

**MASTER-CONTROLLED NETWORKING FOR MOBILE ROBOTICS
APPLICATION**

By
LEE CHIN BIN

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMMUNICATIONS
AND NETWORKING
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2024

REPORT STATUS DECLARATION FORM

Title: MASTER-CONTROLLED NETWORKING FOR
MOBILE ROBOTICS APPLICATION

Academic Session: JUNE 2024

I LEE CHIN BIN
(CAPITAL LETTER)

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

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

Verified by



(Author's signature)



(Supervisor's signature)

Address:

NO.5,Jalan Ampang Jaya
Taman Ampang Jaya
83000 Batu Pahat ,Johor

Goh Hock Guan
Supervisor's name

Date: 12/09/2024

Date: 12/09/2024

DECLARATION OF ORIGINALITY

I declare that this report entitled “**MASTER-CONTROLLED NETWORKING FOR MOBILE ROBOTICS APPLICATION**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.



Signature : _____

Name : LEE CHIN BIN

Date : 12 September 2024

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Ts Dr. Goh Hock Guan who has given me this bright opportunity to engage in an IoT robotic application project. It is my first step to establish a career in IoT robotic field. A million thanks to you.

I also feel appreciated to the entirety of the Department faculty members especially lab assistants for their assistance. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

The goal of this project is to establish a master-controlled networking system for mobile robotics applications. It addresses problems and offers methods, hardware, and software solutions.

This project will be built on top of the GoPiGo3 platform, which has many robots set up as slaves and one robot designated as the master. The slave robots will be outfitted with object storage, while the master robot will get changes, including the installation of an arm. These changes are mostly intended to simulate a warehouse environment, in which items are dispersed and not particularly arranged. The GoPiGo3 robots will link their arms and sensors to enable communication and cooperation.

Dedicated mobile apps will be available for both the master and slave robots. These apps will come with a range of functionality, including as arm control, camera operations, and robot navigation, all of which are intended to provide users a thorough grasp of the robot's capabilities and enable easier control. Java will be used in Android Studio to construct the mobile apps.

Local area network (LAN) and master-slave communication will be used in tandem to facilitate communication between the master and slave robots. After processing orders or data, the master robot will provide the required information to the slave robots. In this situation, when communication is necessary for coordinated actions and reactions, the master might be compared to a commander and the slaves to troops.

TABLE OF CONTENTS

TITLE PAGE	i
DECLARATION OF ORIGINALITY	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	viii
LIST OF TABLES	xi
LIST OF ABBREVIATIONS	x
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement and Motivation	1 - 2
1.2 Objectives	2 - 4
1.3 Project Scope and Direction	4
1.4 Contributions	5
1.5 Report Organization	6
CHAPTER 2 LITERATURE REVIEW	7
2.1 Review of the Technologies	7 - 12
2.1.1 Hardware platform	7
2.1.1.1Raspbeery Pi	7
2.1.2 Firmware /OS	8
2.1.3 Programming language	9 - 10
2.1.3.1 Java	9
2.1.3.2 Python	10
2.1.4 programming paradigm	11
2.1.5 Summary of Review of the Technologies	12

2.2 Review of the Existing Systems/Applications	13 - 21
2.2.1 Master-Slave Robot Communication	13 - 14
2.2.2 Local Area Network	15 - 16
2.2.3 Multi-Robot Communication System	17 - 18
2.2.4 Mobile application	19 - 20
2.2.5 Summary of Review of the Existing Systems	21
CHAPTER 3 PROPOSED METHOD/APPROACH	22
3.1 System Development Models	22 - 26
3.1.1 Waterfall Model	22
3.1.2 Agile Model	23
3.1.3 Incremental Model	24
3.1.4 V-Model	25
3.1.5 Selected Model	26
3.2 System Requirement	27 - 29
3.2.1 Hardware	27 - 30
3.2.1.1 Gopigo3 robot	27
3.2.1.2 GoPiGo3 Electronic Board	28
3.2.1.3 Raspberry pi 3B+	28
3.2.1.4 Arm Gripper	29
3.2.1.5 Servo MG996R	29
3.2.1.6 Metal Servo Motor Hub	30
3.2.1.7 Servo Board PCA9685	30
3.2.2 Software	31-32
3.2.2.1 Android studio	31
3.2.2.2 Geany	31
3.2.2.3 Real VNC Viewer	32
3.2.2.4 MobaXterm	32
3.3 Project Milestone	33-35
3.3.1 FYP1 Timetable and Milestone	33
3.3.2 FYP2 Timetable and Milestone	34
3.4 Estimated Cost	35

CHAPTER 4 PRELIMINARY WORK	36
4.1 System Architecture	36-37
4.1.1 Hardware Architecture	36
4.1.2 Network Architecture	37
4.2 Functional Modules in the System	38
4.3 System Flow	39-50
4.3.1 Main Master	39
4.3.2 Master Server Thread	39
4.3.2.1 Master Send Thread	40
4.3.2.2 Master Received Thread	41
4.3.3 Middleware Thread (Master)	42
4.3.3.1 Server Send Thread (Master)	42
4.3.3.2 Sever Receive Thread	43
4.3.4 Ultrasound Thread (Master)	44
4.3.5 Master Status Thread	44
4.3.6 Slave Main	45
4.3.7 Slave Client Thread	45
4.3.7.1 Slave Send Thread	46
4.3.7.2 Slave Receive Thread	47
4.3.8 Ultrasound (Slave)	48
4.3.9 Slave Status Thread	48
4.3.10 Mobile application TCP	49
4.3.11 Middleware server	50
4.4 GUI Design	51-52

CHAPTER 5: System Implementation	53
535.1 Hardware Setup	53-
5.1.1 GoPiGo3 Default Setup	53
5.1.2 Master GoPiGo3	53
535.1.2.1 Servo MG996R and Robot Gripper	53
545.1.2.2 Servo MG996R and Bracket	54
5.1.2.3 USB Network Adapter	54
5.1.2.4 Servo Board PCA9685 with Servo MG996R, Raspberry Pi, and Battery Casing	55
5.1.3.1 Servo MG996R with 3D Printer Objects, and Battery Casing	56
5.2 Software Setup	57
5.2.1 Raspberry Pi Customize Operating System Install	57
5.2.2 Master Modified GoPiGo3 System	58
5.2.3 Slave Modified GoPiGo3 System	59
5.2.4 Middleware Server installation	59
5.3 Setting and Configuration	60
5.3.1 Middleware Server code	60
5.3.2 Master Main	61
5.3.3 Master listen slave connect	61
5.3.3.1 Master receive from slave	62
5.3.3.2 Master send to slave	63 - 64
5.3.4 Master connect middleware server socket code	65
5.3.4.1 Master send to middleware server	65
5.3.4.2 Master receive form middleware server	66
5.3.5 Master Servo board PCA9685	67
5.3.6 Master Arm code	68
5.3.7 Master Movement code	69
5.3.8 Slave main script	70
5.3.9 Slave connect to Master	70
5.3.9.1 Slave receive form Master	71
5.3.9.2 Slave send to Maste	72

5.3.10 Slave servo	72
5.3.11 Slave movement	73
5.3.12 Slave arm code	74
5.3.13 Master and Slave Status	74
5.3.14 Master and Slave Ultrasound	75
5.3.15 Mobile receives from middleware	76
5.3.16 Mobile Send to middleware	77
5.3.17 Status	78
5.3.18 Send part in any page	79
5.4 System Operation	80
5.4.1 Master and Slave : GoPiGo3 OS	80
5.4.2 Server : Ubuntu 22.04.4 LTS	80
5.5 Concluding Remark	81
Chapter 6 - System Evaluation and Discussion	82
6.1 System Testing and Performance Metrics	82 - 85
6.2 Testing Setup and Res	86 - 95
6.2.1 Taking Task	86 - 87
6.2.2 Movement Together	88 - 89
6.2.3 Avoid Obstacles	90
6.2.4 Alone Movement	91
6.2.5 Arm Control	92
6.2.6 Follow function	93
6.2.7 Status of connect master and server	94
6.2.8 Status of master and slave	94
6.2.9 Status try to connect to server	95

6.3 Project Challenges	96 - 99
6.3.1 Customize Operating System	96
6.3.1.1 Problem	96
6.3.1.2 Solution	97
6.3.2 Hardware Problem and Balancing Problem	98
6.3.2.1 Problem	98
6.3.2.2 Solution	98
6.3.3 Environment Network	99
6.3.3.1 Problem	99
6.3.3.2 Solution	99
6.4 Objectives Evaluation	100-101
6.5 Concluding Remark	102
CHAPTER 7 CONCLUSION AND RECOMMENDATION	103
7.1 Conclusion	103 - 104
7.2 Recommendation	105
REFERENCES	106 - 107
APPENDIX A	
A.1 Weekly Report	A-1
A.2 Poster	A-8
PLAGIARISM CHECK RESULT	
CHECK LISTS	

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1.1.1	Raspberry 3B+.	7
Figure 2.1.2	Graphical user interface of GoPiGo3.	8
Figure 2.1.3.1	Java Logo	9
Figure 2.1.3.2	Python Logo	10
Figure 2.1.4.1	TCP Server and client	11
Figure 2.1.4.2	UDP Server and client	11
Figure 2.2.1	GOTO and FOLLOW algorithms for a master–slave robot system.	13
Figure 2.2.2	Conclusion of the Bluetooth, Wi-Fi and wireless Lan and 3G and concept with WLAN.	15
Figure 2.2.3.1	Communication of Robot Server and Client	17
Figure 2.2.3.2	TCP Server and Client	17
Figure 2.2.4.1	User interface	19
Figure 2.2.4.2	Mobile application in running mode	19
Figure 2.2.4.3	Motor speed for different positions of trackball	19
Figure 3.1.1	Waterfall Degree	22
Figure 3.1.2	Agile Model Degree	23
Figure 3.1.3	Incremental Model Degree	24
Figure 3.1.4	V-Model Degree	25
Figure 3.1.5	Incremental Model of the project	26
Figure 3.2.1.1	Normally Gopigo3 robot	27
Figure 3.2.1.2	GoPiGo3 Electronic Board	28
Figure 3.2.1.3.1	Raspberry Pi 3B+	28
Figure 3.2.1.3.2	40 Extend GPIO	28
Figure 3.2.1.4	Robot Arm Gripper	29
Figure 3.2.1.5	Servo MG996R	29
Figure 3.2.1.6	Metal Servo Motor Hub (MG996R)	30
Figure 3.2.1.7	Servo Board (PCA9685)	30
Figure 3.2.2.1	Android studio Logo	31

Figure 3.2.2.2	Geany Logo	31
Figure 3.2.2.2	Real VNC Viewer Logo	32
Figure 3.2.2.2	MobaXterm Logo	32
Figure 3.3.1	FYP1 Timetable and Milestone	33
Figure 3.3.2	FYP2 Timetable and Milestone	34
Figure 4.1.1.1	GoPiGo 3 hardware architectures	36
Figure 4.1.1.2	GoPiGo3 Hardware Architectures	36
Figure 4.1.2	FYP2 Network Architecture	37
Figure 4.3.	Main Master Flow Chart	39
Figure 4.3.2:	Master Server Thread Flow Char	39
Figure 4.3.2.1	Master Send Thread Flow Chart	40
Figure 4.3.2.2	Master Received Thread Flow Chart	41
Figure 4.3.3	Middleware Thread Flow Chart	42
Figure 4.3.3.1	Server Send Thread Flow Chart	42
Figure 4.3.3.2	Sever Receive Thread Flow Chart	43
Figure 4.3.4	Ultrasound Thread Flow Chart	44
Figure 4.3.5	Master Status Thread Flow Chart	44
Figure 4.3.6	Slave Main Flow Chart	45
45Figure 4.3.7	Slave Client Thread Flow Chart	45
46Figure 4.3.7.1	Slave Send Thread Flow Chart	46
47Figure 4.3.7.2	Slave Receive Thread Flow Chart	47
48Figure 4.3.8	Ultrasound Thread Flow Chart	48
48Figure 4.3.9	Slave Status Thread Flow Chart	48
49Figure 4.3.10	Mobile Application TCP Flow Chart	49
Figure 4.3.11	Middleware Server Flow Chart	50
Figure 4.4.1	Master page	51
Figure 4.4.2	Slave page	51
Figure 4.4.3	Arm page	51
Figure 4.4.4	Status Page	51
Figure 4.4.5	Settings Page	52
Figure 5.1.2.1	Servo MG996R and Robot Gripper	53
Figure 5.1.2.2	Servo MG996R and Bracket	54

Figure 5.1.2.4	Servo Board PCA9685 with Servo MG996R, Raspberry Pi, and Battery Casing	55
Figure 5.1.3.1	Servo MG996R with 3D Printer Objects, and Battery Casing	56
Figure 5.2.1	Raspberry Pi Connect	57
Figure 5.2.2.1.1	wpa_supplicant.conf	58
Figure 5.2.2.1.2	cinch.conf	58
Figure 5.2.3.1	wpa_supplicant.conf	59
Figure 5.2.3.2	wpa_supplicant.conf	59
Figure 5.3.1	Middleware Server code	60
Figure 5.3.3	Master Listen Slave Connect code	61
Figure 5.3.2	Master Main code	61
Figure 5.3.3.1	Master Receive From Slave code	62
Figure 5.3.3.2.1	Master Send to Slave code	63
Figure 5.3.3.2.2	Process_Client_Message 1code	63
Figure 5.3.4.1	send_data code	65
Figure 5.3.4	connect_server code	65
Figure 5.3.4.2	receive_data code	65
Figure 5.3.5	Master Servo board PCA9685 code	67
Figure 5.3.6.1	arm1functios code	68
Figure 5.3.6.2	arm code	68
Figure 5.3.7	Master Movement code	69
Figure 5.3.8	Slave Main code	70
Figure 5.3.9	connect_server code	70
Figure 5.3.9.1	receive_thread code	71
Figure 5.3.9.2	send_thread. code	72
Figure 5.3.10	Slave Sevo. Code	72
Figure 5.3.11	Salve movement code	73
Figure 5.3.12	Salve armmovement code	74
Figure 5.3.12	Master and Slave Status Thread. code	74
Figure 5.3. 14	Master and Slave Ultrasound.code	75
Figure 5.3.15	Mobile Receive from Middleware.code	76
Figure 5.3.16.	Mobile Send to Middleware.code	77

Figure 5.3.18.2	putthemessage code	78
Figure 5.3.17.2	updateCpmmectopmStatus code	78
Figure 5.3.18.1	Send in Any Page code	79
Figure 5.3.18.2	putthemessage code	79
Figure 5.4.1	GoPiGo3 OS	80
Figure 5.4.2	Ubuntu 22.04.4 LTS OS	80
Figure 6.2.1.1	Mobile Master Page	86
Figure 6.2.1.2	Mobile Tasking Task Part1	86
Figure 6.2.1.3	Mobile Tasking Task Part2	87
Figure 6.2.1.4	Mobile Tasking Task Part3	87
Figure 6.2.2.1	Mobile Master Page	88
Figure 6.2.2.2	Movement Together Part1	88
Figure 6.2.2.3	Movement Together Part2	89
Figure 6.2.3	Avoid Obstacles	90
Figure 6.2.4.1	Mobile Master Page	91
Figure 6.2.4.2	Alone Movement	91
Figure 6.2.5.1	Mobile ARM Page	92
Figure 6.2.5.2	Arm Control	92
Figure 6.2.6.1	Mobile Slave Page	93
Figure 6.2.6.2	Follow Function	93
Figure 6.2.7	Mobile Master Page	94
Figure 6.2.8	Mobile Status Page	94
Figure 6.2.9	Mobile Settings Page	94

LIST OF TABLES

Table Number	Title	Page
Table 2.1.5	Summary of Review of the Technologies table	17
Table 2.2.5	Summary of Review of the Existing Systems table	21
Table 3.2.1.2	GoPiGo3 Electronic Board Description table	28
Table 3.2.1.3	Raspberry pi 3B+ Description table	28
Table 3.2.1.4	Arm Gripper Description table	29
Table 3.2.1.5	Servo MG996R table	29
Table 3.2.1.6	Metal Servo Motor Hub table	30
Table 3.2.1.7	Servo Board PCA9685 table	30
Table 3.4	Estimated Cost table	35
Table 6.1	Testing and Output of FYP2	87
Table 6.4	Objectives and Evaluation	100-101

LIST OF ABBREVIATIONS

<i>SLAM</i>	Complementary Metal Oxide Semiconductor
<i>AMRs</i>	Autonomous Mobile Robots
<i>AGCs</i>	Automated Guided Carts
<i>CNC</i>	Computer Numerical Control
<i>3D</i>	Three-Dimensional
<i>SMART</i>	Specific, Measurable, Achievable, Relevant, and Time-bound
<i>LAN</i>	Local Area Network
<i>AGAs</i>	Articulated Robotic Arms
<i>GUI</i>	Graphical User Interface
<i>UI</i>	User Interface
<i>FYP</i>	Final Year Report
<i>GPIOs</i>	General Purpose Input/Output
<i>LEDs</i>	Light-emitting diode
<i>CPU</i>	Central Processing Unit
<i>LPDDR</i>	Low-Power Double-Data Rate Memory
<i>GHz</i>	Gigahertz
<i>HDMI</i>	High-Definition Multimedia Interface
<i>OS</i>	Operating System
<i>JVM</i>	Java Virtual Machine
<i>TCP</i>	Transmission Control Protocol
<i>UDP</i>	User Datagram Protocol
<i>IP</i>	Internet Protocol
<i>WiFi</i>	Wireless Fidelity
<i>WLAN</i>	Wireless Local Area Network
<i>Mbps</i>	Megabits per second
<i>R2R</i>	Robot To Robot
<i>SDK</i>	Software Development Kit
<i>DHCP</i>	Dynamic Host Configuration Protocol
<i>VPS</i>	Virtual Private Network

CHAPTER 1

Introduction

1.1 Problem Statement and Motivation

Everything must first have a problem before it can have a solution. To explain the issue, use the problem statement. There are several issues with the robot business, including the one-to-one control model of the robot system, intrinsic quirks, the limited flexibility of robotic systems in modern warehouse operations, and the lack of remote-control capabilities in certain robot kinds.

The robot system's **one-to-one control paradigm**, in which each robot functions alone without connection to other robots of the same kind, is the initial problem statement. These robots have sophisticated navigation systems that use Path Planning, Odometry, SLAM, and Localization, however they have coordination issues [1]. This design restriction leads to greater mistake rates and inefficiency in practice. For example, delays occur when a consumer submits an order that requires more work than a single robot can do. When trying to do several jobs at once, these robots operate as stand-alone systems, which makes it difficult for them to cooperate and communicate. This causes disputes and ineffective navigation. One prominent problem, for instance, is when two robots try to bring meals at the same time. In this case, the first robot may arrive to the intended delivery location, and the second robot would proceed in the same direction without communicating with the first to let it know it is there. As a result, there are inefficiencies and even service interruptions when the second robot runs into a dispute while attempting to manoeuvre around the first robot. This issue emphasises how crucial it is for the robot waiters to coordinate and communicate better in order to maximise productivity and reduce these kinds of confrontations. Project **need to devise a way for these robots to cooperatively communicate information** throughout operations to maximise system efficacy and reduce mistakes.

CHAPTER 1

The second issue statement relates to the **limited flexibility and intrinsic idiosyncrasies of robotic systems in modern warehouse operations**. A variety of robotic systems are used in warehouses, such as Autonomous Mobile Robots (AMRs), Articulated Robotic Arms, and Automated Guided Carts (AGCs). Although both of these technologies increase productivity and lower labour costs, they each offer unique advantages and disadvantages. While AGCs excel at moving objects along pre-established paths, they are not able to manipulate objects independently. Articulated robotic arms are adept at manipulating items despite their immobility. Autonomous mobile robots (AMRs) can travel and operate items with robotic arms thanks to advanced mapping and sensing technologies[2]. However, articulated manipulators and AGC are inferior to AMR. But while some AMRs lack robotic arms and favour storage over robotic arms for item manipulation, others do the opposite. This diversity highlights the challenges associated with **developing comprehensive solutions** for the warehousing industry.

The final issue statement describes how **certain robots are not capable of being controlled remotely**. This includes ways to enter data such as gaming controllers, wireless keyboard and mouse interfaces, apps for smartphones, and similar devices. While remote-control robots dominate the market, other types of robots, such as CNC machines, 3D printers, agricultural robots, and so forth, lack these convenient features. It's critical to realise that remote control encompasses more than just navigation; it also contains vital functions like **power management, real-time status monitoring, access to robot data**, and more. As such, it is an essential tool for improving robot usability and flexibility. The operation, supervision, and administration of these robots may be hampered by their lack of remote control and monitoring capabilities, especially in situations where **real-time or remote modifications are crucial**. Furthermore, robots designed to do single, specialised jobs could not be as versatile as others, which would limit their use to a smaller number of sectors and activities and, eventually, reduce their total usefulness and flexibility.

1.2 Research Objectives

Following the SMART criteria (Specific, Measurable, Achievable, Relevant, and Time-bound), this project contains three well-defined goals that make monitoring and assessment easier during the project's lifespan. First and foremost, the project seeks to create a Master Robot that can effectively manage a number of Slave Robots. The creation of a Master Robot and many Slave Robots that can work together to complete assigned tasks is the second project goal. Finally, the project aims to create a mobile application that allows for remote surveillance and control over the actions of the Master Robot.

The project's primary goal is to create a **Master Robot that can effectively manage a few Slave Robots**. This novel approach is expected to improve controllability and efficacy in a few sectors. The project increases efficiency in managing many robots by establishing a network-based communication mechanism between the master and slaves and centralizing control with the master robot. Because of this automation, users won't need to intervene manually as often. The project estimates that it will take around eight months to develop this idea to completion.

Secondly, the aim of the project is to develop a **Master Robot and many Slave Robots** that can work together to accomplish **certain tasks** and effectively complete jobs in various industrial domains. To accomplish the goals of our project, these robots will be constructed using state-of-the-art parts such as motors, sensors, and arms. The initiative aims to lower operating costs and time requirements while concurrently addressing important industry concerns.

The last goal of the project is to create a **mobile application that allows for remote management** and oversight of the **Master Robot's functions**. This software will provide consumers insightful insights while smoothly managing the robot's navigation. In order to increase the flexibility and realism of this mobile application, the project plan calls for using certain programming languages and tools. This will improve the user experience and the robot's adaptability by allowing users to operate the robot remotely from any location.

1.3 Project Scope and Direction

A report's project scope is essential since it clarifies the duties, pursuits, products, and objectives that the project will tackle. The project's scopes include a few important goals: First, establishing a local area network (LAN) for master GoPiGo3 robots to use in order to connect with slave GoPiGo3 robots. Second, enabling master and slave GoPiGo3 robot control via a mobile device. Finally, navigation and task execution are made possible by master and slave GoPiGo3 robots.

First, **LAN communication between master and slave GoPiGo3** robots is enabled. The central processing unit is the master GoPiGo3 robot, which gathers and processes data sent by users. The master robot then sends the appropriate commands to the slave robots based on its judgements. The slave robots quickly carry out preprogrammed tasks after obtaining these signals. The utilisation of a LAN is necessary for reliable and effective data sharing across this complete communication structure.

Second, enabling control of the **GoPiGo3 master and slave robots via a mobile device**. The master robot and the mobile application will be facilitated through a middleware server hosted on a public server to communication between master and mobile . Users will manage robot navigation and arm movements through a smartphone application created using Android Studio. Users can choose which robot to operate, with the master robot being the default option. The functionality of the master robot's page will be more advanced than that of the slave robot's page.

Lastly, **task execution** is coordinated by the GoPiGo3 master-slave robot system. The slave robot trails the master robot to the object's destination. Once the arm picks up the object, the slave robot moves in to pick it up. Master and Slave GoPiGo3 robots will follow a line and adhere to mobile instructions throughout the process. If there are remaining duties, the process will be repeated in the second stage.

1.4 Contributions

The term "contribution" describes the activities, endeavors, or assets that people or groups have committed to furthering the goals of our project, which centers on the creation of a mobile application for robot control, adaptive robot design, and networked master-slave robot communication.

Master-Slave Robot Communication:

Our project incorporates a novel technique for robot-slave network communication. This method allows humans to effortlessly command many robots by designating one as the "master" and the rest as "slaves." Like a military unit, this concept uses the slaves as soldiers and the owner as the commander. To overcome obstacles and complete the predefined objectives, the slave robots follow the master robot's instructions and, when necessary, make autonomous decisions. This approach aims to address and resolve the challenge posed by the robot system's one-to-one control paradigm.

Adaptive Robot Design:

Our robot has a flexible design that overcomes the drawbacks of fixed robotic systems like articulated robotic arms (AGAs). The robot can swap its parts, especially its manipulating arm, which is one of its standout features. Because of its versatility, the robot may be tailored to the specific requirements of many sectors. Furthermore, our design guarantees simplicity of maintenance, which is a major advantage over existing robotic systems, by enabling the replacement of specific elements without the need to dismantle the complete robot.

Mobile Application for Robot Control:

We have created a specific mobile application. This software provides several essential functions while resolving the issues with remote control. A user-friendly graphical user interface (GUI) allows users to navigate and send orders to the robot, as well as see live video feeds from it. Users may easily operate the robot from any place using this application.

1.5 Report Organization

The Final Year Project 2 has a total of seven chapters to introduce the project's details. Chapter 1 introduces a brief background, problem statement, objectives, and scope of the project. Chapters 2, 5, and 4 are research on the project to help the project understand the knowledge in this field. Chapter 3: Introduce project used hardware, software, protocol design, timeline and estimated cost. Chapter 4 shows system design of Final Year Project 2 like flowchart, UI, network structure, and others. Chapter 5 introduces the hardware, software setup, and configuration of the system. Chapter 6 introduces the testing, result and objective evaluation. Chapter 7 is the simple conclusion of the whole Final Year Project 2 project and some recommendations for the future.

CHAPTER 2

Literature Reviews

2.1 Review of the Technologies

2.1.1 Hardware platform

2.1.1.1 Raspberry Pi

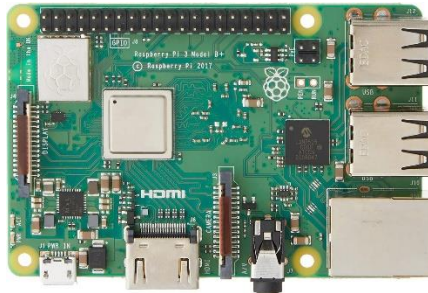


Figure 2.1.1.1: Raspberry Pi 3B+

The Raspberry Pi is a family of single-board computers. What is a Raspberry Pi? This is because it was created by the Raspberry Pi Foundation, a British charity that aims to educate people about computing and create greater access to computing education [3]. The Raspberry Pi is a great piece of hardware for programming, building hardware projects, or even using it in industrial applications. It is a cheap computer that runs the Linux operating system and provides a set of GPIOs to control components such as ultrasonic sensors, servers, and LEDs. The Raspberry Pi lets beginners understand what the Internet of Things is.

This project uses a Raspberry Pi 3B+ as a master and slave robot. The Raspberry Pi 3 B+ is a serial of the Raspberry Pi; it is better than the Raspberry Pi Zero. Raspberry Pi Zero CPU 1GHz, 2.4GHz IEEE, memory 512MB LPDDR2, and another [4]. Compared with the Raspberry Pi 3B+ CPU 1.4GHz, which has 2.4 GHz and 5 GHz IEEE, 1 GB of memory, and another [5]. The Raspberry Pi 3B+ is more powerful than the Raspberry Pi Zero CPU (1 GHz). The GoPiGo3 electric board size is the same as the Raspberry Pi 3 B+.

2.1.2 Firmware /OS

GoPiGo OS

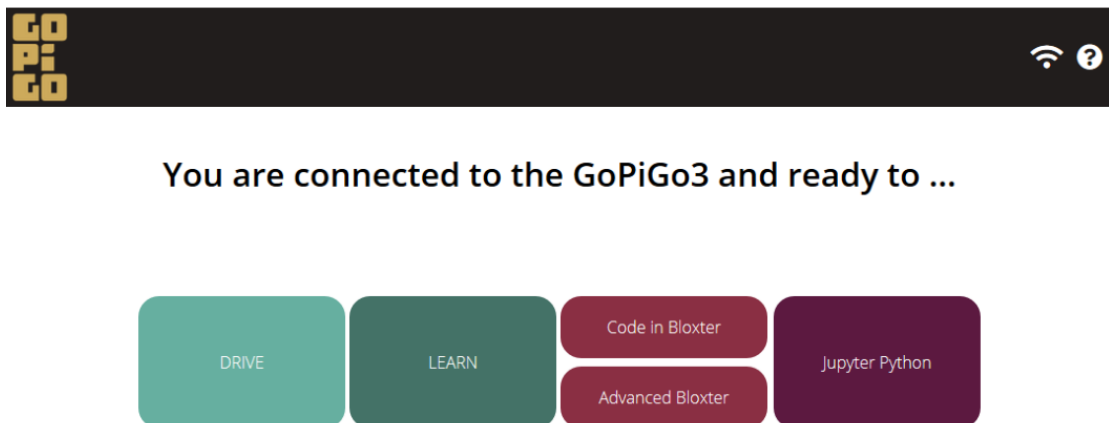


Figure 2.1.2: Graphical user interface of GoPiGo3

GoPiGo OS is the default operating system for GoPiGo robots. It is a graphical user interface that allows users to access different programming tools and experiment with the GoPiGo robot. GoPiGo OS is based on Raspberry OS development, and Raspberry Pi OS is based on Debian development. The tools of GoPiGo OS have basic movement, built-in lessons, Bloxter, and Python [6]. GoPiGo OS has its own hotspot and a special IP address to let users remotely control it. The user just connects to the Wi-Fi call GoPiGo by default and enters 10.10.10.10 at the browser, which will show the figure like 2.1.2. This shows the UI of GoPiGo specialists. If using HDMI to enter, you will see something like the Raspberry Pi OS, but with some GoPiGo movement tools.

In this project, the GoPiGo operating system was integrated into the master robot and slave robots to achieve the project goals. Python will be used for programming tasks on the GoPiGo operating system, making it ideal for robotics applications. In addition, the hotspot function establishes a wireless network between master and slave robots to achieve seamless communication and coordination between master and slave robots.

2.1.3 Programming language

2.1.3.1 Java



Figure 2.1.3.1: Java Logo

Java is the command-programming language used in the word. Java was first released by Sun Microsystems in 1995. Java is a class-based, object-oriented programming language. Java's main purpose is that developers just need to write it once and run it anywhere on platforms that support Java. Java is the most developed application for desktop, web, and mobile devices [7].

The main features of Java are platform independence, simplicity, dynamic flexibility, write-once run-anywhere, and other features. Platform-independent means you can write Java code on any platform, like Windows, Linux, or MacOS. This Java language uses a compiler to convert source code to bytecode, and then the JVM executes the bytecode generated by the compiler. Simple is also a character that develops love of use because Java does not have complex features like pointers, operator overloading, multiple inheritances, and others. Java even supports functions written in other languages, such as C and C++, which are referred to as native methods [7].

In this project, java language is developing mobile applications main language. Java is supported in Android Studio. It brings about the development of robust, scalable, and feature-rich Android apps. Java has a large and extensive library and community support to let this project develop more easily and with more functionality.

2.1.3.2 Python



Figure 2.1.3.2 : Python Logo

Python is also a command that uses a programming language. Python was created in 1980 by Guido Rossum. Now, the Python version has reached 3.13. Python is interpreted, object-oriented, and considered a programming language. Python is the easiest yet most useful programming language and is widely used in the software industry. Python is used in competitive programming, web development, creating software, and modern technological fields like data science, machine learning, and automation tasks [8].

Python is friendly to the beginner programmer. Python's major focus is making it easier for developers to read and understand, while also reducing the lines of code. Another easy language to read and understand, Python is versatile, extensible, and supports multi-platforms. Python is free and open source, so it has a huge and active community. These bring Python hundreds of libraries and frameworks to support development and programming [8].

In this project, Python is the primary language of the robot. Master robot and slave robot using Python language to run functions like reading ultrasound, movement, servo, and others. GoPiGo3 has its own Python library to support the user using the GoPiGo3 electric board. Python brings to this project a simple and fast development.

2.1.4 programming paradigm

Socket Programming

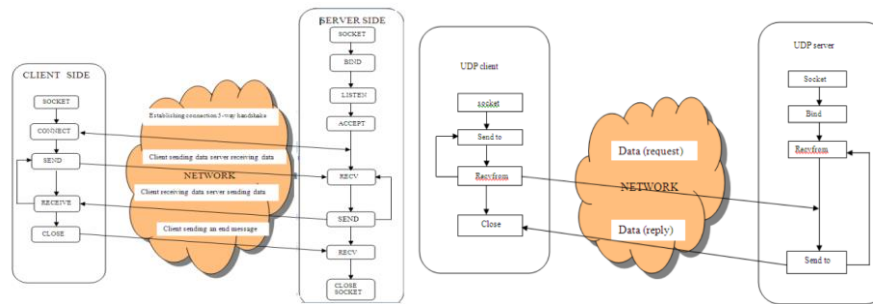


Figure 2.1.4.1 TCP Server and client

Figure 2.1.4.2 UDP Server and client

Socket Programming also can call Client-Server Communication because client and server are using socket to communication to each other .What is socket ?Socket Sockets defined as the endpoints of connection between two computers identified by an IP Address and a port number. In the socket programming import is port and IP address of server .Socket programming have 2 type TCP and UDP [9].

TCP socket programming important part is “Bind”, “Listen” and “Accept” at server socket . “Bind”, that is, fixed at a certain port number. and IP address, “Listen” to wait for incoming requests on the port, and “Accept” to accept connections from clients respectively. The client initiates a three-way handshake with the server and establishes a TCP connection with the server[9].Server and client can send or receive data from each other until socket closing .

UDP socket programming important part is “sendto” and “recvfrom” . UDP is a connection-less, datagram protocol. The client simply sends a datagram to the server using the “sendto” function, which requires the destination address as a parameter. Likewise, the server does not accept connections from clients. Instead, the server just calls the “recvfrom” function, which waits for data to arrive from some client. “recvfrom” returns the client's IP address and datagram so that the server can send a response to the client[9].

In this project will using TCP socket programming create communication master and mobile ,and master and slave communication . Using TCP((Transmission Control Protocol) stable and reliability to communication in project ,ensuring that data exchanges between devices are secure and error-free.

2.1.5 Summary of Review of the Technologies

Table 2.1.5: Summary of Review of the Technologies table

Technologies	Characteristics	Project used
Raspberry Pi	<ul style="list-style-type: none"> - Single-board computers - Cheap computer that runs the Linux operating system and provides a set of GPIOs to control components 	<ul style="list-style-type: none"> - Master and slave robot - Raspberry Pi 3 B+
GoPiGo OS	<ul style="list-style-type: none"> - Operating system for GoPiGo robots -Basic movement, built-in lessons, Bloxter, and Python -Hotspot and a special IP address 10.10.10.10 to let users remotely control 	<ul style="list-style-type: none"> - Master robot and slave robots OS -Hotspot -Python
Java	<ul style="list-style-type: none"> - Write-once run-anywhere - Independence, simplicity, dynamic flexibility 	<ul style="list-style-type: none"> - Developing mobile applications
Python	<ul style="list-style-type: none"> - Friendly to the beginner programmer - Easier for developers to read and understand, while also reducing the lines of code 	<ul style="list-style-type: none"> - Master robot and slave robot using Python language - . GoPiGo3 has its own Python library to support the user using the GoPiGo3 electric board
Socket Programming	<ul style="list-style-type: none"> - TCP socket programming - UDP socket programming 	<ul style="list-style-type: none"> - TCP socket programming creates communication master and mobile - TCP socket programming creates master and slave communication

2.2 Review of the Existing Systems/Applications

2.2.1 Master-Slave Robot Communication

The article is Development of a master–slave robot system for farm operations [10]. In the article had mention what motion control algorithm is used and test in like provide the algorithm is good . The motion control algorithms are GOTO and FOLLOW algorithm .Both is developed for use in a master–slave structure and local communication formats within a multi-robot system.

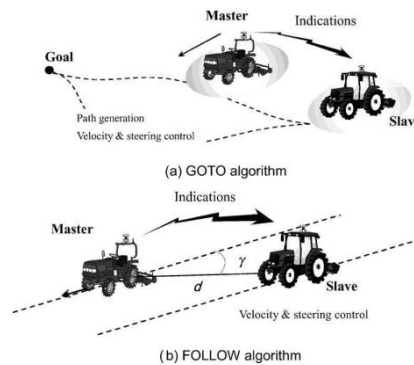


Figure 2.2.1: GOTO and FOLLOW algorithms for a master–slave robot system.

In a nutshell, the GOTO algorithm is what essentially coordinates navigation in a master-slave system, telling the slave where to go or how far to travel from where it is now. The problem of coordinating separate motions inside the master-slave configuration is addressed by this method. It has features that allow the slave to slow down in order to protect the master from harm and to dynamically change its course in order to avoid collisions. The main goal of the paper is to show how cooperative navigation methods may be used, especially when hay harvesting is involved.

A slave robot may follow a master robot closely at a predefined relative distance and angle thanks to the FOLLOW algorithm. Crucially, this relative placement doesn't alter regardless of how fast or which way the master travels. A nonlinear sliding mode controller is used into the algorithm to do this. In simulations, the slave robot independently modifies its speed and angle to maintain the predetermined relative position as the master robot moves and changes its speed. This algorithm's main goal is to encourage two identical machines to cooperate, increasing productivity by guaranteeing accurate and constant following behavior.

Limitation of Previous Studies

In this paper, efficient algorithms for follower behavior and navigation in a master-slave system are presented, along with possible uses. One significant shortcoming of the research, however, is that it only examined situations in which there were two identical machines—one acting as the master and the other as the slave. Although the methods provided here function well, it is not clear whether they can be expanded to support more than one slave robot. This topic remains unresolved since multi-slave situations have not been tested or discussed, which might restrict the algorithm's capacity to adapt to more complicated settings.

Moreover, scaling up to several slaves may provide difficulties since the algorithms mainly use distance and angle computations to operate the robots. The absence of user involvement in the control system is another issue. However, the autonomous nature of the algorithms could be a drawback in circumstances when human interaction or manual control is required, including addressing unforeseen situations or halting the identical machines. Remote control choices combined with autonomous algorithms might provide a versatile and well-balanced approach to robotic operations in a variety of scenarios, all while addressing these problems and improving the system's adaptability.

Proposed Solutions

Expanding and customising the GOTO and FOLLOW-inspired algorithms for usage with many slave robots, particularly on the Gopigo3 platform, is the main goal of the project. This experiment attempts to assess these algorithms' performance in multi-slave circumstances, such those that are often encountered in warehouse robots. By doing this, the research aims to extend these algorithms' initial field of use. The initiative prioritises human control and safety in addition to algorithmic improvements. A remote-control system will be designed and implemented as part of the project in order to manage unpredictable circumstances and guarantee safe robot operation. The danger of accidents may be reduced by giving people direct control over the robots, including the ability to perform an emergency stop when needed, thanks to this remote control. Additionally, the user-friendly interface of the remote-control system allows users to decide when to start or stop robot activities, giving them the freedom to manage the robots safely and efficiently.

2.2.2 Local Area Network

Wireless technologies such as Bluetooth, WiFi, wireless LAN, and 3G are introduced in the article "A Review of Wireless Technology Usage for Mobile Robot Controller" [11]. These technologies improve mobile robot controllers by allowing real-time monitoring, data transfer, and remote operation. The mobile robot controllers using this technology are more adaptable and appropriate for a wider range of robotics applications.

Types of Wireless Technology	Range	Frequency Band(s)	Data Rate
Bluetooth	33 feet	2.4Ghz	1.5Mbps
Wi-Fi IEEE802.11b IEEE802.11g	100-150 feet	2.4Ghz	11Mbps 54Mbps
3G CDMA	Global	800 Mhz – 1900 Mhz	2Mbps

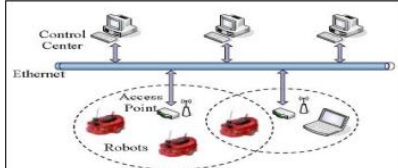


Figure 2.2.2: Conclusion of the Bluetooth, Wi-Fi and wireless LAN and 3G and 2.2.2 concept with WLAN.

Figure 2.2.2 shows a variety of technologies, each with unique features. For example, Ericsson invented Bluetooth technology in 1994. It uses the 2.4GHz frequency spectrum to operate at a range of around 10 metres, allowing for data transfer speeds of up to 1 MB per second. Bluetooth is a popular option for wireless communication and control in a variety of applications because of its benefits, which include cheap cost, low power consumption, and adaptability for use with mobile devices.

Right now, two of the most popular wireless technologies are wireless local area networks (WLAN) and wireless fibre. These networks use the 2.4GHz and 5GHz frequency bands and comply with IEEE 802.11 guidelines. Varying Wi-Fi protocols, including 802.11n (50 Mbps to 100 Mbps), 802.11b (6 Mbps to 11 Mbps), 802.11g (24 Mbps to 54 Mbps), and later generations, might have varying effects on the speed of the network. The paper emphasises how essential WLAN and Wi-Fi are to Robot Communication Systems. For example, figure 2.2.2 shows how operator orders, robot locations, and remote video feeds may be sent between virtual and physical robots over WLAN. Through the use of high-gain antennas and amplifiers, WLAN communication may go more than thirty miles. This feature is essential for improving communication in a variety of applications and enabling remote robot operations.

CHAPTER 2

The frequency band that 3G technology uses for wireless communication is 800MHz–1900MHz. But it's crucial to remember that 4G and 5G technologies, which provide better internet rates and more capabilities, have made 3G less common. Thus, 3G is no longer often used in modern wireless communications. We won't go into the benefits of 3G here since it is becoming less and less relevant and has been mostly replaced by more sophisticated technology.

Limitation of Previous Studies

This article is really from 2012. But 2023 is here. Although the particular technology has changed, the underlying ideas are still applicable. With the advancement of Wi-Fi, protocols like 802.11ac (433 Mbps/sec), 802.11ax (600.4 Mbps/sec), and 802.11be are now available, offering faster data speeds and better performance. To further increase the variety of wireless communication choices, frequency bands have been extended beyond the conventional 2.4GHz to include 5GHz and even 6GHz alternatives [12]. We've moved from 3G to 4G and 5G in the world of mobile phone, which provide more faster and more stable service [13]. These developments emphasise how critical it is to adopt cutting-edge technologies in order to fully realise their potential and preserve the timeless values that form the foundation of wireless communication and control systems.

Proposed Solutions

In actuality, the basic concepts of communication remain crucial for this attempt, even if the specific wireless technology may have evolved over time. A concept that is independent of technological improvements is shown by the use of WLAN to transport data between virtual and real robots, as seen in figure. Using modern WLAN and Wi-Fi technologies may benefit the project by providing faster and more dependable connection, while the core design and principles are still valid. These advancements will make it feasible to execute the network architecture for robot-to-gadget communication efficiently, keeping the project in step with current technology while preserving the core principle of communication.

2.2.3 Multi-Robot Communication System

The author of the study "Adaptive Multi-Robot Communication System and Collision Avoidance Algorithm for Precision Agriculture" [14] introduces the robot-to-robot communication system used in multiple robots in an agricultural field. Using a range of technologies, such as wireless networking, real-time processing, sensing, and the client-server approach, the article focuses on R2R communication.

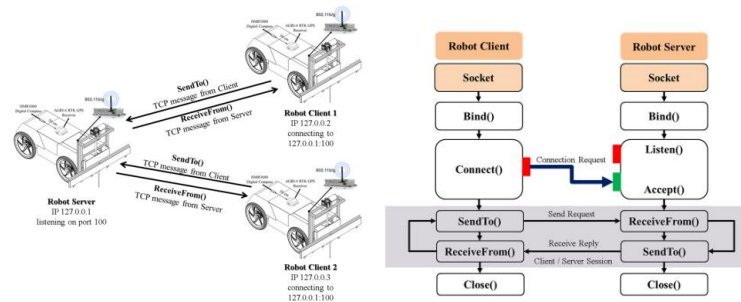


Figure 2.2.3.1 : Communication of Robot Server and Client Figure 2.2.3.2 : TCP Server and Client

The multi-robot wireless communication model, seen in figure 2.2.3.1, is based on the client-server paradigm and uses robots to form multi-robot networks. A mobile robot server and one or more mobile robot clients comprise a client-server paradigm. All data is stored in an array and sent to all robots by the server, which also handles listening for and accepting messages from robot clients. Client just transmits data to server, which server receives. The client-server paradigm, which is seen in figure 2.2.3.2, is one of the most often used communication guidelines in networked systems. Clients in the model only speak with a single server once. Additionally, a server may be in communication with many clients at once. TCP/IP (Transmission Control Protocol/Internet Protocol) and UDP/IP (User Datagram Protocol/Internet Protocol) are the two kinds of communication protocols that are available in client-server models. Robot server and robot client are communicating over TCP/IP in this article. TCP/IP is a communications-based connection protocol that may establish several connections and guarantee error-free transmission. A communication connection is established between the server and client when the server socket binds the port, listens for the client to connect, accepts it, and so on. To connect, the client must know the port number and IP address of the server. Both the robot server and the robot client may send and receive messages over their sockets after they have established a connection.

Limitation of Previous Studies

CHAPTER 2

Less remote-control system. The article has a strong design of communication between server and client at the algorithm, but it does not have a design of a remote-control system to handle some emergency situations. Not only does the remote-control system manage the robot's movement, but it also manages emergency scenarios like power outages or unforeseen obstructions. Remote control systems can be mobile applications, websites, software, and others. For the remote-control system to handle these emergency situations efficiently, it must be dependable and easy to use.

Security Concerns TCP/IP is good for transmitting data between server and client , but it lacks security features. Because the original client server model did not set any encryption or decryption techniques between the client and server, High security can make the data transmitted more secure and protected from unauthorised access. This can let the data not be hacked by the hacker to affect the robot and slave data transfers.

Proposed Solutions

In the project, our remote-control system for handling emergencies will be a mobile application. The smartphone app will display the camera stream, provide users control over the server and client robots' movements, and indicate the TCP/IP connection's condition. This enables users to react to emergencies in a timely manner. A built-in alert system will also be included in the mobile application to inform users of any possible problems or crises. The goal of this all-inclusive solution is to provide users the ability to monitor and control in real-time for effective emergency response. Add the Caesar Cypher technique to TCP/IP to perform encryption and decryption to safeguard the transmission data. More security will be provided by the Caesar cypher approach than by the encryption method. It may have an impact on the stability and dependability of data flow.

2.2.4 Mobile application

The creation of a mobile application for the Android operating system is covered in the article "Remote Control of a Robot Using an Android Mobile Device" [15]. The Android Software Development Kit (SDK)'s Java programming language was used to construct the application, which makes use of Bluetooth technology. This project's primary objective is to create LEGO Mindstorms and a variety of sensors in order to enable remote control of mobile robots.

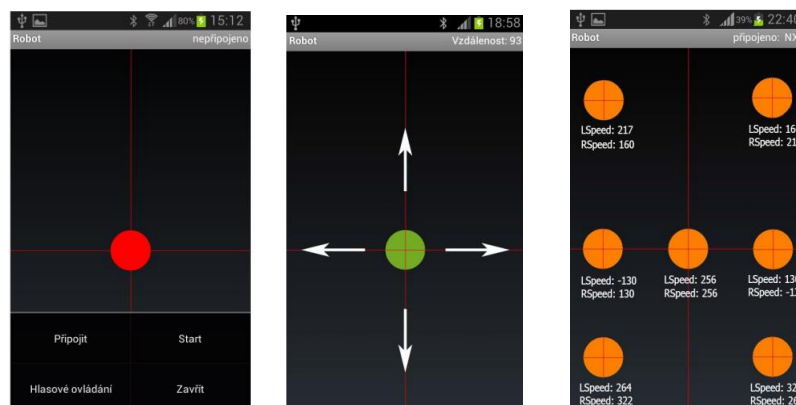


Figure 2.2.4.1: User interface ,Figure 2.2.4.2: Mobile application in running mode, Figure 2.2.4.3: Motor speed for different positions of trackball.

Establishing a connection with the mobile robot and determining if Bluetooth is enabled are two functions of the mobile application. In case of an unsuccessful connection, a red trackball, like the one in figure 2.2.4.1, will be shown to indicate a poor connection. The trackball turns from red to orange to indicate a successful connection after it has been made. Robot navigation is started by the user by pushing a "start" button, which causes the colour to change from orange to green, signifying that the robot is ready to travel. The hue changes from green to blue, signifying outgoing communication, as the user moves the trackball, and the robot reacts by moving in the chosen direction. As shown in Figure 2.2.4.3, the left (LSpeed) and right (RSpeed) motors' speeds are determined by the model control using the variables "Speed" and "Value". When the robot is in the centre position, it advances straight forward. The mobile robot turns left when a user touches it on the left, and right when they contact it on the right. To make the turn, one of the motors and the left side slow down. Robot movement may be controlled dynamically and responsively thanks to the processing of these touch inputs that are sent to the robot via Bluetooth.

Limitation of Previous Studies

Simple function of mobile applications

Though its capabilities may be seen as rather restricted, the mobile application included in the article mainly concentrates on voice control, Bluetooth status display, and rudimentary navigation. The mobile application might be developed to include more sophisticated features to improve its usefulness and problem-solving skills. For example, it may be built to read and show motor data from the robot and ultrasonic sensor data. A more informed and efficient remote control and supervision of the mobile robot would be possible with this added feature, which would provide users real-time insights into the robot's state.

Communication problem

As seen in the mobile application's functionality is restricted by the 10 metre Bluetooth technology's range. To create and sustain a trustworthy connection, the robot and the mobile device must stay within this range. Users using the mobile application for robot control should thus be aware of this restriction. Furthermore, it is vital to comprehend that the mobile application is designed to operate a single robot, and its user interface is optimised for this specific link.

Proposed Solutions

The main aim of the project is to enhance the idea shown in the article via the creation of an Android mobile application with Java programming. The project will include capabilities for commanding robot arms, managing cameras, mapping surroundings, and combining numerous other functions, so considerably improving the application's capability while maintaining basic navigation and status display features. In order to get around communication constraints, WLAN will be used to provide the mobile application and the robot with stronger, longer-lasting communication. Additionally, in order to increase the application's adaptability for users controlling various robotic systems, the project seeks to build an intuitive graphical user interface (GUI) that facilitates the operation of many robots, maybe with the use of numerical IDs for switching between them.

2.2.5 Summary of Review of the Existing Systems

Table 2.2.5: Summary of Review of the Existing Systems table

Project	Strength	Weakness	Critical Comments
Development of a master–slave robot system for farm operations	<ul style="list-style-type: none"> -Two algorithm by using the master and slave system. GOTO and FOLLOW. -GOTO is navigate -FOLLOW is use in follow 	<ul style="list-style-type: none"> -Testing use one master and slave -Not have remote control system 	<ul style="list-style-type: none"> -Create the new algorithm similar with GOTO and FOLLOW -Create a remote system
A Review of Wireless Technology Usage for Mobile Robot Controller	<ul style="list-style-type: none"> -Describe a few wireless technologies like Bluetooth, WIFI and WLAN and 3G -Bluetooth use 2.4GHZ and range is 10 meter -WIFI and WLAN use 802.11n (50mbps to 100mbps) -Concept of WLAN in robot controller 	<ul style="list-style-type: none"> -Versions outdate -In 2023, WI-FI have 2.4GHz and 5GHz -Have 4G and 5G 	<ul style="list-style-type: none"> -Using the concept of WLAN in robot controller -Using the new version like 802.11ac ,4G and 5G into the concept
Adaptive Multi-Robot Communication System and Collision Avoidance Algorithm for Precision Agriculture	<ul style="list-style-type: none"> -Utilizes a client-server paradigm for efficient communication between robots.. -Provides a clear communication model using TCP/IP for reliable data transmission. 	<ul style="list-style-type: none"> -Lacks a remote-control system for managing emergency -Lack of encryption technology in communication protocols 	<ul style="list-style-type: none"> - mobile application for remote control - Caesar Cipher technique for encryption
Remote Control of a Robot Using an Android Mobile Device	<ul style="list-style-type: none"> -Have mobile application to control the robot navigation and status of Bluetooth -How the touch to affect the motor 	<ul style="list-style-type: none"> -Control one device only -Less function in mobile application 	<ul style="list-style-type: none"> -Create a mobile application can control multipin robot. -Create new function like camera, control arm and status of robot

CHAPTER 3

3.1 System Development Models

3.1.1 Waterfall Model

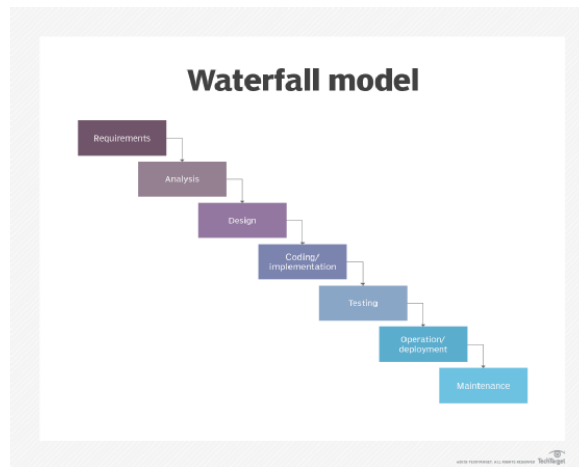


Figure 3.1.1: Waterfall Degree

The waterfall model is one of the theories of software development. This is a step-by-step model. The figure 3.1.1 shows the waterfall model, which divides the life cycle into a set of phases. Once the previous stage is completed, the next stage can begin [16]. The key word has requirements, design, coding, testing, deployment, and maintenance. The advantages include being easy to understand, properly defined, clear milestones, and other benefits. Also have disadvantages such as no feedback path, difficulty changing requests, limited flexibility, and other disadvantages. The project has clear requirements, a small or medium size, few changes, and other characteristics that can be used in the development of the project.

3.1.2 Agile Model

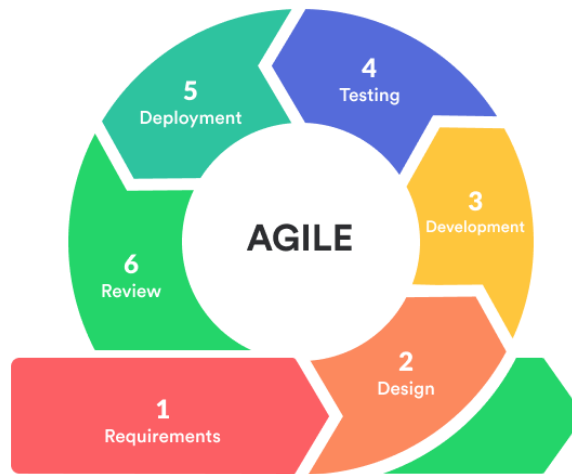


Figure 3.1.2: Agile Model Degree

The Agile model is one of the theories of software development. It is a method of emphasising flexibility, collaboration, and customer focus. This method has been used by Facebook, Google, Amazon, and other companies. Tasks are divided into smaller iterations by the model, or portions that do not directly need long-term planning. At the start of the development process, the requirements and scope of the project are established. Plans are outlined in advance with reference to the quantity, length, and scope of each iteration. [17]. The key phases have Requirements, design, development, testing, deployment, and review. The advantage is that it is efficient to target customer requirements, reduces total development time, allows for anytime change, and other advantages. Disadvantages: importance of formal documents to decisions; high dependence on customer requirements; importance of proper documentation; and other disadvantages. If the project has a high focus on customers, frequent modifications, flexible project schedules and budgets, and other factors, others can accept this method to develop the project.

3.1.3 Incremental Model

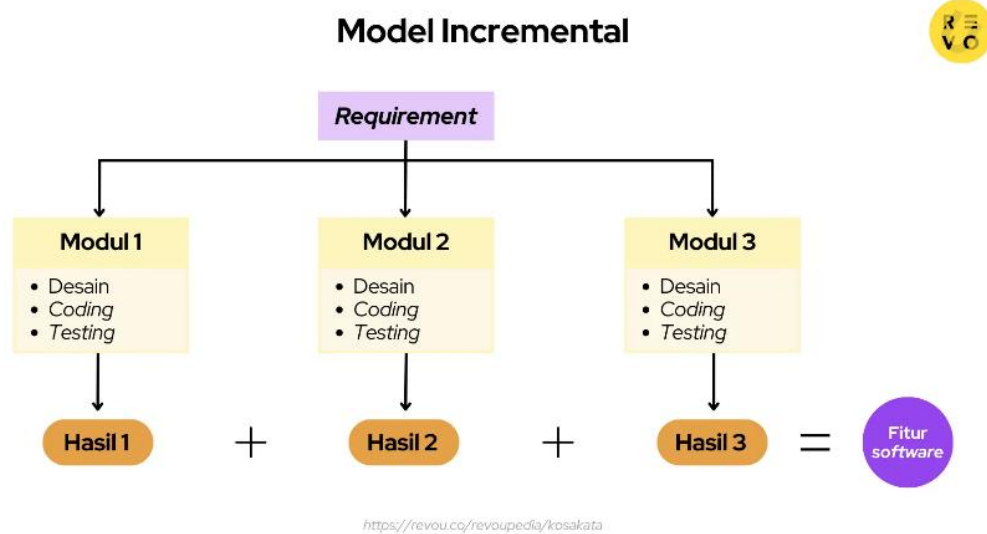


Figure 3.1.3: Incremental Model Degree

The Incremental Model is a software development theory. The requirements are divided into multiple modules of the software development cycle. Each module has a set of requirements, design, coding, and testing phases. The module's functionality is increased with each new version. The procedure continues until the whole system is completed. [18] The advantages are that errors are easier to detect, easier to test and debug, more flexible, and other benefits. Disadvantages include integration challenges, complexity, good planning, and other disadvantages. If the project has these characteristics, like error reduction, requires good planning and design, and requires early realisation of benefits, others can use this method.

3.1.4 V-Model

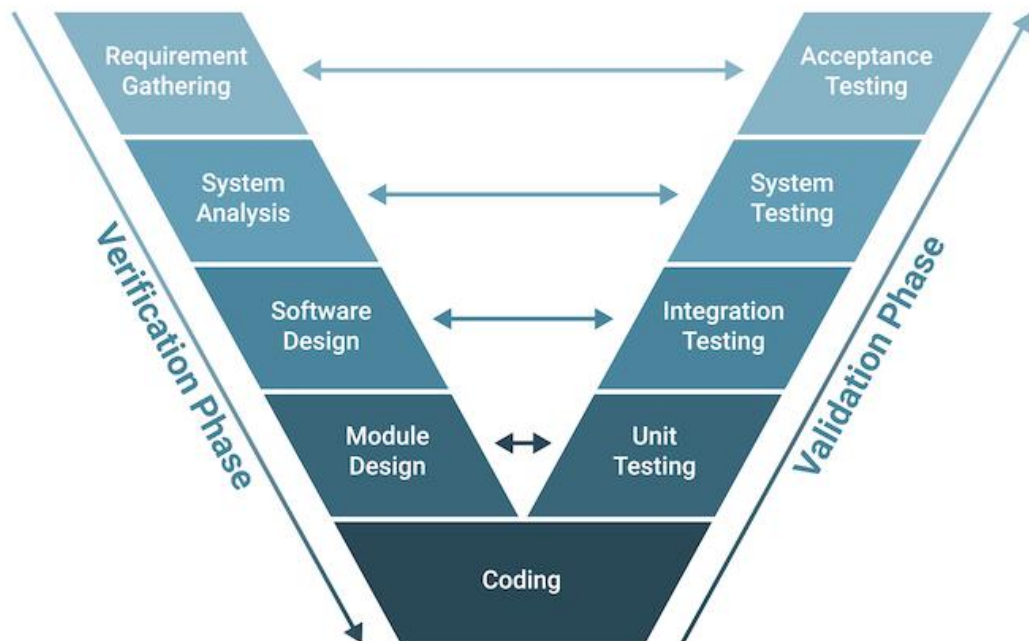


Figure 3.1.4: V-Model Degree

V-Model is a theory of software development. Like 3.1.4., a V-shape development is like a waterfall, but it has a verification and validation model. Its foundation is the assignment of a testing phase to every related development step. Every step's development is closely related to the testing stage. The subsequent phase does not begin until the preceding phase is finished; there is a testing activity for every development activity.[19] The verification phase has requirements for system analysis, software design, module design, and coding. The validation phase has unit testing, integration testing, system testing, and acceptance. These verification and validation processes are joined by the coding phase in a V-shape. Thus, it is known as the V-Model. The advantages are easy to understand, save time, work well for small plans, and other advantages. The disadvantages are least flexibility, high risk and uncertainty., no early prototypes of the software being produced, and other disadvantages. The project is complex, waterfall-like, and safety-critical. This model is good for the project.

3.1.5 Selected Model

Incremental Model

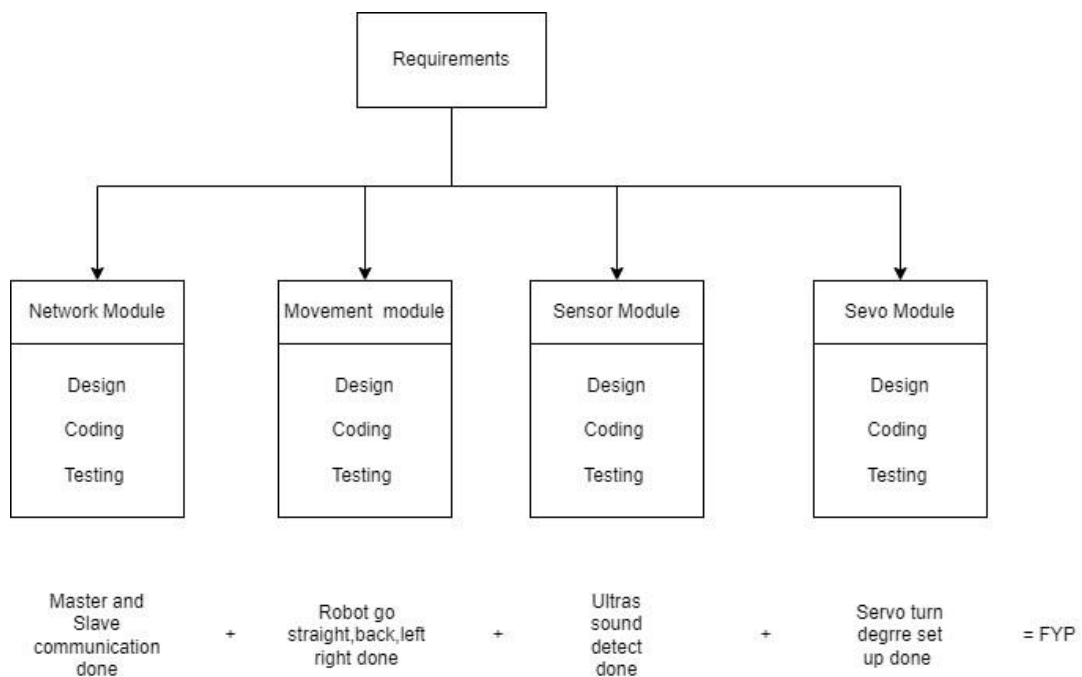


Figure 3.1.5 Incremental Model of the project

In the project, the most suitable model is the incremental model. Compared with other models, the incremental model emphasises breaking the project into smaller, manageable parts, or increments. Each increment delivers a portion of the final product's functionality. Not like the waterfall model, need to plan a full process of development. The incremental model gives development functions one by one. Like the 3.1.5. showing. Not only project into smaller, but incremental model also give risk management may be more focused on each increment rather than the entire project. A robot project is a high-risk project because it has a lot of unknown bugs in testing. In this project, after testing, just adding to the development will reduce the risk of error in the project. The incremental model also brings quality assurance. By testing functionalities incrementally to ensure that issues are identified and addressed early, leading to higher overall product quality.

3.2 System Requirement

3.2.1 Hardware

3.2.1.1 Gopigo3 robot

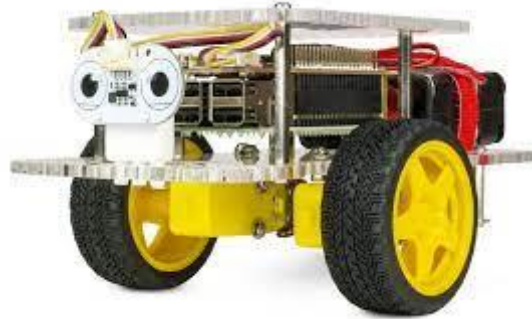


Figure 3.2.1.1: Normally Gopigo3 robot

Gopigo3 robot also can called raspberry pi robot because main process by raspberry pi 3B or 4 but it connects GoPiGo3 board .Gopigo3 is a robot kit built by DUXTER industries. This robot is focus on education so that easy building ,programming and modify . A normal GoPiGo3 robot have one GoPiGo3 electronic board ,one raspberry pi , one distance sensor, one battery , 2 Servo motor and 2 tire. In this report will modify to multiple sensors ,hand ,storage to get the goal [15].

3.2.1.2 GoPiGo3 Electronic Board

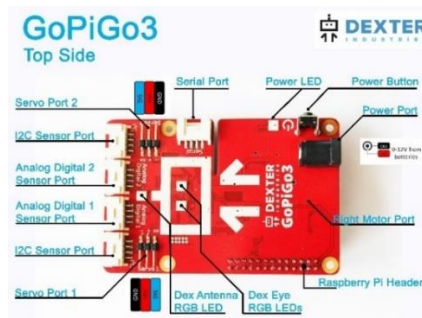


Figure 3.2.1.2: GoPiGo3 Electronic Board

Item	Number	Description
Servo port	2	To connect the servo motor
I2c sensor port	2	To connect the i2c type sensor
Analog digital sensor port	2	To connect analog digital type sensor
Serial port	1	To communication interface through which information transfers
Power port	1	To connect the battery
Led	3	To show the signal and light
Button	1	To on off the power
Raspberry pi pin	1	To connect the board and raspberry pi

Table 3.2.1.2: GoPiGo3 Electronic Board Description table

3.2.1.3 Raspberry pi 3B+



Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO12 (SDA1, I2C)	DC Power 5v	04
05	GPIO13 (SCL1, I2C)	Ground	06
07	GPIO14 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO17 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO9 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CEO_N) GPIO8	24
25	Ground	(SPI_CEO_P) GPIO7	26
27	ID_30 (I2C ID EEPROM)	(I2C ID EEPROM) ID_3C	28
29	GPIO5	Ground	30
31	GPIO6	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Figure 3.2.1.3.1: Raspberry Pi 3B+ Figure 3.2.1.3.2: 40 Extend GPIO

Item	Description
4 x USB port	To connect the USB devices
Lan port	To connect to internet
HDMI port	To connect monitor
Audio port	To output the sound
40 Extend GPIO	To connect the GoPiGo3 electronic board or another sensor
CPU (Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz)	To process of logical and mathematical operations and executing instructions that it is given.
Power Port	To connect to the battery or power supply
Wi-Fi and Bluetooth chip	To let the device, have Wi-Fi and Bluetooth function
CSI camera port	To connect the CSI camera.

Table 3.2.1.2: Raspberry pi 3B+ Description table

3.2.1.4 Arm Gripper



Figure 3.2.1.4: Robot Arm Gripper

Specifications	Description
Material	Hard Aluminum Alloy
Weight	~ 68g
Max. Opening Angle	54mm
Max. Length	108mm (while Claw closed)

Table 3.2.1.4 Arm Gripper Description table

3.2.1.5 Servo MG996R



Figure 3.2.1.5: Servo MG996R

Specifications	Description
Model	MG996R
Stall torque	9.4 kgf·cm (4.8V), 11 kgf·cm (6 V)
Operating speed	0.17 s/60° (4.8 V), 0.14 s/60° (6 V)
Operating voltage	4.8V a 7.2V
Running Current:	500mA.
Stall Current:	2.5A (6V).
Dead band width:	5µs
Weight:	55g.
Dimension	: 40.7 x 19.7 x 42.9 mm approx.

Table 3.2.1.5 Servo MG996R table

3.2.1.6 Metal Servo Motor Hub (MG996R)



Figure 3.2.1.6: Metal Servo Motor Hub (MG996R)

Specifications	Description
Shape	Circular disc-shaped
Teeth Count	25 teeth
Diameter	14 mm
Purpose	Connects to servo motor's output shaft, transmits rotary motion to wheel
Material	Metal

Table 3.2.1.6 Metal Servo Motor Hub table

3.2.1.7 Servo Board PCA9685

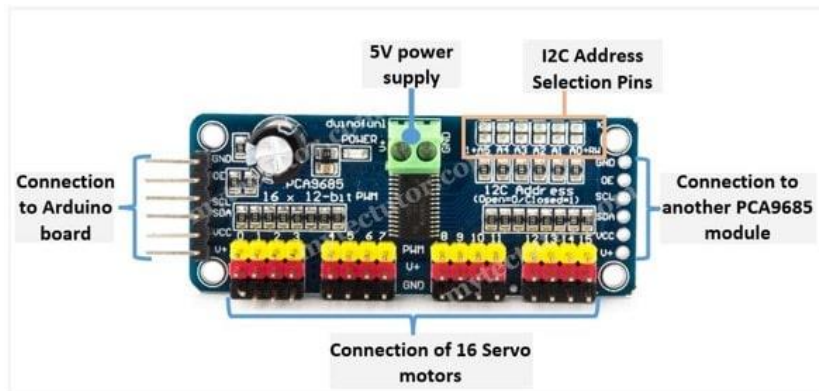


Figure 3.2.1.7: Servo Board (PCA9685)

Specifications	Description
Model	PCA9685
Function	16-channel PWM servo/LED controller
Control Interface	I ² C interface
Operating Voltage	2.3V to 5.5V
PWM Frequency	Adjustable frequency up to 1.6 kHz
Output Current per Channel	25 mA
Number of Channels	16 independent PWM channels
Operating Temperature	-40°C to +85°C
Address Configurability	Up to 62 addresses (configurable by external address pins)

Table 3.2.1.7 Servo Board PCA9685 table

3.2.2 Software

3.2.2.1 Android studio

Android Studio



Figure 3.2.2.1: Android studio Logo

The authorised Integrated Development Environment (IDE) for creating Android apps is called Android Studio. Java and Kotlin as the primary programming languages, it makes the process of developing mobile apps . Developers may easily evaluate and improve their apps while working on them thanks to an IDE feature that lets them see mobile applications in real time. 18]In this project, we will use **Java programming and Android Studio to create a mobile application** that will be used to operate robots.

3.2.2.2 Geany



Figure 3.2.2.2: Geany Logo

Geany is the chosen IDE for the GoPiGo3 robot. It is valued for its lightweight and efficient properties in resource-limited environments such as IoT and robotics. It is the best option for these applications because to its low resource usage. Numerous programming languages, including C++, C, Python, and others, are supported by Geany. [19]. **Python** was chosen for this project because of its ease of use and compatibility with **robotics development**, and it will be used to programme the GoPiGo3. Within the project's parameters, this combination of Geany and Python guarantees a simple and efficient method of operating the GoPiGo3 robot.

3.2.2.3 Real VNC Viewer



Figure 3.2.2.3: Real VNC Viewer Logo

Real VNC Viewer is software that enables you to remotely access a device via a VNC (Virtual Network Computing) server. To enter the device UI page as if they were physically present in front of it by using an IP address, user name, and password. Its software function is like Anydesk or TeamViewer, but it needs the device IP address. In this project, this software is used to remote access the GoPiGo3 robot.

3.2.2.4 MobaXterm

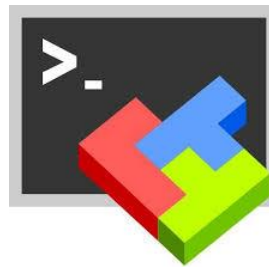


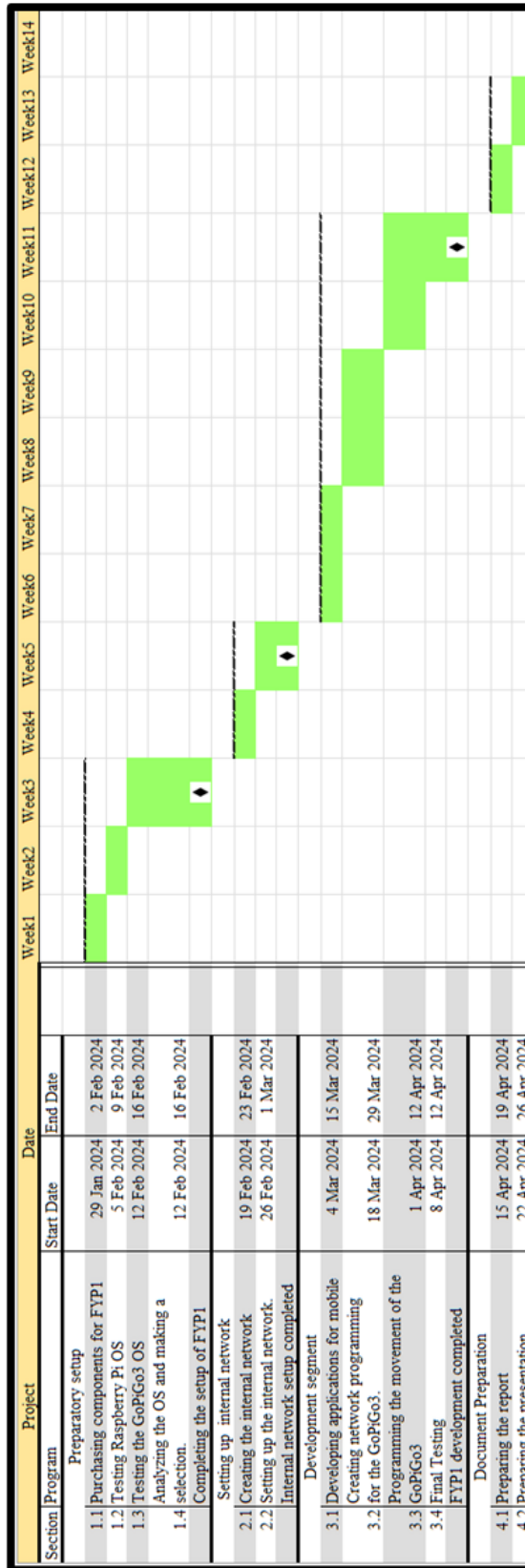
Figure 3.2.2.3: MobaXterm Logo

MobaXterm is software to launch remote sessions such as SSH, Telnet, Rlogin, RDP, VNC, XDMCP, FTP, SFTP, or serial sessions. It has a graphic SFTP browser function. When a remote server is using SSH, the SFTP browser pops up. The user can drag and drop files directly from/to the remote server or read and write the file using MobaTextEditor. In this project, it will support to ssh remote GoPiGo3 Robot and server in middleware to modify code in my own window device.

3.3 Project Milestone

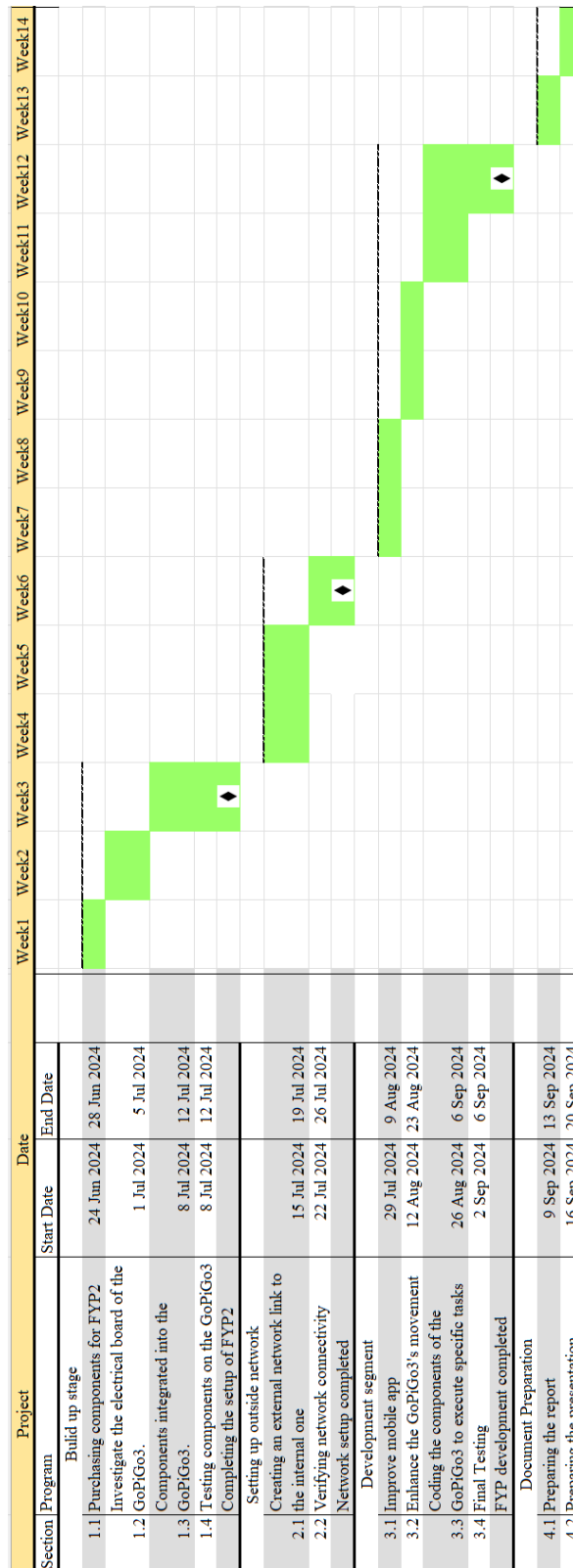
3.3.1 FYP1 Timetable and Milestone

Figure 3.3.1 FYP1 Timetable and Milestone



3.3.2 FYP2 Timetable and Milestone

Figure 3.3.2 FYP2 Timetable and Milestone



3.4 Estimated Cost

No	Compent	Price per unit (MYR)	Quantity	Total Price
1	Sim Card Router	99.00	1	99
2	Robotic arm gripper	13.30	1	13.3
3	Servos Digital MG996R	9.90	5	49.5
4	US-015 Ultrasonic Sensor Distance	9.90	3	29.7
5	12V 2.3AH Battery	49.90	1	49.9
6	Digi Prepaid Sim Card	25.00	2	50
7	Micro SD Card 32GB	14.00	3	42
			Total Cost	333.4

Table 3.4: Estimated Cost table

CHAPTER 4

4.1 System Architecture

4.1.1 Hardware Architecture

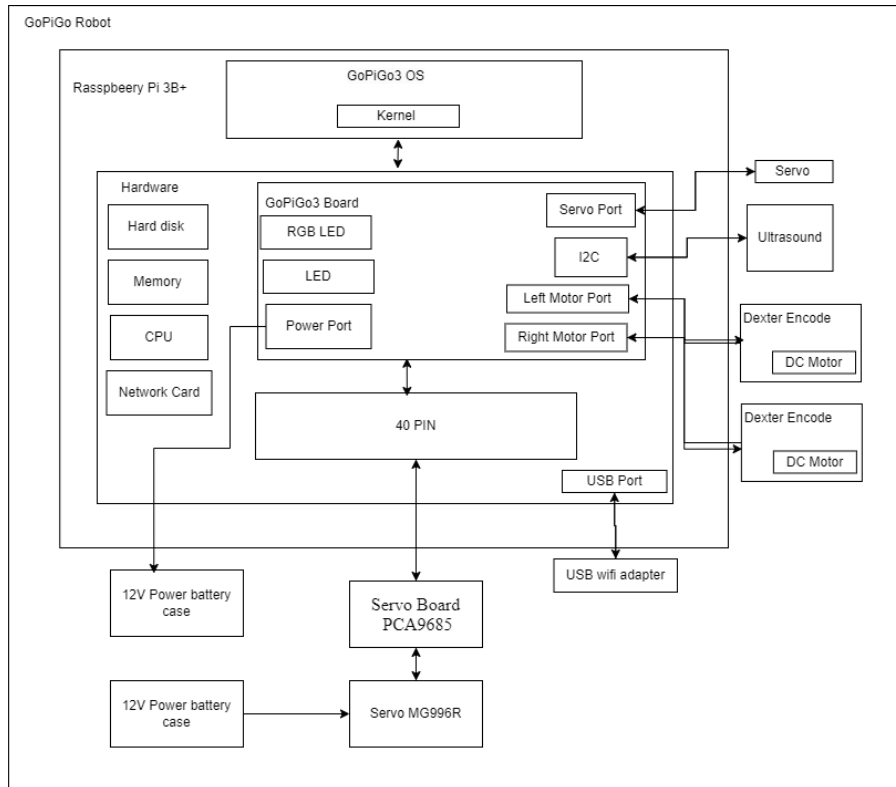


Figure 4.1.1.1: GoPiGo 3 Hardware Architectures

Raspberry Pi interface

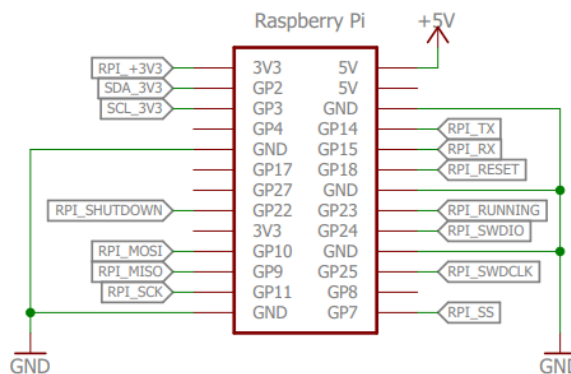


Figure 4.1.1.2: GoPiGo3 Hardware Architectures

Both figures are GoPiGo3 hardware architecture by using a Raspberry Pi board and a GoPiGo3 board to build a robot. Figure 4.1.1.2 shows the GoPiGo3 board using which interface of 40 pins of the Raspberry Pi to connect the GoPiGo3 interface, like I2C, motor port, and servo port.

4.1.2 Network Architecture

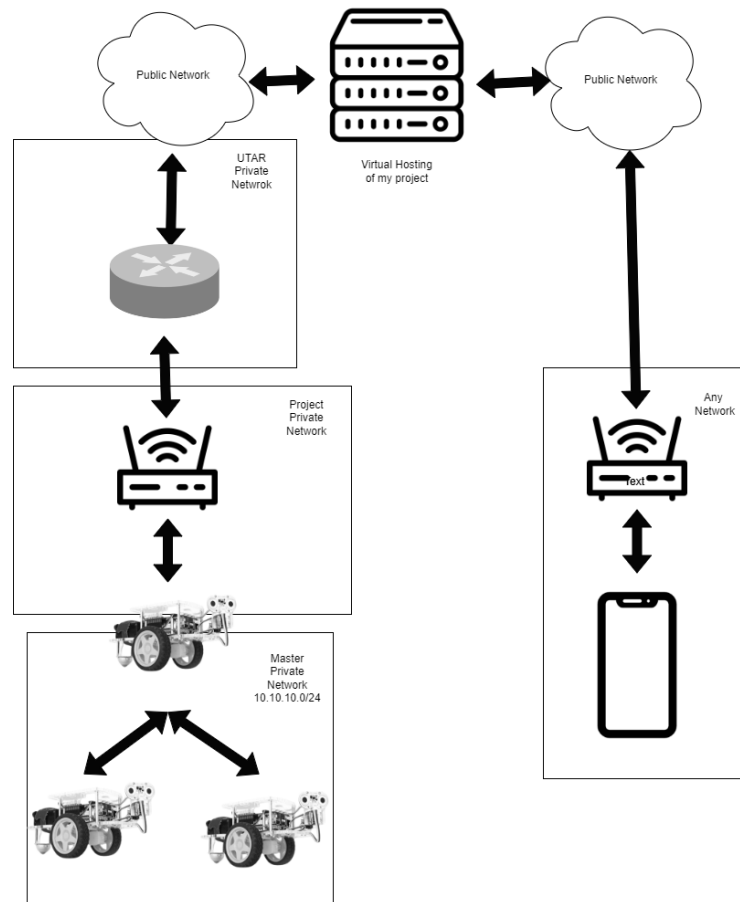


Figure 4.1.2: FYP2 Network Architecture

In this network diagram, the master and slave have their own private network; the IP range is 10.10.10.0/24. Master is connected to a router; there are other private networks. This router facilitates Network Address Translation (NAT) to the UTAR router. The UTAR router then connects to another private network that routes traffic to the public internet.

Mobile can connect to any available network but must communicate with a middleware server. The middleware server serves as a communication hub. Master and mobile will communicate by using a middleware server to do so by-passing messages to each other.

4.2 Functional Modules in the System

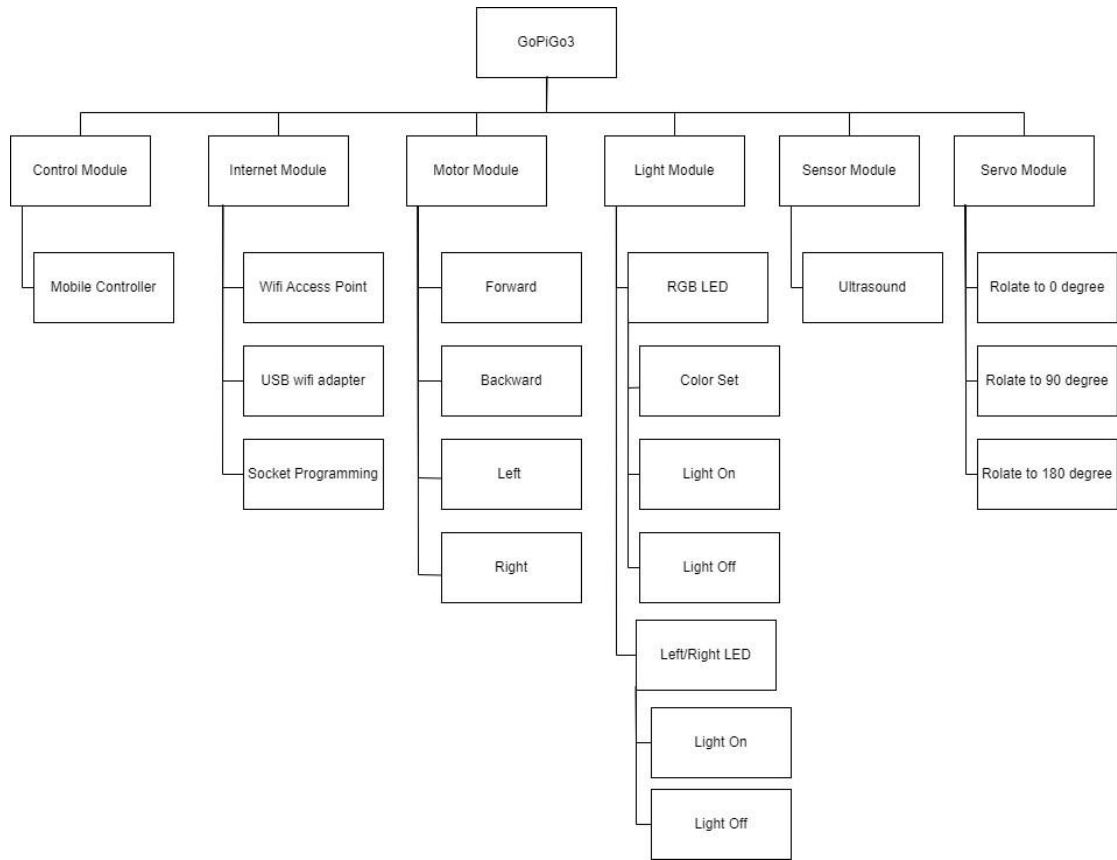


Figure 4.2: Functional Modules Degree

4.3 System Flow

4.3.1 Main Master

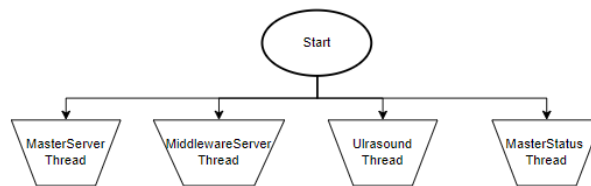


Figure 4.3.1: Main Master Flow Chart

This flowchart is showing that while running the Python script, it will start 4 threads: **MasterServer** Thread is letting the create a TCP server, and **MiddlewareServer** **Therad** is connecting to the middleware server. **Ultrasound Thread** is run the ultrasound. **MasterStatus** is a loop to check the status of robots.

4.3.2 Master Server Thread

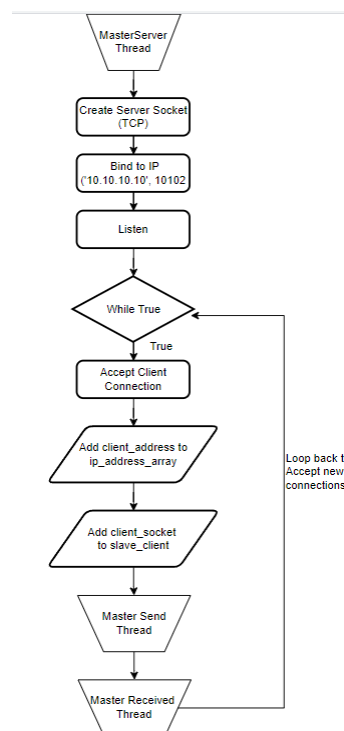


Figure 4.3.2: Master Server Thread Flow Chart

This flowchart explains one of the main components of the MasterServer, specifically the MasterServer Thread. This thread plays a crucial role in creating a TCP socket to allow client connections. When a client is accepted from the socket, the thread creates two additional threads, the **Master Send Thread** and the **Master Receive Thread**. These threads are let the **master to send data to and receive data from the slave**.

4.3.2.1 Master Send Thread

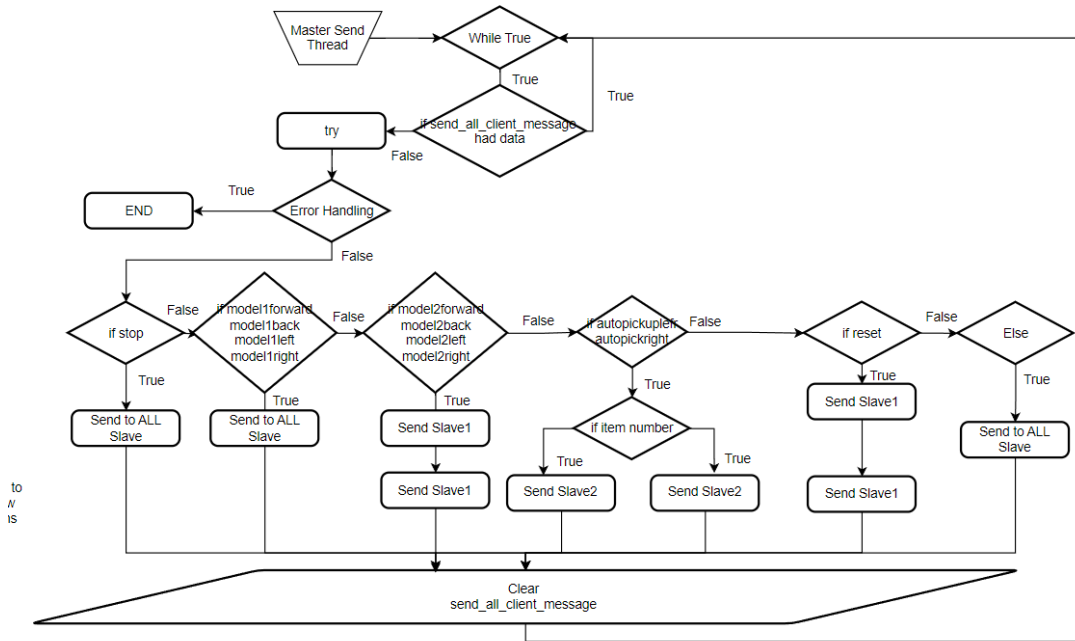


Figure 4.3.2.1 Master Send Thread Flow Chart

This flowchart provides more details about the Master Send Thread within the Master Server Thread. The Master Send Thread continuously checks if the send_all_client_message variable has a value. If it does, the thread sends the message to slave1, slave2, or both, depending on the conditions. The send_all_client_message is a global variable that holds the message intended for the slaves. After sending the message, the variable is cleared to prepare for the next incoming message.

4.3.2.2 Master Received Thread

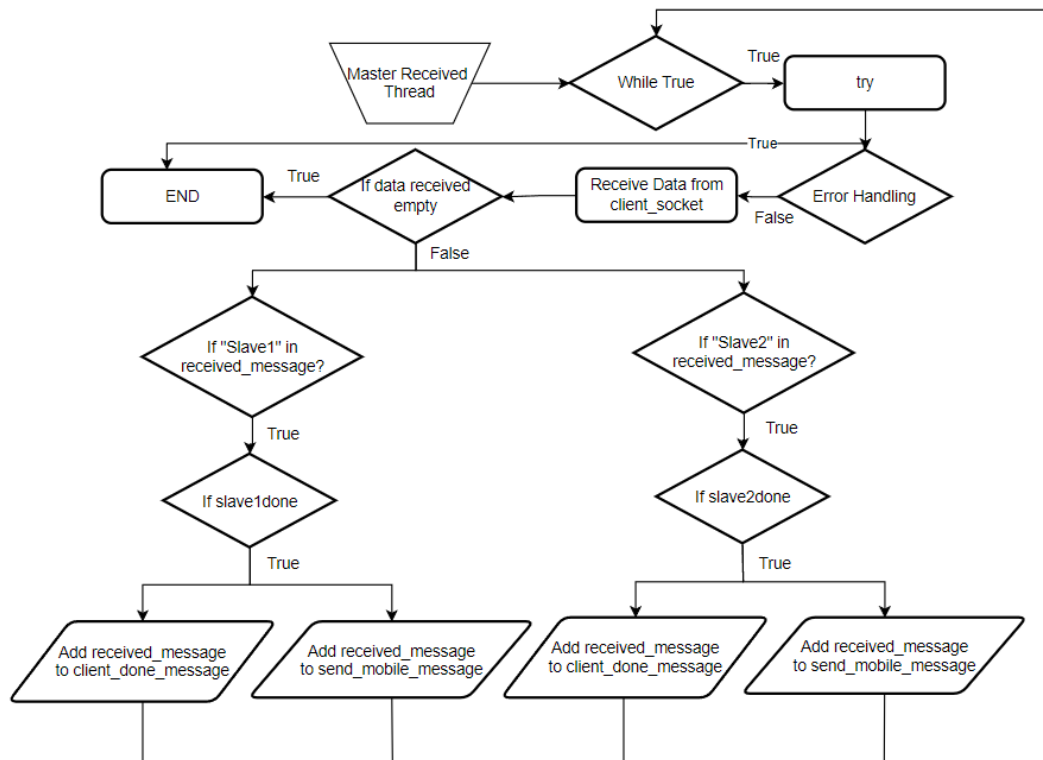


Figure 4.3.2.2 Master Received Thread Flow Chart

This flowchart provides more details about the Master Receive Thread within the Master Server Thread. This thread is responsible for handling messages received from the slaves. Messages from slaves contain the keyword “slave,” which helps identify them. The thread processes these messages and forwards them to either the mobile application or master system as needed. If the data received is empty, it indicates that the client has disconnected.

4.3.3 Middleware Thread (Master)

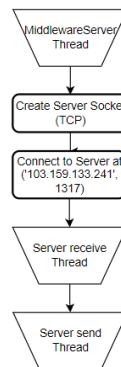


Figure 4.3.3 Middleware Thread Flow Chart

This flowchart explains one of the main components of the MasterServer Thread, specifically the **Master Middleware Thread**. This thread connects the master robot communications to the middleware server. Similar to the Master Server Thread, it also includes two threads, the **Server Receive Thread** and the **Server Send Thread**. Both threads work together to **sending data from the master robot to the middleware server and receiving data from the middleware server**.

4.3.3.1 Server Send Thread (Master)

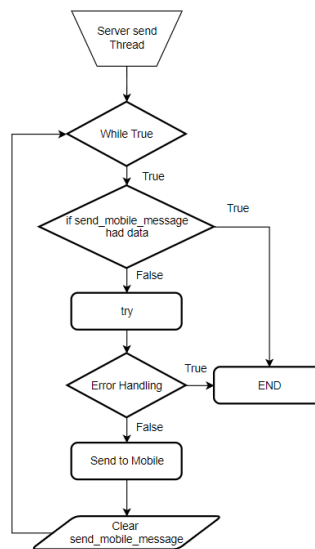


Figure 4.3.3.1 Server Send Thread Flow Chart

The Server Send Thread in the flowchart has a key purpose: if the send_mobile_message global variable contains a message, it means this message needs to be sent to the mobile application through the middleware. The send_mobile_message is sent to the middleware server, and after sending, the variable clear to the next message.

4.3.3.2 Sever Receive Thread

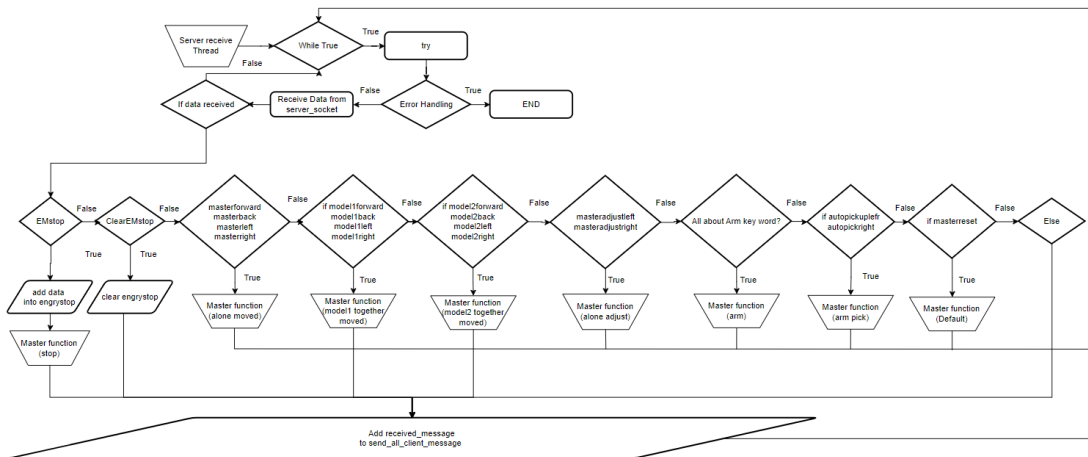


Figure 4.3.3.2 Sever Receive Thread Flow Chart

The flowchart of the Server Receive Thread is designed to process messages from the middleware, which also originate from the mobile application. When data is received from the middleware, the thread determines whether the message is intended for the master robot only or if it needs to be passed to a slave robot. If the message needs to be sent to a slave robot, it is stored in the `send_all_client_message` global variable. This process runs in conjunction with the Master Send Thread.

4.3.4 Ultrasound Thread (Master)

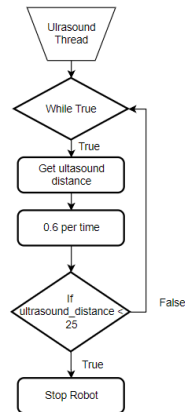


Figure 4.3.4 Ultrasound Thread Flow Chart

This flowchart show a thread that enables the robot to process values from the ultrasound sensor. If the sensor detects that the robot is very close to an object, it will let the robot stop it and move a little backward. This can avoid the robot from colliding with obstacles.

4.3.5 Master Status Thread

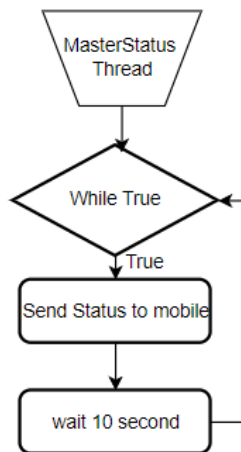


Figure 4.3.5 Master Status Thread Flow Chart

This flowchart represents a simple thread that reads the status of the master robot and sends it to the middleware server every 10 seconds.

4.3.6 Slave Main

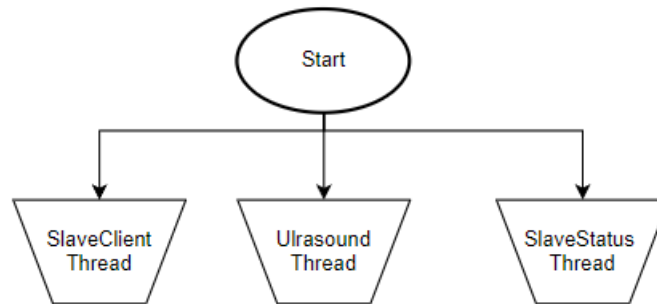


Figure 4.3.6 Slave Main Flow Chart

This flowchart is similar of the master main one, it will start 3 threads: and **Slave Client Therad** is connecting to the master robot. **Ultrasound Thread** is run the ultrasound. **MasterStatus** is a loop to check the status of robots.

4.3.7 Slave Client Thread

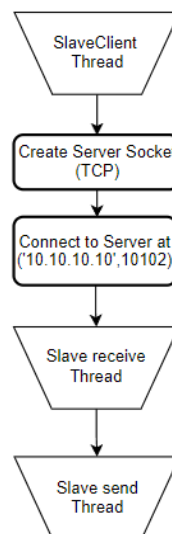


Figure 4.3.7 Slave Client Thread Flow Chart

This is flowchart similar of Middleware Thread, this thread connects the Slave robot communications to the master robot. It has included two threads, the **Slave Receive Thread** and the **Slave Send Thread**. Both threads work together to **sending data from the slave robot to the master robot and receiving data from the master robot.**

4.3.7.1 Slave Send Thread

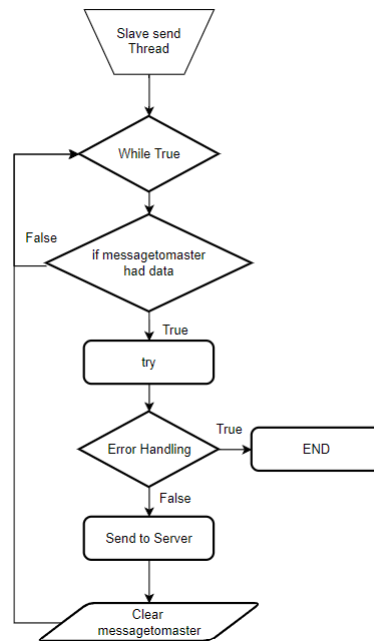


Figure 4.3.7.1 Slave Send Thread Flow Chart

The Slave Send Thread in the flowchart has a key purpose: if the messagetomaster global variable contains a message, it means this message needs to be sent to the master robot by socket IP address and port. The messagetomaster is sent to the master robot, and after sending, the variable clear to the next message.

4.3.7.2 Slave Receive Thread

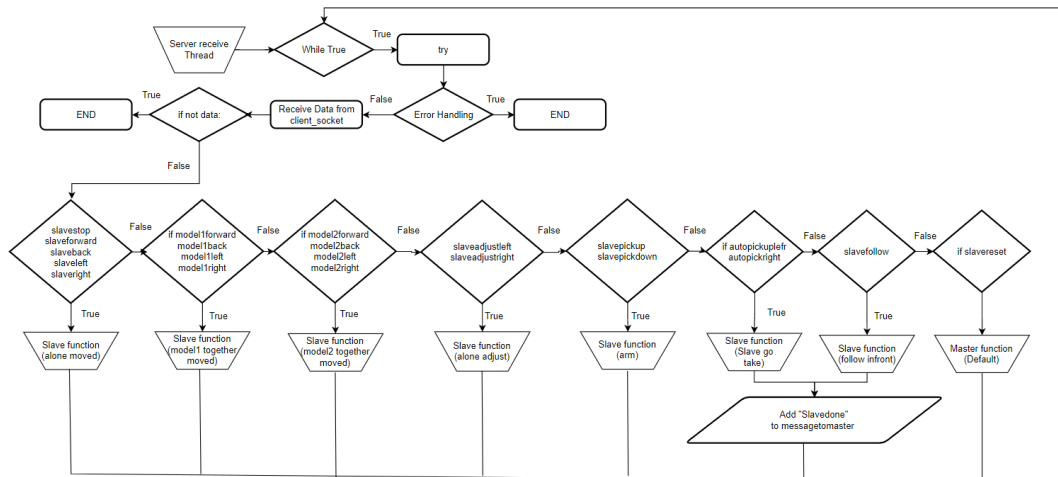


Figure 4.3.7.2 Slave Receive Thread Flow Chart

The flowchart of the Slave Receive Thread is designed to process messages from the master robot. When data is received from the master robot, the thread determines whether the message is intended for the slave robot only or if it needs to be passed to a master robot. If the message needs to be sent to a master robot, it is stored in the `messagetomaster` global variable.

4.3.8 Ultrasound (Slave)

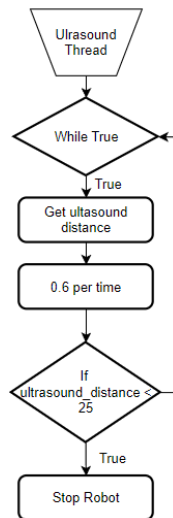


Figure 4.3.8 Ultrasound Thread Flow Chart

This flowchart shows a thread that enables the robot to process values from the ultrasound sensor. If the sensor detects that the robot is very close to an object, it will let the robot stop it and move a little backward. This can avoid the robot from colliding with obstacles.

4.3.9 Slave Status Thread

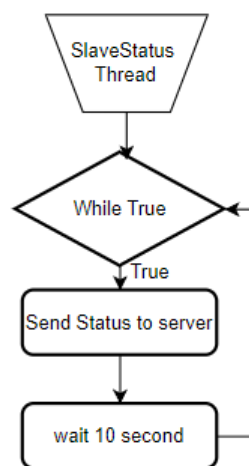


Figure 4.3.9 Slave Status Thread Flow Chart

This flowchart represents a simple thread that reads the status of the slave robot and sends it to the master robot every 10 seconds.

4.3.10 Mobile application TCP

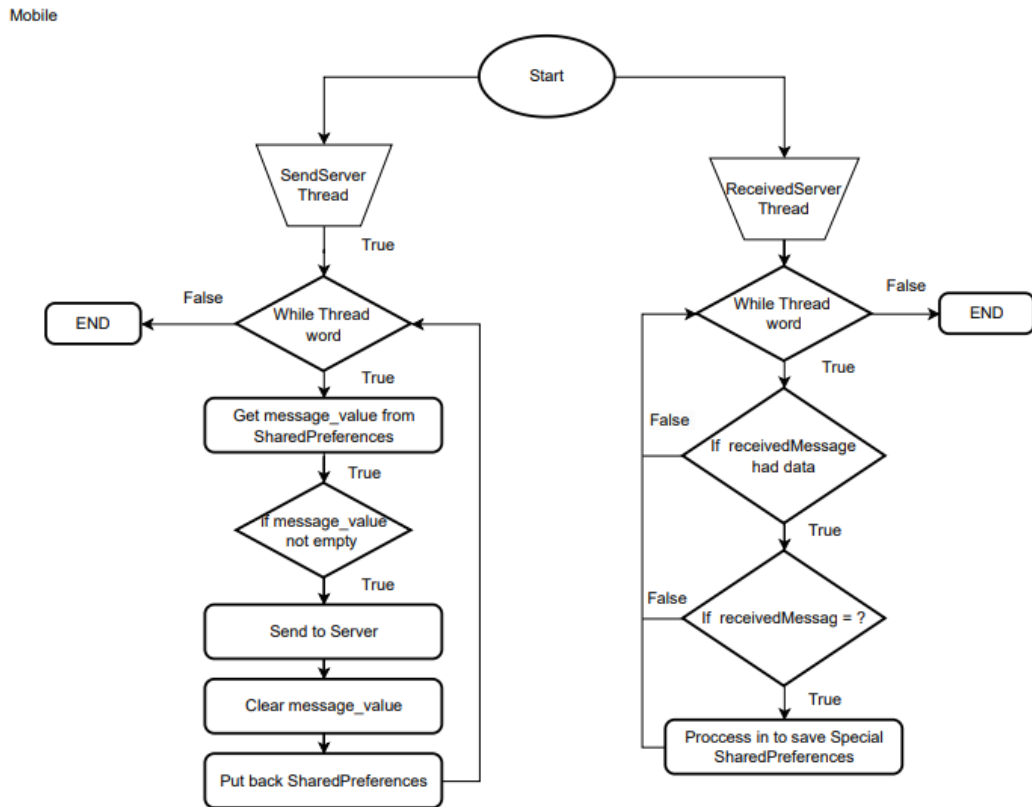


Figure 4.3.10 Mobile Application TCP Flow Chart

This flowchart (4.3.10) illustrates the working flow of the mobile application. It is a key component in the application, enabling communication between the mobile app and the middleware server. Two threads, the **SendServer thread**, which sends signals from the mobile app to the middleware, the **ReceivedServer thread**, which receives and processes messages from the middleware.

4.3.11 Middleware server

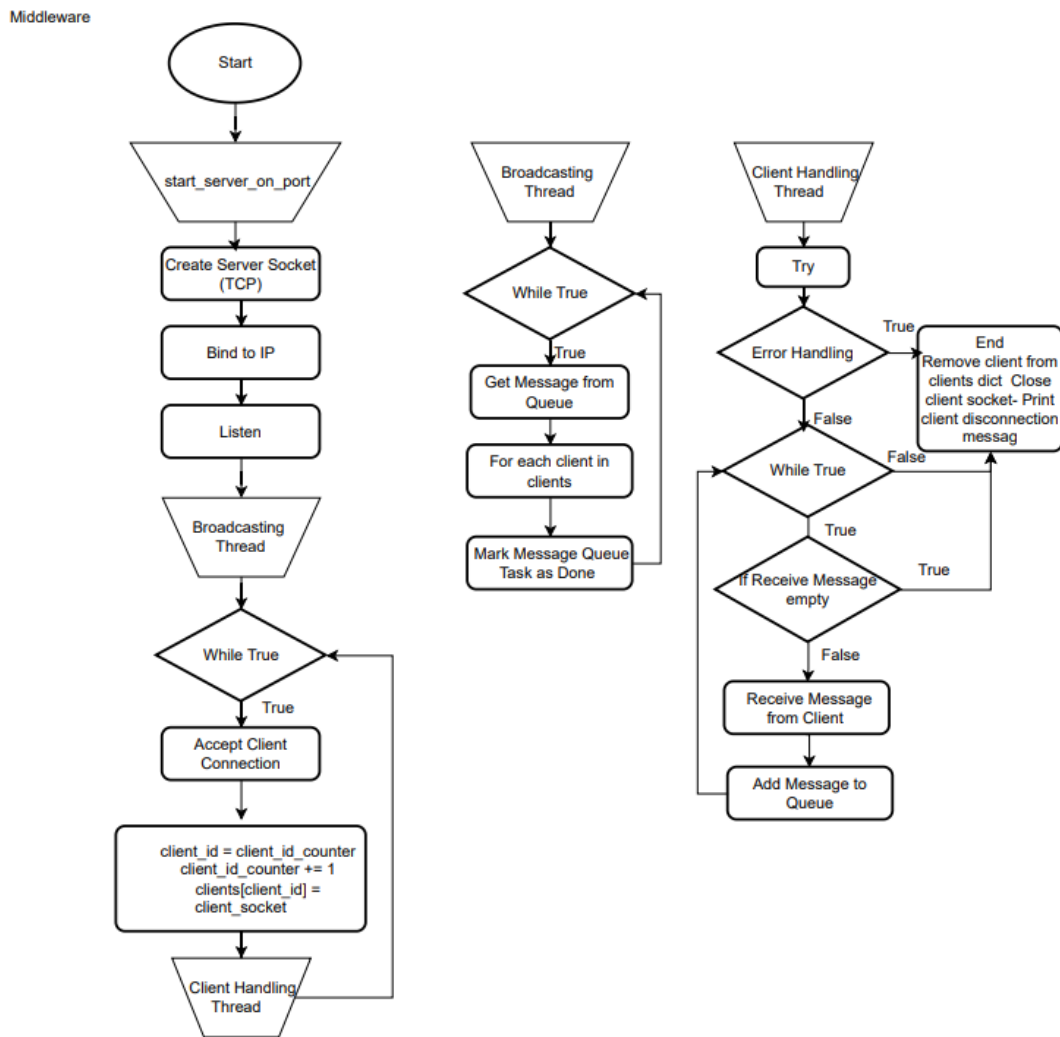


Figure 4.3.11 Middleware Server TCP Flow Chart

This flowchart 4.3.11 explain the middleware server. The **Start_server_on_port** thread creates a TCP socket to accept connections from clients. When a connection is established, it assigns a unique ID to the client and spawns a **Handle_Client_thread** to manage the client's interactions. The **Broadcasting thread** ensures that messages from any client are distributed to all other connected clients.

4.4 GUI Design

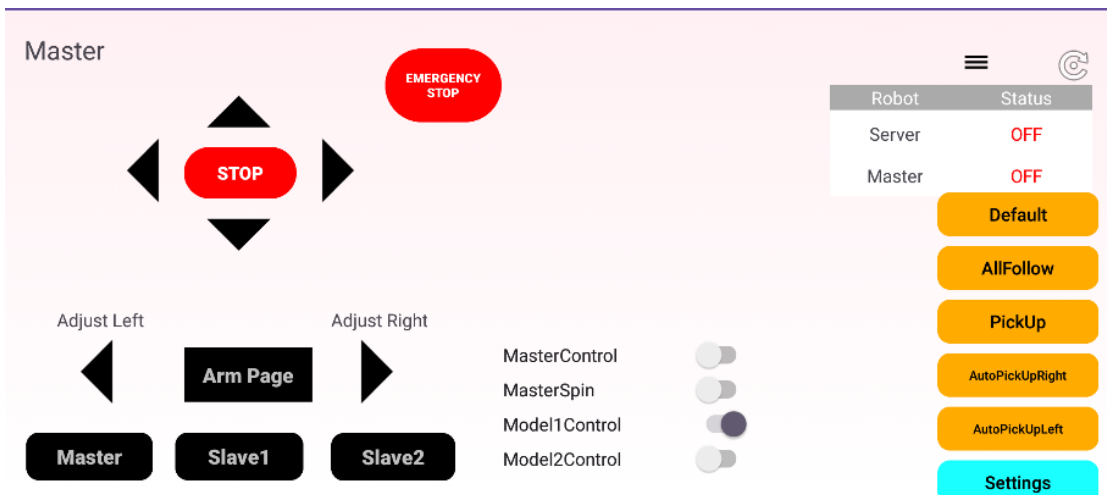


Figure 4.4.1 Master Page

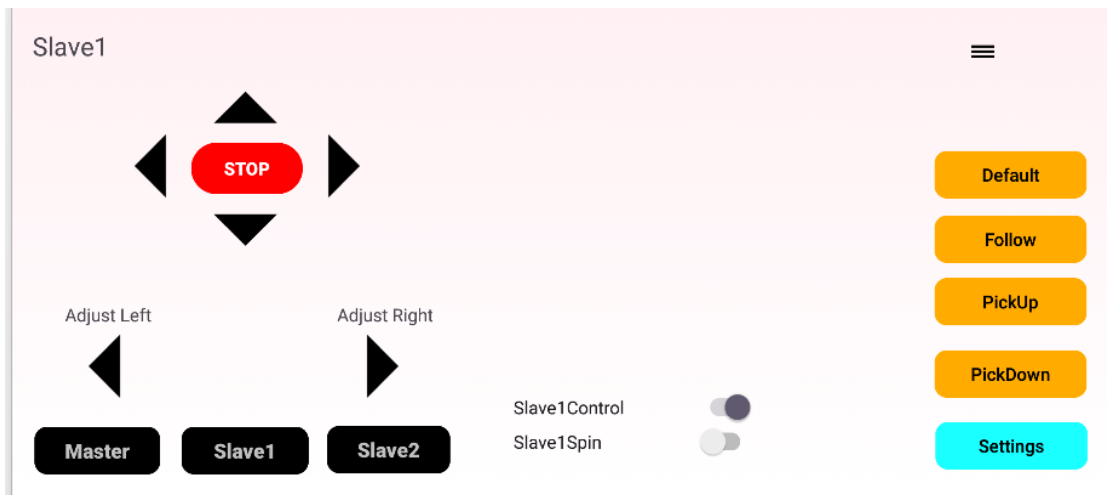


Figure 4.4.2 Slave Page

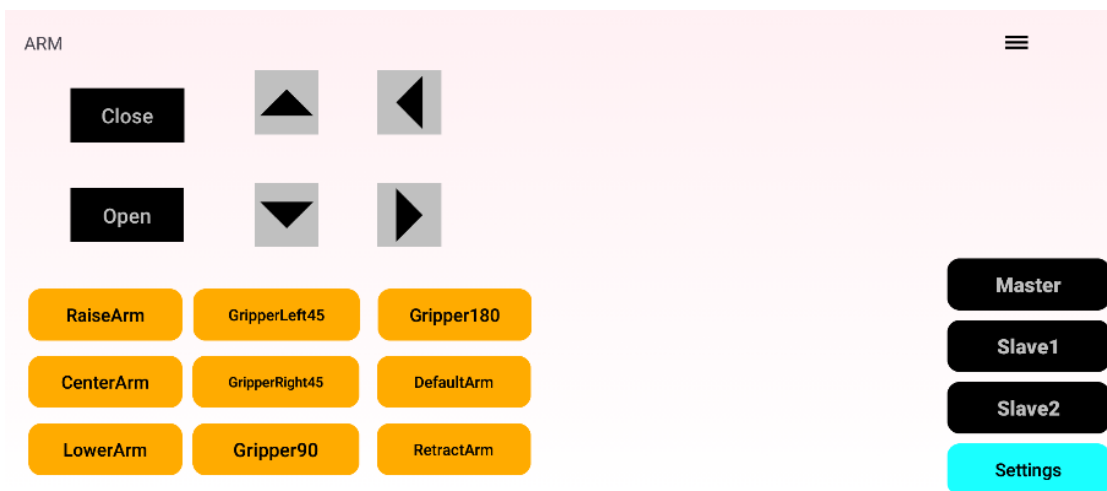


Figure 4.4.3 Arm Page

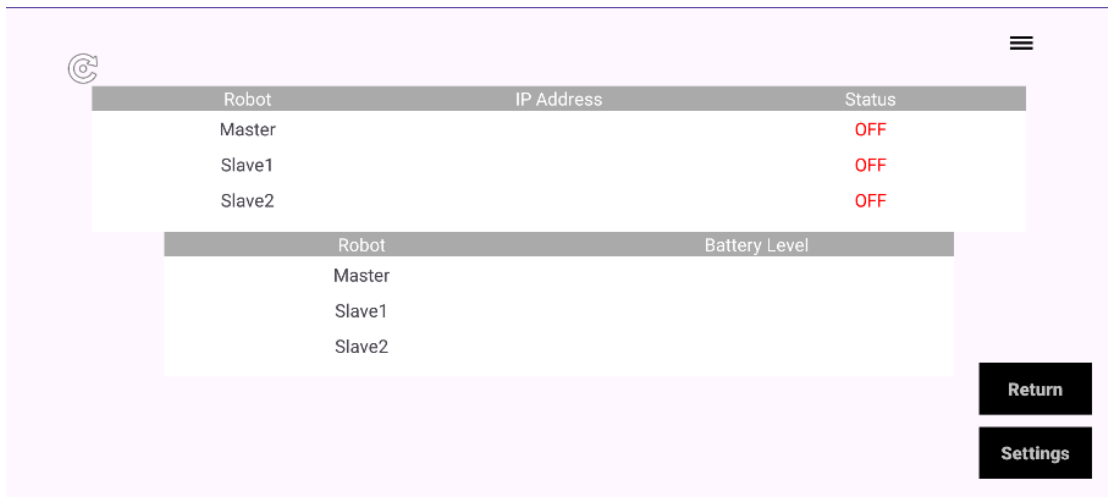


Figure 4.4.4 Status Page

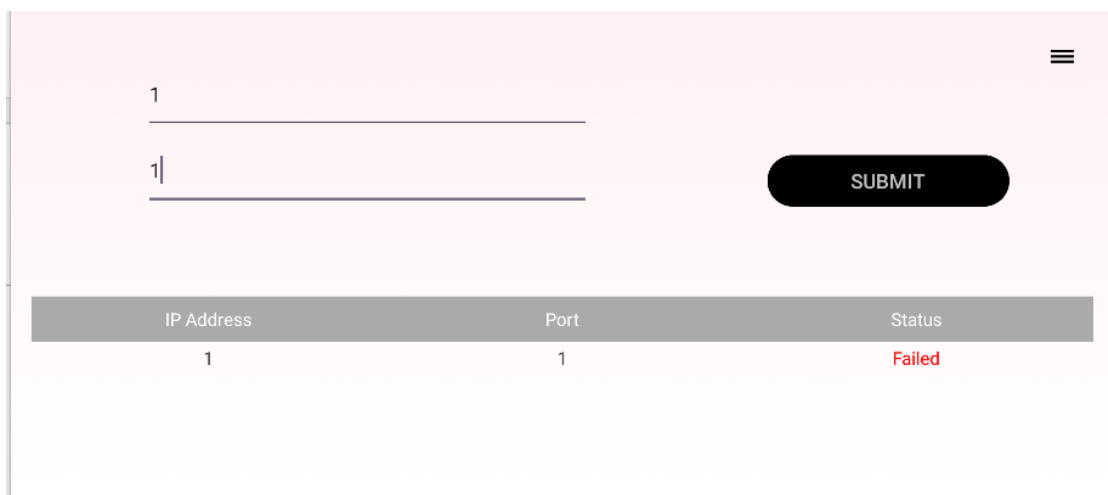


Figure 4.4.5 Settings Page

This image is a high-fidelity wireframe of the mobile app for FYP2, which is a 6-page app. The figure 4.4.1 is page is the main or master page, responsible for the overall control of the master robot as well as individual robots. The figure 4.4.2 is slave page has two variations, focusing on the specific functions and movement control of the slave robots. The figure 4.4.3 arm page is dedicated to controlling the master robot's arm. The figure 4.4.4 is status page displays information such as the robot's IP address, online/offline status, and battery level. Lastly, the figure 4.4.5 is settings page let users to connect to a specific IP address.

CHAPTER 5

System Implementation

5.1 Hardware Setup

5.1.1 GoPiGo3 Default Setup

The GoPiGo3 is a robot designed by Dexter Industries. By default, a few fundamental parts are used in its construction. It connects a Raspberry Pi, encoder motors, servo motors, and an ultrasonic sensor among other components using the GoPiGo3 board. Through its 40-pin connector, the GoPiGo3 board is connected to the Raspberry Pi, and it interfaces with the ultrasonic sensor via I2C. The servo port is used to connect the servo motor, while the motor ports are used to connect the left and right encoder motors. The GoPiGo3 board does not use all 40 of the Raspberry Pi's pins. To find out the pins the GoPiGo3 board uses, examine the hardware components section x.x.x.

5.1.2 Master GoPiGo3

5.1.2.1 Servo MG996R and Robot Gripper



Figure 5.1.2.1 Servo MG996R and Robot Gripper

The robot gripper has two holes on the gear to install together with the MG996R servo and the metal servo motor hub. The servo needs to be adjusted to 120 degrees and close the gripper to install it. At 60 degrees, the gripper will start to open, and at 120 degrees, it will be fully open. The servo is connected to the PCA9685 servo driver board for control.

5.1.2.2 Servo MG996R and Bracket

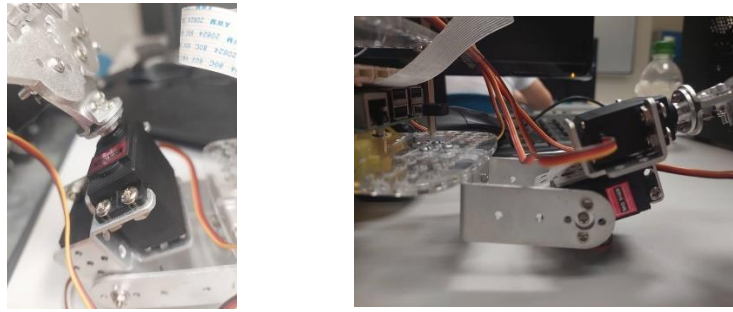


Figure 5.1.2.2 Servo MG996R and Bracket

This project utilizes two brackets to allow the arm to move up, down, left, and right. For the up and down motion, an MG996R servo is used, along with a metal servo motor hub installed on a multi-functional servo bracket. Next setup is install to a U-shaped bracket, enabling vertical movement. During installation, the servo is set to a default of 90 degrees, with the arm positioned downward. When the servo is near 0 degrees, the arm moves upward, and at 90 degrees, it moves downward. The servo cannot exceed 90 degrees, as that is the maximum downward range.

For left and right movement, the robot gripper has two holes on the back also install MG996R servo with a metal servo motor hub . This assembly is installing on a multi-functional servo bracket, which is locked together with the up-down servo bracket. The servo must be adjusted to 0 degrees to align the gripper in a parallel position. At 0 and 180 degrees, the robot gripper will be parallel, while at 90 degrees, it will be vertical. At 45 degrees and 135 degrees, the gripper will be at 45 degrees left and right, respectively. Both servo connects to the PCA9685 servo control board.

5.1.2.3 USB Network Adapter

Simply plug in a USB network adapter to one of the available USB ports on the Raspberry Pi 4 to utilise it. The Raspberry Pi ought should recognise the adapter on its own when it is plugged in.

5.1.2.4 Servo Board PCA9685 with Servo MG996R, Raspberry Pi, and Battery Casing

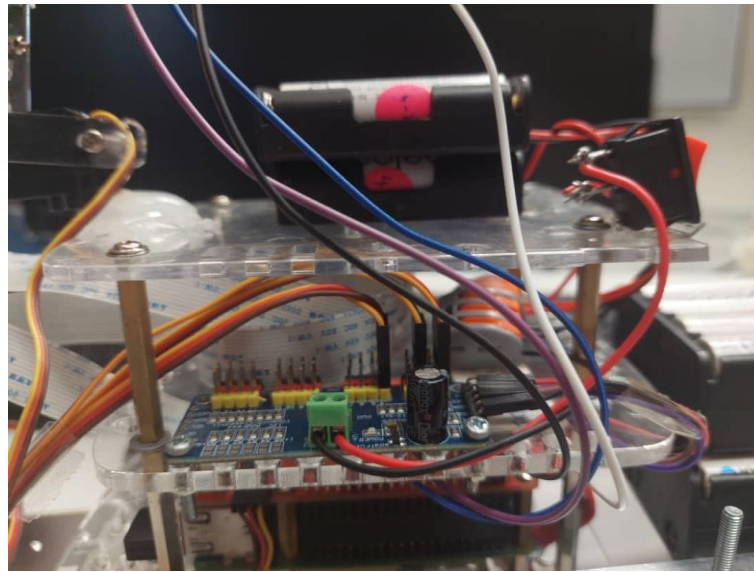


Figure 5.1.2.4 Servo Board PCA9685 with Servo MG996R, Raspberry Pi, and Battery Casing

The PCA9685 servo driver has two separate setups, one for writing inputs to the PCA9685 board and another for powering the servos. The write input setup requires 4 pins: power, ground, SDA, and SCL. For power, connect the Raspberry Pi's 5V pin (pin 2) and ground (pin 6). SDA is connected to GPIO2 (pin 3), and SCL to GPIO3 (pin 5). For the servo power supply, use a battery pack containing 4 AA batteries connected in parallel to increase the current available for the servos. The positive and negative terminals of the battery pack are connected to the servo power and ground, respectively.

5.1.3 Slave GoPiGo3

5.1.3.1 Servo MG996R with 3D Printer Objects, and Battery Casing

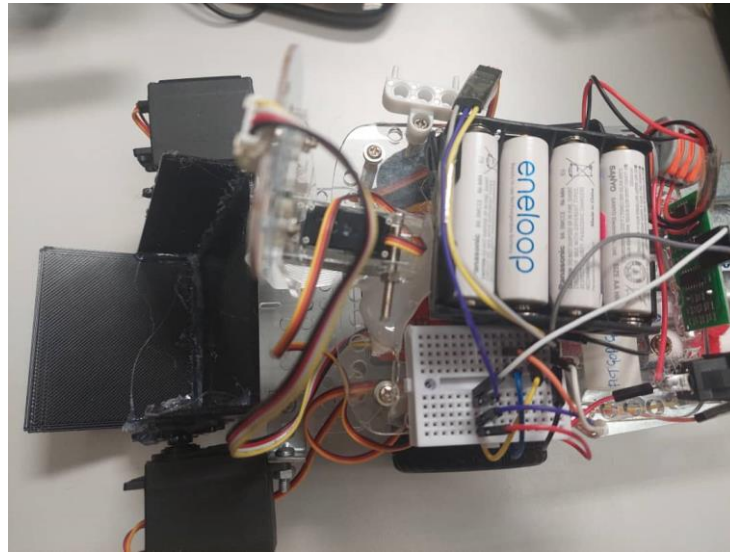


Figure 5.1.3.1 Servo MG996R with 3D Printer Objects, and Battery Casing

A slave robot is built using two MG996R servos and a 3D-printed L-shaped platform to be the arm. The 3D-printed arm is a simple design, and two servos are installed to the arm by using gears. The servos are glued to the left and right sides of the 3D-printed object, enabling the arm to move up and down. Before gluing the servos, they must be adjusted to a neutral position at 90 degrees. When one servo is set to 0 degrees and the other set to 100 degrees, the arm moves upward because the left and right servos rotate in opposite directions (clockwise and counterclockwise). Similarly, when one servo is set to 100 degrees and the other to 0 degrees, the arm moves downward.

For control, an external battery case with 4 AA batteries connected in parallel powers the system. The ground is shared between Raspberry Pi pin 25 and the battery case's negative terminal. The servos are controlled using GPIO pins 5 (pin 29) and 6 (pin 31) to adjust their angles. It's important to connect the ground of pin 25 and the battery case's negative terminal to ensure the servos' operation. Pin 25 allows GPIO5 and GPIO6 to function correctly by completing the circuit, and the negative terminal of the battery case works with the positive terminal to form the power circuit.

5.2 Software Setup

5.2.1 Raspberry Pi Customize Operating System Install

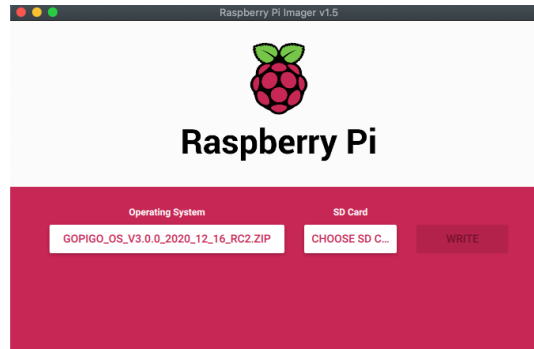


Figure 5.2.1 Raspberry Pi Connect

A specially created operating system (OS) for the Raspberry Pi and GoPiGo3 board, created by Dexter Industries, is required for the GoPiGo3 robot. This is the only operating system that works with the GoPiGo3 library. Download the Raspberry Pi image and flash it onto an SD card to install. The OS image can get at <https://gopigo.io/gopigo-os-v-3-0-3/>.

5.2.2 Master Modified GoPiGo3 System

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
    ssid="RobotNetwork"
    psk="raspberrypi"
    key_mgmt=WPA-PSK
    disabled=1
}

network={
    ssid="dlink-11EC-5GHz"
    psk="uqwzc31800"
    key_mgmt=WPA-PSK
}

network={
    ssid="Blank"
    psk="11411141abxyz"
    key_mgmt=WPA-PSK
}
```

Figure 5.2.2.1 wpa_supplicant.conf

```
pi@MasterGoPiGo:~ $ sudo rm /etc/dnsmasq.d/
cinch.conf README
pi@MasterGoPiGo:~ $ sudo rm /etc/dnsmasq.d/cinch.conf
```

Figure 5.2.2.2 cinch.conf

The master node requires an access point, but the default GoPiGo3 setup includes an access point without password by default. To configure the USB network adapter, first detect the USB interface. Next, modify the wpa_supplicant.conf file to set the network SSID and password like figure 5.2.2.1. Last, restart both the dhcpcd and wpa_supplicant services. Once completed, the USB adapter will obtain a DHCP IP address from the network router.

When want to install or sudo update need to remove one config file like 5.2.2.2. After remove using sudo systemctl restart dnsmasq.service to restart the DNS service. When install pip install adafruit-blinka, pip install adafruit-circuitpython-motor and pip install adafruit-circuitpython-pca9685 to install library of PCA9685

5.2.3 Slave Modified GoPiGo3 System

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
    ssid="Blank"
    psk="11411141abxyz"
    key_mgmt=WPA-PSK
    disabled=1
}

network={
    ssid="MasterGoPiGo"
    key_mgmt=NONE
}
```

Figure 5.2.3.1 wpa_supplicant.conf

```
interface wlan0
static ip_address=10.10.10.1/24
static routers=10.10.10.10
```

Figure 5.2.3.2 wpa_supplicant.conf

The slave need to disable the access point on the slave, stop the `start_as_access_point.service`, which will stop the GoPiGo3 access point service. Then, configure a static IP address in the `dhcpcd.conf` file within the range of `10.10.10.0/24` like the figure 5.2.3.2,. Modify the `wpa_supplicant.conf` file to set the network SSID and password-like figure 5.2.3.1. Finally, restart both the `dhcpcd` and `wpa_supplicant` services. Alternatively, the Wi-Fi network can be selected by clicking the Wi-Fi icon.

5.2.4 Middleware Server installation

The middleware server requires a public IP address to allow the master, slave, and mobile devices to connect. The server OS is Ubuntu and installation of Python, pyhon need socket, threading, and queue libraries. Access the server via the default SSH to code the Python script.

5.3 Setting and Configuration

5.3.1 Middleware Server code

```

"""Handle communication with a single client."""
try:
    while True:
        # Receive message from the client
        message = client_socket.recv(4096).decode('utf-8')
        if not message:
            # Client disconnected
            break
        print(f"Received from {client_id}: {message}")

        # Add the message to the queue
        message_queue.put((client_id, message))
except ConnectionResetError:
    # Handle client connection reset
    pass
finally:
    # Ensure the client is removed and socket is closed
    if client_id in clients:
        del clients[client_id]
    client_socket.close()
    print(f"Client {client_id} disconnected")

def broadcast_messages(clients, message_queue):
    """Broadcast messages from the queue to other clients."""
    while True:
        client_id, message = message_queue.get()
        # Forward the message to all other clients
        for id, conn in clients.items():
            if id != client_id:
                try:
                    conn.send(message.encode('utf-8'))
                except BrokenPipeError:
                    # Handle broken pipe error if a client disconnects suddenly
                    pass
        message_queue.task_done()

def start_server_on_port(port, clients, message_queue):
    """Start the server on a specific port and accept incoming client connections."""
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((HOST, port))
    server_socket.listen(2) # Limit to 2 clients

    print(f"Server started on port {port}, waiting for connections...")

    # Start the message broadcasting thread
    threading.Thread(target=broadcast_messages, args=(clients, message_queue), daemon=True).start()

    client_id_counter = 1
    while True:
        client_socket, addr = server_socket.accept()
        # Assign a new client ID
        client_id = client_id_counter
        client_id_counter += 1
        clients[client_id] = client_socket
        print(f"Client {client_id} connected from {addr} on port {port}")

        sendmessage = "H!\n"
        client_socket.send(sendmessage.encode('utf-8'))
        print("send H! success")

        # Start a new thread for each client
        client_thread = threading.Thread(target=handle_client, args=(client_socket, client_id, clients, message_queue))
        client_thread.start()

```

Figure 5.3.1 Middleware Server code

This code 5.3.1 is middleware code by using Python. The Python script is said about how to create a server that listens on port 1317 at the public IP address 103.159.133.241 by passing the message to the master GoPiGo3 robot and the mobile application. The `start_server_on_port` function creates a server socket using `socket.socket()`, binds it to the specified IP and port with `bind((HOST, port))`, and listens for incoming connections using `listen()`. When a client (either the robot or the mobile application) connects, the server accepts the connection with `accept()` and assigns the client a unique ID. A new thread is created for each client using the `handle_client` function, which continuously listens for messages from that client via `recv()`. These messages are placed into a queue (`message_queue`). The `broadcast_messages` function monitors the queue and forwards the messages to the appropriate clients (excluding the sender) via `send()`. This setup allows the master robot and the mobile application to communicate in real-time.

5.3.2 Master Main

```
def main():
    server_thread = threading.Thread(target=start_server)
    server_thread.start()

    connect_server_thread = threading.Thread(target=connect_server)
    connect_server_thread.start()

    ultrasound_thread = threading.Thread(target=detect_ultrasound)
    ultrasound_thread.start()

    statusupdate_thread = threading.Thread(target=statusupdate)
    statusupdate_thread.start()

    server_thread.join()
    connect_server_thread.join()
    ultrasound_thread.join()
    statusupdate_thread.join()

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        GPIO.cleanup()
```

Figure 5.3.2 Master Main code

The master code launches four threads to do different tasks. The four threads is **start_server 5.3.3**, **connect_server 5.3.4**, **detect_ultrasound 5.3.14**, and **statusupdate 5.3.13**. Each task is assigned to a thread and started to run simultaneously. The try is to detect a KeyboardInterrupt (such as pressing Ctrl+C to stop the script)

5.3.3 Master listen slave connect

```
def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_address = ('10.10.10.10', 10102)
    server_socket.bind(server_address)
    server_socket.listen(5)
    print(f"Server listening on {server_address}")

    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Accepted connection from {client_address}")
        ip_address_array.append(client_address)
        slave_client[client_address[0]] = client_socket

        client_receive_thread = threading.Thread(target=server_receive_data, args=(client_socket,))
        client_send_thread = threading.Thread(target=server_send_data, args=(client_socket,))
        client_receive_thread.start()
        client_send_thread.start()
```

Figure 5.3.3 Master Listen Slave Connect code

This code shows how the master creates a socket and accept the from the slave, the will listen for incoming connections on a specified IP address (10.10.10.10) and port (10102). When a slave connects, the server accepts the connection, it will store slave's IP address in ip_address_array and the slave socket in the slave_client dictionary. For each connected slave, two threads are started: one to handle receiving data (**server_receive_data**) and another for sending data (**server_send_data**).

5.3.3.1 Master receive from slave

```
def server_receive_data(client_socket):
    global ip_address_array, message, client1_message, client2_message, send_mobile_message, client_done_message
    while True:
        try:
            data = client_socket.recv(1024)
            if not data:
                print("Connection closed by client")
                client_socket.close()
                break
            received_message = data.decode('utf-8')
            print("received_message")
            if "Slave1" in received_message:
                print("Slave1re" + received_message)
                if(received_message == "Slave1done"):
                    client_done_message = "slavedone"
                    print("this is " + client_done_message)
                else :
                    send_mobile_message = received_message
            elif "Slave2" in received_message:
                print("Slave2re" + received_message)
                if(received_message == "Slave2done"):
                    client_done_message = "slavedone"
                    print("this is " + client_done_message)
                else :
                    send_mobile_message = received_message

            message = received_message
        except Exception as e:
            print(f"Error receiving data: {e}")
            break
```

Figure 5.3.3.1 Master Receive From Slave code

This is one of the threads of the master to communicate with the slave. This thread's main purpose is receiving data from a connected slave socket in a continuous loop. If no data is received, meaning the slave has closed the connection, then it will close the client socket and exit the loop.

When data is received, it is decoded from bytes to a UTF-8 string and stored in the variable `received_message`. A typical slave will send a message to the master with a key in the format "**slave + number**" to identify which slave the message comes from. After receiving the message, the thread checks whether it is from "Slave1" or "Slave2." If the message indicates, like the key word "**slave1done**" or "**slave2done**," that a slave device is done for some purpose. The system then verifies if the task has been fully completed. If further processing is required, the slave's message is stored in `send_mobile_message`. This variable is then used by another thread to send the message to the middleware server.

5.3.3.2 Master send to slave

```

def server_send_data(client_socket):
    global message, client1_message, client2_message, send_all_client_message, ip_address_array, slave_client, pickup_item
    while True:
        if send_all_client_message:
            try:
                print("Send" + send_all_client_message)
                if send_all_client_message == "stop":
                    client_socket.sendall(send_all_client_message.encode('utf-8'))
                    print("i stop it")
                    send_all_client_message = ""
                elif send_all_client_message in ["modellforward", "stop", "modellback", "modellleft", "modellright"]:
                    client_socket.sendall(send_all_client_message.encode('utf-8'))
                    send_all_client_message = ""
                elif send_all_client_message in ["model2forward", "stop", "model2back", "model2left", "model2right"]:
                    client1_message = client2_message = send_all_client_message
                    send_all_client_message = ""
                    process_client_messages1(SlaveIPAddress, client1_message, 30)
                    print("ok")
                elif send_all_client_message == "allfollow":
                    client1_message = client2_message = send_all_client_message
                    send_all_client_message = ""
                    process_client_messages1(SlaveIPAddress, client1_message, 30)
                    #process_client_messages1()
                    client1_message = client2_message = ""
                    print("Reset complete")
                elif send_all_client_message in ["autopickupleft", "autopickupright"]:
                    client1_message = client2_message = send_all_client_message
                    send_all_client_message = ""
                    pickup_item = pickup_item + 1

                    if(pickup_item == 1):
                        process_client_messages1(SlaveIPAddress, client1_message, 30)
                    #elif (pickup_item == 2)
                    #elif(pickup_item == 3)
                elif send_all_client_message == "testing":
                    client1_message = client2_message = send_all_client_message
                    send_all_client_message = ""
                    process_client_messages1(SlaveIPAddress, client1_message, 30)
                    print("Slave complet")
                else :
                    client_socket.sendall(send_all_client_message.encode('utf-8'))
                    send_all_client_message = ""

                time.sleep(0.1)
            except Exception as e:
                print(f"Error sending data: {e}")
                break

```

Figure 5.3.3.2.1 Master Send to Slave code

```

def process_client_messages1(IPAddress, message, timeout):
    global ip_address_array, slave_client, client_done_message, engrystop

    for client_ip_address in ip_address_array:
        if client_ip_address[0] in slave_client:
            print(client_ip_address[0])
            if client_ip_address[0] == IPAddress:
                print("Message already sent to", IPAddress)
                slave_client[IPAddress].send(message.encode('utf-8'))

    start_time = time.time()

    # Loop until "slavedone" is received or the timeout occurs
    while client_done_message != "slavedone":
        print("Waiting for done message...")

        if engrystop == "stop":
            print("Processing stopped by user")
            break

        if time.time() - start_time > timeout:
            print("Timeout occurred")
            break

        time.sleep(0.1) # Sleep for 100ms to prevent CPU overuse

    client_done_message = "" # Reset done message for the next process

```

Figure 5.3.3.2.2 Process_Client_Message 1code

This is one of the threads of the master to communicate with the slave. This thread's main purpose is to send data from a connected slave socket in a continuous loop when it has a message. **Send_all_client_message** is a variable from the middleware server that can also call mobile applications. If the `send_all_client_message` does not have data, it will loop back until the `send_all_client_message` has a message.

When a message is present in `send_all_client_message`, the thread checks its content and sends the corresponding command to the client socket, such as "stop," "model1forward," "model2back," or "allfollow." Each command triggers a specific action: for example, "model1forward" makes the slave move alongside the master, while "allfollow" instructs all slave robots to approach the master robot. To broadcast a command to all slaves, it uses **`client_socket.sendall(send_all_client_message.encode('utf-8'))`**. In some cases, the thread sends a message to a single client and waits for a response containing the keyword "slavedone." The function **`process_client_messages1`** 5.3.3.2.2 is used to block this thread until one of the slaves completes its task. Before sending a message, the function checks if the client's IP is listed in the `slave_client` dictionary to ensure it hasn't been sent already.

5.3.4 Master connect middleware server socket code

```
def connect_server():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('103.159.133.241', 1317)
    client_socket.connect(server_address)
    print(f"Connected to server at {server_address}")

    receive_thread = threading.Thread(target=receive_data, args=(client_socket,))
    send_thread = threading.Thread(target=send_data, args=(client_socket,))
    receive_thread.start()
    send_thread.start()
```

Figure 5.3.4 connect_server code

This thread enables the master robot to communicate with a mobile device through a middleware server using TCP socket programming. The connection is established with the middleware server at IP address 103.159.133.241 and port 1317. Once connected, two threads are created: one for **sending data** to the middleware server (which also serves as the mobile application) and another for **receiving data** from it. These threads is let aster robot and the mobile device can do bidirectional communication.

5.3.4.1 Master send to middleware server

```
def send_data(server_socket):
    global send_mobile_message
    while True:
        if send_mobile_message:
            try:
                server_socket.sendall(send_mobile_message.encode('utf-8'))
                print("i had send " + send_mobile_message )
                time.sleep(0.05)
                send_mobile_message = ""
            except Exception as e:
                print(f"Error sending data: {e}")
                break
    server_socket.close()
```

Figure 5.3.4.1 send_data code

This thread is a loop sends data from the master robot to the connected middleware server socket. If any messages are stored in the send_mobile_message variable, it attempts to encode and send them to the server. To prevent rapid repeated sending, wait 0.05 seconds, then clear the send_mobile_message variable.

5.3.4.2 Master receive form middleware server

```

def receive_data(server_socket):
    global Client1_message, client2_message, send_all_client_message, client_done_message, engr:
    while True:
        try:
            print("i got try")
            data = server_socket.recv(1024)
            if not data:
                print("Connection closed by server")
                break
            received_message = data.decode('utf-8').strip()
            print(received_message)
            if received_message == "EMstop":
                engrystop = received_message
            elif received_message == "ClearEMstop":
                engrystop = ""
            elif received_message == "stop":
                print("this got")
                send_all_client_message = received_message
                Mastermovement.movmentofrobotModell(received_message)
            elif received_message in ["masterforward", "stop", "masterback", "masterleft", "m:
                print("hello")
                Mastermovement.movmentofrobotmaster(received_message)

            elif received_message in ["masterspinforward", "stop", "masterspinback", "mastersp:
                if(is_busy == False):
                    is_bust = True
                    print("gi")
                    time.sleep(0.1)
                    Mastermovement.spinmovement(received_message)
                    is_bust = False
                else:
                    received_message = ""
            elif received_message in ["modellforward", "stop", "modellback", "modellleft", "m:
                send_all_client_message = received_message
                Mastermovement.movmentofrobotModell(received_message)
            elif received_message in ["model2left", "model2right", "model2back", "model2forwa:
                degree_value2 = Mastermovement.GetModel2Degree()
                print(degree_value2)
                Mastermovement.movmentofrobotModel2(received_message)
                time.sleep(0.5)
                send_all_client_message = received_message
            elif received_message in ["masteradjustleft", "masteradjustright"]:
                Mastermovement.movmentofrobotAdjust(received_message)
            elif received_message in ["openarm", "closearm", "uparm", "downarm", "leftarm", "right:
                Mastermovement.armmovment(received_message)
            elif received_message == "default":
                Mastermovement.defaultsetting()
                pickup_item = 0
            elif received_message == "allfollow":
                Mastermovement.defaultsetting()
                send_all_client_message = received_message
            elif received_message in ["autopickupleft", "autopickupright"]:
                Mastermovement.armmovment("masterpickup")
                send_all_client_message = received_message
            elif received_message == "masterreset":
                Mastermovement.defaultsetting()
            else :
                print("this is else")
                send_all_client_message = received_message
            print(f"Received from server: {received_message}")

```

Figure 5.3.4.2 receive_data code

This thread continuously handles incoming data from the middleware server socket in a loop. It waits to receive data in 1024-byte chunks. When data is received, it is decoded and processed according to the message. If the message is intended for the master robot, it passes the request to the MasterMovement class (5.3.7) to execute the appropriate master function. For commands involving both the master and slave, the thread processes the command and stores the message in send_all_client_message, allowing another thread to send it to the slave device. Last if message not master need do process ,it store the message into send_all_client_message .

5.3.5 Master Servo board PCA9685

```
from adafruit_motor import servo
from adafruit_pca9685 import PCA9685
import time

# Create the I2C bus interface
i2c = busio.I2C(board.SCL, board.SDA)

# Create a PCA9685 class instance and set the PWM frequency to 50Hz, suitable fo
pca = PCA9685(i2c)
pca.frequency = 50

# Create servo objects with custom pulse widths for MG996R servos
servo1 = servo.Servo(pca.channels[0], min_pulse=500, max_pulse=2500)
servo2 = servo.Servo(pca.channels[2], min_pulse=500, max_pulse=2500)
servo3 = servo.Servo(pca.channels[4], min_pulse=500, max_pulse=2500)
servo4 = servo.Servo(pca.channels[6], min_pulse=500, max_pulse=2500)
servo5 = servo.Servo(pca.channels[8], min_pulse=500, max_pulse=2500)
```

Figure 5.3.5 Master Servo board PCA9685 code

The 5.3.5 is the code configures a PCA9685 PWM driver to control three MG996R servos. These servo set at a 50Hz frequency. Servos are connected to channels 0, 2, and 4, with pulse widths ranging from 500 to 2500 microseconds. Channel 0 controls the up and down movement of the servo, channel 2 manages the left and right motion of the arm, and channel 4 adjusts the arm's opening and closing. For example, setting `servo1.angle = 0` positions the servo on channel 0 at 0 degrees.

5.3.6 Master Arm code

```

def arm1function(movedegree):
    global arm1_degree

    movedegree = int(movedegree) # Ensure movedegree is an integer

    print(movedegree)
    if movedegree <= -1 or movedegree >= 81:
        print ("invalid")

    elif arm1_degree < movedegree:
        while arm1_degree < movedegree:
            print(f"Current value: {arm1_degree}")
            arm1_degree += 5
            servo1.angle = arm1_degree
            time.sleep(0.1)
            print(f"Final value: {arm1_degree}")

    elif arm1_degree > movedegree:
        while arm1_degree > movedegree:
            print(f"Current value: {arm1_degree}")
            arm1_degree -= 5
            servo1.angle = arm1_degree
            time.sleep(0.1)
            print(f"Final value: {arm1_degree}")

    else:
        print ("Arm is already at the target degree.")

```

Figure 5.3.6.1 arm1funcios code

```

global arm1_degree ,arm2_degree
if(data_movement == "openarm"):
    servo3.angle = 60
elif(data_movement == "closearm"):
    servo3.angle = 120
elif(data_movement == "uparm"):
    movedegree = arm1_degree - 10
    arm1function(movedegree)
elif(data_movement == "downarm"):
    movedegree = arm1_degree + 10
    arm1function(movedegree)
elif(data_movement == "rightarm"):
    movedegree = arm2_degree + 10
    print(movedegree)
    if movedegree > 0 and movedegree <=180 :
        if(movedegree > 180):
            movedegree = 180

            arm2_degree = movedegree
            servo2.angle = arm2_degree
elif(data_movement == "leftarm"):
    movedegree = arm2_degree - 10

    if movedegree >= 0 and movedegree <=180 :
        if(movedegree > 0):
            movedegree = 0

            arm2_degree = movedegree
            print(arm2_degree)
            servo2.angle = arm2_degree
elif(data_movement == "raisearm"):
    arm1function(10)
elif(data_movement == "centerarm"):
    arm1function(45)
elif(data_movement == "lowerarm"):
    arm1function(50)
elif(data_movement == "gripperleft45"):
    servo2.angle = 45
elif(data_movement == "gripperright45"):
    servo2.angle = 135
elif(data_movement == "gripper90"):
    servo2.angle = 90
elif(data_movement == "gripper180"):
    servo2.angle = 0
elif(data_movement == "defaultarm"):
    arm1function(45)
    time.sleep(0.5)
    servo2.angle = 0
    time.sleep(0.5)
    servo3.angle = 60
elif(data_movement == "retractarm"):
    servo2.angle = 0
    time.sleep(0.5)
    servo3.angle = 60
    time.sleep(0.5)
    arm1function(10)
elif(data_movement == "masterpickup"):
    servo3.angle = 120
    time.sleep(0.5)

```

Figure 5.3.6.2 arm code

Figure 5.3.6.1 is a function to let the channel 0 servo can move smooth up and down . This function moves degree between 0-to-80-degree ,0 is up and 80 is down The function adjusts the servo's in 5-degree increments to reach the target.

Figure 5.3.6.2 is a function to control the degree of the servos, like servos 1, 2, and 3, to perform different actions. Such as opening or closing the arm, raising or lowering it with the 5.3.6.1 function, and controlling the gripper's arm. The function gives some fixed command to let the arm adjust direct. For example, "defaultarm" to set the servo to 1 45 degree, 2 0 degree, and 3 60 degrees mean open the arm.

5.3.7 Master Movement code

```

def movmentofrobotmaster (data_movement):
    global degree_value
    #gpg.set_speed(150)
    if (data_movement == "stop"):
        gpg.stop()
        gpg.set_speed(100)
        gpg.turn_degrees(degree_value)
        time.sleep(0.5)
        degree_value = 0
    elif (data_movement == "masterforward"):
        gpg.set_speed(150)
        gpg.forward()
    elif (data_movement == "masterback"):
        gpg.backward()
    elif (data_movement == "masterleft"):
        gpg.set_speed(100)
        gpg.turn_degrees(-90)
        degree_value = degree_value + 90
        time.sleep(0.5)
        gpg.set_speed(150)
        gpg.forward()
    elif (data_movement == "masterright"):
        gpg.set_speed(100)
        gpg.turn_degrees(90)
        degree_value = degree_value - 90
        time.sleep(0.5)
        gpg.set_speed(150)
        gpg.forward()

def spinmovement (data_movement):
    global spin_degree
    gpg.set_speed(100)
    if (data_movement == "masterspinforward"):
        if (spin_degree == 90):
            gpg.turn_degrees(-90)
            spin_degree = 0

        elif (spin_degree == -90):
            gpg.turn_degrees(90)
            spin_degree = 0

        elif (spin_degree == 180):
            gpg.turn_degrees(-180)
            spin_degree = 0

    elif (data_movement == "masterspinleft"):
        if (spin_degree == 0):
            gpg.turn_degrees(-90)
            spin_degree = -90

        elif (spin_degree == 90):
            gpg.turn_degrees(-180)
            spin_degree = -90

        elif (spin_degree == 180):
            gpg.turn_degrees(-90)
            spin_degree = 0

    elif (spin_degree == -90):
        gpg.turn_degrees(90)
        spin_degree = 0

    elif (spin_degree == 180):
        gpg.turn_degrees(-90)
        spin_degree = 0

def movmentofrobotModel1 (data_movement):
    global degree_value
    #gpg.set_speed(150)
    if (data_movement == "stop"):
        gpg.stop()
        gpg.set_speed(100)
        gpg.turn_degrees(degree_value)
        time.sleep(0.5)
        degree_value = 0
    elif (data_movement == "modellforward"):
        gpg.set_speed(150)
        gpg.forward()
    elif (data_movement == "modellback"):
        gpg.backward()
    elif (data_movement == "modellleft"):
        gpg.set_speed(100)
        gpg.turn_degrees(-90)
        degree_value = degree_value + 90
        time.sleep(0.5)
        gpg.set_speed(150)
        gpg.forward()
    elif (data_movement == "modellright"):
        gpg.set_speed(100)
        gpg.turn_degrees(90)
        degree_value = degree_value - 90
        time.sleep(0.5)
        gpg.set_speed(150)
        gpg.forward()

def movmentofrobotAdjust (data_movement):
    gpg.set_speed(150)
    if (data_movement == "masteradjustleft"):
        gpg.left()
        time.sleep(0.1)
        gpg.stop()
    elif (data_movement == "masteradjustright"):
        gpg.right()
        time.sleep(0.1)
        gpg.stop()

def defaultsetting():
    global arm1_degree, arm2_degree
    gpg.set_speed(150)
    servo.rotate_servo(90)
    arm2_degree = 0
    arm1function(45)
    time.sleep(1)
    servo2.angle = arm2_degree
    time.sleep(1)
    servo3.angle = 60

```

Figure 5.3.7 Master Movement code

This Python script is supporting the main Master Python script can call to handle various actions. It includes functions like model1 forward, adjustleft, spinleft, and others function. Function movmentofrobotmaster is controls the master robot's self-movement, and spinmovement, which allows it to spin independently. Other functions like movmentofrobotModel1 dictate specific movement patterns. These modular functions help the main Master Python script keeping it clear .

5.3.8 Slave main script

```
def main():
    ultrasound_thread = threading.Thread(target=detect_ultrasound)

    connect_server_thread = threading.Thread(target=connect_server)

    statusupdate_thread = threading.Thread(target=statusupdate)

    connect_server_thread.start()
    ultrasound_thread.start()
    statusupdate_thread.start()

    connect_server_thread.join()
    ultrasound_thread.join()

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        Slavemovmet.cleanall()
```

Figure 5.3.8 Slave Main code

The slave code launches three threads to do different tasks. The three threads is **connect_server 5.3.9**, **detect_ultrasound 5.3.14**, and **statusupdate 5.3.13**. Each task is assigned to a thread and started to run simultaneously. The try is to detect a KeyboardInterrupt such as pressing Ctrl+C to stop the script.

5.3.9 Slave connect to Master

```
def connect_server():
    # Create a socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect to the server
    server_address = ('10.10.10.10', 10102)
    client_socket.connect(server_address)
    print(f"Connected to server at {server_address}")

    # Start threads for sending and receiving data
    receive_thread = threading.Thread(target=receive_data, args=(client_socket,))
    send_thread = threading.Thread(target=send_data, args=(client_socket,))
    receive_thread.start()
    send_thread.start()
```

Figure 5.3.9 connect_server code

This code connects to the master robot by using a TCP socket. The connection is made to the IP address 10.10.10.10 on port 10102 because the master robot is listening these IP address and port. Once connected, two threads are started receive_thread and send_thread. The receive_thread is receiving messages from the master robot, while the send_thread handles sending messages to the master robot. Concurrent operation of both threads enables two-way, real-time communication.

5.3.9.1 Slave receive form Master

```

def receive_data(server_socket):
    global ultrasound_distance
    global messagetomaster
    global distance_value,detect_value
    while True:
        try:
            #Here will not the process ,wait the rec
            data = server_socket.recv(1024)
            print(data)
            if not data:
                print("Connection closed by server")
                break
            else:
                received_message = data.decode('utf-8')
                print("received" + received_message)
                if received_message in ["modellforward","stop", "modellback",
                    Slavemovmet.movementofrobotModel1(received_message)
                elif received_message in ["slaveistop","slaveiforward","slaveib
                    Slavemovmet.movementofrobotslavel(received_message)
                elif received_message in ["slavelapinforward","slavelapinback",
                    Slavemovmet.spinmovement(received_message)
                elif received_message in ["model2forward","stop", "model2back"
                    print("ok")
                    Slavemovmet.movementofrobotModel2(received_message)
                    Slavemovmet.defaultsetting()
                    nearmaster(400)
                    messagetomaster = "Slavedone"
                elif received_message in ["slaveladjustleft","slaveladjustright
                    Slavemovmet.movementofrobotAdjust(received_message)
                elif received_message in ["slaveldefault","slavelpickup","slave
                    Slavemovmet.armmovement(received_message)
                elif received_message == "slavelfollow":
                    received_message = ""
                    Slavemovmet.defaultsetting()
                    time.sleep(0.2)
                    nearmaster(400)
                elif received_message == "allfollow":
                    received_message = ""
                    Slavemovmet.defaultsetting()
                    time.sleep(0.2)
                    nearmaster(400)
                    messagetomaster = "Slavedone"
                elif received_message == "detectlefttright":
                    detect_value = -250
                    a = Slavemovmet.detectlefttright()
                    print(a)
                elif received_message == "autopickupleft":
                    detect_value = -250
                    Slavemovmet.movepickupfunction("left")
                    gpg.set_speed(100)
                    nearmaster(165)
                    Slavemovmet.armmovement("slavelpickdown")
                    messagetomaster = "Slavedone"
                    time.sleep(5)
                    Slavemovmet.armmovement("slavelpickup")
                    detect_value = 250
                elif received_message == "autopickupright":
                    detect_value = -250
                    messagetomaster = "Slavedone"

                messagetomaster = "Slavedone"
                elif received_message == "detectlefttright":
                    detect_value = -250
                    a = Slavemovmet.detectlefttright()
                    print(a)
                elif received_message == "autopickupleft":
                    detect_value = -250
                    Slavemovmet.movepickupfunction("left")
                    gpg.set_speed(100)
                    nearmaster(165)
                    Slavemovmet.armmovement("slavelpickdown")
                    messagetomaster = "Slavedone"
                    time.sleep(5)
                    Slavemovmet.armmovement("slavelpickup")
                    detect_value = 250
                elif received_message == "autopickupright":
                    detect_value = -250
                    Slavemovmet.movepickupfunction("right")
                    gpg.set_speed(100)
                    nearmaster(165)
                    Slavemovmet.armmovement("slavelpickdown")
                    messagetomaster = "Slavedone"
                    time.sleep(5)
                    Slavemovmet.armmovement("slavelpickup")
                    detect_value = 250
                elif received_message == "testtakenear":
                    takefrommaster(165)
                elif received_message == "autoplcup":
                    detect_value = -250
                    a = Slavemovmet.detectlefttright()
                    print(a)
                    time.sleep(1)
                    detect_value = 250
                    if (a == "manual"):
                        messagetomaster = "Slavedone"
                        time.sleep(1)
                    else:
                        Slavemovmet.movepickupfunction(a)
                        detect_value = -250
                        gpg.set_speed(100)
                        nearmaster(165)
                        Slavemovmet.armmovement("slavelpickdown")
                        messagetomaster = "Slavedone"
                        time.sleep(5)
                        detect_value = 250
                        #time.sleep(1)
                        #Slavemovmet.pickupfunction()
                        #detect_value = 250
                elif received_message == "testing":
                    print("ok")
                    time.sleep(0.2)
                    messagetomaster = "Slavedone"

```

Figure 5.3.9.1 receive_thread code

This thread is similar to the master received from the middleware 5.3.4.2 ; it handles incoming data from the master socket in a loop. It waits to receive data in 1024-byte chunks. When data is received, it is decoded and processed according to the message. The function invokes multiple methods from the Slavemovmet module in response to incoming messages. These methods regulate the robot's movements and activities, including changing directions, repositioning itself, and carrying out particular tasks like picking up and placing items. Some process end will send a "Slavedone" message to the master to unlock the master. When need to send the message to master, will add the message to messagetomaster to let the send thread of slave do the sending.

5.3.9.2 Slave send to Master

```
# Function to handle sending data to the server
def send_data(server_socket):
    global messagetomaster
    while True:
        try:
            if messagetomaster:
                message = messagetomaster
                print(message)
                server_socket.sendall(message.encode())
                messagetomaster = ""
                time.sleep(0.05)
        except Exception as e:
            print(f"Error sending data: {e}")
            break
    server_socket.close()
```

Figure 5.3.9.2 send_thread. code

This thread is same with the master send to middleware 5.3.3.1 ,it is a loop sends data from the master robot to the connected master robot socket. If any messages are stored in the messagetomaster variable, it attempts to encode and send them to the master robot. It will clear the messagetomaster just wait 0.05 to prevent rapid repeated sending.

5.3.10 Slave servo

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

#servo
servo_pin = 5
GPIO.setup(servo_pin ,GPIO.OUT)
GPIO.setup(6 ,GPIO.OUT)
pwm = GPIO.PWM(servo_pin ,50)
pwm1 = GPIO.PWM(6 ,50)
pwm.start(0)
pwm1.start(0)

def set_angle(angle):
    duty = angle /18 +2
    GPIO.output(servo_pin,True)
    pwm.ChangeDutyCycle(duty)
    time.sleep(0.5)
    GPIO.output(servo_pin,False)
    pwm.ChangeDutyCycle(0)

#<90 is up

def set_angle1(angle):
    duty = angle /18 +2
    GPIO.output(6,True)
    pwm1.ChangeDutyCycle(duty)
    time.sleep(0.5)
    GPIO.output(6,False)
    pwm1.ChangeDutyCycle(0)

#>90 is up
```

Figure 5.3.10 Slave Servo. code

This all code is to explain how the slave is using servo MG996R to up and down. In the slave will using **GPIO 5 (pin 29) and 6 (pin 31)**. The script begins by setting up the GPIO mode to BCM. It configures two GPIO pins (5 and 6) as outputs. Two PWM (pulse width modulation) objects, pwm and pwm1, are initialised on these pins with a frequency of 50Hz and start the pwm. The function **set_angle** is the servo using GPIO 5 (pin 29), which changes the duty cycle to move the servo, waits for 0.5 seconds to allow the servo to reach the desired position, and finally stops the PWM signal. The function **set_angle1** is the servo using GPIO 6 (pin 31), also like that the set_angle

5.3.11 Salve movement

```

def movmentofrobotslavel(data_movement):
    global degree_value
    gpg.set_speed(150)
    if(data_movement == "slavelstop"):
        gpg.stop()
        gpg.turn_degrees(degree_value)
        time.sleep(0.5)
        degree_value = 0
    elif(data_movement == "slavelforward"):
        gpg.forward()
    elif (data_movement == "slavelback"):
        gpg.backward()
    elif(data_movement == "slavelleft"):
        gpg.turn_degrees(-90)
        degree_value = degree_value + 90
        time.sleep(0.5)
        gpg.forward()
    elif(data_movement == "slavelright"):
        gpg.turn_degrees(90)
        degree_value = degree_value - 90
        time.sleep(0.5)
        gpg.forward()

def movmentofrobotModell(data_movement):
    global degree_value
    gpg.set_speed(150)
    if(data_movement == "stop"):
        gpg.stop()
        gpg.set_speed(100)
        gpg.turn_degrees(degree_value)
        time.sleep(0.5)
        degree_value = 0
    elif(data_movement == "modellforward"):
        gpg.forward()
    elif (data_movement == "modellback"):
        gpg.backward()
    elif(data_movement == "modellleft"):
        gpg.set_speed(100)
        gpg.turn_degrees(-90)
        degree_value = degree_value + 90
        time.sleep(0.5)
        gpg.set_speed(150)
        gpg.forward()
    elif(data_movement == "modellright"):
        gpg.set_speed(100)
        gpg.turn_degrees(90)
        degree_value = degree_value - 90
        time.sleep(0.5)
        gpg.set_speed(150)
        gpg.forward()

def spinmovement(data_movement):
    global spin_degree
    gpg.set_speed(100)
    if(data_movement == "slavelspinforward"):
        if(spin_degree == 90):
            gpg.turn_degrees(-90)
            spin_degree = 0

def movepickupfunction(decision_degree):
    if(decision_degree == "left"):
        gpg.set_speed(100)
        gpg.turn_degrees(-90)
        time.sleep(1)
        gpg.set_speed(100)
        gpg.drive_cm(25)

        gpg.set_speed(100)
        gpg.turn_degrees(90)
        time.sleep(1)

        gpg.set_speed(100)
        gpg.drive_cm(130)
        time.sleep(1)

        gpg.set_speed(100)
        gpg.turn_degrees(90)
        time.sleep(1)
        gpg.set_speed(100)
        gpg.drive_cm(2)
        time.sleep(2)
        gpg.set_speed(100)
        gpg.turn_degrees(90)

    elif(decision_degree == "right"):

        gpg.set_speed(100)
        gpg.turn_degrees(90)
        time.sleep(1)
        gpg.set_speed(150)
        gpg.drive_cm(25)

        gpg.set_speed(100)
        gpg.turn_degrees(-90)
        time.sleep(1)

        gpg.set_speed(150)
        gpg.drive_cm(130)
        time.sleep(1)

        gpg.set_speed(100)
        gpg.turn_degrees(-90)
        time.sleep(1)

        gpg.set_speed(150)
        gpg.drive_cm(25.5)
        time.sleep(2)

        gpg.set_speed(100)
        gpg.turn_degrees(-90)

def movmentofrobotAdjust(data_movement):
    gpg.set_speed(150)
    if(data_movement == "slaveladjustleft"):
        gpg.turn_degrees(-90)

```

Figure 5.3.11 Salve movement code

This Python script is supporting the main Slave Python script can call to handle various actions. It includes functions like modellforward, adjustleft, spinleft, and others function. Function movmentofrobotslave is controls the master robot's self-movement, and spinmovement, which allows it to spin independently. Other functions like movmentofrobotModell dictate specific movement patterns. These modular functions help the main Slave Python script keeping it clear .

5.3.12 Slave arm code

```
def armmovement(data_movement):
    if(data_movement == "slaveldefault"):
        gpg.set_speed(150)
        servo.rotate_servo(90)
        set_angle(70)
        set_angle1(110)
    elif(data_movement == "slavelpickup"):
        set_angle(70)
        set_angle1(110)
    elif(data_movement == "slavelpickdown"):
        set_angle(110)
        set_angle1(70)
```

Figure 5.3.12 Slave armmovement. code

This function controls the slave arm with a servo and a 3D printer part. It has simple commands, such as default, pick up, and pick down. Based on the command to adjust the 2 servos with x.x.x before the function.

5.3.13 Master and Slave Status

```
def statusupdate():
    global messagetomaster
    while True:
        messagetomaster = "Slavelonline\n"
        time.sleep(0.2)
        messagetomaster = "SlavelVolt" + str(gpg.volt()) + "\n"
        time.sleep(0.2)
        messagetomaster = "Slavel" + "10.10.10.1" + "\n"
        time.sleep(0.2)
        time.sleep(10)

def statusupdate():
    global send_mobile_message
    while True:
        send_mobile_message = "Masteronline\n"
        time.sleep(0.2)
        send_mobile_message = "MasterVolt" + str(gpg.volt()) + "\n"
        time.sleep(0.2)
        addresses = netifaces.ifaddresses("wlan1")
        ip_address = addresses[netifaces.AF_INET][0]['addr']
        send_mobile_message = "MasterIp" + str(ip_address) + "\n"
        time.sleep(0.2)
        time.sleep(10)
```

Figure 5.3.12 Master and Slave Status Thread. code

The master and slave thread serve the same purpose: continuously looping to update and send status messages about the master and slave robots' states to the mobile application. These functions update the robot's status, and store in the **send_mobile_message** or **messagetomaster** variables. These variables manage the **send_thread** sending the data to the master or middleware. The status updates include key information such as the robot's **online status, power level, and IP address**.

5.3.14 Master and Slave Ultrasound

```
def detect_ultrasound():
    global degree_value, detect_value
    global ultrasound_distance
    global messagetomaster
    ultrasound_sensor = gpg.init_distance_sensor()
    time.sleep(1)
    while True:
        try:
            ultrasound_distance = int(ultrasound_sensor.read_mm())
            Slavemovmet.setvalueultrasound(ultrasound_distance)
            print(f"Distance: {ultrasound_distance} mm")
            time.sleep(0.3)
            if ultrasound_distance < detect_value:
                gpg.stop()
                gpg.backward()
                time.sleep(0.5)
                gpg.stop()
                messagetomaster = "Slaveldone"
            time.sleep(0.1)
        except IOError as e:
            #logging.error(f"Error reading distance sensor: {e}")
            time.sleep(1) # Delay before retrying

        except Exception as e:
            print("error")
            print(f"An error occurred: {e}")
            break # Break the loop if an error occurs
```

Figure 5.3. 14 Master and Slave Ultrasound.code

The master and slave share the same thread for reading the ultrasound sensor and detecting objects in front of the robot. The `ultrasound_sensor = gpg.init_sitance_sensor()` is the function designed by the GoPoGi3 company to read the ultrasound sensor and `ultrasound_sensor.read_mm()` outputs the distance in millimetres as a string. When the sensor detects an object closer than the specified `detect_value`, the robot will stop and move back slightly. The `detect_value` is a global variable, so to deactivate the object avoidance or handle very close readings, you can set `detect_value` to a negative or alternate value. This thread runs every 0.3 seconds, preventing sensor overload from frequent readings and keeping the process smooth.

5.3.15 Mobile receive from middleware

```

receiveThread = new Thread() -> {
    try {
        // Connect to the server
        clientSocket = new Socket(serverIp, serverPort);
        Log.d(TAG, msg: "Connected to server at " + serverIp + ":" + serverPort);

        output = new PrintWriter(new OutputStreamWriter(clientSocket.getOutputStream(), charsetName: "UTF-8"), autoFlush: true);
        BufferedReader input = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(), charsetName: "UTF-8"));

        // Receiving data from the server
        String receivedMessage;
        while (!Thread.currentThread().isInterrupted()) {
            Log.d(TAG, msg: "Waiting to receive message from server...");
            receivedMessage = input.readLine();
            Log.d(TAG, msg: "Received from server: " + receivedMessage);
            if (receivedMessage != null) {
                if (receivedMessage.equals("Hi")){
                    editor.putString("StatusToServer", "success");
                    editor.apply();
                }
                else if (receivedMessage.equals("Masteronline")){
                    editor.putString("StatusToMaster", "success");
                    editor.apply();
                }
                else if (receivedMessage.contains("MasterVolt")){
                    String modifiedString = receivedMessage.replaceAll(regex: "MasterVolt", replacement: "");
                    editor.putString("MasterVolt",modifiedString);
                    editor.apply();
                }
                else if (receivedMessage.contains("MasterIp")){
                    String modifiedString = receivedMessage.replaceAll(regex: "MasterIp", replacement: "");
                    editor.putString("MasterIp",modifiedString);
                    editor.apply();
                }
                else if (receivedMessage.equals("Slaveonline")){
                    editor.putString("StatusToSlave1", "success");
                    editor.apply();
                }
            }
        }
    }
}

```

Figure 5.3.15 Mobile Receive from Middleware.code

This thread handles messages received from the master robot on the mobile side. It enters a loop, waiting for incoming messages from the server and reads text data using `input.readLine()`. When a message is received, the thread processes it and writes, such as `mastervolt`, `masterrip`, and `masterstatus`, into `SharedPreferences`. It can allow other Java classes to access this data. If the middleware disconnects, the thread clears the values stored in `SharedPreferences`.

5.3.16 Mobile Send to middleware

```

sendThread = new Thread() -> {
    try {
        // Wait until the clientSocket is connected
        while (clientSocket == null || clientSocket.isClosed()) {
            Thread.sleep( millis: 100);
        }

        // Continuous loop for sending data
        while (!Thread.currentThread().isInterrupted()) {
            String message_value = sharedPreferences.getString( key: "message_key", defValue: "default_va
            messagetoserver = message_value;
            if (!messagetoserver.isEmpty()) {
                output.println(messagetoserver);
                Log.d(TAG, msg: "Sent to server: " + messagetoserver);

                // Optionally, clear or update messagetoserver after sending
                messagetoserver = ""; // Clear the message or set to new value
                editor.putString("message_key", "");
                editor.apply();
            }

            // Sleep for a while to avoid busy-waiting
            try {
                Thread.sleep( millis: 1000); // Sleep for 1 second before checking again
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt(); // Restore interrupted status
                Log.d(TAG, msg: "Send thread interrupted");
            }
        }
    } catch (Exception e) {
        Log.e(TAG, msg: "Error sending data: ", e);
    }
};

```

Figure 5.3.16. Mobile Send to Middleware.code

This thread sends messages from SharedPreferences using the key "message_key" to the master robot. If a message exists and is not empty, it assigns the message to messagetoserver and sends it to the server via output.println(messagetoserver). After sending the message, it clears both messagetoserver and the corresponding SharedPreferences entry to be ready for the next message. The thread then pauses for 1 second using Thread.sleep(1000) to prevent excessive CPU usage.

5.3.17 Status

```

String detect_master_status = sharedPreferences.getString( key: "StatusToMaster", defValue: "default_value");
String detect_master_battery = sharedPreferences.getString( key: "MaterVolt", defValue: "0");
String detect_master_IP = sharedPreferences.getString( key: "MasterIp", defValue: "X");

String detect_slave1_status = sharedPreferences.getString( key: "StatusToSlave1", defValue: "default_value");
String detect_slave1_battery = sharedPreferences.getString( key: "Slave1Volt", defValue: "0");
String detect_slvae1_IP = sharedPreferences.getString( key: "Slave1Ip", defValue: "X");

String detect_slave2_status = sharedPreferences.getString( key: "StatusToSlave2", defValue: "default_value");
String detect_slave2_battery = sharedPreferences.getString( key: "Slave2Volt", defValue: "0");
String detect_slvae2_IP = sharedPreferences.getString( key: "Slave2Ip", defValue: "X");

updateConnectionStatus(detect_master_status, SetMasterIPStatus);
SetMasterIP.setText(detect_master_IP);
SetMasterBatteryStatus.setText(detect_master_battery);

updateConnectionStatus(detect_slave1_status, SetSlave1IPStatus);
SetSlave1IP.setText(detect_slvae1_IP);
SetSlave1BatteryStatus.setText(detect_slave1_battery);

updateConnectionStatus(detect_slave2_status, SetSlave2IPStatus);
SetSlave2IP.setText(detect_slvae2_IP);
SetSlave2BatteryStatus.setText(detect_slave2_battery);

```

Figure 5.3.17.1 Status page code

```

6 usages
private void updateConnectionStatus(String status, TextView connectionTextView) {
    if (status.equals("success")) {
        connectionTextView.setText("ON");
        connectionTextView.setTextColor(Color.parseColor( colorString: "#00FF00")); // Green color
    } else {
        connectionTextView.setText("OFF");
        connectionTextView.setTextColor(Color.parseColor( colorString: "#FF0000")); // Red color
    }
}
}

```

Figure 5.3.17.2 updateCpmmectopmStatus code

The figure 5.3.17.1 code is used to show data about the master and slave robots on the mobile application after retrieving it from SharedPreferences. Details like the IP address, battery level, and online/offline status are shown. By examining certain settings kept in SharedPreferences, 5.3.17.2 also determines whether the master or slave robots are online or offline. The robots are tagged as offline if any of these variables shows "success" or another specified message. Otherwise, they are regarded as online.

5.3.18 Send part in any page

```

setButtonTouchListener(FunctionStop, message: "stop");
movementsselect(FunctionForward, message: "forward");
movementsselect(FunctionLeft, message: "left");
movementsselect(FunctionRight, message: "right");
movementsselect(FunctionBack, message: "back");
setImageButtonClickListener(FunctionAdjustLeft, message: "masteradjustleft");
setImageButtonClickListener(FunctionAdjustRight, message: "masteradjustright");

//Button Function
Button defaultfunction = findViewById(R.id.buttondefault);
Button AllFollow = findViewById(R.id.buttonallfollow);
Button Pickup = findViewById(R.id.buttonpickup);
Button AutoPickupLeft = findViewById(R.id.buttonautopickupleft);
Button AutoPickupRight = findViewById(R.id.buttonautopickupright);
setButtonClickListener(defaultfunction, message: "default");
setButtonClickListener(AllFollow, message: "allfollow");
setButtonClickListener(Pickup, message: "masterpickup");
setButtonClickListener(AutoPickupLeft, message: "autopickupleft");
setButtonClickListener(AutoPickupRight, message: "autopickupright");

```

Figure 5.3.18.1 Send in Any Page code

```

8 usages
private void putthemessage(final String message) {
    SharedPreferences sharedPreferences = getSharedPreferences("SocketMessagePassing", MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString("message_key", message);
    editor.apply();
    Log.d("Main", message);
}

```

Figure 5.3.18.2 putthemessage code

The page 5.3.18.1 is intended to call certain methods and provide the message that should be delivered to the middleware. It has buttons for the functions of push, untouch, and ontouch. Keywords like "master," "slave," "model1," and others will be added to the message by the movementsselect function. 5.3.18.2 stores it in message_key within SharedPreferences to enable the transmit thread to obtain and pass the created message to the middleware server.

5.4 System Operation

5.4.1 Master and Slave : GoPiGo3 OS

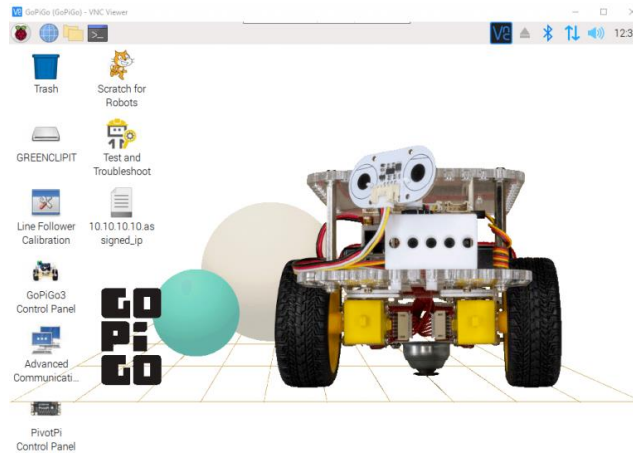


Figure 5.4.1 GoPiGo3 OS

The GoPiGo3 OS is a customized version of Raspberry Pi OS, which is itself based on Linux. Using GoPiGo3 OS is like working with Kali Linux. There are three methods to configure the system. The first method involves directly connecting a monitor, keyboard, mouse, and using HTML to interact with the Raspberry Pi, displaying the interface referred to as figure 5.4.1. The second method is through VNC, where you access the Raspberry Pi by entering its IP address, along with the username pi and the password robots1234, to access the same figure 5.4.1 interface remotely. The third method is SSH, where you use the IP address and port 22 to log in with the same username and password, but only access a command-line interface (no UI).

5.4.2 Server : Ubuntu 22.04.4 LTS

```

Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-117-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:      https://ubuntu.com/pro
Last login: Wed Sep 11 06:59:00 2024 from 121.123.29.130
root@blank:~# █

```

Figure 5.4.2 Ubuntu 22.04.4 LTS OS

Ubuntu, like Raspberry Pi OS, is also based on Linux. However, figure 5.4.2 can only be displayed when accessing the system through SSH using the IP address and port number.

CHAPTER 5

5.5 Concluding Remark

This chapter provides a detailed overview of the hardware and software setup for the master and slave GoPiGo3 robotic system. The master robot uses an MG996R servo connected to a PCA9685 board to control its arm, while the slave robot utilizes an MG996R servo and 3D-printed components to create its arm. It explains how the master and slave arms are controlled and their functions. Both robots use multithreading to run the code, enabling seamless communication between the master, slave, middleware, and mobile application. The communication of slave to mobile application is the slave sends data to the master, the master forwards it to the middleware server, which then transmits it to the mobile application. If the mobile application sends a command to the slave, the process reverses and repeats.

Chapter 6 - System Evaluation and Discussion**6.1 System Testing and Performance Metrics**

Table 6.1 Testing and Output of FYP2

No	Test Features	Test Performance	Expected Output
1	Mobile Application Device Connecting to Server via Network	<ul style="list-style-type: none"> - When the mobile application is opened, it will automatically connect to the virtual machine server, which acts as the middleware server, using the public IP address and port. - On the Settings page, it can enter the IP address and port, then click submit to connect to the middleware server. 	<ul style="list-style-type: none"> - A successful connection will be indicated by an "ON" status in green on the top bar of the mobile application. Otherwise, it will display "OFF" in red. - On the Settings status page, it will display the connection status for the IP address and port. A successful connection will be shown in green, while a failed connection will be shown in red.
2	Master and Slave Devices Successfully Connected to Server and Mobile Application	<ul style="list-style-type: none"> - When the mobile application is opened, it will display the Master Robot's status on the top bar. Pressing the reload icon will refresh the status of the Master Robot and the middleware server. - On the Status page, the status of both the Master and Slave Robots will be displayed in a table. Pressing the reload icon will refresh the statuses of both the Master and Slave Robots. 	<ul style="list-style-type: none"> - A successful connection from the Master Robot will be indicated by an "ON" status in green on the top bar of the mobile application. Otherwise, it will display "OFF" in red. - After a successful connection from the Master Robot, the Robot status page in the mobile application will show that the Slave Robot has connected successfully. If the connection is successful, it will display "ON" in green;

			otherwise, it will display "OFF" in red.
3	Mobile Application Controls Movement of Master and Slave Robots	- On the Master page of the mobile application, select the desired movement from the switch bar, and then press and hold the icon for forward, back, left, or right.	- When press and hold an icon, the Master and Slave Robots will execute the command (forward, back, left, or right) and move together. -When no hold the icon, both the Master and Slave Robots will stop moving and return to their initial position facing forward.
4	Mobile Application Displays Status of Master and Slave Robot	- Opening the Status page in the mobile application will display a table showing the status and battery levels of the robots.	- On the status page of the mobile application, it will display the online status, IP address, and battery level of both the Master and Slave Robots.
5	Mobile Application Assigns Functions to Master and Slave Robots	- On the Master or Slave page, there is a function button. Pressing this button will activate a specific function, which can be tailored for either the Master or Slave Robot, or both robots working together.	- Default button will reset the sensor to its default value. - Pick button will control the arm to pick up or lower objects. - Follow button will allow the Slave Robot to closely follow the robot in front of it.
6	Master and Slave Robots Avoid Obstacles	- Place hand near the ultrasound sensor on the Master Robot or Slave Robot	- Master or Slave Robot will stop and then move back a few cm

7	Master and Slave Robots Perform Tasks	<p>- Place the bottle in a specific location, allowing both the Master and Slave Robots to perform the task manually or automatically.</p>	<p>- Control both the Master and Slave Robots to approach the bottle. Master Robot Lower the arm, position it inside the bottle, grip it, and pick it up.</p> <p>- Slave Robot will move in front of the Master Robot and use the 3D printer to handle the bottle.</p> <p>- Master Robot will move in front of the Slave Robot, while any other Slave Robots will approach the Slave Robot.</p> <p>- Continue to the next stage of the task until both the Master and Slave Robots are fully.</p>
8	Mobile Controls Independent Movement of Master and Slave Robots	<p>- On the Master page of the mobile application, select the alone movement mode from the switch bar, then press and hold the icon for forward, back, left, or right.</p> <p>- On the Slave page of the mobile application, select the alone movement mode from the switch bar, then press and hold the icon for forward, back, left, or right.</p>	<p>- Master or Slave Robot is set to alone movement mode, it will perform the forward, back, left, or right movements independently.</p> <p>-Master or Slave Robot is set to alone spin movement mode, it will rotate left or right by 90 degrees or 180 degrees.</p>

CHAPTER 6

9	Mobile Controls Arm Movement	- On the Arm page of the mobile application, press the function to control the arm. can press like Open, Close, Left, Right, Up, or Down icon to adjust arm.	- The function will move the arm to the specified setting value - Pressing the icon, small adjustments to the arm
10	Server Handles Message Passing	- Mobile application and the Master Robot will send messages to the middleware server.	- The SSH server will display messages received from each client

6.2 Testing Setup and Result

6.2.1 Taking Task

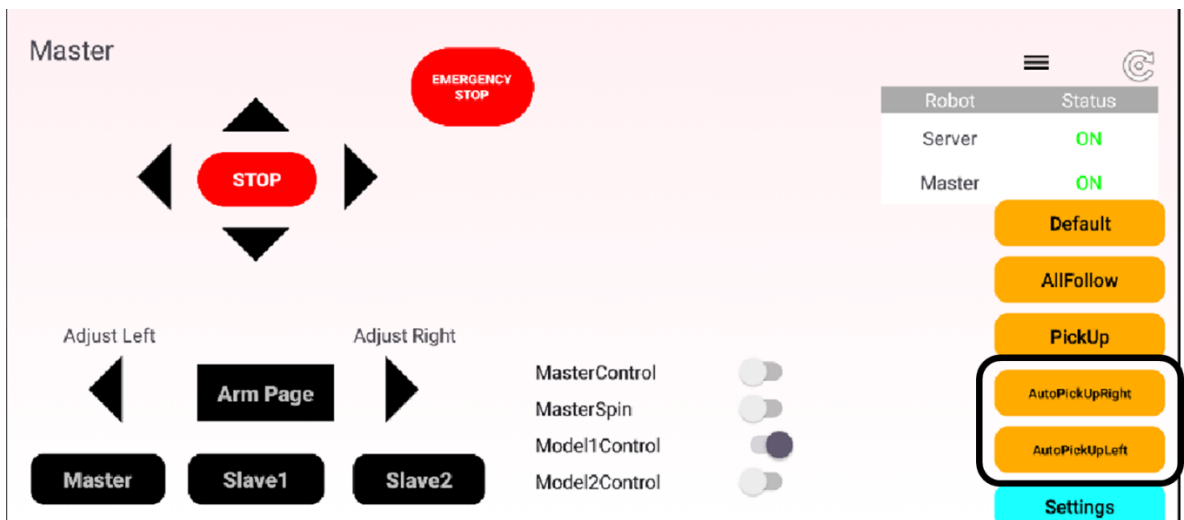


Figure 6.2.1.1 Mobile Master Page

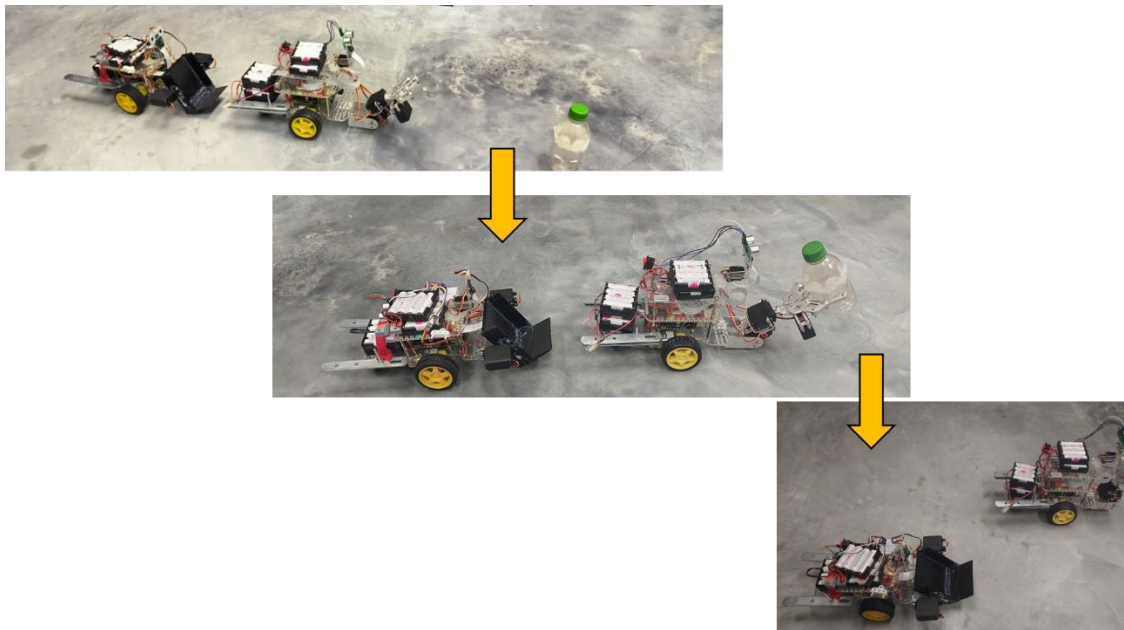


Figure 6.2.1.2 Mobile Tasking Task Part1

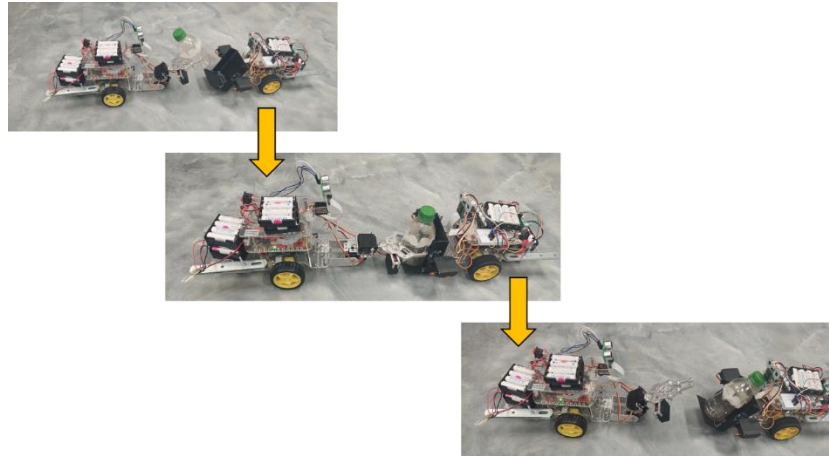


Figure 6.2.1.3 Mobile Tasking Task Part2

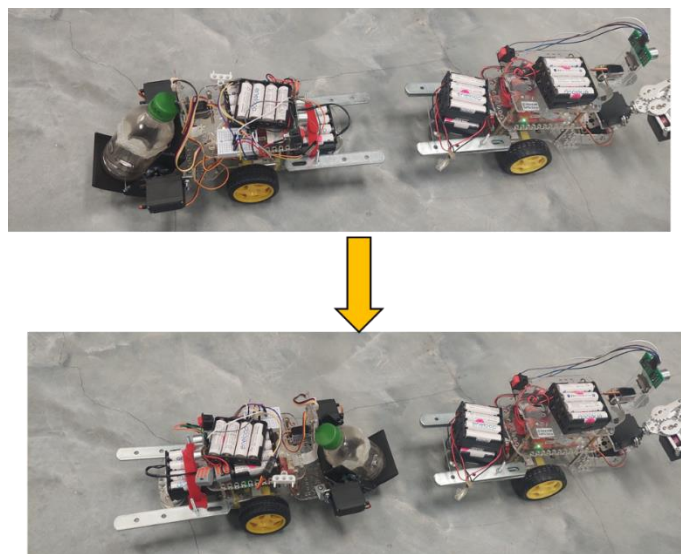


Figure 6.2.1.4 Mobile Tasking Task Part3

The task involves the master robot using its arm to pick up a bottle and put the bottle on the slave robot's arm. First, the master and slave robot will manually go near the bottle, lower its arm, and move toward the bottle. Then figure 6.2.1.1 show decide whether the slave should move left, right, to master in front. After pressing the function button, the master grips the bottle, and the slave moves left and forward like figure 6.2.1.2. Next, figure 6.2.1.3 show near the master and the slave to turn up arm; the master lowers their arms, and the slave will raise their arms, allowing the slave to take the bottle. Finally, like figure 6.2.1.4, the master is in front of the slave as the slave turns back and moves close to the master.

6.2.2 Movement Together

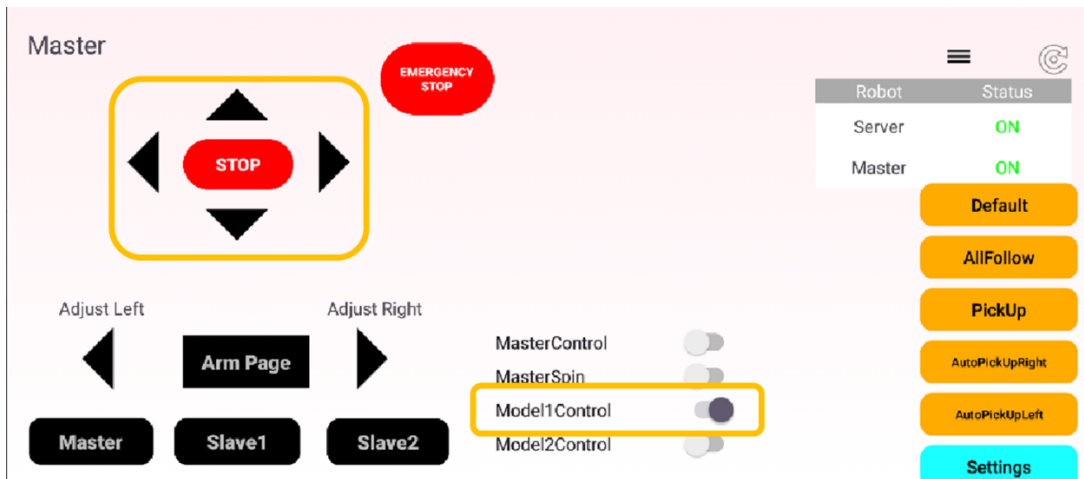


Figure 6.2.2.1 Mobile Master Page

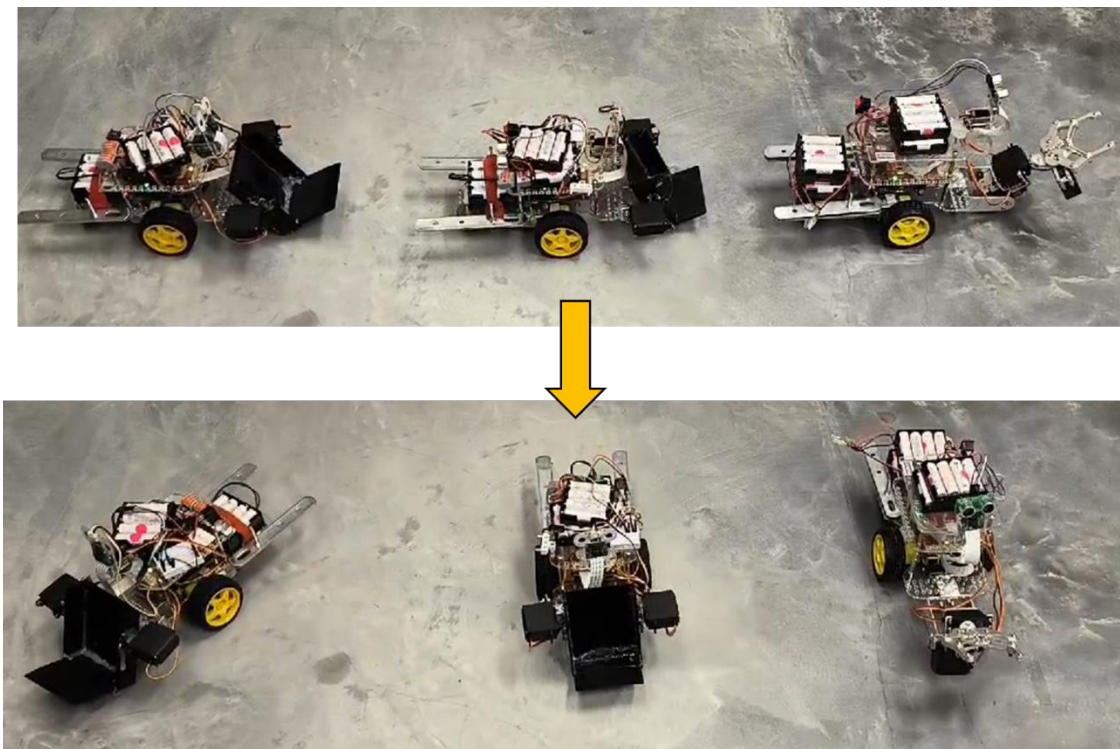


Figure 6.2.2.2 Movement Together Part1

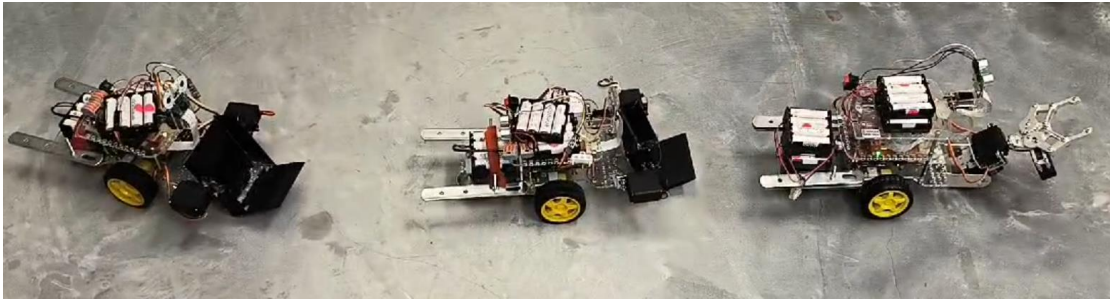


Figure 6.2.2.3 Movement Together Part2

The figure 6.2.2.1 is the master page of the mobile application. As shown in the figure, when the user selects Model1Control and holds down the negative button, both the master and slave robots will move together as depicted in figure 6.2.2.2. If the button is not held, the behaviour changes like figure 6.2.2.3 where both the master and slave robots will return to a 0-degree position. Regardless of whether they move left or right first, they will ultimately reset back to the 0-degree position.

6.2.3 Avoid Obstacles

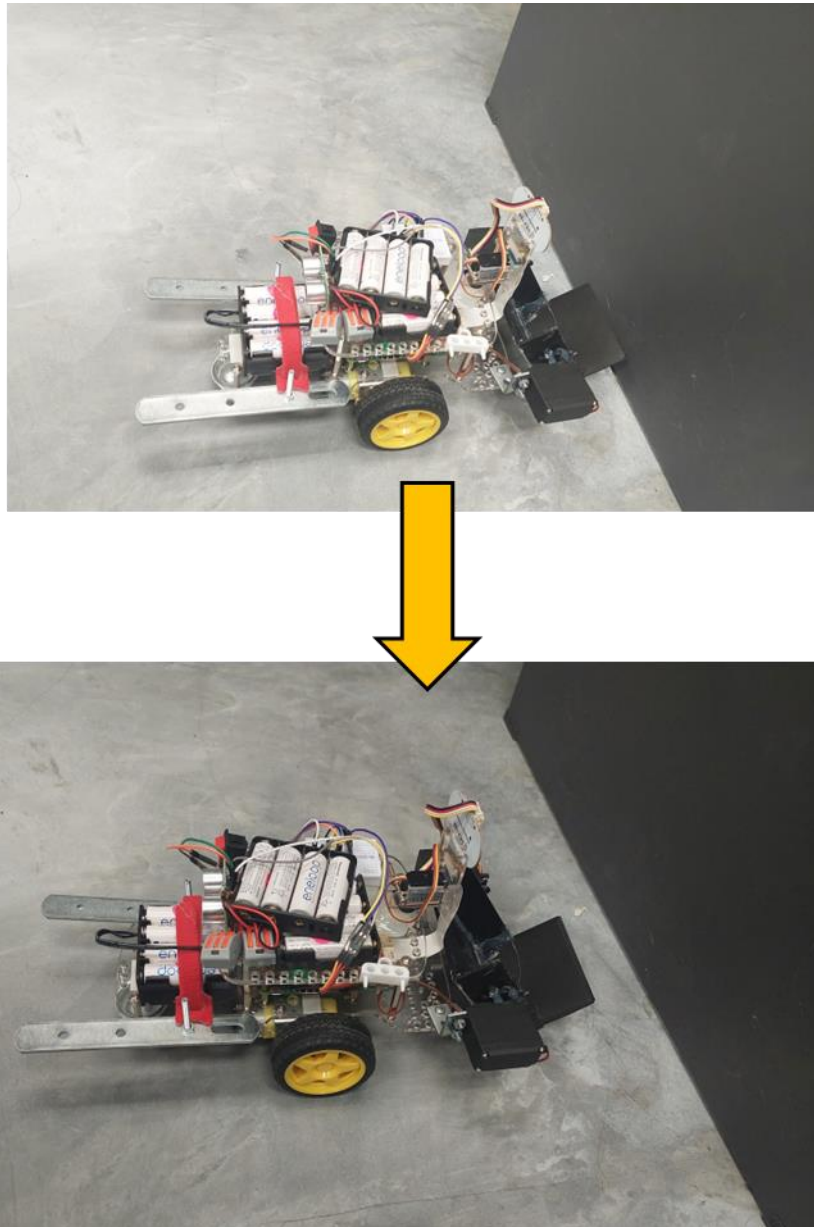


Figure 6.2.3 Avoid Obstacles

This figure 6.2.3 demonstrates the safety mechanism in place for both the master and slave robots. When either robot detects that it is too close to an object, it immediately stops and reverses slightly. This action is to protect the robot's hardware damage.

6.2.4.1 Alone Movement

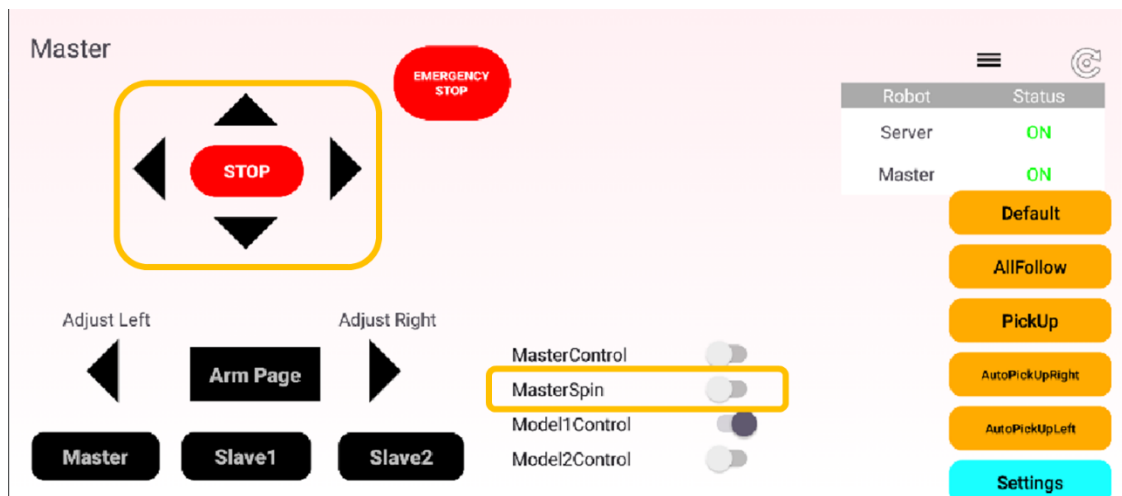


Figure 6.2.4.1 Mobile Master Page

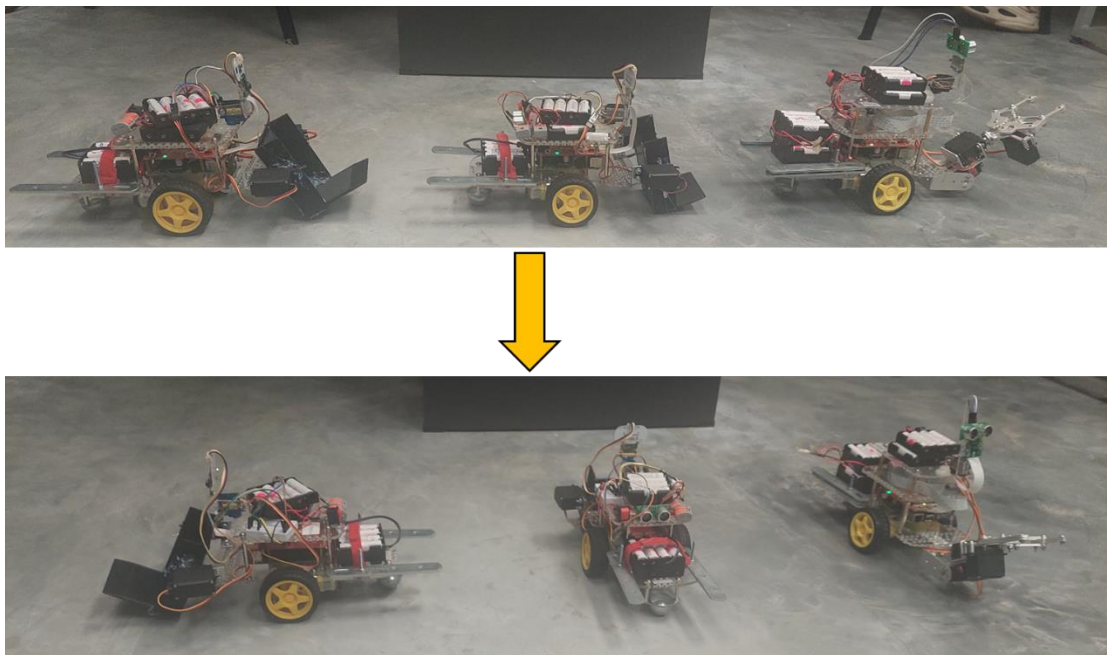


Figure 6.2.4.2 Alone Movement

When selecting the correct model and pressing the navigation button on the master or slave page, as shown in figure 6.2.4.1. On the master page, there are MasterControl and MasterSpin models to control the master robot individually. On the slave page, there are only SlaveControl and SlaveSpin for independent control. When press button, the master and slave robots will move individually, as figure 6.2.4.2. This let the users to control each robot separately to handle specific situations.

6.2.5 Arm Control

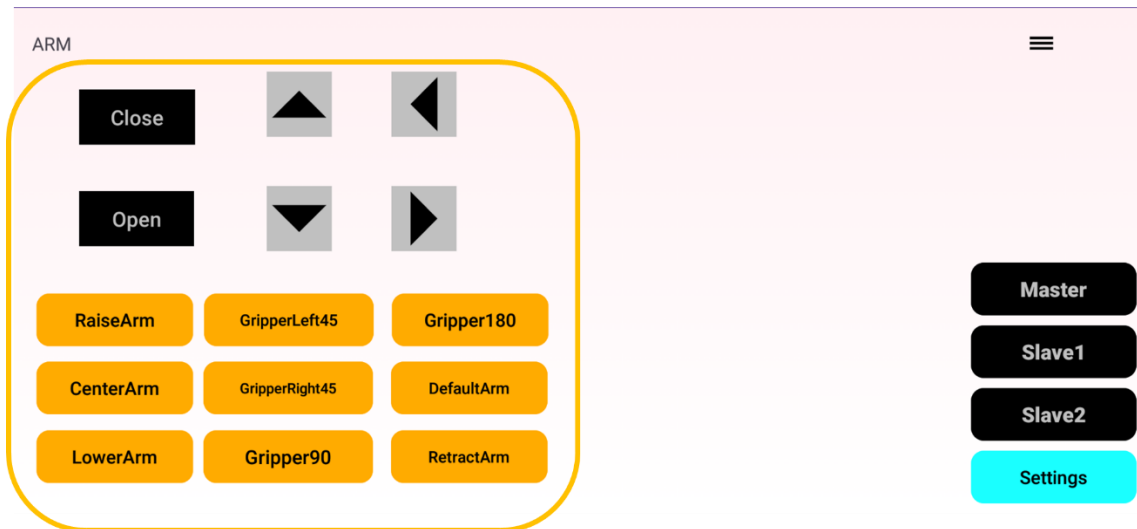


Figure 6.2.5.1 Mobile ARM Page

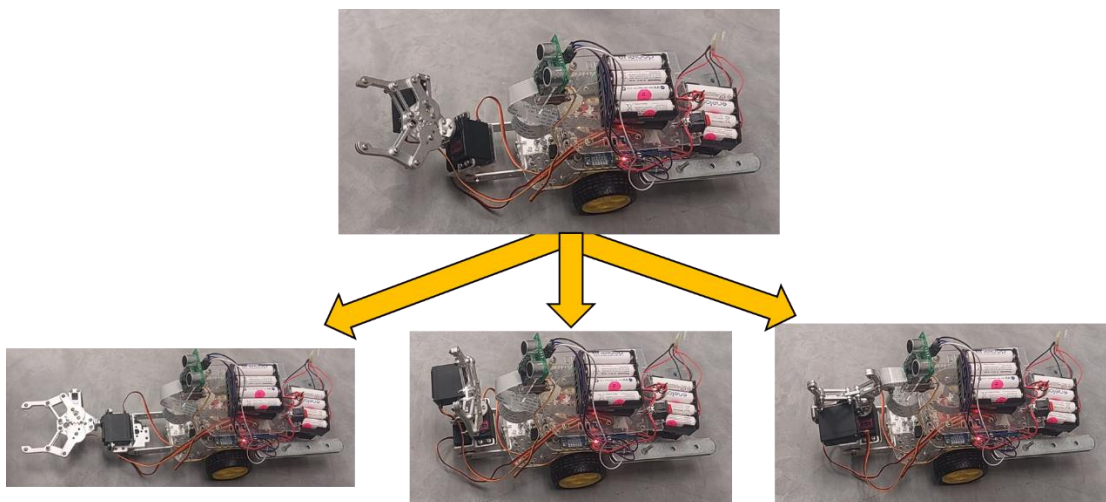


Figure 6.2.5.2 Arm Control

The figure 6.2.5.1 is the arm control page is control of the robot's arm movements. It includes buttons for basic directional controls such as up, down, left, right, as well as opening and closing the gripper slightly. Users also can press pre-configured buttons like RaiseArm, DefaultArm, LowerArm, and other functions. The figure 6.2.5.2 demonstrates the arm transitioning through different positions, from DefaultArm to LowerArm, RaiseArm, and Gripper90.

6.2.6 Follow function

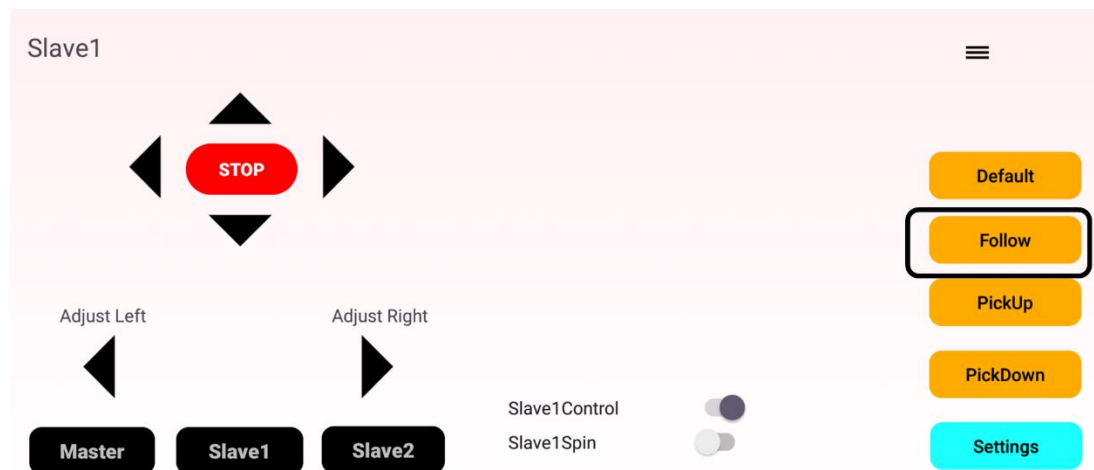


Figure 6.2.6.1 Mobile Slave Page

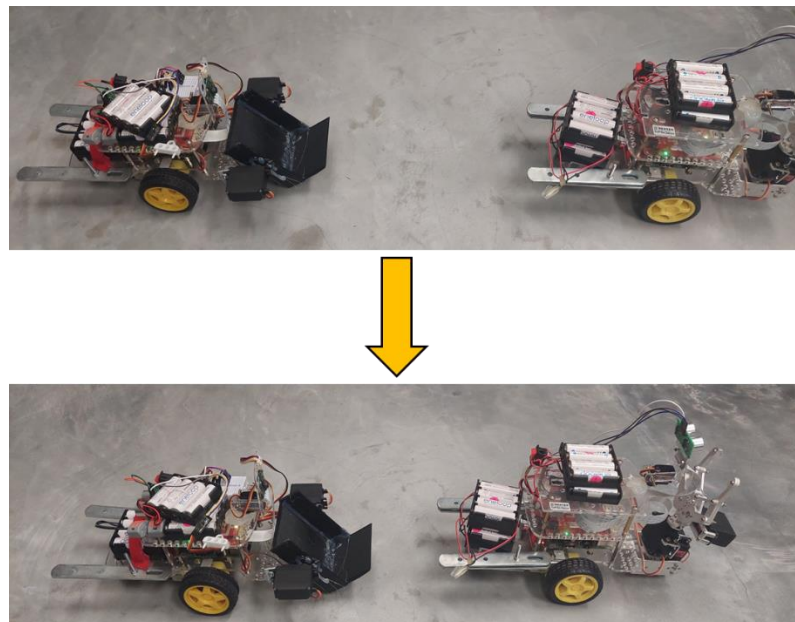


Figure 6.2.6.2 Follow Function

When the function buttons, like the one shown in figure 6.2.6.1, is clicked on the slave page, the slave robot will move closer to the master robot, as illustrated in figure 6.2.6.2. This function is designed to bring the slaver robot near the master robot .It also have other functions can quickly direct the robot to perform various tasks. 6.2.6.1, however, is dedicated to this single action of moving the slave robot closer to the master.

6.2.7 Status of connect master and server

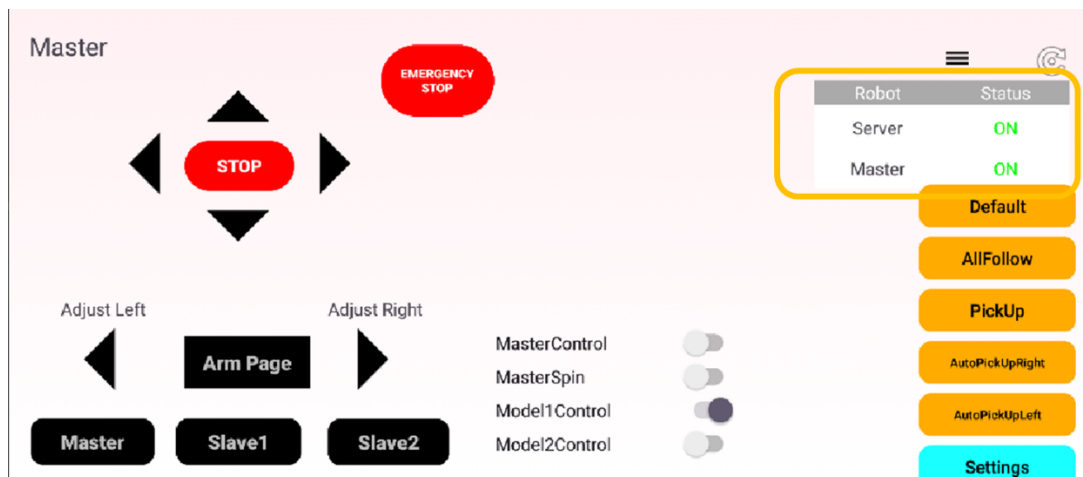


Figure 6.2.7 Mobile Master Page

This figure 6.2.7 shows that when the server and master robot are connected to the mobile application, a status bar at the top indicates that both are online. The figure 6.2.7 also represents the main page of the mobile application, providing full functionality. The slave page is similar page, it will less fewer functions compared to the master's page.

6.2.8 Status of master and slave

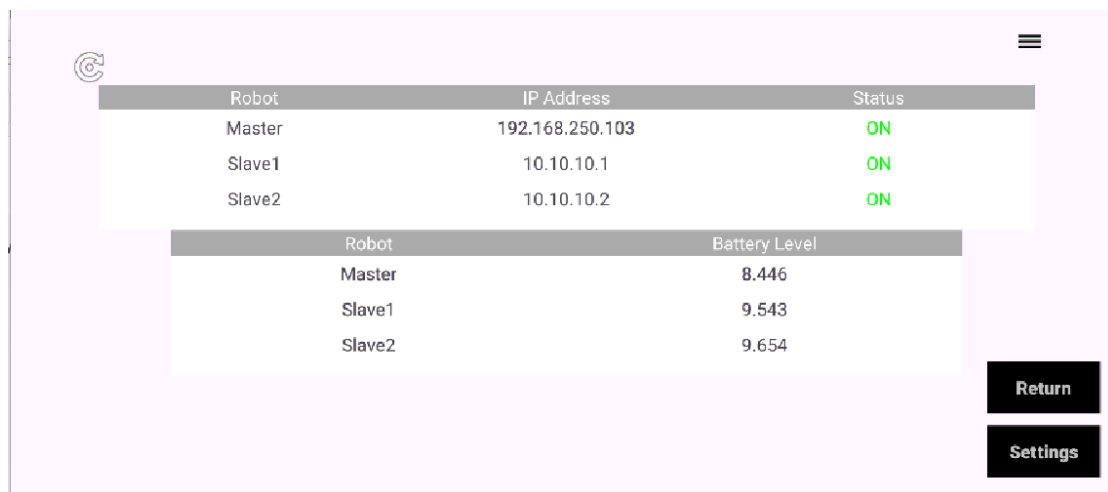


Figure 6.2.8 Mobile Status Page

The figure 6.2.8 displays the status page of the mobile application, which shows key information about the master robot and two slave robots, including their online/offline status, battery levels, and IP addresses. This page is designed to help users monitor the robots' status, check their connectivity, and how much battery life.

6.2.9 Status try to connect to server

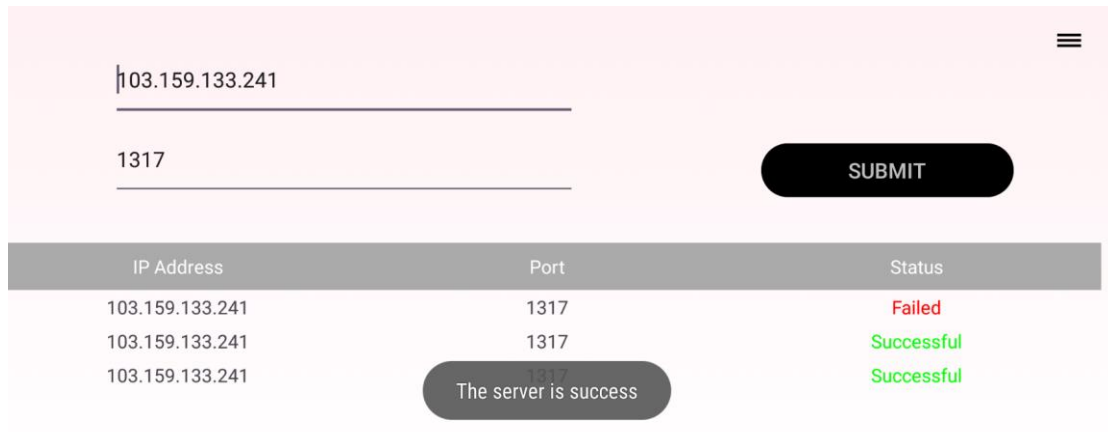


Figure 6.2.9 Mobile Settings Page

The page is the settings page of the mobile application, designed to reconnect to the server when it is down. Upon a successful connection, a message indicating success will be displayed; otherwise, it will show a failure notification. Once connected successfully, the app will display a pop-up message to avoid reconnecting with the same IP address and port.

6.3 Project Challenges

6.3.1 Customize Operating System

6.3.1.1 Problem

The GoPiGo3 and OS developed by Dexter Industries, is a powerful robot for beginners. It includes a user-friendly interface, libraries, and custom ports for specialized sensors. There is no need for manual installation, as everything is automatically set up when the GoPiGo OS is installed onto an SD card. Users can connect to the robot's Wi-Fi, which does not require a password, and access the GoPiGo interface through a browser by entering 10.10.10.10.

While it is good for beginners, but the issues begin to emerge when advancing into more complex development. A significant challenge is the lack of documentation from Dexter Industries on what modifications have been made and which Raspberry Pi pins are in use. Additionally, that is limited online resources compared to other like the ESP32.

In this project, several problems were encountered while modify the GoPiGo3 for internet access. By default, the system restricts Wi-Fi access because it is already assigned to the robot's interface. Although an Ethernet connection can provide an IP address, it cannot establish a connection to the internet. Furthermore, the master system needs two LAN connections ,one for the slave network and another for external internet access.

6.3.1.2 Solution

To set up two LAN connections, a USB network adapter was added to the master robot to create a new LAN while keeping the access point active. Although the GoPiGo3 detected the USB network, it couldn't connect to Wi-Fi. Upon researching, it was discovered that the `wpa_supplicant.conf` file is responsible for Wi-Fi access. After setting the SSID and password, the USB network was able to get an IP address.

For the slave robot, the goal was to disable the GoPiGo3's access point. Research revealed a service called `start_as_access_point.service` on the official website. Stopping this service successfully disabled the access point, but Wi-Fi was still unavailable. Configuring the SSID and password in the `wpa_supplicant.conf` file made the Wi-Fi network selectable. A static IP address was then set using the `dhcpcd.conf` file, but the device still couldn't access the internet.

To solve the internet connectivity issue, a comparison between the original Raspberry Pi OS and the GoPiGo3 OS was made to identify any additional files or services that might block internet access. Through trial and error, it was discovered that a DNS service was preventing the connection. After deleting the `cinch.conf` file in `dnsmasq.d` and restarting the DNS service, both the USB network adapter and network cable were able to access the internet.

This project uncovered several solutions to connect the GoPiGo3 to the internet. A YouTube video detailing the process has been created for reference: https://youtu.be/iSyG6WF4z94?si=xs8cL54HFNQrOyQ_

6.3.2 Hardware Problem and Balancing Problem

6.3.2.1 Problem

There have the challenges with customizing the GoPiGo3 OS of limited documentation. During hardware installation, it was unclear which Raspberry Pi pins were being used. Attempts to find this information on the GoPiGo3 official website were unsuccessful, and no useful details about the GoPiGo3 boards could be found.

Another issue arose when installing the arm on both the master and slave robots. A balancing problem became apparent, as the robots would tip forward if they became too front-heavy

6.3.2.2 Solution

During the research, at GitHub link of <https://github.com/DexterInd/GoPiGo3>, find a helpful information about the hardware. Inside, one of the files contained an electric diagram of the GoPiGo3 board, and after reviewing it, it was found that a small section indicated the Raspberry Pi pins. The electric diagram, labelled as Figure 4.1.1.2, made it clear which Raspberry Pi pins can be used for installing hardware sensors. The GitHub bring a lot of important information such as firmware, language libraries, and other information about the GoPiGo3.

The hardware balancing issue, the weight of the arm was reduced, and additional weight was added on the back of the robot. This can help balance the arm and the robot .

6.3.3 Environment Network

6.3.3.1 Problem

The robot functions within the UTAR Campus environment, while the mobile application is designed for use outside the campus. Since UTAR's network is protected by strict security policies, accessing the external internet from within the campus is allowed, but the project's challenge lies in the mobile application's inability to connect to the master robot using socket programming from outside the campus. Due to security policies, port forwarding is not permitted. This let the socket programming a difficult using .

If the master robot initiates the connection to the mobile application, it complies with UTAR's policy regarding internet access from inside the campus to the outside. However, because the mobile application operates in an external environment, its IP address constantly changes. The public IP is managed by the ISP, and in some cases, the network might require pre-configuration, which may not be allowed by the owner or ISP. This creates an environmental challenge for both the robot and the mobile application.

6.3.3.2 Solution

Both the robot and the mobile application can access the internet, that meaning a public IP address can used to enable them to connect. While this idea is promising, the challenge lies in how to configure socket programming with a public IP address. Upon further research, an idea emerged from studying remote software like TeamViewer, AnyDesk, and others. These platforms use a middleware server call relay server, to handle communication between two devices.

Inspired by this, a Virtual Private Server (VPS) was rented to test passing messages between both clients using socket programming. After testing, the idea is successful, allowing communication between the master robot and the mobile application.

6.4 Objectives Evaluation

Table 6.4 Objectives and Evaluation

No	Objectives	Evaluation
1	Master Robot that can effectively manage a few Slave Robots	<ul style="list-style-type: none"> ✓ Master robot can send to and receive from slave robot by using the socket programming and the thread. (10.10.10.10:10102) ✓ Master can control all slaves together with master to move the movement or alone to control the slaves like forward, left ,right and back. ✓ Master has access point to let the slave connect to the master local LAN Wi-fi area (10.10.10.0/24). Master can also access the Wi-fi by using a USB network adapter to communicate outside the internet
2	Develop Master Robot and many Slave Robots that can work together to accomplish certain tasks	<ul style="list-style-type: none"> ✓ Master robot has arms to move up and down, left and right, open and close the gripper to catch the item. ✓ Slave robots have an arm to handle the item using a MG996R servo and 3D printer part to move up or down. ✓ Master robot will try using the arm to lift the bottle, and slave will go to in front of master to handle the bottle, then master grips the bottle and lowers it. The process is repeated until both the master and slave robots are fully.

3	Mobile application that allows for remote management and oversight of the Master Robot's functions	<ul style="list-style-type: none"> ✓ Develop a mobile application that has a master, slave, arm, status, and settings page to control the master and slave robot. ✓ Master, slave, and arm are controlling the master and slave robot movement, and some function is created. ✓ The status page is showing the master and slave robots are connected or disconnected; it will also be showing the IP address and battery level of the robot. ✓ Mobile applications can use any place that has internet to control the master and slave robots by using a middleware server (103.159.133.241) to pass the message between the master robot and mobile application.

6.5 Concluding Remark

The testing and performance metrics of a middleware server system that controls communication between a mobile application, a Master GoPiGo3 robot, and Slave robots are covered in Chapter 6 of the system assessment. The chapter describes a number of test scenarios, such as establishing a connection between the mobile app and the server, directing the actions of robots, presenting status updates, and carrying out tasks like manipulating objects. Innovative solutions were used to overcome obstacles found in the project, such as modifying the GoPiGo3 OS, hardware balance problems, and network restrictions brought about by school security regulations. One such solution was the use of a virtual private server for message relay. The project goals are evaluated and the system's capacity to coordinate Master-Slave robot tasks via socket programming and remote mobile management—achieving seamless communication across environments—is confirmed as the chapter ends.

7.1 Conclusion

This project addresses critical limitations in current robotic systems, particularly the one-to-one control model that constrains scalability and flexibility. It proposes a more dynamic approach where a Master Robot, powered by a Raspberry Pi 3B+, can manage and coordinate the actions of multiple Slave Robots. This setup is designed to enhance warehouse operations by allowing several robots to work together, performing tasks in unison. A mobile application, developed using Java, enables remote control and surveillance of these robots, expanding the system's utility by allowing real-time monitoring and command execution from a distance.

The robots communicate over a wireless network using technologies such as WiFi. These wireless protocols provide the necessary infrastructure for real-time communication and control, allowing users to manage the robots remotely. The robots are also designed to execute GOTO and FOLLOW-inspired algorithms, enabling them to navigate autonomously in multi-robot environments. This system is built on the GoPiGo3 platform, which is modified to handle multi-slave robot communication and coordination efficiently.

A significant component of this project is the middleware server, which acts as a communication hub between the mobile application, the Master Robot, and the Slave Robots. The system uses multithreading to ensure seamless communication and synchronization between the robots and the user interface. TCP/IP and UDP/IP protocols are utilized to handle data transmission, allowing the system to operate reliably over varying network conditions. The mobile application also provides real-time camera streaming, movement control, and connection status monitoring, offering a user-friendly experience for managing multiple robots simultaneously.

CHAPTER 6

Chapter 6 of the project focuses on system testing and performance evaluation, where various test scenarios are explored. These tests assess the middleware server's ability to maintain stable communication, handle robot commands, and coordinate tasks effectively. Challenges such as network restrictions, hardware balance issues, and modifications to the GoPiGo3 operating system were overcome with innovative solutions. The results confirm that the system is capable of controlling multiple robots through remote mobile management, highlighting its potential for broader applications in fields like logistics and remote surveillance.

In summary, the project demonstrates a scalable, flexible solution for managing multiple robots through a master-slave architecture. By integrating wireless communication technologies, efficient algorithms, and a user-friendly mobile app, it overcomes traditional robotic system limitations, providing a platform for real-time, remote robot management. The project's codebase and resources, including documentation, are available through GitHub <https://github.com/DexterInd/GoPiGo3> and YouTube https://youtu.be/iSyG6WF4z94?si=xs8cL54HFNQrOyQ_, particularly focusing on the GoPiGo3 robotic system.

7.2 Recommendation

The GoPiGo3 robot is powerful and versatile because have the Raspberry Pi run it . However, this project only uses about 50% of the Raspberry Pi's performance, it can more for future improvements, such as better movement, image processing, additional sensors, and other creative idea.

The code currently uses four threads, making it effective in handling various tasks simultaneously, such as sending and receiving data concurrently. However, there are several bugs, both system-related and process-related, due to the Raspberry Pi handling one process at a time. This makes it important to calculate the timing of each thread carefully to ensure smooth operation. **First recommendation: Calculate the time needed for each thread to avoid bottlenecks** and ensure smooth communication between the master and slave robots, using `time.sleep` where necessary to balance task scheduling.

While this project addresses many movement scenarios, including both individual and synchronized movements, it struggles with certain situations. For example, the ultrasound sensor cannot scan some areas, especially those directly in front of the robot. **Second recommendation: Consider integrating additional sensors** like a 3D laser or a fixed map to help the master and slave robots navigate and understand their surroundings more accurately.

Both the master and slave robots are equipped with arms, making them capable of performing various tasks, but they are not fully automated. Currently, they cannot detect objects using their existing sensors, though they are equipped with cameras. **Third recommendation is basic image processing** through the cameras to detect objects and allow the robots to pick it or other actions.

The mobile application is simple but it havd complete status displays and functionality. However, one issue visual feedback is lack to help the user make decisions about the next steps. **Final recommendation is image or video** from the master and slave robots' cameras **put into the mobile application**, enabling the user to better understand the environment and decide on the next action.

REFERENCES

- [1] A Cheong, MWS Lau, E Foo, J Hedley, and Ju Wen Bo, "Development of a Robotic Waiter System" in ScienceDirect, [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405896316322911>(Accessed September. 11, 2023)
- [2]" Guide to warehouse robots: types of warehouse robots, uses, navigation & more". 6river.com[Online].[https://6river.com/guide-to-warehouse-robots/#:~:text=Automated%20Guided%20Carts%20\(AGCs\)%20%E2%80%94,materials%20throughout%20a%20warehouse%20autonomously.](https://6river.com/guide-to-warehouse-robots/#:~:text=Automated%20Guided%20Carts%20(AGCs)%20%E2%80%94,materials%20throughout%20a%20warehouse%20autonomously.) (Accessed September. 11, 2023)
- [3]Opensource.com. "What is a Raspberry Pi?" [Online]. Available: <https://opensource.com/resources/raspberry-pi>. (Accessed: Apr. 18, 2024).
- [4]Raspberry Pi Foundation. "Raspberry Pi 3 Model B+." [Online]. Available: <https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf>. (Accessed: Apr. 18, 2024).
- [5]Raspberry Pi Foundation. "Raspberry Pi Zero 2 W." [Online]. Available: <https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf>. (Accessed: Apr. 18, 2024).
- [6] GoPiGo team. "GoPiGo OS." [Online]. Available: <https://gopigo.io/gopigo-software/>. (Accessed: Apr. 18, 2024).
- [7] GeeksforGeeks. "Introduction to Java." [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-java/>. (Accessed: Apr. 18, 2024).
- [8] GeeksforGeeks. "What is Python? It's Uses and Applications." [Online]. Available: <https://www.geeksforgeeks.org/what-is-python/>. (Accessed: Apr. 18, 2024).
- [9] L. Kalita, "Socket Programming." Available: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=54dc78a981c2e3dcd2f9aacb59491831cf7d5401> (Accessed: Apr. 18, 2024).
- [10] Noboru Noguchi a Jeff Will, John Reid and Qin Zhang , "Development of a master–slave robot system for farm operations" in ScienceDirect.[Online].Available: <https://www.sciencedirect.com/science/article/abs/pii/S0168169904000316> (Accessed: Apr. 18, 2024).

REFERENCES

- [11] Saliyah Kahar, Riza Sulaiman, Antion Satria Prabuwno and Nahdatul Akma Ahmadrn , “A Review of Wireless Technology Usage for Mobile Robot Controller” in ResearchGate. April 2012. [Online]. Available: https://www.researchgate.net/profile/Saliyah-Kahar/publication/232613949_A_Review_of_Wireless_Technology_Usage_for_Mobile_Robot_Controller/links/0fcfd50825b4446099000000/A-Review-of-Wireless-Technology-Usage-for-Mobile-Robot-Controller.pdf (Accessed: Apr. 18, 2024).
- [12] Jorunn D. Newth. “Wi-Fi 7, 6, 5, 4 ... Generations of Wifi and Their Meanings”, eyenetworks.no [Online], <https://eyenetworks.no/en/wi-fi-6-5-4-generations-of-wifi/> (Accessed: Apr. 18, 2024).
- [13]” What are the differences between 2G, 3G, 4G LTE, and 5G networks?”, rantcell.com[Online], <https://rantcell.com/comparison-of-2g-3g-4g-5g.html> (Accessed: September. 11, 2023)
- [14] İ. Ünal, Ö. Kabaş, O. Eceoğlu, and G. Moiceanu, “Adaptive Multi-Robot Communication System and collision avoidance algorithm for precision agriculture,” Applied Sciences, vol. 13, no. 15, p. 8602, Jul. 2023, [Online]. Available: <https://www.mdpi.com/2076-3417/13/15/8602> (Accessed: Apr. 18, 2024).
- [15] Jan Nadvornik and Pavel Smutny ,” Remote control robot using Android mobile device” in ResearchGate.May.2014.[Online] . Available : https://www.researchgate.net/profile/Pavel_Smutny/publication/269299445_Remote_control_robot_using_Android_mobile_device/links/5b4889370f7e9b4637d2f796/Remote-control-robot-using-Android-mobile-device.pdf (Accessed: Apr. 18, 2024).
- [16] GeeksforGeeks. "Waterfall Model - Software Engineering." [Online]. Available: <https://www.geeksforgeeks.org/waterfall-model/>. (Accessed: Apr. 18, 2024).
- [17]JavaTpoint. "Agile Model in Software Engineering." [Online]. Available: <https://www.javatpoint.com/software-engineering-agile-model/>. (Accessed: Apr. 18, 2024).
- [18]JavaTpoint. "Incremental Model in Software Engineering." [Online]. Available: <https://www.javatpoint.com/software-engineering-incremental-model/>. (Accessed: Apr. 18, 2024).
- [19]GeeksforGeeks. "V-Model in Software Engineering." [Online]. Available: <https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/#when-to-use-of-vmodel>. (Accessed: Apr. 18, 2024).

APPENDIX A1

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T1Y4	Study week no.: 2
Student Name & ID: LEE CHIN BIN & 20ACB03441	
Supervisor: Ts Dr. Goh Hock Guan	
Project Title: Master-controlled Networking for Mobile Robotics Application	

1. WORK DONE

Research, purchase, and test the critical components needed for FYP2.

2. WORK TO BE DONE

Discuss how to design and install purchase items into gopigo3 in FYP2

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

-



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T1Y4	Study week no.: 4
Student Name & ID: LEE CHIN BIN & 20ACB03441	
Supervisor: Ts Dr. Goh Hock Guan	
Project Title: Master-controlled Networking for Mobile Robotics Application	

1. WORK DONE

Complete the design and installation of Master GoPiGo3 Robot and conduct installation testing.

2. WORK TO BE DONE

Print 3d part and install other thing to Salve Gopigo3 Robot

3. PROBLEMS ENCOUNTERED

How to balance a robot without destroying the original robot and installing parts.

4. SELF EVALUATION OF THE PROGRESS

I think FYP1 does not have good planning and design

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T1Y4	Study week no.: 6
Student Name & ID: LEE CHIN BIN & 20ACB03441	
Supervisor: Ts Dr. Goh Hock Guan	
Project Title: Master-controlled Networking for Mobile Robotics Application	

1. WORK DONE

Complete the design and installation of Slave GoPiGo3 Robot and conduct installation testing.

2. WORK TO BE DONE

Try to do research and find a solution that allows the mobile phone and Master to communicate without port forwarding.

3. PROBLEMS ENCOUNTERED

The thing can't take power directly from the Raspberry Pi, it needs an additional power supply because some parts require very high current

4. SELF EVALUATION OF THE PROGRESS

I don't have sufficient experience with designing and assembling robots, including installing the ARM, screws, balancing the robot, managing its power, and wiring.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T1Y4	Study week no.: 8
Student Name & ID: LEE CHIN BIN & 20ACB03441	
Supervisor: Ts Dr. Goh Hock Guan	
Project Title: Master-controlled Networking for Mobile Robotics Application	

1. WORK DONE

Find, purchase, and test servers to become middleware for communication between Master GoPiGo3 Robot and Mobile applications

2. WORK TO BE DONE

Try improving the code of the FYP1 Master and Slave GoPiGo3 robots and add new features to the installed runs

3. PROBLEMS ENCOUNTERED

When trying to test some network solutions, buy something that is not intended to solve the problem. And know that servers are not cheap

4. SELF EVALUATION OF THE PROGRESS

I lack research and knowledge in areas such as domain names, virtual private servers, and avoiding unnecessary purchases



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T1Y4	Study week no.: 10
Student Name & ID: LEE CHIN BIN & 20ACB03441	
Supervisor: Ts Dr. Goh Hock Guan	
Project Title: Master-controlled Networking for Mobile Robotics Application	

1. WORK DONE

-

2. WORK TO BE DONE

Still try improving the code of the FYP1 Master and Slave GoPiGo3 robots and add new features to the installed runs

3. PROBLEMS ENCOUNTERED

While improving the code, there were many bugs when doing it in FYP1. After the improvements, there are still many improved bugs.

4. SELF EVALUATION OF THE PROGRESS

Hardware, software, and libraries need to be studied

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T1Y4	Study week no.: 12
Student Name & ID: LEE CHIN BIN & 20ACB03441	
Supervisor: Ts Dr. Goh Hock Guan	
Project Title: Master-controlled Networking for Mobile Robotics Application	

1. WORK DONE

The GoPiGo3 code for Master and Slave communication is complete, including movement control and task completion. The server, as well as communication between the Master and the mobile device, has also been completed

2. WORK TO BE DONE

Improve mobile app user interface and test full FYP2

3. PROBLEMS ENCOUNTERED

In the previous installation work, use too much time

4. SELF EVALUATION OF THE PROGRESS

A good plan is important



Supervisor's signature



Student's signature

APPENDIX A2

POSTER

*Master-controlled Networking
for Mobile Robotics Application*

Introduction
From Karel Čapek's concept to today's reality, robots have revolutionized industries, from education to industrial automation like multiple robot cars optimizing complex tasks through master-slave communication.

Objective

- Master- slave communication
- Mobile Control Application Network communication
- Achieve Collaborative Robot Teamwork

Conclusion
Developing Master-controlled Networking system with GoPiGo3 robots for warehouse simulation, enhancing communication, and providing dedicated mobile control apps.

Developer : Lee Chin Bin
Supervisor :Ts.Dr.Goh Hock Guan

PLAGIARISM CHECK RESULT

PLAGIARISM CHECK RESULT

Turnitin Originality Report

[Document Viewer](#)

Processed on: 12-Sep-2024 18:47 +08
ID: 2451879388
Word Count: 17428
Submitted: 1

FYP2_LEE_CHIN_BIN_CheckpuproseCN.pdf By LEE CHIN BIN

Similarity Index	Similarity by Source
10%	Internet Sources: 6% Publications: 4% Student Papers: 4%

include quoted	include bibliography	excluding matches < 8 words	mode: quickview (classic) report	print	download
1% match (publications) Di-Hua Zhai, Yuangqing Xia, "High-Performance Adaptive Control of Teleoperation Systems", CRC Press, 2023					
<1% match (Internet from 11-Oct-2022) http://eprints.utar.edu.my					
<1% match (Internet from 11-Oct-2022) http://eprints.utar.edu.my					
<1% match (Internet from 11-Oct-2022) http://eprints.utar.edu.my					
<1% match (Internet from 20-Jan-2024) http://eprints.utar.edu.my					
<1% match (Internet from 12-Jun-2022) http://eprints.utar.edu.my					
<1% match (Internet from 20-Sep-2023) http://eprints.utar.edu.my					
<1% match (Internet from 10-May-2024) http://eprints.utar.edu.my					
<1% match (Internet from 18-Aug-2024) http://eprints.utar.edu.my					
<1% match (Internet from 10-Oct-2022) http://eprints.utar.edu.my					
<1% match (Internet from 20-Jan-2024) http://eprints.utar.edu.my					
<1% match (Internet from 20-Sep-2023)					

PLAGIARISM CHECK RESULT

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



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	LEE CHIN BIN
ID Number(s)	20ACB03441
Programme / Course	FICT/CN
Title of Final Year Project	Master-controlled Networking for Mobile Robotics Application

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)
Overall similarity index: <u>10</u> % Similarity by source Internet Sources: <u>6</u> % Publications: <u>4</u> % Student Papers: <u>4</u> %	
Number of individual sources listed of more than 3% similarity: <u>0</u>	
Parameters of originality required, and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

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

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



 Signature of Supervisor

 Signature of Co-Supervisor

Name: Dr GOH HOCK GUAN

Name: _____

Date: 12 September 2024

Date: _____



UNIVERSITI TUNKU ABDUL RAHMAN

**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)**

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB03441
Student Name	LEE CHIN BIN
Supervisor Name	Dr GOH HOCK GUAN

TICK (√)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

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

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date:12/10/2024