

**Application of Intelligent Gesture Control to
Simultaneously Control Multiple Remote Devices**

LIM YUEH SHENG

UNIVERSITI TUNKU ABDUL RAHMAN

**Application of Intelligent Gesture Control to Simultaneously Control
Multiple Remote Devices**

LIM YUEH SHENG


**A project report submitted in partial fulfilment of the
requirements for the award of
Bachelor of Engineering (Honors) Electronic and Communications
Engineering**

**Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

May 2022

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :  _____

Name : LIM YUEH SHENG

ID No. : 1804715

Date : 15 May 2022

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**Application of Intelligent Gesture Control to Simultaneously Control Multiple Remote Devices**” was prepared by **LIM YUEH SHENG** has met the required standard for submission in partial fulfilment of the requirements for the award of **Bachelor of Engineering (Honours) Electronic and Communications Engineering** at Universiti Tunku Abdul Rahman.

Approved by,

Signature :



Supervisor :

Dr. LAI AN CHOW

Date :

12 May 2022

Signature :



Co-Supervisor :

IR. DR. THAM MAU LUEN
ASSISTANT PROFESSOR
LEE KONG CHIAN FACULTY OF ENGINEERING AND SCIENCE
UNIVERSITI TUNKU ABDUL RAHMAN

Date :

15/5/2022

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2022, LIM YUEH SHENG. All right reserved.

ABSTRACT

The famous superhero Iron Man has two well-known icons, Iron Man suits and the super artificial intelligent J.A.R.V.I.S. that Tony Stark can either command it using his voice or swing his hand over the air. Although hand gesture control over hologram display might still be difficult to achieve, hand gesture technologies are already very mature and controlling our daily devices using hand gestures is achievable and practical. This project will show how to control a device or multiple devices using only a webcam and our hand gestures. The hand detection model used in this project is MediaPipe Hands. MediaPipe Hands is a high-fidelity hand and finger tracking solution. Unlike other object detection models, MediaPipe Hands marks and coordinates 21 landmarks in our hands, including every joint of our fingers. We designed a series of tools that detect and design hand gestures using these coordinations. After designing a hand gesture, the next step is designing an application and putting in the hand gestures we designed as control. We designed two applications to demonstrate hand gestures control: driving mode and precision mode. Driving mode demo the immersive and interactive characteristic of hand gesture control. In contrast, precision mode demo precision of hand gestures can achieve. The device we use to demonstrate is a wireless robot car. This report will demonstrate the capability and practicality of hand gesture control and evaluate the potential and limitations of hand gesture control.

TABLE OF CONTENTS

| | | |
|--|---|-------------|
| DECLARATION | | i |
| APPROVAL FOR SUBMISSION | | ii |
| ABSTRACT | | iv |
| TABLE OF CONTENTS | | v |
| LIST OF TABLES | | viii |
| LIST OF FIGURES | | ix |
| LIST OF SYMBOLS / ABBREVIATIONS | | xii |
| LIST OF APPENDICES | | xiii |
| | | |
| CHAPTER | | |
| 1 | INTRODUCTION | 1 |
| 1.1 | General Introduction | 1 |
| 1.2 | Neural networks and CNN | 2 |
| 1.3 | Importance of the Study | 3 |
| 1.4 | Problem Statement | 4 |
| 1.5 | Aim and Objectives | 4 |
| 1.6 | Scope and Limitation of the Study | 4 |
| 2 | LITERATURE REVIEW | 6 |
| 2.1 | Sensor Technologies | 6 |
| 2.1.1 | Single Camera (Vision based, 2D-based) | 7 |
| 2.1.2 | Stereo camera (Vision based, 2D & 3D-based) | 7 |
| 2.1.3 | Depth Sensor (Vision based, 3D-based) | 7 |
| 2.1.4 | Time-of-Flight Sensors (ToF) | 7 |
| 2.1.5 | Leap Motion | 8 |
| 2.1.6 | Marker (Non-Vision Based, 3D-based) | 9 |
| 2.2 | Gesture Identification | 9 |
| 2.2.1 | Spatial modelling and Temporal Modelling | 10 |
| 2.2.2 | Color Identification (Spatial/Visual feature) | 10 |

| | | |
|----------|---|-----------|
| | 2.2.3 Feature extraction (Spatial/Visual feature) | 11 |
| | 2.2.4 Skeletal Model (3D-Spatial Model) | 12 |
| 2.3 | Gesture classification | 12 |
| | 2.3.1 Nearest Neighbor Classifier | 13 |
| | 2.3.2 Convolutional Neural Networks (CNN) | 14 |
| 2.4 | CNN architectures | 17 |
| | 2.4.1 You Only Look Once algorithm (YOLO) | 17 |
| | 2.4.2 Single Shot MultiBox Detector (SSD) | 20 |
| | 2.4.3 MediaPipe Hands | 21 |
| 2.5 | Architectures comparison | 24 |
| 3 | METHODOLOGY AND WORK PLAN | 25 |
| 3.1 | Project overview | 25 |
| 3.2 | Hardware Overview | 25 |
| | 3.2.1 ESP32 DEV KIT DOIT | 26 |
| | 3.2.2 L298N DC Motor Driver | 27 |
| 3.3 | Software Overview | 30 |
| | 3.3.1 Arduino IDE | 30 |
| | 3.3.2 MQTT | 30 |
| | 3.3.3 OpenCV | 30 |
| | 3.3.4 MediaPipe Hand | 31 |
| 3.4 | Methodology | 31 |
| | 3.4.1 Image Pre-processing | 31 |
| | 3.4.2 Call Model and Extract Data | 32 |
| | 3.4.3 Output Data Processing and Organizing | 35 |
| | 3.4.4 Hand angle | 37 |
| | 3.4.5 Angle of 3 points | 39 |
| | 3.4.6 Detect Finger Up | 39 |
| | 3.4.7 Distance of 2 point | 41 |
| | 3.4.8 MQTT setup | 42 |
| | 3.4.9 DC Motor Configuration (Arduino IDE) | 42 |
| 4 | RESULTS AND DISCUSSION | 44 |
| 4.1 | Introduction | 44 |
| 4.2 | Driving Mode | 45 |
| | 4.2.1 Starting Driving Mode | 45 |

| | | |
|----------|--|-----------|
| 4.2.2 | Control And Action in Driving Mode | 46 |
| 4.2.3 | Sensitivity of Driving Mode | 47 |
| 4.2.4 | Accuracy of Driving Mode | 49 |
| 4.2.5 | Limitation of Driving Mode | 49 |
| 4.3 | Precision Mode | 50 |
| 4.3.1 | Starting Precision | 51 |
| 4.3.2 | Control and Action in Precision Mode | 51 |
| 4.3.3 | Sensitivity of Precision Mode | 52 |
| 4.3.4 | Accuracy of Precision Mode | 53 |
| 4.3.5 | Limitation of precision mode | 55 |
| 4.4 | General Parameters | 56 |
| 4.4.1 | MQTT communication delay | 56 |
| 4.4.2 | The Smoothness of The User Interface | 57 |
| 4.5 | Summary | 58 |
| 5 | CONCLUSIONS AND RECOMMENDATIONS | 59 |
| 5.1 | Conclusions | 59 |
| 5.2 | Recommendations for future work | 60 |
| | REFERENCES | 62 |
| | APPENDICES | 66 |

LIST OF TABLES

| | | |
|------------|---|----|
| Table 3.1: | Detailed Cost Breakdown for each component. | 26 |
| Table 3.2: | Details Description for model configuration. | 33 |
| Table 3.3: | Detection of tip & MCP joint to decide if the finger is extended based on hand orientation. | 40 |
| Table 3.4: | DC Motor signal configuration. | 43 |
| Table 4.1: | Voltage & Speed relationship table (Based on actual testing). | 46 |
| Table 4.2: | DC Motor PWM configuration. | 47 |
| Table 4.3: | Control and Action in Precision Mode. | 51 |
| Table 4.4: | DC Motor behaviour mapping. | 54 |

LIST OF FIGURES

| | |
|--|----|
| Figure 1.1: Overview of the interaction between Machine and Human. | 2 |
| Figure 1.2: Optical flex sensor. (Premaratne, Nguyen and Premaratne, 2010) | 2 |
| Figure 2.1: A process model of gesture recognition for human-robot collaboration. (Liu and Wang, 2018). | 6 |
| Figure 2.2: Leap Motion Controller. (<i>HERO-UltraLeap_Product05342_edit.jpg (1600×1067)</i> , no date) | 8 |
| Figure 2.3: Marker for Motion Detection and Face Motion Detection. (<i>FAR CRY 6 Behind The Scenes Face Model Motion Capture Trailer Giancarlo Esposito Antó - YouTube</i> , no date) | 9 |
| Figure 2.4: LEFT: Spatial models of gestures (Pavlovic, Sharma and Huang, 1997) RIGHT: Hand Modelling (Hasan and Mishra, 2012) | 10 |
| Figure 2.5: Multicolored glove (<i>Analog computing returns MIT News Massachusetts Institute of Technology</i> , no date) | 11 |
| Figure 2.6: Left: Zoning. Right: example for loop, junction and branch (Kumar and Bhatia, 2014) | 12 |
| Figure 2.7: Left: 3D wireframe Volumetric model. Right: Skeletal model (Pavlovic, Sharma and Huang, 1997). | 12 |
| Figure 2.8: Image classification in the point of view of machine. (F. Li, 2020a). | 13 |
| Figure 2.9: Operation of K-Nearest Neighbors. Datacamp article: (<i>KNN_finall_ibdm8a.png (591×515)</i> , no date) | 14 |
| Figure 2.10: Illustration of comparing two images pixel by pixel (F. Li, 2020b) | 14 |
| Figure 2.11: Left: Perceptron Source: QUANTSTART: Introduction to Artificial Neural Networks and the Perceptron. Right: three layers ANN (F. Li, 2020b) | 15 |
| Figure 2.12: Structure of CNN. (Saha, 2018) | 15 |
| Figure 2.13: 2x2 max pooling with stride of 2. (F. Li, 2020b) | 17 |

| | |
|---|----|
| Figure 2.14: YOLO evaluation on PASCAL VOC using S=7, B=2, C=20. (PASCAL VOC has 20 labelled classes) (Redmon <i>et al.</i> , 2016). | 20 |
| Figure 2.15: Comparison of AP in various algorithm. (Redmon and Farhadi, 2018), (Lin <i>et al.</i> , 2020) | 20 |
| Figure 2.16: Comparison of SSD architecture and YOLO architecture. (Liu <i>et al.</i> , 2016). | 21 |
| Figure 2.17: Left: MediaPipe framework. (Lugaresi <i>et al.</i> , 2019). Right: MediaPipe Hands framework. (Zhang <i>et al.</i> , 2020) | 23 |
| Figure 2.18: Hand landmark model 21 landmarks. (Inc., 2020) | 23 |
| Figure 2.19: (Left): Hand landmarks with relative depth presented in different shades. The lighter and larger the circle, the closer the landmark is towards the camera. (Right): Real-time multi-hand tracking. Source: (Zhang <i>et al.</i> , 2020) | 23 |
| Figure 2.20: CNN architectures comparison chart | 24 |
| Figure 3.1: Project overview | 25 |
| Figure 3.2: WiFi Robot Car Set up | 26 |
| Figure 3.3: Complete pinout of ESP32 DEV KIT DOIT (Santos, 2018) | 27 |
| Figure 3.4: Duty Cycle in PWM (Last Minute Engineers, 2021). | 29 |
| Figure 3.5: H-Bridge working principle (Last Minute Engineers, 2021). | 29 |
| Figure 3.6: L298N DC Motor Driver Pinout (Last Minute Engineers, 2021). | 29 |
| Figure 3.7: Layers of MediaPipe Hand solution. | 34 |
| Figure 3.8: Model Configurations. | 34 |
| Figure 3.9: Steps for process and organize data. | 36 |
| Figure 3.10: Data Structure of Output raw data and Data Structure after Data Processing. | 37 |
| Figure 3.11: Angle $\angle ABC$ | 39 |
| Figure 3.12: Point choose to calculate Hand Angle | 39 |
| Figure 3.13: Examples of hand gestures in sign language (Yasen and Jusoh, 2019) | 41 |

| | |
|---|----|
| Figure 4.1: Gesture to activate Driving Mode | 45 |
| Figure 4.2: Controlling speed in driving mode. | 47 |
| Figure 4.3: Driving Mode Angle Reference. | 48 |
| Figure 4.4: Sensitivity of Driving Mode. | 48 |
| Figure 4.5: Gesture to activate Precision Mode. | 51 |
| Figure 4.6: Operation in precision mode | 52 |
| Figure 4.7: Sensitivity at the edge of the screen | 52 |
| Figure 4.8: Demonstration example for precision mode. | 54 |
| Figure 4.9: Result of Precision Mode. | 55 |
| Figure 4.10: MQTT message delay | 57 |
| Figure 4.11: FPS in idle, one hand detected and two hands detected. | 57 |

LIST OF SYMBOLS / ABBREVIATIONS

| Abbreviations | Definition |
|----------------------|-------------------------------|
| ANN | Artificial Neural Network |
| CNN | Convolutional Neural Network |
| HMI | Human–Machine Interaction |
| HMM | Hidden Markov Model |
| IOT | Internet Of Things |
| IOU | Intersection Over Union |
| NN | Neural Network |
| SSD | Single Shot Multibox Detector |
| VR | Virtual Reality |
| WHO | World Health Organization |
| YOLO | You Only Look Once Algorithm |

LIST OF APPENDICES

| | |
|---|----|
| Appendix A: Coding for MediaPipe Hand custom function | 66 |
| Appendix B: Coding for Set Up MQTT | 72 |
| Appendix C: Main Coding for Two Applications | 72 |
| Appendix D: Coding for WiFi Car using Arduino IDE | 78 |

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Since COVID-19 pandemic impacts the globe, zero contact or contactless between human is being encouraged to prevent the spread of viruses. Contactless delivery become more and more popular, even World Health Organization (WHO) disseminates the information that avoid physical contact when greeting. Human-machine interaction (HMI) is the interaction between human and machine, is how we interact or communicate with machine (Johannsen, 2011). Figure 1.1 clearly shows us the overview of HMI. HMI is a field that advances very quickly, researcher always searching or improve different method to have a more efficient and precise control on machine or robot. Pandemic forces many people work from home, which greatly increase the interaction between machine and human, and therefore further accelerates the development in HMI.

Based on history of HMI, we can see that the input evolves from fixed set of instruction (computer punched card) to instruction that can based on human interaction (keyboard & mouse) to higher degree of interaction not constraint by key binding and mouse (touchscreen and sound recognition). As the method of interaction evolves, the input freedom and the precision increase. Example of precision increase is mouse control to touchscreen, touchscreen and pinpoint the location we desire more accurately. Example of increase input freedom can be seen from voice control. Nowadays, voice control not only support different language, also robust enough to recognize various accent.

Hand gesture control have a long history. Back in 1983 the first hand-gesture control appears in a form of glove. The glove contains many proximities sensor such as tilt sensor to detect user movement (Premaratne, Nguyen and Premaratne, 2010). Figure 1.2 show hand gesture glove use at that time.

At that time, the accuracy of hand gesture was poor, therefore usage was limited. As the technology of image sensor improved, visual based hand gesture recognition become more popular. Other than that, thank to

improvement of computation power and artificial intelligence, pre-processing of the image become easier and gesture classification become more accurate.

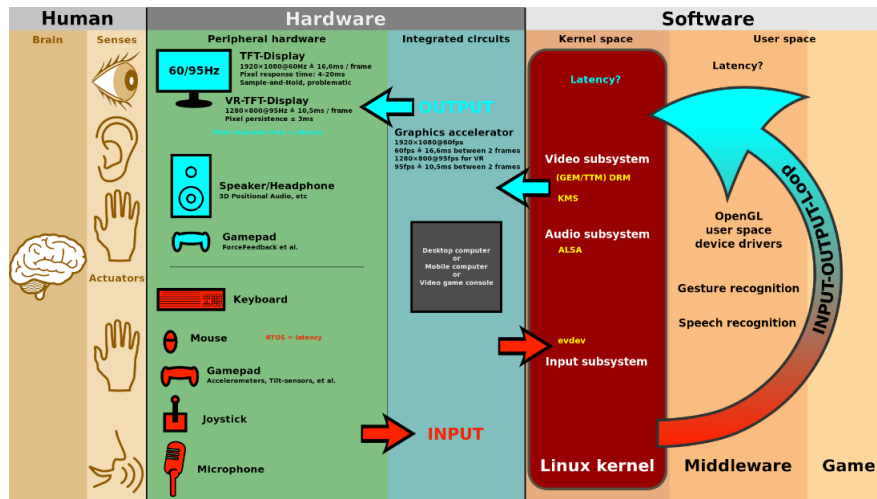


Figure 1.1: Overview of the interaction between Machine and Human.

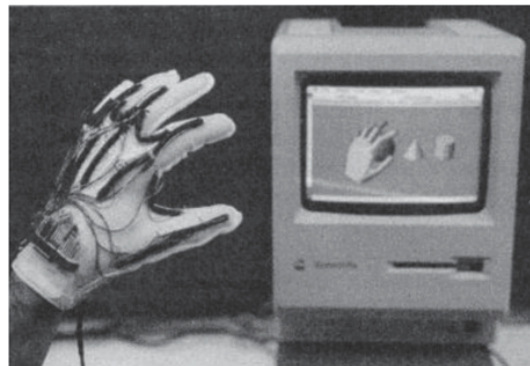


Figure 1.2: Optical flex sensor. (Premaratne, Nguyen and Premaratne, 2010)

1.2 Neural networks and CNN

Before the appearance of convolutional neural network (CNN), vision-based detection can be tedious mainly because we need to design two systems for it; one for extract feature and another use as classifier (Yingxin *et al.*, 2016). The complex design and many image pre-processing works need to be done make vision-based detection have high using threshold, only professional researcher able to work on it, in addition that the performance is mediocre. Appearance of CNN present a huge success in term of performance in extract features and classifications (Lawrence *et al.*, 1997). This stimulates the research in computer vision and its applications.

Today, CNN is one of the fastest growing sectors in deep learning and have indispensable position in deep learning (Li *et al.*, 2021). CNN can be used in many applications, especially in computer vision. Many architectures and networks have been developed and used in different applications. Some applications include safety on autonomous vehicle, vehicle plate recognitions and even latest research study on COVID-19 detection using CNN with X-ray. Later in literature review, we will dive deep into different type of CNN networks and choose the one that fits our applications.

1.3 Importance of the Study

In 2007 the first iPhone released. Before that, most of the device was controlling using button, keyboard or mouse. Smartphone make touchscreen becoming one of the most common interactions between human and machine. Virtual assistance such as Siri, Alexa or google assistance use voice recognition as primary input. In the future, touchscreen, controller or keyboard might not be the only way to control device. Hand gesture might be better in some usage. Sci-Fi movies always use hand gesture combine with holographic display to represent futuristic feel. This might be practical as holographic technologies become more advance and mature.

Other than that, currently most of the Internet of things (IOT) product in the market used smartphone as controller. This can be inconvenient in some situation. For example, if we are controlling a device that requires using both hand, such as Arduino car that have multiple and complex control. Smartphone also have some constraint such as it requires take it out, unlock and open the specific apps in order to use it.

Furthermore, unlike adult trained to use keyboard and mouse. For children, hand gesture has easier learning curve for them to interact with machine. Another limitation for keyboard and controller is lack of ability to customize the control. Control using gesture is customizable, and the solution is adaptive based on the user. Hand gesture study is important, as the technologies become more advance, strength of hand gesture will become more obvious, weakness will be fixed and can be used in more and more applications.

1.4 Problem Statement

Real-time hand gesture control is still not common in the market. Products that use hand gesture control available still got room for improvement. Using hand gesture controlling multiple devices is even rarer. Most of the devices have fixed control and command. Intelligent Gesture Control allows user to customize command, based on their habit or usage, flexible enough for different solutions.

Customizable solution become even more crucial when we discuss controlling multiple devices. Mobility of left hand and right hand is often uneven, same for gesture pose that can be made by left and right hand. Customizability allows control that fits user habit, eliminating the learning cost of using it.

1.5 Aim and Objectives

The main objective of this project is to build a system that can use hand gesture to control multiple devices. The goal can breakdown as follows:

- (i) To investigate gesture recognition system.
- (ii) To develop a real time recognition algorithm which controls multiple devices.
- (iii) To evaluate the performance of the algorithm in terms of recognition accuracy.

1.6 Scope and Limitation of the Study

In this project, hand gesture recognition consists of three parts, (1) camera sensor to capture the image and algorithm that process the image, (2) recognize image and convert it to a command that will control device or devices. (3) Last part will be sent control signal to devices.

First of all, due to limitation of cost. Only single camera sensor will be used. Using single camera sensor have some trade-off such as angles and depth, which will be discuss more details when coming to literature reviews.

Next, for real-time application, the hand recognition process needs to avoid noticeable delay. Delay time will be decided by algorithm and computer processing power. We have limited access to the UTAR computer lab due to the COVID-19 pandemic. Therefore, this project will focus on achieving the

accuracy and resolution of the hand gesture detection as high as possible with limited computer processing power.

CHAPTER 2

LITERATURE REVIEW

2.1 Sensor Technologies

Using gestures to control devices is a type of human-robot collaboration. Figure 2.1 shows the process model of gesture recognition for human-robot collaboration. Before any gesture recognition start, the first step is collecting data from the sensor. Papers like (Liu and Wang, 2018), (Xia *et al.*, 2019), (Chen *et al.*, 2013), (Patel and He, 2018), (Badi, 2016), have summarized various types of sensor technologies to collect data. Different research papers categorize sensors differently; one way is to categorize into image based and non-image based, another way is to categorize into 2D-based and 3D-based. In this FYP report we will briefly go through all the sensors and look for the one that suit our project.

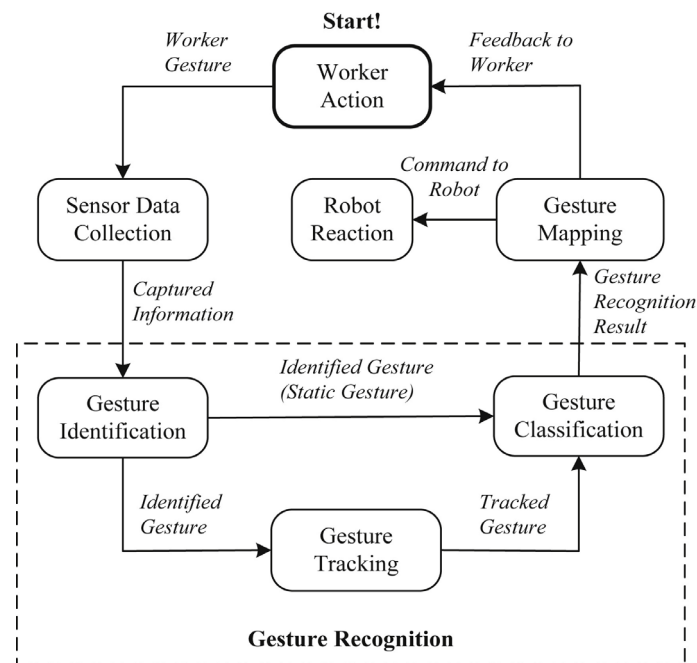


Figure 2.1: A process model of gesture recognition for human-robot collaboration. (Liu and Wang, 2018).

2.1.1 Single Camera (Vision based, 2D-based)

Single-camera is one of the most common sensors that can be used to collect hand gesture data. Nowadays, every smartphone has at least one camera sensor. Due to the easy access and its decent precision, single-camera become the most widely used method. Characteristic of single-camera are:

- (i) Common, easy to access, cheap.
- (ii) Have decent precision.
- (iii) Restricted on view angle.
- (iv) Sensitive to lighting, minor lighting issues can be solved by pre-processing the image.

2.1.2 Stereo camera (Vision based, 2D & 3D-based)

A stereo camera is a type of camera with two or more camera sensors. The working principle is similar to human eye binocular vision. If two or more camera sensors point in the same direction, distance or depth information can be extracted at the intersect area covered by two sensors. Characteristic of a stereo camera are:

- (i) Have higher robustness compare to single camera.
- (ii) Able to extract depth information .
- (iii) Have high requirements on calibration and correct placement for sensors.

2.1.3 Depth Sensor (Vision based, 3D-based)

Unlike stereo camera, depth sensor can output depth information directly without aid of computer processing. There are two types of depth sensor currently popular in the market, Time-of-Flight Sensors and Leap Motion.

2.1.4 Time-of-Flight Sensors (ToF)

In order to measure the depth of an object, infrared light is emitted, and a detector is used to detect the reflected light. By calculating the phase delay of the reflected light, we can know the depth of an object (Hansard *et al.*, 2012). Microsoft Kinect 2.0 is a ToF sensor. Compare to Kinect 1.0 (not ToF sensor) that uses Red-Green-Blue-Depth (RGB-D) mechanism. Kinect 2.0 has better precision and supports a higher frame rate. ToF handles poorly when came to

motion blur. When motion blur occurs in color, we will see that colors blend and have a color fading effect. However, when motion blur happens in depth images, overshoot or undershoot will happen in depth-transition regions, resulting in many errors. Characteristic of Time-of-Flight Sensors are:

- (i) Support higher frame rate.
- (ii) Direct output depth information.
- (iii) Accuracy is depending on power of the emitted IR light; environment light will interfere IR and might affect the result.
- (iv) Greatly affected by motion blur.

2.1.5 Leap Motion

Leap Motion (Figure 2.2) is a relatively new product compare to others. Unlike other sensors that can collect other information (such as body or objects), leap motion focuses on detecting hands only. It has outstanding precision and supports an even higher frame rate (fast detection). Leap Motion precision is in terms of submillimeter, and in the best scenario, it can achieve lower than 0.01mm accuracy (Guna *et al.*, 2014). Due to its fast-tracking and excellent precision, many researchers have been using this device on the project required hand motion to control and precision such as flight simulation system and surgery robot arms. Characteristic of Leap Motion are:

- (i) Submillimeter precision.
- (ii) Support High frame rate.
- (iii) Suitable for an immersive experience such as virtual reality (VR).
- (iv) Only able to detect Hand.



Figure 2.2: Leap Motion Controller. (HERO-UltraLeap_Product05342_edit.jpg (1600×1067), no date)

2.1.6 Marker (Non-Vision Based, 3D-based)

Motion Tracking Marker has been used for a long time in the film industry. This fundamental principle for motion tracking markers is rotoscoping. Rotoscoping is a technique that traces the layout of an object (human in most of the cases in film making) frame by frame. By combining all those frames, we can get an animation of the motion actor are doing. This method can only be seen in professional filmmaking or games cinematic cut-scene. It requires professionals to operate and set up. With proper setup, filmmakers can track very detailed motions. Figure 2.3 shows an example of motion tracking marker, a green color marker with a white dot to track body motion and black dots on the face to track face motion. Characteristic of Motion Tracking Marker are:

- (i) Low to almost real time latency.
- (ii) Extremely high details and precision.
- (iii) Require professional to setup.

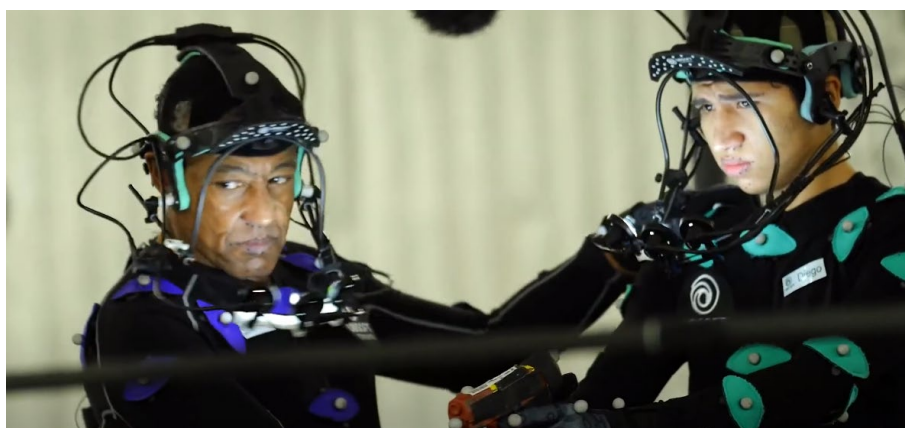


Figure 2.3: Marker for Motion Detection and Face Motion Detection. (*FAR CRY 6 Behind The Scenes Face Model Motion Capture Trailer Giancarlo Esposito Antó - YouTube, no date*)

2.2 Gesture Identification

After collecting the data (picture or video), we can start working on gesture recognition, and the first step of gesture recognition will always be gesture identification. For a machine, a hand is no different from other objects. For them, every data received is just a collection of pixels. Therefore, we need to create a model for the machine to identify which group of pixels is marked as “hand”. According to (Liu and Wang, 2018) there are three popular solutions

for gesture identification. Visual features, learning algorithms, and skeleton models. All the solutions are based on two fundamental modelling concepts, spatial modelling, and temporal modelling. Figure 2.4 shows classification of various hand modelling technique.

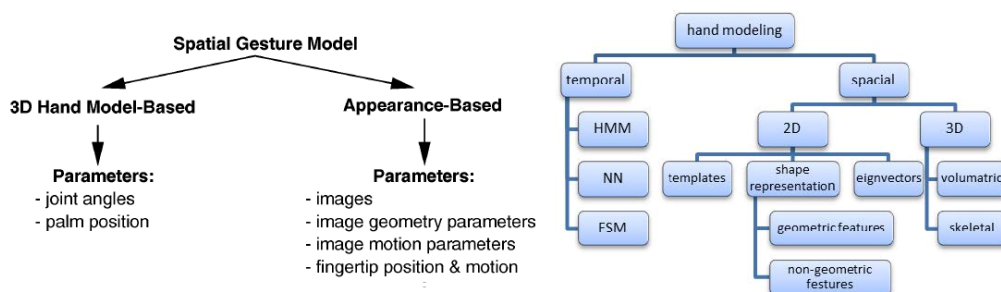


Figure 2.4: LEFT: Spatial models of gestures (Pavlovic, Sharma and Huang, 1997) RIGHT: Hand Modelling (Hasan and Mishra, 2012)

2.2.1 Spatial modelling and Temporal Modelling

The concept of spatial and temporal modelling is first mentioned in (Pavlovic, Sharma and Huang, 1997). At first, the concept of temporal modelling refers to continuous motion when a hand gesture is made. From resting posture to preparation movement, the hand either goes back to another resting posture or prepares for the next gesture movement (Pavlovic, Sharma and Huang, 1997). Spatial modelling on the other hand is based on the geometry properties such as shape, angle, position, etc., to define a gesture. At that time, temporal modelling is too complex to model and visual-based sensors often seen every frame as a static gesture, therefore temporal modelling is ignored.

As researches and computer hardware become more advance, temporal modelling become achievable. Hidden markov model (HMM), neural network (NN), and finite state machine are some of the examples of temporal modelling (Hasan and Mishra, 2012). Today, an advance gesture classification algorithm utilizes both spatial and temporal modelling characteristic to achieve high precision.

2.2.2 Color Identification (Spatial/Visual feature)

Color glove (Figure 2.5) is one of the simplest methods for gesture identification. With plain color background, user gestures can be easily

extracted by computer. This idea came from Robert Wang, MIT Electrical Engineering & Computer Science student when he sought for low-cost gesture hardware. Cost of the glove is only a dollar. Using this method are:

- (i) Low-cost, simple to setup.
- (ii) Good accuracy with plain background.
- (iii) Accuracy will be affected if there are color and lighting of the background.
- (iv) Color identification only achievable when wearing special glove.



Figure 2.5: Multicolored glove (*Analog computing returns | MIT News | Massachusetts Institute of Technology, no date*)

2.2.3 Feature extraction (Spatial/Visual feature)

Color identification is greatly affected by background and lighting. Hence, popular algorithms such as CNN use features to identify objects rather than color. Feature extraction is a huge topic in image processing. There are many approaches and techniques for feature extraction. The basic idea of feature extraction is that the machine will first segmentate the image. Next, it will study the feature of every grid. Study features are edge and shape, the gradient of the line, how many intersect, loop in the image, junction, and branches, etc. This idea can be extend to more complex features such as statistical feature, distance, connectivity, etc. to achieve better recognition (Kumar and Bhatia, 2014). Figure 2.6 shows examples of zoning and feature extraction.

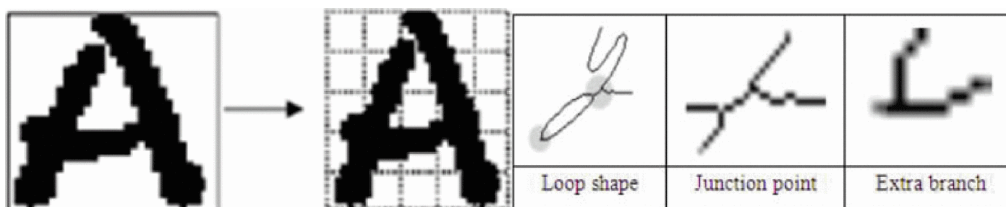


Figure 2.6: Left: Zoning. Right: example for loop, junction and branch (Kumar and Bhatia, 2014)

2.2.4 Skeletal Model (3D-Spatial Model)

Skeletal model (Figure 2.7) is a 3D-spatial model. The main advantage is that instead of using a volumetric model with countless points and parameters, the skeletal model focuses only on the joint, angle, and segment length. This greatly reduces the information we need to process while keeping all the essential data we need. By reducing the number of parameters, the time and computation power needed will be greatly reduced. Hence, it reduces the requirement for real-time applications.

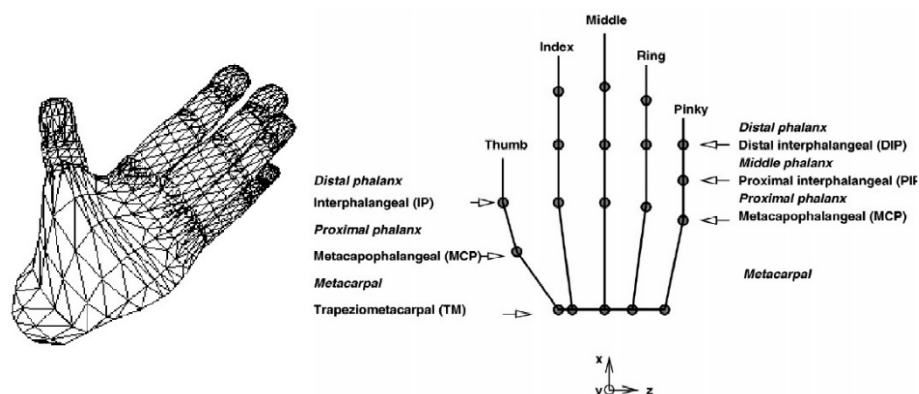


Figure 2.7: Left: 3D wireframe Volumetric model. Right: Skeletal model (Pavlovic, Sharma and Huang, 1997).

2.3 Gesture classification

Gesture classification is a process of labeling objects in the image and categorizes them. This is the last step but the most important step for gesture recognition. Machine will only know the meaning of the gesture after gesture classification. Machine see the image differently from humans. They see images as pixels and every pixel as 3-dimensional arrays of integers ranging

from 0 to 255 (if using RGB format). Figure 2.8 shows image classification in the point of view of machine.

They're always a challenge when discussing gesture classification. Even if we detect a stationary object, a different view angle will result in entirely different pixels, arrays, and numbers for a machine. The classifier should be robust enough to be still able to classify the same object. Furthermore, view angle/camera angle is not the only factor that impacts on the pixels. Scale difference, illumination, deformation, intra-class variation all these factors will affect the accuracy of the classifier (F. Li, 2020a). A good classifier should be robust enough to overcome all these factors. Nowadays, the classifier algorithm has improved and can reach high accuracy with recognition speed under a fraction of seconds.

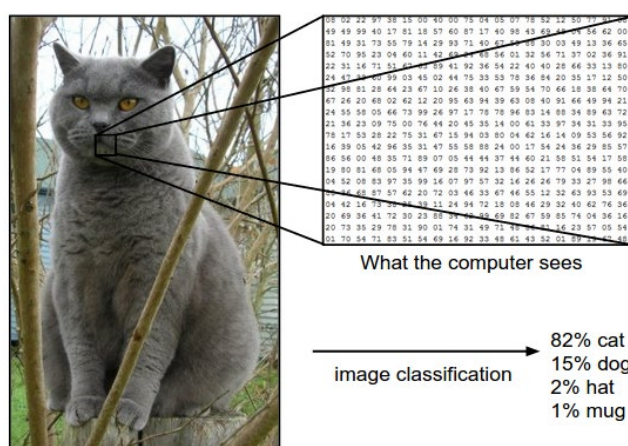


Figure 2.8: Image classification in the point of view of machine. (F. Li, 2020a).

2.3.1 Nearest Neighbor Classifier

Nearest Neighbor Classifier (Figure 2.9) is considered one of the most fundamental methods when we discuss classifiers. The idea is that we had our labeled dataset. When a new data came in, we classify it by looking at which dataset sits closest to it (have the minimum difference) and label the new dataset as that class. One of the simplest methods to compare two images is by comparing pixels by pixels and sum up all the differences. The distance or the difference between two images can be represented using equation below:

$$d1(I1, I2) = \sum_p |I_1^p - I_2^p| \quad (2.1)$$

where, I_1 , I_2 represent vectors of image 1 and image 2. Figure 2.10 shows an example of the operation.

K- Nearest Neighbor is an algorithm that follows this concept. The K here represents k-nearest numbers of points to compare. For example, if $K = 1$, the nearest point will decide the new dataset class. If $k = 5$, instead of using one nearest point, now use five nearest points to decide where the new dataset belongs. Using a higher number of k allows resistance for the outlier.

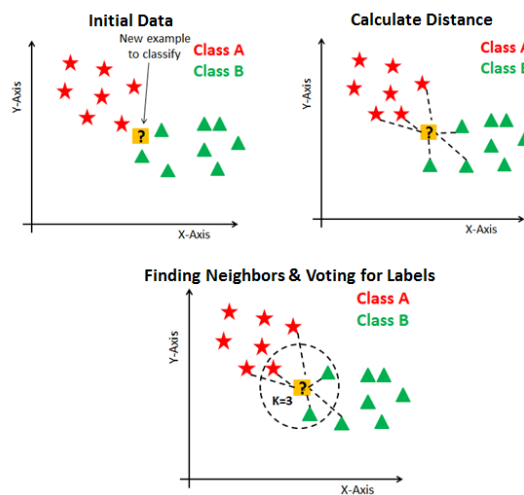


Figure 2.9: Operation of K-Nearest Neighbors. Datacamp article:

(*KNN_final_ibdm8a.png (591×515)*, no date)

| test image | - | training image | = | pixel-wise absolute value differences | → 456 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----------------|-----|---------------------------------------|-------|----|-----|-----|----|----|-----|-----|---|---|-----|-----|--|---|----|----|----|----|---|----|----|-----|----|----|-----|-----|---|----|-----|-----|--|--|----|----|----|---|----|----|----|----|----|----|---|----|---|----|----|-----|--|
| <table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 10px;">56</td><td style="padding: 2px 10px;">32</td><td style="padding: 2px 10px;">10</td><td style="padding: 2px 10px;">18</td></tr> <tr><td style="padding: 2px 10px;">90</td><td style="padding: 2px 10px;">23</td><td style="padding: 2px 10px;">128</td><td style="padding: 2px 10px;">133</td></tr> <tr><td style="padding: 2px 10px;">24</td><td style="padding: 2px 10px;">26</td><td style="padding: 2px 10px;">178</td><td style="padding: 2px 10px;">200</td></tr> <tr><td style="padding: 2px 10px;">2</td><td style="padding: 2px 10px;">0</td><td style="padding: 2px 10px;">255</td><td style="padding: 2px 10px;">220</td></tr> </table> | 56 | 32 | 10 | 18 | 90 | 23 | 128 | 133 | 24 | 26 | 178 | 200 | 2 | 0 | 255 | 220 | | <table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 10px;">10</td><td style="padding: 2px 10px;">20</td><td style="padding: 2px 10px;">24</td><td style="padding: 2px 10px;">17</td></tr> <tr><td style="padding: 2px 10px;">8</td><td style="padding: 2px 10px;">10</td><td style="padding: 2px 10px;">89</td><td style="padding: 2px 10px;">100</td></tr> <tr><td style="padding: 2px 10px;">12</td><td style="padding: 2px 10px;">16</td><td style="padding: 2px 10px;">178</td><td style="padding: 2px 10px;">170</td></tr> <tr><td style="padding: 2px 10px;">4</td><td style="padding: 2px 10px;">32</td><td style="padding: 2px 10px;">233</td><td style="padding: 2px 10px;">112</td></tr> </table> | 10 | 20 | 24 | 17 | 8 | 10 | 89 | 100 | 12 | 16 | 178 | 170 | 4 | 32 | 233 | 112 | | <table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 10px;">46</td><td style="padding: 2px 10px;">12</td><td style="padding: 2px 10px;">14</td><td style="padding: 2px 10px;">1</td></tr> <tr><td style="padding: 2px 10px;">82</td><td style="padding: 2px 10px;">13</td><td style="padding: 2px 10px;">39</td><td style="padding: 2px 10px;">33</td></tr> <tr><td style="padding: 2px 10px;">12</td><td style="padding: 2px 10px;">10</td><td style="padding: 2px 10px;">0</td><td style="padding: 2px 10px;">30</td></tr> <tr><td style="padding: 2px 10px;">2</td><td style="padding: 2px 10px;">32</td><td style="padding: 2px 10px;">22</td><td style="padding: 2px 10px;">108</td></tr> </table> | 46 | 12 | 14 | 1 | 82 | 13 | 39 | 33 | 12 | 10 | 0 | 30 | 2 | 32 | 22 | 108 | |
| 56 | 32 | 10 | 18 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 90 | 23 | 128 | 133 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | 26 | 178 | 200 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0 | 255 | 220 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 20 | 24 | 17 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 10 | 89 | 100 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 16 | 178 | 170 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 32 | 233 | 112 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 46 | 12 | 14 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 82 | 13 | 39 | 33 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 10 | 0 | 30 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 32 | 22 | 108 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 2.10: Illustration of comparing two images pixel by pixel (F. Li, 2020b)

2.3.2 Convolutional Neural Networks (CNN)

Before discussing CNN, we need to talk about the artificial neural network (ANN). ANN is a model that mimics our human neuron system. Perceptron is the fundamental unit inside a neural network. It consists of five components;

inputs (There will be one input as bias), weights, sum, non-linearity, and output. Figure 2.11 (right), shows an example of three layers ANN. Every circle is a perceptron and because all the perceptrons are connected to each other, we called it a fully connected neural network.

CNN is a type of ANN, where commonly used in computer vision. The presence of CNN is to fill up ANN weakness when came to image processing. The working principle of CNN and ANN is similar, but ANN did not scale well on large images. Nowadays, a full HD image has the size of 1920×1080 . If we want to build a neural network for it, the first layer of the neural network will have $1920 \times 1080 \times 3 > 6$ million (assume RGB, width*height*color component) weights to consider. That is a lot of computation if we use ANN, and 6 million only consists of one layer. If we need to build a multi-layer neural network, the computation will be impractical.

There are three major operations in CNN. Convolution, Pooling and rectified linear activation function (ReLU). Illustrated in Figure 2.12.

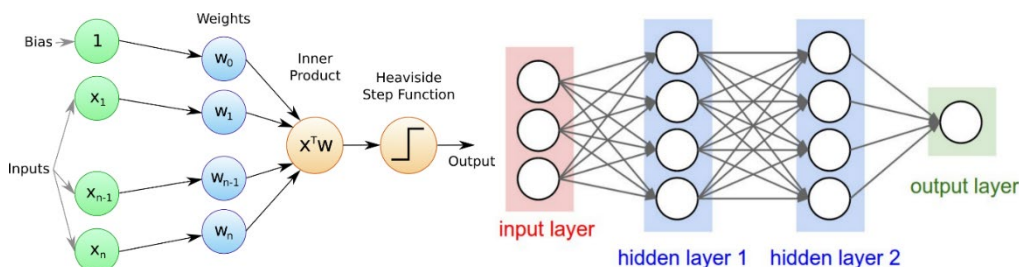


Figure 2.11: Left: Perceptron Source: QUANTSTART: Introduction to Artificial Neural Networks and the Perceptron. Right: three layers ANN (F. Li, 2020b)

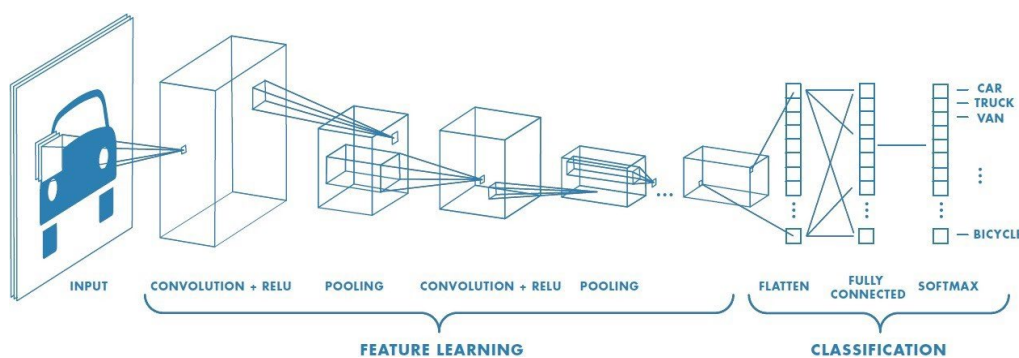


Figure 2.12: Structure of CNN. (Saha, 2018)

2.3.2.1 Convolution layer

Convolution is the most important layer used to extract features. In this layer, we will have a set of filters. These filters usually came in small sizes. Take ConvNet as an example; a typical filter for the first layer has 5x5x3 (height x width x RGB color). The filter will then scan through the entire image and produce a feature map. After all, filters convolve with the image, we will get a stack of feature maps. This stack of feature maps will then become a higher level of filter, and the process continues.

For example, a hand consists of five fingers. Therefore, we have five filters, one filter for the thumb, one filter for the index finger, one filter for the middle finger, and so on. After the first filter convolves, we got a feature map for the thumb; 2nd filter gave us a feature map for the index finger, etc. All the five features map then combine and form a filter call hands. This hierarchy continues, and the convolution process continues for the higher-level feature. The finger feature might come from distal phalanx, middle phalanx, and proximal phalanx. If we go even lower level, all these features might come from curves and lines. Of course, this is just an analogy example for better understanding. Inside a machine, all these represent by arrays of numbers

2.3.2.2 Pooling layer

The main purpose of pooling is to reduce the dimension so that the computation power is reduced. Pooling layers often place between the convolution layer and the next convolution layer. The two common types of pooling are max pooling and average pooling. Take the illustration above as an example, 2 by 2 max-pooling means that in 2 by 2 array, take the maximum number. Average pooling will be taking the average. Stride of two means slides two arrays after a pooling is done. In the example shown in Figure 2.13, 16 arrays shrink to 4 arrays after the pooling layer. Other than reducing the dimension, this operation will also reduce the chance of overfitting. Therefore, pooling layers is critical to make CNN practical to compute.

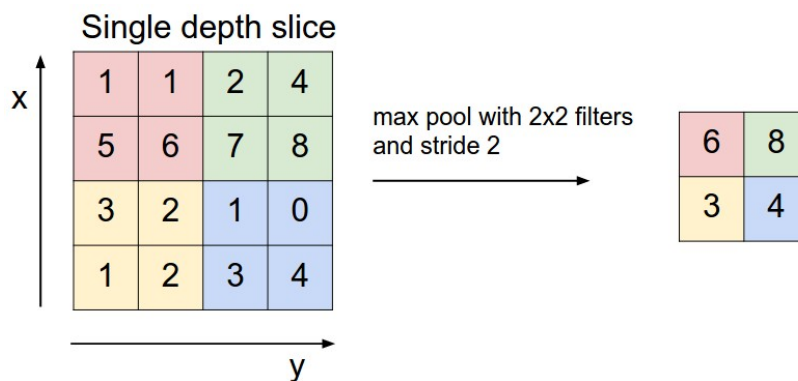


Figure 2.13: 2x2 max pooling with stride of 2. (F. Li, 2020b)

2.3.2.3 ReLU layer

In ReLU layer, an elementwise activation function will be applied such as $\max(0, x)$. Basically, what this function does is only keep the positive value; other values will be set to zeros. The main purpose of ReLU layer is to add non-linearity to the model.

2.4 CNN architectures

Nowadays, most of the advanced computer vision algorithms design around CNN architectures. CNN architectures have been proven that is effective in many applications. In this project, the object detection algorithm used will also be based on CNN. Therefore, we need to learn different algorithms to compare and search for the best suits for our project.

2.4.1 You Only Look Once algorithm (YOLO)

YOLO is a real time object detection algorithm. YOLO algorithm can process real-time images at 45 frames per second (Redmon *et al.*, 2016). This algorithm performs so quickly thanks to its single regression problem architecture. YOLO has 24 convolutional layers followed by two fully connected layers. It has alternating 1x1 convolutional layers to remove feature spaces from previous layers. In short, YOLO works bounding box detection and class probability in one shot.

YOLO algorithms work as such. First, the image will be divided into $S \times S$ grids. Each grid cell will predict B bounding boxes. Each bounding box has five predicted parameters x, y, w, h , and confidence. Where (x, y) represents

the coordinate of the center of the bounding box relative to grid cells, (w,h) represents the width and height relative to the whole image. Each grid cell can use to predict C number of classes. But only one class can be existing in each grid cell. Since each grid will have B bounding boxes, the confidence level will calculate using intersection over union (IOU) between bounding boxes. To know what class is inside the bounding box, YOLO use a probability equation as follow:

$$P_r(Class_i|Object) * P_r(Object) * IOU_{pred}^{truth} = P_r(Class_i) * IOU_{pred}^{truth} \quad (2.2)$$

Figure 2.14 shows a YOLO evaluation example on PASCAL VOC S=7, B=2, C=20. The drawback of YOLO is very obvious. It highly depends on the spatial properties of the image. Because every grid cells only allow having one class, it struggles when came to small object detection. Another drawback will be the difficulty of recognizing objects that are deformed or having an unusual aspect ratio.

In YOLOv2 (Redmon and Farhadi, 2017), the fully connected layers in YOLO was replaced with anchor boxes. Anchor boxes solved YOLO one class per grid cells problem. By using anchor box, now classification happen between grid cell and anchor box. Anchor box work as such, for example YOLO will has output of

$$y = \begin{bmatrix} P \\ x \\ y \\ w \\ h \\ c_1 \\ c_2 \end{bmatrix} \quad (2.3)$$

where P is probability, (x,y,w,h) is bounding parameters, (c₁, c₂) is Boolean value whether class being detected. When two object is detected, YOLOv2 output will be

$$y = \begin{bmatrix} P \\ x \\ y \\ w \\ h \\ c_1 \\ c_2 \\ P \\ x \\ y \\ w \\ h \\ c_1 \\ c_2 \end{bmatrix} \quad (2.4)$$

this is for case with two anchor boxes. If n object is detected in one single grid cell, n anchor boxes will be needed and the matrix will go on.

Besides major anchor box improvement in YOLOv2, YOLOv2 and YOLOv3 also improved a lot in terms of precision and detection speed. YOLOv3 introduce a new Darknet-53 backbone which has 53 convolutional layers compare to DarkNet-19 use in YOLOv2. Figure 2.15 shows Comparison of AP in various algorithm.

Hand gesture recognition using YOLO applied in (Chhaged *et al.*, 2021), where hand gesture recognition allows specially-abled individuals to use video calls. As we know COVID-19 pandemic makes video calls become more common, the quick recognition and low latency characteristic of YOLO become a good candidate for this application. Even the latest version of YOLO, YOLOv4 is selected for this application and all the messages tested can be detected successfully (Chhaged *et al.*, 2021).

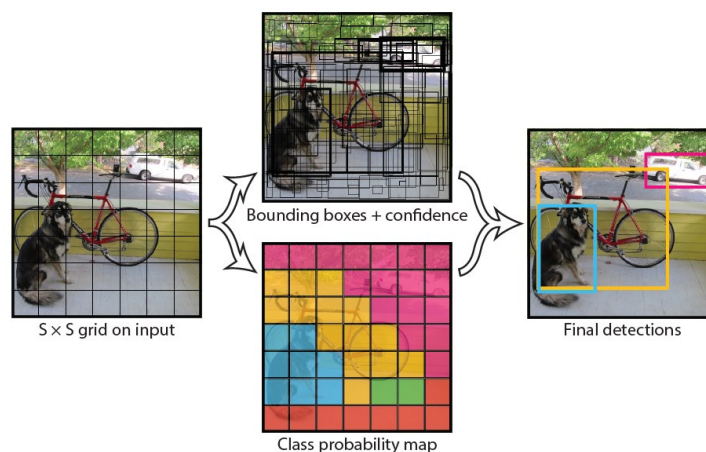


Figure 2.14: YOLO evaluation on PASCAL VOC using $S=7$, $B=2$, $C=20$.
(PASCAL VOC has 20 labelled classes) (Redmon *et al.*, 2016).

| | backbone | AP | AP ₅₀ | AP ₇₅ | AP _S | AP _M | AP _L |
|---------------------------|--------------------------|-------------|------------------|------------------|-----------------|-----------------|-----------------|
| <i>Two-stage methods</i> | | | | | | | |
| Faster R-CNN+++ [5] | ResNet-101-C4 | 34.9 | 55.7 | 37.4 | 15.6 | 38.7 | 50.9 |
| Faster R-CNN w FPN [8] | ResNet-101-FPN | 36.2 | 59.1 | 39.0 | 18.2 | 39.0 | 48.2 |
| Faster R-CNN by G-RMI [6] | Inception-ResNet-v2 [21] | 34.7 | 55.5 | 36.7 | 13.5 | 38.1 | 52.0 |
| Faster R-CNN w TDM [20] | Inception-ResNet-v2-TDM | 36.8 | 57.7 | 39.2 | 16.2 | 39.8 | 52.1 |
| <i>One-stage methods</i> | | | | | | | |
| YOLOv2 [15] | DarkNet-19 [15] | 21.6 | 44.0 | 19.2 | 5.0 | 22.4 | 35.5 |
| SSD513 [11, 3] | ResNet-101-SSD | 31.2 | 50.4 | 33.3 | 10.2 | 34.5 | 49.8 |
| DSSD513 [3] | ResNet-101-DSSD | 33.2 | 53.3 | 35.2 | 13.0 | 35.4 | 51.1 |
| RetinaNet [9] | ResNet-101-FPN | 39.1 | 59.1 | 42.3 | 21.8 | 42.7 | 50.2 |
| RetinaNet [9] | ResNeXt-101-FPN | 40.8 | 61.1 | 44.1 | 24.1 | 44.2 | 51.2 |
| YOLOv3 608 × 608 | Darknet-53 | 33.0 | 57.9 | 34.4 | 18.3 | 35.4 | 41.9 |

Figure 2.15: Comparison of AP in various algorithm. (Redmon and Farhadi, 2018), (Lin *et al.*, 2020)

2.4.2 Single Shot MultiBox Detector (SSD)

Single Shot MultiBox Detector (SSD) is another type of CNN algorithm that can achieve fast detection. SSD is the first CNN network that does not have a resampling stage for pixels or features for the bounding box (Liu *et al.*, 2016). SSD architecture is based on feed-forward CNN which only produce fixed size bounding box. To increase the robustness of the algorithm, SSD use “multibox” approach where SSD will apply anchor box with different aspect ratio in every pixel and detect the object.

Although SSD has very comparable performance in terms of speed and accuracy, compare with YOLO. But their architecture is different; SSD uses VGG-16 as the backbone and has several convolution stages with

different sizes compared with YOLO only one convolution layer. Figure 2.16 shows comparison of SSD architecture and YOLO architecture.

Both (Tang *et al.*, 2019) and (Liu *et al.*, 2019) show the feasibility of using SSD architecture in hand gesture recognition. (Liu *et al.*, 2019) use SSD to capture hand gesture that represent alphabet, and able to achieve minimum accuracy more than 93.8%. On the other hand, (Tang *et al.*, 2019) use SSD targeting on non-contact HMI where similar to our project. Their result also able to achieve accuracy up to 95.05%.

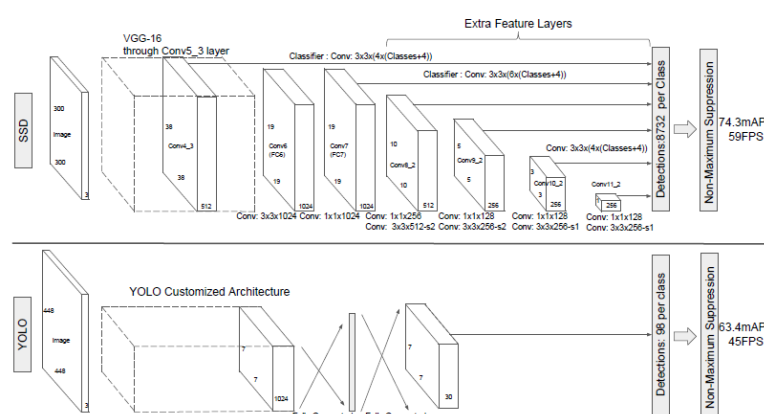


Figure 2.16: Comparison of SSD architecture and YOLO architecture. (Liu *et al.*, 2016).

2.4.3 MediaPipe Hands

MediaPipe is a framework for object detection compatible with various sensors and various object detection algorithms (Lugaresi *et al.*, 2019). Figure 2.17 shows MediaPipe framework architecture. The main purpose of MediaPipe is to allow researchers to rapid prototyping their models with other reusable components. MediaPipe Hands is one of the models that utilize MediaPipe framework that focus on detecting hand gesture.

MediaPipe Hands consist of two models, a palm detector and a hand landmark model (hand skeleton model) (Zhang *et al.*, 2020). Palm detector is used to detect hands with bounding box. This model is based on BlazeFace (Bazarevsky *et al.*, 2019), a SSD model optimized for inference on mobile GPUs. Since palm and fist are easier to detect compare to finger joint. Therefore, palm will be detected first, hand landmark model will be used to

further detect afterward. Palm detector consists of two key processes, a feature extractor similar to Feature Pyramid Networks (Lin *et al.*, 2016) that allow recognition of palm on a different scale. Next, it will minimize the focus during training to support different anchors amount and high scale variance.

Hand landmark model is a skeleton model that output 2.5D coordinates. This model has three outputs:

- (i) Relative depth and (x, y) coordinates of 21 hand landmarks show in Figure 2.18.
- (ii) Flag indicate the present of hand in image.
- (iii) Boolean variable indicates left or right hand.

Hand landmark model uses the same topology to Hand Keypoint Detection in Single Images using Multiview Bootstrapping (Simon *et al.*, 2017). Hand Keypoint Detection uses multiple cameras to extract 3D keypoints information and the model is based on Convolutional Pose Machines architecture (CPMs) (Wei *et al.*, 2016). Hand Keypoint Detection is 3D and using multiple cameras, while the Hand landmark model uses a single camera and 2.5D coordinates. The concept of relative depth shows Figure 2.19.

Because MediaPipe Hands using MediaPipe framework (process shown in Figure 2.17). MediaPipe framework came with an extensible set that solved many pre and post-processing tasks, such as model inference and data transformations.

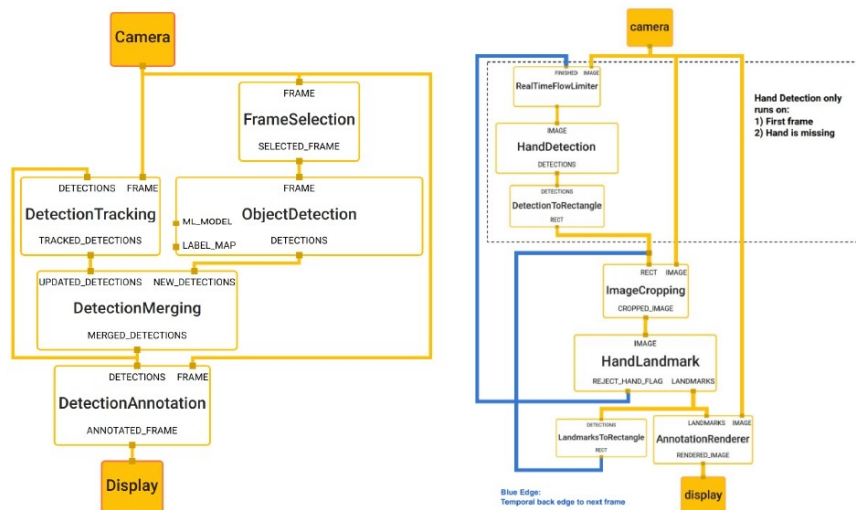


Figure 2.17: Left: MediaPipe framework. (Lugaresi *et al.*, 2019). Right: MediaPipe Hands framework. (Zhang *et al.*, 2020)

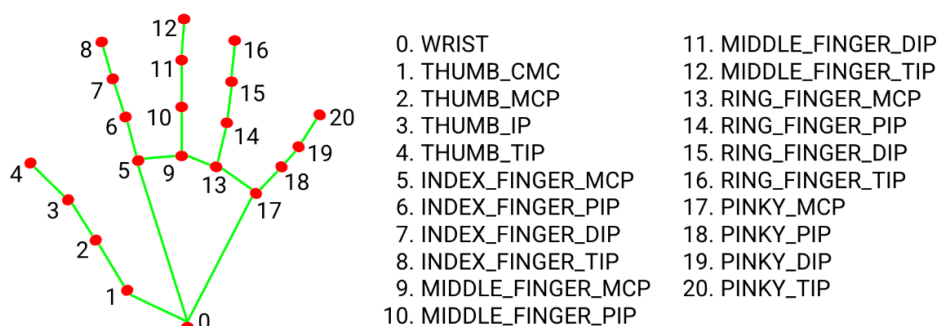


Figure 2.18: Hand landmark model 21 landmarks. (Inc., 2020)

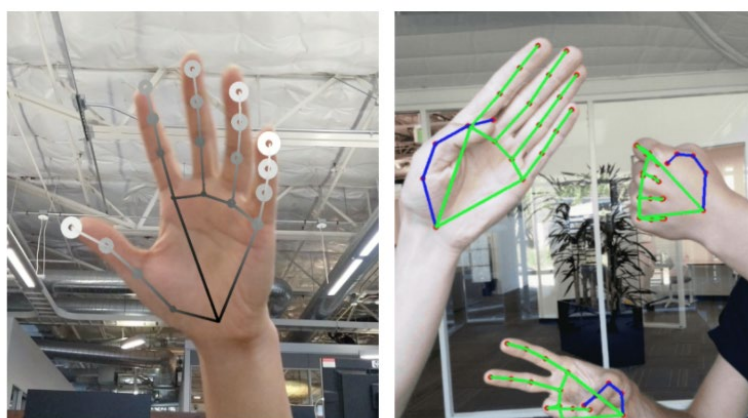


Figure 2.19: (Left): Hand landmarks with relative depth presented in different shades. The lighter and larger the circle, the closer the landmark is towards the camera. (Right): Real-time multi-hand tracking.

Source: (Zhang *et al.*, 2020)

2.5 Architectures comparison

The comparison chart (Figure 2.20), shows that all three architectures provide real-time detection, and all have decent accuracy and only require a normal webcam. The focus of our project will be pursuing higher accuracy, higher resolution and lowest latency as possible. MediaPipe Hands is a relatively new algorithm, and not research paper publish any application about it yet. Its 21-landmark gives us high flexibility on hand gesture detection, which is suitable in our project.

COMPARISON CHART

| CNN ARCHITECTURES | YOLOv3 | SSD | MediaPipe Hands |
|--|----------------------------|---|-----------------------------|
| Detection Speed | up to 45 Frames per second | up to 59 Frames per second | up to 30 Frames per second |
| Detection Accuracy | 93% | 95.05% | 95.7% in Palm Detection |
| Detection Output | Bounding Box | Bounding Box | Bounding Box & 21 landmarks |
| Equipment Needed | Webcam | Webcam | Webcam |
| Research Paper that use this Architectures | (Chhaged et al., 2021) | (Tang et al., 2019) & (P. Liu et al., 2019) | - |

Figure 2.20: CNN architectures comparison chart

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Project overview

The overview of this process consists of three parts, illustrate in Figure 3.1. Input from the camera sensor, use python to do hand gesture recognition, process hand gesture images into useful data, convert it to the command we desire, and output to devices we wish to control.

This project will use only a single camera to capture all the visual information. Next, the OpenCV library will extract information from the sensor, create a frame that shows real-time captured data, provide a user interface, etc. Then, MediaPipe Hands will be used as hand detection algorithm. After that, hand gesture data need to be processed into meaningful commands. Our goals are to use hand gestures to control or achieve some action. Therefore, extracting hand gesture patterns is essential. We designed some functions to help us detect hand gesture patterns and two applications to show the capability of hand gesture control. The last step will be sending commands to devices and demonstrating our hand gesture applications.

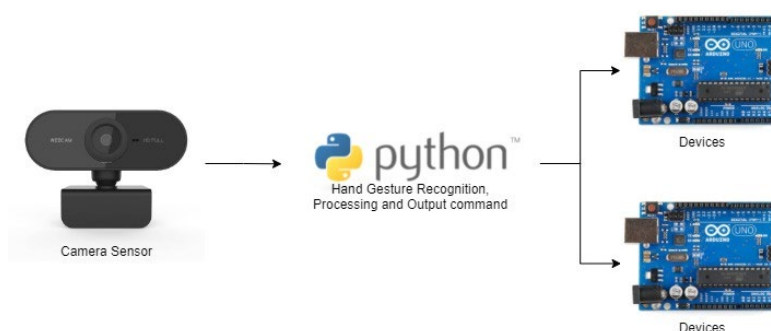


Figure 3.1: Project overview

3.2 Hardware Overview

The hardware used in this project demonstrates how to control a device using gesture and shows the capability and precision gesture control can achieve. The device we built in this project to demonstrate is a WiFi Robot Car. The WiFi Robot Car has components of ESP32 DEV KIT DOIT, L298N DC

Motor Driver, 2 DC motors, a power supply, and a car chassis to hold up all the pieces. The list of cost breakdowns for all components used is shown in Table 3.1. The WiFi Robot Car setup is shown in Figure 3.2.

Subsequence paragraphs should be indented 1.5 cm (0.59 inches) from the left margin. General alignment for texts in a paragraph should be “justified”.

Table 3.1: Detailed Cost Breakdown for each component.

| Hardware Component | Cost |
|-------------------------|--------------|
| ESP32 DEV KIT DOIT | RM 32.00 |
| L298N DC Motor Driver | RM 11.00 |
| DC motors | RM 3.50 each |
| Car Chassis with Wheels | RM 18.00 |
| Battery Holder | RM 2.50 |
| Breadboard | RM 4.00 |

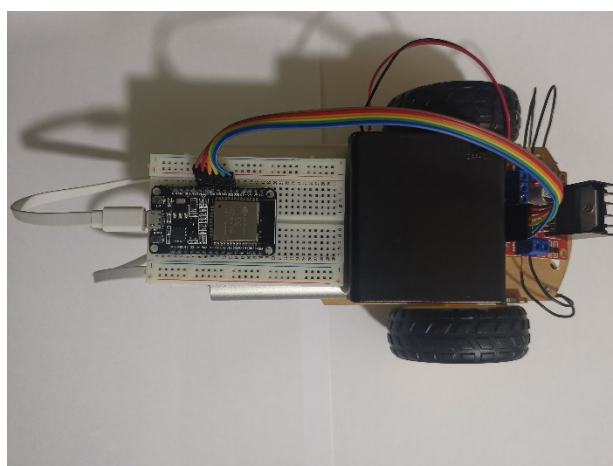


Figure 3.2: WiFi Robot Car Set up

3.2.1 ESP32 DEV KIT DOIT

Most of the WiFi Robot Car projects can be found on the Internet using Arduino UNO + ESP8266 combo to control L298N DC Motor Driver wirelessly. When choosing a microcontroller, there are two main considerations in this project. The First will be whether the component support WiFi, second will be whether the microcontroller has PWM output channels. ESP32 is the ESP8266 successor. Compared to its predecessor, ESP32 adds an

extra CPU core, has more GPIOs, uses HT40 mode in WiFi compared to ESP8266 HT20 mode, and supports Bluetooth 4.2. ESP32 can be found in standalone chips and many types of the development board. The development board we choose for this project is ESP32 DEV KIT DOIT. Features that came with this board are onboard RESET (EN) and BOOT buttons. It also comes with a USB-to-UART interface which allows us to directly program ESP32 using Arduino IDE without any external converter such as “UC00A (FTDI) USB to UART Converter”. This board can also be powered using a micro-USB connector with a power bank so that we do not need to supply power via VIN or 3.3V pins.

These features make the setup easy, and there is no need to worry about the power source. Built-in USB-to-UART also makes programming ESP32 quick and easy. Suitable for prototyping where configuring Arduino coding and program ESP32 are very frequent. Figure 3.3 shows the complete pinout of ESP32 DEV KIT DOIT. The thing that needs to be noticed is that the pin connecting to L298N DC Motor Driver needs to be PWM supported. Not all pin in ESP 32 DEV KIT DOIT is PWM supported.

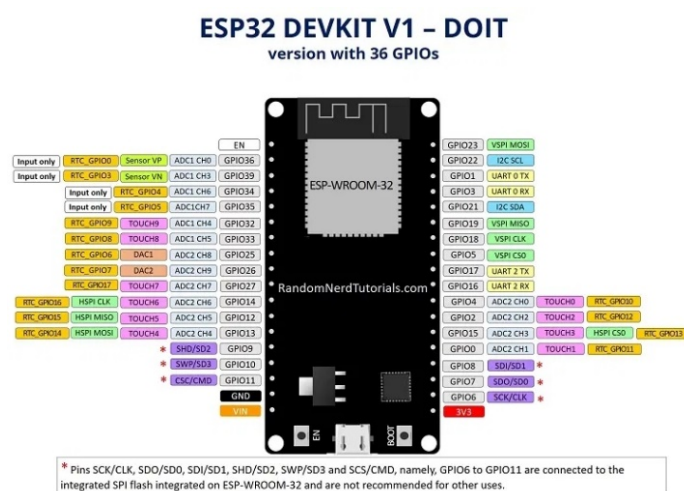


Figure 3.3: Complete pinout of ESP32 DEV KIT DOIT (Santos, 2018)

3.2.2 L298N DC Motor Driver

L298N is one of the very common modules used when controlling a DC motor. There are two parameters when controlling a DC motor: speed and direction. We can control these two parameters with the following techniques.

- (i) Pulse-width modulation (PWM) – To control the rotation speed of the motor.
- (ii) H-Bridge – To control the direction of motor rotation.

PWM is a technique that sends a series of ON-OFF pulses in a period of time. Duty Cycle tells us the proportion of “ON” in the period of time. The speed of the DC motor can be controlled by the voltage supplied. Using the PWM technique, different duty cycle results in different average voltage and, therefore, can control motor speed. Figure 3.4 clearly illustrates how the duty cycle can achieve different average voltages.

H-Bridge is a type of circuit that can switch the polarity of a voltage applied to a load. The working principle of the H-Bridge is shown in Figure 3.5. In L298N Module, there are four output and seven input pins, shown in Figure 3.6. Having four output pins allows us to control 2 DC motors with 1 module (2 output pins for one motor). L298N module itself requires 5V to power. There is a jumper above inputs 1 and 2. If the jumper is in place, power is supplied to VCC and the ground and supplied to both the module and 2 DC motors. This is convenient because we only need one power source for the L298N module and DC motors. However, the downside is the voltage received by the DC motor will be capped at 5V. The jumper can be removed if we want to have a larger range of voltage control on DC motor (L298N support up to 12V). If the jumper is removed, VCC will power the DC motor while 5V input is used to power the L298N module, but if so, we will need two different power sources to power them.

Next, ENA and ENB are used to control the speed of motors. Input pins 1,2, and 3,4 are used to control the direction of motors. The signal feed to these pins will be decided by ESP32 and is programmable using Arduino IDE.

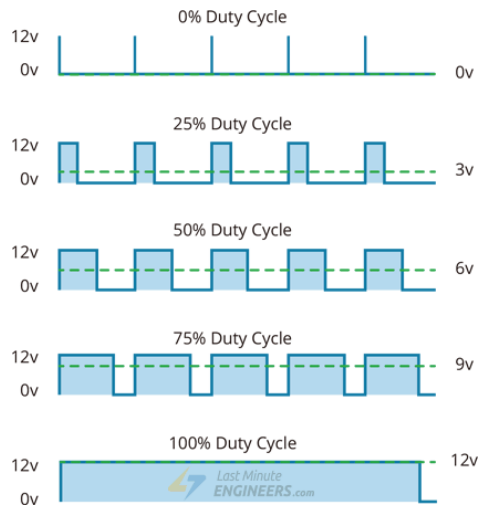


Figure 3.4: Duty Cycle in PWM (Last Minute Engineers, 2021).

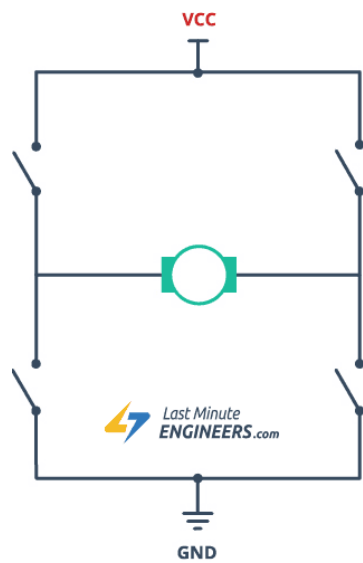


Figure 3.5: H-Bridge working principle (Last Minute Engineers, 2021).

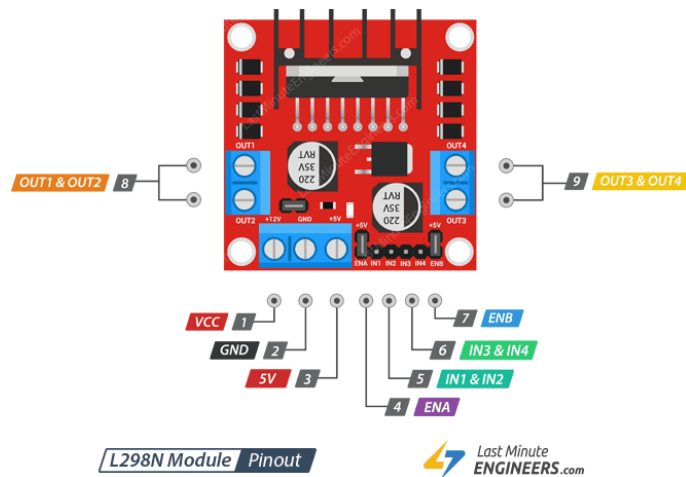


Figure 3.6: L298N DC Motor Driver Pinout (Last Minute Engineers, 2021).

3.3 Software Overview

The software used in this project can mainly be separated into two categories. Libraries used in the python environment are mainly used to extract and process hand gesture data. Libraries used in Arduino IDE are mainly used to control ESP32 and control the movement of WiFi Robot Car.

3.3.1 Arduino IDE

Arduino IDE is open-source software designed by Arduino.cc. In our project, Arduino IDE is mainly used to write, compile and upload the code to our microcontroller ESP32. Arduino IDE not only supports ESP32 it also supports many Arduino modules and microcontrollers.

3.3.2 MQTT

Message Queuing Telemetry Transport (MQTT) is a lightweight publish/subscribe messaging protocol designed for M2M (machine to machine) telemetry in low bandwidth environments. MQTT is one of the common communication protocols used in wireless or IOT applications. Other communication protocols such as LoRaWAN and Zigbee are also widely used in IoT applications. But we choose MQTT in our project mainly due to its quick setup and lightweight characteristic. MQTT comprises three key elements, subscriber, publisher, and broker. Quick setup is because subscribing to a topic or publishing a topic is just a few lines of code, and many tutorials can be easily found online. Setting up a broker is also not essential for prototyping. Many online MQTT brokers are free to use, and no installation is required; just configured on the devices correctly will do. MQTT is not only suitable for prototyping, it supports various authentications and data security mechanisms, and it's also easily scalable to control more devices.

3.3.3 OpenCV

OpenSource Computer Vision Library, also known as OpenCV, is an open-source library that includes several hundreds of computer vision algorithms. In this project, OpenCV is being used in real-time image processing and image capture. For example, basic functions such as "VideoCapture" and "read" allow us to capture video from a webcam and further process it in python. It

also integrated well with libraries such as NumPy, which is very useful in this project because many data collected from the MediaPipe Hand model need to be processed before using it in our application. There are other functions, such as drawing different geometric shapes, which is convenient for us to create a simple UI for users to interact with.

3.3.4 MediaPipe Hand

MediaPipe Hand will be the main model used in this project to detect and extract hand information. The details concept of MediaPipe Hand has been discussed in Section 2. In this project, the MediaPipe Hand model only extracts raw coordinate information. Most of the challenges in this project will be how to process those data and convert them into something we need. With data process and creativity added in, this model can be used to create many applications.

3.4 Methodology

3.4.1 Image Pre-processing

We usually need to go through data pre-processing and cleaning steps in machine learning and computer vision projects. These steps usually take a good amount of time and are crucial for later, such as building the learning model (Mohamed Elgendy, 2020). Before training the model, some common pre-processing techniques convert color images to grayscale to reduce computation complexity. Another method commonly used is data augmentation. Augmenting existing datasets such as scaling and rotation allows the neural network we train to expose more data variations. Since we are using a trained model in this project, our pre-processing steps will be different. Our focus will be more on what the pre-process is required to fit the model best. For example, if we want to use the trained model MobileNet to do some object recognition. In order to get the best accurate result, we can call TensorFlow Keras MobileNet pre-process layer to help us do the pre-processing. Basically, what the layers do is some image resizing and pixels normalize scaling. The same thing needs to be done in this project for the best accurate outcome from MediaPipe Hands.

We need to do two main things: the first will be conversion of color model, and the second will be flipping around the y-axis (mirror flip). Most of the OpenCV functions operate in BGR order. Therefore, without conversion, we have no problem using functions such as “imread”, “imwrite” and “imshow”, imshow will show us RBG color as usual without any problem. But MediaPipe Hands mainly process data in RGB color. Therefore, BGR to RGB conversion will be required for the best result. Without flipping, the left hand will appear on the right side on the screen, and vice versa. The code needed for the pre-processing is as below:

```
# Mirror flip webcam Img
img = cv2.flip(img,1)
# Converting BGR to RGB
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

3.4.2 Call Model and Extract Data

After processed the video data. The next step will be to feed into the model and extract the data we need. Usually, when we want to use a function from a library is rather simple. For example, calling “pandas.read_csv()” to read csv file using pandas or using “cv2.VideoCapture()” to capture video information from webcam using OpenCV. But it is different in our project. Due to MediaPipe architecture, calling the model to detect hand can be tricky, and we need to go through a few layers before getting the information we need. In this project, many functions were created. Therefore, to make it easier to manage and organize, we make all the functions object-oriented. We created a class, and fit most of the functions became methods to be more organized and easier to call later in the project.

Figure 3.7 shows the layers of the MediaPipe solution. There are two data that we need to extract from this model. First will be the x and y coordinates of the 21 landmarks; second will be hand label telling us is it left hand or right hand. To get those pieces of information, we need to call the model. MediaPipe is an extensive library that consists of many solutions. Two of the solutions we used in this project will be hands to detect the 21 landmarks and drawing utilities to help us draw and show the landmarks. Figure 3.8 shows the layers to call the model and its configurations. Table 3.2 has a details description of each configuration option.

Table 3.2: Details Description for model configuration.

| Configuration Options | Description |
|-----------------------------|---|
| mode (Static_Image_Mode) | If set to FALSE, it will treat input as a video stream, which we are using in this project. If set to TRUE, hand detection will run on every input image and might result in high latency. |
| max_num_hands | Maximum number of hands to detect. Default set to 2. |
| model complexity | Either 0 or 1. Default set to 1, which provides higher accuracy. Can be set to 0 if having high inference latency. |
| min_detection_confidence | Minimum confidence value for the model to be considered hand successfully detected. Default set to 0.5, range from 0 to 1. |
| min_tracking_confidence | Minimum confidence value for the landmark-tracking model to be considered tracked successfully. If failed to track, it will automatically look for the next input image. Default set to 0.5, range from 0 to 1. This parameter is ignored if the mode is set to TRUE (Static_Image_Mode). |

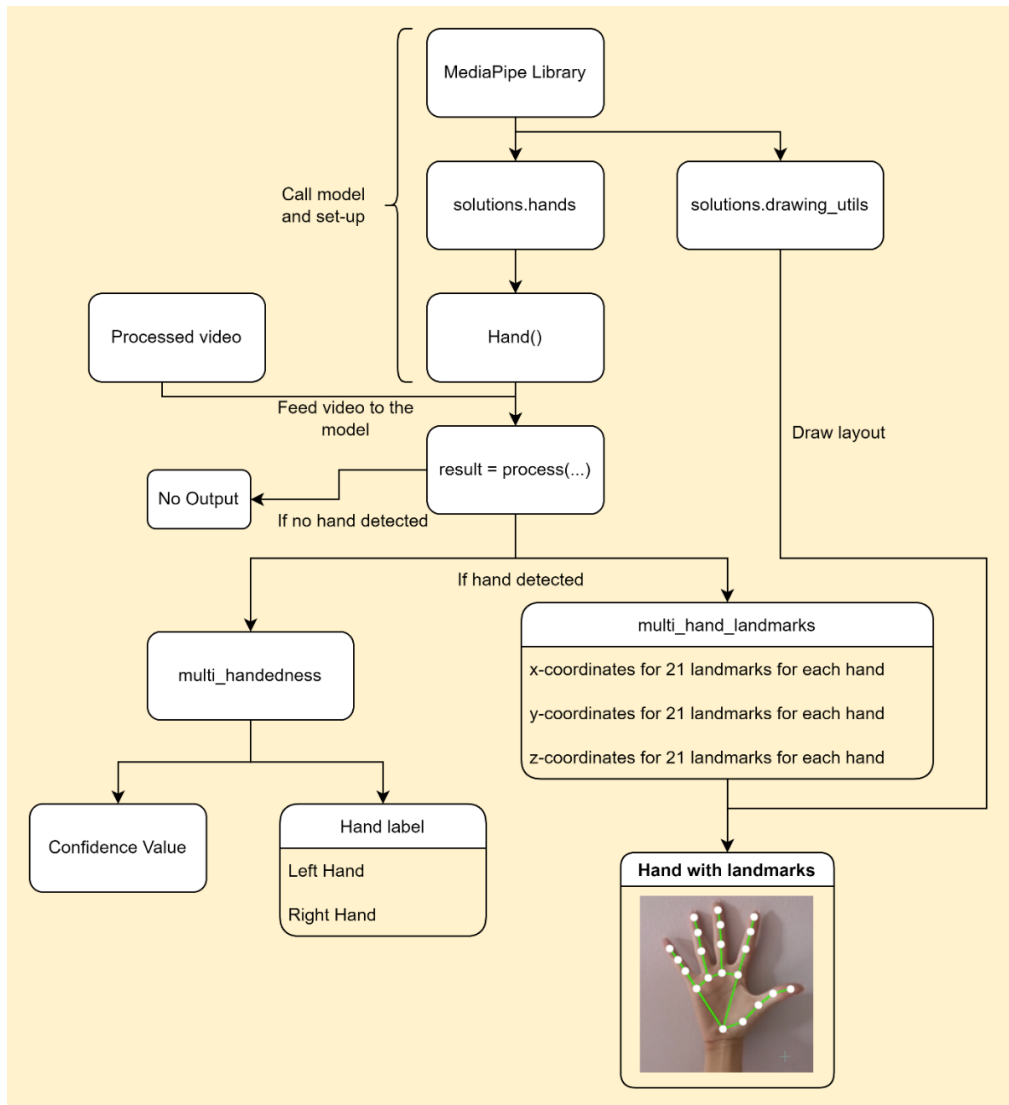


Figure 3.7: Layers of MediaPipe Hand solution.

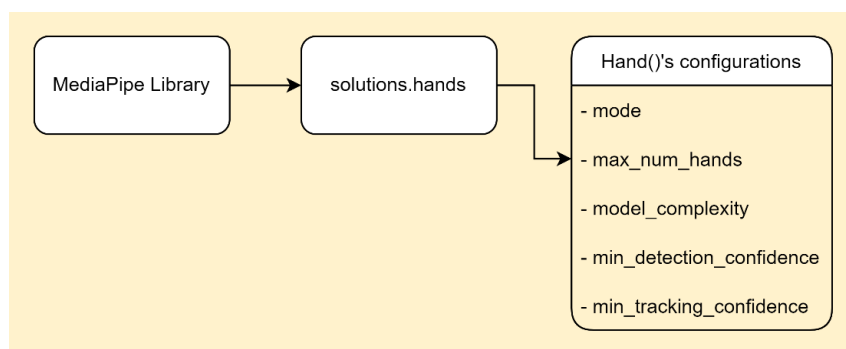


Figure 3.8: Model Configurations.

3.4.3 Output Data Processing and Organizing

Next step after we obtain the data will be to process and organize it. Figure 3.10 shows that the model's output data is not very optimized for our usage. For example, the 21 landmarks are stored in a 2 layers list, and x, y, and z coordinates are stored in the form of dictionary. Meaning that if we need want to know the x, y coordinate of landmark one, we will have to call “multi_hand_landmarks[0][1].x” to get the x-coordinate of landmark one the first hand. Not only that, we have no guarantee that hand one is left hand or right hand. This will change based on the detection situation. Other than that, the x, y, and z coordinates that output from the model is normalized coordinate. We need to scale it back based on the resolution of the video we capture. Therefore, our aim in this stage will be,

- (i) Link 21 landmarks with hand type labels to differentiate the landmarks from which hand.
- (ii) Scale back the normalized coordinate based on the video resolutions.
- (iii) Create the custom 22nd landmark.
- (iv) Create a custom parameter called “hand angle”.
- (v) Combine all the information. The output processed data will be a dictionary that consists of 3 keys, hand type, hand angle, and 22 landmarks.

The convenience of organizing in this structure is apparent when doing application design.

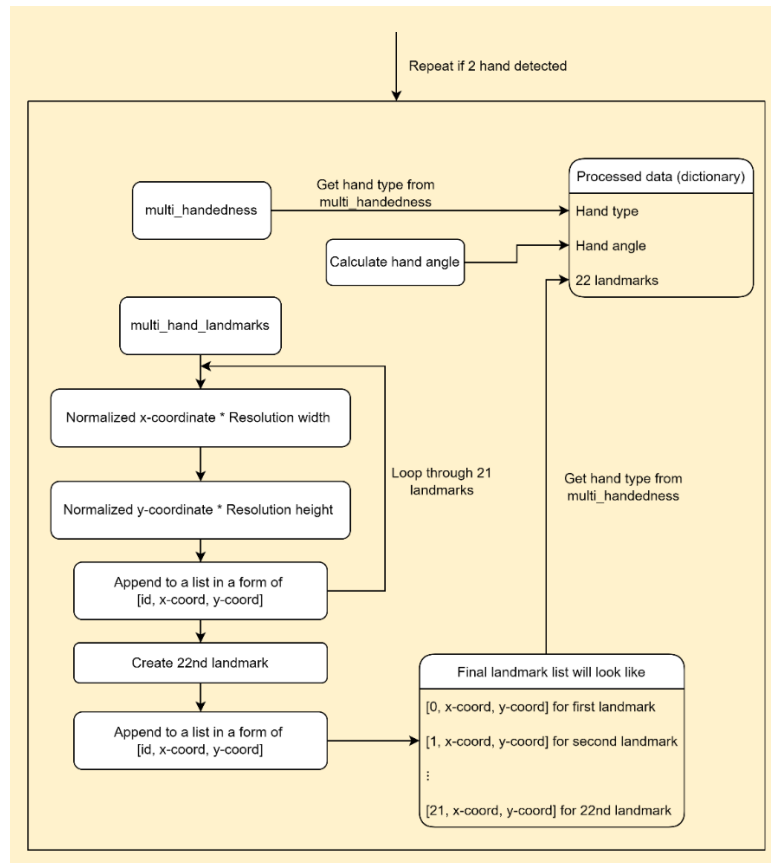


Figure 3.9: Steps for process and organize data.

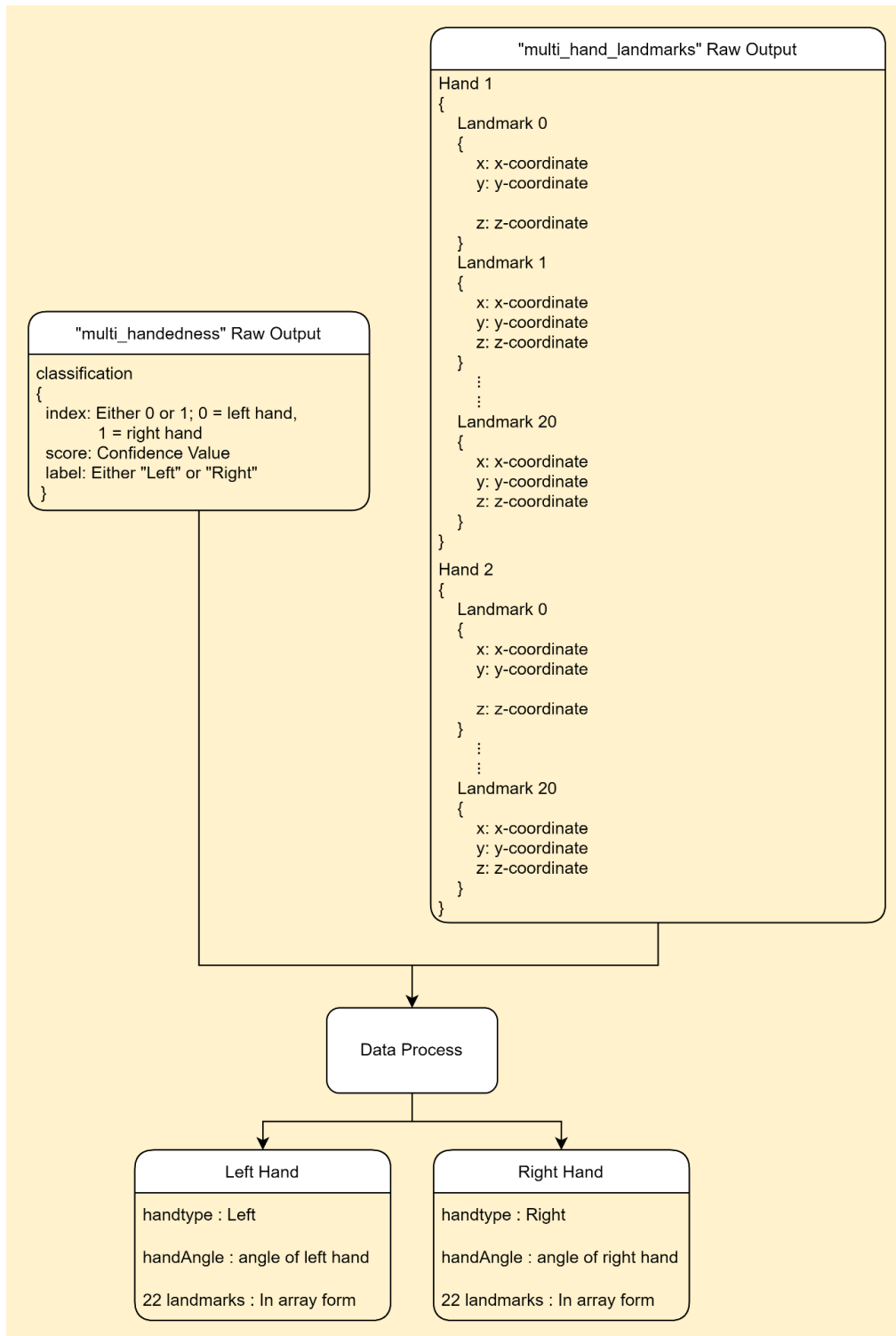


Figure 3.10: Data Structure of Output raw data and Data Structure after Data Processing.

3.4.4 Hand angle

When designing a gesture or detecting a gesture, it is crucial to know the hand's orientation. Whether the user's hand is facing upward, downward, left,

or right. This is important because, for the same gesture, it is a totally different set of coordination when facing a different direction. To calculate the hand angle, we can use the dot product angle formula to calculate the angle between two vectors. Imagine we have 3 points ABC as in Figure 3.11, to find the angle $\angle ABC$, we can use the formula as follow:

$$\vec{BA} = B - A \quad (3.1)$$

$$\vec{BC} = B - C \quad (3.2)$$

$$\vec{BC} \cdot \vec{BA} = |\vec{BC}| \cdot |\vec{BA}| \cdot \cos\theta \quad (3.3)$$

$$\cos\theta = \frac{(\vec{BC} \cdot \vec{BA})}{(|\vec{BC}| \cdot |\vec{BA}|)} \quad (3.4)$$

$$\theta = \cos^{-1}\left(\frac{(\vec{BC} \cdot \vec{BA})}{(|\vec{BC}| \cdot |\vec{BA}|)}\right) \quad (3.5)$$

After knowing the equation, the equation can be easily implemented using NumPy. NumPy has all the functions we need to complete the formula, from creating vectors to inverse cosine. The challenge will focus on which 3 points to choose in order to get the hand orientation. Three points we choose to calculate hand angle are the custom 22nd landmark we created previously, landmark 0 and modified landmark 0 (with y-coordinate + 1) shown in Figure 3.12. The reason to choose these three points is that they align and form a straight line when the hand points upward. This is also why we created the 22nd landmark previously because it aligns with our wrist. If the hand is pointing to the right, the coordinate of the 22nd landmark will be at the right-hand side, and the angle calculated will be 90°. The hand angle calculated is fairly accurate because turning your hand can be seen as rotate the center of your hand palm (22nd landmark) based on your wrist (landmark 0). It is almost impossible to rotate your hand without rotating your hand palm.

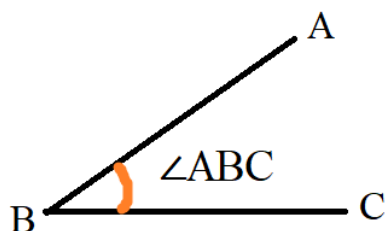
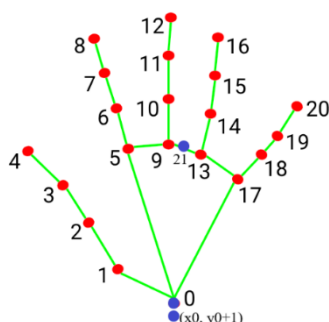
Figure 3.11: Angle $\angle ABC$ 

Figure 3.12: Point choose to calculate Hand Angle

3.4.5 Angle of 3 points

Finding an angle of 3 points is the extension from the hand angle—both using the same formula and coding. Different is hand angle have the 3 points being fixed, while this is a general function that take 3 points as input and outputs the angle between them. Useful when designing gestures and applications.

3.4.6 Detect Finger Up

FingerUp is a function that detects which finger is up (fully extended) and which one is not. This function is one of the most important functions for designing gestures. Figure 3.13 show some of the examples of hand gesture. We can observe that most hand gestures use a combination of finger up and orientation. By detecting which finger is extended and knowing the orientation, we can design most of the hand gestures except some unique gestures such as Q or X (refer to Figure 3.13), which have some unique angle or the finger is partially extended.

One of the simplest methods to detect whether the finger is up is by comparing the coordinates. By comparing the Metacarpophalangeal Joint (MCP, the very bottom joint of the finger) and the tip of the finger, if the tip

position is higher than MCP, we can conclude the finger is extended. For example, in Figure 3.12, landmark 8 is on top of landmark 5, and we can conclude the index finger is extended. Another problem worth mentioning is that position that detects the thumb extending or not is different from other fingers. Even if not extended, the tip position will still be on top of the MCP joint. If we want to detect the right-hand thumb, we need to detect if the tip is on the left of the MCP joint. Because if we extend, the tip will be on the left; when bent, the tip will be on the right. This method works very well, except it only works when hand orientation faces upward. But this still can be solved if we know the hand's orientation. Using Table 3.3, we can easily code out all the situations and have a good finger up detection function.

Table 3.3: Detection of tip & MCP joint to decide if the finger is extended based on hand orientation.

| Hand Orientation | Left-hand Thumb | Right-hand Thumb | Index to pinky finger |
|------------------|--|---|--|
| Upward | Tip on the right (compared to MCP) | Tip on the left (compared to MCP) | Tip on the top (compared to MCP) |
| Downward | Tip on the left | Tip on the right | Tip on the bottom |
| Left | Tip on the top | Tip on the bottom | Tip on the left |
| Right | Tip on the bottom | Tip on the top | Tip on the right |

The coordinate method is handy throughout the project except when designing the “driving mode” application in our project, where we focus on immersive experience compared to assigned hand gestures. “Driving mode” is one of the applications where we allow the user to put their hand like holding the steering wheel to control a device; details will be described in later chapter. But the focus here is that while turning the steering wheel, hand orientation is not precisely in four directions (up, down, left, right), it can be anything in between. When the hand orientation is not exactly in the four directions, the coordinate method's fault detection is very high.

Therefore, the following solution will be using angles. When we extend our finger, we change the angle between the tip, Proximal

Interphalangeal joint (PIP) and MCP. If we refer to the index finger in Figure 3.12 is the angle between landmark 8, landmark 6, and landmark 5. Using the function that finds the angle of the 3 points we created previously, we can set the finger is extended when the angle reaches a certain threshold. The finger is bent when the angle is below a certain threshold to detect which finger is up easily. Even though the coordinate method is more robust when hand orientation is correct, detecting finger up using angle is more versatile and flexible for designing applications.



Figure 3.13: Examples of hand gestures in sign language (Yasen and Jusoh, 2019)

3.4.7 Distance of 2 point

If we have coordinates of 2 points, finding the distance between them can be easily done using the following equation:

$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (3.6)$$

Also equivalent in python:

$$Distance = \text{math.hypot}(x_2 - x_1, y_2 - y_1)$$

This function can be helpful when designing hand gestures such as pinch. It also can use to create some trigger/click hand gestures. For example,

we can design when the tip of the index finger is very close to the tip of the middle finger (distance very short) as a click action.

3.4.8 MQTT setup

So far, we have discussed extracting hand coordinate data from the model and some of the tools and functions we design to design a hand gesture. The second part of our project is controlling a device or multiple devices wirelessly using hand gestures, and the communication protocol we are using is MQTT. There are only three components in MQTT.

- (i) Publisher/Sender (Our Python Program).
- (ii) Subscriber/Receiver (WiFi Robot Car).
- (iii) Broker.

The setup of publisher and subscriber is compulsory, but for the broker, we can either use an online broker or set up a local broker. Publisher and Subscriber setup is very similar, even though Arduino IDE and Python use different languages, but the setup procedure is the same, just library used and syntax is different. The process to setup MQTT is:

- i. Create MQTT client instances
- ii. Connect to broker.
- iii. Subscribe to a Topic (Subscriber/Receiver Only).
- iv. Setup Callback function (Subscriber/Receiver Only).
- v. Publish message to a Topic (Publisher/Send Only)

3.4.9 DC Motor Configuration (Arduino IDE)

Previously we discussed how L298N DC Motor Driver controlled the speed and direction of the motor. This section will explain the software coding for ESP32 to send the correct signal to L298N DC Motor Driver. L298N DC Motor Driver uses three pins to control one DC motor, Enable, Input 1, and Input 2 (Figure 3.6). In this project, we are using 2 DC motors. We can make the car move forward if both wheels spin in the same direction. If we want to turn left, we can let the right motor spin while disabling the left motor. The enable and disable need to be translated to the “HIGH” and “LOW” signals and sent to DC Motor Driver to complete the motion. Table 3.4 shows the

configuration of the Input signal based on movement, where Input 1 and Input 2 is for the left motor, Input 3 and Input 4 is for the right motor. After knowing the configuration, it can be easily coded using Arduino IDE as follow.

```
//For Clockwise motion (forward), in_1 = Low , in_2 = High
digitalWrite(in_1,LOW) ;
digitalWrite(in_2,HIGH) ;
//For Counterclockwise motion (backward), in_1 = High , in_2 = Low
digitalWrite(in_1,HIGH) ;
digitalWrite(in_2,LOW) ;
//For Brake, in_1 = Low , in_2 = Low
digitalWrite(in_1,LOW) ;
digitalWrite(in_2,LOW) ;
```

To control the speed, we need to control the PWM, and PWM can be controlled using the coding below.

```
//Control PWM Range from 0 to 255
analogWrite(pwm,255) ;
```

By controlling the PWM, we control the duty cycle and control the average voltage feed to the motor, at last controlling the speed of the motor. The coding above demonstrates the concept of controlling the DC motor driver by setting the signal on “HIGH” and “LOW” and controlling the PWM. In practical implementation, some libraries provide more user-friendly functions where you can control direction and duty cycle in one line of code.

Table 3.4: DC Motor signal configuration.

| Direction | Input 1 | Input 2 | Input 3 | Input 4 |
|-----------|---------|---------|---------|---------|
| Forward | 0 | 1 | 0 | 1 |
| Backward | 1 | 0 | 1 | 0 |
| Left | 0 | 0 | 0 | 1 |
| Right | 0 | 1 | 0 | 0 |
| Stop | 0 | 0 | 0 | 0 |

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In this project, we designed two applications. The first application, “driving mode” focuses on interactive and immersive experiences. The second application, “precision mode” tests the accuracy limit hand gesture can achieve. In this chapter, we will discuss two terms sensitivity and accuracy. Sensitivity and accuracy can be defined as below:

$$Sensitivity = \frac{\text{Number of true positive assessment}}{\text{Number of all positive assessment}} \quad (4.1)$$

$$Accuracy = \frac{\text{Number of correct assessments}}{\text{Number of all assessments}} \quad (4.2)$$

In this project, most of the gesture detection is original. Therefore, it is tough to quantify our results. We can still adapt the concept of sensitivity and accuracy in our results. Sensitivity in our application indicates that the model detects the gesture when the user makes the gesture. High sensitivity means every hand motion can be detected. Accuracy indicates that among all the detections, how many detections are correct. High accuracy means all the gesture being detected is correct.

This chapter consists of three sections, evaluate the performance of two applications and evaluate some general parameters. The detailed breakdown is as below:

- i. Driving Mode:
 - Starting Driving Mode
 - Control And Action in Driving Mode
 - Sensitivity of Driving Mode
 - Accuracy of Driving Mode
 - Limitation of Driving Mode
- ii. Precision Mode:
 - Starting Precision Mode

- Control And Action in Precision Mode
- Sensitivity of Precision Mode
- Accuracy of Precision Mode
- Limitation of Precision Mode

iii. General parameters:

- MQTT communication delay
- User Interface Smoothness

4.2 Driving Mode

Driving mode is an augmented reality application that allows users to control a virtual steering wheel to mimic the action of driving the actual car. We will go through how to activate this mode, its control, and how the control converts to action in the WiFi robot car. Next, we will evaluate the driving mode's sensitivity, accuracy, and limitation.

4.2.1 Starting Driving Mode

In order to use this application, we will need to go into or start the application first. Both hands pose like holding a steering wheel will trigger Driver Mode, shown in Figure 4.1. Needing to activate the application before using it is helpful if we have many different applications fit in one program. This can avoid operation interference between applications and allow the same hand gesture to be reused in different applications representing different actions.

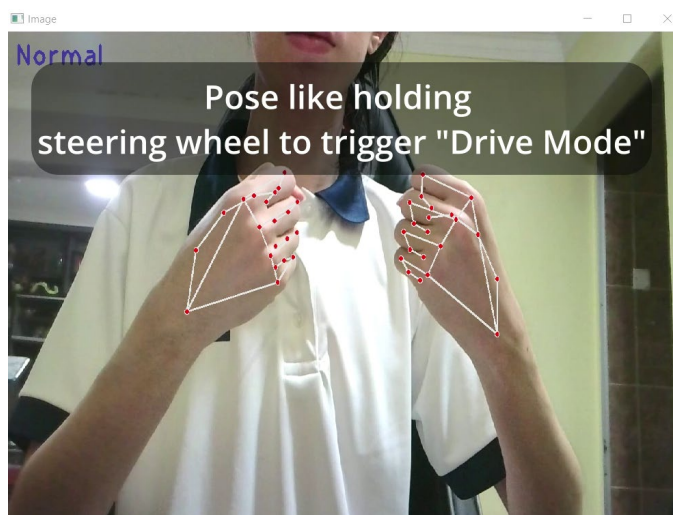


Figure 4.1: Gesture to activate Driving Mode

4.2.2 Control And Action in Driving Mode

We simplify the action of driving a car into two components: control the car's speed and control the car's direction. A car's speed and direction control is not a discrete but continuous value. Previously we discussed that the speed of the DC motor could be controlled by the voltage supplied, and the voltage supplied can be controlled by the PWM. But in practical implementation, small changes/ continuous changes in voltage have no noticeable difference in the speed of the motor. Another limitation is the power source and motor driver. Because we use one power source for both DC motor and motor driver, the voltage range we can play with is 0 – 5V. Table 4.1 shows that the motor is not running until the voltage reaches 3.5V. The actual usable voltage range is even lesser. Therefore, even though continuous control can be efficiently designed based on the continuous angle of hand gestures, the control in this application is discrete control due to hardware limitations.

Directional control can be achieved using speed different characteristics. For example, if the right wheel runs faster than the left wheel, the car will slowly run toward the right. Knowing how to control direction and speed, we can plot a table list down all the possible control. The PWM of the motor can be formulated as below:

$$\text{Left Motor PWM} = 70 + \text{High} + \text{Left} \quad (4.3)$$

$$\text{Right Motor PWM} = 70 + \text{High} + \text{Right} \quad (4.4)$$

In ESP32 coding, we just code according to Table 4.2. To summarize, we can control the direction and speed of the WiFi robot car. We just need to angle our hands like turning a steering wheel to control the direction. To control the speed, raise the right thumb to speed up one level, raise the left thumb to slow down one level, a total of five levels of speed.

Table 4.1: Voltage & Speed relationship table (Based on actual testing).

| PWM | Average Voltage | Speed |
|---------|-----------------|-------------|
| 0 - 70% | 0 - 3.5V | Not Running |

| | | |
|------|------|-----------|
| 80% | 4V | Low |
| 90% | 4.5V | High |
| 100% | 5V | Max Speed |

Table 4.2: DC Motor PWM configuration.

| Speed | Direction | Motor Rotation | Left Motor PWM (%) | Right Motor PWM (%) |
|---------------|-----------|------------------|--------------------|---------------------|
| Forward High | Left | Clockwise | 100 | 90 |
| | Forward | Clockwise | 90 | 90 |
| | Right | Clockwise | 90 | 100 |
| Forward Low | Left | Clockwise | 90 | 80 |
| | Forward | Clockwise | 80 | 80 |
| | Right | Clockwise | 80 | 90 |
| Stop | - | | 0 | 0 |
| Backward Low | Left | Counterclockwise | 90 | 80 |
| | Forward | Counterclockwise | 80 | 80 |
| | Right | Counterclockwise | 80 | 90 |
| Backward High | Left | Counterclockwise | 100 | 90 |
| | Forward | Counterclockwise | 90 | 90 |
| | Right | Counterclockwise | 90 | 100 |

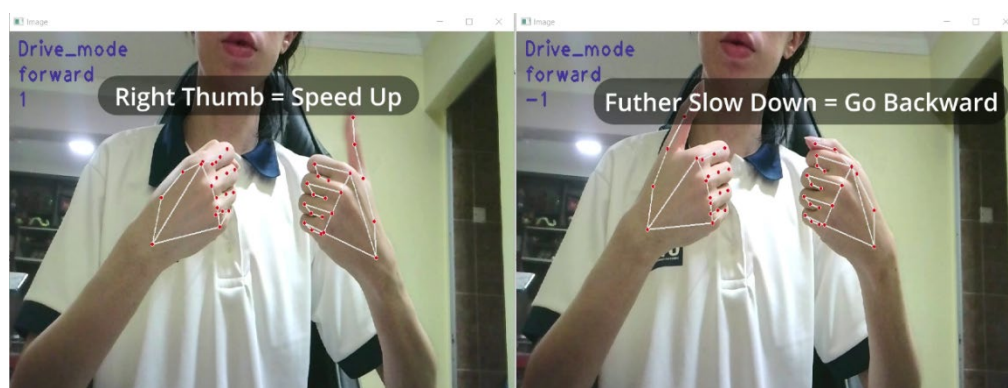


Figure 4.2: Controlling speed in driving mode.

4.2.3 Sensitivity of Driving Mode

Because we only have three directions, we only need to split 360° into three sectors. First, we need to define 0 degrees. Second, we need to set a threshold

for the left and right directions. Angle calculated is based on the custom 22nd landmark (Middle of middle finger and ring finger) of both hands with horizontal, Figure 4.3. We define 0 degrees as both 22nd landmark line with horizontal line. The threshold we set here is 20 degrees. When the angle between both hands is larger than 20°, the direction is left; when smaller than -20°, the direction is right. Figure 4.4 shows the actual implementation. The threshold setting cannot be too small because it is almost impossible for our hands to be completely still if the threshold is set too low, for example, 5°. This application will become highly sensitive; slight movement will make the car change direction and make the user hard to control, resulting in a bad user experience. The threshold of 20° came from trial and error, where I imagine holding a steering wheel and wanting to turn to the right, and 20° is the angle comfortable for me. This application aims to bring an interactive and immersive experience to the user. Therefore, when designing parameters, we focus more on Quality of Experience (QoE) than Quality of Service (QoS) performance metrics. This is one of the characteristics of hand gestures; different people have different hand movement habits, and being very flexible to fit the user can achieve high QoE.

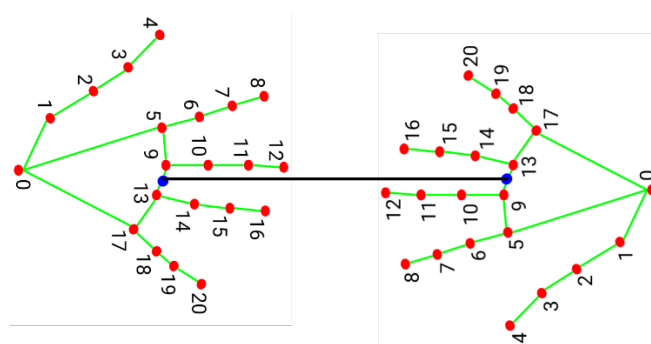


Figure 4.3: Driving Mode Angle Reference.

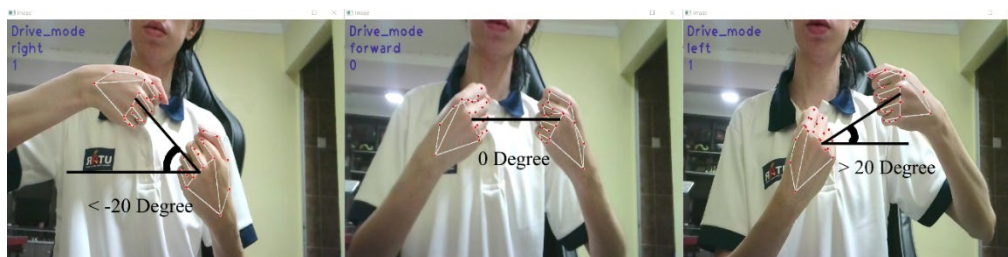


Figure 4.4: Sensitivity of Driving Mode.

4.2.4 Accuracy of Driving Mode

In driving mode, two gestures need to be detected. The angle of two hands to control the direction, and the thumb up to control the speed of the motor. The accuracy of the angle of the two hands is very high because the model constantly detects the back of the hands, and there is space for error (reached threshold only change direction). Change of direction only happens when the angle reaches a certain threshold. The accuracy of thumb up depends on the accuracy of the “Detect Finger Up” functions. Previously, we discussed two methods to detect the finger up: the coordinate method (high accuracy but only for fixed orientation) and the angle method (slightly lower accuracy but suitable for different orientations). We are using the angle method, and there is almost no false detection when moving forward because the hand stays fixed in orientation. But when turning left or right, sometimes there will be false detection, where it detects the thumb is up. This happens because it hides behind our hands when the thumb is not extending. Thumb information is not captured constantly by the camera. Resulting in false detection.

4.2.5 Limitation of Driving Mode

Limitations of driving mode can separate into three aspects:

- (i) Detection Model Limitation.
- (ii) Design Limitation.
- (iii) Hardware Limitation.

The detection model limitation is the limitation due to the detection model. When the detection model detects a hand, it always tries to figure out the 21 landmarks even when an obstacle blocks some part of the hand. For example, when we hold our fists in driving mode, fingers blocked by the palm will still be detected by the model. This results in false detection. This can only be avoided if our design only has gestures that show all the fingers.

Design limitation is a limitation that can be solved with a better design using the same model. Driving mode is a simplified virtual driving simulator used to show hand gesture potential and can be used to control WiFi cars or even drones. But when compared to actual driving, hyper realistic simulation, or even controlling an actual car, there are still many limitations.

Limitations in this application that can be improved by better design (using the same model) are:

1. Our virtual steering wheel turning is based on a 360° angle, while the steering wheel on the car often can turn 2 – 3 circles. In our demonstration, our WiFi car does not have so many directions/angles that require a wide range of input but using better hardware might be a design limitation we will face.
2. Speed switching using the thumb is only one stage. Speed up and slow down in driving mode is stage by stage, speed up one stage or slow down one stage. In practice, it is better to have instant speed control. Our demonstration only has five speed stages, and the brake is in the middle. If the range of speed we can control becomes larger due to hardware upgrades. For example, ten stages of speed variation. If we were running at high speed and met an obstacle want to brake instantly, design in our application could not do it. The speed control mechanism can be improved by better design.

There are two hardware limitations facing in driving mode:

1. The first limitation is space and power source limitation. We discussed previously that the speed control range in our demonstration is very small due to the DC motor driver. The small WiFi car module limits the space to fit an additional power source. With one power source supply to both DC motor driver and motors. The highest voltage we can provide to motors is 5V (DC motor driver will burn if it exceeds 5V). Therefore, the speed range we can control is small.
2. The wheels of the WiFi car in our project will not turn angle. The left and right movement is achieved by relative speed between both motors. This kind of control has less accuracy and control in angle turning. Not a very obvious flaw in driving mode since the driving mode seldom needs to change the angle. But this limitation is magnified in precision mode.

4.3 Precision Mode

Precision mode is an application that demonstrates the precision limit using hand gestures. Able to do precise operation using hand gesture open up more possibility for hand gesture applications. Similar to driving mode, we will go through how to activate this mode, what control it has, and how the command

converts to action. Next, we will evaluate the sensitivity, accuracy, and limitation.

4.3.1 Starting Precision

To start precision mode, we need to pose “1” for both hands (Extend index finger only). Show in Figure 4.5. The starting gesture of precision and driving modes allows users to switch modes anytime.



Figure 4.5: Gesture to activate Precision Mode.

4.3.2 Control and Action in Precision Mode

Table 4.3 lists down all the control and action in precision mode. In precision mode, the aim is to allow the user to set a custom path that commands the WiFi car to move from the initial point to the destination at the desired distance and angle. Therefore, the control we have is, setting the initial point, planning the path, and setting the destination point. The command will not send to the car unless the user confirms the distance and angle. Allow the user to fine-tune until the correct length and angle. A complete operation flow shows in Figure 4.6.

Table 4.3: Control and Action in Precision Mode.

| Gesture | Action |
|--------------------------------------|--------------------------------------|
| Pinch | Set Initial Point |
| Index finger drag from Initial Point | Pull out a line to indicate the path |
| Upward Gun gesture | Set and Adjust Final Point |

| | |
|-----------------|--|
| Thumbs Up | Confirm the path send the command to the car |
| Shaka sign/ Six | Clear and reset all points and path |



Figure 4.6: Operation in precision mode

4.3.3 Sensitivity of Precision Mode

The sensitivity of precision mode is very high. Almost all the gestures can be successfully detected except when the hand is too close to the edge of the screen. Show in Figure 4.7. One reason the initial point can be set freely in any position in our design is to compensate for the edge effect. The distance pixel ratio in this application is 1cm to 10pixels. The window resolution is 1280*720, theoretical maximum distance we can command is 128cm to the left-right and 72cm forward-backward. But due to the sensitivity problem. To maximize the space we can use, the initial point can be set freely. If the user wants to make a forward operation, set the initial point lower to create more space forward, same with the backward operation.

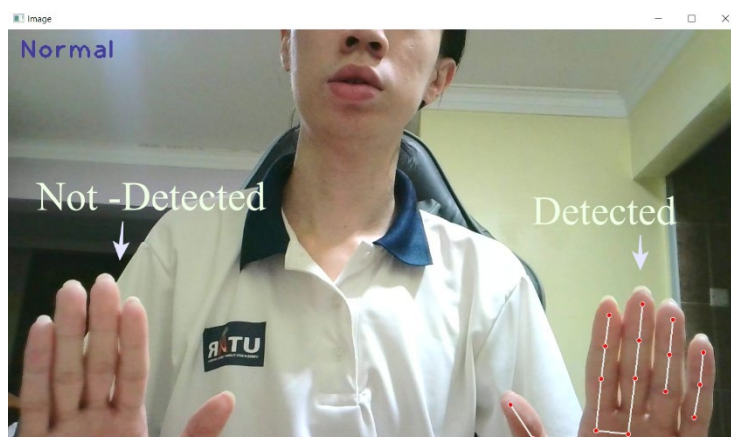


Figure 4.7: Sensitivity at the edge of the screen

4.3.4 Accuracy of Precision Mode

Two accuracies to evaluate in precision mode. The first one is accuracy in detecting all the hand gestures. The second one is the path accuracy executed by the WiFi car. In this application, we have a total of 5 hand gestures. Our design handles all hand gestures very well except for thumbs up. The only false detection that happens is on the thumbs up hand gesture. It often false detects thumbs up as a shaka sign, where it clears out all the path instead of sending the command to the car. This is because the finger up algorithm used in this project uses the angle method instead of the coordinate method. This has to do with when our hand transitions from gun gesture to thumb up gesture, the angle of our pinky finger being detected as extending. This false detection can be solved if we use the coordinate method because the shaka sign and thumb up have distinct differences in coordinate. Additionally, both gestures have no orientation issue, unlike driving mode.

Before discussing the path accuracy, we need to know how the distance and angle in python are converted to WiFi car action. There are a lot of factors that can affect the performance of a DC motor. To minimize the effect, we need to fix some of the parameters and make some assumptions. We need to assume every motor rotated at the same angle when the same amount of voltage is supplied in the same period. Next, we will fix all the motor activities in 80% PWM. Choosing 80% is that it is the slowest but most stable PWM. 70% sometimes the motor does have enough power to rotate, 90% is too fast, slower rotation meaning more refined/better accuracy. After some testing, we figure out that under 80% PWM, 1 second can move 23cm, and 450 milliseconds can rotate 90 degrees (Table 4.4). We can formulate the time needed for any distance and angle as below:

$$Run\ time = \frac{distance * 1000\text{milliseconds}}{23\text{cm}} \quad (4.5)$$

$$Rotate\ time = \frac{angle * 450\ \text{milliseconds}}{90\ \text{degrees}} \quad (4.6)$$

Figure 4.8 shows a demonstration example. The distance and angle are set to 30cm, 30.47°. Based on the formula above, the run time will be 1304 milliseconds and rotate time will be 152 milliseconds. Operation time can be easily coded in Arduino IDE. The movement of the WiFi car will first rotate right for 152ms, then run forward 1304ms. Figure 4.9 shows the result of precision mode. The measuring point of a WiFi car should be the center between both wheels because there is the center of rotation. From Figure 4.9, we can see that distance is exactly 30cm as we set, but the angle is 37.5 degrees. The accuracy executed by the car is not accurate due to hardware limitations. Distance accuracy is much higher than angle, usually will be precisely what we set; in the worst case, no more than ± 3 cm. Angle is tough to achieve as precise as we set, usually will deviate ± 10 degrees, worst case can go up to ± 20 degrees. But this is solvable with better hardware (feedback system on motor).

Table 4.4: DC Motor behaviour mapping.

| PWM | Action | Time | Result |
|------|------------------|------------------|------------|
| 80 % | Forward-Backward | 1 second | 23 cm |
| 80 % | Left-Right | 450 milliseconds | 90 degrees |

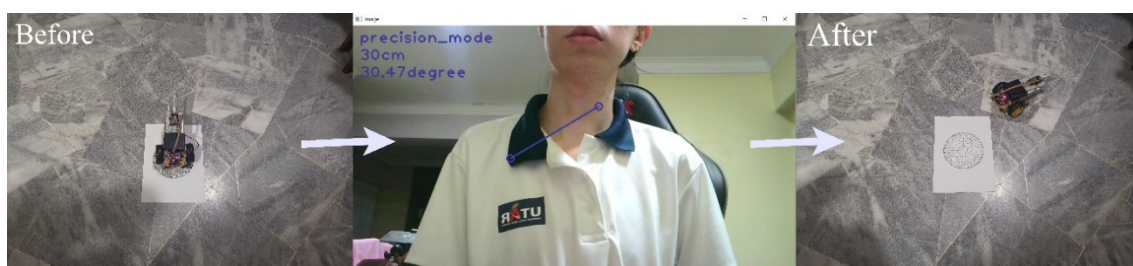


Figure 4.8: Demonstration example for precision mode.

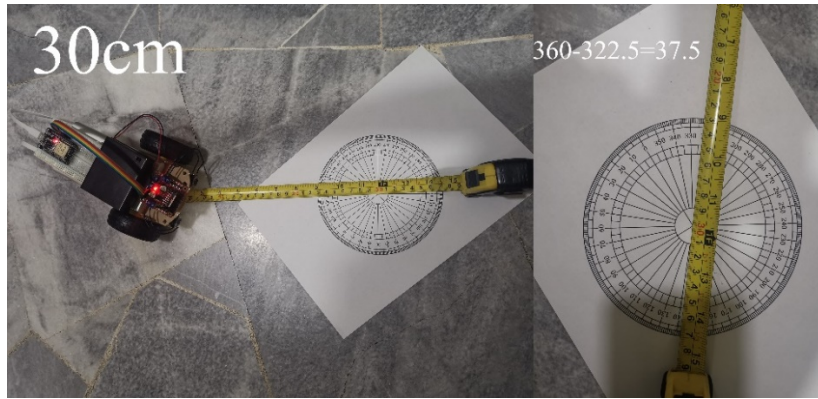


Figure 4.9: Result of Precision Mode.

4.3.5 Limitation of precision mode

Same as driving mode, the limitation of precision mode can separate into three aspects:

- (i) Detection Model Limitation.
- (ii) Design Limitation.
- (iii) Hardware Limitation.

For detection model limitation, hand gestures might not be detected when close to the edge of the window. Even able to detect, it will often result in false detection. This is not a severe problem when came to practical implementation. What can be done is to distance the camera from the user so that the user has more space for hand gesture movement. Even if the edge problem is not solved, it is normal for the model not to detect correctly because it is not at the edge meaning only part of the hand is captured by the camera.

We can set a border and exclude all the detection outside the border to prevent edge false detection.

The main design limitation is the distance to pixel ratio is fixed to 1:10. For example, in a 1280*720 resolution window, the theoretical maximum distance we can command is 128cm to the left-right and 72cm forward-backward. This can be solved by adding a variable ratio, letting the user change the ratio so they can control further distance or smaller distance for precise application. Although only one limitation is mentioned here, many improvements to precision mode can be made. For example, currently, this

application only supports one node to another node (initial to a destination point). This can be improved to many nodes. For undoing, currently only can clear off all the paths, but having many nodes can consider design a gesture used to remove one node only to improve user experience.

One of the most significant hardware limitations in precision mode is too many factors affecting the motor rotation. In the design, we neglected many factors. But in practice, the weight of the car body, battery voltage, and different motors will affect the rotation at the same voltage. Even though both motors are identical, they have slightly different and rotate at different speeds in the same voltage. It is usually negligible, but this difference is amplified when it comes to angle rotation, especially working at small angles. This can be solved in many ways, changing the DC motor to a stepper motor that has high precision or adding a rotary encoder for the DC motor and making a closed-loop PID system. With a PID system, the motor can turn at a precise angle, significantly improving the angle turning accuracy we faced in our project.

4.4 General Parameters

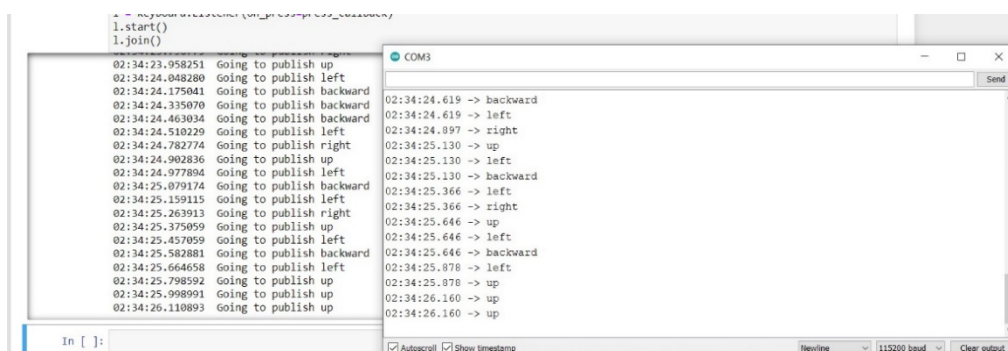
General parameters are parameters that are not limited to application. For example, the system's response time, in this project, all communication is done using the MQTT protocol. Therefore, the response time depends on the MQTT protocol and is the same across all applications. Another parameter will be the smoothness of the user. This can be measured using frame per second (FPS). If the FPS is very low, the user will experience laggy and affect the hand gesture control.

4.4.1 MQTT communication delay

The response time of a system is crucial. Not only in driving mode and precision mode, but for all applications. Take driving mode, for example; it is terrible if the car takes a few seconds delay to turn left after turning the steering wheel. It brings a bad user experience and indicates users are losing control of the devices. Figure 4.10 shows the delay of the MQTT protocol. Delay can calculate using:

$$\text{Delay} = \text{Message receive time} - \text{Message sending time} \quad (4.7)$$

The delay of MQTT is not fixed. If we take a few samples to calculate, we can get the delay range from 80 to 271ms. Human reaction time is around 200 to 250ms. Therefore, we can conclude the response time of MQTT is good enough for our project. The response is almost instant, and the delay is unnoticeable.



The screenshot shows a terminal window with the following output:

```

1.start()
1.join()
02:34:23.958251 Going to publish up
02:34:24.048280 Going to publish left
02:34:24.175041 Going to publish backward
02:34:24.335070 Going to publish backward
02:34:24.463034 Going to publish backward
02:34:24.510229 Going to publish left
02:34:24.782774 Going to publish right
02:34:24.902836 Going to publish up
02:34:24.977894 Going to publish left
02:34:25.079174 Going to publish backward
02:34:25.159115 Going to publish left
02:34:25.263913 Going to publish right
02:34:25.375059 Going to publish up
02:34:25.457059 Going to publish left
02:34:25.582881 Going to publish backward
02:34:25.664658 Going to publish left
02:34:25.798592 Going to publish up
02:34:25.998991 Going to publish up
02:34:26.110893 Going to publish up
  
```

Next to it is a COM3 window showing received commands:

```

02:34:24.619 -> backward
02:34:24.619 -> left
02:34:24.897 -> right
02:34:25.130 -> up
02:34:25.130 -> left
02:34:25.130 -> backward
02:34:25.366 -> left
02:34:25.366 -> right
02:34:25.646 -> up
02:34:25.646 -> left
02:34:25.646 -> backward
02:34:25.878 -> left
02:34:25.878 -> up
02:34:26.160 -> up
02:34:26.160 -> up
  
```

Figure 4.10: MQTT message delay

4.4.2 The Smoothness of The User Interface

One of the indications to present smoothness of user is frame per second. The higher the FPS, the more instant feedback can be sent to the user. Figure 4.11 shows that in idle (no hand), the fps is stable at an average of 16 fps. In one hand and two hands present, only four fps drop, stable at an average of 12 fps. The smoothness of 12fps is acceptable but not very idea. The average frame rate in a movie is 24fps, whereas most gamers nowadays request 60 fps or higher. The limiting factor of the fps is mainly due to the low spec of the PC I am using. If we compare the frame drop of idle and detection, we notice that the model and application are very light. The same programming run on a better machine definitely can bring better smoothness and experience.

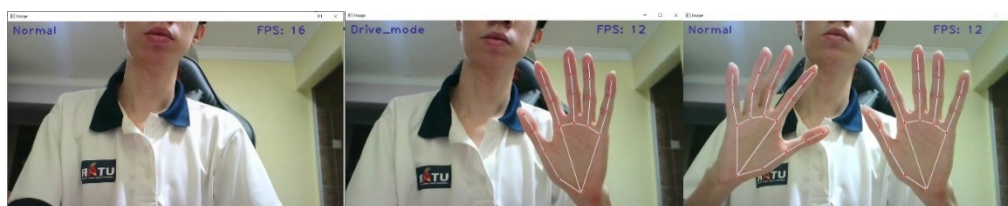


Figure 4.11: FPS in idle, one hand detected and two hands detected.

4.5 Summary

In this project, we designed two applications driving mode and precision mode. The driving mode has good sensitivity and accuracy in controlling the direction. But for controlling the speed using thumb up, false detection can happen when turning left and right at a large angle. The precision mode also has high sensitivity except when the hand is at the edge. Accuracy of precision is good in the distance but yet to improve in angle. Both applications' limitations are either design limitations or hardware limitations, which can be solved using better hardware or better design. The MQTT protocol delay is very low, ranging from 80 to 271ms. The FPS is very stable at 12 fps and can be improved using a better machine.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In this project, both applications designed achieved their goal. An immersive and interactive experience for driving mode and showing the precision hand gestures can be achieved in precision mode. Although the result is not perfect, the result is good enough when considering the hardware used in the demonstration, only DC motors and ESP32, for hand gesture detection only use ordinary webcam. Most of the afford put into this project is hand gesture detection and design. When the system can detect all the hand gestures, the remaining part is just creativity in designing the application. Most hand gesture detection, such as “finger up” and “hand angle” is very original because not many people have developed applications using MediaPipe yet. Also, due to the pandemic, all of the projects are done at home, and I cannot go to campus to test my project with other users. Feedback from other users definitely helps improve this project.

Hand gesture detection is a very mature technology. The reason hand gesture control is not yet popular and widespread is that there still is not a framework that can easily define a hand gesture and give it meaning. Take the MediaPipe hand model we use in this project. The detection/identification of hand and joint is very accurate and stable, but it does not provide any meaning, just a bunch of coordination. Without a framework or tools that help to define and design hand gestures, it is hard to design an application that uses them. Therefore, most of the time spent and challenges is designing those tools to help identify hand gestures.

At this moment, the MediaPipe hand model provides the highest flexibility and is closest to the scene in the movie Iron man. As other technologies such as virtual reality and holographic display became more mature and common, hand gestures definitely can work together and bring the experience to the next level.

5.2 Recommendations for future work

The first significant improvement that needs to be done will be upgrading the demonstration hardware. Only hardware/motor that can achieve high precision can test the full potential of hand gesture precision. Next, we should find more users to test out the applications we designed, test the adaptability of the applications, and receive feedback from users to know which part needs to improve further. After that, we should evaluate the hand gesture designed tools such as “finger up” or find a better algorithm that can detect finger up better. The last but most important recommendation is finding some actual problem that can be solved using hand gestures. Due to the flexibility hand gesture can achieve. Every application will need a different gesture; this is where creativity kicks in. Figure out what the gesture needed and find a suitable algorithm to detect the gesture.

REFERENCES

Analog computing returns | MIT News | Massachusetts Institute of Technology (no date). Available at: <https://news.mit.edu/2010/gesture-computing-0520> (Accessed: 18 August 2021).

Badi, H. (2016) ‘RETRACTED ARTICLE: A Survey on Recent Vision-Based Gesture Recognition’, *Intelligent Industrial Systems*, 2(2), pp. 179–191. doi: 10.1007/s40903-016-0046-9.

Bazarevsky, V. *et al.* (2019) ‘BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs’. Available at: <https://arxiv.org/abs/1907.05047v2> (Accessed: 20 August 2021).

Chen, L. *et al.* (2013) ‘A Survey on Hand Gesture Recognition’, in *Proceedings of the 2013 International Conference on Computer Sciences and Applications*. USA: IEEE Computer Society (CSA ’13), pp. 313–316. doi: 10.1109/CSA.2013.79.

Chhajed, R. R. *et al.* (2021) ‘Messaging and Video Calling Application for Specially Abled people using Hand Gesture Recognition’, *2021 6th International Conference for Convergence in Technology, I2CT 2021*. doi: 10.1109/I2CT51068.2021.9417924.

F. Li, J. J. and Y. S. (2020a) ‘CS231n: Convolutional Neural Networks for Visual Recognition’, *Stanford University*, pp. 1–9. Available at: <https://cs231n.github.io/classification/> (Accessed: 18 August 2021).

F. Li, J. J. and Y. S. (2020b) ‘CS231n: Convolutional Neural Networks for Visual Recognition’, *Stanford University*, pp. 1–9. Available at: <https://cs231n.github.io/neural-networks-3/> (Accessed: 18 August 2021).

FAR CRY 6 Behind The Scenes Face Model Motion Capture Trailer Giancarlo Esposito Antó - YouTube (no date). Available at: https://www.youtube.com/watch?v=vPApo_WHq2Q (Accessed: 18 August 2021).

Guna, J. *et al.* (2014) ‘An Analysis of the Precision and Reliability of the Leap Motion Sensor and Its Suitability for Static and Dynamic Tracking’, *Sensors*, 14(2), pp. 3702–3720. doi: 10.3390/s140203702.

Hansard, M. *et al.* (2012) *Time-of-Flight Cameras: Principles, Methods and Applications*. Springer Publishing Company, Incorporated.

Hasan, M. and Mishra, P. (2012) ‘Hand Gesture Modeling and Recognition using Geometric Features: A Review’, *Canadian Journal on Image Processing and Computer Vision*, 3, pp. 12–26.

HERO-UltraLeap_Product05342_edit.jpg (1600×1067) (no date). Available at: https://cms.ultraLeap.com/app/uploads/2020/02/HERO-UltraLeap_Product05342_edit.jpg (Accessed: 18 August 2021).

Inc., G. (2020) *Hands - mediapipe*. Available at: <https://google.github.io/mediapipe/solutions/hands> (Accessed: 20 August 2021).

Johannsen, G. (2011) 'Human-machine Interaction', in. *KNN_final1_ibdm8a.png (591×515)* (no date). Available at: https://res.cloudinary.com/dyd911kmh/image/upload/f_auto,q_auto:best/v1531424125/KNN_final1_ibdm8a.png (Accessed: 18 August 2021).

Kumar, G. and Bhatia, P. K. (2014) 'A detailed review of feature extraction in image processing systems', *International Conference on Advanced Computing and Communication Technologies, ACCT*, pp. 5–12. doi: 10.1109/ACCT.2014.74.

Last Minute Engineers (2021) *In-Depth: Interface L298N DC Motor Driver Module with Arduino*, *lastminuteengineers.com*. Available at: <https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/> (Accessed: 4 April 2022).

Lawrence, S. *et al.* (1997) 'Face recognition: a convolutional neural-network approach', *IEEE Transactions on Neural Networks*, 8(1), pp. 98–113. doi: 10.1109/72.554195.

Li, Z. *et al.* (2021) 'A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects', *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21. doi: 10.1109/TNNLS.2021.3084827.

Lin, T.-Y. *et al.* (2016) 'Feature Pyramid Networks for Object Detection'. Available at: <https://arxiv.org/abs/1612.03144v2> (Accessed: 20 August 2021).

Lin, T. Y. *et al.* (2020) 'Focal Loss for Dense Object Detection', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2), pp. 318–327. doi: 10.1109/TPAMI.2018.2858826.

Liu, H. and Wang, L. (2018) 'Gesture recognition for human-robot collaboration: A review', *International Journal of Industrial Ergonomics*, 68, pp. 355–367. doi: 10.1016/J.ERGON.2017.02.004.

Liu, P. *et al.* (2019) 'Hand Gesture Recognition Based on Single-Shot Multibox Detector Deep Learning', *Mobile Information Systems*, 2019. doi: 10.1155/2019/3410348.

Liu, W. *et al.* (2016) 'SSD: Single shot multibox detector', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2.

Lugaresi, C. *et al.* (2019) ‘MediaPipe: A Framework for Building Perception Pipelines’. Available at: <https://arxiv.org/abs/1906.08172v1> (Accessed: 19 August 2021).

Mohamed Elgendy (2020) *Deep Learning for Vision Systems*.

Patel, N. and He, S. (2018) ‘A Survey on Hand Gesture Recognition Techniques, Methods and Tools’, *International Journal of Research in Advent Technology*, 6(6). Available at: www.ijrat.org (Accessed: 18 August 2021).

Pavlovic, V. I., Sharma, R. and Huang, T. S. (1997) ‘Visual interpretation of hand gestures for human-computer interaction: A review’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7), pp. 677–695. doi: 10.1109/34.598226.

Premaratne, P., Nguyen, Q. and Premaratne, M. (2010) ‘Human Computer Interaction Using Hand Gestures’, in Huang, D.-S. *et al.* (eds) *Advanced Intelligent Computing Theories and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 381–386.

Redmon, J. *et al.* (2016) ‘You only look once: Unified, real-time object detection’, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 779–788. doi: 10.1109/CVPR.2016.91.

Redmon, J. and Farhadi, A. (2017) ‘YOLO9000: Better, faster, stronger’, in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pp. 6517–6525. doi: 10.1109/CVPR.2017.690.

Redmon, J. and Farhadi, A. (2018) ‘YOLOv3: An Incremental Improvement’. Available at: <https://arxiv.org/abs/1804.02767v1> (Accessed: 18 August 2021).

Saha, S. (2018) ‘A Comprehensive Guide to Convolutional Neural Networks — Towards Data Science’, *Towards Data Science*, pp. 1–8. Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> (Accessed: 18 August 2021).

Santos, S. (2018) *ESP32 Pinout Reference: Which GPIO pins should you use? | Random Nerd Tutorials, RANDOM NERD TUTORIALS*. Available at: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/> (Accessed: 22 April 2022).

Simon, T. *et al.* (2017) ‘Hand Keypoint Detection in Single Images using Multiview Bootstrapping’, *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January, pp. 4645–4653. Available at: <https://arxiv.org/abs/1704.07809v1> (Accessed: 19 August 2021).

Tang, J. *et al.* (2019) ‘Position-Free Hand Gesture Recognition Using Single Shot MultiBox Detector Based Neural Network’, in *Proceedings of 2019 IEEE International Conference on Mechatronics and Automation, ICMA 2019*. Institute of Electrical and Electronics Engineers Inc., pp. 2251–2256. doi: 10.1109/ICMA.2019.8816203.

Wei, S.-E. *et al.* (2016) ‘Convolutional Pose Machines’, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December, pp. 4724–4732. Available at: <https://arxiv.org/abs/1602.00134v4> (Accessed: 19 August 2021).

Xia, Z. *et al.* (2019) ‘Vision-Based Hand Gesture Recognition for Human-Robot Collaboration: A Survey’, *2019 5th International Conference on Control, Automation and Robotics, ICCAR 2019*, pp. 198–205. doi: 10.1109/ICCAR.2019.8813509.

Yasen, M. and Jusoh, S. (2019) ‘A systematic review on hand gesture recognition techniques, challenges and applications’, *PeerJ Computer Science*, 2019(9), p. e218. doi: 10.7717/peerj-cs.218.

Yingxin, X. *et al.* (2016) ‘A Robust Hand Gesture Recognition Method via Convolutional Neural Network’, in *2016 6th International Conference on Digital Home (ICDH)*, pp. 64–67. doi: 10.1109/ICDH.2016.023.

Zhang, F. *et al.* (2020) ‘MediaPipe Hands: On-device Real-time Hand Tracking’. Available at: <https://arxiv.org/abs/2006.10214v1> (Accessed: 13 August 2021).

APPENDICES

Appendix A: Coding for MediaPipe Hand custom function

```

import cv2
import mediapipe as mp
import math
import time
import numpy as np
import imutils
import pyautogui

# Function to convert
def listToString(s):
    # initialize an empty string
    str1 = ""
    # traverse in the string
    for ele in s:
        str1 += str(ele)
    # return string
    return str1

# Find angle from horizontal of start point to end
point
def find_angle(start_point, end_point):
    #Get hand angle
    start_point = np.array([start_point[0],
start_point[1]])
    end_point = np.array([end_point[0],
end_point[1]])
    start_point_hori = np.array([start_point[0]+1,
start_point[1]])

    ba = start_point_hori - start_point
    bc = end_point - start_point

    cosine_angle = np.dot(ba, bc) /
(np.linalg.norm(ba) * np.linalg.norm(bc))
    angle = np.arccos(cosine_angle)
    angle = round(np.degrees(angle), 2)
    return angle

def within_radius(center, point, radius):
    if (point[1]-center[0])**2 + (point[2]-
center[1])**2 <= radius**2 :
        return True
    else:
        return False

```

```

# Convert pixel dist & angle to MQTT command
def convert_to_MQTT(distance, angle,
move_direction):
    # Process angle to decide turn left or right
    if angle <= 90:
        if move_direction == "forward":
            angle = 90 - angle
            direction = "03"
        else:
            angle = angle-90
            direction = "04"
    elif 90 < angle <= 180:
        if move_direction == "forward":
            angle = angle-90
            direction = "04"
        else:
            angle = 90 - angle
            direction = "03"

    # remove negative value on angle
    angle = abs(angle)

    turn_time = round(angle/90*450)
    run_time = round(distance/23*1000)

    # Direction + time = command
    turn_command = direction +
str(turn_time).zfill(4)
    if move_direction == "forward":
        run_command = "01" +
str(run_time).zfill(4)
    else:
        run_command = "02" + str(run_time).zfill(4)

    return turn_command, run_command

class custom_mediapipe:
    def __init__(self, mode=False, max_hands=2,
model_complexity = 1,
min_detection_confidence=0.5,
min_tracking_confidence=0.5):
        # Parameters
        self.mode = mode
        self.max_hands = max_hands
        self.model_complexity = model_complexity
        self.min_detection_confidence =
min_detection_confidence
        self.min_tracking_confidence =
min_tracking_confidence
        # Call solution "hands"

```

```

        self.mp_hands = mp.solutions.hands
        # Call detection model and set-up parameters
        self.hands = self.mp_hands.Hands(self.mode,
self.max_hands, self.model_complexity,

self.min_detection_confidence,

self.min_tracking_confidence)
        # Call solution "drawing_utils"
        self.mp_drawing = mp.solutions.drawing_utils

    def findhand(self, img, draw = True):
        all_hand_data = []

        imgRGB = cv2.cvtColor(img,
cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)

        if self.results.multi_hand_landmarks:
            # Get 2 hand information
            for hand_num, hand_landmark in enumerate
(self.results.multi_hand_landmarks):
                hand_data = {}
                landmarks_list = []

                # Get left hand or right hand label
                self.handtype =
self.results.multi_handedness[hand_num].classificati
on[0].label
                hand_data["handType"] =
self.handtype

                # Get coordinate
                for id, lm in
enumerate(hand_landmark.landmark):
                    h,w,c = img.shape
                    cx, cy = int(lm.x*w) ,
int(lm.y*h)

                    landmarks_list.append([id,cx,cy])

                    # Create new landmark, between 9 &
13
                    x_mid =
int((landmarks_list[9][1]+landmarks_list[13][1])/2)
                    y_mid =
int((landmarks_list[9][2]+landmarks_list[13][2])/2)

                    landmarks_list.append([21,x_mid,y_mid])

```

```

        #Get hand angle
        a = np.array([landmarks_list[0][1],
landmarks_list[0][2]])
        b = np.array([landmarks_list[0][1],
landmarks_list[0][2]+1])
        c = np.array([landmarks_list[13][1],
landmarks_list[13][2]])

        ba = a - b
        bc = c - b

        cosine_angle = np.dot(ba, bc) /
(np.linalg.norm(ba) * np.linalg.norm(bc))
        angle = np.arccos(cosine_angle)
        angle = round(np.degrees(angle), 2)

        if landmarks_list[13][1] <
landmarks_list[0][1]:
            angle = 360-angle

        hand_data["handAngle"] = angle
        hand_data["landmarks_list"] =
landmarks_list
        all_hand_data.append(hand_data)

        #self.handtype =
self.results.multi_handedness[0].classification[0].l
abel
        for hand_landmarks in
self.results.multi_hand_landmarks:
            if draw:

self.mp_drawing.draw_landmarks(img, hand_landmarks,
self.mp_hands.HAND_CONNECTIONS)

        return img, all_hand_data

    # Find angle
    def threePointAngle(self, point1, point2,
point3):
        #Get hand angle
        a = np.array([point1[1], point1[2]])
        b = np.array([point2[1], point2[2]])
        c = np.array([point3[1], point3[2]])

        ba = a - b
        bc = c - b

        cosine_angle = np.dot(ba, bc) /

```

```

(np.linalg.norm(ba) * np.linalg.norm(bc))
    angle = np.arccos(cosine_angle)
    angle = round(np.degrees(angle), 2)
    return angle

# Detect how many finger is up
def fingerup(self, hand_data):
    landmarks_list = hand_data["landmarks_list"]
    hand_type = hand_data["handType"]

    tipsID = [4,8,12,16,20]
    fingerup_list = []
    vertical_hand_orientation = 'None'
    horizontal_hand_orientation = 'None'
    if self.results.multi_hand_landmarks:
        #Check finger down
        if landmarks_list[9][2] <
landmarks_list[0][2]:
            vertical_hand_orientation = 'Up'
        else:
            vertical_hand_orientation = 'Down'
        #Check left right orientation
        if landmarks_list[9][1] <
landmarks_list[0][1]:
            horizontal_hand_orientation = 'Left'
        else:
            horizontal_hand_orientation =
'Right'

        # Check thumb angle
        thumb_angle =
self.threePointAngle(landmarks_list[1],
landmarks_list[2], landmarks_list[3])

        if 170 < thumb_angle < 180:
            fingerup_list.append(1)
        else:
            fingerup_list.append(0)

        # Check other Fingers
        for id in range (0,4):
            if
self.threePointAngle(landmarks_list[5+4*id],
landmarks_list[6+4*id],
landmarks_list[8+4*id]) > 100:
                fingerup_list.append(1)
            else:
                fingerup_list.append(0)

```

```
        return fingerup_list,
vertical_hand_orientation,
horizontal_hand_orientation

def distance(self, point1, point2, img = None):
    if self.results.multi_hand_landmarks:
        x1,y1 = point1[1:]
        x2,y2 = point2[1:]
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

        info = (x1, y1, x2, y2, cx, cy)

        #find distance
        length = math.hypot((x2-x1), (y2-y1))

        if img is not None:
            cv2.line(img, (x1,y1), (x2,y2),
(182, 89, 155), 3)
            return length, img, info
        else:
            return length, info
```

Appendix B: Coding for Set Up MQTT

```

# Setup MQTT
import paho.mqtt.client as mqtt
flag_connected = 0

def on_connect(client, userdata, flags, rc):
    global flag_connected
    flag_connected = 1
    print("MQTT connected")

def on_disconnect(client, userdata, rc):
    global flag_connected
    flag_connected = 0
    print("MQTT disconnected")

#Create new client instance
client =mqtt.Client('testclient')
print("Created Client Instance")
client.on_connect = on_connect
client.on_disconnect = on_disconnect
#Connect to broker
broker_hostname = '192.168.0.107'
broker_port = 1883
# Not connect by me
client.connect(broker_hostname, broker_port)

```

Appendix C: Main Coding for Two Applications

```

# Initialize Parameters
wCam, hCam = 1280, 720
wScreen, hScreen= 1920, 1080
operate_mode = "Normal"
move_state = "None"
pmove_state = "None"
draw_start_check = False
draw_line_check = False
draw_end_check = False
send_check = False
center = [0,0]
center_end = [0,0]
speed = 0
can_trigger = True
drive_command = [0,0,0]
pdrive_command = [0,0,0]
cTime = 0
pTime = 0

# Capture Webcam
cap = cv2.VideoCapture(0)

```

```

cap.set(3, wCam)
cap.set(4, hCam)
detector = custom_mediapipe()
client.loop_start()

while True:
    success, img = cap.read()
    # Mirror flip webcam Img
    img = cv2.flip(img,1)
    #Setup mediapipe
    img, hands = detector.findhand(img)

    if hands:
        # Hand 1
        hand1 = hands[0]
        # If 2 hand, hand1 change to left hand
        if len(hands) == 2 and hands[0]["handType"]
== "Right":
            hand1 = hands[1]
            landmarks_list1 = hand1["landmarks_list"] #
List of 21 Landmark points
            handType1 = hand1["handType"] # Handtype
Left or Right
            handAngle1 = hand1["handAngle"] # Hand angle

            fingers1, ver_hand_ori1, hor_hand_ori1 =
detector.fingerup(hand1)

            if len(hands) == 2:
                # Hand 2
                hand2 = hands[1]
                # Hand 2 change to right hand
                if hands[0]["handType"] == "Right":
                    hand2 = hands[0]
                    landmarks_list2 = hand2["landmarks_list"] #
List of 21 Landmark points
                    handType2 = hand2["handType"] # Hand Type
"Left" or "Right"
                    handAngle2 = hand2["handAngle"] # Hand angle

                    fingers2, ver_hand_ori2, hor_hand_ori2 =
detector.fingerup(hand2)

#####
#####
    # Drive mode
    if len(hands) == 2 and 40 <handAngle1< 70 and
290 <handAngle2< 320:
        operate_mode = "Drive_mode"

```



```

    if len(hands) == 2 and operate_mode ==
"Drive_mode":

        # Drive mode Direction
        deltaX = landmarks_list1[21][1] -
landmarks_list2[21][1]
        deltaY = landmarks_list2[21][2] -
landmarks_list1[21][2]
        if deltaX:
            angle = np.arctan(deltaY / deltaX) * 180
/ np.pi

        if fingers1 == [1,1,1,1,1] and fingers2 ==
[1,1,1,1,1]:
            move_state = "brake"
        else:
            if angle > 20:
                move_state = "left"
            elif angle < -20:
                move_state = "right"
            elif -20 < angle < 20:
                move_state = "forward"

        # Drive mode speed
        if can_trigger and fingers1 == [1,0,0,0,0]:
            can_trigger = False
            if speed > -2:
                speed -= 1
        elif can_trigger and fingers2 ==
[1,0,0,0,0]:
            can_trigger = False
            if speed < 2:
                speed += 1
        if fingers1 == [0,0,0,0,0] and fingers2 ==
[0,0,0,0,0]:
            can_trigger = True

        # Thumb angle
        thumbangle =
detector.threePointAngle(landmarks_list1[1],landmark
s_list1[2],landmarks_list1[3])

        # Driving mode convert to command
        # Speed judgement (+ve is forward, -ve is
backward)
        if speed>0:
            drive_command[0] = 1
        elif speed<0:

```

```

        drive_command[0] = 2
    else:
        drive_command[0] = 0
        # High speed judgement
        if abs(speed) == 2:
            drive_command[1] = 1
        else:
            drive_command[1] = 0
        # Direction judgement
        if move_state == "left":
            drive_command[2] = 1
        elif move_state == "right":
            drive_command[2] = 2
        else:
            drive_command[2] = 0

    cv2.putText(img, str(move_state), (20, 100),
                cv2.FONT_HERSHEY_PLAIN, 3,
(154,51,60), 3)
    cv2.putText(img, str(speed), (20, 150),
                cv2.FONT_HERSHEY_PLAIN, 3,
(154,51,60), 3)

    #If drive_command changed then send
    if drive_command != pdrive_command:
        #Publish message
        client.publish("car/move",
listToString(drive_command))
        pdrive_command = drive_command.copy()

#####
#####
    # Precision mode
    if len(hands) == 2:
        if fingers1[1:] == [1,0,0,0] and
fingers2[1:] == [1,0,0,0] and operate_mode !=
"precision_mode":
            operate_mode = "precision_mode"

    if operate_mode == "precision_mode":
        # Draw circle
        if hands:
            distance, info =
detector.distance(landmarks_list1[4],landmarks_list1
[8])
            if distance < 30 and draw_start_check ==
False:
                center = info[4:]
                draw_start_check = True

```

```

        #If circle drawn, touch and drag
        if draw_start_check == True and
within_radius(center, landmarks_list1[8], 5):
            draw_line_check = True

        # Set end point
        if draw_start_check == True and
draw_line_check == True and fingers1 == [1,1,1,0,0]:
            center_end = landmarks_list1[8][1:]
            draw_end_check = True

        #Reset
        if fingers1 == [1,0,0,0,1]:
            draw_start_check, draw_line_check,
draw_end_check= 0,0,0

        if draw_start_check:
            cv2.circle(img, center, 10, (154,51,60),
3)

        if draw_line_check:
            if draw_end_check:
                cv2.line(img, center, center_end,
(154,51,60), 3)
            else:
                cv2.line(img, center,
landmarks_list1[8][1:], (154,51,60), 3)
            if draw_end_check:
                cv2.circle(img, center_end, 10,
(154,51,60), 3)

        #Calculate distance between start and end
point
        if draw_start_check and draw_line_check and
draw_end_check:
            # Distance
            dist = math.hypot(center[0] -
center_end[0], center[1] - center_end[1])
            # Pixel scale to cm
            dist = round(dist/10)
            dist_show = str(dist) + ("cm")
            cv2.putText(img, dist_show,
(20,100),cv2.FONT_HERSHEY_PLAIN, 3, (154,51,60), 3)
            # Angle
            angle_1 = find_angle(center,center_end)
            angle_show = str(angle_1) + "degree"
            cv2.putText(img, str(angle_show),
(20,150),cv2.FONT_HERSHEY_PLAIN, 3, (154,51,60), 3)
            # Check move forward or backward
            if center_end[1] < center[1]:

```

```

        move_direction = "forward"
    else:
        move_direction = "backward"

    turn_commmmand, run_command =
convert_to_MQTT(dist, angle_1, move_direction)

    if fingers1[1:] == [0,0,0,0] and
landmarks_list1[4][2] < landmarks_list1[3][2]:
        send_check = True

    # Precision mode command
    if send_check:
        send_check = False
        draw_start_check, draw_line_check,
draw_end_check= 0,0,0
        #Publish message
        client.publish("car/mnt", turn_commmmand)
        time.sleep(1)
        client.publish("car/mnt", run_command)
        print("sent")

    #Resize
    cv2.namedWindow('Image', cv2.WINDOW_NORMAL)
    cv2.resizeWindow("Image", 1280, 720)

    cTime = time.time()
    fps = 1/(cTime-pTime)
    pTime = cTime

    time_print = "FPS: " + str(int(fps))

    #Show word
    cv2.putText(img, str(time_print), (950, 50),
cv2.FONT_HERSHEY_PLAIN, 3, (154, 51, 60), 3)

    #Show img
    cv2.putText(img, str(operate_mode),
(20, 50), cv2.FONT_HERSHEY_PLAIN, 3, (154, 51, 60), 3)
    cv2.imshow("Image", img)
    if cv2.waitKey(10) & 0xFF == ord('q'):
#        cap.release()
        cv2.destroyAllWindows()
        client.loop_stop()
        break

```

Appendix D: Coding for WiFi Car using Arduino IDE

```

#include <Robojax_L298N_DC_motor.h>
// MQTT library setup
#include <WiFi.h>
#include <PubSubClient.h>
#include "esp_wifi.h"

// motor 1 settings
#define CHA 0
#define ENA 14 // this pin must be PWM enabled pin
if Arduino board is used
#define IN1 27
#define IN2 26

// motor 2 settings
#define CHB 1
#define ENB 32 // this pin must be PWM enabled pin
if Arduino board is used
#define IN3 33
#define IN4 25

// WiFi setup
const char *ssid = "Wifi_1234-TIME2.4Ghz"; // Enter
your WiFi name
const char *password = "40502926"; // Enter WiFi
password

// MQTT Broker setup
const char *mqtt_broker = "192.168.0.107";
const char *mqtt_username = "ESP32_Yo";
const char *mqtt_password = "public";
const int mqtt_port = 1883;
long lastReconnectAttempt = 0;

//Topic to subscribe
const char *topic = "car/mode";
const char *topic2 = "car/move";
const char *topic3 = "car/mnt";

WiFiClient espClient;
PubSubClient client(espClient);

const int CCW = 2; // do not change
const int CW = 1; // do not change

#define motor1 1 // do not change
#define motor2 2 // do not change

```

```

//Initialize the move_state
String move_state = "Original";
String car_mode = "None";
unsigned long previousMillis = 0;
unsigned long interval = 8000;
//Move and time variable
unsigned long move_interval = 0;
unsigned long previousMillis2 = 0;
//Drive mode (left, right, high)
int left = 0;
int right = 0;
int high = 0;

// for two motors without debug information // Watch
video          instruciton          for          this          line:
https://youtu.be/2JTMqURJTwg
Robojax_L298N_DC_motor  robot(IN1,  IN2,  ENA,  CHA,
IN3,  IN4,  ENB,  CHB);

//MQTT callback function
void callback(char *topic, byte *payload, unsigned
int length) {

    char messageBuffer[30];
    memcpy(messageBuffer, payload, length); //copy
the payload to a buffer
    messageBuffer[length] = '\0'; //terminate with
a zero to convert to a C style string

    if (strcmp(topic, "car/mode") == 0)
    {
        Serial.println(messageBuffer);
        car_mode = messageBuffer;
    }
    else if (strcmp(topic, "car/move") == 0)
    {
        if (payload[0] == '1'){
            move_state = "forw";
        }
        else if (payload[0] == '2'){
            move_state = "backw";
        }
        else{
            move_state = "brake";
        }

        Serial.print("New move_state is :");
        Serial.println(move_state);

        // To check left right and high

```

```
    if (payload[1] == '1'){
        high = 10;
    }
    else{
        high = 0;
    }
    if (payload[2] == '1'){
        left = 10;
    }
    else if (payload[2] == '2'){
        right = 10;
    }
    else{
        left = 0;
        right = 0;
    }
}
else if (strcmp(topic, "car/mnt") == 0)
{
    //Record the time interval
    char smallbuffer[5];
    smallbuffer[0] = payload[2];
    smallbuffer[1] = payload[3];
    smallbuffer[2] = payload[4];
    smallbuffer[3] = payload[5];
    smallbuffer[4] = '\0';

    move_interval = atoi(smallbuffer);
    Serial.println(move_interval);

    //Set pMillis to cMillis so timer start from 0
    long cMillis = millis();
    previousMillis2 = cMillis;

    //Set 2nd digit as move state
    if (payload[1] == '1')
    {
        move_state = "forward";
    }
    else if (payload[1] == '2')
    {
        move_state = "backward";
    }
    else if (payload[1] == '3')
    {
        move_state = "left";
    }
    else if (payload[1] == '4')
    {
```

```

        move_state = "right";
    }
}

//MQTT reconnect function
boolean reconnect() {
    //what is "arduinoClient"
    if (client.connect("arduinoClient")) {
        // Once reconnected, print
        Serial.print("Reconnected");
        // Once connected, publish an announcement...
        client.publish(topic,"hello world");
        // ... and resubscribe
        client.subscribe(topic);
        client.subscribe(topic2);
        client.subscribe(topic3);
    }
    return client.connected();
}

void motormove(String &move_state){
    //Move based on time
    if (move_interval)
    {
        long now2 = millis();
        if (now2 - previousMillis2 >= move_interval)
        {
            move_state = "None";
            move_interval = 0;
        }
    }

    //Motor moving state (forward, backward, left,
    right)
    if (move_state == "forward")
    {
        robot.rotate(motor1, 80, CW); //run motor1 at 60%
        speed in CW direction
        robot.rotate(motor2, 80, CW); //run motor1 at 60%
        speed in CW direction
    }
    else if (move_state == "backward")
    {
        robot.rotate(motor1, 80, CCW); //run motor1 at
        60% speed in CCW direction
        robot.rotate(motor2, 80, CCW); //run motor1 at
        60% speed in CCW direction
    }
    else if (move_state == "left")

```



```

    {
        robot.rotate(motor1, 80, CCW); //run motor1 at
60% speed in CCW direction
        robot.rotate(motor2, 80, CW); //run motor1 at 60%
speed in CW direction
    }
    else if (move_state == "right")
    {
        robot.rotate(motor1, 80, CW); //run motor1 at 60%
speed in CW direction
        robot.rotate(motor2, 80, CCW); //run motor1 at
60% speed in CCW direction
    }
    else if (move_state == "forw")
    {
        robot.rotate(motor1, 80+left+high, CW); //run
motor1 at 60% speed in CW direction
        robot.rotate(motor2, 80+right+high, CW); //run
motor1 at 60% speed in CCW direction
    }
    else if (move_state == "backw")
    {
        robot.rotate(motor1, 80+left+high, CCW); //run
motor1 at 60% speed in CW direction
        robot.rotate(motor2, 80+right+high, CCW); //run
motor1 at 60% speed in CCW direction
    }
    else
    {
        robot.brake(1); //motor1 stop
        robot.brake(2); //motor2 stop
    }

    if (move_state == "for")
    {
        robot.rotate(motor1, 80+left+high, CW); //run
motor1 at 60% speed in CW direction
        robot.rotate(motor2, 80+right+high, CW); //run
motor1 at 60% speed in CCW direction
    }
}

void setup() {
    // Initialize motor
    Serial.begin(115200);
    robot.begin();

    //Turn Off Power Saving Mode

```

```

esp_wifi_set_ps(WIFI_PS_NONE);

//Connect WiFi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
    unsigned long currentMillis = millis();
    if ((WiFi.status() != WL_CONNECTED) &&
(currentMillis - previousMillis >=interval))
    {
        WiFi.disconnect();
        WiFi.begin(ssid, password);
        Serial.println("Retry Connecting to WiFi..");
        previousMillis = currentMillis;
    }
}

//Setup MQTT client and callback
client.setServer(mqtt_broker, mqtt_port);
client.setCallback(callback);

//Error Check
while (!client.connected()) {
    String client_id = "esp32-client-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the
public mqtt broker\n", client_id.c_str());
    if (client.connect(client_id.c_str(),
mqtt_username, mqtt_password)) {
        Serial.println("Public mqtt broker
connected");
    } else {
        Serial.print("failed with state ");
        Serial.print(client.state());
        delay(2000);
    }
}

//Subscribe Topic
client.subscribe(topic);
client.subscribe(topic2);
client.subscribe(topic3);
}

void loop() {
    if (!client.connected()) {
        long now = millis();
        if (now - lastReconnectAttempt > 5000) {
            lastReconnectAttempt = now;

```

```
        // Attempt to reconnect
        if (reconnect()) {
            lastReconnectAttempt = 0;
        }
    }
}
else{
    //Call function to move motor
    motormove(move_state);
    //Loop MQTT to receive message
    client.loop();
}
}
```