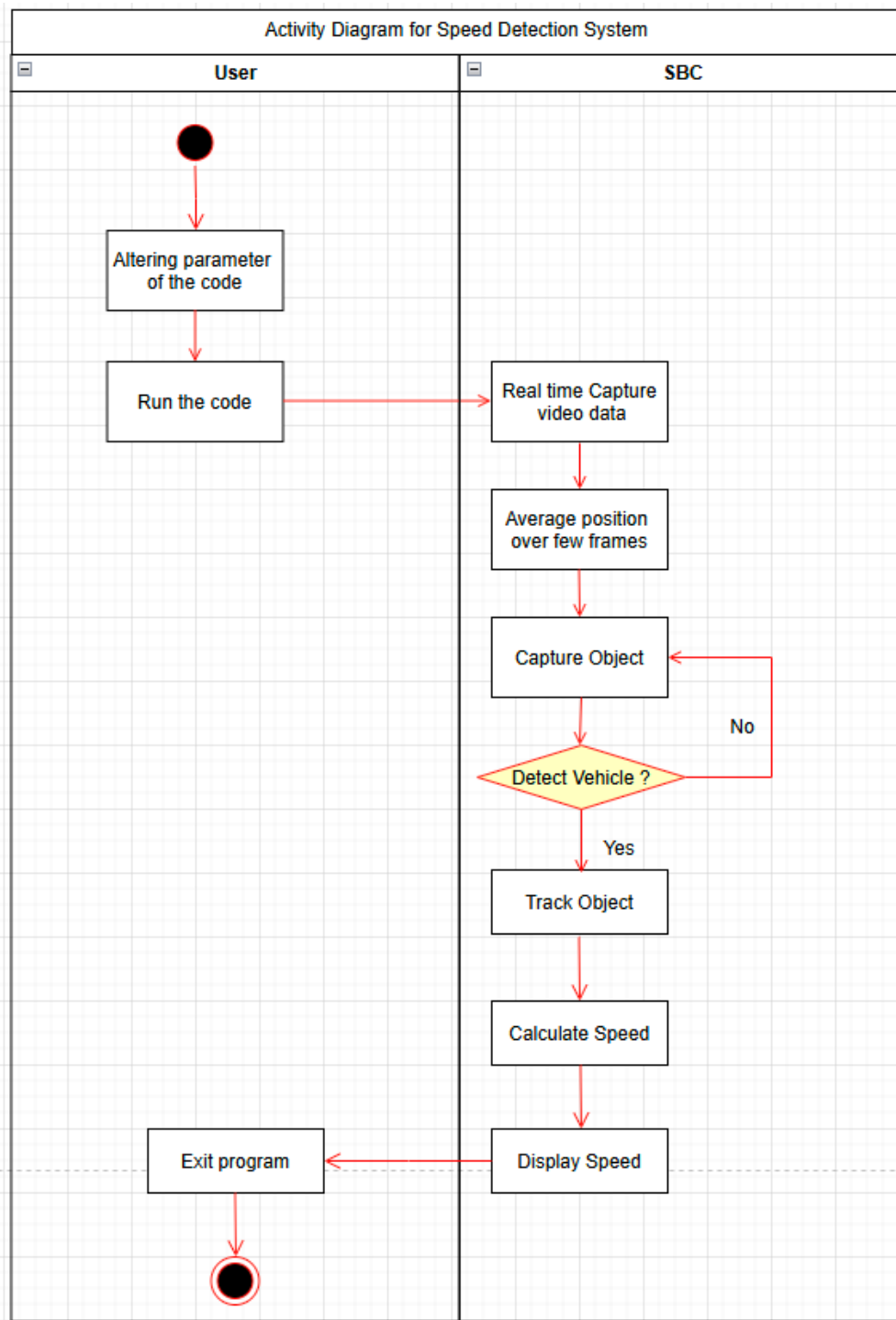


Figure 3.8 Activity Diagram of Speed Detection System

First, the user would need to provide distance per pixel in meter and enter into the code before launching the application. When launching the application, OpenCV and Yolo v4 tiny will execute. The webcam will capture and output the video data for detection. First, the system will average position over few frames, this is to solve the false speed detection issue

due to noise on low light condition. The pixel positions due to minor fluctuations of noise which lead to low detection accuracy. This causes an unmoving toy car to show speed results. Implementing this algorithm enhances the result by averaging positions over a few frames. Next, Yolo v4 tiny will capture vehicle object based on trained dataset. It will only detect vehicles such as cars, buses, bikes, trucks and bicycles. This further enhances the precision compared to OpenCV based detection where it will give false speed detection when conditions such as multiple vehicles pass by or when human walks beside the sideroad and one vehicle pass by. Yolo v4 tiny will only track the object when it detects that the object is a vehicle before performing speed calculations. After performing calculations from the above formula, it will output the speed result on the 7segment display.

3.2 Timeline

FYP1 timeline

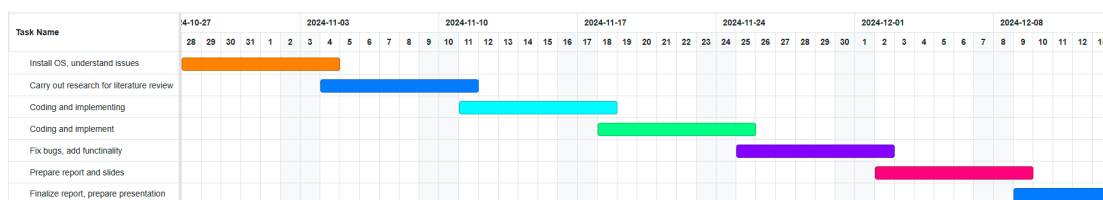


Figure 3.9 Gantt Chart FYP1

In Week 1, I have installed the Linux Book Worm OS for the raspberry PI and tested the pre-written software from GitHub with real life testing and understand the current development and issue faced by the project. On week 2, I perform research to find the best methods and tools that can solve the issue such as false detection and algorithm that can perform accurate vehicle detection and do a literature review. In week 3, block diagram are drawn for rough idea of the method that show how the system works and compile all the documentation and report. In week 4, I start to have better understanding of the vehicle speed detection implementation method and start to perform coding. The code works and it is able to detect the speed of a toy car with some minor issue. In week 5, I spend time to optimize the code to make it run more efficiently and smoothly, I also added 7segment display support for the system to display speed of the object. During week 6, I continue to prepare the report and slide for presentation and also continue to research more method to optimize the system. By week 7, the report will be finalized for submission and for the presentation.

FYP2 timeline

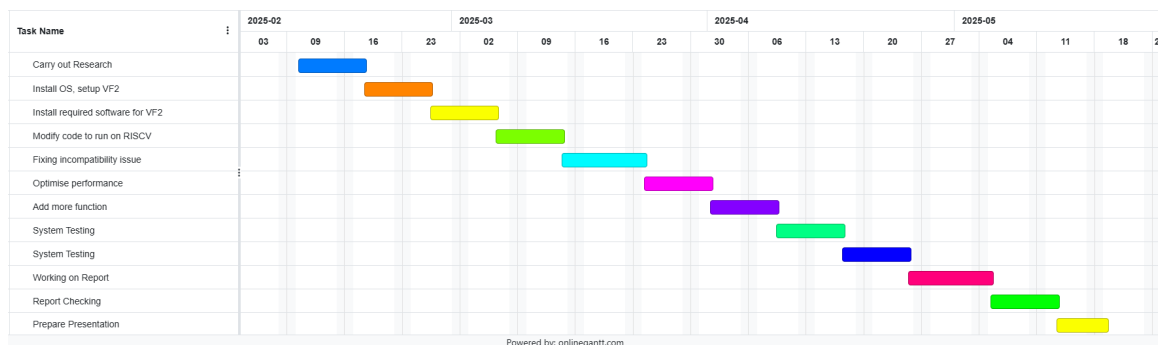


Figure 3.9.1 Gantt Chart FYP2

In Week 1, I carried out research for the Vision Five V2, which include surfing the forum for more info and finally decide to use Debian which is officially provided by StarFive, and understand the current development and issue faced by the project when using openCV. The whole week 2, I perform an installation of the Debian OS, and internet set up for the VF2. In week 3, I try to install required software for the VF2 which is needed to run the Speed Detection Program. In week 4, I start to modify the code in order to make it run on VF2. The code face a massive amount on incompatibility issue. On week5, the code is able to work after going through lots of troubleshooting and code fixing. In week 6, I spend time to optimize the code to make it run more efficiently and smoothly, VF2 runs very slow when using Yolo Object Detection. During week 7, I continue to add more functions into the system. On week 8, I perform system testing on toy car. On week9, I perform system testing on real vehicles around my housing area. During week 10, I am preparing the report and continuing to optimize the system for improvement. On week11, I checked the report format before finalizing. By week 12, the report will be finalized for submission and start preparing for the presentation.

CHAPTER 4 System Design

4.1 System Block Diagram

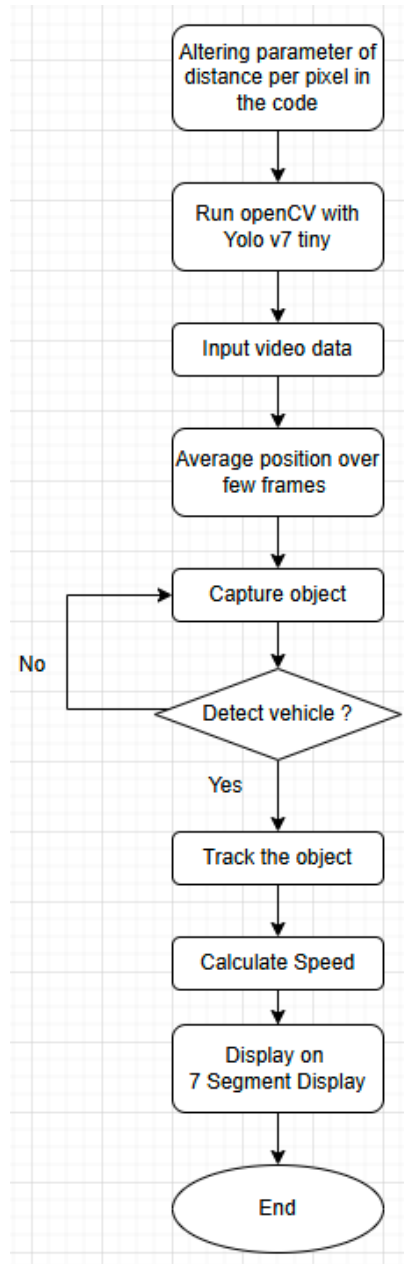


Figure 4.1 Block Diagram of Speed Detection System

To address the disadvantage of existing speed detection systems, this project proposes an innovative and cost-effective solution that aim to enhances traffic speed monitoring in sensitive areas such as school zones. The system is designed to utilize the adaptability of video-based detection, integrated with real-time processing on a single-board computer, to address the specific challenges of student over speeding in and around school areas.

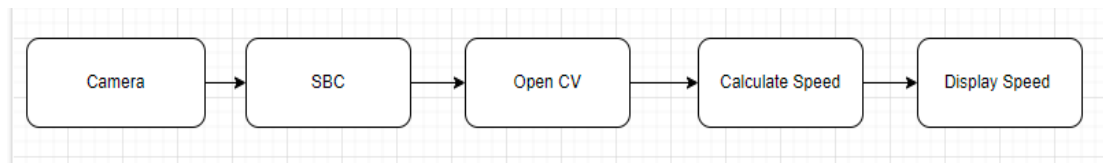


Figure 4.2 General Work Procedure

4.2 System Components Specifications

1. Web Camera or Camera Module is used to continuously captures video of the roadway, providing the raw data needed for vehicle detection and speed analysis. The placement and orientation of the camera are important to ensure a clear view of the vehicles. A camera module or USB webcam is used for video capture input. These cameras are selected due to cost effective consumer hardware and good compatibility with the selected SBC and their ability to capture high-resolution images at sufficient frame rates. The system can be configured to capture video at different resolutions depending on the required detection accuracy and available processing power.

2. Single-Board Computer for Real-Time Processing. For this project we uses a single-board computer (SBC), such as the Raspberry Pi and VisionFive v2 to perform on-site processing of the camera feed. The SBC handles all computational tasks, including object detection, tracking, and speed calculation[8], without relying on external processing such as servers or cloud-based services. This decentralized approach ensures that the system is both cost-effective and scalable. SBC is chosen due to its affordability, low power consumption, and sufficient processing capabilities. The SBC also connected with other peripheral devices like webcam or camera module and the 7-segment display.

3. OpenCV, which is an Open-Source Software and allow Scalability, the system is built using open-source software and open source OS, this is to ensure that it can be easily customized, extended, and scaled. The use of open-source libraries like OpenCV allows for ongoing development and community contributions, making it possible to adapt the system for different use cases or integrate it with other traffic monitoring solutions [2]. OpenCV, form the core of the software stack. The system architecture is designed to be modular, allowing for the addition of new features or integration with larger monitoring systems. The scalability of the solution is

achieved through the ease of replication and deployment across multiple locations, with minimal setup requirements.

4. Calculate speed, user would need to provide the approximate distance(m) per pixel of the camera's field of view, so that the system could calculate the speed of the vehicle travelling through the detection range. User could also change the resolution and fps in the setting based on the camera model's specification they use.

5. Display speed using 2-Digit 7-Segment Display. After calculating the vehicle's speed, the system displays the result on a large 2-digit 7-segment display. This instant feedback serves as an alert for speeding drivers by making them aware of their current speed. The display is simple yet effective, providing real-time information in a highly visible format. The 7-segment display is connected through GPIO pins of the SBC.

4.3 Circuits and Components Design

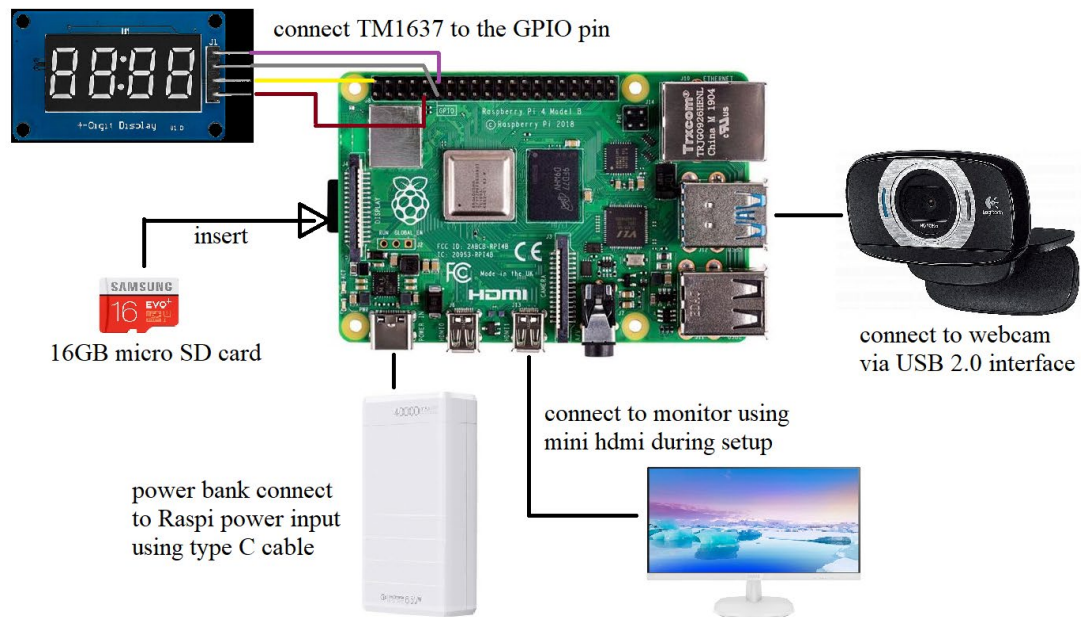


Figure 4.3 Circuit and Component design for Raspi 4B

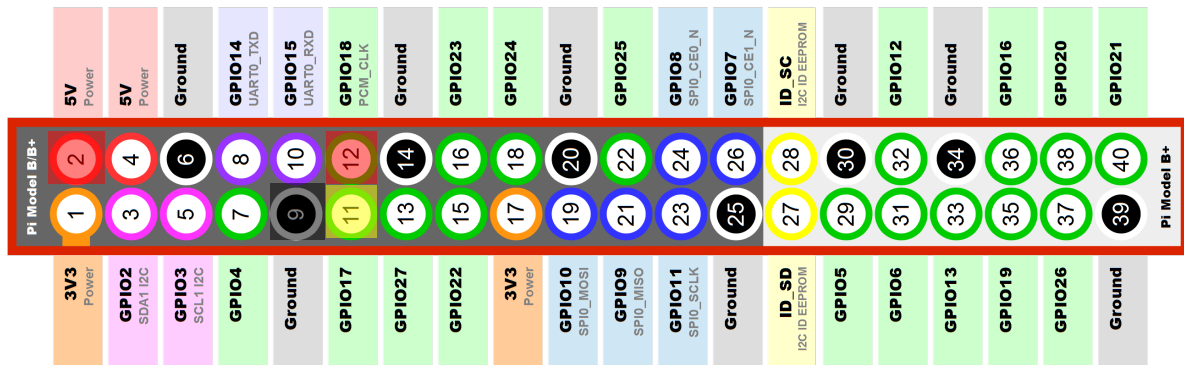


Figure 4.4 GPIO diagram for Raspberry Pi 4B

The GPIO pin connected to the Raspberry PI 4b is pin2, pin9, pin11 and pin12. Pin2 is 5v power and pin9 is ground pin used to power up the TM1637. While the pin11 (GPIO17) and pin12 (GPIO16) is the clock pin to control the LED of the TM1637.

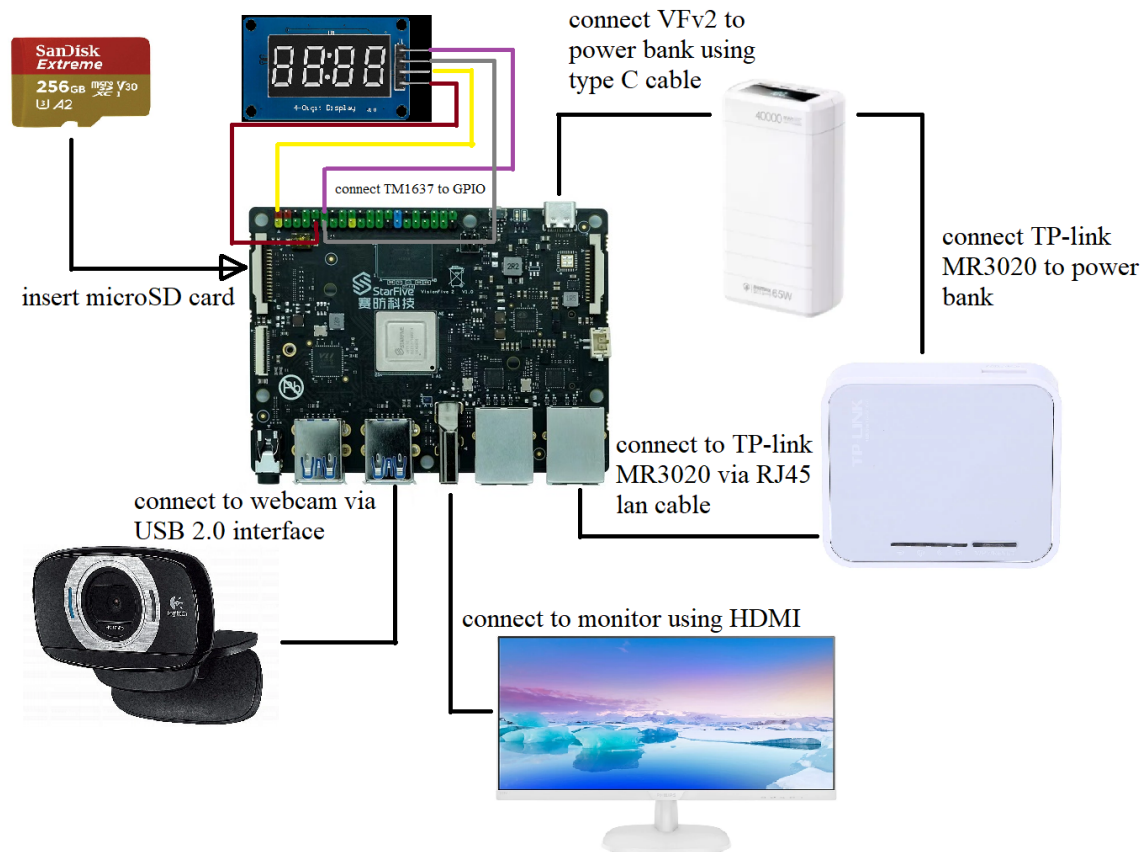


Figure 4.5 Circuit and Component design for Vision Five V2

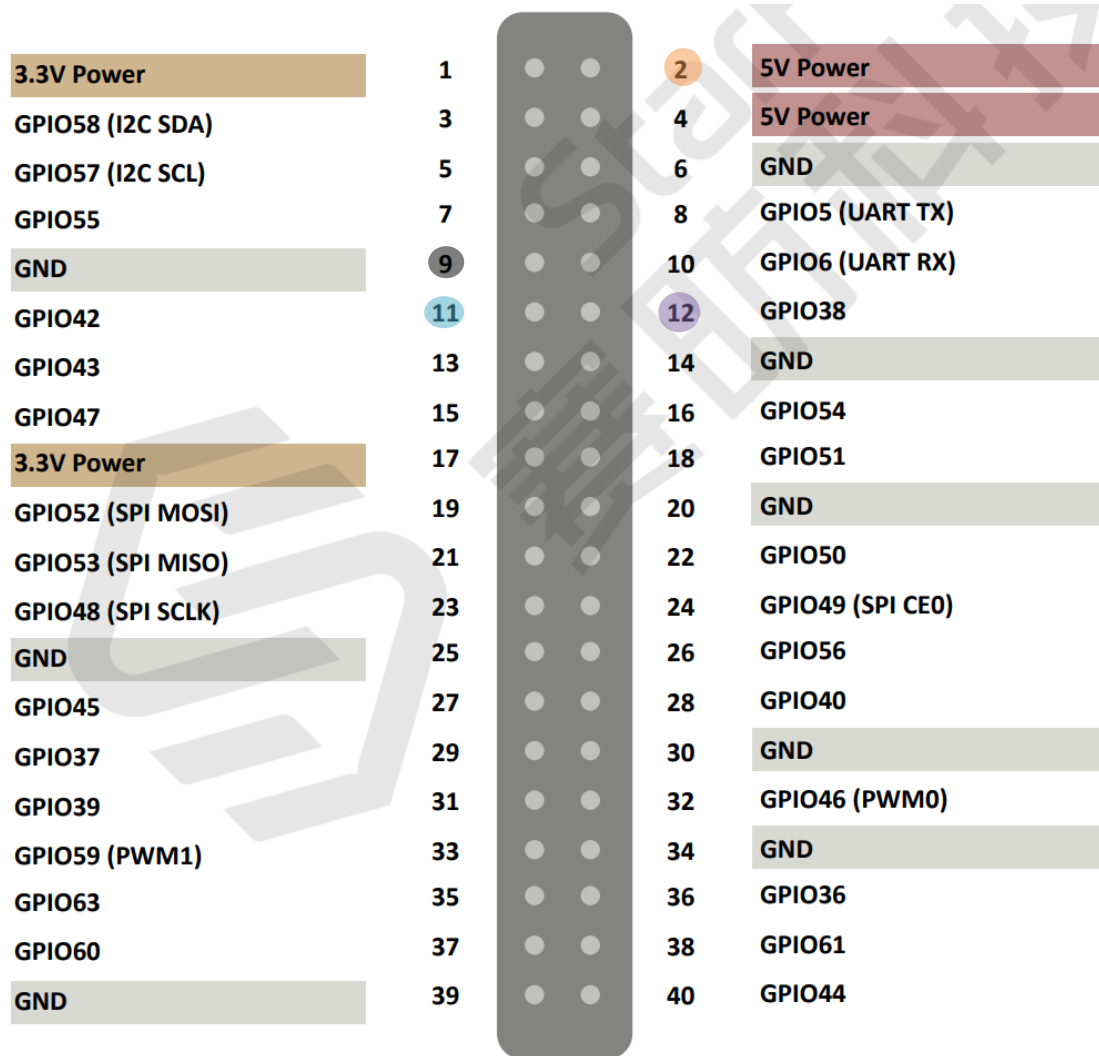


Figure 4.6 GPIO diagram for Vision Five v2

The GPIO pin connected to the Vision Five v2 is pin2, pin9, pin11 and pin12. The GPIO pin out diagram is actually same as the Raspberry PI GPIO layout. But the configuration of the GPIO ID is different. Pin2 is 5v power and pin9 is ground pin used to power up the TM1637. While the pin11 (GPIO42) and pin12 (GPIO38) is the clock pin, where it generate signal to control the LED of the TM1637.

4.4 System Components Interaction Operations

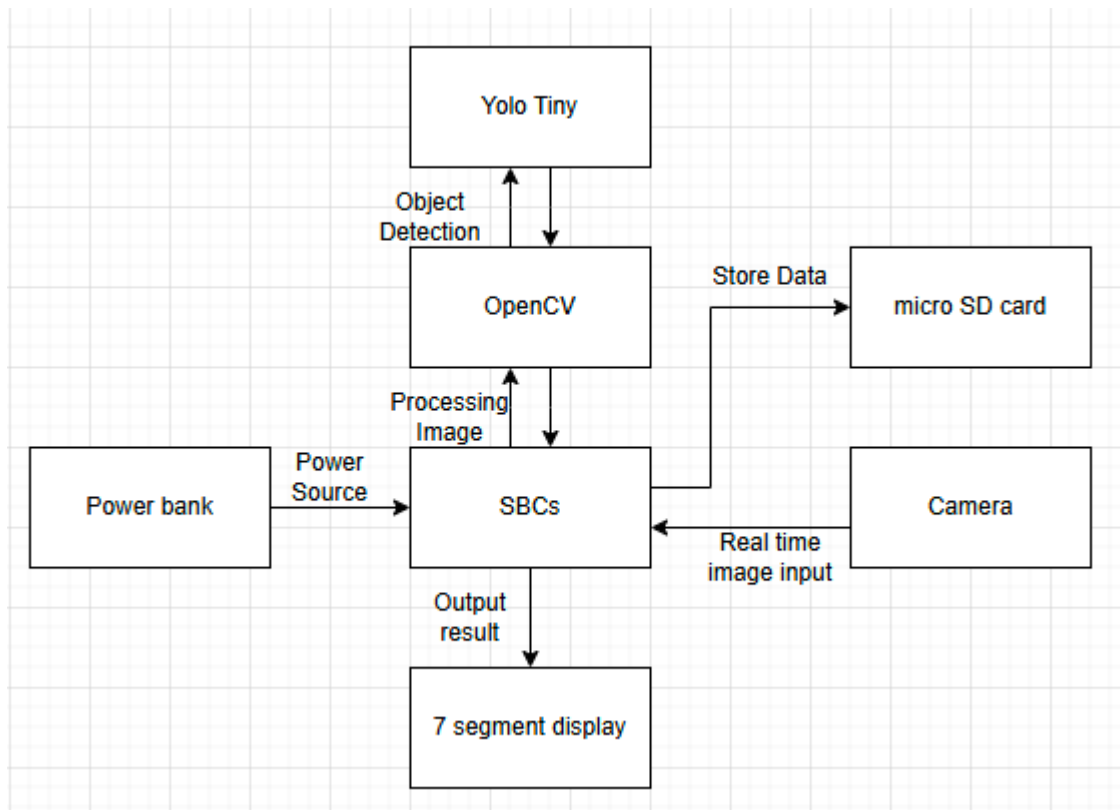


Figure 4.7 System Components Interaction Operations for raspberry PI

In this project, power bank is used to power up the SBCs so that we can mount the speed detection system more flexible without need to rely on power socket. Webcam is an image sensor and is used to capture real time image information to the SBCs in order to process the data. It is being connected via USB2.0 interface. The micro SD card is to store OS needed for the SBCs to boot and operate, and also to allow SBC to store data. Last but not least, the TM1637 is configured to show speed of the vehicle that pass through the detection line.

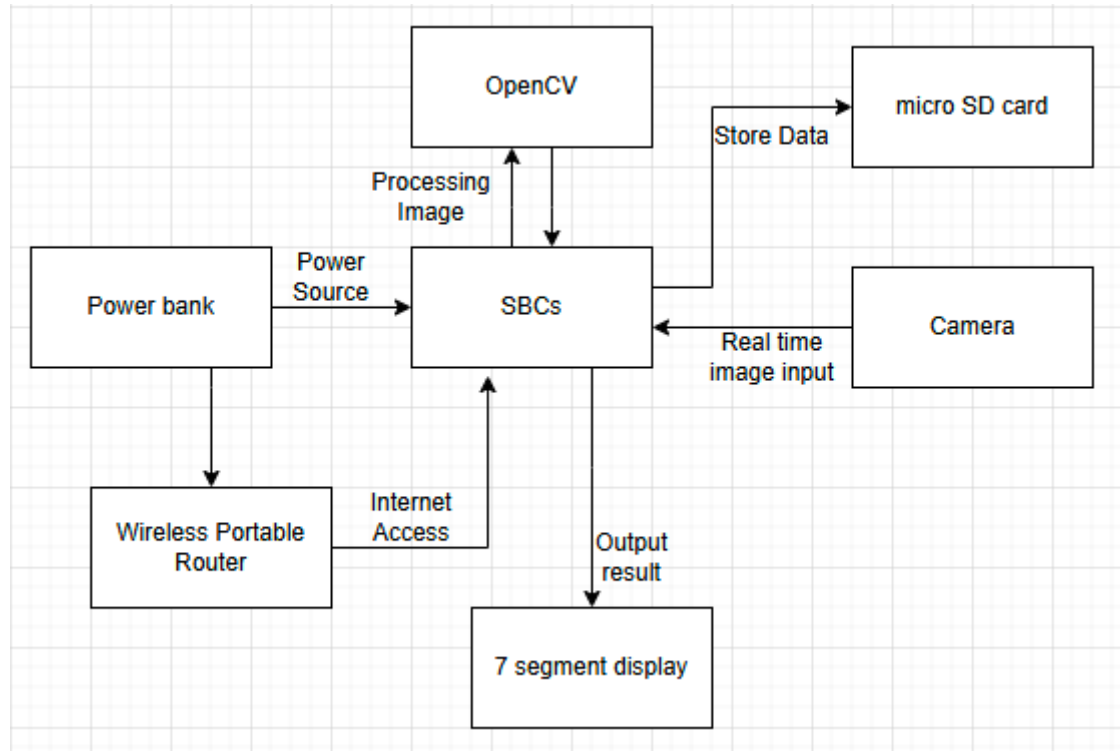


Figure 4.8 System Components Interaction Operations for Vision Five v2

For Vision Five v2, most of the setup is almost identical like the Raspberry Pi 4b, but due to no WiFi hardware build into the SBCs itself, we need an external wireless portable router to access to internet, which is the TP-link MR3020. This portable router also need an external power to run, we will need to connect the power input to the power bank to operate.

CHAPTER 5

System Implementation

5.1 Hardware Setup

The Raspberry Pi 4B and Vision Five v2, the main hardware involved in this project is a single board computer (SBC). This system will leverage OpenCV, an open-source computer vision library, for real time image detection and processing. It is used for testing and deploying the traffic speed detection system.

Description	Specifications
Model	Raspberry PI 4 Model B
Processor	Broadcom BCM2711 4Core Cortex A72@1.8Ghz (ARM v8)
Operating System	Linux Debian Book Worm OS
Web Camera	Logitech Webcam C615 (1080p30fps 78degree FOV)
Memory	8GB LPDDR4-3200 SDRAM
Storage	16GB C10 Samsung Evo micro SD card
7 Segment Display	TM1637 4digit 7segment display 0.36inch
Power	Remax 65w 40000Mah Power Bank

Table 5.1 Specifications of Raspi 4B system

Description	Specifications
Model	Vision Five v2
Processor	Star Five JH7110 64Bit SoC with RV64GC @1.5Ghz Quad Core
Operating System	Debian OS
Network	TP-Link MR3020 Portable 3G 4G Wireless N Router
Web Camera	Logitech Webcam C615 (1080p30fps 78degree FOV)
Memory	8GB LPDDR4 SDRAM
Storage	256GB C10 UHS-I SanDisk Extreme Micro SD card
7 Segment Display	TM1637 4digit 7segment display 0.36inch
Power	Remax 65w 40000Mah Power Bank

Table 5.2 Specifications of VFv2 system

CHAPTER 5

Hardware Setup

We will need to config the hardware, first we connect the hardware to the SBC. The Logitech c615 is connected to the SBC using USB interface. Next, we connect the 7segment display TM1637 through GPIO port, it has 4pins which is VCC, GND, CLK, DIO connected to pin2(5v power), pin 9(ground), pin12, pin11. After connecting all the peripherals, we then connect the raspberry PI to power source and it will boot up automatically. Both SBC, RaspberryPi and VisionFive v2 share the same GPIO pinout configurations.

Example of Hardware setup :



Figure 5.1 Example of system setup



Figure 5.2 Example of system setup2

5.2 Software Setup (Raspberry Pi)

First, we use Raspberry Pi Imager on our laptop to install Raspberry Pi OS into a microSD card. Raspberry Pi OS was selected as the operating system due to its compatibility with the Raspberry Pi hardware and its extensive community support. The OS is lightweight, optimized for ARM architecture, and includes pre-installed tools for Python programming, and remote access. The Raspberry Pi imager can be downloaded from this link :

<https://www.raspberrypi.com/software/>

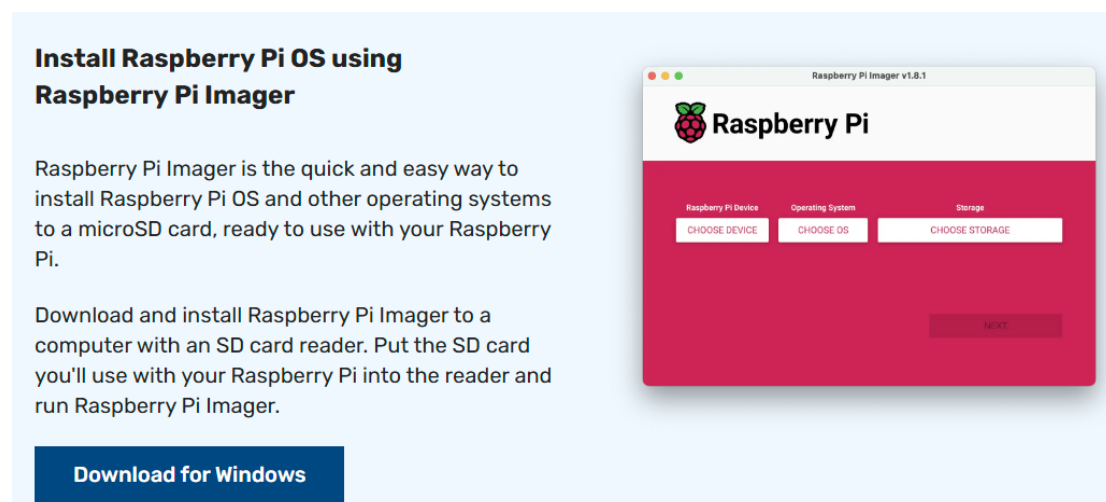


Figure 5.3 Official download page of Raspberry Pi

After downloading the software, we connect an SD card reader to the laptop and launch the Raspi Imager :



Figure 5.4 Main page of Raspi Imager

We click choose OS, then choose Raspberry PI OS

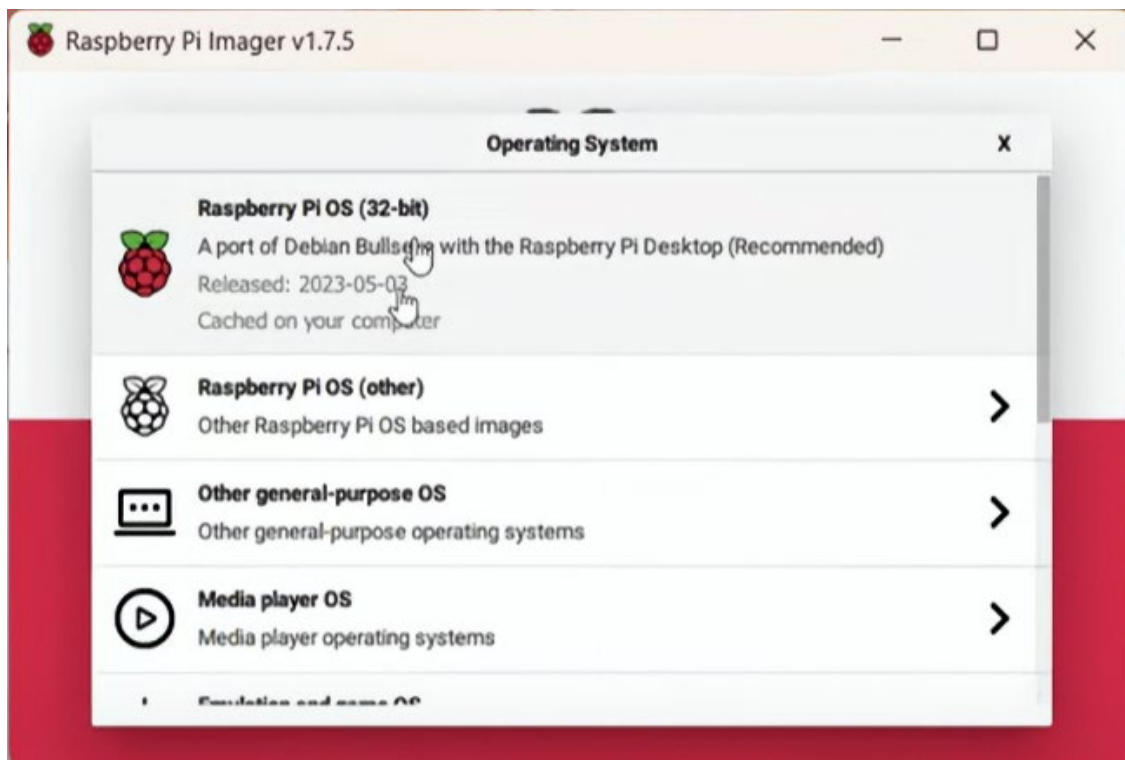


Figure 5.5 Selecting OS

Next, we choose storage and select our microSD card

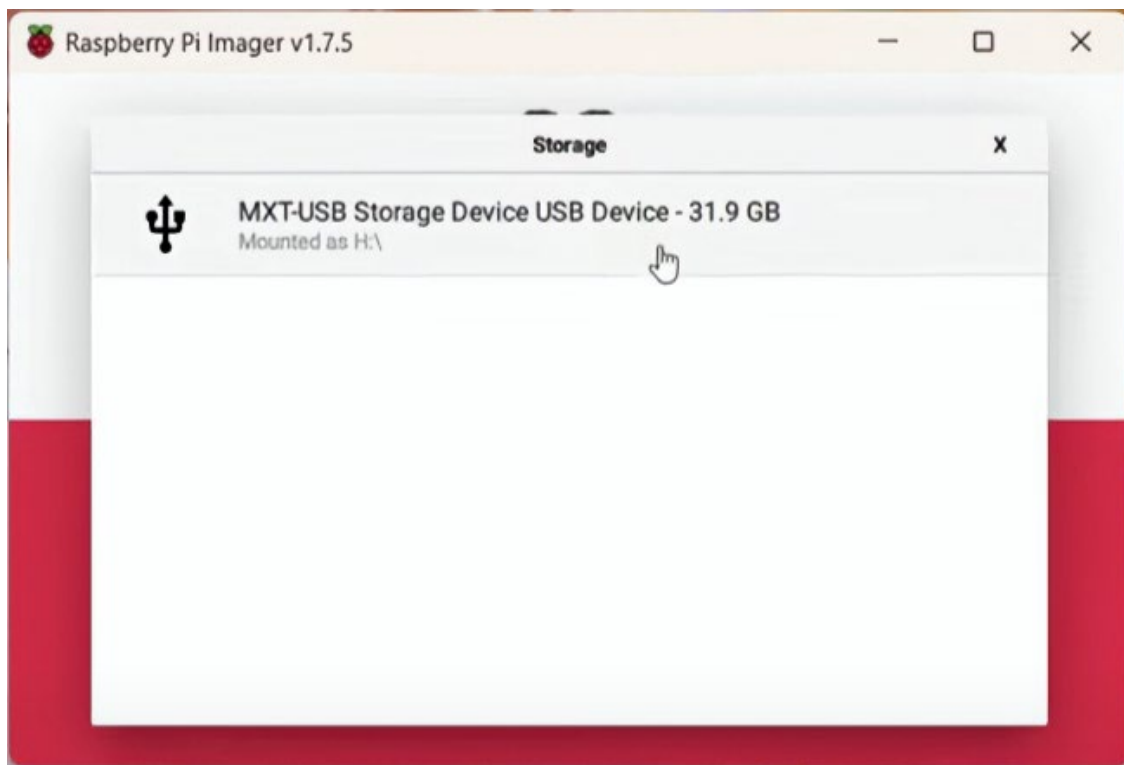


Figure 5.6 Select storage device

We then click on the write button to install Raspi OS onto the SD card.

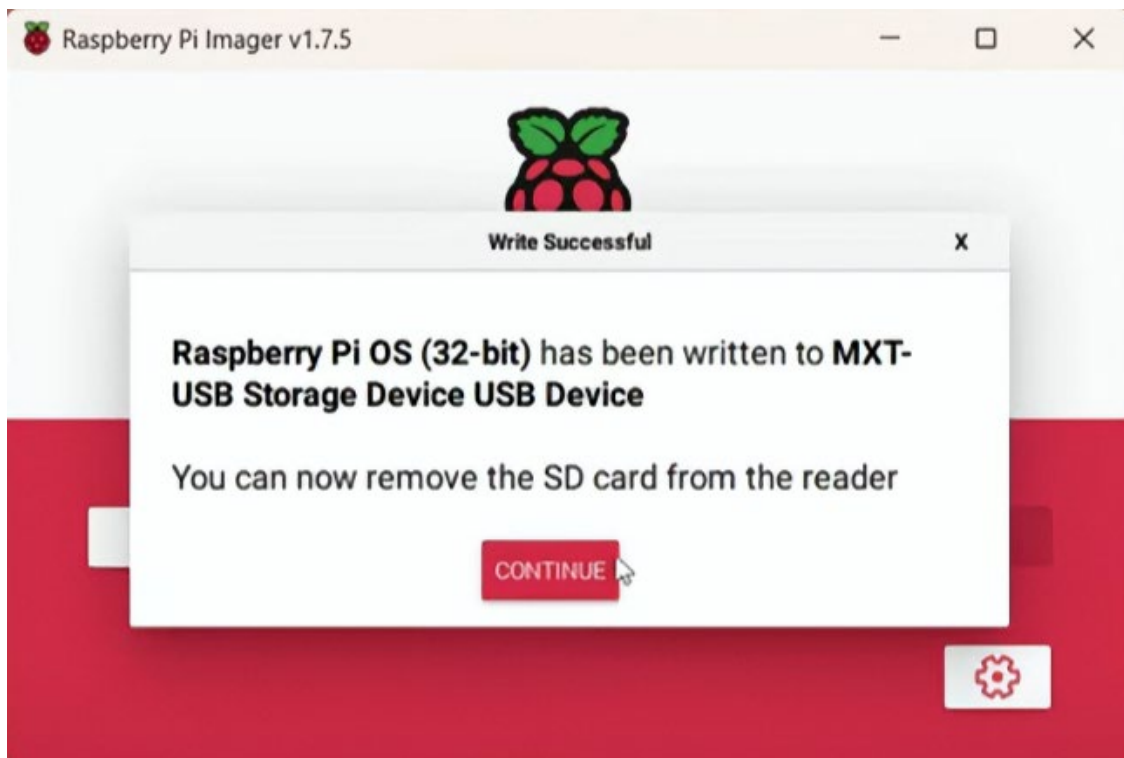


Figure 5.7 Installation complete

After the OS install successfully, we then remove the microSD card from the laptop and insert into the Raspberry Pi and boot up by connecting the PI to a power source.



Figure 5.8 Inserting micro SD into Raspi 4B

We can first setup the PI Connect so that later we can use laptop to remotely access the PI within the same network environment without any monitor and other peripherals like mouse and keyboard connect to the Raspi. The PI Connect also supports copy and paste function from different platform, for example copy the code from the laptop to the PI directly and vice versa, which is a very convenient function during coding sessions.

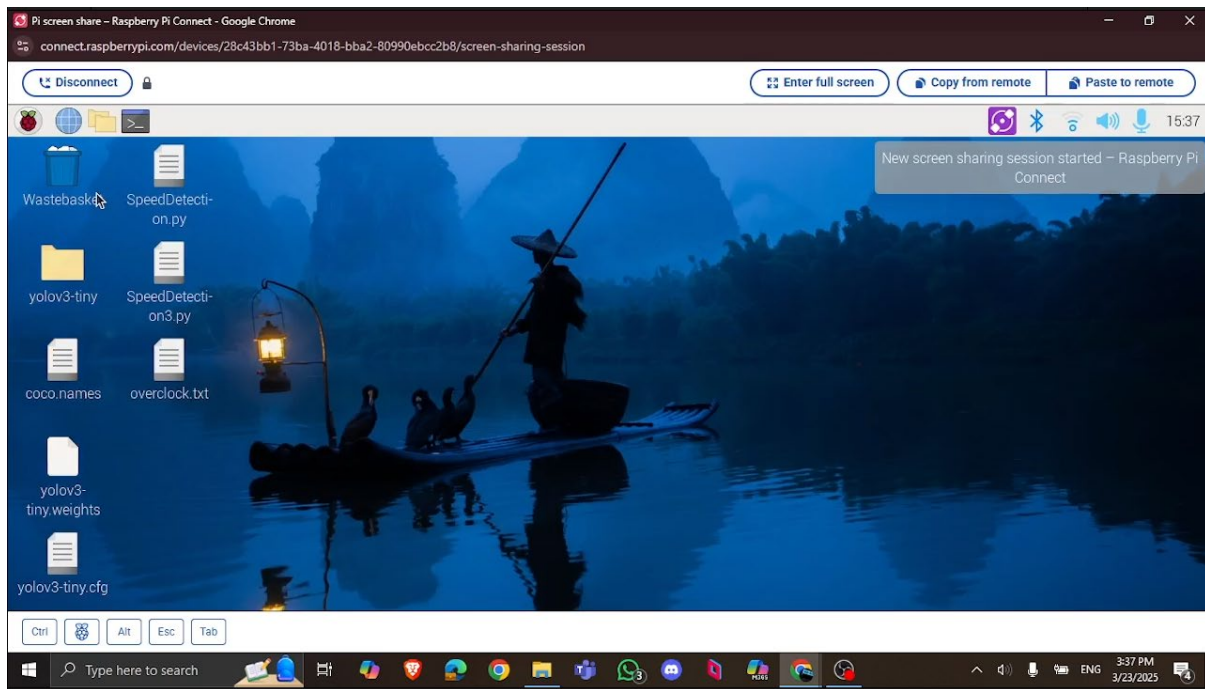


Figure 5.9 Pi connect

5.2 Software Setup (VF v2)

Star Five Debian OS can be download from this onedrive link :

<https://onedrive.live.com/?authkey=%21AAAs7oQT2992Eg8&id=8DAB77C937089CE9%21316442&cid=8DAB77C937089CE9>

Since we are using SD card as storage for VFv2, we choose the latest release, and click the sd folder and download the image file.

Engineering Release > 202409				
	Name ↑ ▾	Modified ▾	File size ▾	Sharing
	emmc	9/30/2024	1.00 GB	Shared
	flash	9/30/2024	3.02 MB	Shared
	nvme	9/30/2024	1.02 GB	Shared
	sd	9/30/2024	1.02 GB	Shared
	install_full.sh	9/30/2024	858 bytes	Shared
	Release_Notes_for_Debian_Image_202409.p...	9/30/2024	35.4 KB	Shared
	SHA256SUM.txt	9/29/2024	376 bytes	Shared
	StarFive-debian-repo.key	9/30/2024	3.15 KB	Shared
	StarFive-Debian-Upgrade.sh	9/30/2024	3.27 KB	Shared

Figure 5.9.1 VFv2 OS download page

Next, We use Win32Disk Imager on our laptop to install Debian OS into a microSD card.

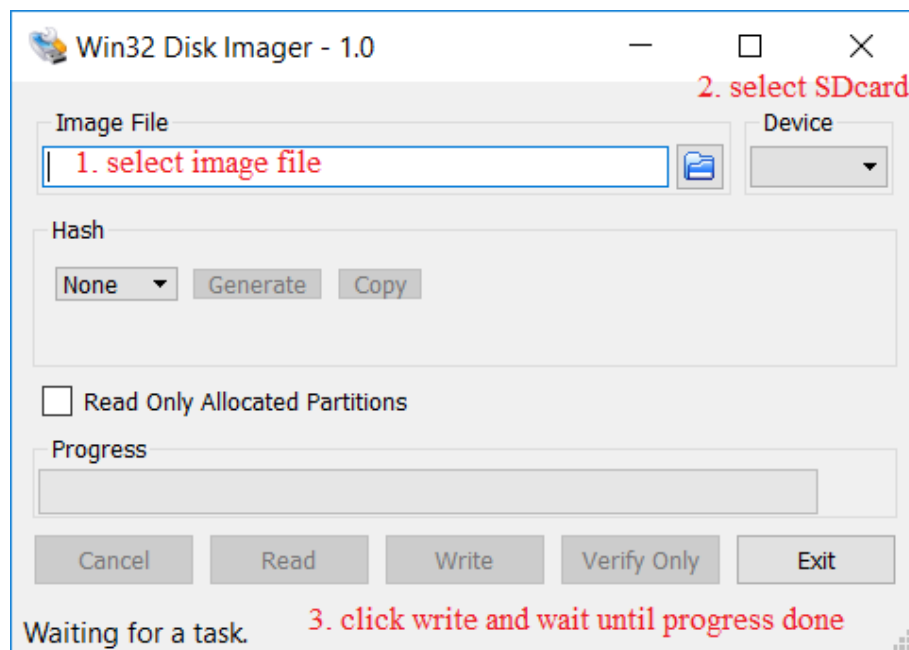


Figure 5.9.2 Disk Imager

Debian OS was selected as the operating system due to its official support with the Vision Five v2 hardware and its community support from RISC-V forum. The OS is lightweight, optimized for RISC-V architecture, and includes pre-installed tools such as Mozilla Firefox browser and Python. After the OS install successfully, we then remove the microSD card from the laptop and insert into the VisionFive v2 and boot up. We need to connect Vision Five v2 to an external monitor to continue the setup process because it does not have any remote access functionality build into the OS, we also need to connect other peripherals like keyboard and mouse to setup the system. I strongly recommend to stick a heatsink to the VFv2's processor to reduce the temperature, because the CPU runs hot during heavy loads. During testing, a heatsink is attached for a prolonged period of testing.

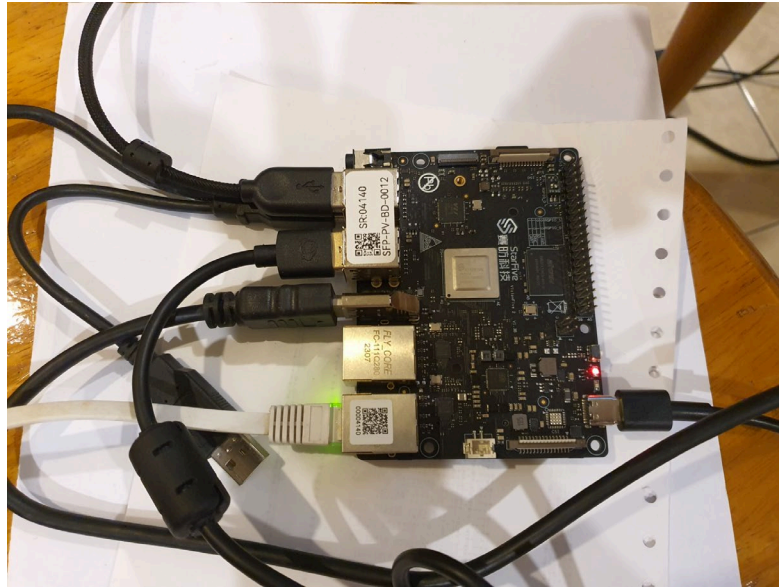


Figure 5.9.3 Connecting all the peripherals to the VFv2

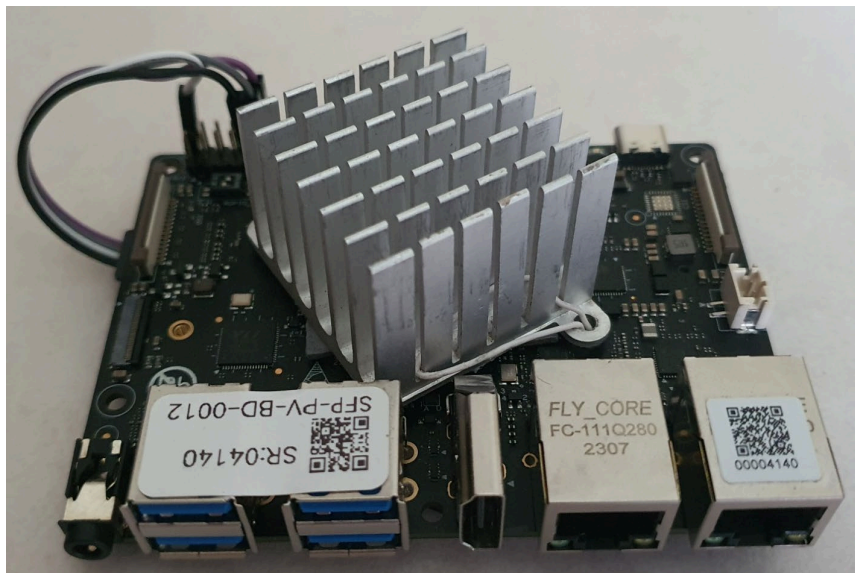


Figure 5.9.4 Attached heatsink on the CPU of VFv2

Vision Five v2 does not have any WiFi chipset prebuild, so a USB WiFi dongle or LAN is needed in order to connect to the internet. We would need to connect and set up the TPLINK portable WiFi router using a browser. First, we adjust the router's physical switch to WISP mode, we enter the IP address of the portable router into the browser which is 192.168.0.254, we enter the username and password provided from the router's sticker. After logging in, we navigate inside the user interface of the portable wifi router to connect WiFi.

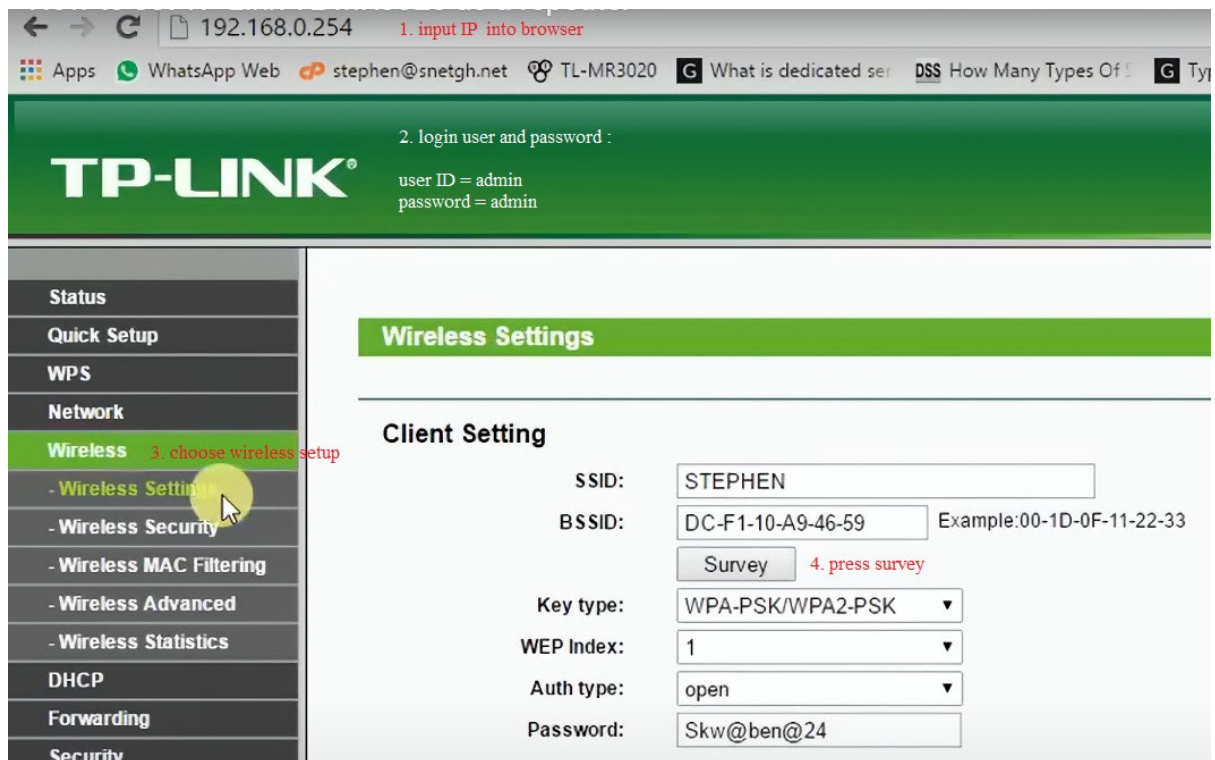


Figure 5.9.5 TPlink router setup user interface

After entering the UI, we choose wireless setup, then click the survey button to search for our WIFI network, we then click connect.

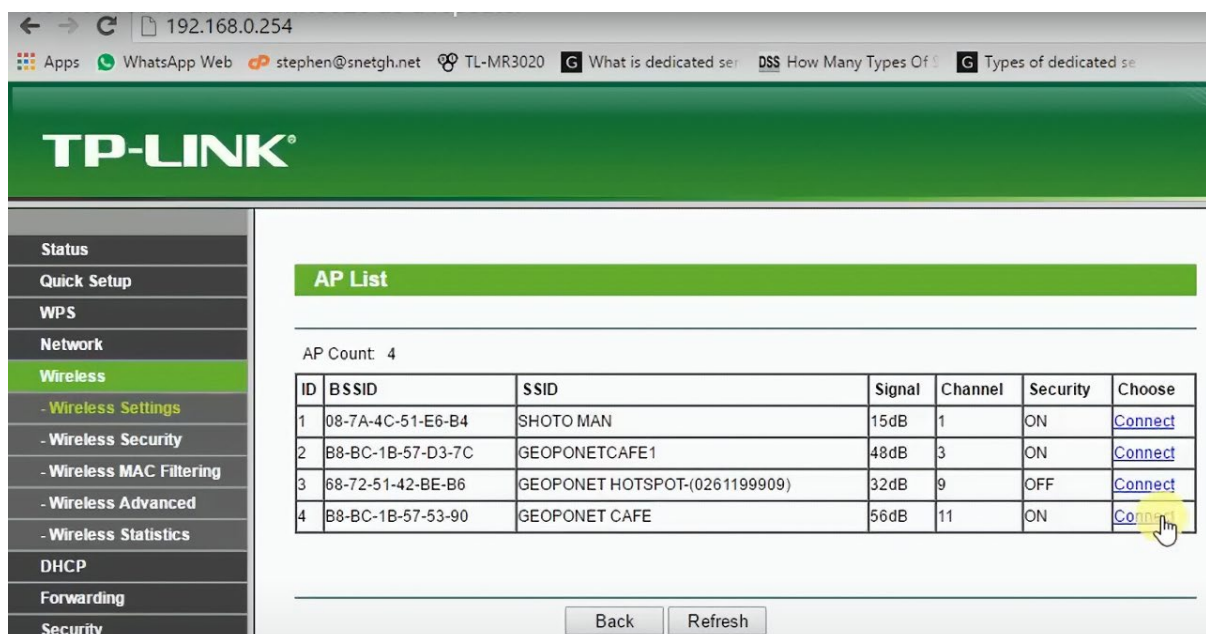


Figure 5.9.6 Selecting wireless network

Client Setting

SSID:

BSSID: Example:00-1D-0F-11-22-33

Key type: ▼

WEP Index: ▼

Auth type: ▼

Password: Enter the password and click save

Figure 5.9.7 Setting up password

After selecting the wireless network, we input the password and our VFv2 will be connected to internet.

Next we will need to update and install the following python library, we enter command into the terminal :

Update command : *sudo apt update*

Python

The primary programming language for this project.

Controls the logic for motion detection, speed calculation, and display output.

It can Integrates different libraries for real-time processing and manages GPIO interactions for the 7-segment display.

Install python command : *sudo apt install python3 -y*

OpenCV

Handles real-time image processing and object detection. It captures video frames from the camera module and applies background subtraction and motion detection to track vehicles.

It can determine the time taken for a vehicle to pass between two reference points and enhances image processing speed through optimized functions.

Additional command needed for VFv2 to install CanberraGTK module to display the view of camera properly.

Install OpenCV command : *pip3 install opencv-python*

Install OpenCV command for VFv2 : *pip install opencv-python-headless*

Extra command for VFv2 : *sudo apt install -y v4l-utils libv4l-dev libcanberra-gtk3-module*
pip install tf-lite-runtime

You Only Look Once(YOLO) Tiny

In this process, we use Yolo tiny to further enhance the object detection so that it can eliminate false detection, for example a human walk into the detection range, and in the same time a vehicle such as car, bike, bus, or bicycle drive through the detection range, it will detect only the vehicle's speed. This model trains massive amount of vehicle object image so that it can filter out other object like animal and human. This method required to implement Yolo v4 tiny into OpenCV, we choose this version because it is an optimize version and it can run well on limited compute resources system such as SBCs. Also we need a COCO dataset which is use for object detection, segmentation and captioning. File can be downloaded from this link :

<https://github.com/AlexeyAB/darknet/releases>



Figure 5.9.8 Git Hub repository

https://github.com/kiyoshiiriemon/yolov4_darknet/blob/master/cfg/coco.names

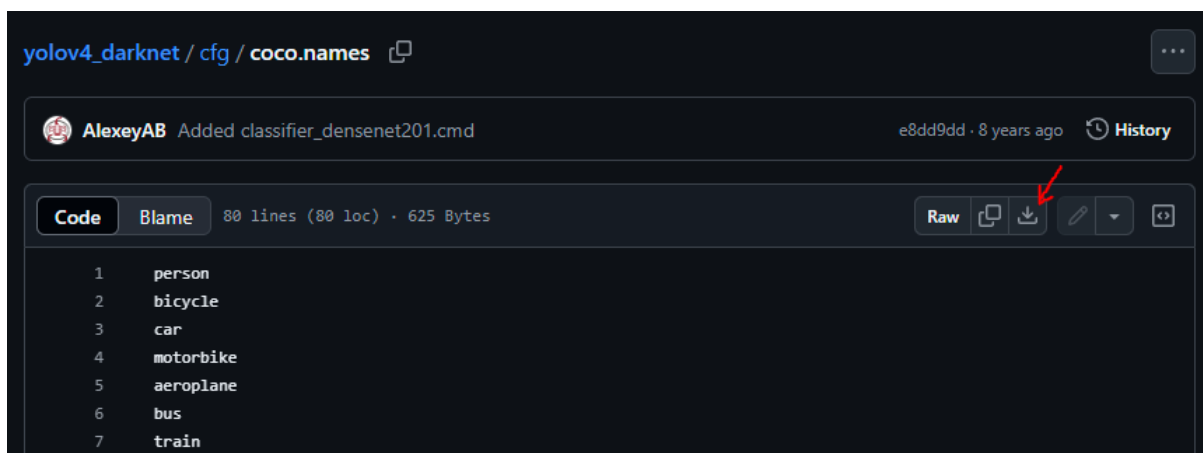


Figure 5.9.9 Download coconames from Git Hub

click download as raw file

Numpy

Use for numerical computation calculations for image processing. It assists in distance and speed calculations by handling numerical data efficiently. It supports OpenCV operations that require fast mathematical computations.

Install numpy command : *pip3 install numpy*

Install numpy command for VFv2 : *pip install numpy\<2*

RPi GPIO & GPIOD

The RPi GPIO is used to configure the GPIO pins on the Raspberry Pi to control the 7-segment display, it works exclusively with Raspberry PI's GPIO only. Pin assignments were carefully documented for accuracy and to avoid conflicts with other components. GPIO18 and GPIO17 were used because it provide Clock Signal required for the TM1637 chipset to work. Next, a suitable display driver is needed in order to control the display to make it function. We found a driver from GitHub, making some modification of the code and it could finally display correctly. The GPIOD function is same as RPi GPIO, it works with Vision Five v2. It also need some modification of the code so that TM1637 could display correctly.

Install RPi GPIO command (raspi only) : *pip3 install RPi.GPIO*

Install GPIOD command (VFv2 only) : *pip3 install gpiod*

Imutils

Simplifies common image processing functions for OpenCV. It resizes and adjusts image frames for optimized processing, and facilitates easier contour detection and motion tracking. Provides convenient functions such as rotating, cropping, and translating images.

Install imutils command : *pip3 install imutils*

Python threading library

The performance result when running Yolo v4 Tiny on a single core makes the detection fps drop massively. By implementing python threading library, it allow the CPU to utilize multi core to process the frame in parallel while detecting object. This is implemented inside the code.

Other Python library

Additional command needed for VFv2 to get it working for our system, due to compatibility issue, multiple python library are needed to be installed :

pip install opencv-python imutils numpy pillow

pip install tf-lite-runtime

pip install psutil

pip install matplotlib

5.3 Setting and Configuration

The hardware involved in this project is SBCs and webcam. The SBCs will detect and capture an object from the camera, it will recognize it as a vehicle, then it will calculate the speed of the vehicle. A toy car is used for testing and deploying this speed detection system in learning computer vision.

The following are the steps to set up motion detection :

- i. Calibrate the distance between the camera and the road
- ii. Activate motion detection algorithm using OpenCV to track vehicles moving through the camera's field of view.
- iii. Contour detection is used to identify moving objects.
- iv. OpenCV Tracking the displacement of the vehicle across frames.

5.4 System Operation for Raspberry PI (with screenshot)

- User need to launch the program first by entering in terminal:
source myprojectenv/bin/activate
python3 SpeedDetection3.py

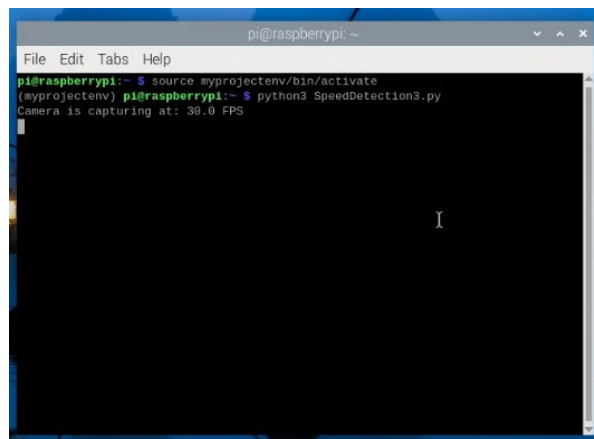


Figure 5.9.9.1 Terminal of Debian OS

- the OpenCV will show the vision of the detection range capture by camera.

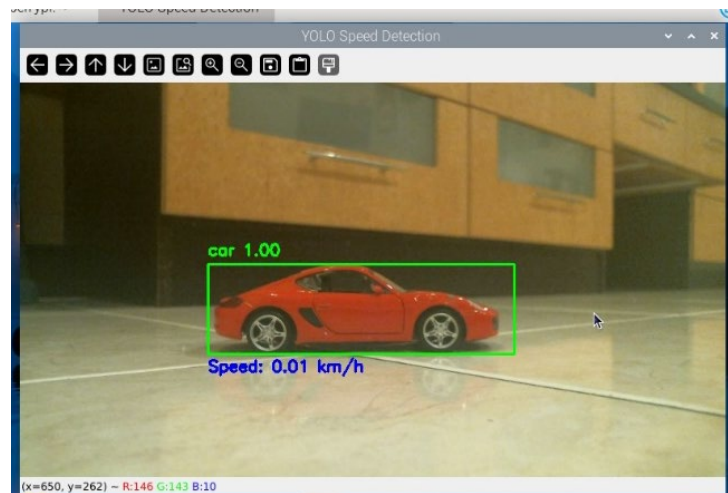


Figure 5.9.9.2 The system is detecting speed of a toy car

- User need to mark down the detection range



Figure 5.9.9.3 The marking on the table

- User need to measure the distance of the detection range

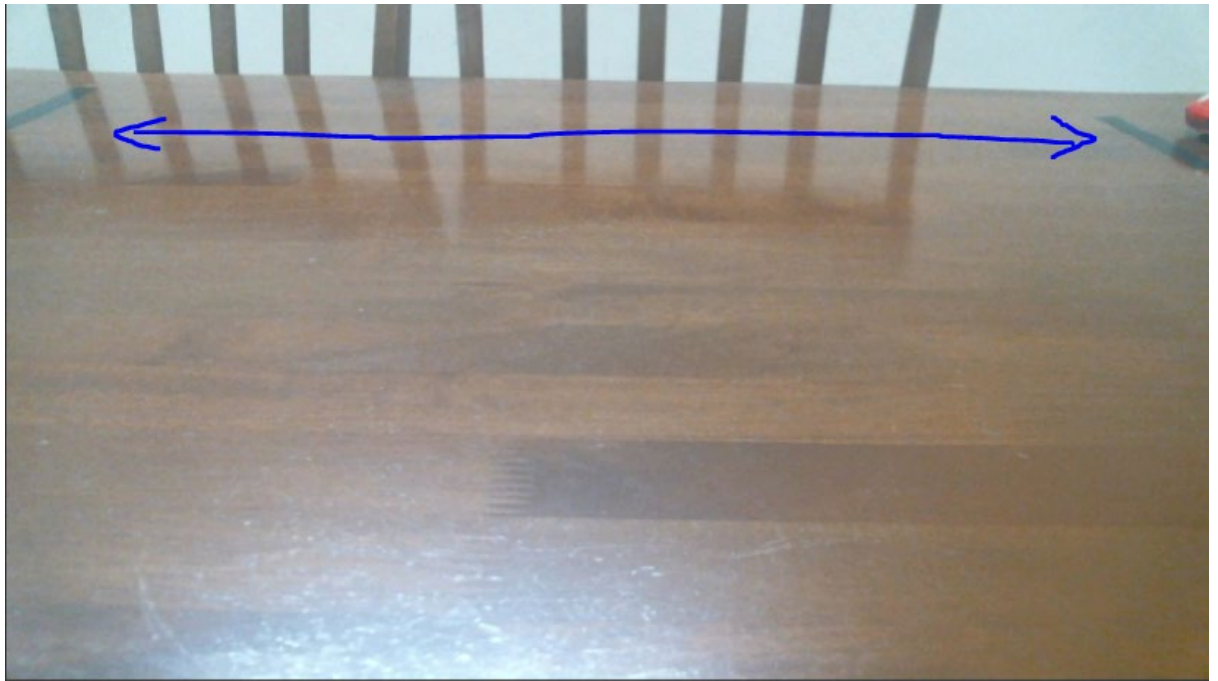


Figure 5.9.9.4 Measure the marking on the table

- User shall input the distance per pixel (m) into the code
- The default value of the webcam for horizontal is 800, so meter divide by 800s pixel

```
def estimate_speed(time_diff, pixel_movement):
    pixel_to_meter = 0.015 # Calibrated based on your setup
    #calibrated based on the actual scene to convert pixel move
    speed = (pixel_movement * pixel_to_meter) / time_diff # m/s
    speed_kph = speed * 3.6 # convert to km/h
    return speed_kph
```

Figure 5.9.9.5 Shows the coding to edit meter per pixel

- User shall input the detection pixel (width and height resolution) of the camera
- The default resolution is 800x240, this resolution is chosen because it captures the image in 16:9 aspect ratio, which is wider compared to 320x320 which is a 4:4 aspect ratio. Not only that, it provides more pixels for the width in order to perform speed calculations.

```
Display.Clear()
Display.SetBrightness(1.0) # Set brightness to maximum

# Define a class for threaded video capture
class VideoCaptureThreading:
    def __init__(self, src=0):
        self.capture = cv2.VideoCapture(src) #input source camera
        self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, 800) #default 800
        self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 240) #default 240
        self.capture.set(cv2.CAP_PROP_FPS, 15) #default 30fps, change to 15f
        self.grabbed, self.frame = self.capture.read()
        self.started = False
        self.read_lock = threading.Lock()
```

Figure 5.9.9.6 Shows the coding to edit width and height

CHAPTER 5

- User shall input the frame rate of the camera
 - The default frame rate is 30fps which is recommended for daytime.
 - Frame rate should set to 15fps during low light environments for accurate speed predictions.

```
Display.Clear()
Display.SetBrightness(1.0) # Set brightness to maximum

# Define a class for threaded video capture
class VideoCaptureThreading:
    def __init__(self, src=0):
        self.capture = cv2.VideoCapture(src) #input source camera
        self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, 800) #default 800
        self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 240) #default 240
        self.capture.set(cv2.CAP_PROP_FPS, 30) #default 30fps, change to 15fps on low light
        self.grabbed, self.frame = self.capture.read()
        self.started = False
        self.read_lock = threading.Lock()
```

Figure 5.9.9.7 Shows the coding to edit FPS

- User shall input the source of the camera to capture video data
 - This needs to be adjusted when the user has two or more cameras connected to the SBC

```
Display.Clear()
Display.SetBrightness(1.0) # Set brightness to maximum

# Define a class for threaded video capture
class VideoCaptureThreading:
    def __init__(self, src=0):
        self.capture = cv2.VideoCapture(src) #input source camera
        self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, 800) #default 800
        self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 240) #default 240
        self.capture.set(cv2.CAP_PROP_FPS, 30) #default 30fps, change to 15fps on low light
        self.grabbed, self.frame = self.capture.read()
        self.started = False
        self.read_lock = threading.Lock()
```

Figure 5.9.9.8 Shows the coding to edit source of camera

- User shall input the smoothing factor by tuning it slightly to match lighting condition of the environment [user can adjust between 0 to 1], this is important to prevent noise from affecting the speed prediction result.

```

# Load class labels
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]

# Start threaded video capture
cap = VideoCaptureThreading().start()

# Initializations
alpha = 0.2 # Adjust to find the right balance for your setup
## Smoothing factor between 0 (no smoothing) and 1 (full smooth:
pixel_movement = 0 #need to adjust and tuning, Initial movement
#calibrated based on the actual scene to convert pixel movement
previous_x = 0 # Initialize previous x-coordinate

```

Figure 5.9.9.9 Shows the coding to edit smoothing factor

- User could adjust the vehicle counting timer to prevent duplicate counts of vehicle

```

# Vehicle Counting Logic
if vehicle_detected:
    if time.time() - previous_detection_time > 4: # Avoid duplicate counts
        vehicle_count += 1
        previous_detection_time = time.time()
    print(f"Total Vehicles Counted: {vehicle_count}")

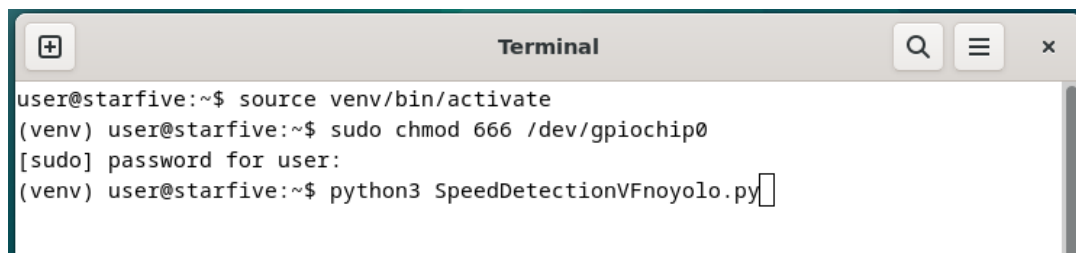
```

Figure 5.9.9.9.1 Shows the coding to edit vehicle counting timer

- User could exit the program by pressing 'q'

5.4 System Operation for Vision Five v2(with screenshot)

- User need to first enter the password to operate the OS, the password is starfive
- User need to launch the program first by entering in terminal:
 source venv/bin/activate (this is to activate the virtual python environment)
 sudo chmod 666 /dev/gpiochip0 (this is to allow the virtual environment to have access
 to GPIO chip to control the TM1637 display
 (it will request user to input password again, password is starfive)
 python3 SpeedDetectionVFnoyolo.py (launch the program)



```

user@starfive:~$ source venv/bin/activate
(venv) user@starfive:~$ sudo chmod 666 /dev/gpiochip0
[sudo] password for user:
(venv) user@starfive:~$ python3 SpeedDetectionVFnoyolo.py
    
```

Figure 5.9.9.2 Shows the terminal of VFv2

- the OpenCV will show the vision of the detection range.

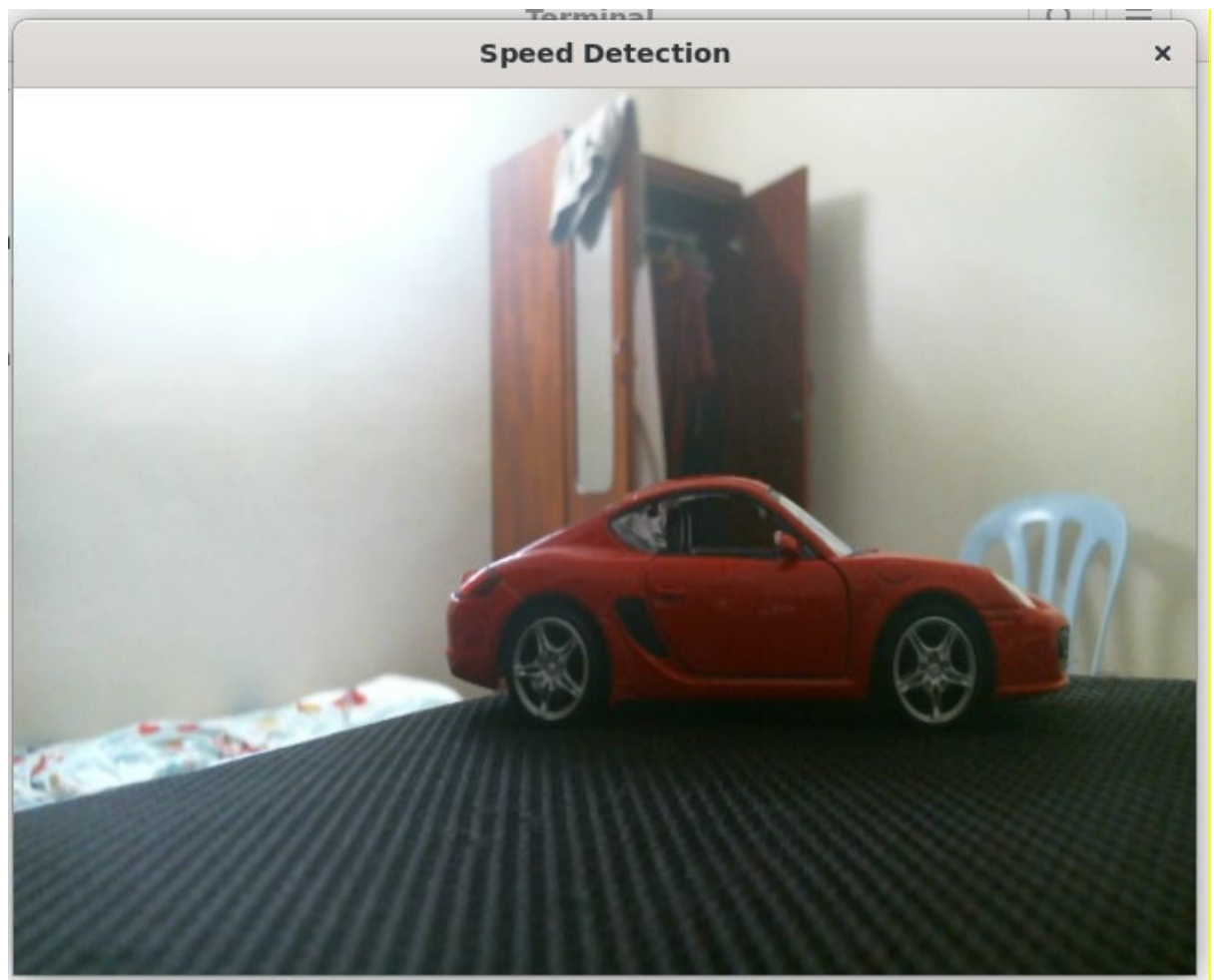


Figure 5.9.9.3 Shows the OpenCV windows in VFv2

- User need to mark down the detection range



Figure 5.9.9.4 Shows the table with marking

- User need to measure the distance of the detection range

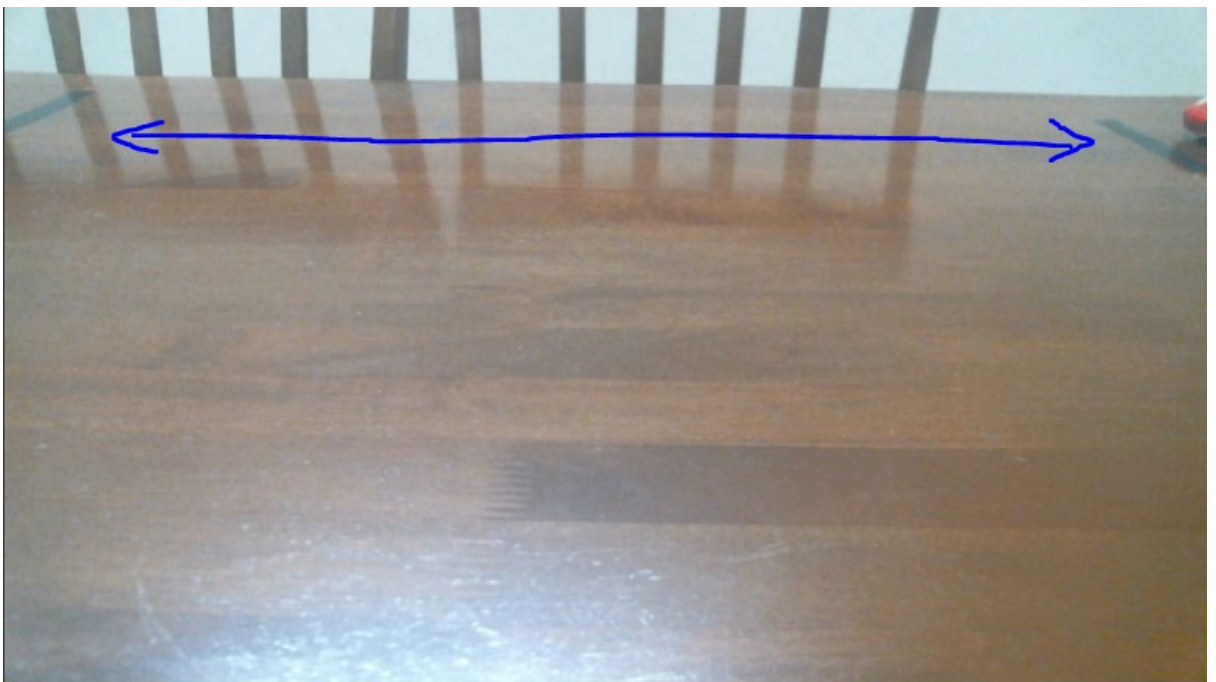


Figure 5.9.9.5 Measure the marking distance

- User shall input the distance per pixel (m) into the code
 - The default value of the webcam for horizontal is 800, so meter divide by 800 pixel


```

#initialize background subtractor
bg_subtractor = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=50)
# Initializations
previous_speed = 0.0 #added for VF2
alpha = 0.2 # Adjust to find the right balance for your setup
## Smoothing factor between 0 (no smoothing) and 1 (full smoothing)
pixel_movement = 0 #need to adjust and tuning, Initial movement set to zero, for first-calculation
#calibrated based on the actual scene to convert pixel movement into real-world speed accurately
previous_x = 0 # Initialize previous x-coordinate

# Speed estimation variables
prev_time = time.time()

def estimate_speed(time_diff, pixel_movement):
    # Convert pixel movement to real-world speed (adjust scaling factor based on testing)
    pixel_to_meter = 0.015 # Example scaling factor (adjust based on your setup)
    speed_mps = (pixel_movement * pixel_to_meter) / time_diff # Speed in meters per second
    speed_kph = speed_mps * 3.6 # Convert to km/h
    return speed_kph

```

Figure 5.9.9.6 Show code to modify meter per pixel

- User shall input the detection pixel (width and height resolution) of the camera
-The default resolution is 800x480

```

# Define a class for threaded video capture
class VideoCaptureThreading:
    def __init__(self, src=4):
        self.capture = cv2.VideoCapture(src) #input source camera #default src
        self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, 800) #default 640
        self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480) #default 480
        self.capture.set(cv2.CAP_PROP_FPS, 30) #default 30fps
        self.grabbed, self.frame = self.capture.read()
        self.started = False
        self.read_lock = threading.Lock()

        # Verify the FPS settings

        actual_fps = self.capture.get(cv2.CAP_PROP_FPS)
        print(f"Camera is capturing at: {actual_fps} FPS")

```

Figure 5.9.9.7 Show code to modify resolution of the camera

- User shall input the frame rate of the camera
-set the frame rate to 30fps during great light condition, set to 15fps on low light condition

```

# Define a class for threaded video capture
class VideoCaptureThreading:
    def __init__(self, src=4):
        self.capture = cv2.VideoCapture(src) #input source camera #default src
        self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, 800) #default 640
        self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480) #default 480
        self.capture.set(cv2.CAP_PROP_FPS, 30) #default 30fps ←
        self.grabbed, self.frame = self.capture.read()
        self.started = False
        self.read_lock = threading.Lock()

# Verify the FPS settings

actual_fps = self.capture.get(cv2.CAP_PROP_FPS)
print(f"Camera is capturing at: {actual_fps} FPS")

```

Figure 5.9.9.8 Show code to modify FPS of the camera

- User shall input the source of the camera to capture video data

```

# Define a class for threaded video capture
class VideoCaptureThreading:
    def __init__(self, src=4):
        self.capture = cv2.VideoCapture(src) #input source camera #default src ←
        self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, 800) #default 640
        self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480) #default 480
        self.capture.set(cv2.CAP_PROP_FPS, 30) #default 30fps
        self.grabbed, self.frame = self.capture.read()
        self.started = False
        self.read_lock = threading.Lock()

# Verify the FPS settings

actual_fps = self.capture.get(cv2.CAP_PROP_FPS)
print(f"Camera is capturing at: {actual_fps} FPS")

```

Figure 5.9.9.9 Show code to modify resolution of the camera

- User shall input the smoothing factor by tuning it slightly to match lighting condition of the environment [between 0 to 1]

```

#initialize background subtractor
bg_subtractor = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=50)
# Initializations
previous_speed = 0.0 #added for VF2
alpha = 0.2 # Adjust to find the right balance for your setup ←
## Smoothing factor between 0 (no smoothing) and 1 (full smoothing)
pixel_movement = 0 #need to adjust and tuning, Initial movement set to zero, for first-calculation
#calibrated based on the actual scene to convert pixel movement into real-world speed accurately
previous_x = 0 # Initialize previous x-coordinate

# Speed estimation variables
prev_time = time.time()

def estimate_speed(time_diff, pixel_movement):
    # Convert pixel movement to real-world speed (adjust scaling factor based on testing)
    pixel_to_meter = 0.015 # Example scaling factor (adjust based on your setup)
    speed_mps = (pixel_movement * pixel_to_meter) / time_diff # Speed in meters per second
    speed_kph = speed_mps * 3.6 # Convert to km/h
    return speed_kph

```

Figure 5.9.9.9.1 Show code to modify smoothing factor

- User could exit the program by pressing 'q' in the program

5.5 Implementation issues and challenges for Raspberry PI 4B

Library Compatibility issue :

Some Python libraries require specific versions to function correctly. Compatibility issues were resolved by installing alternative versions using pip command.

Camera Angle Configuration :

We need to adjust the camera angle with good lighting conditions. Low light reduces the accuracy of the detection. The adjustments of camera angle placement and calibration take longer time and need to be precisely calculated.

Noises causes incorrect result :

Initial tests showed incorrect speed results. It display speed when the toy car is not moving, this is because of the noise pixel which is causing false detection. Implementing Smoothing Pixel Movement with Alpha Blending which helps to reduce noise in the speed estimation calculation when the car is stationary or moving slowly. It helps smooth out small inconsistencies in detected movements by averaging movements over iterations.

Long Range vehicle undetectable :

This happens when the distance of the toy car and the camera is approximately 85cm, it become unable to identify and detect the vehicle and speed. This is due to the limitation of compute resources, we need to limit the resolution of the raspberry Pi 4B in order to reach 30fps for more accurate speed detection results.

Higher resolution reduce detection rate :

When running max supported resolution of 800x488 as input, due to a heavier processing, the system missed a few detection due to increased latency. This can be fixed by reducing resolution which solves the issue, but this introduces another issue such as blurry image, thus reducing the detection range.

Issue face when implementing TM1637 display :

This module is mainly designed to work with Raspberry Pi Pico and Arduino UNO, which uses micro python as the driver. Although the pin is compatible with Raspberry Pi, there are not many resources available online, and most of the information is already outdated. I would need to modify the driver code with some help of the forum and tune the code in order to be able to run without issue.

5.5 Implementation issues and challenges for Vision Five v2

Low Performance Compared to ARM SBCs :

The RISC-V architecture is still in early development, leading to poor software optimization and lower performance than Raspberry Pi 4B. Slower CPU and GPU result in very low FPS (around 6 frame per minute) when running object detection models like YOLO even with very low resolution like 320x240. The program also very unstable and frequently freeze and not responding, when running with Yolo, the CPU usage reach 373% which being monitored using top command. After removing YOLO, the system runs in acceptable framerate on 640x480 resolution.

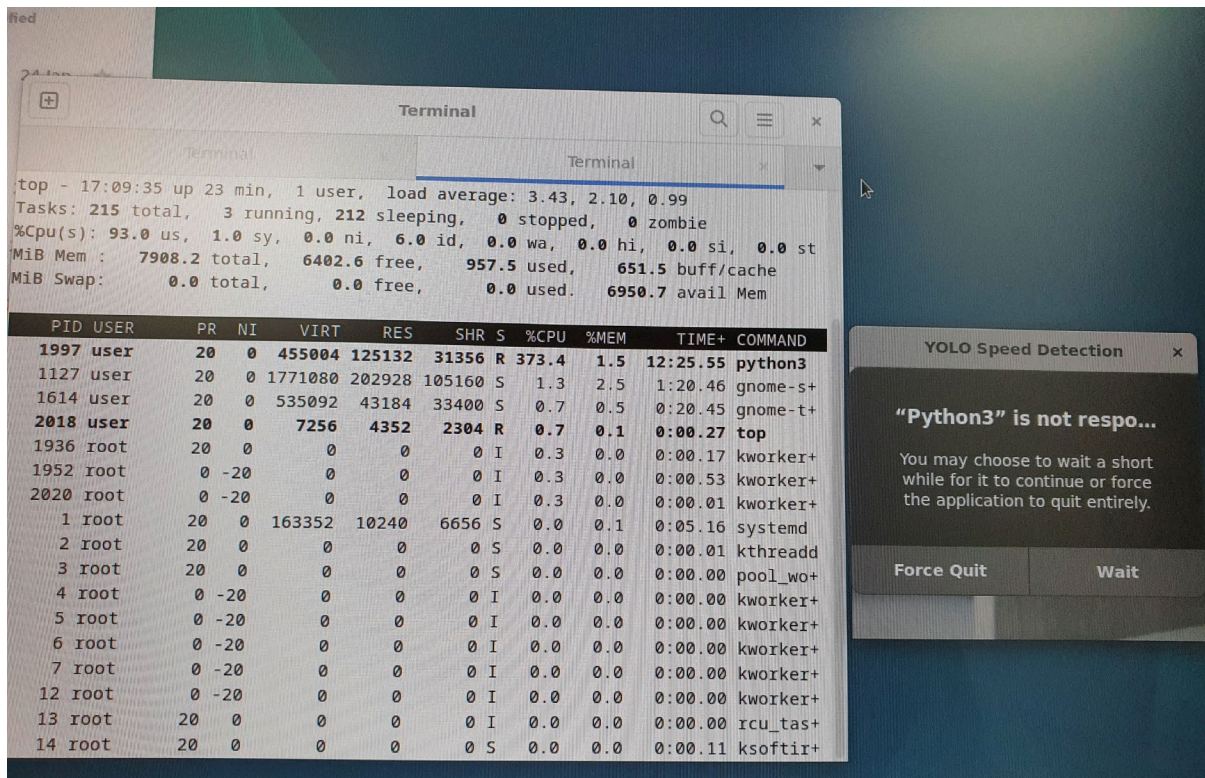


Figure 5.9.9.9.2 Show high cpu usage and program not responding

After removing YOLO, the system runs in acceptable framerate on 640x480 resolution.

Limited Software Support for RISC-V architecture :

Many precompiled Python libraries such as NumPy, OpenCV are not officially available for RISC-V. During installation, installing dependencies takes longer and often fails due to missing packages. The solution is to use precompiled packages when possible instead of building from source.

OpenCV, Numpy and imutils installation issue :

Directly install numpy and OpenCV fails due to missing dependencies on RISC-V. OpenCV crashes due to missing GUI dependencies. Both issue never happens on RaspberryPi platform. OpenCV headless version and Libcannberra GTK module is needed in order to display the camera output and GUI correctly. Numpy and imutils takes very long time to install, this is because RISC-V architecture lacks readily available precompiled python wheels on the PyPI repository, the system would forced to compile heavy library from source code locally which leads to long installation time. This issue is overcome by letting the system install the software overnight.

Driver support from GPU vendor :

GPU vendor such as NVIDIA only provide CUDA and TensorRT support for x86 and ARM platform. No CUDA support exists for RISC-V, which means there is no way to accelerate YOLO using external GPU.

Camera Index Change

There was a random issue where the camera index would change unexpectedly after rebooting or reconnecting devices. This made the system unable to consistently access the intended camera, causing initialization failures. The solution involved verifying camera device nodes using v4l2-ctl commands and dynamically selecting the correct video index at runtime.

Unable to access GPIO chip in Virtual Environment

When operating inside a Python virtual environment, the program initially could not access the GPIO chip due to restricted permissions. This issue was resolved by adding the user to the gpio group and running the environment with appropriate system permissions.

5.6 Concluding Remark (Raspi 4B)

The proposed method for traffic speed detection using a Raspberry Pi 4B, a consumer level webcam, and OpenCV demonstrates a promising foundation for low-cost and compact traffic monitoring systems. The use of low-cost hardware (Raspberry Pi and a webcam) makes the system economically viable and accessible for smaller-scale applications. The integration of OpenCV for real-time image processing aligns with the system's goal of lightweight and efficient computation. Real-time detection at a reduced resolution 800x488 ensures stable frame rates for basic operation.

Although the system struggles significantly in low-light conditions, which limits its usability at night or in poorly lit areas. It is suitable to be implemented in school zone where less vehicle will be travel in school area during nighttime. The inability to accurately measure high speeds restricts its application in realistic traffic scenarios such as highways where vehicles typically travel much faster. This however can be overcome if the school zones have multiple road bumps that limit vehicle driving speed.

The computational limitation, the Raspberry Pi's limited resources pose a bottleneck, especially when advanced models like YOLO are used in the system. The system could only use Yolo v4 tiny which is a lightweight version of YOLO tuned for resources limited system to run smoothly. By adding preprocessing techniques such as background subtraction or adaptive thresholding can be implemented within OpenCV, which require minimal additional computational power. These methods can enhance detection robustness without significant hardware changes. It could improve low-light detection and assist in maintaining object recognition consistency. In Long-Term Viability, the proposed method is feasible with incremental upgrades and adjustments, especially in environments with medium to low vehicle speed and basic lighting conditions. For broader adoption in real-world traffic monitoring, additional optimization and hardware scalability will be necessary to meet higher accuracy and performance demands. If this system needs to detect high speed vehicle on highway, doppler sensors will be needed for better accuracy.

5.6 Concluding Remarks (Vision Five v2)

Implementing a vehicle speed detection system on the VisionFive V2 presents multiple challenges due to limited software support, lower processing power, and hardware compatibility issues compared to more established ARM-based SBCs like the Raspberry Pi. Due to weaker computation power, it gives unacceptable results when running the program with YOLO. After applying software optimization such as removing YOLO from the code, and by using background subtraction, multithreading, the system could runs at 800x480 with 30FPS, thus making real-time speed estimation feasible.

VisionFive V2 is still in its early stages of software maturity, careful optimizations and workarounds make it possible to develop practical applications. Future improvements in RISC-V software support and hardware acceleration will further enhance its viability for computer vision applications. For now, this project successfully demonstrates that a real-time speed detection system can be adapted and optimized for the VisionFive V2, despite its hardware limitations.

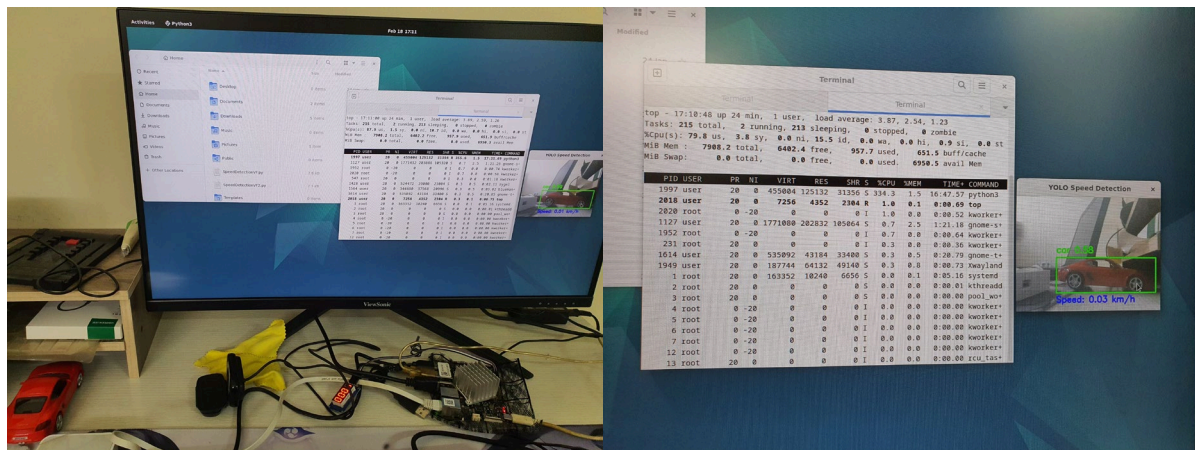


Figure 5.9.9.9.3 Vision Five v2 running YoLo with low fps

After careful troubleshooting, strategic use of system commands, and adjustments to the development workflow, these challenges were successfully overcome on this platform, allowing the project to progress smoothly despite the limitations of the platform.

CHAPTER 6

6.1 System Testing and Performance Metrics

System testing and performance metrics in this project are defined by the system's ability to accurately, efficiently, and reliably detect, track, and display vehicle speeds in real-time within school zones. Detection Accuracy, which is the ability of the system to accurately identify and track moving vehicles within the camera's field of view. This includes the precision of object detection algorithms in correctly identifying vehicles and the accuracy of speed calculations based on real-time image processing.

Next, the environmental robustness, where we test the system's capability in maintaining accuracy and functionality across various environmental conditions, including changes in lighting (day/night), weather (rain), and occasional blocking object.

6.1.1 Accuracy and functionality

There are a few checks needed to be carried out because there are several factors that can impact the system's accuracy :

Situation 1

Test the accuracy of the system, when other object appears in the detection range at the same time.

Procedure Number	P1
Method	Testing
Applicable Requirements	The system needs to correctly identify the vehicle speed without being affected by other moving object such as leaf and human movement
Purpose/Scope	To test whether it will give false reading when vehicle object pass by the detection range
Items Under Test	Vehicle passing through the detection range
Precautions	The vehicle object needs to be moving
Special Conditions/Limitations	The system should filter out other object movement except vehicle like cars
Equipment/Facilities	Raspberry PI/Vision Five v2
Data Recording	Yes

Table 6.1 Verification for Procedure 1

Situation 2

The system should be able to identify vehicle during low light condition.

Procedure Number	P2
Method	Testing
Applicable Requirements	The system needs to correctly identify the vehicle passing through the detection range during low light condition (night time).
Purpose/Scope	To test if the system would be able to work at night
Items Under Test	Car passing through the detection range during night
Precautions	The test needs to be conduct during nighttime
Special Conditions/Limitations	The system should be able to detect the vehicle and give correct speed reading
Equipment/Facilities	Raspberry PI/ Vision Five v2
Data Recording	Yes
Acceptance Criteria	The system able to detect vehicle and give result
Procedures	1. Adjust the smoothing factor, then wait for vehicle to pass through the detection range 2. Determine the result given
Troubleshooting	Refine the parameter tuning

Table 6.2 Verification for Procedure 2

Situation 3

The system should be able to detect all the vehicle that pass through the detection range

Procedure Number	P3
Method	Testing
Applicable Requirements	The system needs to detect all the vehicle that pass through the detection range
Purpose/Scope	To test the rate of detection of the system
Items Under Test	Any Vehicle
Precautions	The vehicle needs to be moving with various speeds
Special Conditions/Limitations	There are different types of vehicles on the market, such as motorbike, bicycle, lorry, van and pickup trucks
Equipment/Facilities	Raspberry PI/ Vision Five v2
Data Recording	Yes
Acceptance Criteria	The system able to recognize it as a car and give speed readings.
Procedures	1. Adjusting the position of the camera, then let multiple vehicles pass through the detection range 2. Determine the result given
Troubleshooting	None

Table 6.3 Verification for Procedure 3

6.2 Testing Setup and Result for both system :

Setting for the testing setup :

Pixel per meter = 0.015

Resolution = 800 x 480

FPS = 30

Smoothing Factor/Alpha = 0.2

Condition : day, noon, and night



Figure 6.1 Testing location is at the housing area in front of my house.

The camera is 90degree angle towards the incoming vehicle

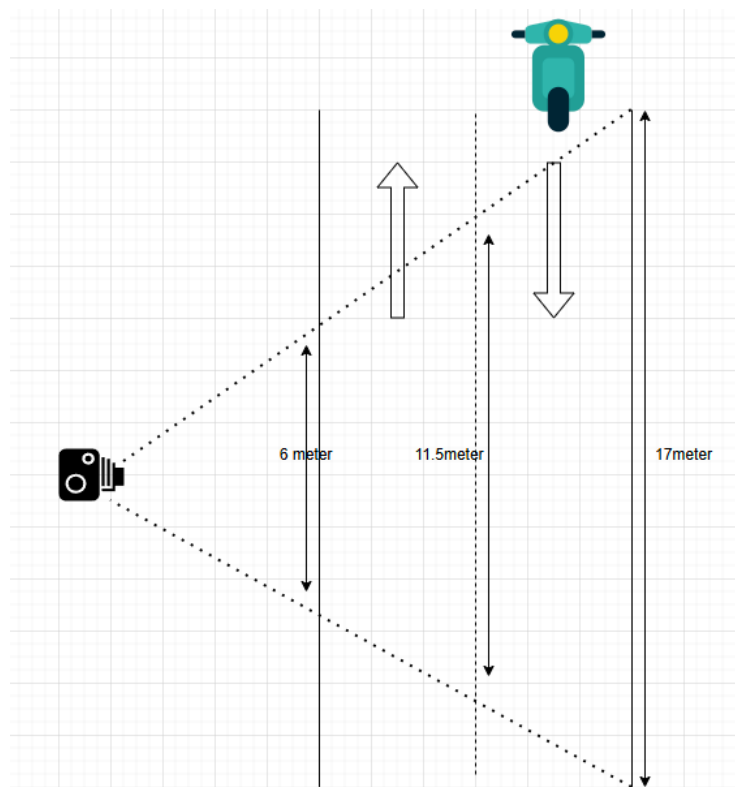


Figure 6.2 Example of the testing setup

During testing, I found that most of the users do not follow the traffic rules in residential area. Because vehicle did not travels frequently in residential area, most of them prefer to drive in the middle of the road even though it is a two way lane. This will cause false reading because the calculation is different if they travel at different lanes, which is mainly caused by the angle. The camera mounting method 1 is only suitable for single lane road. However we could use the middle lane, which is 11.5meter to divide the pixel, to get the average reading, however it might not get as accurate as using a proper doppler sensor to get the speed reading.

Also, a vehicle counting function had been implemented for Raspi 4B, this function will calculate how many vehicles pass through the detection range. There are two methods to implement this function, first is when the system detects the vehicle, it will count as 1. The second method is using timer to detect vehicle. Both methods had advantages and disadvantages. For the first method, if there is an obstacle blocking the vehicle for a second, it might count the vehicle twice. The second method, if the vehicle stops at the detection range for an interval eg: 3seconds (can be adjusted in the system), it will be counted twice, and even more. This means if the vehicle stops at the detection range for too long, the system will count multiple times.

The second method was chosen because it can reduce the false count due to obstacles, and it is more suitable to implement in the school area because drivers are not allowed to stop at the middle of the road. But in the testing condition, please note that there are neighbors that park their car in houses which cause vehicle counting to be inaccurate.

This function is not implemented for VFv2 because any moving object will be counted not limited to vehicle, this will cause the counting result to be unmeaningful.

The vehicle counter will show at the top left of the program.

A recording function had been implemented for the VFv2 due to limitation of remote support. The system will perform recording when detecting motion, and stop recording when there are no motion detected. This function fails in the end due to bugs of OpenCV where it could not output files after recording. This function is not implemented on Raspi 4B due to performance limitations and storage limitations. VFv2 will record the footage using the screen recorder build into the OS itself, and Raspi 4B will use Pi Connect and laptop for screen recording. Please note that there's latency penalty when using Pi Connect during recording.

Situation1

The system we implement for Raspi 4B should be able to correctly identify the vehicle speed without being affected by other moving object such as leaf and human movement because of the object detection algorithm by Yolo. However, the VisionFive v2 couldn't deliver a good result when running object detection algorithm, the system can only uses motion detection, which will be greatly affected by any other moving object.

Raspi 4B result

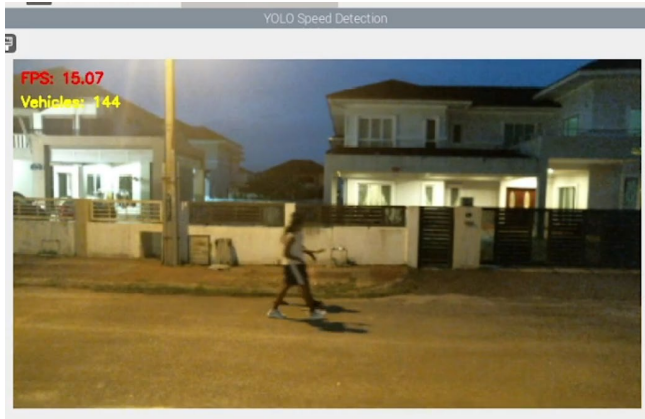


Figure 6.3 shows the capture image of raspi 4B

The Raspi4B system with object detection will not read other object speed other than vehicle.

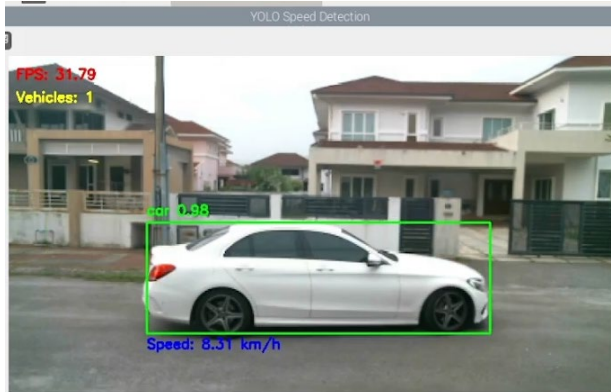


Figure 6.4 shows the capture image of raspi 4B detecting vehicle

The speed detected of the moving vehicle remains accurate even when other objects like auto gate is moving.

VFv2 result :

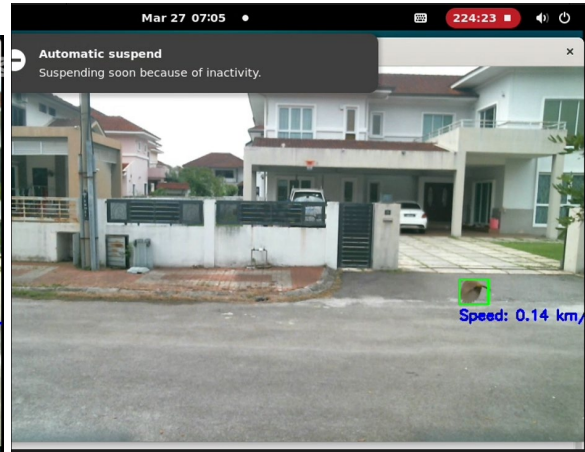
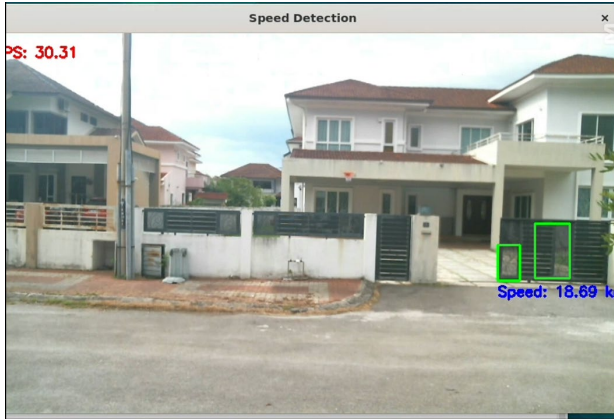


Figure 6.5 & 6.6 show VFv2 detects the Auto gate and dry leaves moving and show speed.

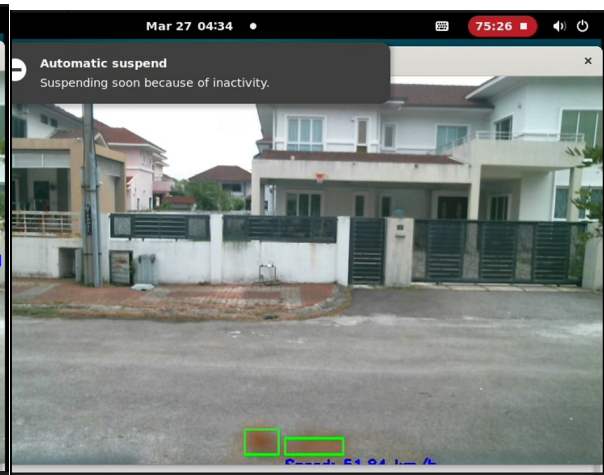
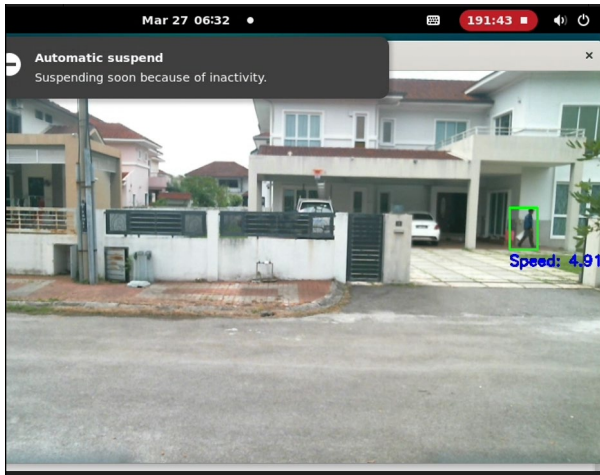


Figure 6.7 & 6.8 show VFv2 detects the human movement and ant moving and show speed. The system detects any movement and calculate speed, like humans and ants. This will greatly affect the accuracy of the system. This is because it detects multiple moving objects in the same frame while only giving one result. The system will calculate the movement between the two objects.

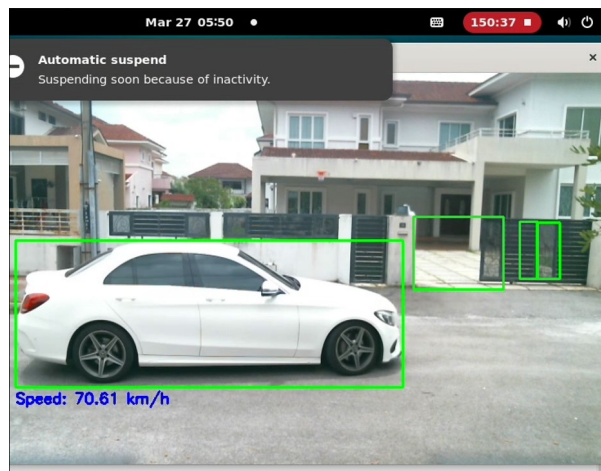


Figure 6.9 False speed detected of the moving vehicle when the auto gate is moving with VFv2

Conclusion for Situation 1 :

This shows that during daytime, Raspi 4B gives a better overall accuracy using object detection compare to VFv2 where the accuracy will get affected when other moving object is around. The moving object can be multiple moving car that exist in the detection range. For school zones, especially on weekdays, there are many cars travelling on the road, which makes the VFv2 unsuitable for this application.

Situation 2

Before performing testing on situation 2, I implemented a real time FPS counter, and I found that the system can capture the frame in 30fps, but in low light conditions, it can only capture frame in 15fps, the performance drop in half. Not only that, the vehicle detection also become weaker, as shown in the picture. In bright environments, it recognizes it 84% likely is a vehicle. But in dark environments, it reduces to 61% likely as a vehicle. This testing is held where the vehicle is in an idle position without moving.

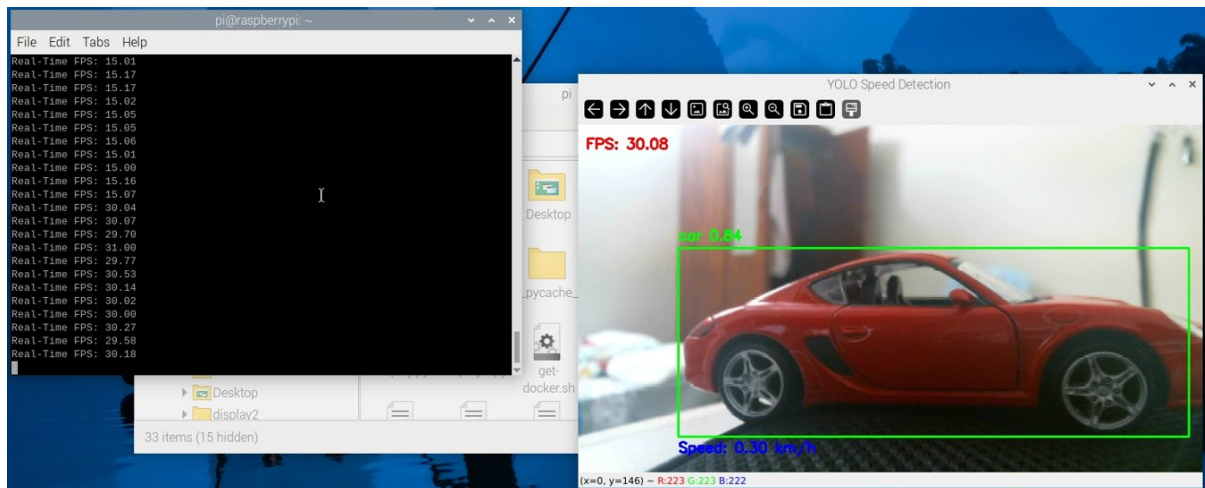


Figure 6.9.1 FPS captured in bright environment on Raspi

CHAPTER 6

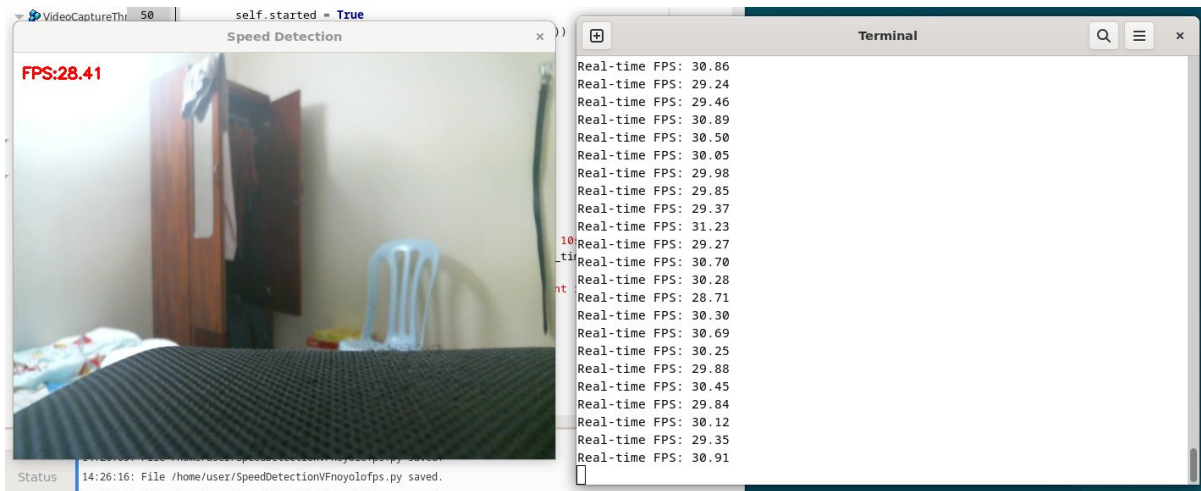


Figure 6.9.2 FPS captured in bright environment on VFv2

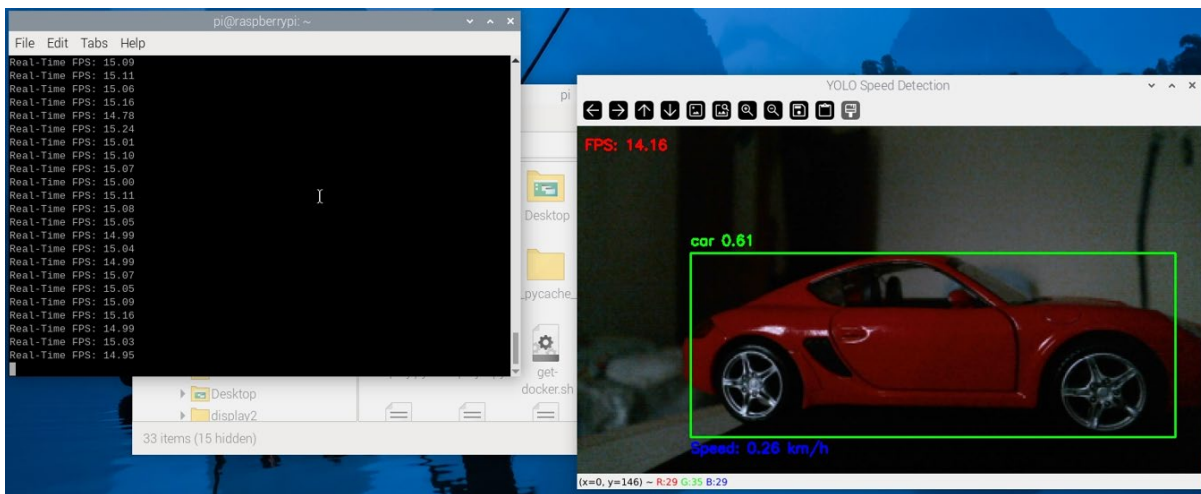


Figure 6.9.3 FPS captured in dark environment on Raspi

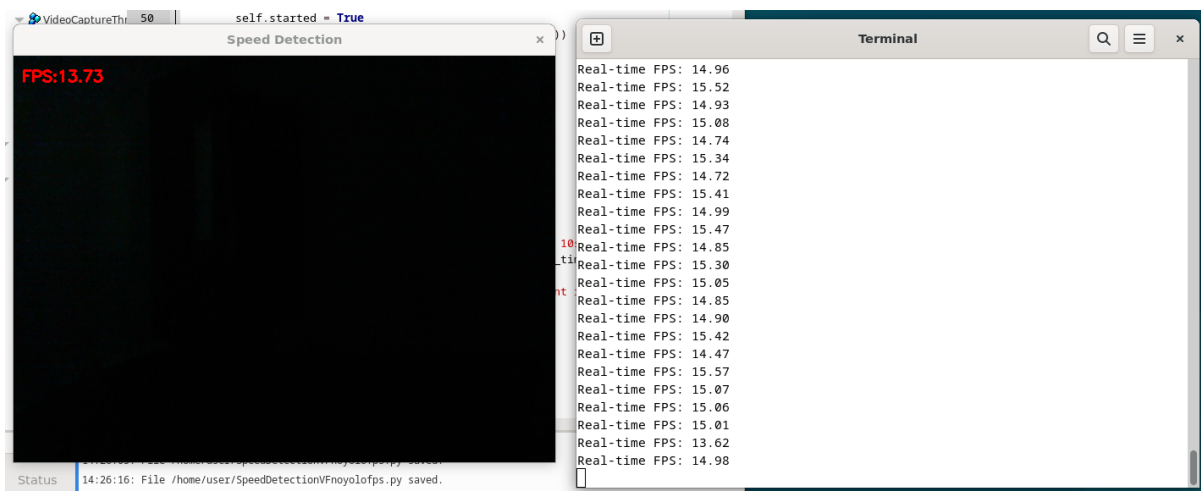


Figure 6.9.4 FPS captured in dark environment on VFv2

The result might appear worse in real world testing if the vehicle is moving. This issue is likely caused by the Auto Exposure adjustment of the webcam, where it increases exposure to brighten the image, and more exposure will cause each frame to take more time, thus reducing the FPS. After disabling the auto exposure, setting a fixed exposure, increasing ISO gain which can brighten the image, and even reducing the resolution, the FPS still drop to half. The same real time FPS test was conducted on VFv2, and the result shows the same. Finally, we can conclude that it is a hardware limitation of the camera module, where this camera prioritizes capturing more light per frame than maintaining FPS. To perform testing on low light environment, the setting had to be changed to 15fps to get a correct speed prediction. During a mild light environment, the camera will be able to capture at 25fps. We can see that the camera will adjust the FPS input based on light, dim and dark environment.

To make this system more superior and automated, a dynamic FPS function is implemented, where it will adjust the FPS dynamically based on the lighting conditions, so that user don't have to adjust the FPS every time to match the environment factor.

Before the tests are conducted, I use my own vehicle to perform multiple testing to ensure the system is accurate. The housing area has a lot of road bumps, the highest speed a vehicle can go is around 30~40km/h. Any speed higher than 60km/h is not possible because there's a lot of road bumps on the road and the distance between each road bumps is close.

Raspi 4B result :

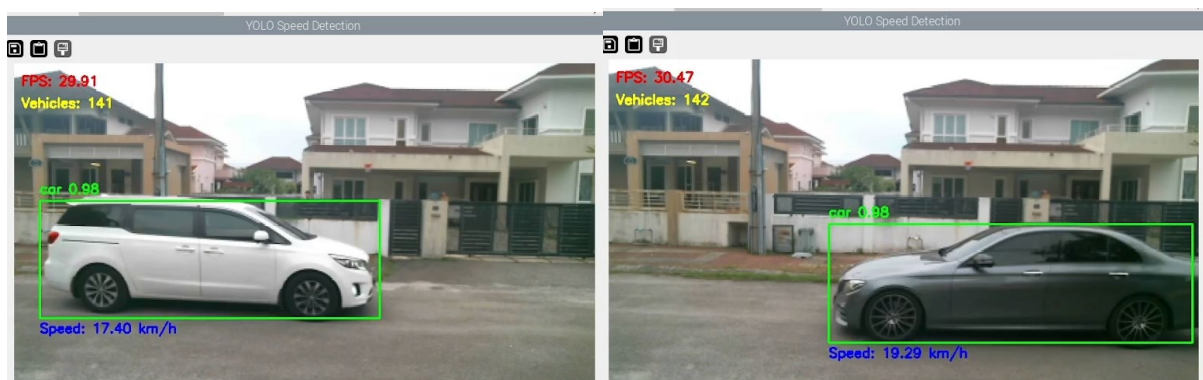




Figure 6.9.5 Shows the system tested in daytime with Raspi 4B
During daytime, the light condition is good, the system able to detect the speed with good accuracy.

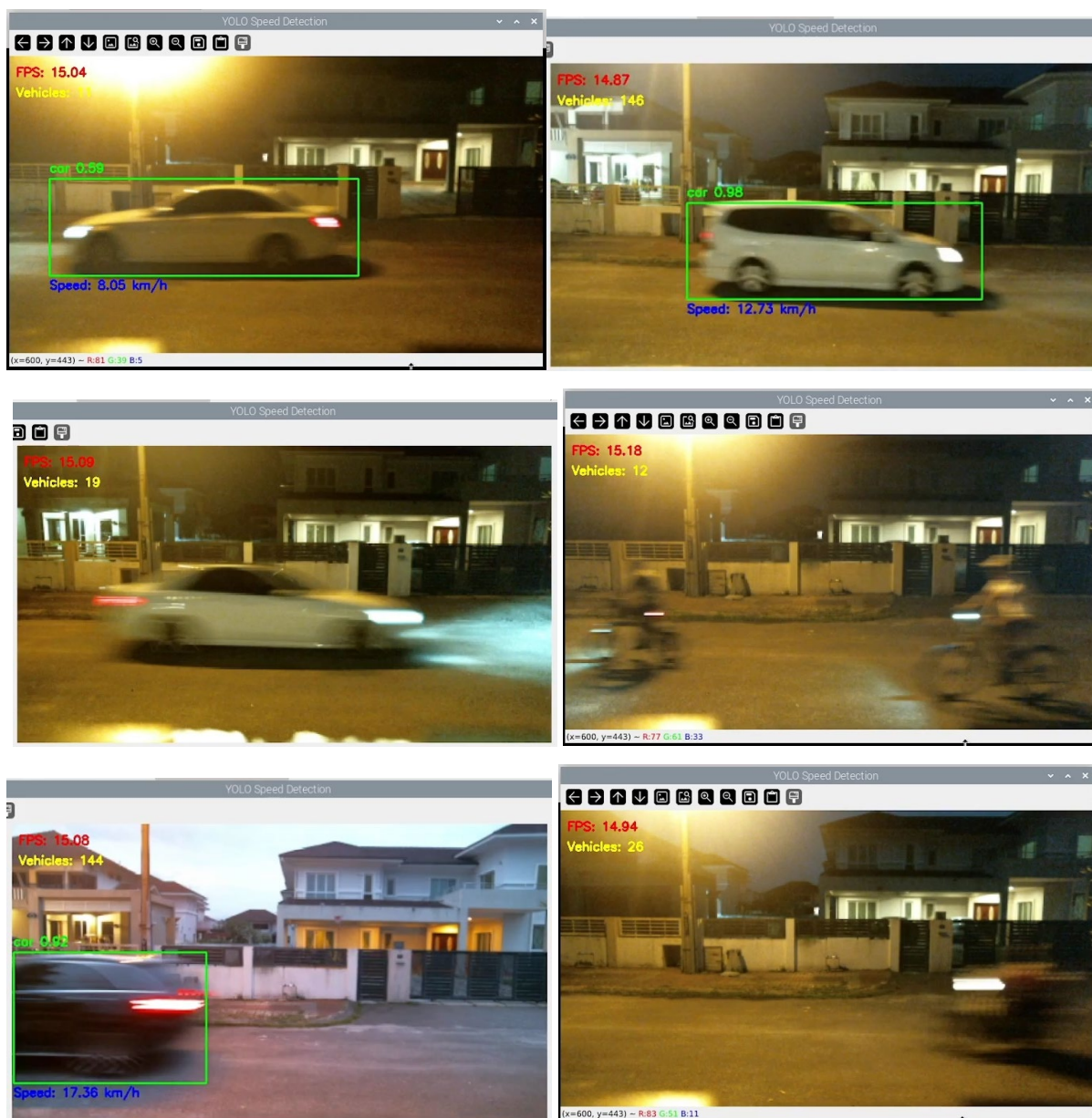


Figure 6.9.6 shows the system tested in during night time with Raspi 4B

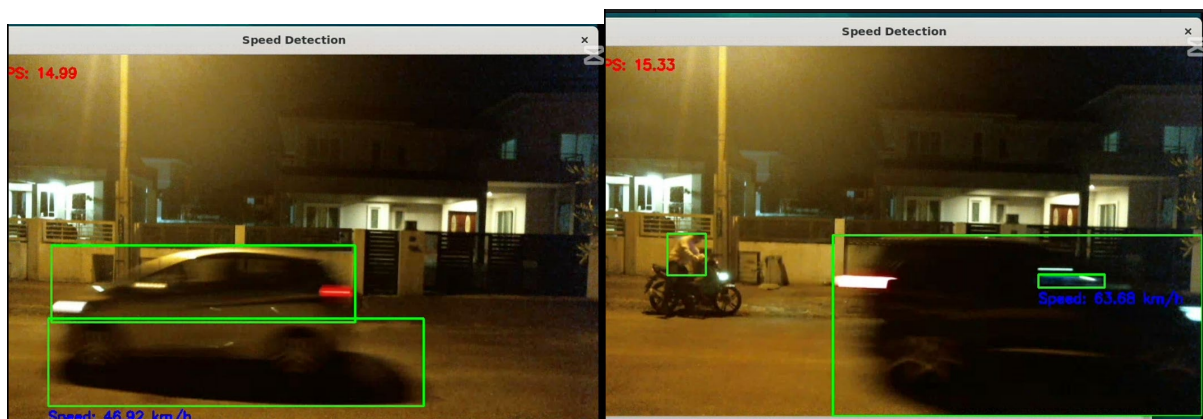
CHAPTER 6

Show the result is not very accurate during low light conditions. If the vehicle travel too fast, the motion blur due to low fps will caused the reading to be inaccurate. Sometimes the system unable to capture vehicle speed at all during low light even though they are not travelling at high speed such as bicycle and motorbike.

VFv2 result :



Figure 6.9.7 shows the system tested in during day time with VFv2
VF v2 detection result during day time. The speed is consider accurate.



But at night, the FPS is lower, it can still get some results, but the detection accuracy drops.

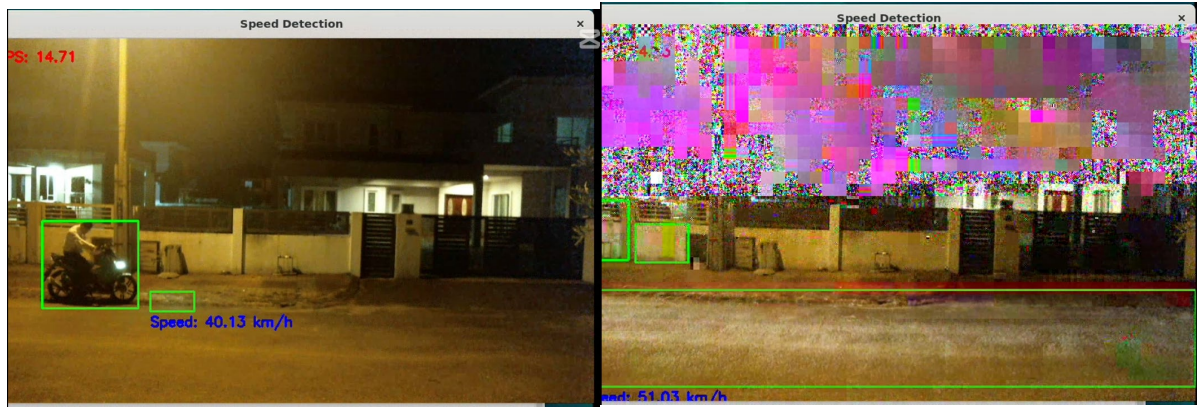


Figure 6.9.8 shows the system tested in during night time with VFv2

The system also detects the moving light spot that comes from the vehicle and calculating speed. This shows that without object detection, the system works poorly.

Conclusion for situation 2

Based on the observation from both of the system, we can see that Raspi 4B and VFv2 perform great during good lighting conditions. But during night, both of the system fail to deliver a good result. Raspi 4B system missed some vehicles that travel fast, and give inaccurate speed due to motion blur. On VFv2 system, we can see that the system can detect speed of the vehicle without issue but it also detects the moving light spot from the vehicle and give false reading. Both system did not perform well at low light condition.

Situation 3

For Situation 3, we test the detection rate for both systems. This is used to test the model and see if all the vehicles can be detected in the range. During this testing, the system is not calibrated perfectly, we only focus on the detection rate for this section.

Raspi 4b Result :



Figure 6.9.9 shows the system unable to detect vehicle if obstacle blocking using Raspi4B

Based on the result, we can see that the model will be unable to recognize it as car when object overlap it, such as human and gate



Figure 6.9.9.1 shows the system able to detect different types of vehicle using Raspi4B

CHAPTER 6

Next, when the bike rider stands beside the motorbike, it still can detect it as motorbike. Other vehicles like bicycle, car and bus can be detect without issue.



Figure 6.9.9.2 shows the system able to detect two vehicle while calculate speed using Raspi4B. Next, this model is able to detect 2 car but only calculate one car speed, which will not mess up the speed reading of the moving vehicle.

However, this model would not work with certain types of vehicle, for example there is a pickup truck which detected as car when it first appear, fail to verify as a vehicle when it fully appear in the detection range, this is cause by the dataset in the Yolo model lack of training this kind of vehicle.



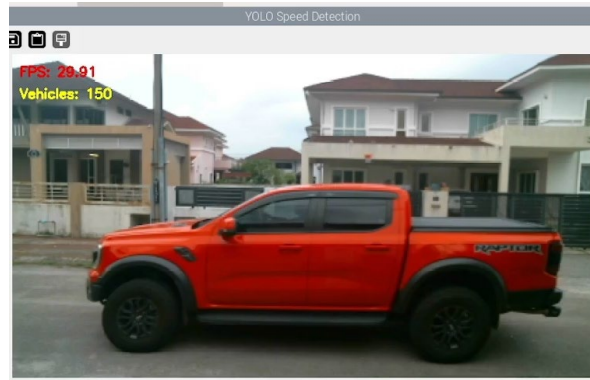


Figure 6.9.9.3 shows the system unable to detect speed of truck and van using Raspi4B

The system is unable to verify it as a vehicle, thus unable to predict the speed. The picture above shows the whole vehicle that entered into the detection range but the system did not recognize the object as vehicle.

We also found that the system facing frame drop issue due to overheating causes the system to miss a vehicle. We carry out the test from 11am to 4pm, the weather in the afternoon is very hot. The temperature on the day is 32degree Celsius.



Figure 6.9.9.4 shows the system overheating and skip frame on Raspi4B

When detecting the truck, the frame drops and it only able to capture the front of the truck, and the end of the truck, the system couldn't capture the whole body of the truck, and cannot recognize it as a vehicle, thus did not give a speed result.

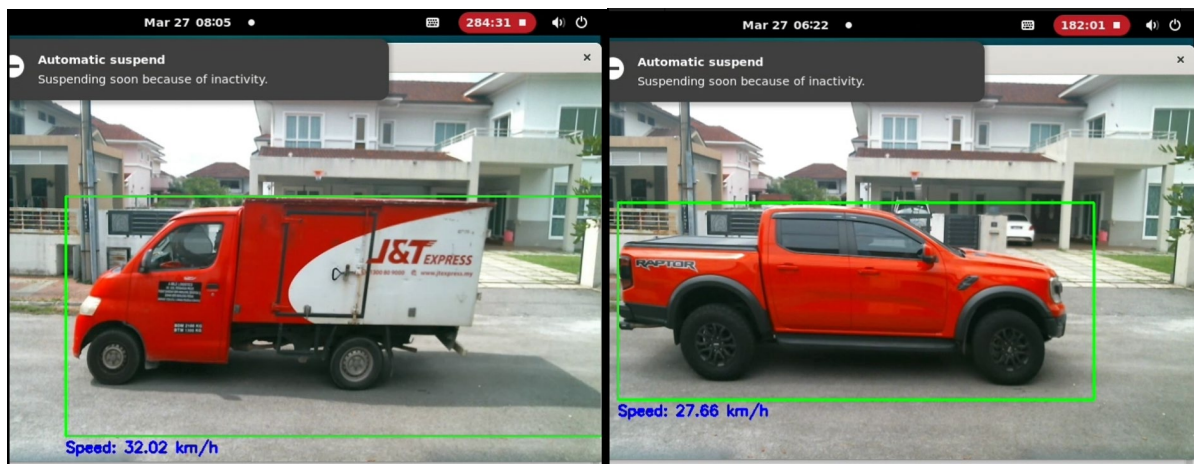
VFv2 result :

Figure 6.9.9.5 shows any vehicle speed using VFv2 without Yolo. The VFv2 could detect small trucks and the pickup truck's speed without issue. The system did not face any frame drop issue even with long period of testing.



Figure 6.9.9.6 shows the system unable to detect speed when 2 vehicle go different direction. When two moving vehicles are inside the detection range with different direction, the VFv2 will not calculate speed.

Conclusion for situation 3

For Raspberry PI 4b, we can conclude that the system will fail to recognize some type of vehicles like pickup trucks. The overheating issue of the system also reduces the detection rate, better airflow is needed for the system. Based on the result above, the model shows very powerful detection for some cars vehicle like Sedan, for example the car that park into neighbor's house, even though only the rear of the car is exposed to the camera it could still detect it as a car. Also, the system holds an advantage where it will only detect one car at a time, so that it will not give false reading when multiple cars is inside the detection range.

For VFv2, the system did not face any overheating. Also, the system is able to detect anything that moves in the detection range. It has a 100% detection rate during daytime with exchange of higher false detection. But when multiple objects are moving in different directions, the system will not be able to give speed readings.

6.3 Project Challenges Faced

Implementing an effective vehicle speed detection system using a single-board computer (SBC) in school zones involves navigating various technical and practical challenges.

Processing Performance of the SBCs

First, Processing Power Limitations. SBCs like Raspberry Pi are cost-effective, price around 75USD but have limited computational power, making real-time video processing for object detection and speed calculation challenging, especially with deep learning models such as YOLO. To accommodate this limitation, the system can utilize lightweight or optimized versions of YOLO such as YOLOv4 tiny. Adjusting frame resolution and capture rates may also help balance processing demands with accuracy. While Vision Five v2 is slightly more expensive than the Raspi 4B which is around 100USD, it performs much worse than the raspberry PI in YOLO and it could only run effectively when YOLO is removed. This RISCv SBCs is not a good choice for this project due to much weaker processing power and a much higher price tag.

Prices		
	Raspberry Pi 4	VisionFive 2
1GB	\$35	N/A
2GB	\$45	\$55
4GB	\$55	\$65
8GB	\$75	\$85

Figure 6.9.9.7 Price tag for both SBCs

Real-Time Performance issue

Low-latency detection and speed calculation are crucial for providing immediate feedback to drivers. Although the webcam running at 30fps is able to capture almost all vehicles travelling in medium to low speed, it will still miss captured vehicle travel at high speed. High frame

rates are necessary for accurate vehicle tracking. Changing to better Webcam could solve this issue. During night, the low light environment will cause the camera to capture at 15fps, which will cause incorrect speed reading or even worse, unable to capture the vehicle. This is a hardware limitation of the webcam module. This webcam's sensor focuses on capturing more light per frame rather than maintaining FPS even though the auto exposure is disabled. Changing to a better low light performance camera which is design for low light environment, and dynamic FPS function which adjusts the calculation according to the fps capture by the camera should work in this case.

Data Calibration and the way of car traveling

The third issue is Data Calibration and Accuracy for Speed Calculation. Due to lacking of Doppler Speed Radar Sensor for the speed detection system, accurate calibration becomes important for converting pixel displacement into real-world speed; errors in calibration can lead to significant discrepancies in measurements. Careful calibration of camera placement, field of view, and reference distances is crucial. Establishing fixed markers within the camera's view will aid in accurately converting pixel data into real-world units. Regular recalibration is also necessary to maintain precision. Also, for the mounting method that we perform testing results, we saw that some of the vehicles that travel at the slight left, or slightly right at the road, will give inaccurate readings. The system calibration is set to the middle of the road to get the average value, which means we cannot get the perfect accurate reading.

Recording Footage for the VFv2

Finally, during the implementation of recording functions on VFv2, bug encountered. It looks like the bug is due to OpenCV's error where it could not save the file locally. Many methods were used to solve this issue including permission issue, changing file format, modifying resolution, it could finally output the file, but the file was unable to read using media player.

Recording Footage for the VFv2

The build in recording function comes with the OS is used to record vehicle that passes through to analyze the performance of this device. However, I have faced another issue, where the recording function will not work without monitor plugged into the SBC, without a monitor connected, it will only record black screen footage without any information. I would need to

have a HDMI to VGA adapter to trick the system to think the monitor is still plugged in. Next, I also found that the video recording will start to get distorted after 10 minutes of recording.

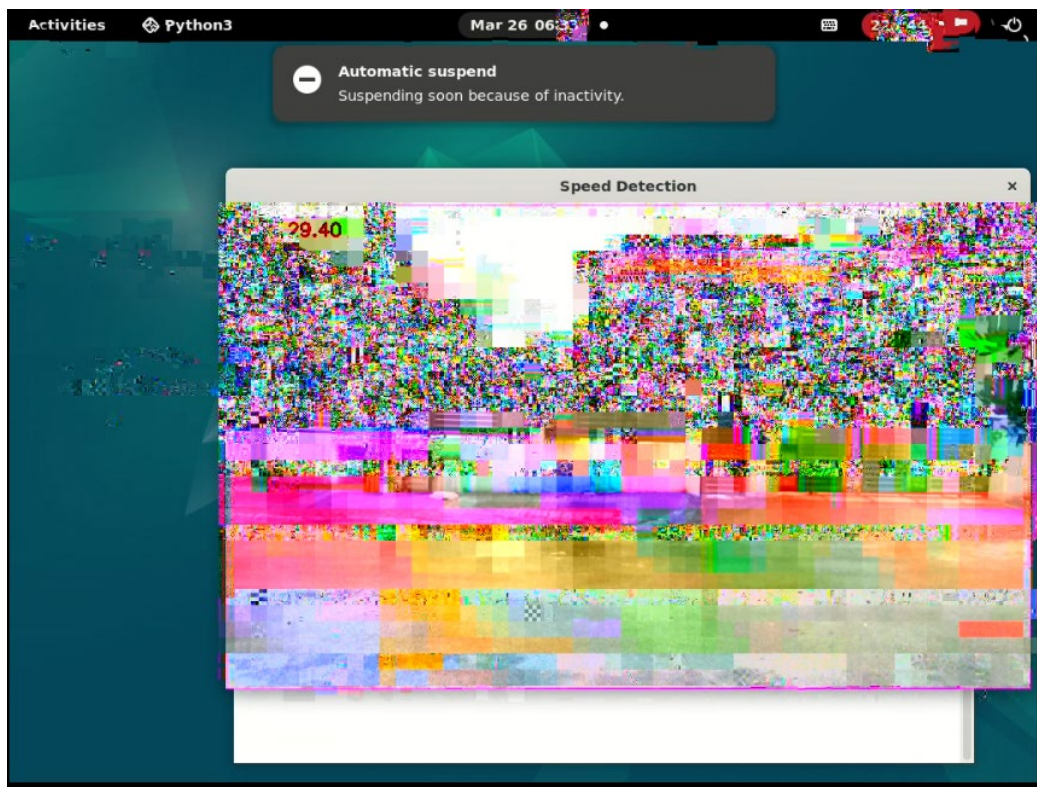


Figure 6.9.9.8 Image distortion when perform recording on VFv2

This had been solved by limiting the resolution of the recording, I adjust the recording system to record only portion of the screen which covers the program's windows, this can temporary solve the video distortion issue, but after 7 hours of recording, the distortion will reappear again slowly. This is likely caused by bottleneck of the CPU or memory where it records high resolution using software-based encoders like x264. It might also be caused by encoding buffer overflow, where the data entropy and compression load increases after a long period of time.

Recording Footage for Raspi 4b

For Raspi 4B, due to storage space limitation (16GB SD card storage) it could not record for long period of time, and also to prevent the recording function from competing resources with the CPU that running Yolo, the build in RaspberryPi Connect software and laptop is used. Laptop will connect remotely to Raspi 4B and perform screen recording to get the result.

Program freezing

Program unresponsive bug also happens on VFv2 when it detects any object faster than the speed TM1637 could show, the program will become unresponsive, this can be fixed by setting an upper limit on the speed display to prevent crash if there is false reading when the detected speed is too high. Limits the max speed shown by the system to 99km/h fix the issue.

6.4 Objectives Evaluation

Detection Accuracy

Raspberry Pi 4B performs better with YOLO-based object detection consistently identified vehicles accurately without being affected by background movements (leaves, humans, auto gate). It was able to maintain speed calculation precision even when multiple objects were present.

While VisionFive V2 system based on motion detection (due to lack of computational power for YOLO), detected any moving object including leaves, gates, and even small objects like ants. This caused significant inaccuracies in speed reading and false triggers. Raspberry Pi 4B clearly offers higher precision detection and robustness against background noise compared to VisionFive V2.

This concludes that object detection is crucial in order to achieve better detection accuracy.

Low Light Performance

Raspberry Pi 4B's detection performance degraded significantly at night. The system experienced a drop from 30 FPS to 15 FPS, resulting in motion blur and occasional failure to detect fast-moving vehicles. The system performs inconsistent, where it could not detect vehicle movement sometimes. However, the YOLO model still partially recognized vehicle shapes.

VisionFive V2 suffered from the same FPS drop. Detection became worse because the system detected headlights and light reflections as moving objects, falsely interpreting them as vehicle movement. Speed reading was often incorrect in dark environments.

Both systems struggle in low-light environments, but Raspberry Pi 4B handles vehicle distinction slightly better because of object classification, whereas VisionFive V2 struggles heavily without object filtering.

Vehicle Detection Rate

Raspberry Pi 4B capable of detecting a wide range of vehicles like cars, bicycles, and motorbikes. However, it sometimes failed to recognize certain pickup trucks due to dataset limitations in YOLO training. Overheating during long tests caused frame drops, affecting detection reliability.

VisionFive V2 had achieved 100% detection rate of any movement during daytime but had no filtering. False positives increased significantly where any motion will be detected and calculated speed. When multiple moving vehicles appeared, the system often failed to assign a correct speed reading.

Raspberry Pi 4B provides better selective detection, while VisionFive V2 detects every movement indiscriminately.

System Stability and Recording Capability

VisionFive V2 screen recording function implemented via the OS could record initially but suffered distortion after prolonged recording (around 7 hours) due to likely encoder buffer overflows or cumulative system memory issues. Smaller recording windows mitigated the issue temporarily.

Raspberry Pi 4B used remote laptop recording via Pi Connect due to limited onboard storage (16GB) and to prevent CPU overload. However, network-induced latency slightly degraded the screen recording's smoothness.

Category	Raspberry Pi 4B	Vision Five V2
Overheating	High, needs better cooling	Low, passive cooling
Processing Power	Good for YOLO Tiny	Poor for YOLO
Recording Capability	Remote via Pi Connect	Local but unstable for >7h
Object Filtering	Accurate	Poor , will detects any movement

Table 6.4 Evaluation table between two SBCs

6.5 Concluding Remark

This project aimed to develop a cost-effective and efficient vehicle speed detection system using a single-board computer (SBC) integrated with a camera and a 2-digit 7-segment display. The primary objective was to create a solution capable of enhancing road safety in school zones by identifying and displaying vehicle speeds in real time.

This project offers a practical solution to address over speeding issues in school zones, leveraging cost-effective hardware and open-source tools. It shows that the balance between speed and detection accuracy is a trade-off, especially on hardware-limited platforms like the Raspberry Pi. Fine-tuning the parameters and potentially retraining models of YOLO could theoretically improve results. Incorporating YOLO and preprocessing techniques can address most limitations, but real-time performance will still require lightweight models or better hardware. Raspberry Pi which is ARM platform, benefits from strong software optimization, better GPU support, it could still provide higher FPS in object detection tasks and more stable driver support for camera and GPIO. Raspberry Pi 4B remains the better choice for AI-based applications due to its mature ecosystem.

For Vision Five v2, which uses an emerging platform RISC-V, lack of optimized library. The software compatibility issue and weaker CPU performance lead to very low FPS when running YOLO. After implementing software optimizations like background subtraction, and multi-threaded processing, it can still perform real-time speed detection efficiently. But without YOLO object detection, any moving object including moving leaves will be detected as motion and will calculate speed which leads to false detection. This platform faces software and hardware limitations that restrict its capabilities for demanding AI tasks like YOLO-based object detection.

This project successfully demonstrated the potential for a low-cost, scalable speed detection system designed for school zones. While challenges remain, the findings lay a solid foundation for further enhancements, contributing to safer road environments and showcasing the capabilities of SBC-based solutions in real-world applications.

CHAPTER 7

7.1 Conclusion

The vehicle speed detection system developed in this project achieved the intended objectives by successfully demonstrating the capability to detect moving vehicles and estimate their speeds in real time using embedded single-board computers. Both the Raspberry Pi 4B and Vision Five V2 platforms were explored, designed, and rigorously tested under various environmental and operational conditions.

The Raspberry Pi 4B, with its higher computational power, enabled the integration of object detection through the YOLO model, resulting in greater detection accuracy and selective speed measurement. However, challenges such as overheating and reliance on external systems for video recording highlighted its limitations for long-duration, standalone deployments.

The Vision Five V2, despite its limited processing capabilities, proved to be a stable and energy-efficient platform suitable for continuous operation during daytime conditions. The RISC-V architecture presents challenges such as limited software support, lower processing power, and lack of GPU acceleration. Its performance was optimized through software techniques such as resolution reduction, multithreading, and dynamic frame management. However, its dependence on simple motion detection instead of object classification resulted in higher false detection rates, particularly in complex or low-light environments. Compared to the Raspberry Pi 4B, which benefits from better software optimization, higher FPS, and GPU support, the Vision Five V2 still struggles with real-time AI inference. Despite this, the project highlights the potential of RISC-V single-board computers for edge computing and lightweight vision applications, particularly as software support improves over time. While Vision Five V2 is not yet ideal for high-performance AI workloads, this project successfully demonstrated that real-time speed detection is possible through software optimization. As RISC-V hardware and software ecosystems mature, we can expect better AI acceleration and improved support for computer vision applications, making it a viable alternative for future projects.

Throughout the project, valuable engineering skills were developed, including real-time video processing, hardware-software integration, performance optimization, and system troubleshooting under constrained resources. The project's findings emphasize the importance of carefully matching hardware capabilities with application requirements, and the need for robust system design when operating in diverse real-world conditions.

In conclusion, this project has successfully delivered a functional prototype capable of real-time speed detection, and it offers a strong foundation for further research and improvement, such as upgrading optical sensors, implementing lightweight deep learning models, or adding dedicated AI accelerators to enhance performance and reliability for future intelligent traffic monitoring systems.

7.2 Recommendations

To improve the Vehicle Speed Detection System, we can Enhancing Detection Accuracy by Fine-tune the motion tracking parameters to reduce false detections. This can be done by experimenting with higher-resolution background models for better object differentiation.

An alternative hardware for AI acceleration, for example the Raspberry PI AI HAT+, can offload the OpenCV object detection processing to the AI acceleration module and free CPU resources. The additional CPU resources can be used to increase the input resolution from the camera and FPS which will help increase the detection rate. However, it only supports Raspberry Pi 5.



Figure 7.1 Raspberry PI AI HAT+

For VFv2, future research & development of RISC-V, implementing RISC-V GPU acceleration when future software optimizations become available, or implement it on a future Vision Five SBCs with better CPU performance. There might be a custom AI accelerator module design for AI workloads that can work with Vision Five V2 in the future which can help accelerate the YOLO workloads.

CHAPTER 7

Upgrade to Higher-Performance Hardware such as Nvidia Jetson Nano, the system will benefit from more powerful platforms that include built-in AI acceleration to enable real-time deep learning-based object detection without significant performance trade-offs.

Implement the system with doppler sensor is still the most cost effective way currently. Doppler sensors provide more accurate and reliable speed measurements regardless of lighting, weather conditions, or background complexity. This would significantly reduce false readings caused by irrelevant motion and enhance the overall robustness of the system, and no calibration needed for different mounting point like the camera.

REFERENCES

- [1] *Detect speed with a raspberry Pi, camera and OpenCV.* (2023, February 16). Core Electronics. <https://core-electronics.com.au/guides/detect-speed-raspberry-pi/>
- [2] *How to make vehicle speed detector system using raspberry Pi.* (2023, September 11). Electronics For You. <https://www.electronicsforu.com/electronics-projects/diy-vehicle-speed-detection-device-with-raspberry-pi>
- [3] *Vehicle speed detection system utilizing YOLOv8: Enhancing road safety and traffic management for metropolitan areas.* (n.d.). arXiv.org e-Print archive. <https://arxiv.org/html/2406.07710v1>
- [4] Neuvision. (2023, October 17). *Real-time traffic monitoring using roadside-mounted LiDAR sensors.* Neuvision | solid-state lidar, lidar sensor suppliers, lidar technology, lidar sensor. <https://cdn.neuvition.com/media/blog/real-time-traffic-monitoring-using-roadside-mounted-lidar-sensors.html>
- [5] Lindquist, J. (2022, October 27). *Police speed guns: Differences between LIDAR and RADAR?* Kustom Signals Inc. <https://kustomsignals.com/blog/radar-vs-lidar-are-there-any-significant-differences-between-them-to-detect-objects>
- [6] GceLab. (n.d.). *What is inductive loop detector? 8 important points.* Gurukul of Civil engineers- Online Civil Engineering Courses for Creative Minds. <https://www.gcelab.com/blog/what-is-inductive-loop->

Reference

[detector-and-advantages](#)


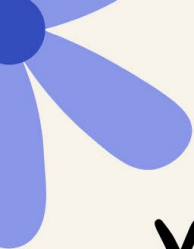
- [7] Elsevier. (2018, July 18). Vehicle speed measurement model for video-based systems.

ScienceDirect. <https://www.sciencedirect.com/science/article/pii/S0045790618317774>

- [8] *How to estimate speed with computer vision*. (2024, April 15).

Roboflow Blog. <https://blog.roboflow.com/estimate-speed-computer-vision/>

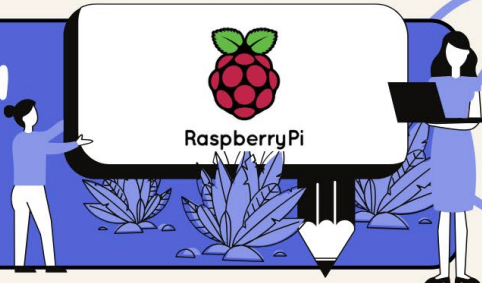
POSTER



Vehicle Speed Detection using Machine Vision on SBC

SBC : Raspberry Pi

- A low power
- Cost effective
- Open source
- Real time analysis



System Features :

- Hardware: Raspberry Pi, Camera Module, 7-Segment Display
- Technology: YOLO for real-time object detection
- Performance: High accuracy at low speeds during good lighting condition

Use case

- Traffic management
- Speed monitoring in residential areas
- Integration with smart city systems

