University of Nottingham
UK | CHINA | MALAYSIA

EEEE1002: Applied Electrical and Electronic Engineering: Construction Project

Individual Lab Report 3

Covering Weeks 6 and 7 of the Software Route

Abstract

Over the course of two weeks, a wireless control system using a web accessed controller and image recognition was implemented to the EEEBot. This was done using the addition of a raspberry pi to the current EEEBot system.

In the first week, the wireless control system was implemented using a web accessed control dashboard that the user can manipulate to input commands. These commands were then sent to the EEEBot using the MQTT protocol and executed using the ESP32. However, due to the motor needing a certain voltage level to start moving, as well as other issues, the EEEBot could not be controlled with the precision intended

In the second week, the image recognition was implemented using a camera attached to the raspberry pi. The camera feed is then fed into a program run on the raspberry pi which then would determine if there is a sign in the frame seen as well as what it meant. After the sign was determined – left, right or forward – a command is sent through I^2C to the ESP32. The ESP32 would the command the EEEBot to move accordingly. However, due to an unstable MPU, it became difficult to implement features for smoother control.

Contents

Introduction

This lab report presents the work undertaken during lab weeks 6 and 7 for the software route. The main theme that covers both weeks of work is the use of the Raspberry pi and its ability to communicate with the ESP32 over a wireless connection. This functionality is used to both control the EEEBot manually and with a control system.

Wireless communication has played a vital role in our civilization today and has facilitated our modern way of life. Wireless communication is responsible for fast and flexible data transfer and has helped innovate every field possible, becoming vital to the infrastructure of the internet and modern forms of communication.

In week 6, the challenge tackled was to implement a method to control the EEEBot using the raspberry pi's Node-Red user interface. The outline is for the user to be able to control the EEEBot using inputs on the Node-Red user interface dashboard which acts like a controller. The commands from the dashboard are sent to the ESP32 over a WIFI connection using the message queuing telemetry transport (MQTT) protocol. The ESP32 will then interpret the message and move accordingly.

In week 7, the challenge is to implement a control system that would allow the EEEBot to recognize a sign – an image of an arrow either going straight, left or right – and move accordingly. Similar to before, the command is from the raspberry pi, however, I^2C communication is used instead of wireless communication.

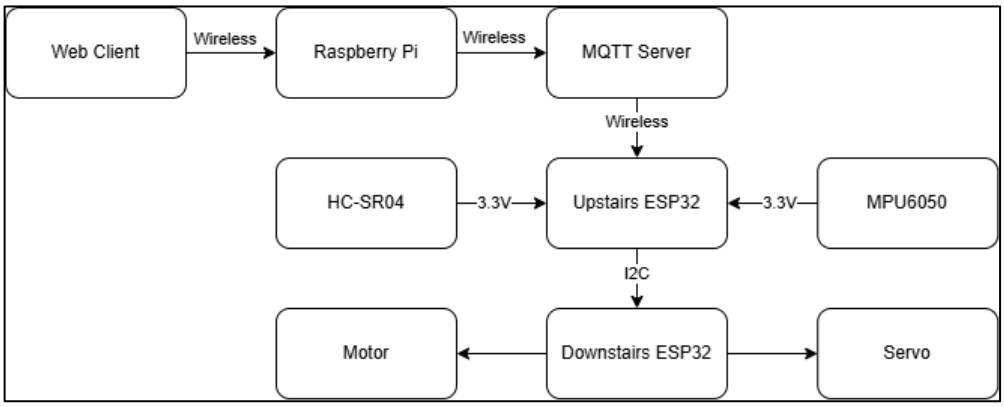

Figure 2 - Week 6 System Overview Diagram

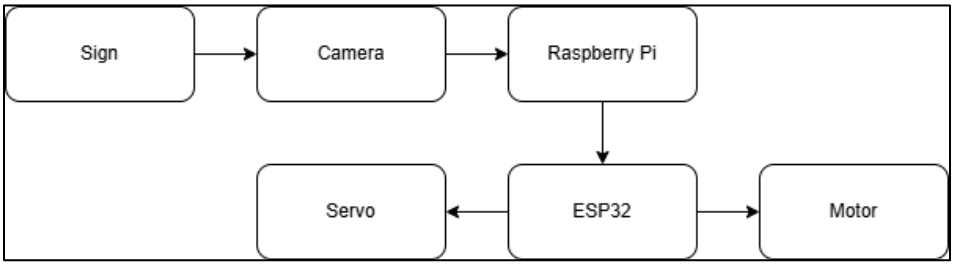

Figure 1 - Week 7 System Overview Diagram

Wireless Vehicle Control and Sensory Feedback

The task for project week 6 is to use the raspberry pi's Node-Red dashboard to allow the user to input commands and control the EEEBot over a WIFI connection. The dashboard acts as a digital controller which then sends commands to the EEEBots ESP32 using the MQTT protocol. The dashboard will also display data from the sensors on the EEEBot which includes the HC-SR04 and the MPU6050 which measures distance [1] and rotation [2] respectively.

Communication methods

There are two forms of communication methods used between the devices which are I^2C and MQTT. I^2C communication is used between the devices and sensors on the EEEBot while MQTT is used between the raspberry pi and the upstairs ESP32.

Message Queuing Telemetry Transport (MQTT)

Message queuing telemetry transport or MQTT is a protocol used for wireless communication between 2 or more devices. The protocol uses a publish-subscribe messaging scheme and is designed for low-bandwidth, high latency networks [3] [4]. The protocol uses a broker which acts as a middleman where all devices on the network are connected to and all communication passes through. To send a message, the message is published to a broker under a label called a topic. This message can then be seen and received by all other devices connected to the broker that are subscribed to the topic.

For the purposes of the project week, both devices will be both a publisher and subscriber as both devices will be sending and receiving information. The raspberry pi will be sending data regarding the speed of the motors and the angle of the servo to the ESP32 with any other output information. The ESP32 will send data received from the MPU6050 and HC-SR04 to the raspberry pi to be displayed.

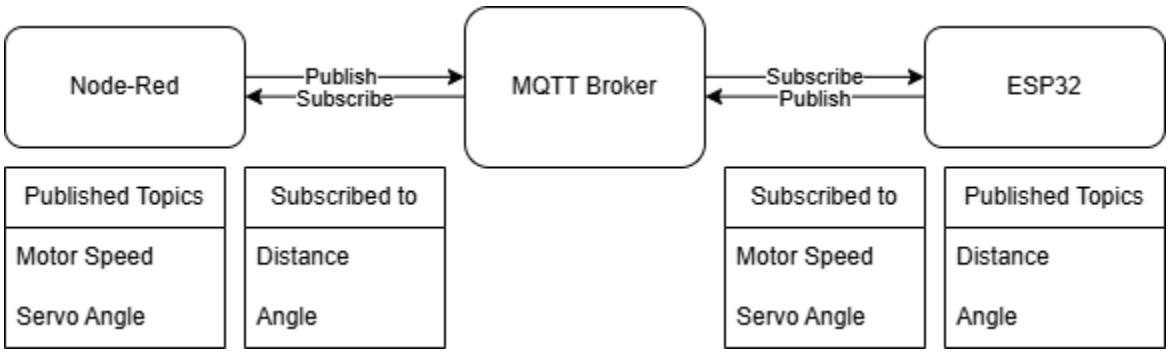

Figure 3 - MQTT Block Diagram

I²C communication

I²C is a method of wired digital communication used between 2 or more devices. The protocol uses a master-slave scheme where the master device – the ESP32 – can write to and request data from slave devices. The protocol uses two bi-directional lines, which are the SCL (serial clock) and the SDA (serial data) lines which are used to both communicate and control communication [5] [6]. To communicate between a master and slave, the master sends a signal with the slave address to notify which device it intends to communicate with. The slave then sends an acknowledgement signal and communication begins.

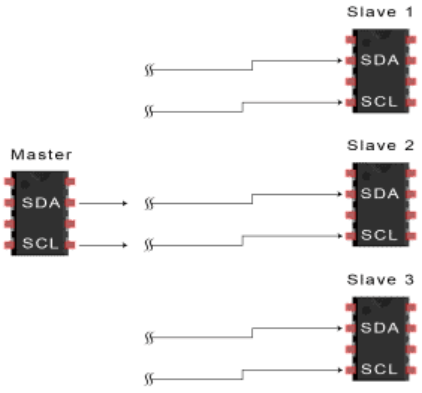

Figure 4 - I²C Diagram [5]

For this project, the master device is the upstairs ESP32 and the rest of the devices – MPU6050, HC-SR04 and the downstairs ESP32 – being the slave devices. The MPU6050 and HC-SR04 will send the data recorded on their respective sensors to the upstairs ESP32 to be sent to the raspberry pi while the downstairs ESP32 will only receive data from the upstairs ESP32.

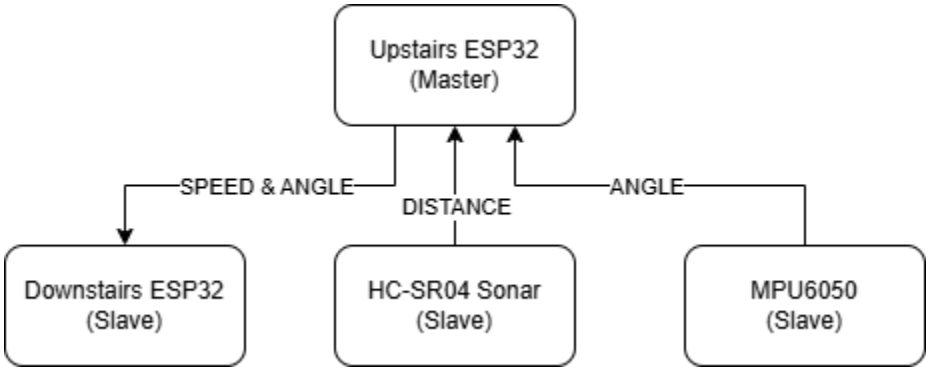

Figure 5 - I²C Block Diagram

Breadboard Testing

First, it is important to have a practical understanding of how communication works. To do this, a simple test is used which is to set up a button on the Node-Red dashboard which turns on an LED through the ESP32 connected to a breadboard. The setup is shown in Figure 6.

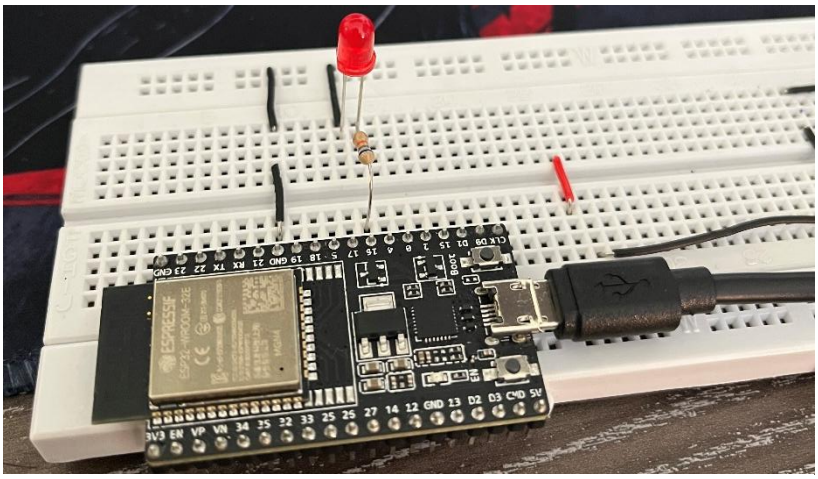

Figure 6 - Communication Testing on breadboard

```
void callback(char *topic, byte *message, unsigned int length)
{
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++)
    {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();

    if (String(topic) == "esp32/output")
    {
        Serial.print("Changing output to ");
        if (messageTemp == "on")
        {
            Serial.println("on");
            digitalWrite(ledPin, HIGH);
        }
        else if (messageTemp == "off")
        {
            Serial.println("off");
            digitalWrite(ledPin, LOW);
        }
    }
}
```

Figure 7 - Communication Breadboard Testing Code

The LED can then be controlled from the Node-Red dashboard using a button. The setup can be done in Node-Red using the example code provided in the instructions booklet provided as shown below in **Error! Reference source not found.**.

The ESP32 is connected to an LED using GPIO pin 15 [7] and is powered using a micro-USB cable. The code used to display the LED is shown below in Figure 7 where if the message received on the topic of “esp32/output” is “on” the light will turn on and if the message is “off” the light will turn off. Note that the “esp32” is the main topic while the “output” is the subtopic [8]. The libraries used are mainly used for MQTT communication as well as to connect to the internet, being the Wi-Fi, PubSubClient [9] and Wire libraries.

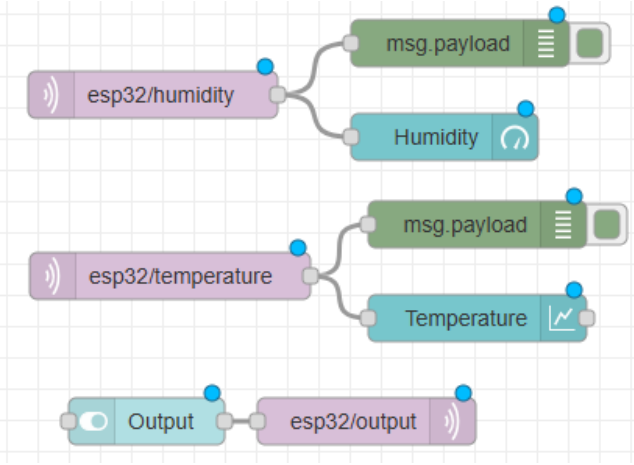

Figure 8 - Node-Red Example Code

Displaying Sensor Data

The next task for the project week is to display the data obtained from the sensors onto the dashboard of the raspberry pi. This requires the EEEBot to not only receive data but to send it as well. The first step is to get the data from the sensors to the ESP32. This uses the aforementioned I^2C communication which connects the MPU6050 and HC-SR04 to the ESP32 with the ESP being the master device. The ESP will process the data accordingly by converting the time to distance [1] as well as keeping track of the angle from the change in angle. Then, using the MQTT protocol, the data is published as a string under the appropriate topic. The raspberry pi retrieves the data it is subscribed to and displays it on the dashboard.

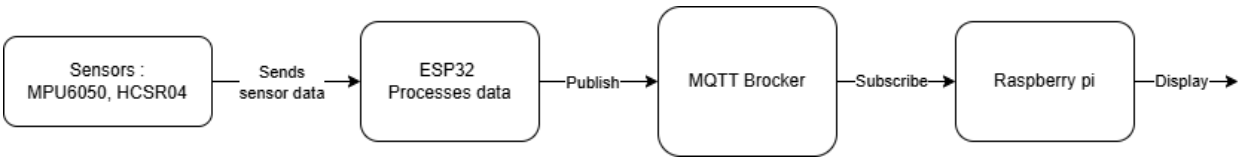

Figure 9 - Display Sensors Data-Flow Diagram

The dashboard is set up to display each data point using a gauge type display. The angle displayed is in the range of -180° and 180° while the distance is in the range of 0 to 100 cm. The display is updated in real time and uses the flow shown in Figure 11.

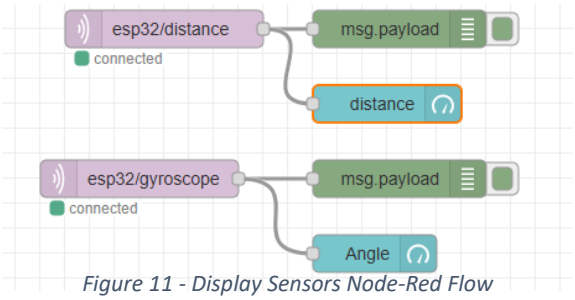

Figure 11 - Display Sensors Node-Red Flow

As for the code on the ESP, the flow of operations can be seen in the flowchart in Figure 10.

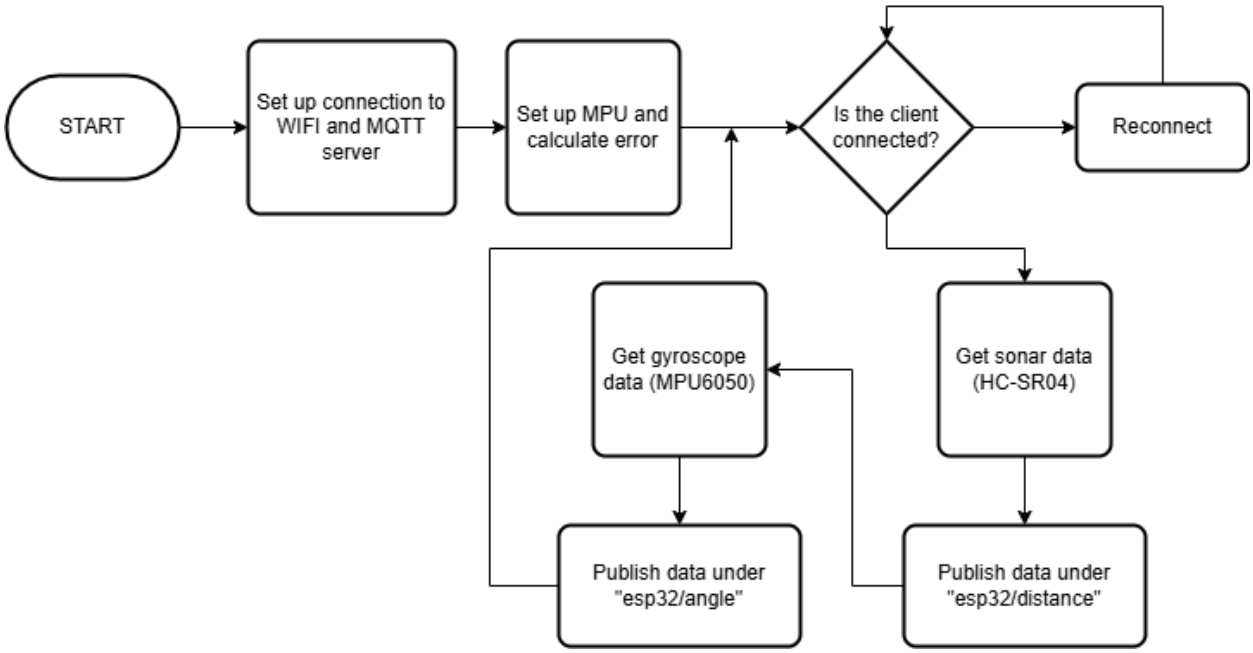

Figure 10 - Display Sensors Code Flowchart

During set up, the ESP will connect to the WIFI first using a provided SSID and network password. Then, it will connect to the MQTT server which is the IP address of the raspberry pi. Then, it will set up any connections to the sensors before going into the loop section. In the main loop, it will first verify that it is connected and then begin to get the data from each sensor. The raw data from each sensor is processed into a desired form and then published one at a time. A part of the main loop can be seen in Figure 12 while the full code can be seen in the appendix.

```
// get sonar data

digitalWrite(sonar_trig, LOW);
delayMicroseconds(2);
digitalWrite(sonar_trig, HIGH);
delayMicroseconds(10);
digitalWrite(sonar_trig, LOW);

// Convert and publish distance

duration = pulseIn(sonar_echo, HIGH);
distance = (duration * .0343) / 2;
dtostrf(distance, 1, 2, distanceString); // convert numbers to a string
// Display distance for verification
// Serial.print("distance X: ");
// Serial.println(distanceString);
client.publish("esp32/distance", distanceString); // publishing the data
```

Figure 12 - Display Sensors ESP32 Code Snippet

The output from the serial monitor is shown in Figure 13 and is then displayed on the dashboard in Figure 14.

```
distance X: 30.32
distance X: 31.38
distance X: 31.34
distance X: 31.15
distance X: 31.45
distance X: 30.74
distance X: 31.23
distance X: 31.54
distance X: 31.44
distance X: 29.73
```

Figure 13 - Display Sensors Serial

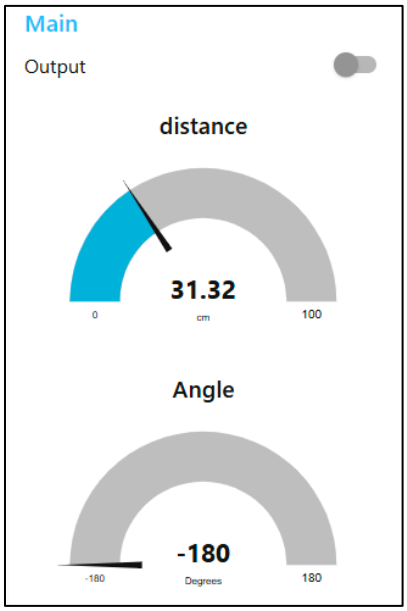

Figure 14 - Display Sensors Dashboard Display

Vehicle Control

To control the EEEBot, a sensible method is to emulate a controller as to have intuitive control for the users. However, there is a problem with this as there is no built-in analog stick input or such provided by the Node-Red dashboard. To circumvent this, the solution employed is to use two separate sliders, one to control the motor speed and one to control the angle. The combination of these two sliders results in a similar effect that is effective in controlling the EEEBot.

The first implementation was to have the speed control slider be between -127 and +127 speed while the servo angle slider is between ±90°. The 127 is used as it gave the largest number an 8-bit signed integer can represent; while the 90° was used as it the EEEBot would only need to turn 90° in either direction – left or right. The wide range of values was designed to give the most precise control of the vehicle as possible. This, however, was not possible as once the information reached the top ESP it was not able to be sent to the bottom ESP where the commands were to be executed in one go due to being over 8-bits once combined – going over the byte size and adding an extra step in sending the message. The first proposed solution was to send the information in two rounds, the first always being the motor speed while the second was to be the servo angle. This would require that at the end – the downstairs ESP – the message would need to be arranged by order which would be difficult as when receiving a signal, the ESP will call a function which can then be used to process the information. The problem lies with the fact that this function is called every time a signal is received – once for the motor speed and once for the servo angle. This means another flag must be used to keep track of incoming messages which cannot rely on the loop this function completely ignored the main loop and takes priority. This can be done but a more simple solution was used instead.

```
// i2C com set up
Wire.begin(downstairsAdd);
Wire.onReceive(adjust_param);
Serial.begin(115200);
```

Figure 15 - Vehicle Control Code Snippet 1

The second solution implemented was to reduce the range of the two sliders so that they are both ±7 instead. This means that they can both be represented as a signed 4-bit integer so that one can be binary shifted and added together, resulting in one 8-bit message that can be sent across using I^2C. To do this, the raspberry pi sends the values in the range, then the top ESP adds 8 to the value to make it a 4 bit unsigned integer, if the topic of the message is "esp32/servo_angle" the result is binary shifted left by 4 bits. Then, the two values are added together and transmitted to the bottom ESP.

```
} else if (String(topic) == "esp32/servo_angle")
{
  next_angle = strtol(messageTemp, NULL, 10); //convert string from pi to integer
  angle = (next_angle + 8) << 4; //adds 8 to make it a 4 bit message and shift by 4
  Serial.print("servo angle: "); //verifys the message
  Serial.println(next_angle);
```

Figure 16 - Vehicle Control Code Snippet 2

At the end – downstairs ESP – the message is separated into two 4-bit messages using the bitwise 'and' operation and shifting the values so they contain only the 4 least significant bits and store as a signed integer. These numbers can then be processed into a suitable range to be used to control the motors (0-255) and servo angle (0 -180). The scheme for doing so is displayed in the flowchart in Figure 17. Note that on receive, the values calculated are stored in a global variable and is then adjusted – put to use – in the main loop of the program. A snippet of this is shown in figure 16 while the full code is available in the appendix.

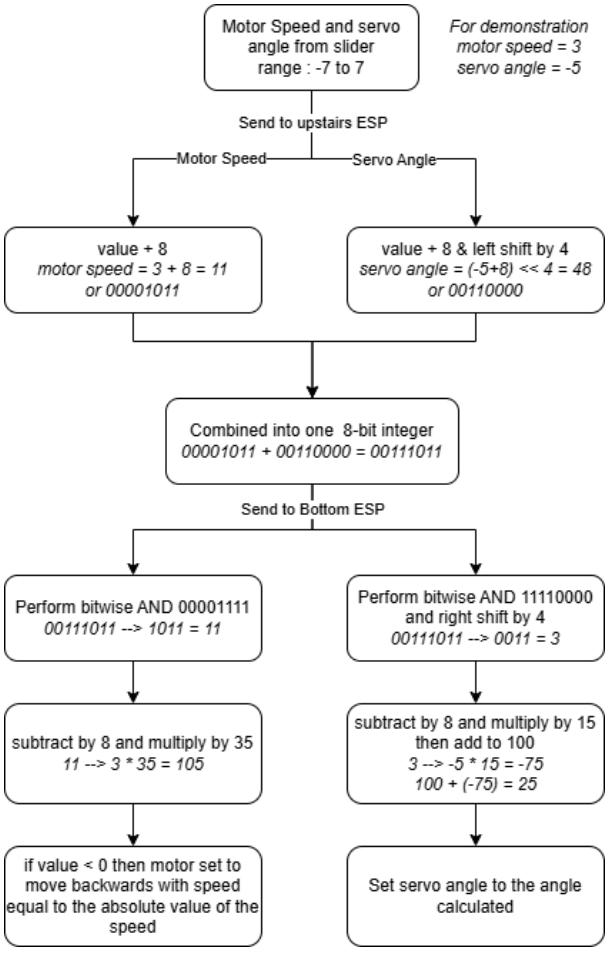

Figure 17 - Data Flowchart for Vehicle control

```
void adjust_param(int bytes)
{
  value = Wire.read();
  // read values received to verify
  // Serial.print("value: ");
  // Serial.println(value);

  speed = value & 15;
  speed -= 8;
  angle = value >> 4;
  angle -= 8;

  //Display values to verify
  // Serial.print("Speed: ");
  // Serial.println(speed);
  // Serial.print("Angle: ");
  // Serial.println(angle);
  // Serial.println();

  speed = speed * 35;
  angle = 100 + (angle * 15);
```

Figure 18 - Vehicle Control Downstairs ESP Code Snippet

Another function was also implemented called the auto-stop which will force the EEEBot to stop by bringing the motor speed to zero if the distance to an object infront is less than 20cm, detected by the HC-SR04. This feature also slows the car down proportionally to the distance in front to allow a smoother stop. The function can be activated using a button on the dashboard.

The Node-Red flow is shown in Figure 19 which creates the dashboard shown in Figure 21. After successful testing, it can be seen that the EEEBot responds to the commands given in the dashboard – example shown in Figure 22.

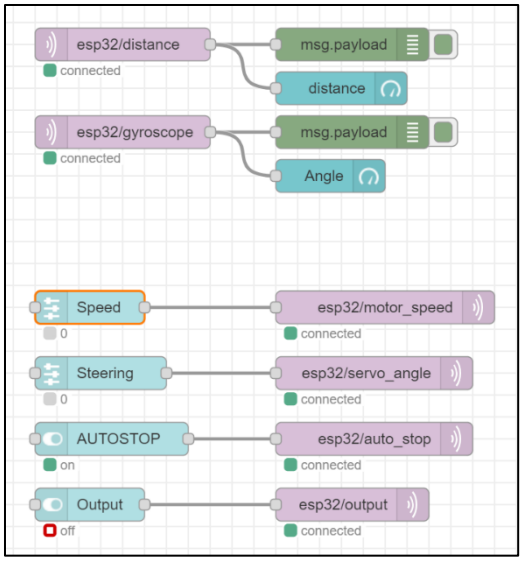

Figure 19 - Vehicle Control Node-Red Flow

```
WiFi connected
IP address:
172.20.10.11
MPU6050 Found!
GyroErrorX: -0.01
GyroErrorY: 0.04
GyroErrorZ: 0.01
Attempting MQTT connection...connected
Message arrived on topic: esp32/servo_angle. Message: -6
servo angle: -6
transmitting motor speed level: 8
Message arrived on topic: esp32/servo_angle. Message: -5
servo angle: -5
transmitting motor speed level: 8
Message arrived on topic: esp32/servo_angle. Message: -4
servo angle: -4
```

Figure 20 - Vehicle Control Serial Monitor

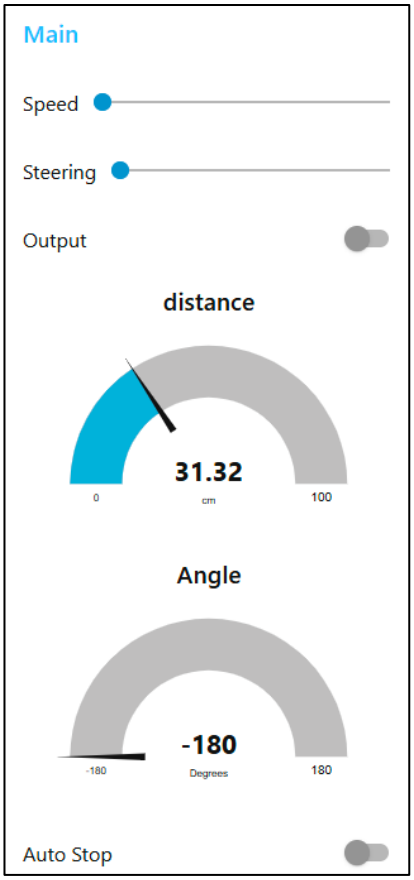

Figure 21 - Vehicle Control Dashboard

EEEBot Control Testing

To test the EEEBot control, the vehicle went through an obstacle course set up to demonstrate precise control. The performance on the obstacles was suboptimal. This is mainly due to the lack of responsiveness from the vehicles to the controls – latency issues – as well as mechanical issues from both the servo and motors.

The first complication, latency, causes the inputs to take a considerable amount of time until they are acted upon. This means that the EEEBot often went off course as the command to stop the vehicle did not process in time. Another side effect of this is that it is often not intuitive to the user how the vehicle will react to a certain command as it is difficult to see a response in the expected time frame.

The second limitation is the mechanical limitations and how the components react to an input. The first is the servo which turns a different amount on either side, with the turning left being more sensitive than turning right. This means that controlling the car can be awkward as the user will need to keep in

mind to turn left much less than they would expect. The second is the motor which requires a maximum power to get started. The motor requires that power level supplied be the maximum – fastest setting possible – for the motor to overcome inertia and start moving but can slow down to the desired speed. This means that if the vehicle stops, it will require an abrupt start.

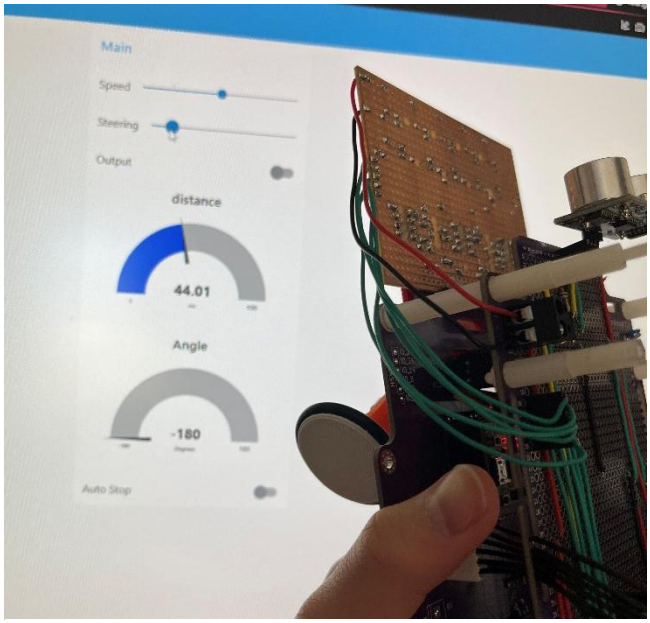

Figure 22 - Vehicle Control Demonstration

Another issue with the vehicle control is the auto-stop feature. The problem is that the function only operates when the user changes the speed of the vehicle meaning that it will assist the user stop the car when they are close to the wall but if the user left the vehicle as is, the function would not take effect and the car would still crash. To fix this, the code for the auto stop simply needs to be copied to the main loop. This would result in the feature taking action at all times to check for an obstacle.

The combination of these complications mentioned above results in a vehicle that is difficult to control despite meeting the requirements for the project. Some improvements can be made to certain parts of the code such as the servo control so that when turning left, a smaller value is used to achieve a more similar angular displacement on either side with the same input as the user would expect

Command Driven Maze Navigation

The objective for project week 7 was to build an image recognition software that will then give an output that will control the EEEBot accordingly.

Project Setup

The image recognition software was to be built using the programming language C++ and the OpenCV library. OpenCV is an open-source computer vision and machine learning software library [10] that will help provide tools for building the image recognition model in this project week. The software is run on a raspberry pi which will provide the real time camera feed to process. The raspberry pi is connected to Wi-Fi which will enable it to be remotely controlled from a laptop using the RealVNC Viewer application [11]. The raspberry pi is connected to an ESP32 using I^2C communication [5] with the pins on the board which will allow it to communicate with the ESP and control the EEEBot.

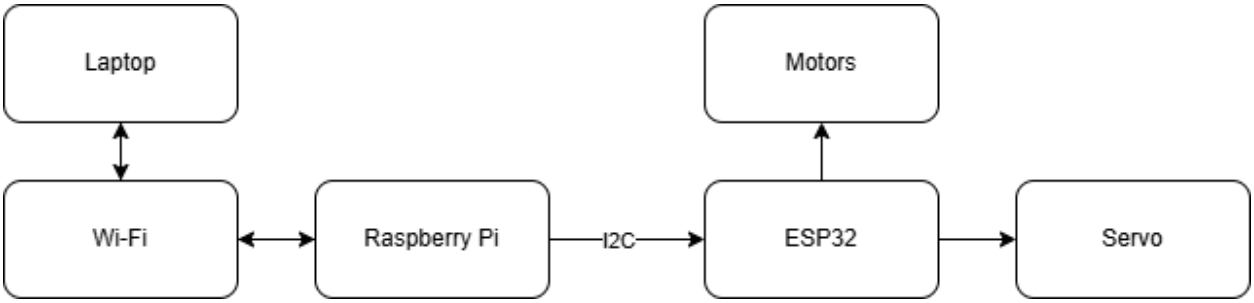

Figure 23 - Image Recognition Setup Diagram

For I^2C communication, the raspberry pi will be the master device and send commands to the ESP which will in turn command the motor and servo. As the raspberry pi is connected using I^2C and does not need to send information over 2 Wi-Fi using MQTT like the previous project week, the upstairs ESP can be neglected until sensors are to be incorporated. This simplifies the communication as it removes devices that need to be accounted for. For I^2C, the raspberry pi is connected using the SCL and SDA pins which are pins number 3 and 5 in Figure 24. Power to the raspberry pi can be provided through a micro-USB cable or using pins 2 and 6. To use a micro-USB connector to power the pi, an adapter must be used to both interface the cable as well as to act as a regulator between the pi and the battery as it is a sensitive component.

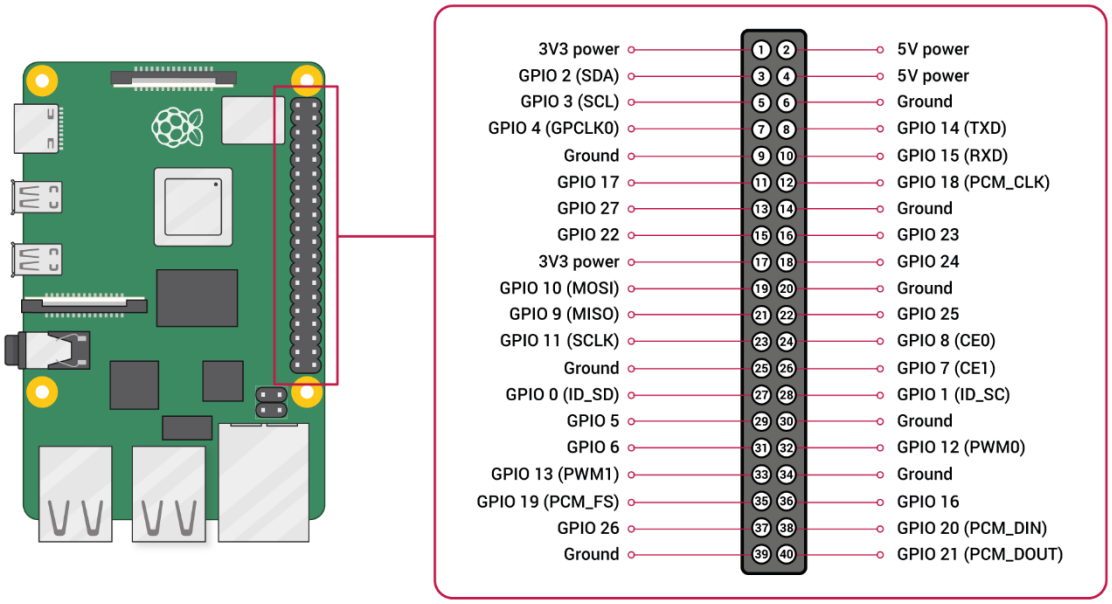

Figure 24 - Raspberry Pi Pinout [16]

Image Recognition

The image recognition is, as mentioned before, programmed in C++ using the OpenCV library. For the program to recognize a symbol – Figure 25 – it must compare what it sees to a template. This means that the symbol in an image must first be isolated from the rest of the image for the program to be able to compare it. To do this, the program will perform the steps shown in Figure 26. This process is performed on every frame of the camera feed.

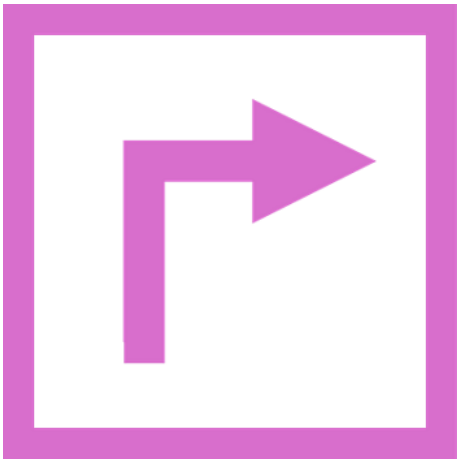

Figure 25 - Right Symbol

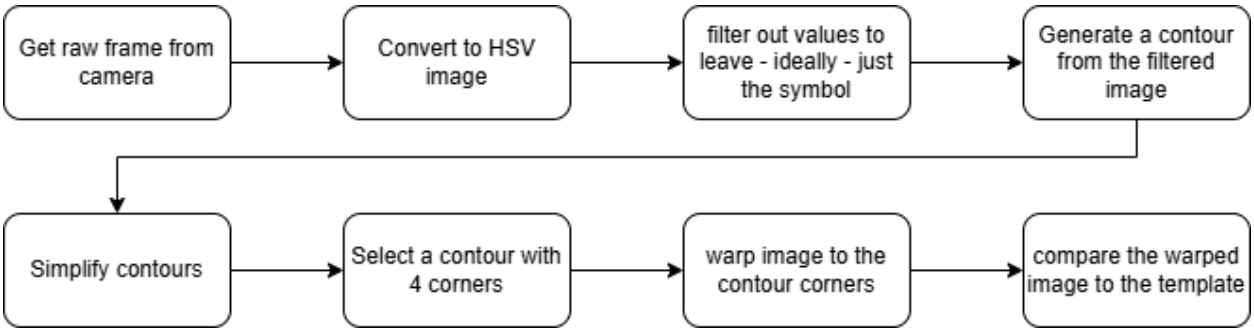

Figure 26 - Image Recognition Steps Block Diagram

Image Processing Steps

To process a frame [12] of the camera feed to recognize the symbol, first the image is turned into a HSV format from RGB. A HSV image is used as it is more intuitive and practical to use when applied to image recognition due to how colors are represented [13]. Then, the image is filtered to only show a specified range of values. The range is determined by manual testing using another piece of software that allows the user to adjust the filter values and show the result in real time as shown in Figure 28. The image on the left is the original while the image on the right has the filter applied. This process gave the values (87, 50, 73) for the minimum and (111, 255, 170) for the maximum - Figure 27.

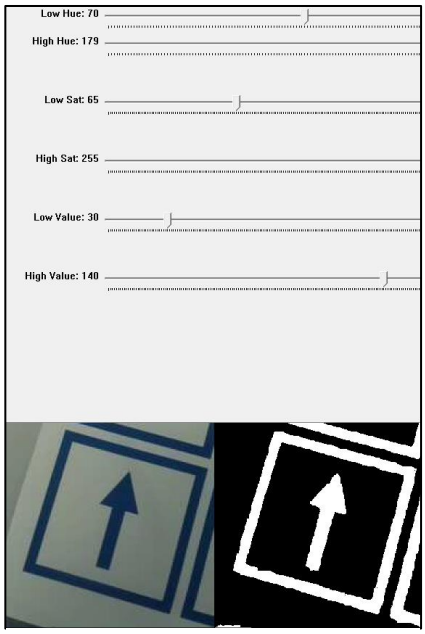

Figure 28 - Getting HSV Filter Values

```
cvtColor(frame, frame_hsv, COLOR_BGR2HSV);

inRange(frame_hsv, Scalar(87, 50, 73), Scalar(111, 255, 170), gray_frame);
```

Figure 27 - Image Recognition Code Snippet 1

Then, a contour is applied which outlines the edges – Figure 30. The contours are stored as vectors [14] and so by applying a contour, the coordinates of these contours on the image can be obtained as well. However, the contours are too detailed, storing a large number of coordinate points and so they are simplified. The simplified contours – Figure 29 – store a much smaller number of coordinate points. These contours can be sorted to leave only those which contain only four coordinate points – as a square has four corners. In Figure 31, the contours that contains 4 coordinate points are highlighted in blue.

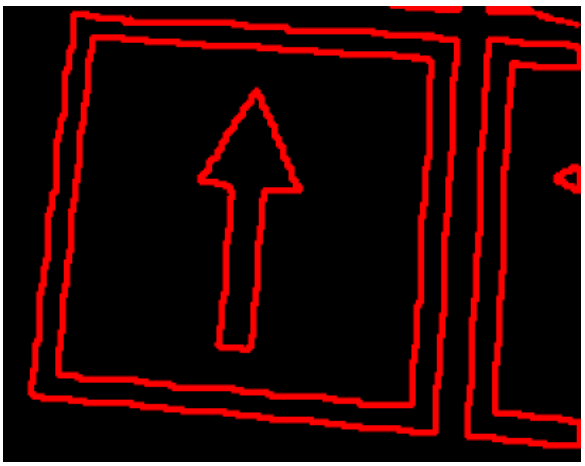

Figure 30 - Image Contours

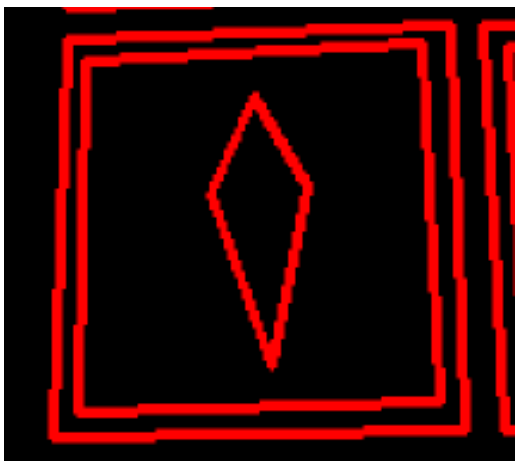

Figure 29 - Simplified Contours

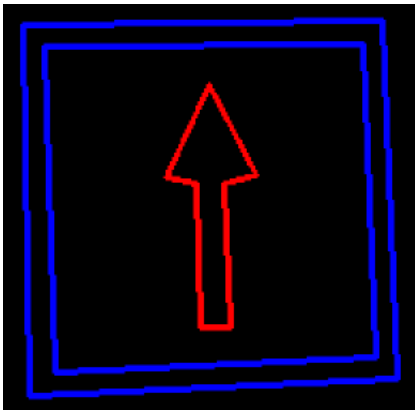

Figure 31 - Highlighted Simplified Contours

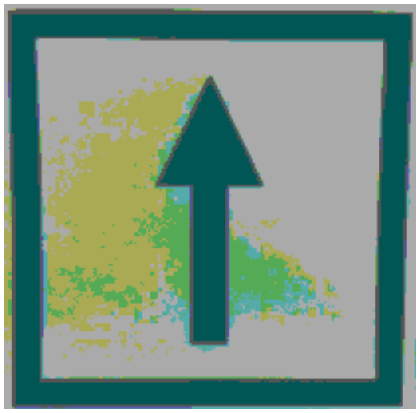

Figure 32 - Warped Image

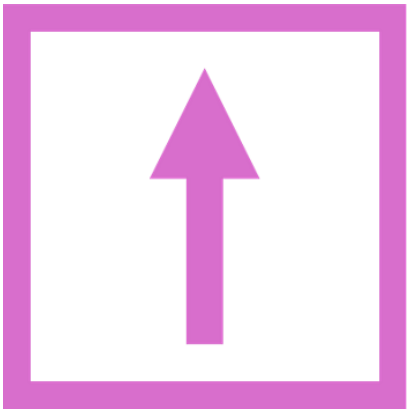

Figure 33 - Forwards Symbol

Using these four coordinate points, the image can be warped so that the four coordinate points become the corners of the visible image – shown in Figure 32. This warped image acquired from processing the frame through the steps above is now in the apropriate format for the software to be able to compare to the templates as they now have the same dimensions as well as having focused in on the subject. This warped image is compared to all three templates – one for each direction : left, right and forwards. The comparison produces a percentage match, quantifying how similar the images are and the one it is most similar to is the direction of the sign. However, there must also be a net to check if what the software is looking at is a symbol at all and so the comparison must be at least above 90% a match to be considered a symbol at all. This value of 90% was chosen by observing output characteristics where if image warp was performed adequately, it would result in an image match value of 90% or higher on average.

```
warpPerspective(frame, warped_frame, perspectiveMat, Size(350, 350));
matchTemplate(warped_frame, forwards_img, forwards_result, 3);
matchTemplate(warped_frame, left_img, left_result, 3);
matchTemplate(warped_frame, right_img, right_result, 3);
```

Figure 34 - Image Recognition Code Snippet 2

The image recognition demostrations shown below in Figure 35 and Figure 36 displays the key stages of image recognition in three separate windows, plain, contours and warped. The plain window displays the camera feed. The contours window displays the simplified contours applied with the red contours being the general cotours, the blue being those with four coordinate points and the green being the contour selected to warp the image.

Using this method resulted in one major issue which is the selection of which contours to use. This is because currently the first contour with four coordinate points found is used to warp the image for image recognition. This means that while the symbo could be visible in frame, the software would select the wrong area to focus on. To fix this, the contours could be organized by size by calculating the area of the contours and selecting the largest one. This can dramatically increase the accuracy of the software but also increase the complexity of the code and slow it down.

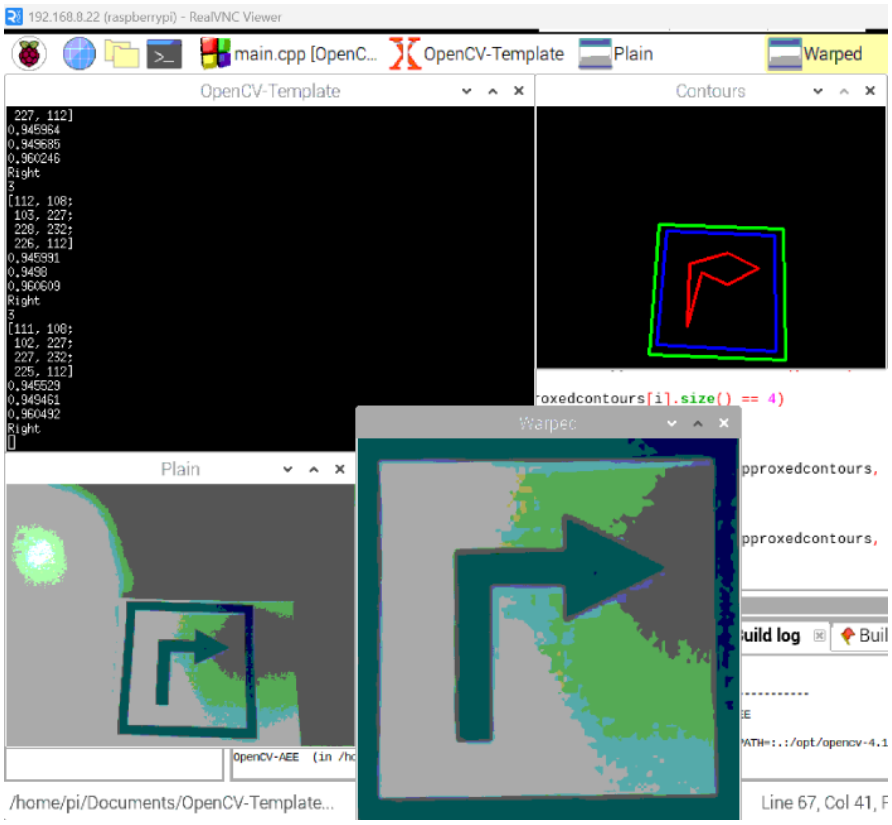

Figure 35 - Image Recognition Demonstration Right

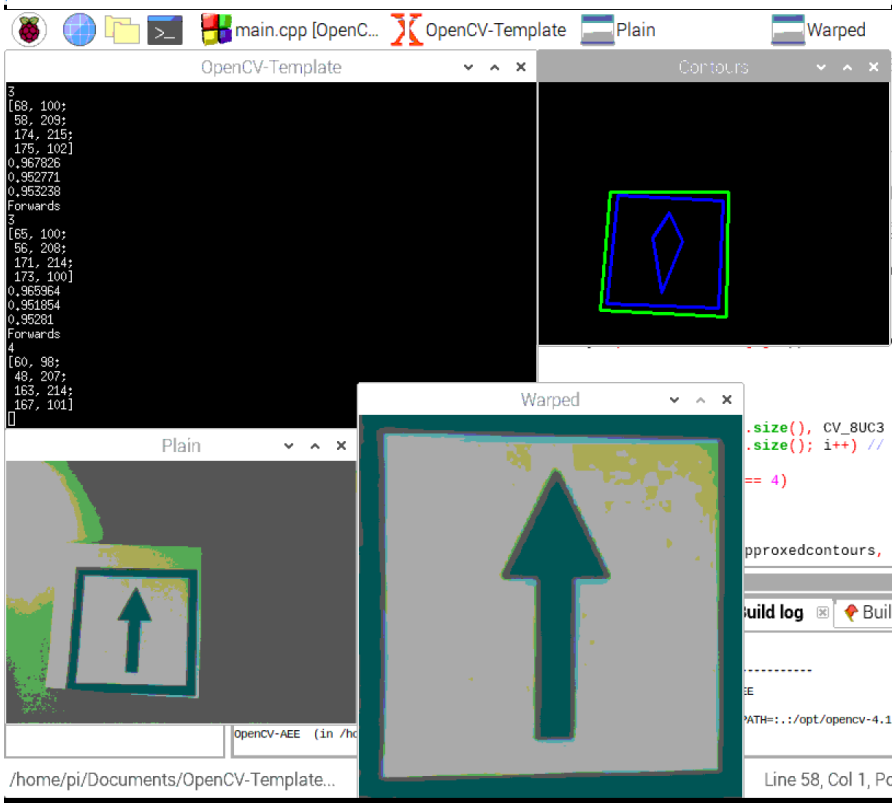

Figure 36 - Image Recognition Demonstration Forwards

Another issue is that even when the correct contour is chosen to warp the image, the orientation of the contours used can result in the warped image being flipped upside down or in an orientation that is not wanted. This results in the program drawing an unwanted conclusion such as identifying the symbol as a right instead of a left. An example of which is shown below in Figure 37. With further testing – where the program was shown each symbol 5 times – it can be seen that this occurs more often than expected – as shown in Table 1.

Test	Correct	Incorrect	Unidentified
Forwards	4	1	0
Left	3	2	0
Right	3	2	0

Table 1 - Image Recognition Testing

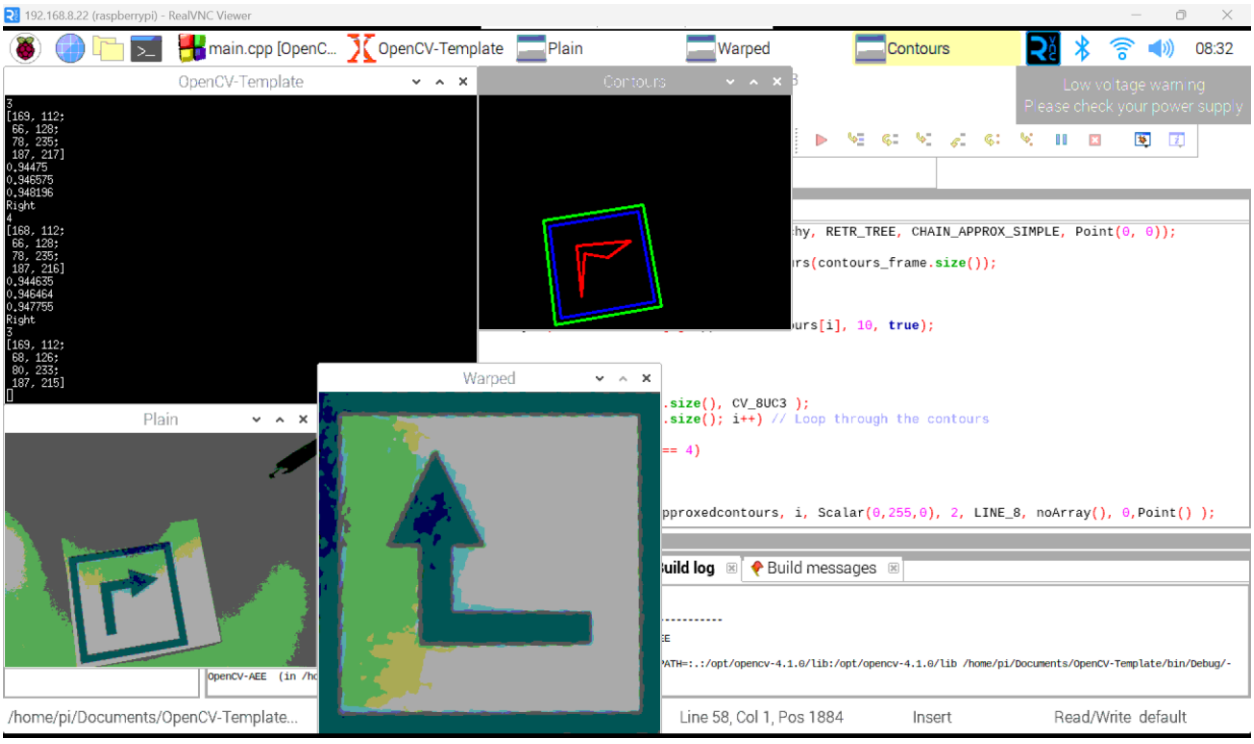

Figure 37 - Image Recognition Demonstration Error

Vehicle Control & Testing

To control the EEEBot, the raspberry pi sends an integer between 1 and 3 to the ESP32 based on the image recognition output. These commands are then interpreted as a command to move forwards, turn left or turn right on the ESP32 using the code shown in Figure 38. The downstairs ESP32 is the only ESP communicated with as mentioned above and it acts as the slave device. The main program loop is empty, meaning the adjust_param function is only called when a signal is received from the raspberry pi and the EEEBot will remain idle in between. The movement commands on the ESP32 runs for 1000 miliseconds each. This was done to simplify the code but results in a poor turn that over shoots the target of a 90° turn. To improve on this, the upstairs ESP can be incorporated in order to utilize the MPU6050 in aiding turning. However, the MPU6050 requires significant tuning due to the error in it's output, constantly outputting an angular velocity value of around 0.01 - 0.2. This results in the gyroscopic data being unusable despite the attempt at calibrating using the calculate_IMU_error function shown in Figure 39.

```
void adjust_param(int bytes)
{
    value = Wire.read();
    Serial.print("value: ");
    Serial.println(value);

    motors(255, 255);
    if (value == 1)
    {
        goForwards();
        delay(1000);
        stopMotors();
    }
    else if (value == 2)
    {
        TurnLeft(1000);
    }
    else
    {
        TurnRight(1000);
    }
}
```

Figure 38 - Vehicle Control Code Snippet 2

```
void calculate_IMU_error()
{
    c = 0;
    // Read gyro values 200 times
    while (c < 500)
    {
        sensors_event_t a, g, temp;
        mpu.getEvent(&a, &g, &temp);
        // Sum all readings
        GyroErrorX += g.gyro.x;
        GyroErrorY += g.gyro.y;
        GyroErrorZ += g.gyro.z;
        c++;
    }
    // Divide the sum by 200 to get the error value
    GyroErrorX = GyroErrorX / 500.0;
    GyroErrorY = GyroErrorY / 500.0;
    GyroErrorZ = GyroErrorZ / 500.0;
}
```

Figure 39 - Vehicle Control Code Snippet 3

Another issue vehicle control is that every frame that the image recognition software detects a symbol it will send a command. This results in the command being executed multiple times and so the final destination of the EEEBot becomes drastically different from the intended. To fix this, a flag could be implemented to show that an action is already in progress, preventing the EEEBot from executing another action or receiving any commands. This is a solution to the problem that can be implemented and tested.

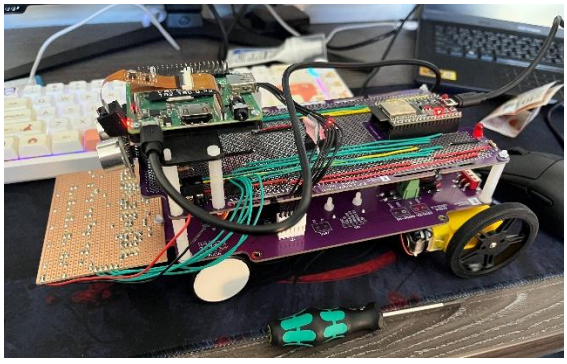

Figure 40 - EEEBot Setup

Conclusion

In the two lab weeks covered – weeks 6 and 7 – two forms of control were successfully implemented onto the EEEBot. The first – covered in week 6 – was the wireless control that uses the raspberry pi's dashboard as a controller. The second – covered in week 7 – was a command driven maze navigation that implemented an image recognition system to control the vehicle.

The wireless control was implemented using the MQTT communication protocol to send wireless commands from the raspberry pi to the ESP32 as well as I^2C communication to allow the components on the vehicle to communicate through wired transmission. The inputs on the dashboard used sliders that allowed the motor speed and steering angle to be controlled. The dashboard also displayed real time sensor measurements from the EEEBot which included distance and angle using the readings from the HC-SR04 and the MPU6050. Despite the code working as intended, latency issues (0.5 – 2 seconds) as well as mechanical issues concerning the motor and servo results in rigid movements and imprecise control. The implementation of an auto function was also attempted but did not work as intended due to only activating with a user input.

As for the command driven maze navigation, the raspberry pi was used to run a C program that takes an input from an onboard camera and processes each frame in order to find and identify a command sign. The image worked mostly to specification as it is able to identify the symbols and commands with a threshold of a 90% match. However, the control system derived from the output will often result in the EEEBot moving in an unexpected manner. This is because a movement command is sent every frame that a command is detected, resulting in the vehicle moving far more than intended. This, coupled with the lack of help from sensor readings caused the vehicle to not be as effective despite a working image recognition system.

The results from both lab weeks follow a common theme of working well in a very controlled environment but lacks refinement to be practically implemented. They work as a proof of concept to be built upon. Many mechanical challenges presented here can also be overcome using well designed software such as MPU calibration as well as smoother servo control. The wireless control can be developed further to be used in other applications such as remote-control robots for hard to navigate terrain while the command driven control can be developed into autonomous vehicles to follow road signs. Many improvements to both systems can be made such as an auto stop function that can be activated at any time or a command queueing system for the command driven maze navigation so that when one command is being executed, others are ignored.

References

[1] CodeChamps, "Distance Measurement Using HC-SR04 Via NodeMCU," Autodesk instructables, [Online]. Available: https://www.instructables.com/Distance-Measurement-Using-HC-SR04-Via-NodeMCU/. [Accessed 02 04 2025].

[2] TDK, "MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Device," [Online]. Available: https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/#:~:text=Six-Axis%20(Gyro%20%2B%20Accelerometer)%20MEMS%20MotionTracking™%20Device&text=The%20MPU-6050%20devices%20combine,complex%206-axis%20MotionFusion%20algorithms.. [Accessed 18 01 2025].

[3] EMQX Team, "Mastering MQTT: The Ultimate Beginner's Guide for 2025," EMQX, 26 02 2025. [Online]. Available: https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt. [Accessed 02 04 2025].

[4] HiveMQ, "MQTT Essentials," HiveMQ, [Online]. Available: https://www.hivemq.com/mqtt/. [Accessed 30 04 2025].

[5] S. Campbell, "basics of the i2c communication protocol," [Online]. Available: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/. [Accessed 02 04 2025].

[6] N. Zambetti, K. Söderby and J. Hylén, "Inter-Integrated Circuit (I2C) Protocol," Arduino, 20 09 2024. [Online]. Available: https://docs.arduino.cc/learn/communication/wire/. [Accessed 30 04 2025].

[7] SunFounder, "Learn Arduino/Raspberry pi/ESP32," SunFounder, [Online]. Available: https://docs.sunfounder.com/projects/esp32-starter-kit/en/latest/components/component_esp32_extension.html. [Accessed 02 04 2025].

[8] HiveMQ Team, "mqtt essentials part 5 mqtt topics best practices," hivemq, 20 02 2024. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/. [Accessed 02 04 2025].

[9] knolleary, "pubsubclient," 20 05 2020. [Online]. Available: https://github.com/knolleary/pubsubclient. [Accessed 02 04 2025].

[10] OpenCV, "About," OpenCV, [Online]. Available: https://opencv.org/about/#:~:text=OpenCV%20(Open%20Source%20Computer%20Vision,perception%20in%20the%20commercial%20products.. [Accessed 03 04 2025].

[1] RealVNC, "How to Set Up a Raspberry Pi RealVNC Remote Access Connection," RealVNC, 12 08 2021. [Online]. Available: https://www.realvnc.com/en/blog/raspberry-pi-vnc/?lai_vid=OX0mxmrzpS0K&lai_sr=0-4&lai_sl=l&lai_p=1. [Accessed 03 04 2025].

[2] Mux Video Glossary, "frame," Mux, [Online]. Available: https://www.mux.com/video-glossary/frame. [Accessed 03 04 2025].

[3] N. Glover, "hsv vs rgb," [Online]. Available: https://handmap.github.io/hsv-vs-rgb/. [Accessed 03 04 2025].

[4] W3Schools, "C++ vectors," w3schools, [Online]. Available: https://www.w3schools.com/cpp/cpp_vectors.asp. [Accessed 03 04 2025].

[5] N. Alexeev, "How many bytes can be sent in a single i2c message?," 25 03 2015. [Online]. Available: https://electronics.stackexchange.com/questions/161508/how-many-bytes-can-be-sent-in-a-single-i2c-message. [Accessed 02 04 2025].

[6] ASASSY, "Introduction to the Raspberry Pi GPIO and Physical Computing," [Online]. Available: https://learn.sparkfun.com/tutorials/introduction-to-the-raspberry-pi-gpio-and-physical-computing/gpio-pins-overview. [Accessed 03 04 2025].

Appendix: Week 6 Upstairs ESP32 Code

```
#include <WiFi.h>

#include <ESPmDNS.h>
#include <NetworkUdp.h>
#include <ArduinoOTA.h>

#include <esp_now.h>
#include <PubSubClient.h>
#include <Wire.h>

// Sonar

#define sonar_echo 35
#define sonar_trig 14
#define sonar_led 32

float duration, distance;
char distanceString[8];

// Gyroscope

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#define gyro_led 18
float x = 0, y = 0, z = 0;
float GyroErrorX, GyroErrorY, GyroErrorZ;
int c = 0;
char zAxisString[8];
Adafruit_MPU6050 mpu;

// Raspberry pi communication

const char *ssid = "Pun’s iPhone";
const char *password = "password";
const char *mqtt_server = "172.20.10.10";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;

// downstairs communication
```

```
#define downstairsAdd 0x04
int auto_stop = 0;

void setup()
{

    Serial.begin(115200);

    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);

    // sonar set up

    pinMode(sonar_led, OUTPUT);
    pinMode(sonar_trig, OUTPUT);
    pinMode(sonar_echo, INPUT);

    // mpu set up

    if (!mpu.begin())
    {
        Serial.println("Failed to find MPU6050 chip");
        while (1)
        {
            delay(10);
        }
    }
    Serial.println("MPU6050 Found!");

    mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
    mpu.setGyroRange(MPU6050_RANGE_500_DEG);
    mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

    delay(100);
    calculate_IMU_error();
}

void setup_wifi()
{
    delay(10);
    // we start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
```

```
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

int8_t speed = 8, angle = 8, next_speed, next_angle;
uint8_t transmission;

void callback(char *topic, byte *message, unsigned int length)
{
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    char messageTemp[3];

    for (int i = 0; i < length; i++)
    {
        Serial.print((char)message[i]);
        messageTemp[i] = (char)message[i];
    }
    Serial.println();

    if (String(topic) == "esp32/motor_speed")
    {
        next_speed = strtol(messageTemp, NULL, 10);
        if (auto_stop == 1)
        {
            if (distance < 20)
            {
                next_speed = 0;
            }
        }
        speed = next_speed + 8;
        Serial.print("motor speed level: ");
```

```
        Serial.println(next_speed);
    }
    else if (String(topic) == "esp32/servo_angle")
    {
        next_angle = strtol(messageTemp, NULL, 10); // convert string from pi to
integer
        angle = (next_angle + 8) << 4;              // adds 8 to make it a 4 bit
message and shift by 4
        Serial.print("servo angle: ");              // verifys the message
        Serial.println(next_angle);
    }
    else if (String(topic) == "esp32/auto_stop")
    {
        if (messageTemp == "on")
        {
            Serial.println("auto stop on");
            auto_stop = 1;
        }
        else if (messageTemp == "off")
        {
            Serial.println("auto stop off");
            M
                auto_stop = 0;
        }
    }

    Serial.print("transmitting motor speed level: ");
    Serial.println(speed);
    transmission = speed + angle;
    Wire.beginTransmission(downstairsAdd);
    Wire.write(transmission);
    Wire.endTransmission();
    delay(10);
}

void reconnect()
{
    // loop until we're reconnected
    while (!client.connected())
    {
        Serial.print("Attempting MQTT connection...");
        // attempt to connect
        if (client.connect("ESP8266Client"))
        {
            Serial.println("connected");
```

```
            // subscribe
            client.subscribe("esp32/output");
            client.subscribe("esp32/auto_stop");
            client.subscribe("esp32/motor_speed");
            client.subscribe("esp32/servo_angle");
        }
        else
        {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // wait 5 seconds before retrying
            delay(5000);
        }
    }
}

void loop()
{
    if (!client.connected())
    {
        reconnect();
    }
    client.loop();

    // get sonar data

    digitalWrite(sonar_trig, LOW);
    delayMicroseconds(2);
    digitalWrite(sonar_trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(sonar_trig, LOW);

    // Convert and publish distance

    duration = pulseIn(sonar_echo, HIGH);
    distance = (duration * .0343) / 2;
    dtostrf(distance, 1, 2, distanceString);
    // Serial.print("distance X: ");
    // Serial.println(distanceString);
    client.publish("esp32/distance", distanceString);

    // get mpu data

    sensors_event_t a, g, temp;
```

```
    mpu.getEvent(&a, &g, &temp);

    x += floor(((g.gyro.x - GyroErrorX) / 131 * 10) * 100) / 10;
    y += floor(((g.gyro.y - GyroErrorY) / 131 * 10) * 100) / 10;
    z += floor(((g.gyro.z - GyroErrorZ) / 131 * 10) * 100) / 10;

    // Convert and publish gyroscope data
    dtostrf(z, 1, 2, zAxisString);
    // Serial.print("Rotation X: ");
    // Serial.print(x);
    // Serial.print(", Y: ");
    // Serial.print(y);
    // Serial.print(", Z: ");
    // Serial.print(z);
    // Serial.println(" Degs");
    // client.publish("esp32/gyroscope", zAxisString);
    delay(10);
}

void calculate_IMU_error()
{
    c = 0;
    // Read gyro values 200 times
    while (c < 500)
    {
        sensors_event_t a, g, temp;
        mpu.getEvent(&a, &g, &temp);
        // Sum all readings
        GyroErrorX += g.gyro.x;
        GyroErrorY += g.gyro.y;
        GyroErrorZ += g.gyro.z;
        c++;
    }
    // Divide the sum by 200 to get the error value
    GyroErrorX = GyroErrorX / 500.0;
    GyroErrorY = GyroErrorY / 500.0;
    GyroErrorZ = GyroErrorZ / 500.0;
    Serial.print("GyroErrorX: ");
    Serial.println(GyroErrorX);
    Serial.print("GyroErrorY: ");
    Serial.println(GyroErrorY);
    Serial.print("GyroErrorZ: ");
    Serial.println(GyroErrorZ);
}
```

Appendix: Week 6 Downstairs ESP32 Code

```
#include <Wire.h>
#define downstairsAdd 0x04

#include <ESP32Servo.h>

Servo steeringServo;

#define enA 33 // enableA command line
#define enB 25 // enableB command line

#define INa 26 // channel A direction
#define INb 27 // channel A direction
#define INc 14 // channel B direction
#define INd 12 // channel B direction

// setting PWM properties
const int freq = 2000;
const int ledChannela = 11; // the ESP32 servo library uses the PWM channel 0 by
default, hence the motor channels start from 1
const int ledChannelb = 12;
const int resolution = 8;

int steeringAngle = 90; // variable to store the servo position
int servoPin = 13;      // the servo is attached to IO_13 on the ESP32

void setup()
{
    // put your setup code here, to run once:

    // i2C com set up
    Wire.begin(downstairsAdd);
    Wire.onReceive(adjust_param);
    Serial.begin(115200);

    // servo and motor setup

    ledcAttachChannel(enA, freq, resolution, ledChannela);
    ledcAttachChannel(enB, freq, resolution, ledChannelb);

    // allow allocation of all timers
    ESP32PWM::allocateTimer(0);
    ESP32PWM::allocateTimer(1);
```

```
    ESP32PWM::allocateTimer(2);
    ESP32PWM::allocateTimer(3);
    steeringServo.setPeriodHertz(50);          // standard 50Hz servo
    steeringServo.attach(servoPin, 500, 2400); // attaches the servo to the pin
using the default min/max pulse widths of 1000us and 2000us

    pinMode(INa, OUTPUT);
    pinMode(INb, OUTPUT);
    pinMode(INc, OUTPUT);
    pinMode(INd, OUTPUT);

    // initialise serial communication
    Serial.println("ESP32 Running"); // sanity check
}

int mode = 0;
int value, current_angle = 100;
int speed = 0, neg_speed, angle;

void loop()
{
    Serial.print("Speed: ");
    Serial.println(speed);
    Serial.print("Angle: ");
    Serial.println(angle);

    steeringServo.write(current_angle);
    if (speed > 0)
    {
        motors(speed, speed);
        goForwards();
    }
    else if (speed < 0)
    {
        neg_speed = speed * -1;
        motors(neg_speed, neg_speed);
        stopMotors();
        goBackwards();
    }
    else
    {
        stopMotors();
    }
    delay(50);
}
```

```
void adjust_param(int bytes)
{
    value = Wire.read();
    // read values received to verify
    // Serial.print("value: ");
    // Serial.println(value);

    speed = value & 15;
    speed -= 8;
    angle = value >> 4;
    angle -= 8;

    // Display values to verify
    //  Serial.print("Speed: ");
    //  Serial.println(speed);
    //  Serial.print("Angle: ");
    //  Serial.println(angle);
    //  Serial.println();

    speed = speed * 35;
    angle = 100 + (angle * 15);

    steeringServo.write(angle);
    current_angle = angle;
}

void moveSteering(int move_from, int move_to)
{
    int increment = 1, index;
    if (move_to < move_from)
    {
        increment = -1;
    }

    for (index = move_from; index != move_to; index += increment)
    {
        steeringServo.write(index);
        delay(5);
    }
}

void motors(int leftSpeed, int rightSpeed)
{
    // set individual motor speed
```

```
    // the direction is set separately

    // constrain the values to within the allowable range
    leftSpeed = constrain(leftSpeed, 0, 255);
    rightSpeed = constrain(rightSpeed, 0, 255);

    ledcWrite(enA, leftSpeed);
    ledcWrite(enB, rightSpeed);
    delay(25);
}

void TurnLeft(int duration)
{
    steeringServo.write(90);

    for (steeringAngle = 90; steeringAngle >= 45; steeringAngle -= 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    goAntiClockwise();
    delay(duration);
    stopMotors();
    for (steeringAngle = 45; steeringAngle <= 90; steeringAngle += 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
}

void TurnRight(int duration)
{
    steeringServo.write(90);
    for (steeringAngle = 90; steeringAngle <= 160; steeringAngle += 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    goClockwise();
```

```
    delay(duration);
    stopMotors();
    for (steeringAngle = 160; steeringAngle >= 90; steeringAngle -= 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
}

void TurnAround()
{
    steeringServo.write(90);
    for (steeringAngle = 90; steeringAngle >= 45; steeringAngle -= 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    goAntiClockwise();
    delay(650);
    stopMotors();
    for (steeringAngle = 45; steeringAngle <= 160; steeringAngle += 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    goClockwiseBack();
    delay(470);
    stopMotors();
    for (steeringAngle = 160; steeringAngle >= 100; steeringAngle -= 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
}

void goForwards()
{
```

```
    digitalWrite(INa, HIGH);
    digitalWrite(INb, LOW);
    digitalWrite(INc, HIGH);
    digitalWrite(INd, LOW);
}

void goBackwards()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, HIGH);
    digitalWrite(INc, LOW);
    digitalWrite(INd, HIGH);
}

void goClockwise()
{
    digitalWrite(INa, HIGH);
    digitalWrite(INb, LOW);
    digitalWrite(INc, LOW);
    digitalWrite(INd, LOW);
}

void goClockwiseBack()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, HIGH);
    digitalWrite(INc, LOW);
    digitalWrite(INd, LOW);
}

void goAntiClockwise()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, LOW);
    digitalWrite(INc, HIGH);
    digitalWrite(INd, LOW);
}

void goAntiClockwiseBack()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, LOW);
    digitalWrite(INc, LOW);
    digitalWrite(INd, HIGH);
}
```

```
void stopMotors()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, LOW);
    digitalWrite(INc, LOW);
    digitalWrite(INd, LOW);
}
```

Appendix: Week 7 ESP32 Code

```
#include <Wire.h>
#define downstairsAdd 0x04
#define piAdd 0x22

#include <ESP32Servo.h>

Servo steeringServo;

#define enA 33 // enableA command line
#define enB 25 // enableB command line

#define INa 26 // channel A direction
#define INb 27 // channel A direction
#define INc 14 // channel B direction
#define INd 12 // channel B direction

// setting PWM properties
const int freq = 2000;
const int ledChannela = 11; // the ESP32 servo library uses the PWM channel 0 by
default, hence the motor channels start from 1
const int ledChannelb = 12;
const int resolution = 8;

int steeringAngle = 90; // variable to store the servo position
int servoPin = 13;      // the servo is attached to IO_13 on the ESP32

void setup()
{
    // put your setup code here, to run once:

    // i2C com set up
    Wire.begin(piAdd);
    Wire.onReceive(adjust_param);
    Serial.begin(115200);

    // servo and motor setup

    ledcAttachChannel(enA, freq, resolution, ledChannela);
    ledcAttachChannel(enB, freq, resolution, ledChannelb);

    // allow allocation of all timers
    ESP32PWM::allocateTimer(0);
```

```
    ESP32PWM::allocateTimer(1);
    ESP32PWM::allocateTimer(2);
    ESP32PWM::allocateTimer(3);
    steeringServo.setPeriodHertz(50);          // standard 50Hz servo
    steeringServo.attach(servoPin, 500, 2400); // attaches the servo to the pin
using the default min/max pulse widths of 1000us and 2000us

    pinMode(INa, OUTPUT);
    pinMode(INb, OUTPUT);
    pinMode(INc, OUTPUT);
    pinMode(INd, OUTPUT);

    // initialise serial communication
    Serial.println("ESP32 Running"); // sanity check

    steeringServo.write(90);
}

int value;

void loop()
{
}

void adjust_param(int bytes)
{
    value = Wire.read();
    Serial.print("value: ");
    Serial.println(value);

    motors(255, 255);
    if (value == 1)
    {
        goForwards();
        delay(1000);
        stopMotors();
    }
    else if (value == 2)
    {
        TurnLeft(1000);
    }
    else
    {
        TurnRight(1000);
    }
```

```
}

void moveSteering(int move_from, int move_to)
{
    int increment = 1, index;
    if (move_to < move_from)
    {
        increment = -1;
    }

    for (index = move_from; index != move_to; index += increment)
    {
        steeringServo.write(index);
        delay(5);
    }
}

void motors(int leftSpeed, int rightSpeed)
{
    // set individual motor speed
    // the direction is set separately

    // constrain the values to within the allowable range
    leftSpeed = constrain(leftSpeed, 0, 255);
    rightSpeed = constrain(rightSpeed, 0, 255);

    ledcWrite(enA, leftSpeed);
    ledcWrite(enB, rightSpeed);
    delay(25);
}

void TurnLeft(int duration)
{
    steeringServo.write(90);

    for (steeringAngle = 90; steeringAngle >= 45; steeringAngle -= 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    goAntiClockwise();
    delay(duration);
    stopMotors();
```

```
    for (steeringAngle = 45; steeringAngle <= 90; steeringAngle += 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
}

void TurnRight(int duration)
{
    steeringServo.write(90);
    for (steeringAngle = 90; steeringAngle <= 160; steeringAngle += 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    goClockwise();
    delay(duration);
    stopMotors();
    for (steeringAngle = 160; steeringAngle >= 90; steeringAngle -= 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
}

void TurnAround()
{
    steeringServo.write(90);
    for (steeringAngle = 90; steeringAngle >= 45; steeringAngle -= 1)
    {                                       // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    goAntiClockwise();
    delay(650);
    stopMotors();
    for (steeringAngle = 45; steeringAngle <= 160; steeringAngle += 1)
```

```
    {                                          // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                             // waits 15ms for the servo to reach
the position
    }
    goClockwiseBack();
    delay(470);
    stopMotors();
    for (steeringAngle = 160; steeringAngle >= 100; steeringAngle -= 1)
    {                                          // goes from 180 degrees to 0 degrees
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                             // waits 15ms for the servo to reach
the position
    }
}

void goForwards()
{
    digitalWrite(INa, HIGH);
    digitalWrite(INb, LOW);
    digitalWrite(INc, HIGH);
    digitalWrite(INd, LOW);
}

void goBackwards()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, HIGH);
    digitalWrite(INc, LOW);
    digitalWrite(INd, HIGH);
}

void goClockwise()
{
    digitalWrite(INa, HIGH);
    digitalWrite(INb, LOW);
    digitalWrite(INc, LOW);
    digitalWrite(INd, LOW);
}

void goClockwiseBack()
{
    digitalWrite(INa, LOW);
```

```
    digitalWrite(INb, HIGH);
    digitalWrite(INc, LOW);
    digitalWrite(INd, LOW);
}

void goAntiClockwise()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, LOW);
    digitalWrite(INc, HIGH);
    digitalWrite(INd, LOW);
}

void goAntiClockwiseBack()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, LOW);
    digitalWrite(INc, LOW);
    digitalWrite(INd, HIGH);
}

void stopMotors()
{
    digitalWrite(INa, LOW);
    digitalWrite(INb, LOW);
    digitalWrite(INc, LOW);
    digitalWrite(INd, LOW);
}
```

Appendix: Week 7 Raspberry Pi Image Recognition Code

```
// password
//  Include files for required libraries
#include <stdio.h>

#include "opencv_aee.hpp"
#include "main.hpp" // You can use this file for declaring defined values and
functions
#include "pi2c.h"

Pi2c car(0x22); // Configure the I2C interface to the Car as a global variable

using namespace std;
using namespace cv;
using namespace cv::xfeatures2d;
using std::cout;
using std::endl;

string forward_path = "symbols_comp/forwards.png";
string left_path = "symbols_comp/left.png";
string right_path = "symbols_comp/right.png";
Mat forwards_img = imread(forward_path);
Mat left_img = imread(left_path);
Mat right_img = imread(right_path);

void setup(void)
{
    setupCamera(320, 240); // Enable the camera for OpenCV
}

int main(int argc, char **argv)
{
    setup(); // Call a setup function to prepare IO and devices

    vector<Point> end_points;
    end_points.push_back(Point(0, 350));
    end_points.push_back(Point(350, 350));
    end_points.push_back(Point(350, 0));
    end_points.push_back(Point(0, 0));

    cv::namedWindow("Plain"); // Create a GUI window called photo
    cv::namedWindow("Warped");
    cv::namedWindow("Contours");
```

```
    while (1) // Main loop to perform image processing
    {
        Mat flipped_frame, frame;

        while (frame.empty())
            frame = captureFrame(); // Capture a frame from the camera and store
in a new matrix variable

        //            flip(flipped_frame, frame, 1);

        Mat gray_frame, frame_hsv, equalised_frame;
        cvtColor(frame, frame_hsv, COLOR_BGR2HSV);

        inRange(frame_hsv, Scalar(87, 50, 73), Scalar(111, 255, 170),
gray_frame);

        std::vector<std::vector<cv::Point>> contours_frame;
        std::vector<Vec4i> hierarchy;
        cv::findContours(gray_frame, contours_frame, hierarchy, RETR_TREE,
CHAIN_APPROX_SIMPLE, Point(0, 0));

        std::vector<std::vector<cv::Point>>
approxedcontours(contours_frame.size());

        for (int i = 0; i < contours_frame.size(); i++)
        {
            cv::approxPolyDP(contours_frame[i], approxedcontours[i], 10, true);
        }

        int square_frames[100] = {-1};
        int count = 0;
        Mat drawing = Mat::zeros(gray_frame.size(), CV_8UC3);
        for (int i = 0; i < approxedcontours.size(); i++) // Loop through the
contours
        {
            if (approxedcontours[i].size() == 4)
            {
                if (count == 0)
                {
                    drawContours(drawing, approxedcontours, i, Scalar(0, 255, 0),
2, LINE_8, noArray(), 0, Point());
                }
                else
                {
```

```
                    drawContours(drawing, approxedcontours, i, Scalar(255, 0, 0),
2, LINE_8, noArray(), 0, Point());
                }
                square_frames[count] = i;
                count++;
            }
            else
            {
                drawContours(drawing, approxedcontours, i, Scalar(0, 0, 255), 2,
LINE_8, noArray(), 0, Point()); // Draw each in red
            }
        }

        // cout << approxedcontours[square_frames[0]] << "\n";

        Mat warped_frame;
        Mat perspectiveMat;
        vector<float> forwards_result, left_result, right_result;

        cout << approxedcontours.size() << "\n";
        if (square_frames[0] == -1)
        {
            cout << "no frame\n";
            warped_frame = frame;
        }
        else
        {
            cout << approxedcontours[square_frames[0]] << "\n";
            perspectiveMat = findHomography(approxedcontours[square_frames[0]],
end_points);
            warpPerspective(frame, warped_frame, perspectiveMat, Size(350, 350));
            matchTemplate(warped_frame, forwards_img, forwards_result, 3);
            matchTemplate(warped_frame, left_img, left_result, 3);
            matchTemplate(warped_frame, right_img, right_result, 3);
            cout << forwards_result[0] << "\n"
                 << left_result[0] << "\n"
                 << right_result[0] << "\n";
            if (forwards_result[0] > left_result[0] && forwards_result[0] >
right_result[0])
            {
                cout << "Forwards\n";
                car.i2cWriteArduinoInt(1);
            }
            else if (left_result[0] > right_result[0])
            {
```

```
                cout << "Left\n";
                car.i2cWriteArduinoInt(2);
            }
            else
            {
                cout << "Right\n";
                car.i2cWriteArduinoInt(3);
            }
        }

        cv::imshow("Plain", frame); // Display the image in the window
        cv::imshow("Warped", warped_frame);
        cv::imshow("Contours", drawing);

        int key = cv::waitKey(1); // Wait 1ms for a keypress (required to update
windows)

        key = (key == 255) ? -1 : key; // Check if the ESC key has been pressed
        if (key == 27)
            break;
    }

    closeCV(); // Disable the camera and close any windows

    return 0;
}
```