EEEE1002: Applied Electrical and Electronic Engineering: Construction Project

Individual Lab Report 2

Covering Weeks 4 and 5

Student ID Number: 20645251

Report Due Date: 09/01/2025

Abstract

In the two lab weeks documented in lab report two, the objective is to implement sensors to the EEEBot to give it new capabilities in order to complete two sets of tasks. The first is to implement sonar and gyroscopic sensors to the EEEBot. The aim is to give the EEEBot the capability to navigate a maze by detecting its surroundings and processing that information in real time. The second is to implement optical sensors to the EEEBot. These sensors will allow the EEEBot to follow a black line path while correcting its own trajectory as it moves to stay on that path. This report will cover the operation and implementation of each sensor to the EEEBot and how the data collected from each sensor is used to complete each task as well as the result of said tasks.

Contents

1 Introduction

This lab report is a continuation of the first lab report – Individual Lab Report 1. In the first lab report, fundamental skills required and related to building EEEBot as well as simple circuits were covered alongside the construction of the EEEBot. During those lab weeks – weeks one to three – the EEEBot was given the capability to move and to be controlled by code uploaded to it but lack the ability to adjust to situations in real time as it had no method to gather information of its surroundings.

In this lab report, the focus over the two lab weeks is to implement sensors to EEEBot to allow it to become more autonomous. The task/challenges are used as a benchmark to test and evaluate the EEEBots abilities to adapt to its environment; these challenges are maze navigation and line following.

Section two and the first half of section four will mainly cover the function of the sensors used each week as well as the construction of how it is implemented into the EEEBot. Then, the control logic implemented will be explained. This covers how the EEEBot will navigate the maze as well as how it follows the line using the PID method as detailed in sections 3 and 4.4.

Both of these challenges are being implemented and integrated into a larger system in the real world to use for systems such as autonomous cars. They are also being used separately with lines following being used in industrial situations for operations like moving equipment around a warehouse on a set path while maze navigation can be used to find the shortest path or traversing unknown terrain.

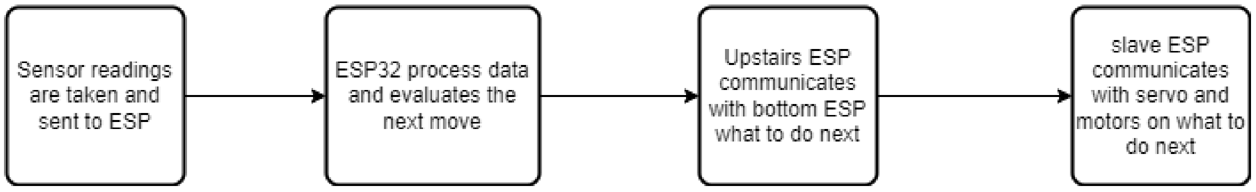

Sensor readings are taken and sent to ESP
ESP32 process data and evaluates the next move
Upstairs ESP communicates with bottom ESP what to do next
slave ESP communicates with servo and motors on what to do next

Figure 1 - System overview diagram

2 The Integration of the Additional Sensor Subsystems

To perform the maze navigation, more sensors were added to the EEEBot to give it the capabilities needed. These sensors included the HC-SR04 sonar and MPU6050 gyroscope and accelerometer.

2.1 HC-SR04

The HC-SR04 is an ultrasonic sensor used for non-contact range measurement – to detect the distance to the nearest object directly in front of the sensor. The sensor works by emitting a sonar signal of a specific frequency out and recording the time taken for that signal to come back. As it works using sonar it is unaffected by ambient lighting and can be used in the dark. A specific pre-determined frequency is used so that the sensor does not confuse it for random noise. The actual output of the sensor is the time between the signal being emitted and received however distance – in meters – can be calculated by using the speed of the signal in air where

$$Distance = V_{signal}/2T \tag{1}$$

A '2' is used in the equation as we only want half the time since the time in this accounts for the time to the object and back. The sensor can be used to measure distances between 2 and 400 cm in front of the sensor with a precision of 3mm.

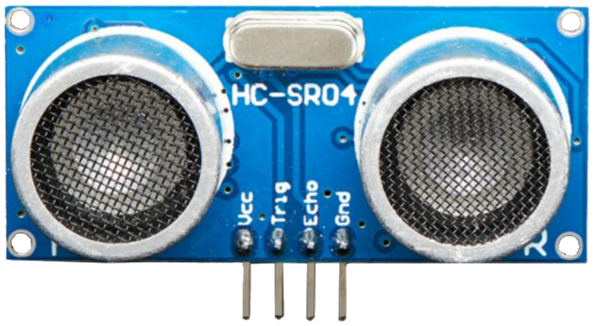
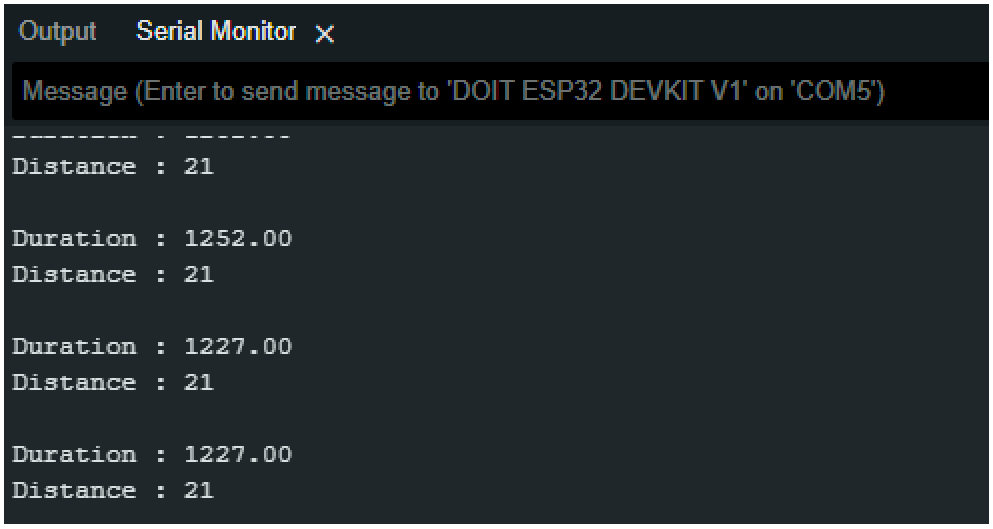


Figure 2 - HCSR04 [4]

The HC-SR04 has 4 pins which are (from left to right) the V_{CC}, trigger, echo and ground pins. The voltage input and ground pins are to power the sensor while the trigger pin is used to emit a sonar signal and the echo pin sends a signal when the sonar signal returns. The trigger pin can be set to a high signal for a period to indicate that a signal is being emitted. The duration that the pin is on is typically not long, around 10ms, as to not cause the signals going out to interfere with the signals coming in. Once the reflected signal is received, the echo pin outputs a signal proportional to the time taken to receive the reflected signal.

The HC-SR04 is connected to an ESP32 that can then read the data from the sensor to interpret it and give the sensor commands using a program written in C.


Output
Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

```
Distance : 21

Duration : 1252.00
Distance : 21

Duration : 1227.00
Distance : 21

Duration : 1227.00
Distance : 21
```

Figure 3 - HCSR04 Serial Monitor Output

2.2 MPU6050

The MPU6050 is a six-axis motion tracking device that works both as a gyroscope and accelerometer. The device combines a three-axis accelerometer and a three-axis gyroscope on the same dye. The device detects changes in motion, i.e. angular velocity and acceleration in reference to gravity. The MPU6050 has a range of up to ±2000°/s for angular velocity and ±16g for acceleration. [1]

The MPU6050 has 8 pins being V_{CC}, ground, serial clock (SCL), serial data (SDA), auxiliary serial data, auxiliary serial clock, I^2C address select and interrupt.

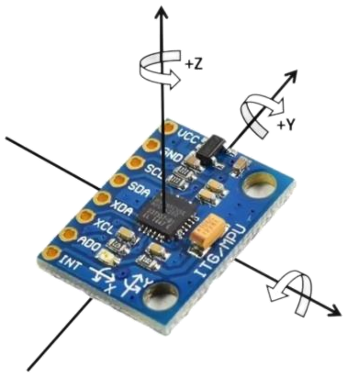
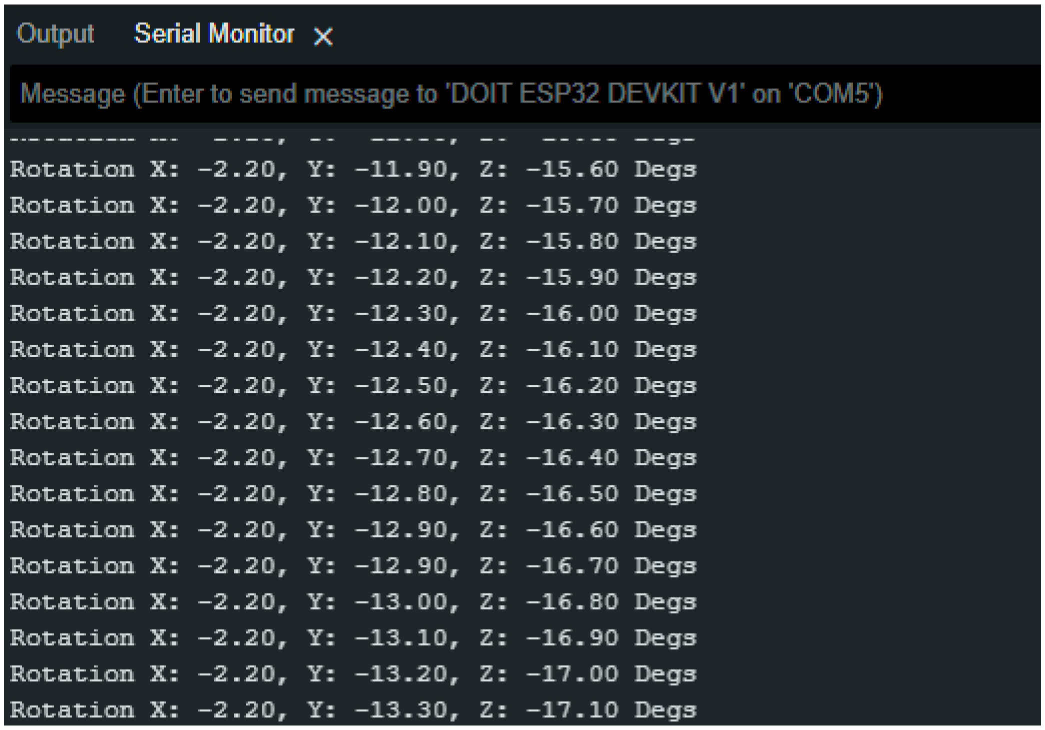


Figure 4 - MPU6050 [3]

The pins we used to connect it to the ESP32 are the SCL and SDA pins in order to read data from it. These are connected to the SCL and SDA pins on the ESP32 respectively. When programming, the MPU6050 can be uniquely against other devices also connected to the ESP32 using the SCL and SDA as it has its own address code. When programming the MPU6050, it is easy to use a library to help simplify the program. The libraries used are the 'Adafruit MPU6050' and 'Adafruit Sensor'.

For our use case during this lab week, only the angular position of the EEEBot is needed, therefore only the angular velocity data is used. Each cycle, the angular velocity is read and is used as an average. This value is then used to calculate the change in angular position and sum it to the previously recorded angular position. This is done for all 3 axis each cycle.


Output
Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

```
Rotation X: -2.20, Y: -11.90, Z: -15.60 Degs
Rotation X: -2.20, Y: -12.00, Z: -15.70 Degs
Rotation X: -2.20, Y: -12.10, Z: -15.80 Degs
Rotation X: -2.20, Y: -12.20, Z: -15.90 Degs
Rotation X: -2.20, Y: -12.30, Z: -16.00 Degs
Rotation X: -2.20, Y: -12.40, Z: -16.10 Degs
Rotation X: -2.20, Y: -12.50, Z: -16.20 Degs
Rotation X: -2.20, Y: -12.60, Z: -16.30 Degs
Rotation X: -2.20, Y: -12.70, Z: -16.40 Degs
Rotation X: -2.20, Y: -12.80, Z: -16.50 Degs
Rotation X: -2.20, Y: -12.90, Z: -16.60 Degs
Rotation X: -2.20, Y: -12.90, Z: -16.70 Degs
Rotation X: -2.20, Y: -13.00, Z: -16.80 Degs
Rotation X: -2.20, Y: -13.10, Z: -16.90 Degs
Rotation X: -2.20, Y: -13.20, Z: -17.00 Degs
Rotation X: -2.20, Y: -13.30, Z: -17.10 Degs
```

Figure 5 - MPU6050 Serial Monitor Output

2.3 I^{2}C Communication

I^{2}C is a protocol used for communication between devices with one master device and one or more slave devices connected to it. The master device can write to or read from all slave devices connected to it. In this week, one more ESP32 microcontroller is added to the board with the new one being the master and the one that came first being the slave. In this new circuit, the master controls the operations of the EEEBot and is also connected to both the MPU6050 and HC-SR04. I^{2}C communication only requires the SQL and SDA lines to work and so makes its topology very simple and easy to work with. However, it can only send and receive 8 bits of data so the message must be thought out before being sent. [2]

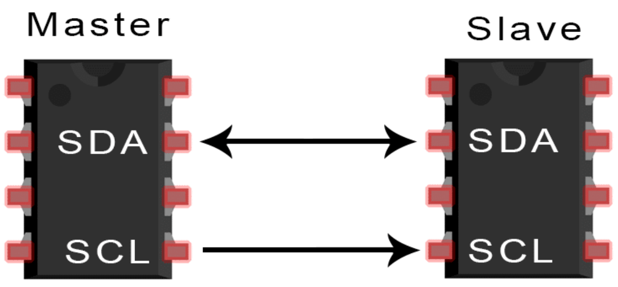


Figure 6 - I^{2}C Communication Diagram [10]

2.4 Stripboard Construction

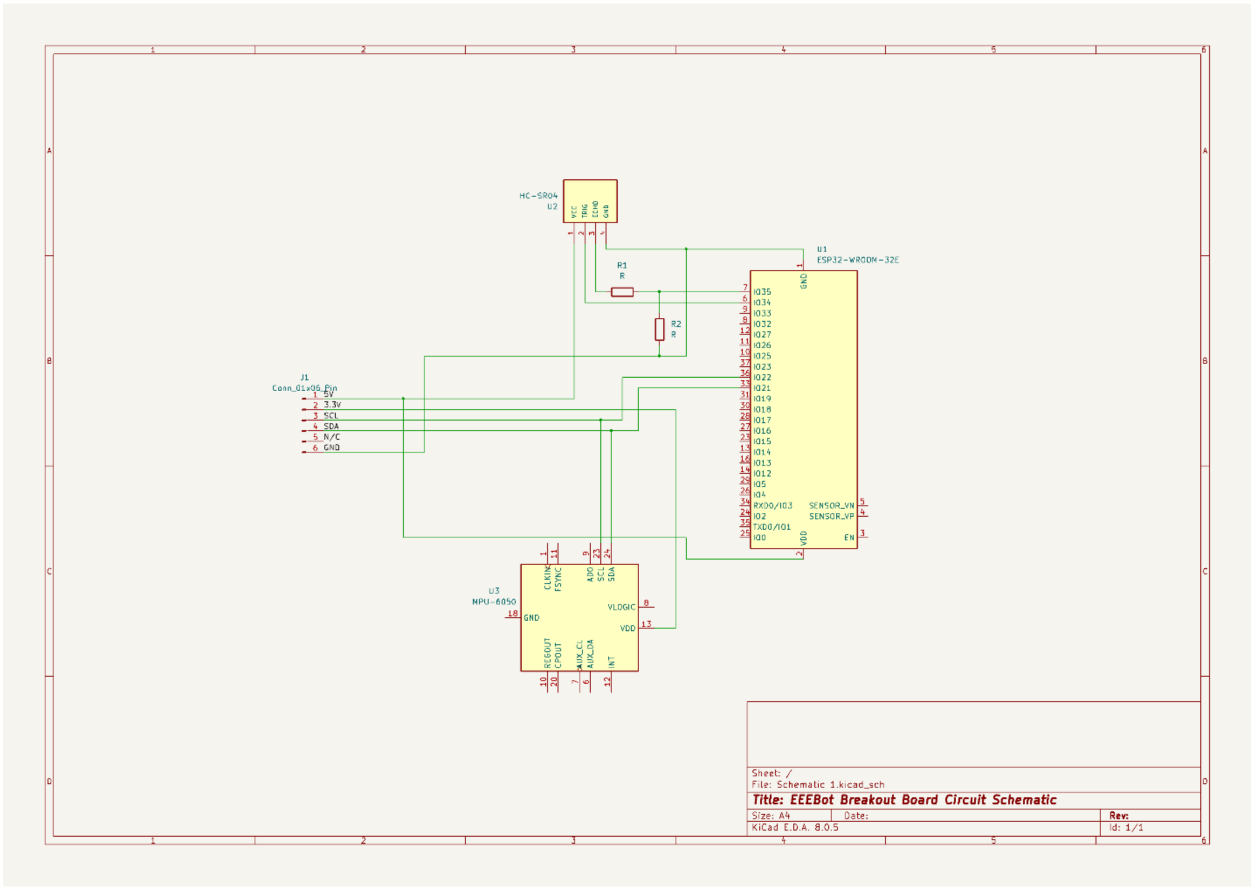


Figure 8 - EEEBot Breakout Board Circuit Schematic

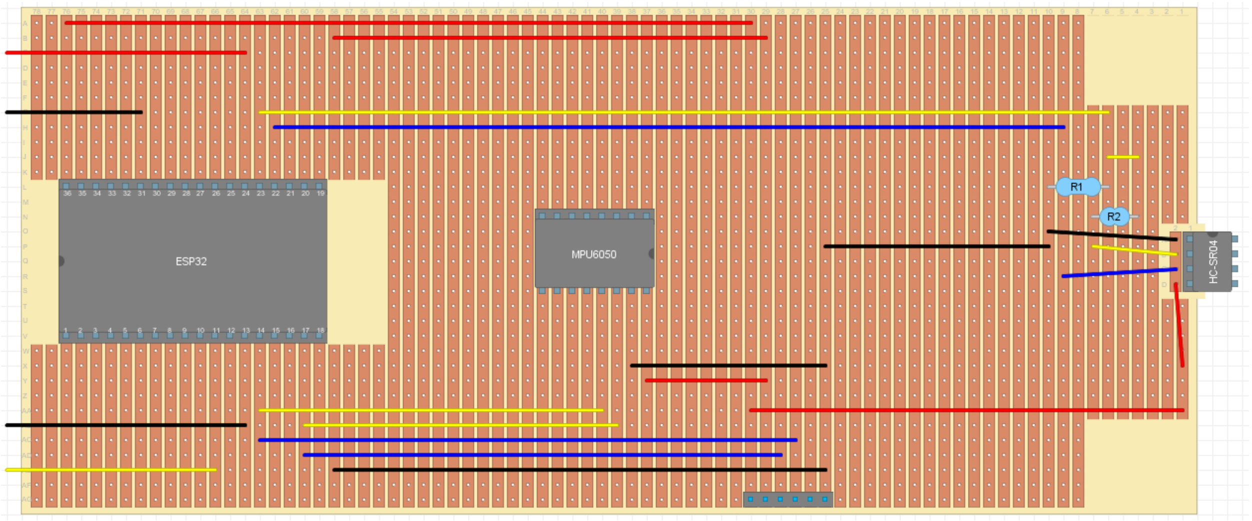


Figure 7 - EEEBot Breakout Board Layout Plan

The implementation of the new sensors onto the EEEBot is done on a breakout board using the plan shown above in Figure 7 which was designed based on the schematic shown in Figure 8. The schematic was designed to connect the HC-SR04 and the MPU6050 to the ESP32 so that it can be read from and controlled. The design used information from the data sheets of each component - [3] and [4]. The six labelled connectors in Figure 8 are used to connect the top and bottom layers of the EEEBot, providing power and ground to the second level as well as connect the master and slave ESP32s. The breakout board sits on top of the mainboard supported by 8 nylon standoffs in the four corners of the board. The layout of the breakout board intends to group together wires and section them off in order to leave space for any future components. The main issue with the design is that it initially intended to use GPIO pin 34 as the trigger pin to send an output to the HC-SR04. However, this cannot be done as it was later discovered that GPIO pin 34 is designated input only and cannot output a signal [5]. To compensate, a wire was added to connect GPIO pin 34 to GPIO pin 14 instead. Then, the track from the original wire to GPIO 34 was cut and the problem was fixed. No other issues regarding the stripboard layout and design occurred after this.

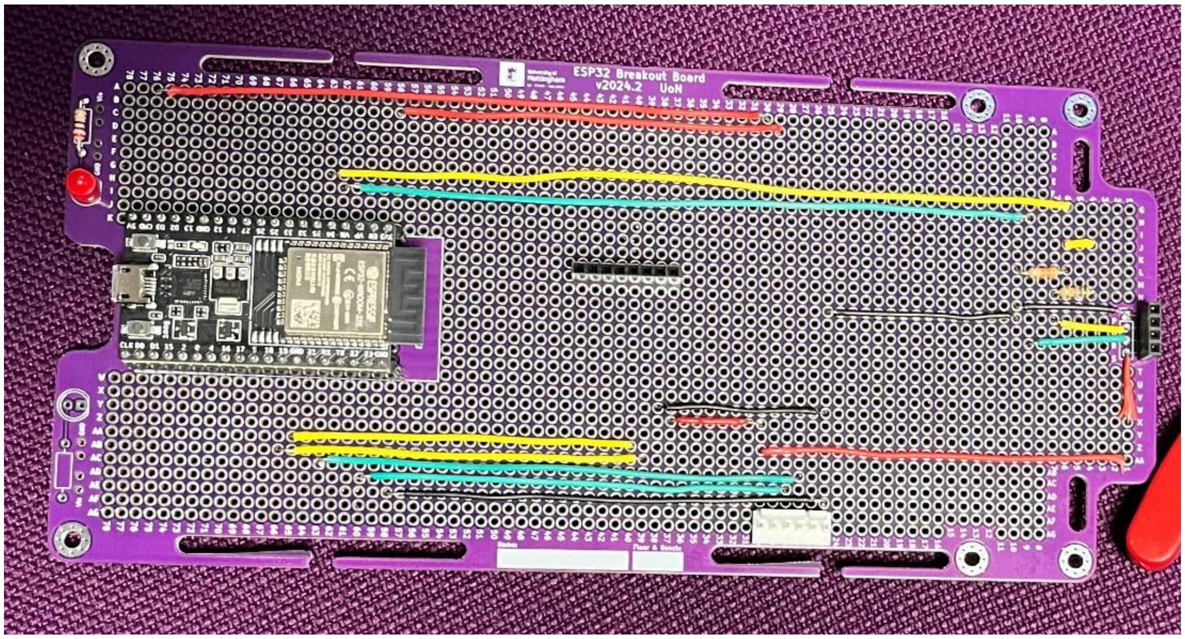


Figure 9 - EEEBot Breakout Board Constructed

3 The Design & Implementation of the Maze Navigation Solution

3.1 Outline

To navigate the maze, the EEEBot will follow the simple method of hugging the left wall. This way, it will prevent the EEEBot from going back the same way it came from and eventually reach the exit. To do this, the EEEBot will travel in a straight line until it reaches a wall where it will then attempt to turn left, if it cannot turn left as there is another wall it will make a U-turn and end up in a position that is the same as turning right from the perspective of its original orientation. Using this method, the EEEBot will be able to navigate simple mazes and not crash into a wall.

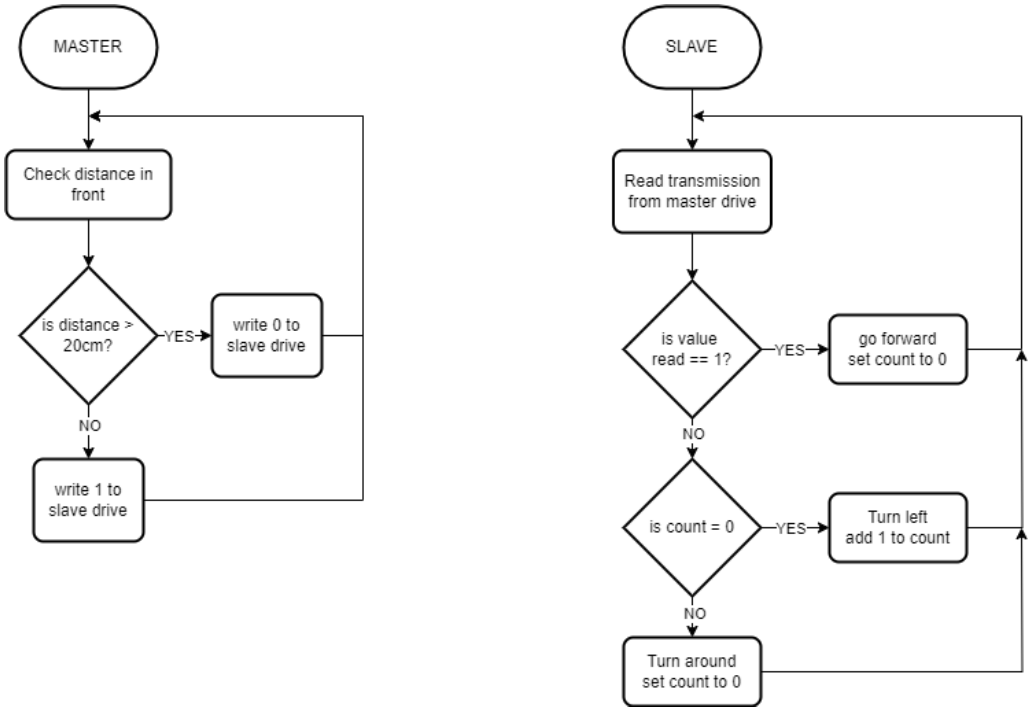
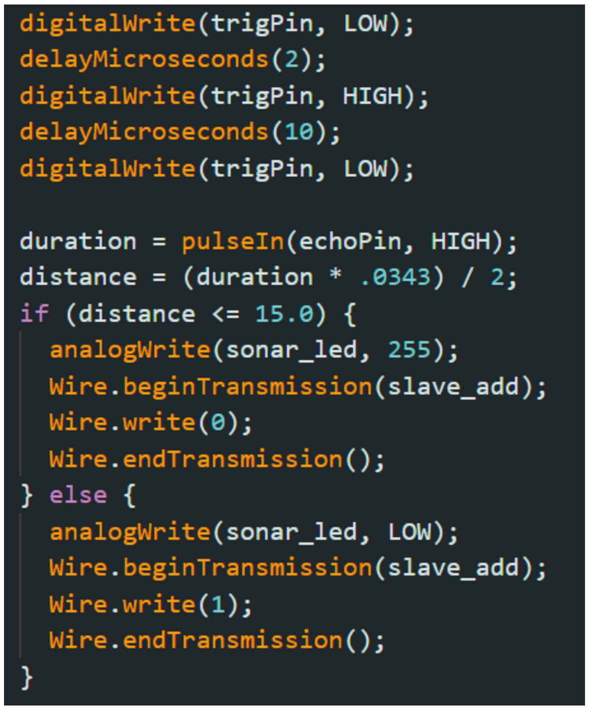


Figure 10 - Maze Navigation Solution Logic Flowchart

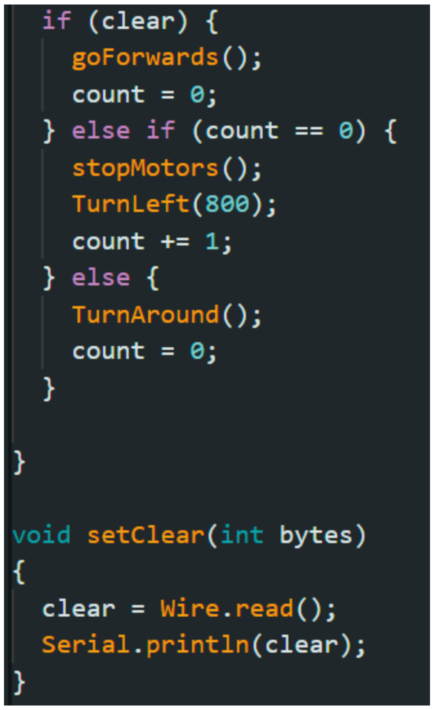
The code follows the logic shown in Figure 10. The program is separated into two main parts, one being for the upstairs – master – ESP32 and one for the downstairs – slave – ESP32. Most of the code for maze navigation revolves around the HC-SR04 as for the planned solution to work the EEEBot only needs to detect the wall in front of itself while knowing its orientation is not vital to success. Another reason the planned solution overlooks the MPU data was due to its unreliability. The data from the MPU contains lots of noise especially in the z axis – the axis used. This meant that the recorded position of the EEEBot would deviate from the actual position to the point where it was unusable within seconds. This meant that without finely tuning the output or the logic to validate outputs, the data from the MPU could not be used. This resulted in the MPU data being completely neglected for the final code solution.

3.2 Implementation and Evaluation



```
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

duration = pulseIn(echoPin, HIGH);
distance = (duration * .0343) / 2;
if (distance <= 15.0) {
  analogWrite(sonar_led, 255);
  Wire.beginTransmission(slave_add);
  Wire.write(0);
  Wire.endTransmission();
} else {
  analogWrite(sonar_led, LOW);
  Wire.beginTransmission(slave_add);
  Wire.write(1);
  Wire.endTransmission();
}
```

Figure 12 - Master Sketch Code

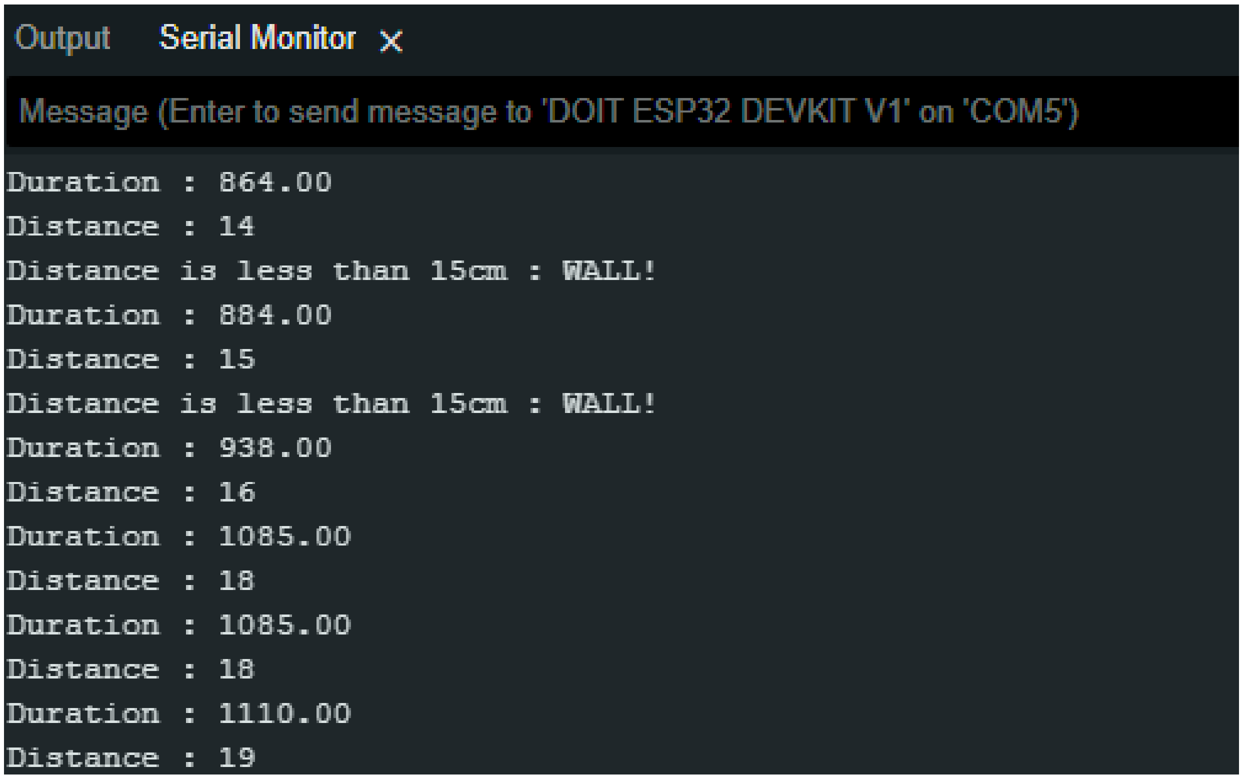


```
  if (clear) {
    goForwards();
    count = 0;
  } else if (count == 0) {
    stopMotors();
    TurnLeft(800);
    count += 1;
  } else {
    TurnAround();
    count = 0;
  }

}

void setClear(int bytes)
{
  clear = Wire.read();
  Serial.println(clear);
}
```

Figure 11 - Slave Sketch Code Snippet

Full code can be found in the appendix

To detect when the EEEBot is coming up to a wall, the HC-SR04 sensor is triggered and measured every 10 milliseconds and a value is sent from the master to slave ESP32 to indicate whether to continue forwards or to stop. Once a signal to stop is sent, the EEEBot stops and a count is incremented, showing that this is the first time the bot has run into a wall. The bot will first attempt to turn left by 90 degrees and detect again if there is an obstacle; if there is one the bot will then try to turn around – 180 degrees – and end up in a position that is equivalent to turning right in respect to its original position. It is better to validate its position using the gyroscope; however, this was overlooked as turning in a direction can be programmed far more easily than tuning the gyroscope to work within the time frame given.


Output
Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

```
Duration : 864.00
Distance : 14
Distance is less than 15cm : WALL!
Duration : 884.00
Distance : 15
Distance is less than 15cm : WALL!
Duration : 938.00
Distance : 16
Duration : 1085.00
Distance : 18
Duration : 1085.00
Distance : 18
Duration : 1110.00
Distance : 19
```

Figure 13 - Maze Navigation Serial Monitor Output

The maze navigation method shown above does have some flaws as it is very rudimentary. A major problem with the method is that it does not account for dead ends. When the EEEBot reaches a dead end, it will first try to turn left and face another dead end; then, it will try to U-turn and go right, at which point it will still be facing a wall. To fix this, add one more condition to the counter that if it reaches 3 – that if there is a wall after the U-turn – the EEEBot will turn right and return the way it came.

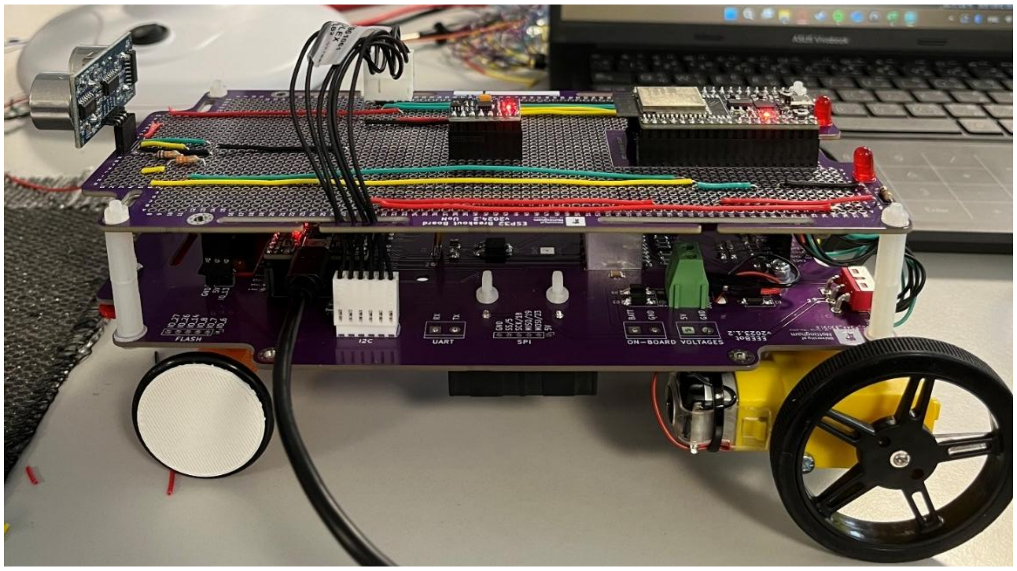


Figure 14 - EEEBot Maze Navigation Full Construction

4.0 Design and Implementation of Line Following Solution

The aim of the line following is for the vehicle to correct its own path once it detects that it has deviated from the path – black line on the floor. The EEEBot will detect a path using a series of optical sensors and use the information to give the necessary output in order to stay on the path.

4.1 Optical Sensors

The optical sensor used is a photodiode. The photodiode outputs current inversely proportional to the light level on it. This means that a high light level on the diode will output a high current. The photodiode is then connected to a transimpedance amplifier which takes a current input and gives a voltage output. The OP-AMP is needed as the output from the photodiode is very small and by amplifying the signal, it can then be more easily distinguished between a high and low signal level. The signal shown in Figure 17 has a gain value of 6.67 calculated using the gain equation for an inverting OP-AMP

Figure 15 - ZSPD053B-S40 Photo Diode [8]

$$Gain = -\frac{R_f}{R_i} \qquad \text{(2) from [6]}$$

Figure 16 - IR333C Infrared LED [9]

An infrared LED is placed next to the photodiode. As it is powered it emits an infrared light on the ground which is then reflected or absorbed depending on the color of the surface. The black line will absorb the light and result in a low current in the photodiode while the opposite is true for the white floor.

The output is then connected to the ESP32 through an analog digital converter (ADC) pin. The value of voltage level can then be read and used for line following with a range between 0 and 4096.

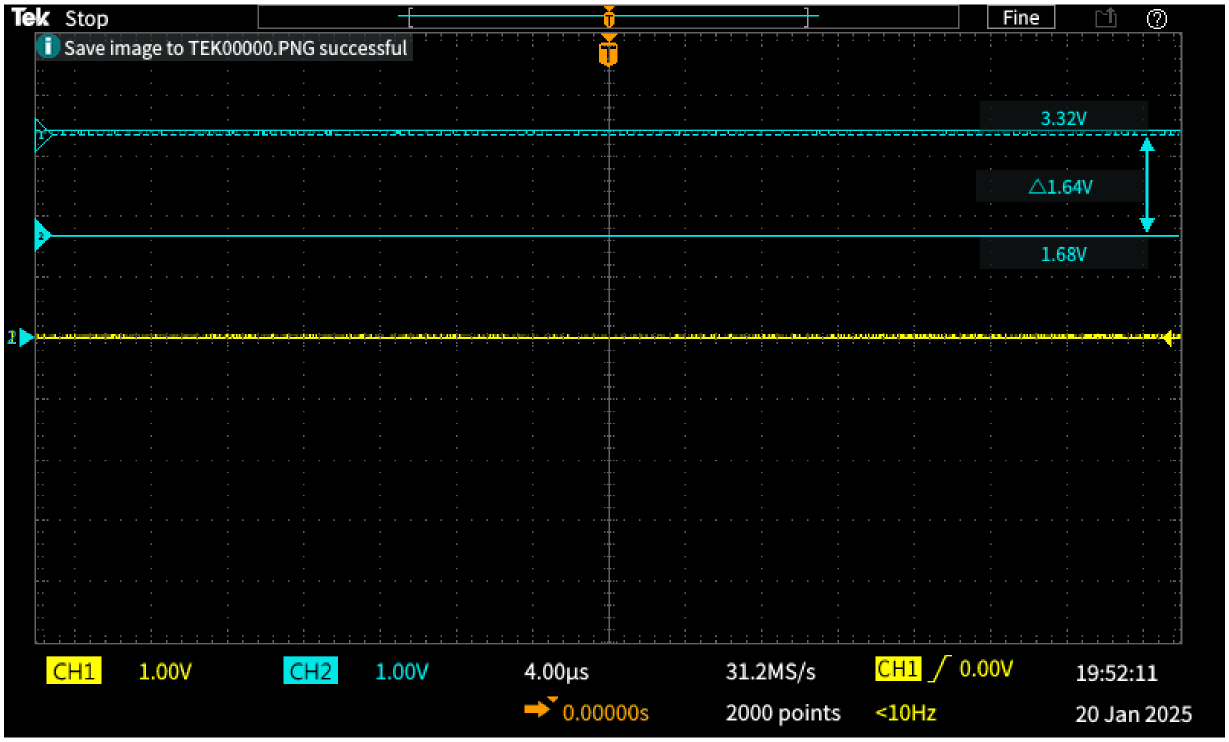


Figure 17 - Photo Diode Amplified Output

4.2 Proportional Integral Derivative (PID) Line Following

Proportional integral derivative or PID is a control algorithm that takes the error between a setpoint and the current point – measurement – to produce an output to correct the error. The error is calculated using

$$error = setpoint - weightedAverage \tag{3}$$

The weighted average is calculated using the equation

$$WeightedAverage = \frac{\sum_{i=0}^{N} Reading_i \times Weighting_i}{\sum_{i=0}^{N} Reading_i} \tag{4}$$

Where N is the number of readings – sensors. Refer to [7]

The PID algorithm uses three errors, the instantaneous error, the integration of the error and the differentiation of the error. The three values are then weighed using different constants to tune how much impact each value has on the output. The sum of these weighted error values then become the output to the hardware – primarily the servo – to correct the path of the EEEBot. The output from the error is scaled for different purposes where

$$servoAngle = centreAngle + output \tag{5}$$

$$leftMotorSpeed = baseSpeed + (K * output) \tag{6}$$

$$rightMotorSpeed = baseSpeed - (K * output) \tag{7}$$

The instantaneous error is the most important and so have the largest value of K – weighting constant. It is the main value that controls the output. The integral part of the error is the running sum of the error since it was last zero meaning that its effect should start small and increase over time as the error is not corrected. This means that the value of K for the integral must be very small as the integral will increase very quickly over time. The derivative of the error is the rate of change – how fast error is increasing; meaning that a quick change in error value requires quick action. The weighting for each error value is tweaked manually as the EEEBot is tested to give the fastest and most stable line following solution.

4.3 Construction

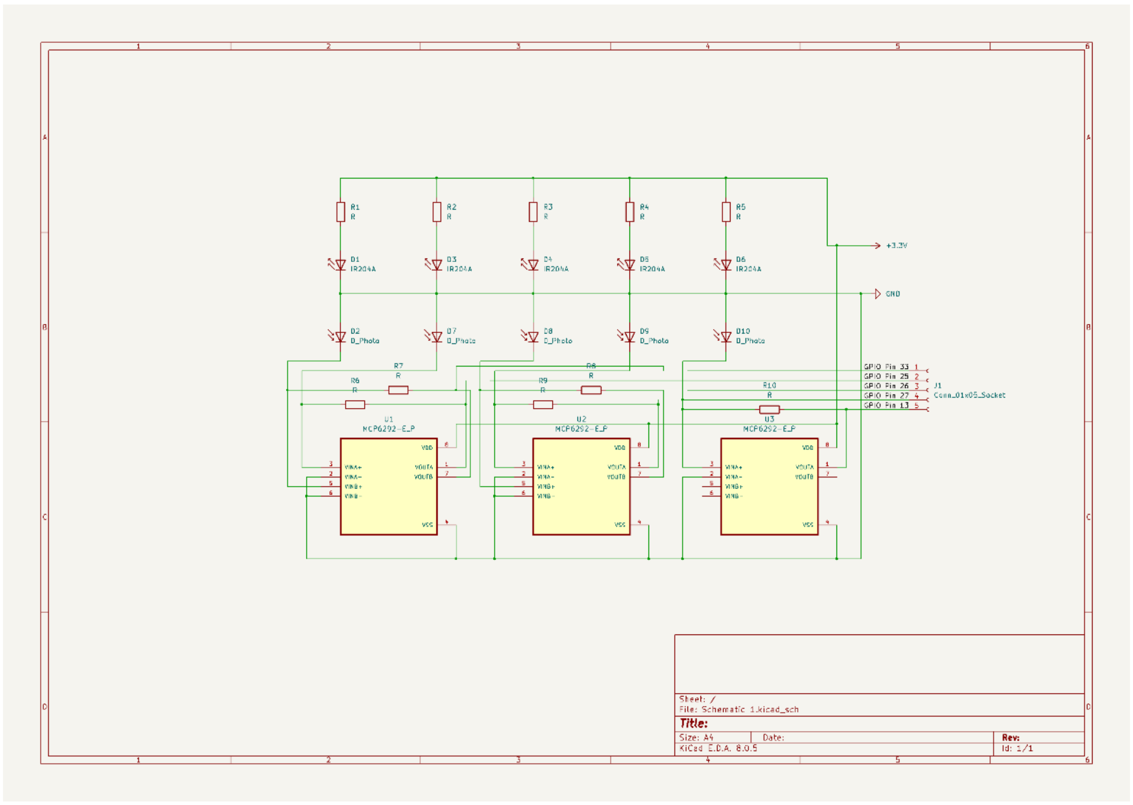


Figure 18 - EEEBot Line Following Stripboard Circuit Schematic

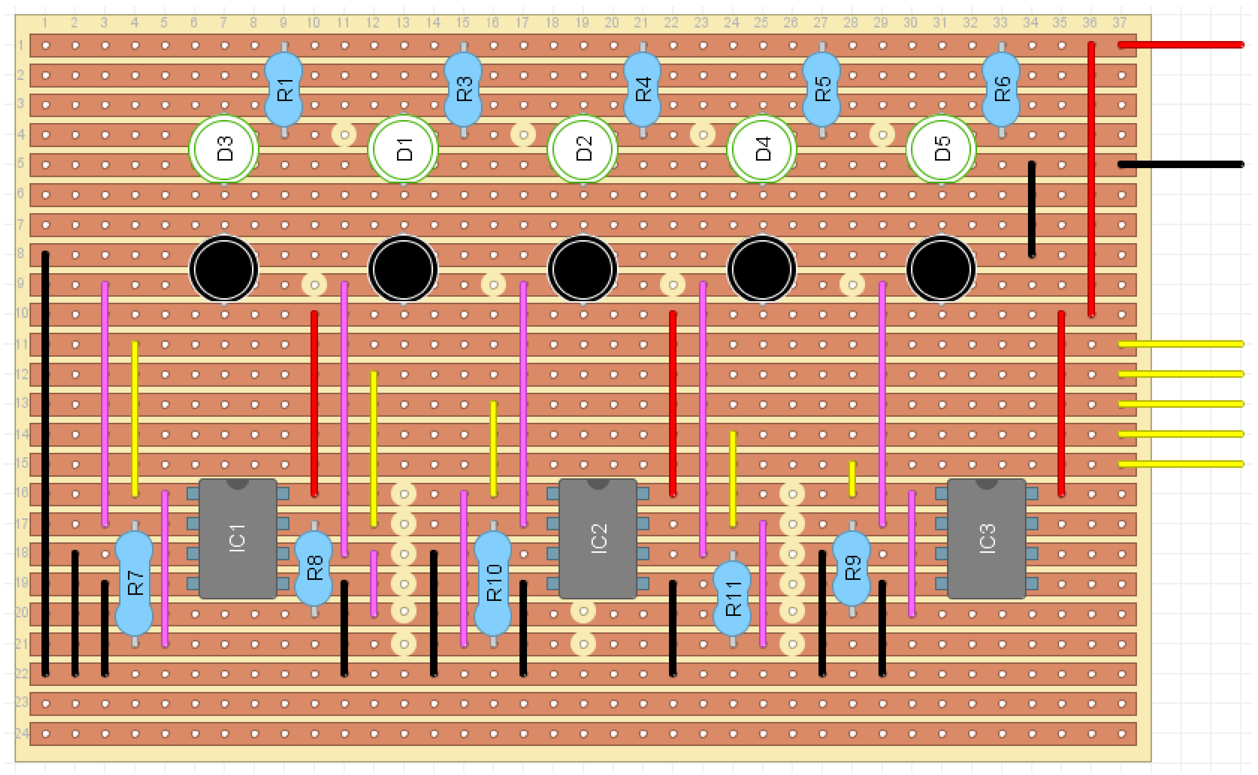


Figure 19 - Path Following Stripboard Plan

The photodiodes and IR LEDs as well as the amplifiers were all wired on a single stripboard so as to save space on the breakout board of the EEEBot. The stripboard was built according to the plan shown in Figure 19 which was made based on the circuit schematic shown in Figure 18. The resistance values of R1 to R5 in Figure 18 are 105Ω while R6 to R10 are 1.2MΩ. During testing, R6 to R10 were planned to be 10MΩ, however, they proved to be too sensitive and so were changed for the final design. More is explained in section 4.4 Programming.

The wires leading off to the side were then connected to the breakout board using the plan shown in Figure 20 with the green wires being the new lines to connect the amplifier output to the ESP32.

The pins used are GPIO 33, 25, 26, 27 and 13 as they had ADC capabilities. This allowed us to read the digital value of the output from the amplifier in the code editor and use it to calculate error.

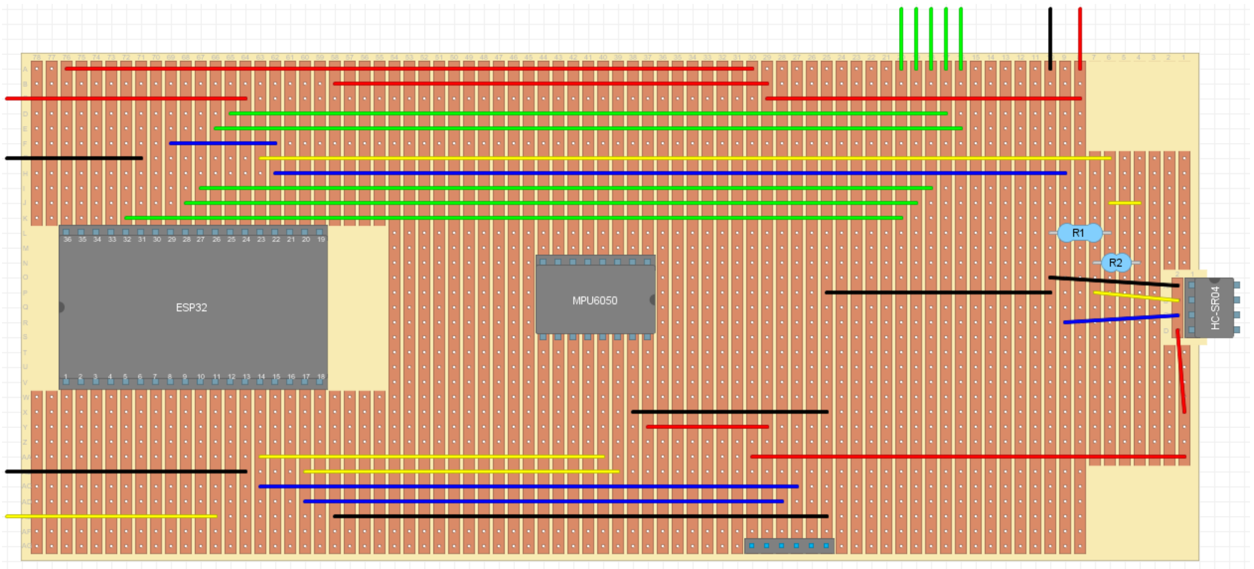
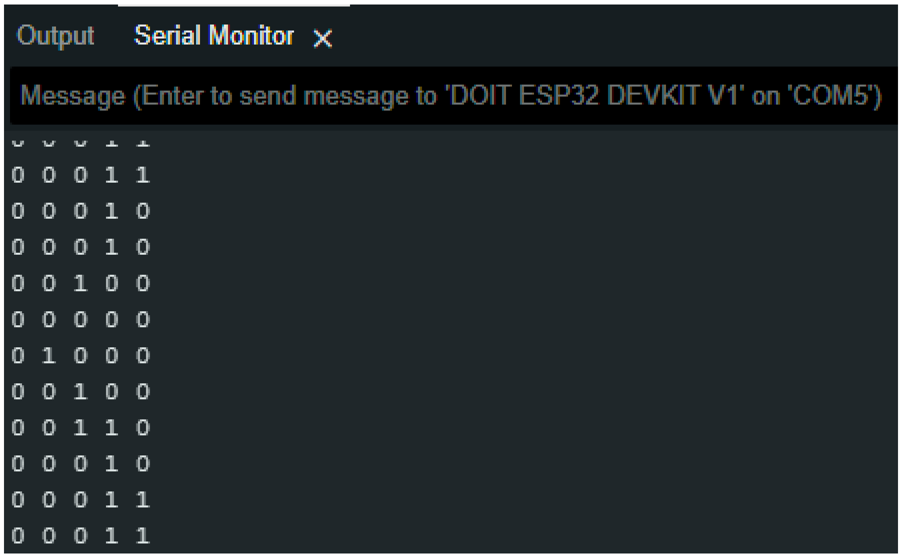


Figure 20 - Line Following Breakout Board Plan

4.4 Programming

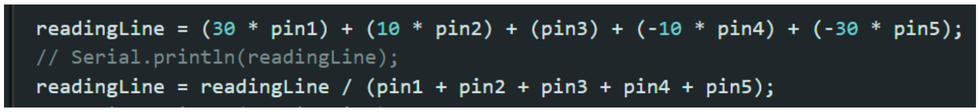
The programming is separated into 2 main parts, the simple line following and PID line following. The first is simple line following. The method used is to read the data of the photo diodes from the amplifier and then run it through a function that determines a 1 or 0 with 1 meaning the line is detected. The boundaries for determining this are manually set and can be tuned. The values are then multiplied by 2^n from n = 0 to 4 and add them together to generate a 5-bit series to indicate which photo diode is on. For example, 4 = 00100 meaning the bot is in the middle or 24 = 11000 meaning the EEEBot is to the right of the line. This value is then sent to the downstairs ESP32 from the upstairs ESP32 to be processed. Once the value is read, the change in servo angle is assigned using switch cases. This method worked decently well as it completed half of the course before failing.



```
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')
0 0 0 1 1
0 0 0 1 0
0 0 0 1 0
0 0 1 0 0
0 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 1 1 0
0 0 0 1 0
0 0 0 1 1
0 0 0 1 1
```

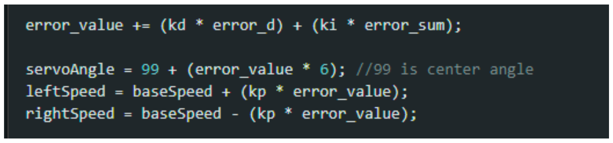
Figure 21 - Simple Line Following Serial Monitor Output

For the second part, PID line following, first the value from the amplifier is read through the GPIO pins on the upstairs – master – ESP32. The values are processed by multiplying by their respective weightings and summed then divided by the unweighted sum to produce a weighted average. The weighted average value is then sent to the downstairs ESP32 as an integer. The downstairs ESP32 then reads the values and calculates the error. The different error values are then derived from the value read which then produces an output to the servo and motors. The hardware components affected are then written to by the downstairs ESP32. The full code can be found in the appendix



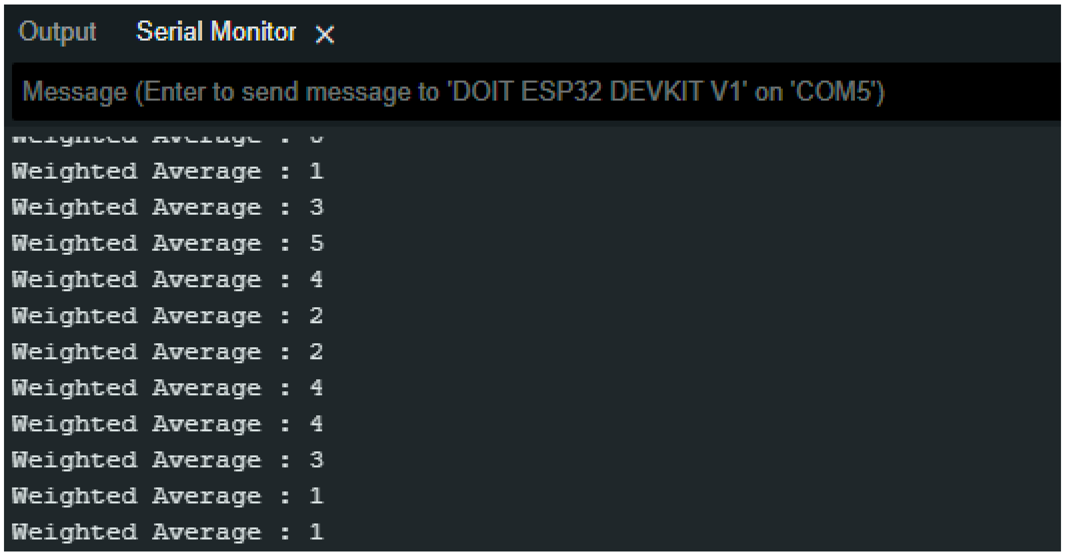
```
readingLine = (30 * pin1) + (10 * pin2) + (pin3) + (-10 * pin4) + (-30 * pin5);
// Serial.println(readingLine);
readingLine = readingLine / (pin1 + pin2 + pin3 + pin4 + pin5);
```

Figure 22 - Line Following Upstairs Code Snippet



```
error_value += (kd * error_d) + (ki * error_sum);

servoAngle = 99 + (error_value * 6); //99 is center angle
leftSpeed = baseSpeed + (kp * error_value);
rightSpeed = baseSpeed - (kp * error_value);
```

Figure 23 - Line Following Downstairs Code Snippet



Output
Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

```
Weighted Average : 1
Weighted Average : 3
Weighted Average : 5
Weighted Average : 4
Weighted Average : 2
Weighted Average : 2
Weighted Average : 4
Weighted Average : 4
Weighted Average : 3
Weighted Average : 1
Weighted Average : 1
```

Figure 24 - Weighted Average Serial Monitor Output

4.5 Result and Conclusion

The line following worked as intended and followed the test circuit to completion. There were some jittering due to the sensitivity of the servo when turning left as it turned far further than the specified angle unlike when turning right. This meant that when correcting itself by turning left – the EEEBot was to the right of the line – the EEEBot would often overshoot and so would need to correct again by turning right. The constants for each kind of error can be tweaked to be more fine-tuned as well as the weightings for each sensor however it currently accomplishes the task it is designed for and there is no need to change them as of yet. On the other hand, the method was not tested for a 90° turn or sharper meaning it may not work well with such. This means the method will need further testing on the EEEBot and see if it will work well with the aforementioned conditions. Another method to help optimize the line following is to implement linear regression or another form of simple machine learning to find the best values for the constants.

5.0 Conclusion

In the two lab weeks covered in this lab report, we had set out to add sensors to the EEEBot to give it the capabilities to complete two tasks, maze navigation and line following. As this report illustrates, the goal set out was accomplished, however, not without setbacks.

In the first lab week covered – lab week 4 – the HC-SR04 sonar sensor was added and integrated into the EEEBot giving it the ability to detect the surroundings. The main issue encountered when integrating the sonar was connecting it to the ESP32 to read its output digitally. The initial plan was to connect it to GPIO pins 34 and 35. However, it was later discovered that GPIO pin 34 was input only, can only read information, while it needed to be used as an output. This issue was quickly solved by changing the pin used. Next, the MPU6050 gyroscope was implemented into the EEEBot. This allowed the EEEBot to record its current orientation based on its angular movements. The main issue with this is that the MPU is overly sensitive to movements and can register very minor disturbance – noise – as movement, making the data from it unusable without proper tuning. This meant that when trying to implement the code solution for maze navigation, the MPU data was neglected despite the potential to be very useful as a check. Overall, despite the lack of reliable MPU data, the maze navigation was a success.

In the second lab week – lab week 5 – an optical sensor in the form of a photo diode was implemented for line following. For implementing the photo diode as well as the infrared LED used to emit light for the photo diode to pick up, there were no issues. The main concern was the resistor values used to determine the gain values. Initially, the gain value used was 10MΩ. This worked well during breadboard testing as it gave distinct values between a light and dark surface. However, when used on a real floor where the light levels were not constant, the gain value made it so that the readings were overly sensitive and unusable. The first solution tried did not go as planned, however, the second did. The second solution was to use a smaller gain value and so changing the resistors from 10MΩ to 1.2MΩ. This solved the issue. Another hurdle to cross was the layout of the stripboard. This was due to the stripboard being relatively small and so the wiring must be efficient. This was also solved easily. The last hurdle was the code implementation itself. The PID solution proved to be difficult to implement as it clashed with my initial solution for simple line following. However, this was solved resulting in a line following solution that worked out well all through the course.

In conclusion, the two project weeks gave minimal resistance in general with only a small number of setbacks that took some time to overcome while most issues were fixed promptly.

References

[1] jazhe, "ElectronicCats/mpu6050," 18 12 2024. [Online]. Available: https://github.com/ElectronicCats/mpu6050. [Accessed 18 01 2025].

[2] Arduino, "Inter-Integrated Circuit (I2C) Protocol," 20 09 2024. [Online]. Available: https://docs.arduino.cc/learn/communication/wire/. [Accessed 24 02 2025].

[3] 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].

[4] amponitor, "Ultrasonic Distance Measurement HC-SR04," 01 11 2024. [Online]. Available: https://apmonitor.com/dde/index.php/Main/UltrasonicDistanceSensor. [Accessed 17 01 2025].

[5] lastminuteengineers, "ESP32 Pinout Reference," [Online]. Available: https://lastminuteengineers.com/esp32-pinout-reference/.

[6] Indian Institute of Technology Roorkee, "INVERTING AND NON-INVERTING AMPLIFIERS USING OP AMPS," [Online]. Available: https://iitr.ac.in/Academics/static/Department/Physics/Analog%20Electronics/Operational_Amplifier.pdf.

[7] Department of Electrical & Electronic Engineering, "EEEE1002: Optical Sensors & Optical Line Following," 2024.

[8] CHAULIGHT, "2306261518_Chau-Light-ZSPD053B-S40_C5337502 IR Receiver," 2021.

[9] EVERLIGHT, "Technical Data Sheet 5mm Infrared LED , T-1," 2005.

[10] S. Campbell, "Basics of the I2C Communication Protocol," [Online]. Available: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/.

Appendix

Week 4 Slave Sketch- Maze Navigation

```
#include <Wire.h>
#define slave_add 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(slave_add);
    Wire.onReceive(setClear);
    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 clear = 1; // if there is no obstacle in front clear = 1
int count;

void loop()
{

    int leftSpeed = 255;
    int rightSpeed = 255;
    steeringServo.write(99);
    motors(leftSpeed, rightSpeed);
    if (clear)
    {
        goForwards();
        count = 0;
    }
    else if (count == 0)
    {
        stopMotors();
        TurnLeft(800);
        count += 1;
    }
    else
    {
        TurnAround();
        count = 0;
    }
}
```

```
void setClear(int bytes)
{
    clear = Wire.read();
    Serial.println(clear);
}

void moveSteering()
{
    steeringServo.write(0);

    for (steeringAngle = 0; steeringAngle <= 180; steeringAngle += 2)
    {                                       // goes from 0 degrees to 180 degrees
in steps of 1 degree
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    for (steeringAngle = 180; steeringAngle >= 0; steeringAngle -= 2)
    {                                       // 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 ResetSteering()
{
    steeringServo.write(90);
    delay(200);

    for (steeringAngle = 90; steeringAngle > 0; 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
    }
    delay(500);
    for (steeringAngle = 0; steeringAngle <= 200; steeringAngle += 1)
    {                                       // goes from 0 degrees to 180 degrees
in steps of 1 degree
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
```

```
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    delay(500);
    for (steeringAngle = 200; 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 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);
}
```

Week 4 Master Sketch - Maze navigation

```
// initialize for communication

#include <Wire.h>
#include <math.h>
#define slave_add 0x04

// Defining values for the sonar

#define trigPin 14
#define echoPin 35
#define sonar_led 32

float duration;
int distance;

// Defining values for 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;

Adafruit_MPU6050 mpu;

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

    // set pins

    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    pinMode(sonar_led, OUTPUT);
    pinMode(gyro_led, OUTPUT);
    Serial.begin(115200);

    // setup for gyroscope

    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();

    delay(100);
}

void loop()
{
    // put your main code here, to run repeatedly:

    // Sonar code
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);
    distance = (duration * .0343) / 2;
    if (distance <= 15.0)
    {
        analogWrite(sonar_led, 255);
        Wire.beginTransmission(slave_add);
        Wire.write(0);
        Wire.endTransmission();
    }
    else
    {
        analogWrite(sonar_led, LOW);
        Wire.beginTransmission(slave_add);
        Wire.write(1);
        Wire.endTransmission();
    }

    // Serial.print("\n\nDuration : ");
    // Serial.print(duration);
    // Serial.print("\nDistance : ");
    // Serial.print(distance);

    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;

    Serial.print("Rotation X: ");
    Serial.print(x);
    Serial.print(", Y: ");
    Serial.print(y);
    Serial.print(", Z: ");
    Serial.print(z);
    Serial.println(" Degs");

    if (((z >= 85) && (z <= 95)) || ((z >= -95) && (z <= -85)))
    {
        digitalWrite(gyro_led, HIGH);
    }
    else
    {
        digitalWrite(gyro_led, LOW);
    }

    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);
}
```

Week 5 Downstairs Code Sketch - Line Following

```
#include <Wire.h>
#define slave_add 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

// line following setup
int lineChecked;
#define baseSpeed 180
int old_error_value = 0, error_sum = 0, error_value = 0;

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

    // i2C com set up
    Wire.begin(slave_add);
    Wire.onReceive(callOnReceive);
    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 clear = 1; // if there is no obstacle in front clear = 1
int count;
float leftSpeed = baseSpeed;
float rightSpeed = baseSpeed;

void loop()
{

    motors(leftSpeed, rightSpeed);
    if (clear)
    {
        goForwards();
        count = 0;
    }
    else if (count == 0)
    {
        stopMotors();
        steeringServo.write(99);
        TurnLeft(800);
        count += 1;
    }
    else
    {
        steeringServo.write(99);
        TurnAround();
        count = 0;
    }
    delay(20);
}

void callOnReceive(int bytes)
{
    int valuesRead;
    valuesRead = Wire.read();
    // Serial.println(valuesRead);
    // setClear(valuesRead & 0b100000);
    followLinePID(valuesRead);
}

void followLine(int valuesRead)
{
    lineChecked = valuesRead;
    // Serial.println(lineChecked);
    switch (lineChecked)
    {

    case 0b00100:
        // Serial.println("Center");
        turn_steering(0);
        break;

    case 0b00110:
        // Serial.println("Slight Left");
        turn_steering(-10);
        break;
    case 0b00010:
        // Serial.println("Medium Left");
        turn_steering(-20);
        break;
    case 0b00011:
        // Serial.println("Medium Far Left");
        turn_steering(-30);
        break;
    case 0b00001:
        // Serial.println("Far Left");
        turn_steering(-40);
        break;

    case 0b01100:
        // Serial.println("Slight Right");
        turn_steering(10);
        break;
    case 0b01000:
        // Serial.println("Medium Right");
        turn_steering(15);
        break;
    case 0b11000:
        // Serial.println("Medium Far Right");
        turn_steering(20);
        break;
    case 0b10000:
        // Serial.println("Far Right");
        turn_steering(35);
        break;

    default:
        break;
    }
}

void turn_steering(int steeringAngle)
{
    steeringServo.write(99 + steeringAngle);
    leftSpeed += 0.4 * steeringAngle;
    rightSpeed += -0.4 * steeringAngle;
    motors(leftSpeed, rightSpeed);
}

void followLinePID(int valuesRead)
{
    int8_t weightedAverage = valuesRead;
    int error_d;

    float kp = 0.3, kd = 0.3, ki = 0.01;
    float servoAngle;

    error_value = weightedAverage - 1;
    if (error_value == 0)
    {
        error_sum = 0;
    }
    else
    {
        error_sum += old_error_value;
    }

    error_d = old_error_value - error_value;

    Serial.println(error_value);

    error_value += (kd * error_d) + (ki * error_sum);

    servoAngle = 99 + (error_value * 6); // 99 is center angle
    leftSpeed = baseSpeed + (kp * error_value);
    rightSpeed = baseSpeed - (kp * error_value);

    // Serial.println(error_value);
    // Serial.println(servoAngle);

    steeringServo.write(servoAngle);
    motors(leftSpeed, rightSpeed);
    old_error_value = error_value;
}

void setClear(int valuesRead)
{
    clear = valuesRead >> 5;
    Serial.println(clear);
}

void moveSteering()
{
    steeringServo.write(0);

    for (steeringAngle = 0; steeringAngle <= 180; steeringAngle += 2)
    {                                       // goes from 0 degrees to 180 degrees
in steps of 1 degree
        steeringServo.write(steeringAngle); // tell servo to go to position in
variable 'steeringAngle'
        delay(15);                          // waits 15ms for the servo to reach
the position
    }
    for (steeringAngle = 180; steeringAngle >= 0; steeringAngle -= 2)
    {                                       // 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 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);
}
```

Week 5 Upstairs Sketch – Line Following

```
// initialize for communication

#include <Wire.h>
#include <math.h>
#define slave_add 0x04

// Defining values for the sonar

#define trigPin 14
#define echoPin 35
#define sonar_led 32

float duration;
int distance;

// Defining values for 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;

Adafruit_MPU6050 mpu;

// Defining values for line following
#define farLeft 13
#define closeLeft 27
#define Middle 26
#define closeRight 33
#define farRight 25

#define Threshold 1750
int pin1, pin2, pin3, pin4, pin5;
int readingLine;

int clear_ahead;

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

    // Line following setup

    pinMode(farLeft, INPUT);
    pinMode(closeLeft, INPUT);
    pinMode(Middle, INPUT);
    pinMode(closeRight, INPUT);
    pinMode(farRight, INPUT);

    // Sonar setup

    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);

    // LED pins

    pinMode(sonar_led, OUTPUT);
    pinMode(gyro_led, OUTPUT);
    Serial.begin(115200);

    // setup for gyroscope

    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();

    delay(100);
}

void loop()
{
```

```
// put your main code here, to run repeatedly:

// Sonar code

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

duration = pulseIn(echoPin, HIGH);
distance = (duration * .0343) / 2;
// sonar logic code - tells downstairs board if bot is close to wall
if (distance <= 20.0)
{
    analogWrite(sonar_led, 255);
    clear_ahead = 0;
}
else
{
    analogWrite(sonar_led, LOW);
    clear_ahead = 32;
}

// Sonar testing prints
// Serial.print("\n\nDuration : ");
// Serial.print(duration);
// Serial.print("\nDistance : ");
// Serial.print(distance);

// path following
check_path_PID();

// Gyroscope code

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;

Gyroscope testing prints
    Serial.print("Rotation X: ");
Serial.print(x);
```

```
    Serial.print(", Y: ");
    Serial.print(y);
    Serial.print(", Z: ");
    Serial.print(z);
    Serial.println(" Degs");

    // gyroscope LED logic

    if (((z >= 85) && (z <= 95)) || ((z >= -95) && (z <= -85)))
    {
        digitalWrite(gyro_led, HIGH);
    }
    else
    {
        digitalWrite(gyro_led, LOW);
    }

    delay(20);
}

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);
}
```

```
int check_threshold(int pin)
{
    if (pin < Threshold)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void check_path_PID()
{
    pin1 = analogRead(farLeft);
    pin2 = analogRead(closeLeft);
    pin3 = analogRead(Middle);
    pin4 = analogRead(closeRight);
    pin5 = analogRead(farRight);

    readingLine = (30 * pin1) + (10 * pin2) + (pin3) + (-10 * pin4) + (-30 *
pin5);
    // Serial.println(readingLine);
    readingLine = readingLine / (pin1 + pin2 + pin3 + pin4 + pin5);
    // Serial.println(readingLine);
    Wire.beginTransmission(slave_add);
    Wire.write(readingLine);
    Wire.endTransmission();
}

void check_path(int clear_path)
{
    pin1 = check_threshold(analogRead(farLeft)) * 1;
    pin2 = check_threshold(analogRead(closeLeft)) * 2;
    pin3 = check_threshold(analogRead(Middle)) * 4;
    pin4 = check_threshold(analogRead(closeRight)) * 8;
    pin5 = check_threshold(analogRead(farRight)) * 16;

    readingLine = pin1 + pin2 + pin3 + pin4 + pin5 + clear_path;

    Serial.print(pin1);
    Serial.print(" ");
    Serial.print(pin2);
    Serial.print(" ");
    Serial.print(pin3);
    Serial.print(" ");
    Serial.print(pin4);
    Serial.print(" ");
    Serial.println(pin5);
    Serial.println(analogRead(closeRight));

    Wire.beginTransmission(slave_add);
    Wire.write(readingLine);
    Wire.endTransmission();
}
```