Autonomous Object-Seeking Robot using Smartphone for Vision and Control

Prepared by:
Karan Joseph Abraham (ABRKAR004)

Prepared for:
Prof. Simon Winberg EEE4113F
Department of Electrical Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the requirements for a Bachelor of Science degree in Electrical and Computer Engineering.

May 23, 2025

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work, and is in my own words (except where I have attributed it to others).
4. I have not paid a third party to complete my work on my behalf. My use of artificial intelligence software has been limited to such shown in Appendix B.
5. I have not allowed and will not allow anyone to copy my work with the intention of passing it off as his or her own work.
6. I acknowledge that copying someone else's assignment or essay, or part of it, is wrong, and declare that this is my own work.

Signature: 
Word count: 5202
Karan Joseph Abraham

Date: May 23, 2025

Acknowledgements

I would like to thank the following people for helping me with this project:

Prof. Simon Winberg, my supervisor: for supporting me financially in acquiring the robot kits. For his advice during our weekly meetings. It was a joy working with him.

Dr Chris Abraham, my brother: for reviewing this project report and always willing to give advice at any time, whether for this project or anything throughout my degree journey.

Mishay Naidoo: for helping me with an ESP32 related issue.

My parents: for supporting me financially in enrolling for this course and throughout my degree.

The African Robotics Unit: for giving me access to the lab and its equipment. As well as providing their support and a space for me to work on the project.

Aditya Nair, my friend: for supporting me and giving advice throughout the project.

Abstract

This thesis explores repurposing old phones for controlling robots. The objective is to autonomously control a wheeled mobile robot. To date, there are many projects using Android phones to control robots. However, none of these projects use the phone's camera for autonomous control. The project investigates the feasibility of using a phone's camera to measure relative position of an object and determine the control action required to approach the object. In order to do this, a wheeled robot with an arm is assembled and interfaced with an Android smartphone using a microcontroller.

It was found that the phone could be effectively integrated in the robot for autonomous control. Lateral position could be determined using the phone's camera; however depth perception requires an additional sensor which provides accurate readings. Additionally, in order to pick up the block, accurate and reliable servos are required to manoeuvre the arm. It was also found that even a low-end phone is sufficient for the processing needs of autonomous robot control.

Contents

List of Figures

Abbreviations

API Application Programming Interface

COM Centre of Mass

GA Graduate Attributes

JVM Java Virtual Machine

SDK Software Development Kit

SoC System on Chip

TCP Transmission Control Protocol

UCT University of Cape Town

WAN Wide Area Network

Chapter 1

Introduction

The purpose of this project is to see how well smartphones can be used in an embedded system. This project explores this by repurposing an old smartphone to be used as the 'brain' of a robot. The phone's camera and System on Chip (SoC) are leveraged to autonomously control the robot.

This robot could have many applications. Such an application could be tidying up a room. There are several aspects to tidying up: The robot needs to find an object out of place; go to the object; and put the object back in its place.

The proposed system comprises a phone and embedded hardware, which consists of a wheeled robot, a robotic arm, and a microcontroller to interface them with the phone. This is illustrated in figure 1.1.

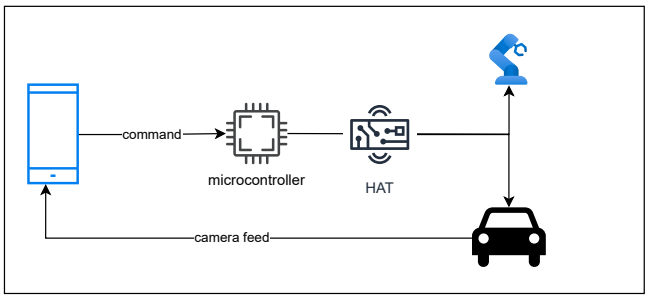


Figure 1.1: Basic proposed system

1.1 Background and Motivation

Waste electronic equipment in South Africa was measured to be 6.6 kg per capita in 2014. This was above the global average measured [5]. This is bad for the environment and for the economy as more electronics need to be manufactured unnecessarily. Most of the waste can be reused for specific purposes i.e. embedded applications. For example, if a phone's screen is irreparably damaged, but it's camera and SoC are still working, it could be reused as a sensor and controller for an autonomous robot. This is cheaper than buying a camera module and a powerful microcontroller. Phones are more powerful than microcontrollers, so they are more than capable to be used in embedded applications. Also, all smartphones are equipped with cameras with good enough resolution that image processing can be done with it.

1.2 Objectives

The objective of this project is finding out how a phone can be re-purposed for an embedded system. In this project, a phone is used as a sensor and a co-processor for a retriever robot. The robot should be able to find and fetch specific objects. In order to find and approach the object autonomously, the phone's camera and processor (and memory) are used. This robot should be able to move along a flat surface in order to retrieve the object, and therefore needs wheels.

The primary objective is to configure a phone to autonomously control a wheeled mobile robot (for finding and approaching the object).

In order to fulfil the primary objective the subsystems need to be implemented:

1. Communication between the phone and the microcontroller needs to be programmed.
2. The phone needs to be programmed to find the object and send commands to the microcontroller.
3. The microcontroller needs to be programmed to control the actuators of the robot.

In order for the phone to autonomously control the robot, the following procedure needs to be implemented:

1. The phone's camera should be used to see what's in front of it
2. The phone's processor should be used to tell the robot what to do
3. The phone needs to send the command to a micrcontroller
4. The microcontroller needs to move the relevant motors to fulfil the command

An additional objective of the robot is to pick up the block (for retrieval), therefore the robot needs a robotic arm. The robotic arm would be controlled by the microcontroller.

1.3 Scope and Limitations

This project is a high-level software design / experiment. Existing hardware is used as well as existing Application Programming Interface (API)s to control the hardware.

In this project, the object to be retrieved is a $2cm$ wide blue block. The phone's camera and processor are used to find the block using colour classification. The processor is also used to determine the lateral (x) position of the block. The depth (y) position of the block is determined using the embedded system (the ESP32, actuators and ultrasonic sensor) only. The phone is only used for finding and approaching the block. The picking up of the block is executed by the embedded system alone. The robot does not bring the object back to its initial location in this iteration of the project. This would be implemented in future work.

No computer vision object detection is to be used. Only rudimentary image analysis is used. Therefore, there are limitations on what objects can be used in what background.

The robot is small (about soccer ball sized) and surface based. The arm is small, so has limited reach (in vertical axis and out of rover bounds). Arm is weak so can only pick up objects under a certain mass. It cannot even reliably hold its own weight when the arm is out-stretched. The arm has a simple gripper, so it can only pick up orthogonal shapes.

1.4 Report Outline

The report outline is described in the following list.

- **Chapter 1**: The Introduction provides a background to the study, sets out its purpose and the objective.
- **Chapter 2**: The Literature Review section looks at the state of the art for phones being used to control robots autonomously.
- **Chapter 3**: The Methodology section shows the steps taken to fulfil the system requirements.
- **Chapter 4**: The Design section describes the system's subsystems and how they fit together
- **Chapter 5**: The Experimentation section shows the experiments done to validate if the robot is meeting the requirements. It shows the result of the experiments, which are analysed and explained.
- **Chapter 6**: The Conclusions summarize the findings of the study: how feasible using a phone is to control a robot. Finally, recommendations for future work on the project are discussed.

Chapter 2

Literature Review

This chapter reviews the literature related to the topic of this project. Chapter 1 states the objective of this project, testing the feasibility of repurposing a phone to autonomously control a robot. No studies were found about this specific topic; however, a few tangential topics were found and reviewed.

2.1 Interfacing Android Phone with Embedded System

[6] interfaces a smartphone to their robot over wifi using Wide Area Network (WAN) (using http). The robot is manually controlled, but commands are sent in real time without significant lag. They use a Raspberry Pi to interface a phone and a webcam to an Arduino which controls the motors. This is illustrated in figure 2.1. They use Flutter to develop the mobile phone application to run on multiple platforms including Android and IOS.

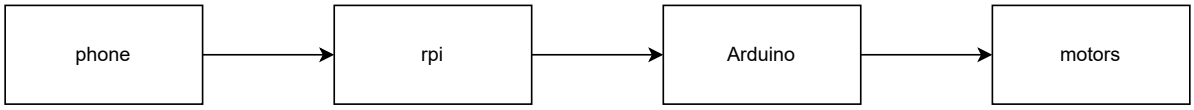


Figure 2.1: Design of rescue robot

[7] uses an Android phone in conjunction with an NXT brick (a microcontroller by Lego) to control a robot (by Lego Mindstorms). However, they do not show any of their testing in the paper. They claim that they can send commands from the Android phone to the NXT brick. They use a ultrasonic sensor which measures how far obstacles are away from it, but do not produce any accuracy or repeatability tests on it.

[8] built a smart DC home system using the STM32F407 Discovery microcontroller and Node-MCU ESP8266 microcontroller board. The system is aimed at reducing household energy consumption. They use the ESP32 board to send the energy consumption data to an Android phone for monitoring. They use the Transmission Control Protocol (TCP) to send the data. They use MIT App Inventor, a block-based visual editor, to develop the mobile phone application.

2.2 Microcontrollers Used to Control a Robot

[9] design and implement a wheeled robot with an ultrasonic sensor (to measure distance from objects directly in front of it). They claim to successfully use a Raspberry Pi 3 to control a robot.

[10] uses an ESP32 to control a car robot using Bluetooth. The current consumption of the ESP32 microcontroller is $280mA$. With wifi and bluetooth is $750mA$.

2.3 Arm Control

[11] constructed a robotic arm for their rover. They achieved a positioning accuracy less than than one centimeter (for a target placed less than 1 m away) and a rotational accuracy better than 0.5 degrees.

2.4 Sensing

[1] conducted an experiment on a wheeled mobile robot comparing the performance of control using visual feedback and odometric feedback. They performed parking and trajectory following tasks. In the parking task, odometry had a significant final residual error of order -2. The final error when using visual feedback was more insignificant of order -3 (about 10 times smaller). This is shown in Figure 2.2. This makes visual feedback more suitable for accuracy. They found this improvement of visual feedback to be at the cost of $5ms$ sampling time.

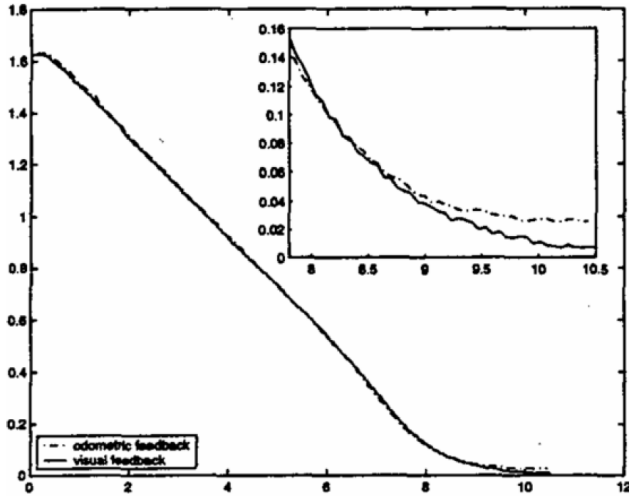


Figure 2.2: Plot of odometric feedback error compare to visual feedback error [1]

Chapter 3

Methodology

This chapter describes the method of implementing the project. The time-frame of the project was three months. It started with a week of reviewing available robots and literature. This was followed by about two months of practical work. This left about two weeks for writing up the report.

The timeline of the project is shown in figure 3.1.

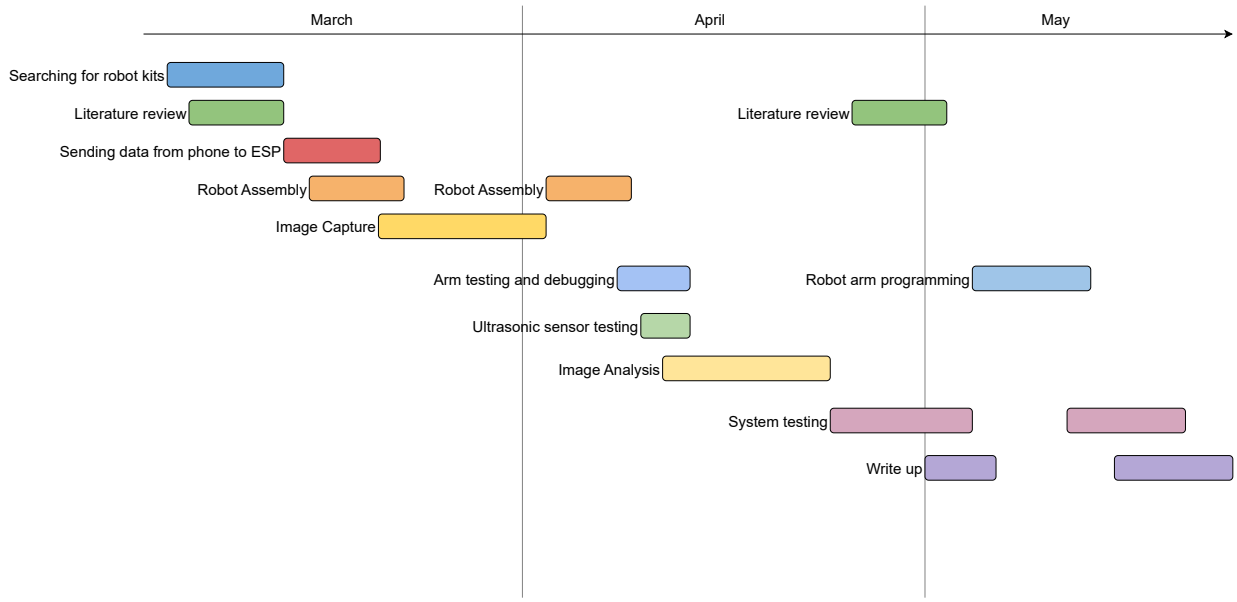


Figure 3.1: Timeline of work done on project

3.1 Functional Requirements

Initial requirements were based on the topic brief provided by the supervisor of this project, Prof. S. Winberg. During the initial stages of the project, the requirements were refined according to the capabilities of the robot kits acquired. The requirements were also adjusted for the time frame of the project. The final refined requirements are as follows:

1. Phones camera must be able to capture photo frames
2. Phone must be able to analyse frames quickly
3. Phone must be able to communicate with microcontroller

4. Microcontroller must be able to read from sensors and control actuators
5. Wheels must be able to move robot
6. Servos must be able to manoeuvre arm
7. Ultrasonic sensor must be able to measure distance of object in front of robot

3.2 Design Choices

A rover and a robotic arm (with a rotating base) controlled by a microcontoller were looked for on the website of RS components, Sparkfun, Takealot, and Robotico. Products found locally were favoured as it would avoid import tax and expedite delivery. This is important as the budget and time-frame of this project was limited.

3.2.1 Microcontrollers

A microcontroller is needed to interface the phone with the actuators. The Raspberry Pi 3, ESP32 and Arduino have been used to control robots as shown in the literature (chapter 2). The Acebott Smart Car Starter Kit and their Robot Arm Car Expansion Pack are designed to interface with each other through a ESP32-WROOM-32 module. The module is equipped with Bluetooth, Wi-Fi and a USB-C port, either which could be used to interface with the phone. Therefore, the ESP32 was used.

3.2.2 Wheeled Robot

The mobile robot requires wheels to move to the object. Several robot car kits are available commercially which could be used to implement the retriever robot. The two most suitable kits are shown in table 3.1. The Arduino Smart Robot Car Kit [1] is a wheeled robot that is interfaced with the Arduino UNO R3.

The Acebott Smart Car Starter Kit (QD001) [2] uses an ESP32 board.

Product	**Price (R)**
Arduino Smart Robot Car Kit	1200
Acebott Smart Car Starter Kit (QD001)	1870

Table 3.1: Price comparison of robot car kits

3.2.3 Robotic Arm

When the robot reaches the object, it needs an arm to pick up the object. Table 3.2 lists available arm kits available commercially. The Acebott 4-DOF Robot Arm Kit [3] uses an ESP. It is more expensive, but could possibly be used with the cheaper Arduino Smart Robot Car Kit. Together they would total R2450. The Acebott Robot Arm Car Expansion Pack (QD007) [4] is an expansion pack of the

[1] https://www.takealot.com/arduino-smart-robot-car-kit/PLID72722636?srsltid=AfmBOoqja_I-hL4aBQWke8nXlvMZ0guEQaW8KKtGjUY9kBOOhUY8ex8v
[2] https://acebott.com/stem-kits/explorer-series/acebott-omni-wheel-car-robot-starter-kit-collection/
[3] https://robotico.co.za/products/acebott-4-dof-robot-arm-kit
[4] https://robotico.co.za/products/acebott-robot-arm-car-expansion-pack

Acebott Smart Car Starter Kit (QD001). Together they would total R2490. This is only slightly more expensive, and it is already designed to integrate together. Therefore, the QD001 + QD007 were purchased.

Product	**Price (R)**
Acebott 4-DOF Robot Arm Kit	1250
Acebott Robot Arm Car Expansion Pack (QD007)	620

Table 3.2: Price comparison robot arm kits

While the robot kits were being delivered the literature review and phone programming were started.

3.2.4 Android Application Developement Platform

Flutter and MIT App Inventor are used in the literature (chapter 2). They both have the advantage of being cross platform. MIT App Inventor is a block-based visual editor making it easier to use for those with little experience in software development. However, it lacks many features compared to Flutter and Android Studio.

Android studio is aimed at developing native applications for Android. Developing a native Android application has platform specific optimizations making it faster than cross platform applications [12].

For the purpose of the objective: delivering a phone-controlled robot, the phone needs to process the field of view and tell the robot what to do at fast rate. Additionally, Android phones are by far the most predominant smartphones, especially in the lower-end market. It is also open source, making it more suitable for customization (which is useful in embedded applications). Therefore, Android Studio was chosen as the development platform for the phone.

3.3 Reviewing Literature

Literature was explored on IEEE Xplore digital library . Studies on developing a rover with a robot arm autonomously controlled by a phone were not found, so studies on the subsystems were explored:

- robot controlled by user using phone
- phone interfacing with robot using microcontroller
- types of odometry (sensing position of robot)

3.4 Implementation

3.4.1 Sending data from phone to ESP

An Android phone and an ESP32-WROOM-32 microcontroller were programmed to communicate via Wi-Fi. It was then tested on a test Android application that was developed and using a PC application (to output the data that the ESP32 received).

3.4.2 Smart Car

When the kits arrived, the Smart Car was assembled. The manufacturer of the robot kits, ACEBOTT, provided tutorials for assembly. They also provided an API for controlling the robot. They provided tutorials for using the API which were read and used for the application of this robot.

3.4.3 Image Capture and Analysis

The Android phone application was programmed using object-oriented programming principles. Tutorials were then looked for to access the phone's camera. Many did not work because the APIs used were outdated, due to the rapid updates made on Android's API. Android Developers' CameraX tutorial [13] was used, but the application threw compatibility errors. This problem was resolved by replacing Java 8 methods with Java 11 (detailed in chapter 4). The image analysis implementation involved finding an adequate condition for 'blue colour classification' (detailed in the Image Analysis section in chapter 4).

3.4.4 Robot Arm Programming and Debugging

A significant portion of the project timeline was required when programming the robot arm. The arm caused the robot started behaving unexpectedly. Eventually, it was discovered that it was due to issues regarding the shoulder servo (detailed further in chapter 5). The shoulder servo had been damaged, which was affecting the other actuators. Eventually the issue was resolved, and the arm was made functional again.

The final part of the practical work was testing the integrated system. Integration testing required a significant amount of time since many slight modifications were made after testing.

3.5 Report Writing

A large part of the report writing was dedicated to creating the figures. The series of photos had a number of sub-figures, which Latex is does not format well automatically. Therefore, custom Latex code was required to display the series of photos concisely. Additionally, technical diagrams were made using Draw IO to make the report more clear and concise.

Chapter 4

Design

4.1 Introduction

This chapter describes how the robot works. The robot and its procedure are described, then the subsystems are described. The code for the subsystems is available in the project GitHub repository [14].

Chapter 1 stated the objective of the robot and the subsystems required as shown in figure 1.1. The robot comprises a chassis, wheels, a arm, and computing hardware: Android smartphone, ESP32, and HAT (Hardware Attached on Top). This model is similar to the model designed by [6] except the raspberry pi (the middle man between the phone and the microcontroller) is cut out. Instead the the phone sits on the robot and connects to the microcontroller via Wi-Fi direct. Additionally, the robot has an ultrasonic sensor, which is interfaced with the ESP32. The implemented system is illustrated in figure 4.1.

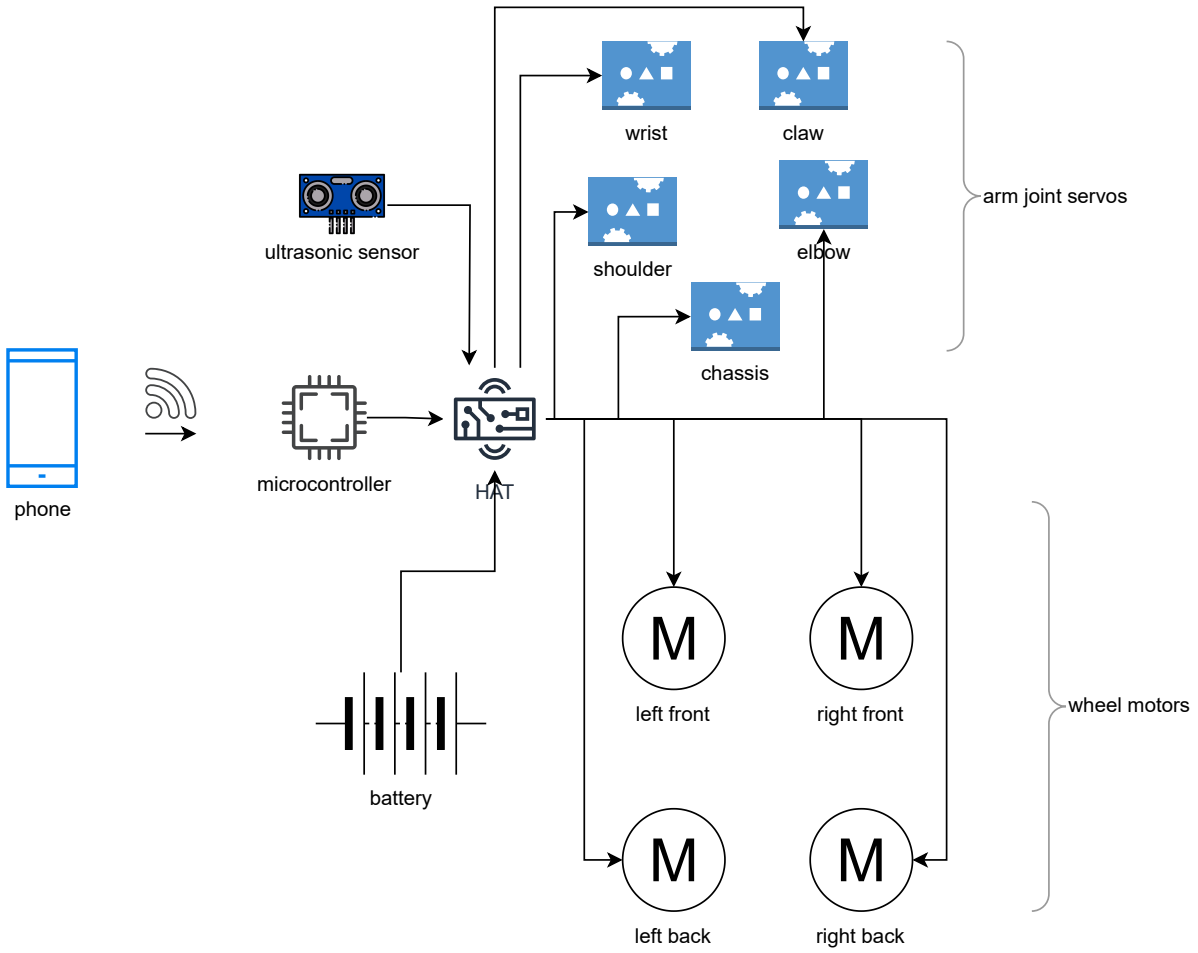


Figure 4.1: Diagram of the implemented system

4.2 Physical Design of Robot

The Smart Car was assembled and tested. It can move forward and backward. It has mecanum wheels; therefore, it can turn left and right on its axis, and it can move directly leftward and rightward. The wheels are controlled individually by a motor each. Thereafter, the Robot Arm was assembled with the Smart Car. The arm has 5 degrees of freedom. It can: turn the entire arm in the yaw direction, pitch the entire arm, pitch the elbow, roll the wrist and open/close the claws. Each degree of freedom is controlled by servos. The robot has two sensors, a line tracing sensor and an ultrasonic sensor. Only the ultrasonic sensor is used for the purposes of this project. The actuators and sensor are controlled by the ESP32. The phone was integrated on board the robot. Figure 4.1 shows a diagram of the subsystems and how they are connected.

Figure 4.2 shows the final assembled robot. The phone is placed on the robot such that its camera is in the middle of the robot. This ensures that $x = 0$ is at the centre of the camera field of view. The coordinate system of the robot is illustrated in figure 4.3.

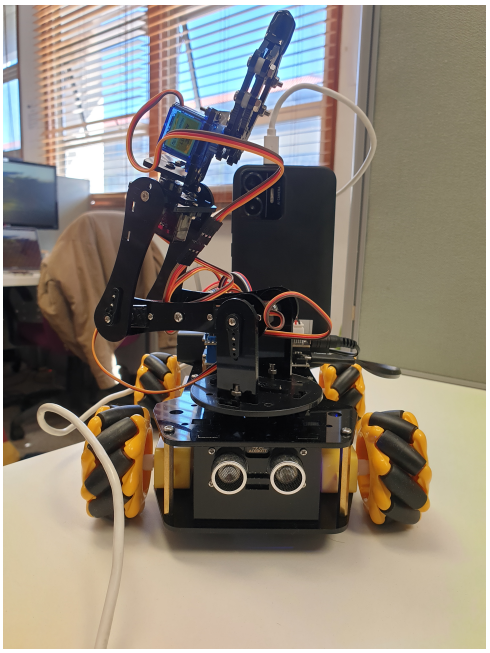


Figure 4.2: Photo of robot

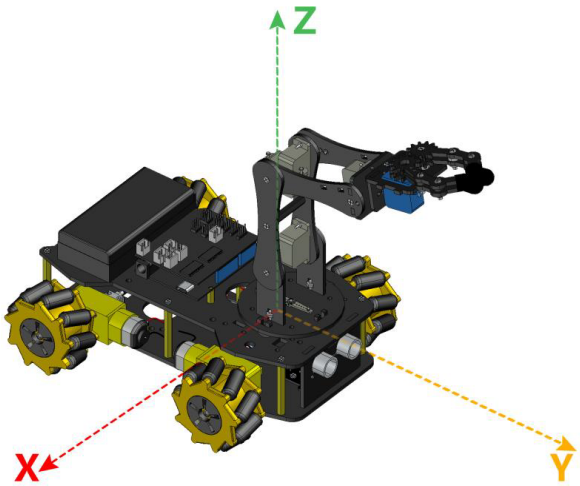


Figure 4.3: Absolute coordinate system

4.3 Robot Task Procedure

The objective of the robot is to retrieve an object. This robot is programmed to retrieve a blue block: 2 cm wide cube, made of wood. The block has to be lying on the same surface as the robot. The Android phone is used for aligning the robot to the block. Once the robot is aligned with a certain range, it proceeds to move forward. Since the robot is not perfectly aligned, as it moves forward the misalignment will increase (as it moves closer to the object the angle error increases) and may fall out of the alignment range. Then the robot would have to realign and move forward again. This would be repeated until the object falls out of the robots field of view. At this stage the object would be within reach of the robot. Ideally, the object would also be directly in front of the robot. Therefore, the robot would use its ultrasonic sensor to determine the distance of the block from the robot, and then would

move the arm to block. It would pick up the block. The final stage (stage 4) would be to bring it to the robot's initial position; however this stage is not implemented in this design.

The procedure is illustrated in figure 4.4 .

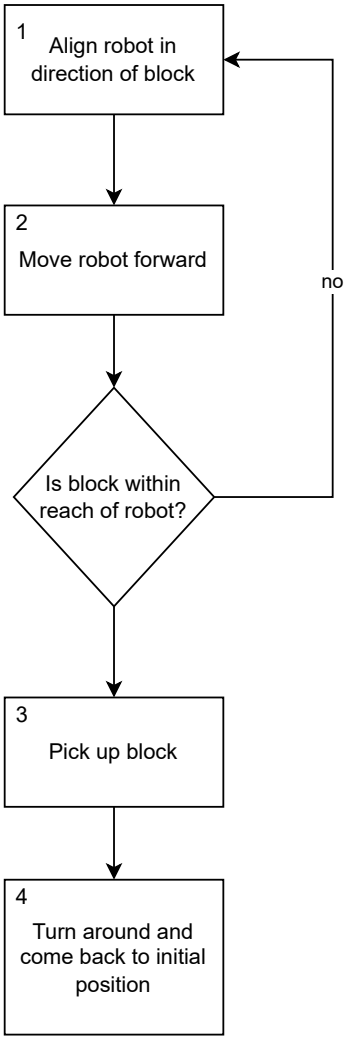


Figure 4.4: Block Diagram of Robot Task Procedure

This procedure is executed by the robot by running `My Camera Capture App` on the Android phone and `move-to-and-pick-up-block.py` on the ESP. This is illustrated in figure 4.6. `move-to-and-pick-up-block.py` is described in algorithm 1. The python script can be found under `Robot Code` in the project repository [14].

Algorithm 1 Entire program running on ESP32

```
Initialize arm servos
Move arm to out-of-way position
Activate WiFi access point
Set up TCP socket
while true do
    Establish connection to phone
    Receive command
    if command = 'l' then
        Turn robot left
    else if command = 'r' then
        Turn robot right
    else if command = 's' then
        Stop robot
    else if command = 'f' then
        Measure distance using ultrasonic sensor
        if distance > 14 cm then
            Move robot forward
        else
            Stop robot
        end if
    else if command = 'd' then
        Stop robot
        Open claw
        Move arm to block
        Close claw
        Lift arm
        Exit loop
    end if
end while
```

Stage 1: Align to block

This subsection describes the first stage of the robot's task procedure, which is how it aligns to the block.

1. The phone camera captures a photo which the phone analyses to determine the lateral position of the object, relative to the robot.
2. The phone accordingly sends a command to the ESP32 such as 'left' to turn left.
3. The ESP32 listens for commands and moves the actuators to execute the command.
4. The action is captured by the camera which is analysed to determine the updated displacement of the object, and the cycle is repeated.

The cycle is illustrated in figure 4.5.

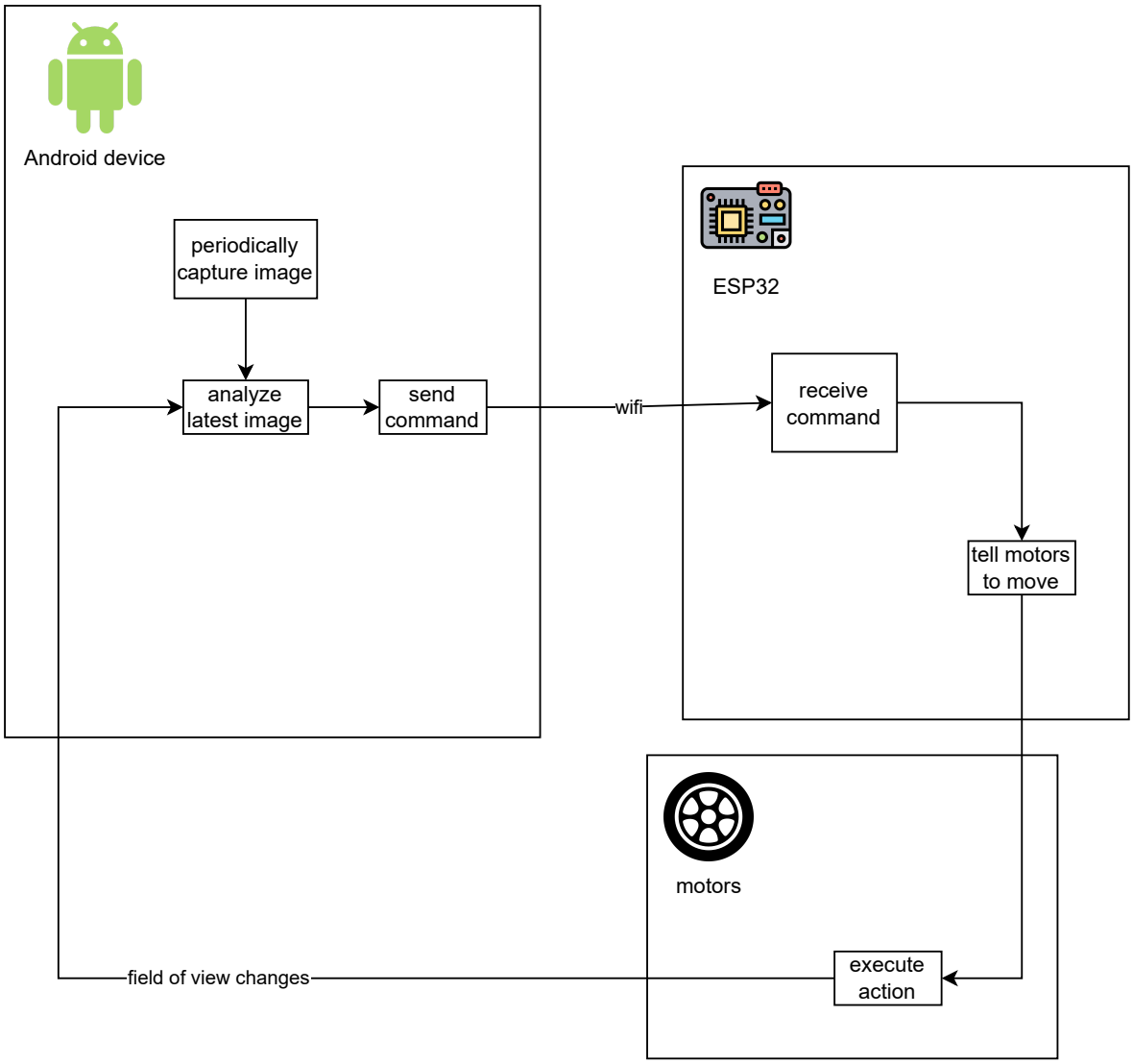


Figure 4.5: Block diagram depicting the align process

The cycle is repeated until the block's lateral position is in the centre of the camera's frame. At this point the phone stops turning the robot.

The programming part of the system consists of the Android environment and the ESP32 environment. A flowchart of the code is shown in figure 4.6.

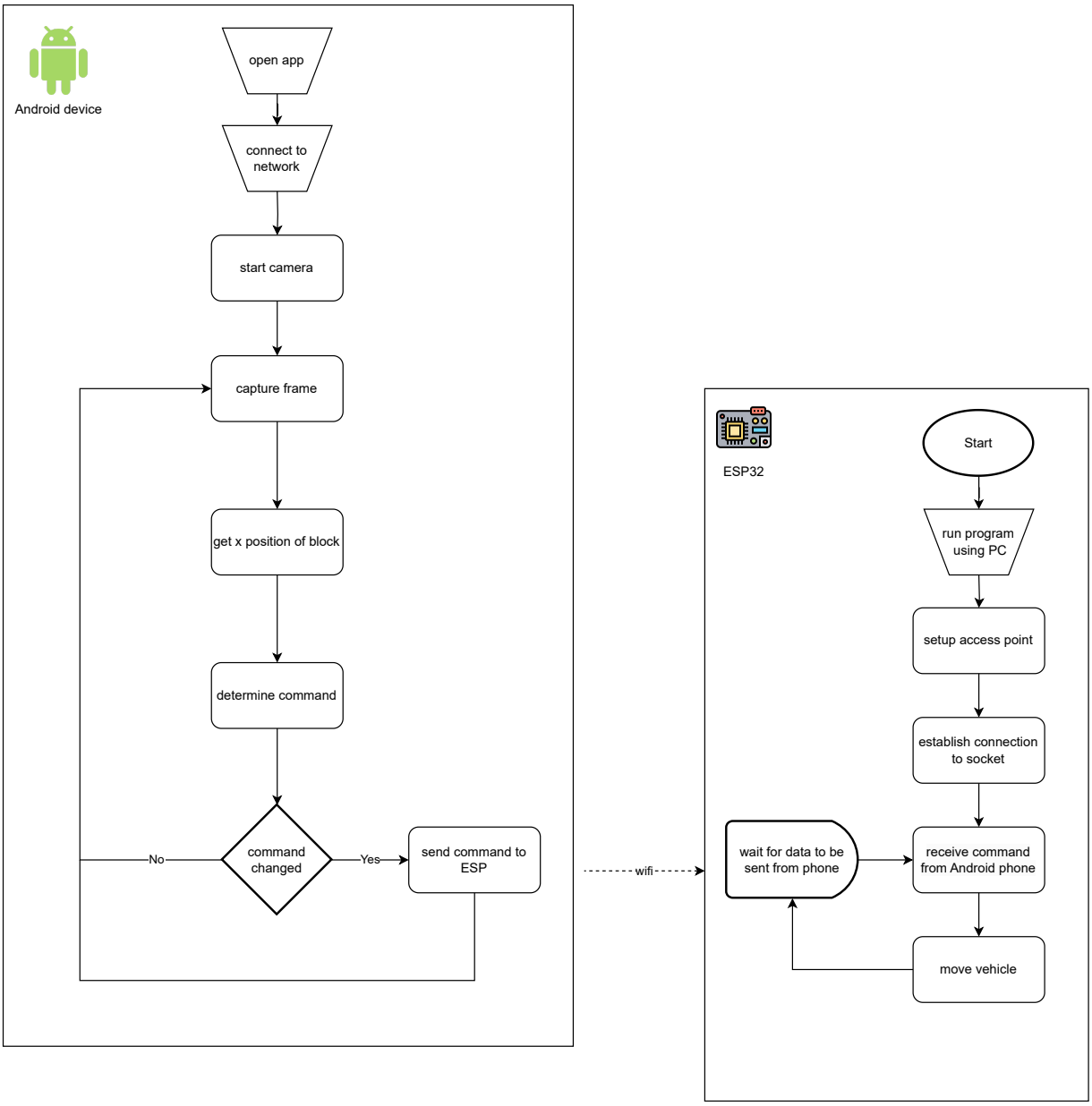


Figure 4.6: Flowchart of code for aligning

Stage 2: Move towards block

Once the robot is aligned to the block, the phone moves the robot forward. It keeps moving forward until the block is out of alignment or the block is out of frame. When the block goes out of frame, then it is within reach of the robot's arm. The implementation for this stage is detailed in the Android Software section of this chapter.

Stage 3: Picking up block

The picking up procedure is as follows:

- Open claws

- Move end-effector to block
- Close claws
- Pick up block

The robot should know where the block is in space (relative to it). Using the coordinate system shown in figure 4.3, the robot should know the x, y and z positions. The z position is always 0 because as stated in the scope of the project, the block is placed on the same surface as the robot. The x position should be approximately 0 because stage 1 aligns the robot to the block. Finally, the y position is measured using the ultrasonic sensor. However, during testing it was discovered that the sensor can be inaccurate. The implementation for this stage is detailed in the Embedded System section of this chapter.

4.4 Android Software

This section describes the processes in the Android application: `My Camera Capture App`. This Android application is able to find the blue block, determine its lateral position, and send commands to the ESP32 (based on the block's position). The application is run on a Hisense U965 smartphone with 2 GB RAM (plus 2 GB virtual RAM if there is space on ROM), running on Android 11. It has a 5 MP camera on its back.

The processes are image capture, image analysis, and sending commands. The Android code is illustrated in figure 4.6.

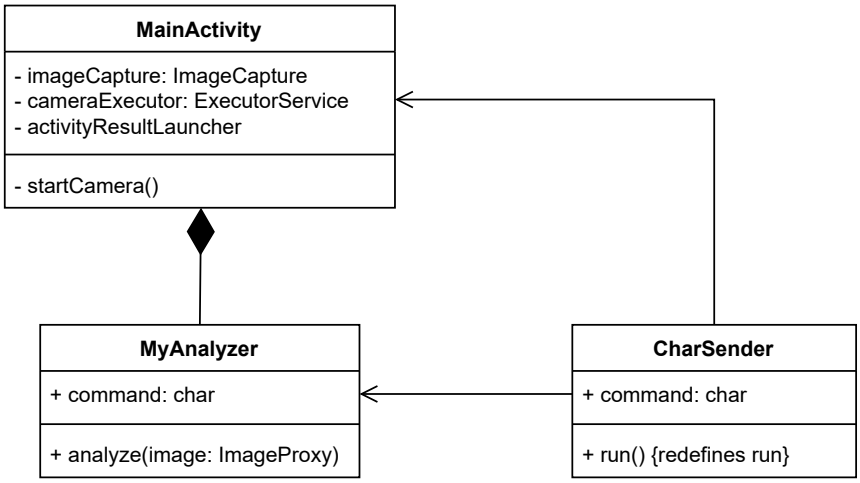


Figure 4.7: UML diagram of Android application

The application is design using object-oriented principles. It is made up of three classes: `MainAcitivity`, `MyAnalyzer` and `CharSender` as shown in figure 4.7. `MainActivity` is the main class that is the entry point of the application. It contains the code for image capture and analysis. The analysis is handled by an inner class, `MyAnalyzer`. `CharSender` is a separate class for sending data via Wi-Fi to the ESP. It used by `MyAnalyzer` for sending commands. `CharSender` is also used in `MainActivity` for sending a

stop command before the application goes inactive (e.g. user minimizes application or turns off phone screen).

The application was developed in Android Studio using Java, Kotlin and Gradle. Kotlin is the new official programming language used in Android development. It is interoperable with Java. This means Java classes can be used within Kotlin. Kotlin uses Java Virtual Machine (JVM) to compile to byte-code.

The versions of the software used are stated below:

- Software Development Kit (SDK): 24 - 35
- JVM: 11
- Gradle: 8.11.1
- Android Gradle Plugin: 8.9.0

The source code for the classes can be found in the project repository [14] under the My-Camera-Capture-App/app/src/main/java/com/example/myapplication folder.

4.4.1 Image Capture

This part of the program was implemented with the help of Android Developer's CameraX tutorial [13] .

When following the tutorial [13] , the following runtime error was encountered:

> Execution failed for task ':app:compileDebugKotlin'. Inconsistent JVM-target compatibility detected for tasks 'compileDebugJavaWithJavac' (1.8) and 'compileDebugKotlin' (11).

Java 1.8 is another name for Java 8. [13] uses methods from Java 8. Using Java 11 methods instead resolved the error. In `build.gradle` file (link in Appendix C), under `compileOptions`, `JavaVersion.VERSION_11` was used.

The `CameraX` library was used to capture the stream of photos. This is implemented in the `startCamera` function in `MainActivity`. A `cameraProviderFuture` is instantiated. Within `cameraProviderFuture`'s listener, algorithm 2 is implemented.

Algorithm 2 Start Camera with Preview and Image Analysis

```
1: cameraProvider ← cameraProviderFuture.get()
2: cameraSelector ← selectCamera(BACK_CAMERA)
3: imageAnalyzer ← ImageAnalysis instance
4: imageAnalyzer.setAnalyzer(cameraExecutor, MyAnalyzer instance)
5: cameraProvider.unbindAll()
6: cameraProvider.bindToLifecycle(this, cameraSelector, [preview, imageAnalyzer])
```

`imageAnalyzer.setAnalyzer(...)` sets `imageAnalyzer` to receive and analyse images. This signals to the camera that is should start sending photo frames [15].

`cameraProvider.bindToLifecycle(...)` activates the camera and starts sending frames to `imageAnalyzer`'s use case.

Therefore, frames are captured periodically and passed to the `analyze` function (defined in the `MyAnalyzer` class). The `analyze` function runs the analysis on the latest frame.

4.4.2 Image Analysis

This phase is implemented in `MyAnalyzer`'s `analyze` function (see figure 4.7). `analyze` is called periodically as the camera captures a new frame and `analyze` finishes executing. By the time `analyze` finishes executing, the camera would have captured many frames. `ImageAnalyzer` by default only uses the latest frame and discards the older frames [15].

Blue pixel classification

The program iterates through every pixel of the frame and identifies the blue pixels. A pixel is classified as blue if its colour components satisfy the following conditions:

$$b > 2r \quad \text{and} \quad g > r$$

where r, g, and b denote the red, green, and blue value of the pixel, respectively. This threshold was established through experimentation.

Lateral position of block

In each iteration, if the pixel is classified as blue, its x position is added to a weighted sum. The weighted sum is divided by the total mass to determine the centre of mass x coordinate (of the block). This is illustrated in equation 4.1.

$$\text{Average x-coordinate of blue pixels} = \frac{\sum x_{\text{blue pixel}}}{N_{\text{blue pixels}}} \tag{4.1}$$

Determining command

Once the position (in the x axis) is determined, a command is determined based on the position. This is described in algorithm 3. This code makes the robot approach the block. If the block moves out of frame, it likely means the robot has arrived at the block and approach stage is done. If the camera can still see the block, and the block is in alignment within a range, it moves forward. If the block is out of alignment it should move left/right. The code described by lines 5 and 6 in algorithm 3 was added because the minimum speed of the robot's wheel motors were too fast, causing massive overshoot. To solve this problem, the phone sends turn and stop commands alternately to effectively slow down turning speed further.

If the determined command is different to previous command, then the command is sent to the ESP. If the command is unchanged, nothing is sent. The command is sent to the ESP32 using `CharSender`.

Algorithm 3 Determine command

```
if (Block is not in frame) then
    command ← 'd'                                   ▷ done
else if (Block is within setpoint bound) then
    command ← 'f'                                   ▷ forward
else if (Previous command was 'l' or 'r') then
    command ← 's'                                   ▷ stop
else if (Block is in the left half of the image) then
    command ← 'l'                                   ▷ left
else if (Block is in the right half of the image) then
    command ← 'r'                                   ▷ right
else
    command ← 's'
end if
```

4.4.3 Sending Command

An instance of `CharSender` can be run in a thread (parallel to the main thread). `CharSender` is a Java class that provides for sending character type data to the ESP.

4.5 Communication between Phone and ESP

The phone determines the action required. This needs to be sent to the ESP32 to actuate the servos to execute the action.

Wi-Fi was used to connect the phone to the ESP, as shown in figure 4.8.

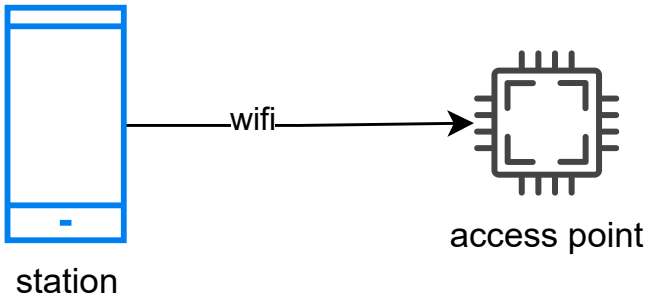


Figure 4.8: Phone connection to ESP

The ESP32 (hosts a Wi-Fi server) sets up a Wi-Fi hotspot called `Robot_Arm_Car`, to which the smartphone connected.

```
ap = network.WLAN(network.AP_IF)
ap.config(essid='Robot_Arm_Car', password='12345678')
ap.active(True)
```

On the ESP32 (in Python) a TCP socket is created which is bound to port 80. The IP address of the ESP32 is “192.168.4.1”.

```
addr = socket.getaddrinfo('0.0.0.0', 80)[0]
```

```
s = socket.socket()
s.bind(('0.0.0.0', 80))
s.listen(1)
```

On the smartphone (in Java/Kotlin) a socket client is created to connect to the corresponding IP address and port.

4.6 Embedded System

4.6.1 Hardware Connections

The robot has two sensors: ultrasonic sensor and line tracing sensor. It has two sets of actuators: 4 motors for the wheels and 5 servos for the arm.

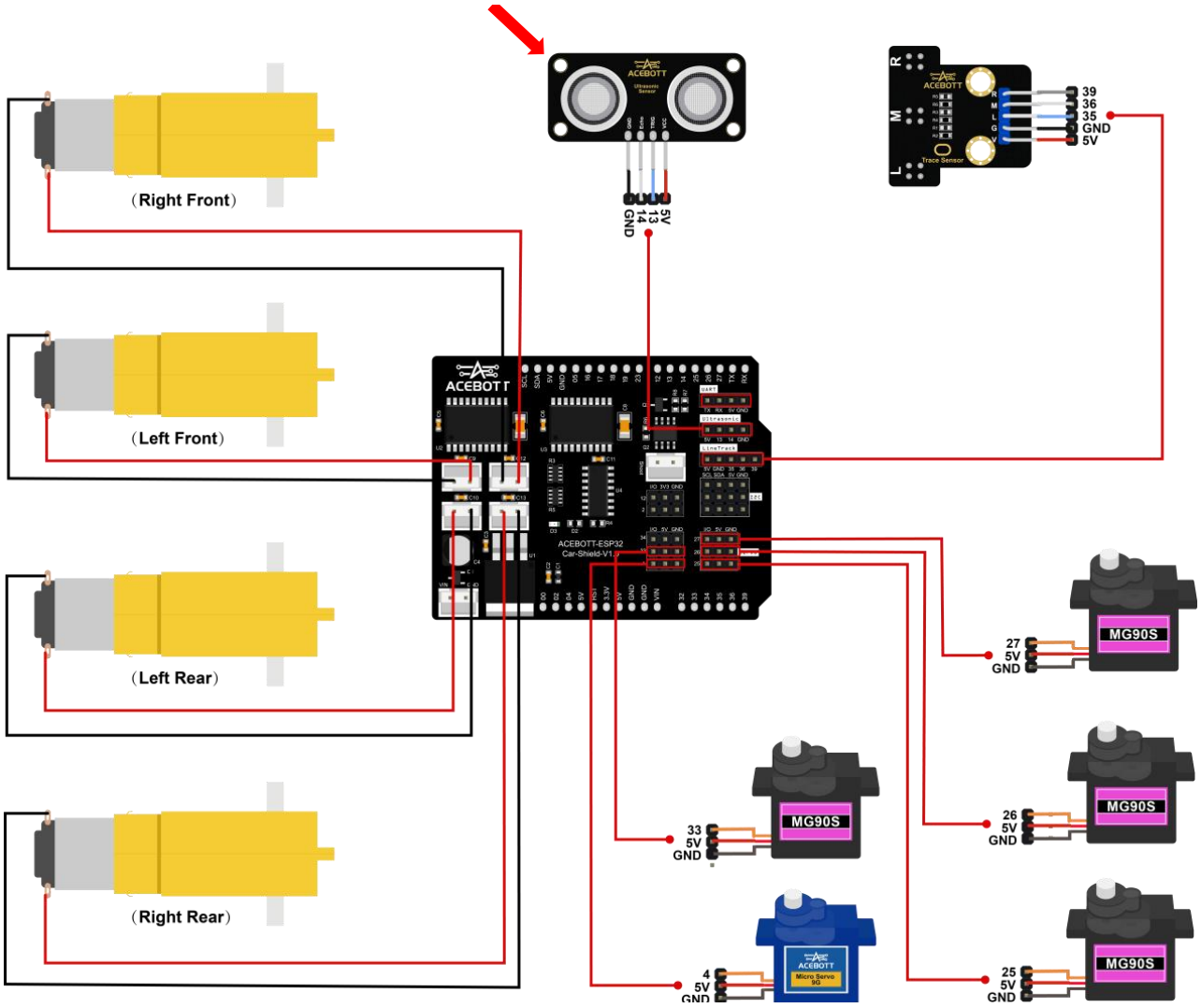


Figure 4.9: Circuit diagram consisting of HAT, actuators and sensors [2]

Figure 4.9 shows where the sensors and actuators are connected on the ESP32 board. Each servo's corresponding GPIO pin is shown in table 4.1.

Therefore, the final corresponding connection of the five servos of the robot arm are:

Servo	**GPIO Pin**
Chassis	25
Shoulder	33
Elbow	27
Wrist	26
Claws	4

Table 4.1: Servo GPIO pins

The claw servo moves from 90 to 180 degrees. The other servos move from 0 to 180 degrees. All servos (except the claw servo) were initialized to 90 degrees before assembly. The initial placement of the claw resulted in it not being able to grip the block. This is because at the minimum claw angle, the claws were not close enough to apply sufficient friction to pick up the object. Therefore, to solve this problem: the claws were detached, the claw servo was set to 130 degrees, and then the claws were reattached in the same orientation. This results in the claws touching before turning down to 90 degrees; therefore, allowing the claws to be able to fully close.

Figure 4.10 shows the arm's initial positions of chassis, shoulder, elbow and wrist at 90 degrees.

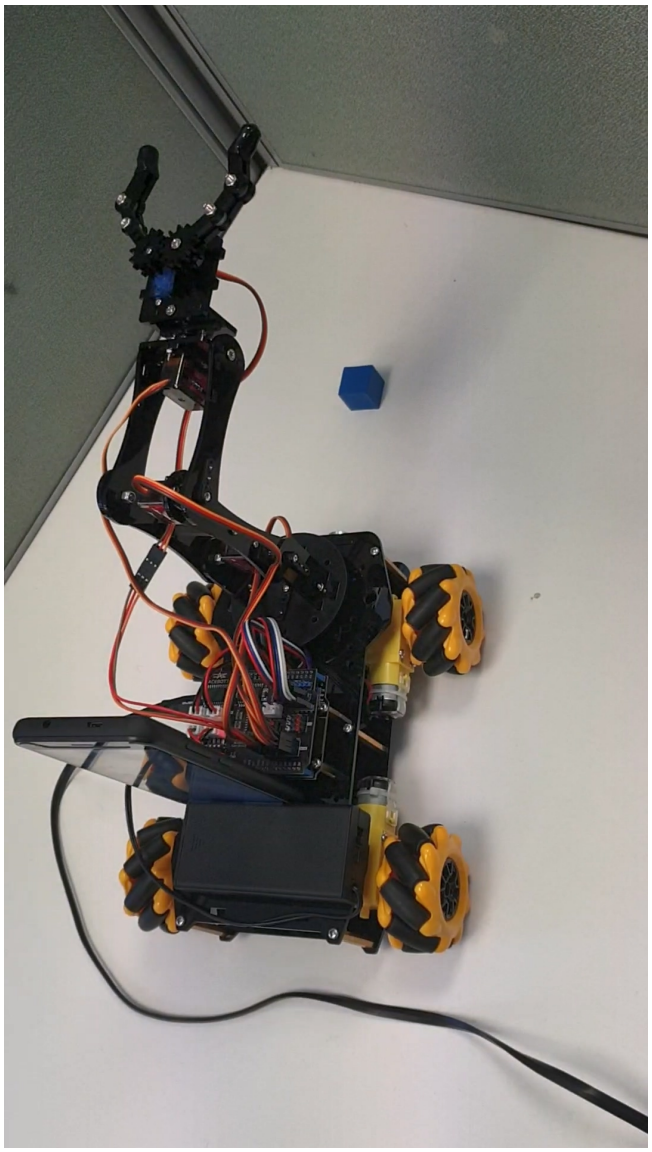


Figure 4.10: Photo of robot with servos of chassis, shoulder, elbow and wrist at 90 degrees

Battery

The robot starts malfunctioning when battery is low. One effect is that the motors stop turning. The robot uses '18650' batteries to power the microcontroller, sensors and actuators [3]. The batteries have a nominal voltage of $3.7V$ and a maximum voltage of $4.2V$. They belong to the African Robotics Unit at University of Cape Town (UCT).

4.6.2 ESP32 Software

As mentioned in the Design Choices subsection, the ESP32-WROOM-32 microcontroller module was used for this project. The ESP32 runs Python 3.4.0 using MicroPython v1.24.1 .

As mentioned eariler, the ESP32 runs `move-to-and-pick-up-block.py` which can be found under the 'Robot code' folder in the project repository [14]. The script is illustrated in algorithm 1.

The program uses an API (provided by ACEBOTT [3][2]) to interface the ESP32 with the sensors and

actuators. The links for the API libraries used are in Appendix C.

Ultrasonic Sensor

To use the ultrasonic sensor, the `ACB_Ultrasonic` class is used (provided by ACEBOTT). The `ACB_Ultrasonic` class provides a method (`get_distance`) for getting the measured distance from the ultrasonic sensor. Getting the measured distance is illustrated below:

```
Ultrasonic = libs.ultrasonic.ACB_Ultrasonic(Trig_PIN = 13, Echo_PIN = 14)
Ultrasonic_distance = Ultrasonic.get_distance()
```

When running script on ESP, make sure battery is on and supplies no less than nominal voltage. If an `EADDRINUSE` error is encountered, perform a hard reset on the ESP, by pressing the button labelled 'RST' which is shown in the top left of figure 4.11 .

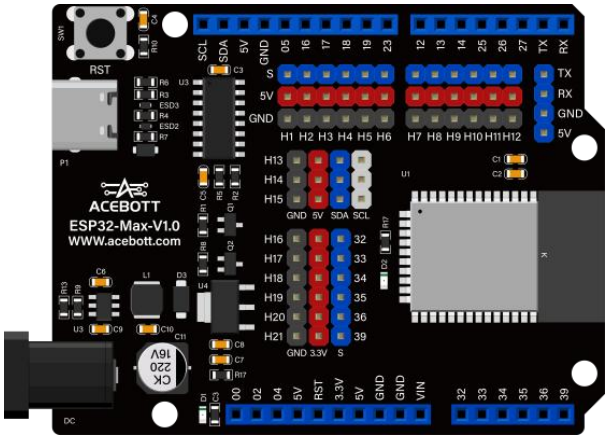


Figure 4.11: Image of ESP32 board [3]

Wheel Motors

ACEBOTT's API [3] provides the `ACB_Vehicle` class to control the wheel motors. Its `move` function takes the given direction and speed, and signals the motors to move appropriately. To move the robot forward the following code is needed:

```
vehicle = libs.vehicle.ACB_Vehicle()
speeds = 130
vehicle.move(vehicle.Forward, speeds)
```

This moves the robot forward indefinitely at the lowest speed that the motors are designed for. `Forward` is a integer constant defined in `ACB_Vehicle`. Other directions used are `Stop`, `Contrarotate` (to turn left), and `Clockwise` (to turn right). The direction used is dependant on the command sent from the phone. Higher speeds are available, but not used because the lowest speed itself is faster than the ideal speed for the operations of this robot.

Arm Servos

The `ACB_Car_Arm` library from [2] was slightly modified and used to control the arm. The code was changed such that the wrist servo doesn't move to an initial angle, and the claw servo moves to 120°instead of 90°. This is because the wrist servo is not used in the operations and the claws were fitted at larger angle than used in the tutorial.

The library provides functions to move each servo to specified angle. For example to move the shoulder to 40°, the following command would code would be used:

```
from libs.ACB_CAR_ARM import *
ARM_init(chassis_pin, shoulder_pin, elbow_pin, wrist_pin, claws_pin)
time.sleep(1)
shoulder_cmd(0)
```

`time.sleep(1)` makes the Python script pause for 1 second. This is put to allow the robot time to execute the movement before the next movement. Otherwise, the movements are inaccurate.

Chapter 5

Experimentation

This chapter details the experiments set-up to test if the robot was able to execute the objective of retrieving an object. Chapter 4 describes the procedure of the robot to fulfil the objective. The procedure was as follows:

1. Align robot in direction of block
2. Move towards the block within reaching distance
3. Pick up block
4. Turn around and come back to initial position

The block used in the experiments is a 2 cm wide cube made of wood. The colour of the block is blue. Subsystems are tested then subsystems are progressively integrated and tested.

The tests start out by testing the operations of the Android phone. It goes on to test the communication between the phone and the ESP32. Then the embedded system is tested, including the ultrasonic sensor and the arm servos. The chapter ends with the full integration test of the robot.

5.1 Android Phone Tests

5.1.1 Timing Camera Frame Period

For this test, the `analyze` function was made empty, in order to time how quickly it would be called again. This would tell us how fast the frames are being captured. If the phone takes long to process the scene, it would be helpful to know how much of the time is being spent on the capture process and how much on the analysis process.

Test procedure: The `analyze` function content is replaced with: getting the current time in milliseconds using Java's `System.currentTimeMillis()`. Java classes can be used in Kotlin. This function returns the current wall clock time of the Android device. It is measured in milliseconds in epoch time [16].

```
override fun analyze(image: ImageProxy) {

        val startAnalyzeTime = System.currentTimeMillis()
        val repDuration = startAnalyzeTime - previousStartTime

        Log.d("Timing", "Rep duration: $repDuration")
```

The repetition duration is printed to Android Studio's Logcat window using the `Log.d` function.

The application is uploaded to the phone using Android Studio. Android Studio Meerkat (2024.3.1) is the version used. The application is uploaded to the phone via USB cable. In order for this to be allowed, developer mode is enabled on the phone. Once it is enabled, the device is selected by opening the menu item shown in figure 5.1 .

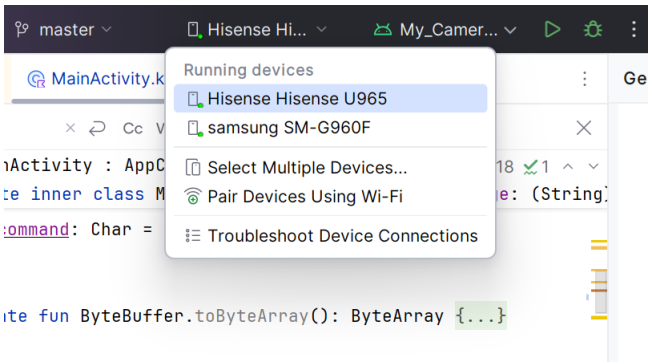


Figure 5.1: Menu item in Android Studio for selecting device to upload application

Then the 'run' button (with the play icon) is clicked and the application is uploaded to the selected device.

Result: The average frame period was 30 ms when the phone was held upright. When the phone lay flat with the camera facing down is took about 90 ms between frames. A possible explanation for this is that imageCapture tries to focus the camera before capturing a frame. When the camera is facing down, it may be adjusting its focus for a longer time.

5.1.2 Block lateral position alignment

The block aligned range threshold was tested by placing the block in different x locations.

(a) block-out-of-alignment

(b) block-in-alignment

Figure 5.2: Comparison of robot view in different x positions

Figure 5.2a shows the camera view of the block when the block is out of alignment. Here the position of the block at $x = -164$ (block's Centre of Mass (COM) is 164 pixels to the left). Figure 5.2b shows the camera view of the block when the block is sufficiently aligned with the robot. Here the position of the block at $x = 28$ (block's COM is 28 pixels to the right).

5.1.3 Range when aligned test

The range threshold was tested by placing the block in different x locations.

5.2 Communication between Phone and ESP

A test application was created to send a user-entered command to the ESP. Figure 5.3 shows a screenshot of the test app. It shows the user inputting `'F'` to tell the ESP32 to move the robot forward.

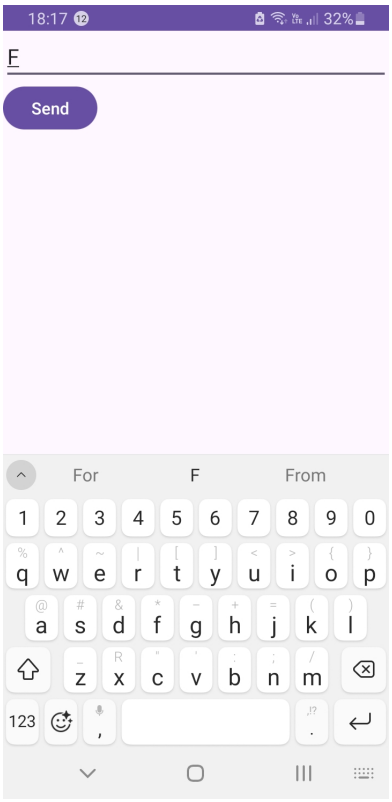


Figure 5.3: Test application for providing interface for entering command to send to ESP

The `7.3Web_control_car.py` (from [3]) script was run on a dummy ESP32-WROOM-32. This script includes the code to receive data from the TCP socket. The script was run in Thonny on a PC. The command string was received and the ESP32 printed out the command in the Thonny shell. Figure 5.4 shows a screenshot of the Thonny application with the shell at the bottom. It shows that the `'F'` string was received and interpreted as `'move forward'`.

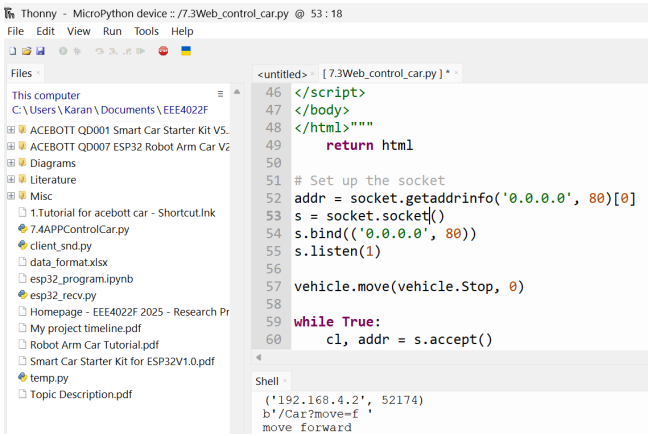


Figure 5.4: Screenshot of Thonny application showing that the command was received

5.3 Embedded System Tests

This section shows the tests done on the embedded system. However, the wheel motors are tested in the 'Find and Approach Object' test, which is in the following section. This is because the wheel motors usage is dependent on commands sent by the phone.

5.3.1 Ultrasonic Sensor Measuring

The following experiments were conducted in an office cubical.

The experiments were done at UCT, Rondebosch, Cape Town, where the average air pressure (in April) was 1017 mbar [17].

Static measuring

The set-up was like this: The 2cm wide block (to be retrieved) is placed directly in front of the ultrasonic sensor at different distances. The set-up is shown in figure 5.5

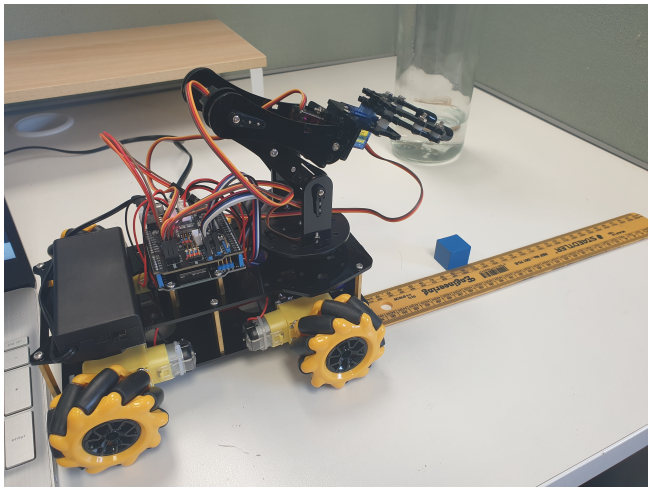


Figure 5.5: Photo of ultrasonic test

Figure 5.6 shows a side view of the experiment for clarity.

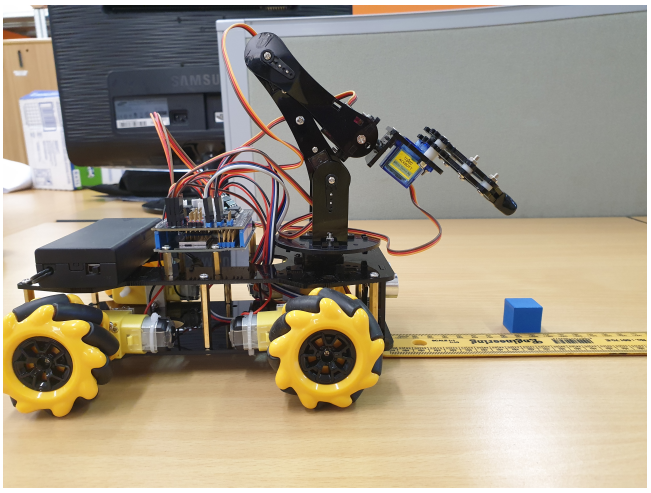


Figure 5.6: Side view of ultrasonic test

The actual distances are measured from the front of ultrasonic sensor, using a ruler, as shown in figure 5.7.

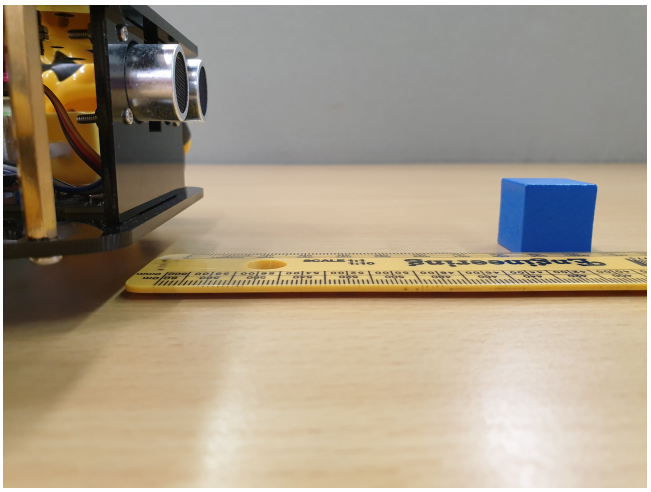


Figure 5.7: Ruler measurement from ultrasonic sensor to block

The ultrasonic sensor is told to take a reading, and it is compared to the physically measured distance. `ultrasonic-test.py` (the version in commit `45f19c0`) is used to take the ultrasonic sensor readings. This is shown in listing 5.1.

Listing 5.1: Code for testing ultrasonic sensor

```
import libs.ultrasonic
import time

Trig_PIN = 13
Echo_PIN = 14

Ultrasonic = libs.ultrasonic.ACB_Ultrasonic(Trig_PIN,Echo_PIN)
```

```
while True:
    UT_distance = Ultrasonic.get_distance()
    print(UT_distance) # cm
```

Thonny was used to upload and run the program on the robot's ESP. It uses the `ACB_CAR_ARM` found in the GitHub repository. It is a slightly modified version of the class provided by ACEBOTT [2].

Table 5.1 tablulates the ultrasonic reading for each distance.

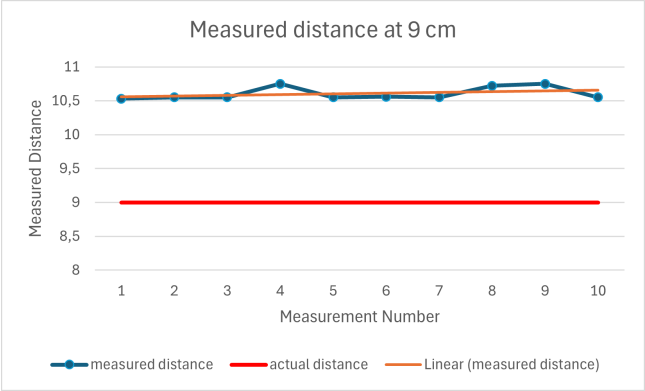
Actual distance (cm)	**Ultrasonic (cm)**
3	does not detect
6	sometimes DND
9	10.606
12	12.891
15	15.671
18	18.493
21	21.816
24	25.629
27	26.868
30	30.01

Table 5.1: Ultrasonic distance measurement for varying distances of block from ultrasonic sensor

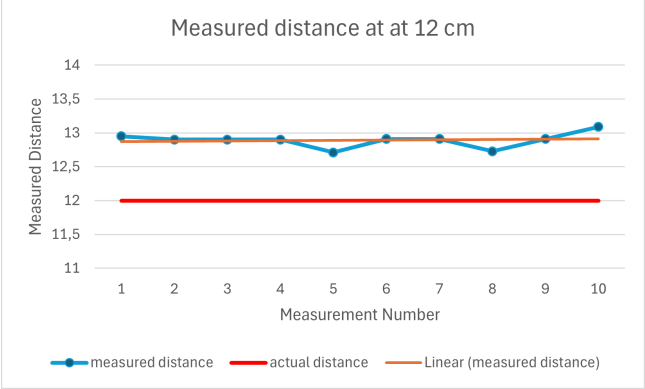
Table 5.1 shows that the shortest distance that the sensor reliably detects the cube is 9 cm, which the sensor measures as 10.606 cm. Therefore, the robot was programmed to not come within 10.606 cm of the block (otherwise it may crash into it).

The furthest distance that the arm could reach is 17cm. Therefore, a range from 9 to 17 cm was considered.

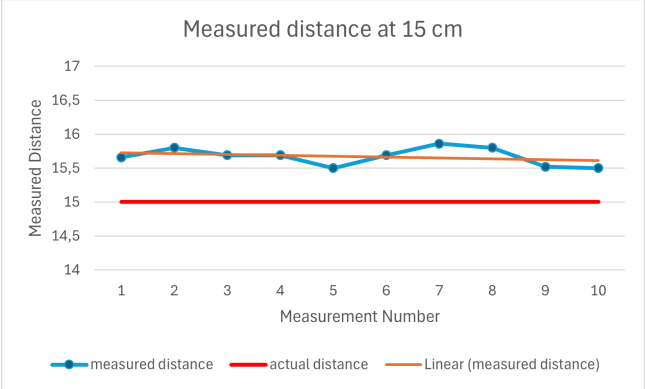
Figure 5.8 shows multiple readings at each distance within the considered range.



(a) Measured distance at 9 cm



(b) Measured distance at 12 cm



(c) Measured distance at 15 cm

Figure 5.8: Readings at each distance: 9cm, 12cm, 15cm

Dynamic measuring

The readings were also taken while the car was moving. Figure 5.9 shows the readings while the car moved at a constant speed (`speeds = 130`). `time = time.ticks_us()` should of been added in listing 5.1, so that time scale could be used in figure 5.9 .

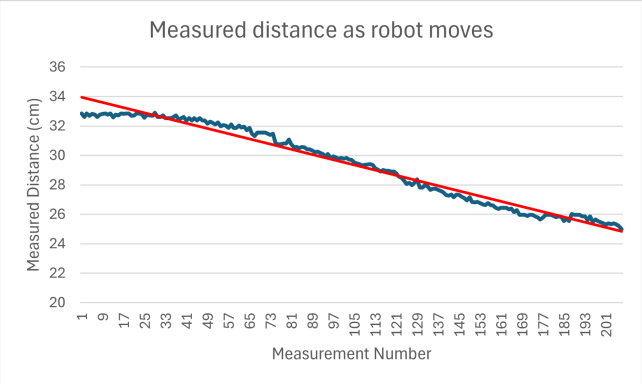


Figure 5.9: Plot of measured distance as robot moves (in blue) compared to line of best fit (in red)

5.3.2 Pick up object Test

Here, the arm servos were tested using the ESP. The objective of the test was to pick up the block.

Once the robot has reached the block, it needs to pick it up. This test tests if the robot can pick up the block.

This experiment runs `pick-up-block.py`. Figure 5.10 shows the each stage of the robot picking up the block.

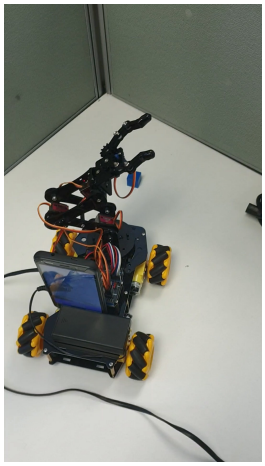
5.4 Integrating the Phone to the Embedded System

The objective of this test was to test if the robot could find, and approach the blue block. This test combines the Android phone, ESP, ultrasonic sensor, and the actuators (motors and servos). The phone had to be placed on the robot.

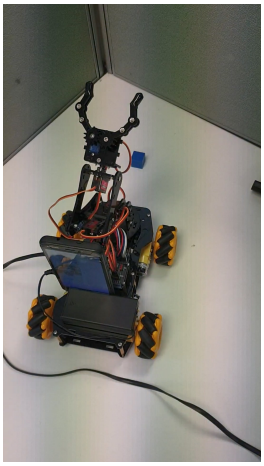
5.4.1 Phone Camera positioning

Camera was tested on the right side of the robot as shown in Figure 5.11. This set-up has a blind spot on the left side of its field of view. Camera was tested on the left side of the robot as shown in Figure 5.12. This set-up has a blind spot on the right side. Camera was tested on the centre of the robot as shown in Figure 5.13. This set-up has a blind spot in the centre. However, in this set-up, the blind spot is only there because of the arm. Therefore, if the arm is moved the blind spot could be removed.

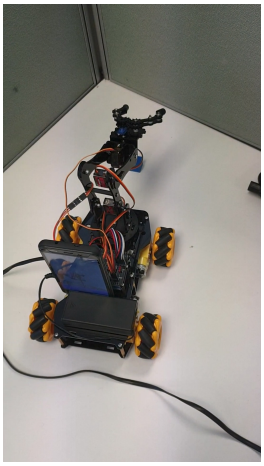
Camera was kept on the centre of the robot with the arm rotated to the side as shown in Figure 5.14. This gives an almost unobstructed field of view.



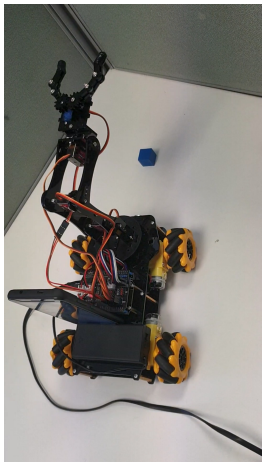
(a) initial position



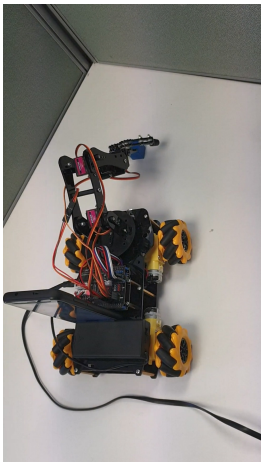
(b) align arm



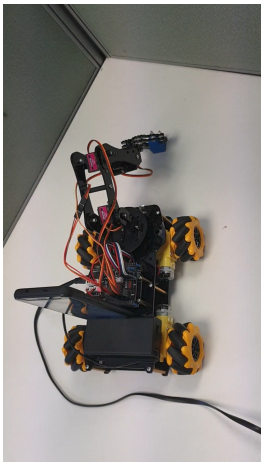
(c) shoulder to 90°



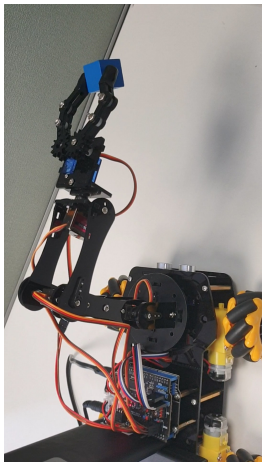
(d) elbow to 90°



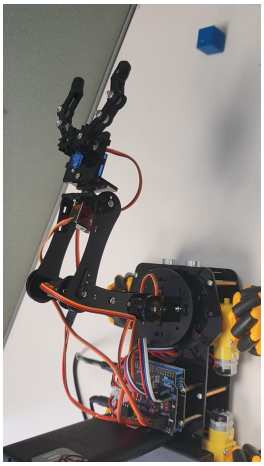
(e) move to block



(f) grab block

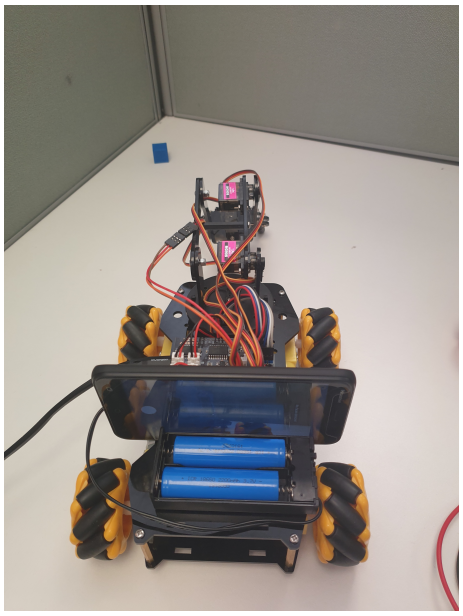


(g) pick up block

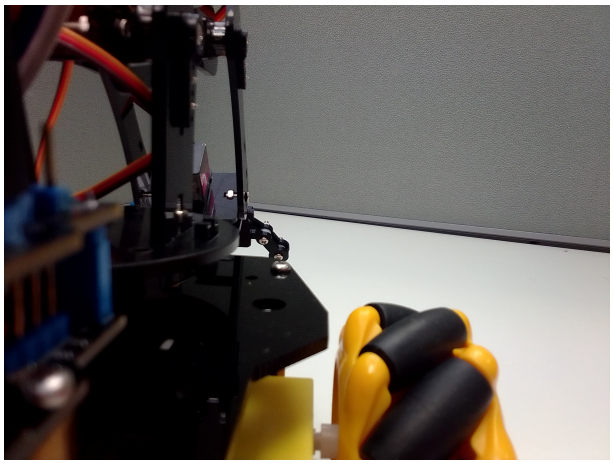


(h) let go of block

Figure 5.10: Series of photos of robot picking up block [4]

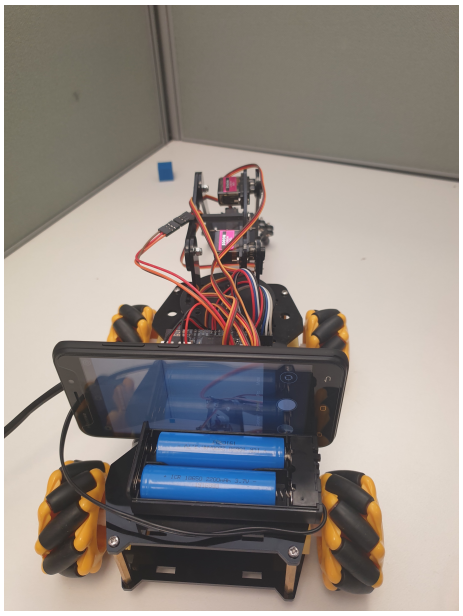


(a) Set-up

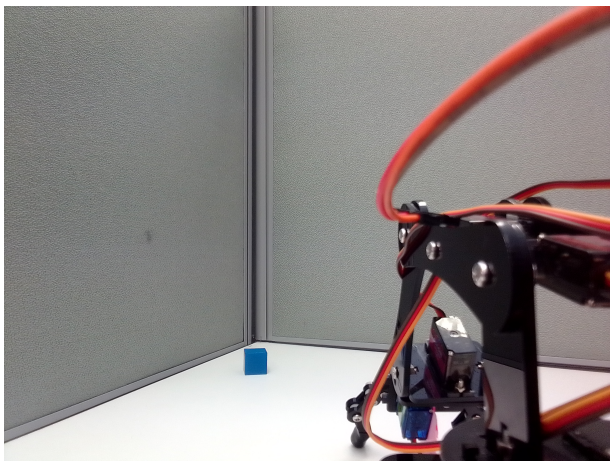


(b) Field of view

Figure 5.11: Camera placed on right side of robot

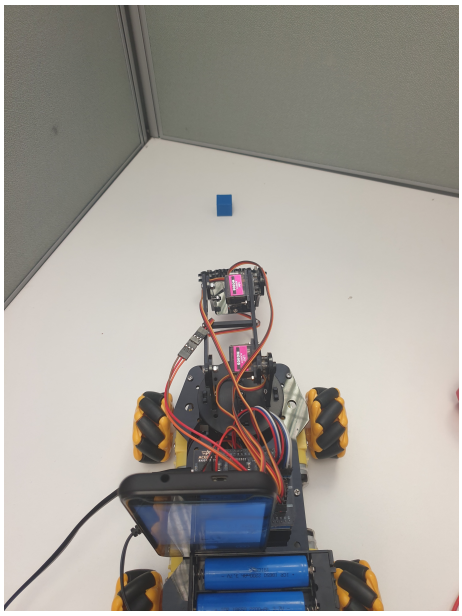


(a) Set-up

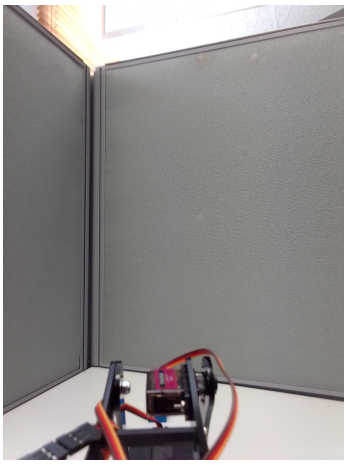


(b) Field of view

Figure 5.12: Camera placed on left side of robot

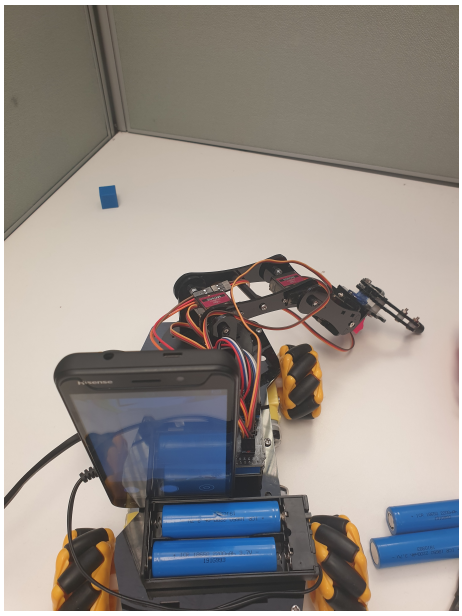


(a) Set-up



(b) Field of view

Figure 5.13: Camera placed on centre of robot



(a) Set-up

(b) Field of view

Figure 5.14: Camera placed on centre of robot with arm to side

Once, a suitable placement was found for the phone (with the camera positioned with a clear field of view), the test could be done.

5.4.2 Find and Approach Object Test

The procedure of the test is shown in the following list:

1. The block was placed within the robot's field of view.
2. The `align-to-robot.py` (shown in commit 4de63c2 of the repository [14]) was uploaded and run on the ESP.
3. The `My Camera Capture App` was opened on the phone.
4. The `Robot_Arm_Car` network was selected.
5. The robot autonomously moved to the block.

Three tests were done with different initial positions. Figure 5.15 shows the initial and final positions of each test.

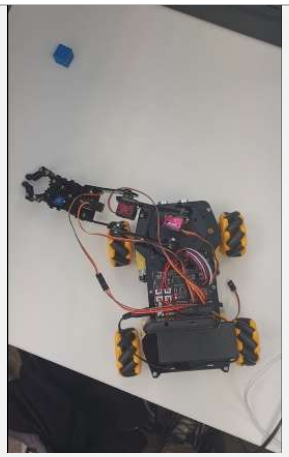
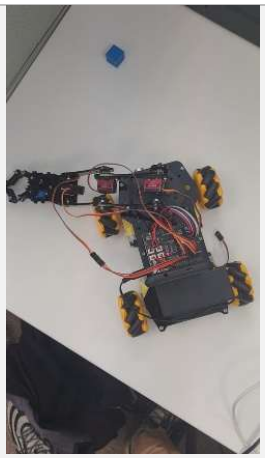
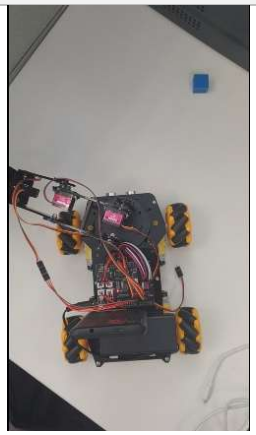
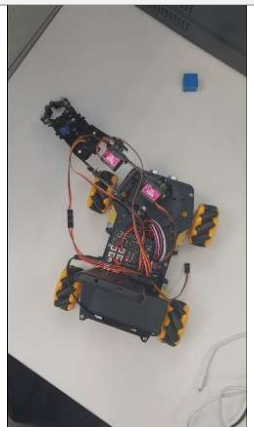
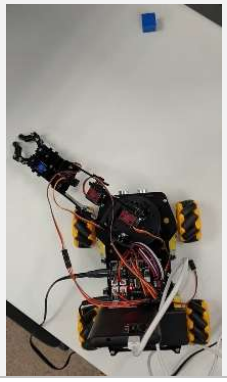
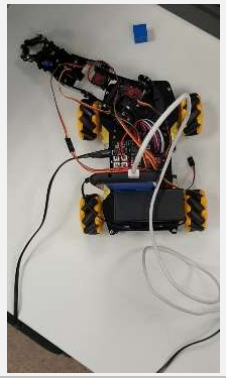
initial position	final position
	
	
	

Figure 5.15: Initial and final positions of each test

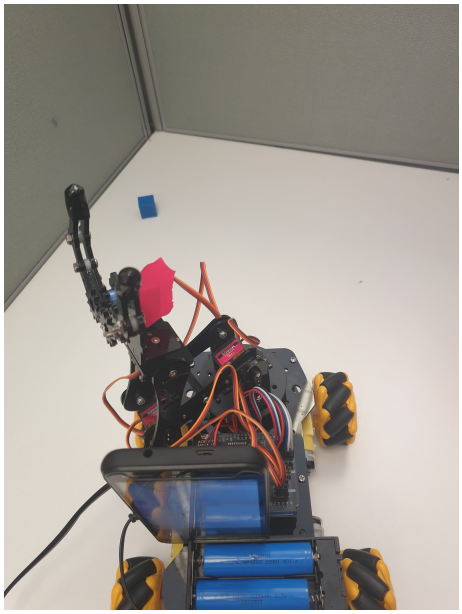
In all tests, the robot drove to the block, with the block directly in front of it and within reach.

However, when testing moving the arm from this position, it was found that the shoulder servo would collapse or stop working. This was investigated and it was concluded that the servo was not powerful enough to undertake the required torque at this position to overcome gravity. The gravitational torque is proportional to radius (in the x-y plane) of the end-effector, and at that position the COM was almost at its maximum radius. Therefore, another position was looked with a shorter radius.

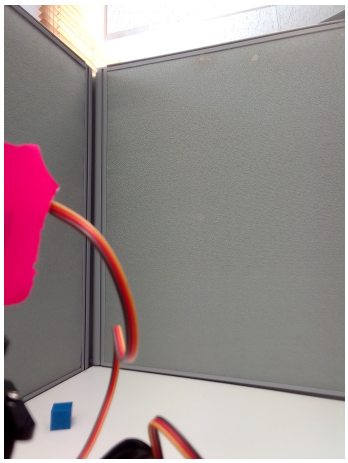
A suitable position was found: the arm to the side and the elbow raised as shown in Figure 5.16. This provided an almost unobstructed field of view and with a short enough radius such that the shoulder servo could continue to function.

The elbow angle was set as large as possible without going unstable. It was discovered that elbow angle above 45°(with the shoulder angle at 0°) causes the arm to go unstable (the elbow servo would continuously oscillate or the shoulder servo would collapse).

Therefore, this set-up shown in figure 5.16 was used for arm initialization.



(a) Set-up



(b) Field of view

Figure 5.16: Camera placed on centre of robot with arm to side and elbow up

5.4.3 Arm initialization

The find and approach object test was repeated with the robot first moving its arm out of the camera's field of view as shown in figure 5.16 . Figure 5.17 shows snapshots of the robot moving its arm out of the way and then finding and approaching the block. A video of the demonstration `Demo videos/align with arm initial.mp4` can be found in the project repository.

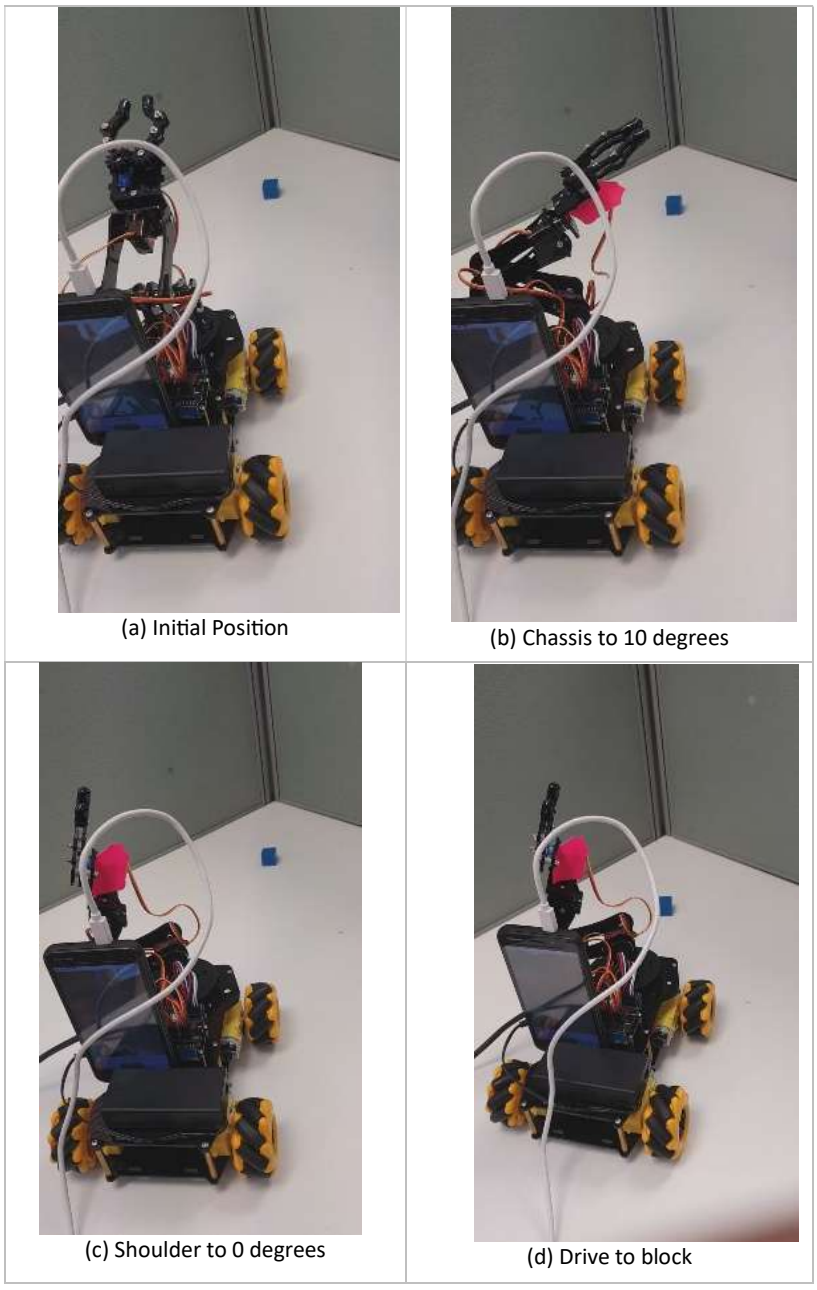


Figure 5.17: Initialize arm, find and approach block test

5.5 Full Integration Test

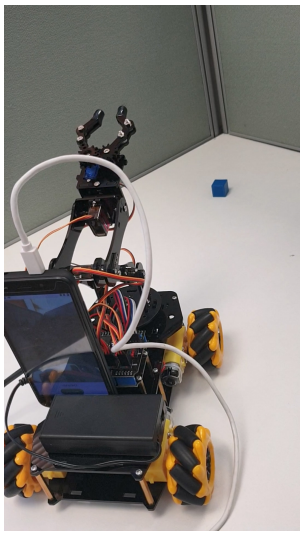
This test tests the full system: the objective is for the robot to find, approach and pick up the block. This experiment runs `Robot code/move-to-and-pick-up-block.py`.

The general procedure is as follows:

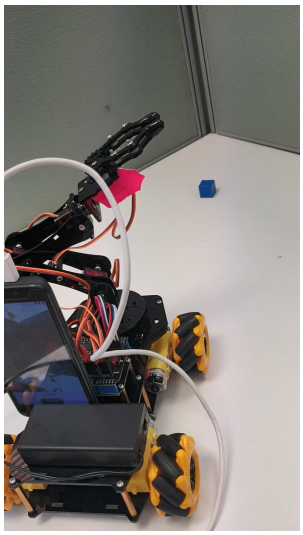
1. Move arm out of way of camera
2. Drive to block
3. Pick up block

Figure 5.18 shows every step of the robot executing the task. Figure 5.18j shows that the arm misses the block. The block was moved by human intervention to the claw before it closed. It managed to pick the block up, but very slightly, as shown in figure 5.18k. The robot was unable to pick up the block completely. This is because the shoulder servo buckled under the load.

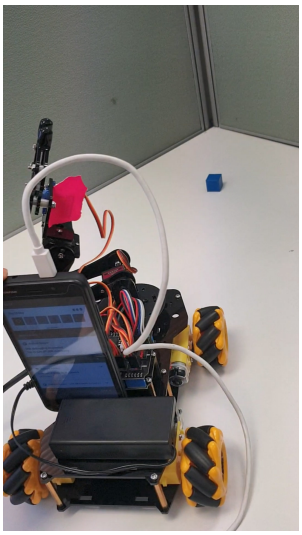
The arm misses the block in the y direction. This is determined by ultrasonic sensor, which has significant measurement noise as shown in 5.8. Another integration test was done, and the arm reached much closer to the block, but not enough to grab it. This is shown in figure 5.19.



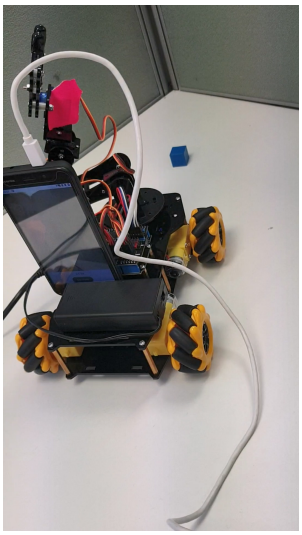
(a) initial position



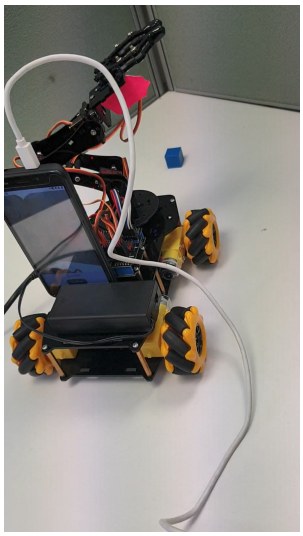
(b) move arm to side



(c) move shoulder to 0°

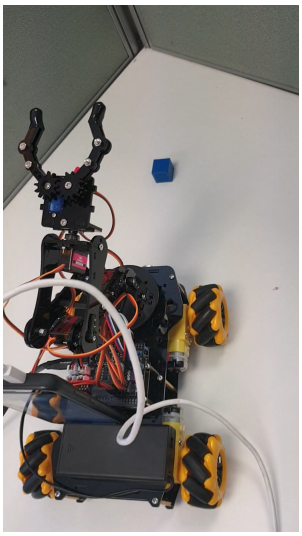


(d) drive to block

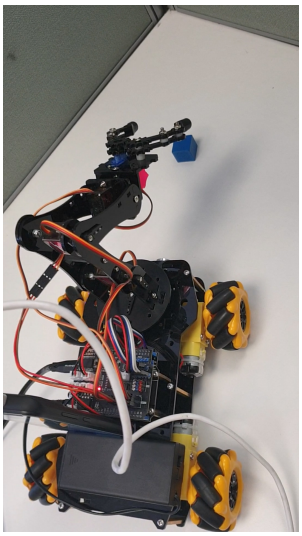


(e) move shoulder to 40°

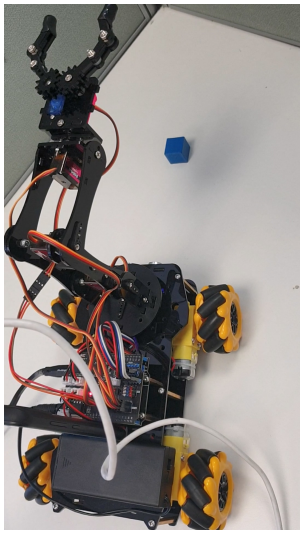
(f) open claw



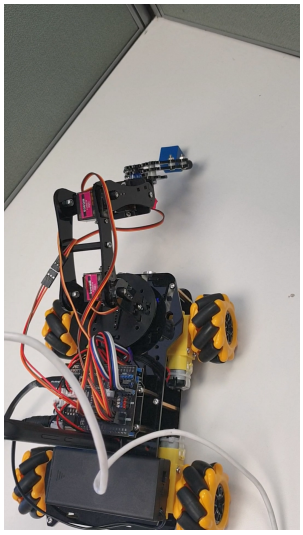
(g) turn arm to centre



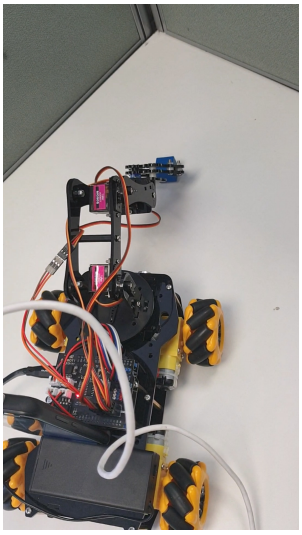
(h) move shoulder to 90°



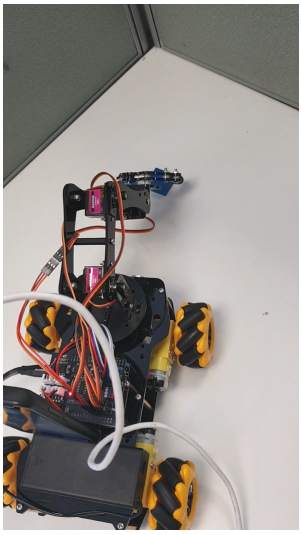
(i) move elbow to 90°



(j) move arm to block

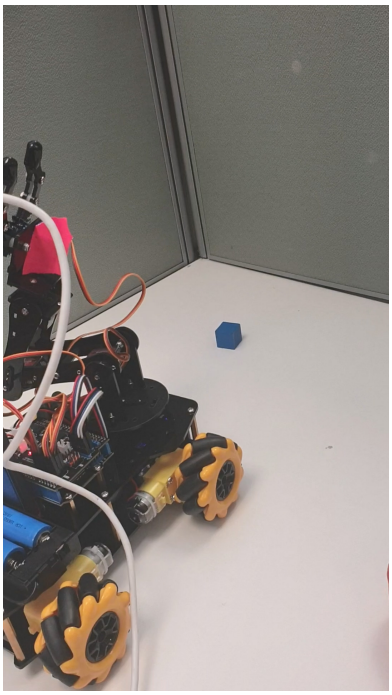


(k) pick up block

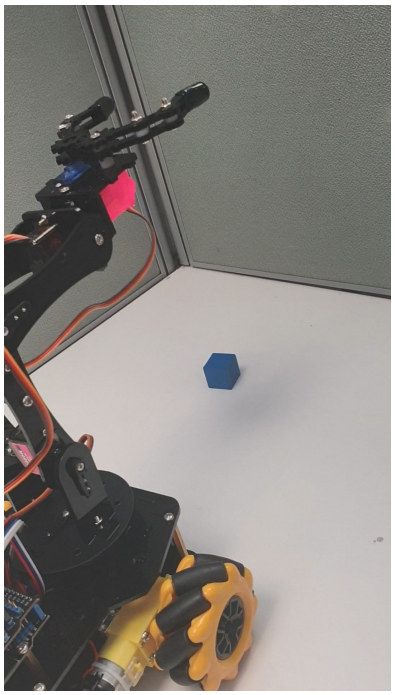


(l) release block

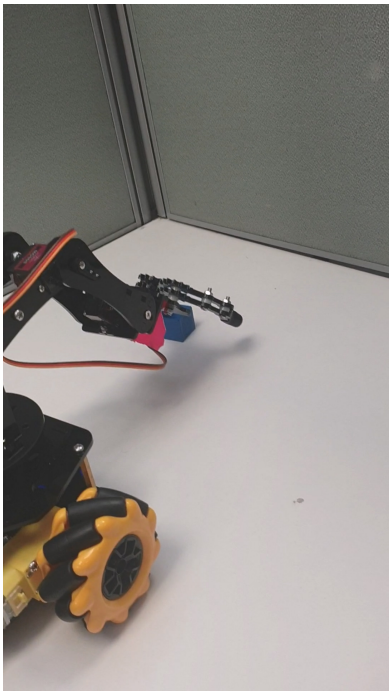
Figure 5.18: Full integrated test with slight pick-up [4]



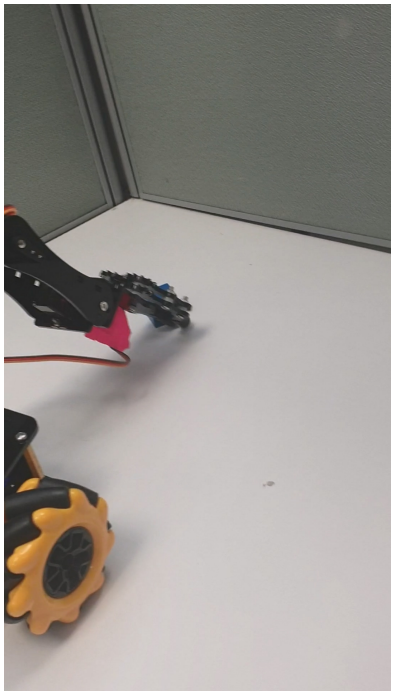
(a) robot within reaching distance of block



(b) arm moved to 90°position with claw open



(c) arm moved towards block



(d) claws closing

Figure 5.19: Series of photos showing robot trying to pick up block again

Chapter 6

Conclusions

Old smartphones are being discarded despite being capable for embedded systems, specifically robotics applications. This project has demonstrated the feasibility of repurposing phones as dedicated device in a robotics application. A low-end Android phone's camera and SoC has been shown to be adequate for vision and control of the robot in tandem with an ESP32 microcontroller.

This report describes how this system was implemented and the code is published on a public GitHub repository [14]. Therefore, the project can be reproduced and its functionality can be extended.

The phone code can be implemented on many Android devices. Therefore, with slight modifications most phones can be interfaced with various robotics platforms. This creates a new resource in the form of used or even damged phones.

Chapter 5 tested the procedure of the robot to fulfil the objective. The procedure was broken down into the following stages:

1. Align robot in direction of block
2. Move towards the block within reaching distance
3. Pick up block
4. Turn around and come back to initial position

Stages 1 and 2 were implemented successfully. Stage 3 failed as the robot could not move its arm to the block accurately, due to the measurement noise of the ultrasonic sensor. However, it was able to pick the block up if the block was placed between its claws. Stage 4 was not implemented as it was out of the scope of this project. However, this was enough to show the effectiveness of utilizing smartphones (even low-end ones like the one used in this project) for embedded projects.

6.1 Recommendations

For this type of project, it is recommended that an accurate ultrasonic sensor be used for accurate distance measurements. Additionally, it is advised to use good quality servos that can hold its own weight at outstretched positions and is built to be used for long periods of time.

6.2 Future Work

Stage 3 would be completed by getting better measurements for the y position of the block. Stage 4 of the robots procedure 'Turn around and come back to initial position'. The image processing could be improved by using artificial intelligence models. This would make the robot useful for real world applications such a tidying up. The system would be made more user friendly by implementing a user interface on a PC. The PC would connect to the phone perhaps over the existing Wi-Fi network hosted on the ESP.

Bibliography

[1] A. De Luca, G. Oriolo, L. Paone, and P. Giordano, "Experiments in visual feedback control of a wheeled mobile robot," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, 2002, pp. 2073–2078 vol.2.

[2] ACEBOTT, "Acebott qd007 esp32 robot arm car v2.3 tutorial," ACEBOTT, Tech. Rep., 2025. [Online]. Available: https://acebott.com/tutorial/?srsltid=AfmBOorYL_Oy0HhSmnT_fSkpibs5hneBzH6BVEkCA_2gYtp-fJx01WLR

[3] ——, "Acebott qd001 esp32 smart car kit tutorial," ACEBOTT, Tech. Rep., 2025. [Online]. Available: https://acebott.com/tutorial/?srsltid=AfmBOorYL_Oy0HhSmnT_fSkpibs5hneBzH6BVEkCA_2gYtp-fJx01WLR

[4] Mico, "How to arrange 8 figures, with subcaptions, in a 3-row form," 2016. [Online]. Available: https://tex.stackexchange.com/questions/317764/how-to-arrange-8-figures-with-subcaptions-in-a-3-row-form#:~:text=In%20the%20following%20code%2C%20the,%2Dlast%20%5Chspace*%20directive.

[5] I. Ilankoon, Y. Ghorbani, M. N. Chong, G. Herath, T. Moyo, and J. Petersen, "E-waste in the international context – a review of trade flows, regulations, hazards, waste management strategies and technologies for value recovery," *Waste Management*, vol. 82, pp. 258–275, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0956053X18306366

[6] P. J. Narayana Raju, V. Gaurav Pampana, V. Pandalaneni, G. Gugapriya, and C. Baskar, "Design and implementation of a rescue and surveillance robot using cross-platform application," in *2022 International Conference on Inventive Computation Technologies (ICICT)*, July 2022, pp. 644–648.

[7] J. Nádvorník and P. Smutný, "Remote control robot using android mobile device," in *Proceedings of the 2014 15th International Carpathian Control Conference (ICCC)*, May 2014, pp. 373–378. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.uct.ac.za/document/6843630

[8] F. I. A. Sunarko, E. Sunarno, D. S. Yanarati, and E. Kusumawati, "Smart dc home for energy saving with android-based real-time energy monitoring," in *2022 International Electronics Symposium (IES)*, 2022, pp. 156–161.

[9] S. S. Pujari, M. S. Patil, and S. S. Ingleshwar, "Remotely controlled autonomous robot using android application," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Feb 2017, pp. 588–593.

[10] S. Dey and T. Bera, "Design and development of a smart and multipurpose iot embedded system device using esp32 microcontroller," in *2023 International Conference on Electrical, Electronics, Communication and Computers (ELEXCOM)*, Aug 2023, pp. 1–6.

[11] G. Franchini, S. Chiodini, M. Ghetti, and M. Pertile, "Mechatronic design and positioning accuracy characterisation of a robotic arm for exploration rovers," in *2023 IEEE 10th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, 2023, pp. 452–457.

[12] BrowserStack, "Flutter vs android studio: Core differences," 2025. [Online]. Available: https://www.browserstack.com/guide/flutter-vs-android-studio#:~:text=Performance:%20Native%20apps%20built%20with,apps%20are%20generally%20more%20lightweight.

[13] Android Developers, "Getting started with camerax," Google, Tech. Rep., 2024. [Online]. Available: https://developer.android.com/codelabs/camerax-getting-started#0

[14] K. Abraham, "Milo: an object retreiver robot," 2025. [Online]. Available: https://github.com/karanimaan/Object-Retreiver-Robot-Code

[15] Android Developers, "androidx.camera.core."

[16] ——, "System clock," Google, Online tutorial, 2025.

[17] CustomWeather. (2025, April) Annual weather averages near rondebosch. [Online]. Available: https://www.timeanddate.com/weather/@7302802/climate#:~:text=Averages%20in%20Rondebosch-,High%20Temp:%2080%20%C2%B0F,Visibility:%2010%20mi

Appendix A

Graduate Attributes

This chapter states what Graduate Attributes (GA)s have been met and where they were met in the project.

GA 1: Problem Solving

I have interfaced a spare android phone with a robot (that I assembled) which can move to an object. I managed to use the phone's camera to capture a live stream of photo frames, to be analysed to determine the position of a certain object relative to the robot.

Since object detection would take too long to implement in this project, I implemented a rudimentary way of detecting the object. I am using a blue coloured block in a non-blue background, and implemented an algorithm which calculates the lateral position of the block by classifying each pixel (of the latest frame) as blue or not. This allowed the robot to know which way to turn in order to align to the block.

Within the same program I created a TCP client socket which could send the needed command. The ESP32 microcontroller (hereafter referred to as "ESP") controls the robot and I setup a server on the ESP which listens for the command sent by the phone.

Although the phone could send commands quickly to the ESP and then to the motors, the minimum motor speed was too fast to stop in time when the robot aligned to the block. Therefore, the robot was made to stop before aligning to the block, and to alternately stop and go when within a certain range of the desired angle range, which makes it move slower in that range so not to overshoot.

GA 4: Investigations, Experiments and Data Analysis

I am conducting experiments on the robot: testing the ultrasonic sensor ranging, moving car to setpoint, and arm end-effector setpoint test. The shortest distance that reliably detects the cube is 9 cm, which the ultrasonic sensor measures as 10.606 cm. Therefore, the robot was programmed to not come within 10.606 cm of the block (otherwise it may crash into it).

Timing experiments were also conducted for: photo capture, and sending data from phone to ESP. The average frame period was 30 ms when the phone was held upright. Timing sending data is yet to be tested; however, I do not think it is an essential test.

I am also conducting experiments of the "navigating to block" by placing the block at different positions relative to the robot, and after letting the program run, seeing how well aligned it is (getting final

lateral position), and how far away it is (from the block) .

If the servos of the arm work, I will do arm end-effector setpoint test, and, finally, a whole system test, where the robot will perform "navigating to block" (from different positions) and retrieve block to user.

GA 5: Use of Engineering Tools

I am using Draw.io to draw diagrams for the hardware of the system, as well as the software logic running on the phone and ESP. I am using Android Studio to program the phone and Thonny to program the ESP. I am using Overleaf and Latex to write the report.

GA 6: Professional and Technical Communication

I wrote a professional technical report.

GA 8: Individual Working

I have done all the project on my own.

GA 9: Independent Learning Ability

I looked for and followed tutorials on the internet showing: how to use a phone's camera to capture a live stream of frames how to analyze them; and how to send data from a phone to another device with Wi-Fi. I followed the tutorials provided by Acebott to assemble the robot.

Appendix B

AI Usage

Overleaf's Writefull equation generator was used to generate equation 4.1 .

ChatGPT was used as well. It was used to generate the Latex code structure of figure 5.19 using the following prompt: `create latex for 4 subfigures of phone photos`

Appendix C

Source Code Links

The source code for the Android application and ESP programs can be found on this GitHub repository: https://github.com/karanimaan/Object-Retreiver-Robot-Code [14].

The links to the code used in the final design are shown below:

Android classes: https://github.com/karanimaan/Object-Retreiver-Robot-Code/tree/master/My-Camera-Capture-App/app/src/main/java/com/example/myapplication

Gradle file: https://github.com/karanimaan/Object-Retreiver-Robot-Code/blob/master/My-Camera-Capture-App/app/build.gradle.kts

Python API libraries: https://github.com/karanimaan/Object-Retreiver-Robot-Code/tree/master/Robot%20code/libs

move-to-and-pick-up-block.py : https://github.com/karanimaan/Object-Retreiver-Robot-Code/blob/master/Robot%20code/move-to-and-pick-up-block.py