University of Nottingham
UK | CHINA | MALAYSIA

Title: Maze Navigation & Line Following

Applied Electrical and Electronic Engineering: Construction Project (EEEE1002 UNUK) (FYR1 24-25): Technical Report 2

Date of Submission: 12/02/25

Abstract

The core of autonomous vehicles in the capability to attain movement without human intervention. This lack of human intervention provides immense efficiency benefits not just for industries but for the general public. The contents of this paper demonstrate the implementation of sensors in order to truly produce a vehicle capable of driving without the aid of external control. To showcase such abilities, I've demonstrated my attempted approaches to two challenges. The first one is a maze in which my vehicle should be able to traverse on its own through avoiding obstacles via the uses of sensors. The other is a line following challenge in which my vehicle should be able to follow a set path completely independently. Though my attempts in both these challenges may not have gotten the results I may have wished my approach to these challenges should convey the idea of higher levels of automation and the possible uses in industry.

Table of Contents

Contents

Introduction

The basis of the implementation of these challenges is the utilisation of sensors. For the maze navigation challenges it hinges on two specific sensors. An ultra-sound sensor specifically the HC-SR04 and an IMU specifically the MPU-6050. Ultra-Sound sensors provide the ability for the vehicle to be able to sense incoming objects by not only emitting ultrasound waves but also measuring the reflected ultrasound waves to know the distances between the vehicle and objects. An IMU (Intertia Measurement Unit) can measure the speed but also the rotational angle of the vehicle therefore allowing the vehicle to know its direction and move according to the software implementation. As for the line following challenge it hinges on photodiodes that generate a current depending on the changes in light intensity. Since the line in this case is black with a white background, there would be a change in measured light intensity thus causing a response via a change in current produced by the photodiode which when amplified and set a logic level can cause a response by the vehicle.

1. Design & Implementation of additional sensor subsystems

This section delves into the designs of the two main sensors used for the maze navigation challenge. As mentioned prior the sensors I will be using is the HC-SRO4 ultrasound sensor and the MPU-6050 IMU. The following section describes both the hardware and software designs required to integrate these sensors into the EEEbot.

1.1 Ultrasound Subsystem

1.1.1 Principle of Operation

The basic concept of an ultrasound sensor is the emission and subsequent absorption of ultrasound waves. The main function of the ultrasound sensor is to emit an ultrasound wave that travels a certain distance before being reflected. In which case, the ultrasound sensor absorbs the reflected wave and calculates the distance travelled via the time it takes between emission and absorption. The benefits of ultrasound sensors are its accuracy of up to 3mm but also the fact that due to its high frequency of over 20kHz it is inaudible to humans therefore it won't be a source of distraction or distress for people around it. However, it is range is rather limited from 2cm to around 400cm. In this specific use case, a limited range such as this is suitable however in more practical use cases where measurements may need to be taken in the metres or kilometres an ultrasound sensor may not be the most optical sensors. The way distance is calculated by the ultrasound sensor is by using the equation.

$$S = D/T$$

S in this case being the speed of sound which in dry air at 20˚ is 343 m/s. T is the time it takes for the ultrasound wave to reflect back to the sensor and be detected. By rearranging the equation for distance, we get the equation.

$$D = S \times T$$

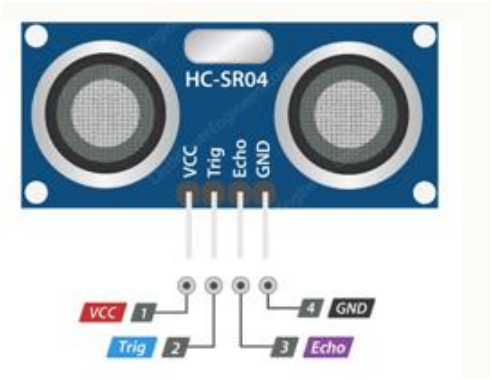


Figure 1: HC-SR04 Schematic

The ultrasound sensor I will be using for this case will be the HC-SR04. The reason I opted for the HC-SR04 is for its simple four wire interface and its ability to be easily integrated into the Arduino IDE software platform allowing for enhanced control over the sensor. As seen on Figure 1: HC-SR04 Schematic the HC-SR04 has four pins: Vcc, GND, Trig, Echo. Vcc is the power input and should be set to 5V as the HC-SR04 is a 5V device. The trig and echo pins are controlled by the ESP32 and set a logic level. The trig is set at HIGH for 10µs to send a pulse. The echo pin is set at HIGH with the pulse and upon the pulse being emitted be set to LOW.

1.1.2 Schematic Design & Simulation

1.1.2.1 Voltage Corrections

In order to first connect the HR-SR04 to the ESP32 some changes to voltages must be made first to prevent any damage to the components. The ESP32 is a 3.3V device whereas the HR-SR04 is a 5V device. As such, the voltage must be stepped down before connecting the two components together.

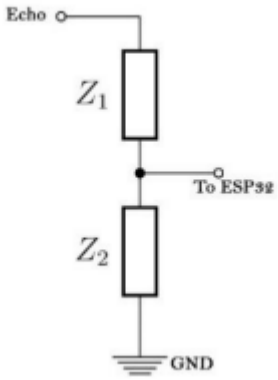


Figure 2:Voltage Divider Schematic

This is a basic schematic detailing what would need to be done to step down the voltage. By using a voltage divider circuit, we can split the 5V input and divert 3.3V to the ESP32 and the rest to the R2 resistor. The equation to calculate the suitable resistors would be:

$$\frac{Vout}{Vin} = \frac{R1}{R1 + R2}$$

Since we know the input is 5V and the desired output is 3.3V we can rearrange the equation to get:

$$R1 = \frac{33}{17} R2$$

Therefore since 33/17 is approximately 2 we can set R1 as 1kΩ and R2 as 2kΩ.

1.1.2.2 Sensor and ESP32 Integration

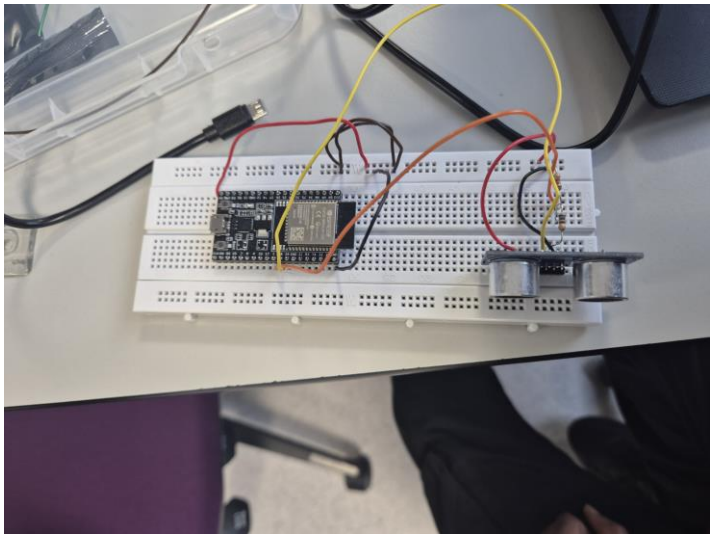


Figure 3:Initialisation sensor design

Figure 3 shows the first breadboard design to initialise the HR-SR04 and get some readings to make sure the sensor works as intended.

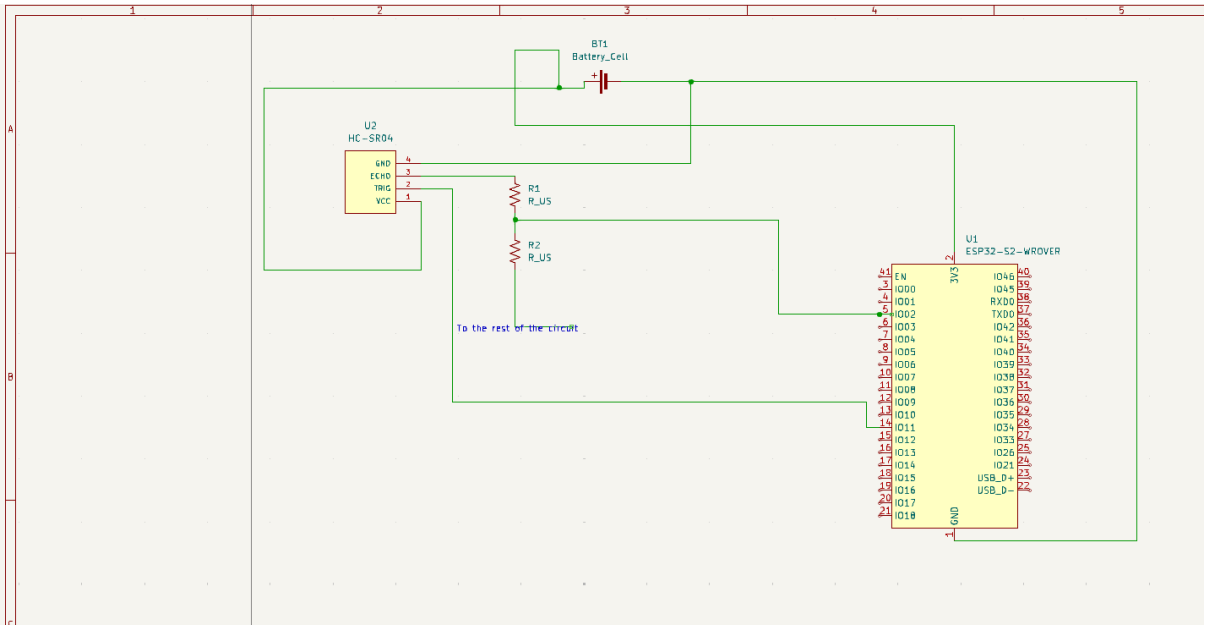


Figure 4:Sensor initialisation schematic

Figure 4 showcases in more detail the correct connections between the ESP32 and the HR-SR04. The trigger pin and the echo pin are connected to the GPIO pins 13 and 5 respectively which would allow for the esp32 to control the sensor via software. As mentioned in 1.1.2.1 Voltage Corrections

a voltage divider circuit is required between the echo pin and the esp32 to prevent a higher voltage being inputted into the esp32.

```
#include <HCSR04.h>

const byte triggerPin = 18;

const byte echoPin = 5;

UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);

void setup () {

  Serial.begin(9600);

}

void loop () {

  float distance = distanceSensor.measureDistanceCm();

  Serial.println(distance);

  delay(500);

}
```

This simple software is used to initialise the HC-SR04. Prior to the void setup, the trigger and echo pins are defined, and the distance sensor is set to those two pins. Within the void setup the connection is initialised. In the void loop, a float variable distance is set to the distance sensor measurements and every 500ms this distance value in cm is measured and then printed out. In order to further demonstrate the capabilities of the HR-SR04 they plotted the distances measured on a graph in order to ensure that the HR-SR04 was able to detect objects that was either far away or close to it. The way they went about this was taking an object such a box and moving it close to the sensor for 1-2 seconds then slowly moving the box away from the sensor every second.

Figure 5:HR-SR04 Initialisation Code

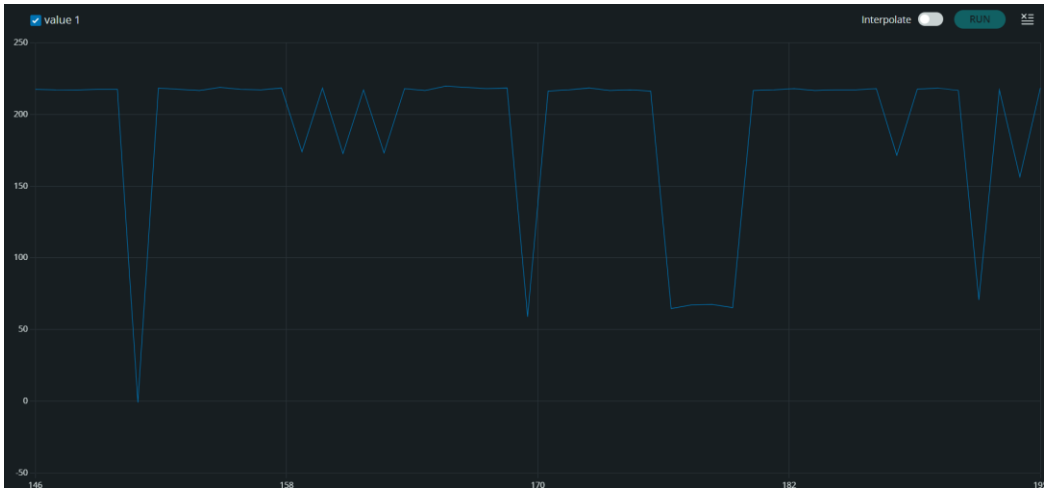


Figure 6:Arduino Sensor Output

As displayed in Figure 6 the sensor was able to detect objects from a range of distances. The sudden dip in the graph showcases the box moving away from the sensor and the quick rise in the graph shows the box moving closer to the sensor. What Figure 6 also displays is the sensitivity of the sensor. The sensitivity of the sensor is defined as the slope of the output characteristic curve (DY/DX in Figure 1) or, more generally, the *minimum input of physical parameter that will create a detectable output change*[1]. As the objects is moving towards and away from the sensor the response time is extremely quick (response time being the difference in time between a change in input and a change in the output). While generally there is a direct relationship between sensitivity and response time as a more sensitive sensor will react to changes in input quicker thus leading to higher response times however this depends on the task. Tasks that are more geared towards special navigation actually see an inverse relationship between sensitivity and response times, whereas tasks like the function currently being tested on the HR-SR04 that are geared towards taking measurements of exact distances see the general direct relationship between sensitivity and response times[2].

1.1.2.3 Testing Sensor Responses to specific Ranges

During the maze navigation challenge the vehicle must be able to understand when it's getting too close to an obstacle and move accordingly as a result software must be used so the sensor reacts to a certain range of measurements. In practical uses this range will be 10cm to 100cm however during the testing face the range was decreased from 1cm to 10cm to make the output far clearly. The approach in this case was to drive an LED and have it been at its brightest at 1cm and be completely off at 10cm.

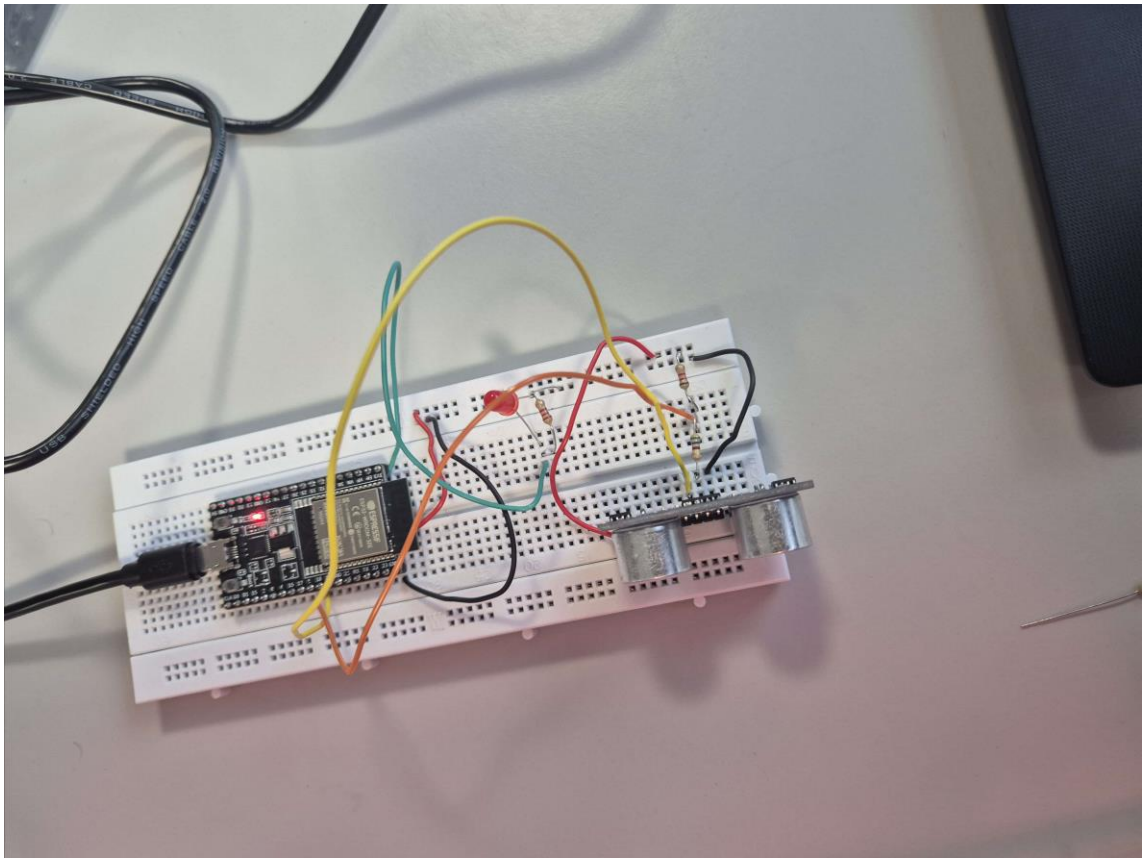


Figure 7:Breadboard LED Layout

By using the same general layout used in Figure 3:Initialisation sensor design to initialise the sensor an LED can be added to the circuit and controlled by the ESP32 to respond to changes in the sensor's measurements.

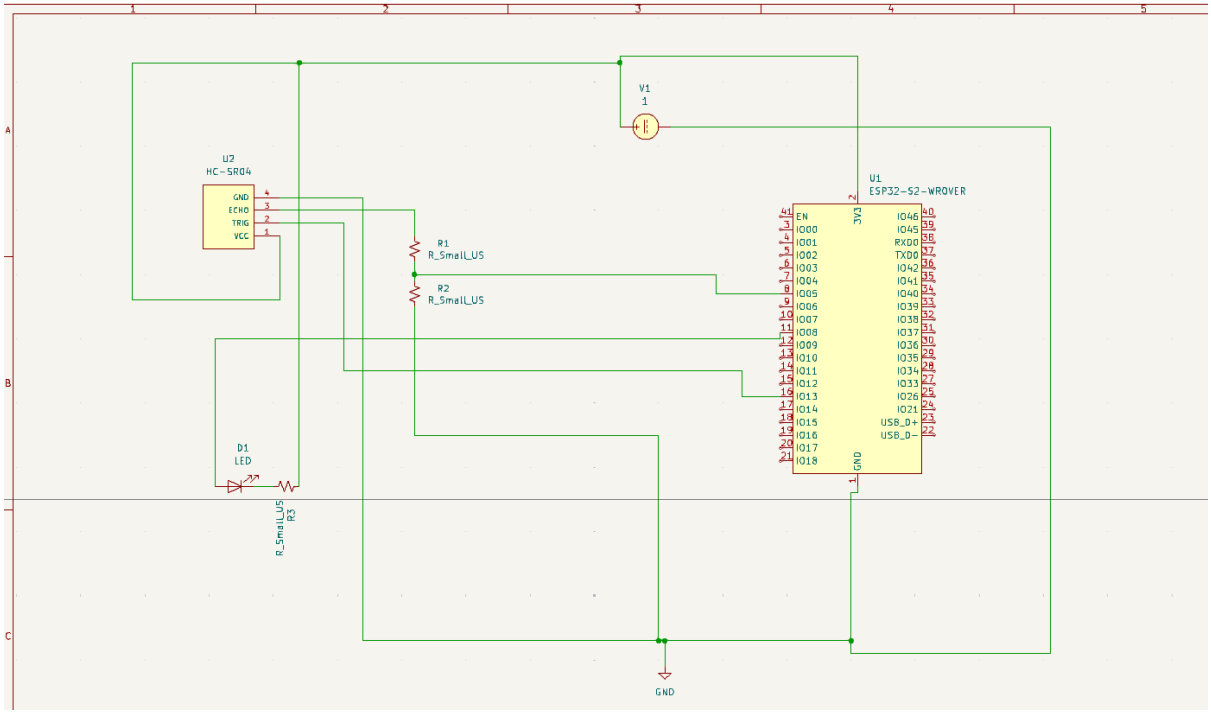


Figure 8:LED Schematic

As seen on the schematic the LED is connected to the GPIO pins of the ESP32 enabling control over the brightness of the LED. Through the usage of software, we can map the brightness of the LED to the distance measurement of the HC-SR04. This is done via manipulating the duty cycle of the LED, duty cycle being defined as the ratio of time a device or system is in an ON or OFF state. For instance, an LED having a 50% duty cycle would mean that the LED is on 50% of the time. However, due to the speed in which the LED switches from an ON or OFF state the LED appears to be dimmer.

```
int duty = map(distance, 10, 1, 0, 255);
  analogWrite(ledPin, duty);
```

The main function needed to dim the LED is the map function. The map() function remaps a number from one range to another. That is, a value of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between[3]. For this context, the distance value of 10 is set to a PWM value of 0 and the distance value of 1 is set to a PWM value of 255. PWM in this case is the way in which the duty cycle of the LED is controlled with 255 being the brightest at a 100% duty cycle and 0 being the OFF state of the LED. The reasoning for the use of PWM (Pulse Width Modulation) is due to its ability to precisely control the power delivered to different devices. It involves rapidly switching the power supply on and off at varying intervals while maintaining a constant voltage level. “In PWM, the on and off switching of the power supply occurs in cycles. Each cycle consists of a period and a duty cycle. The period represents the total time of one cycle, while the duty cycle represents the duration during which the power supply is on.”[4]

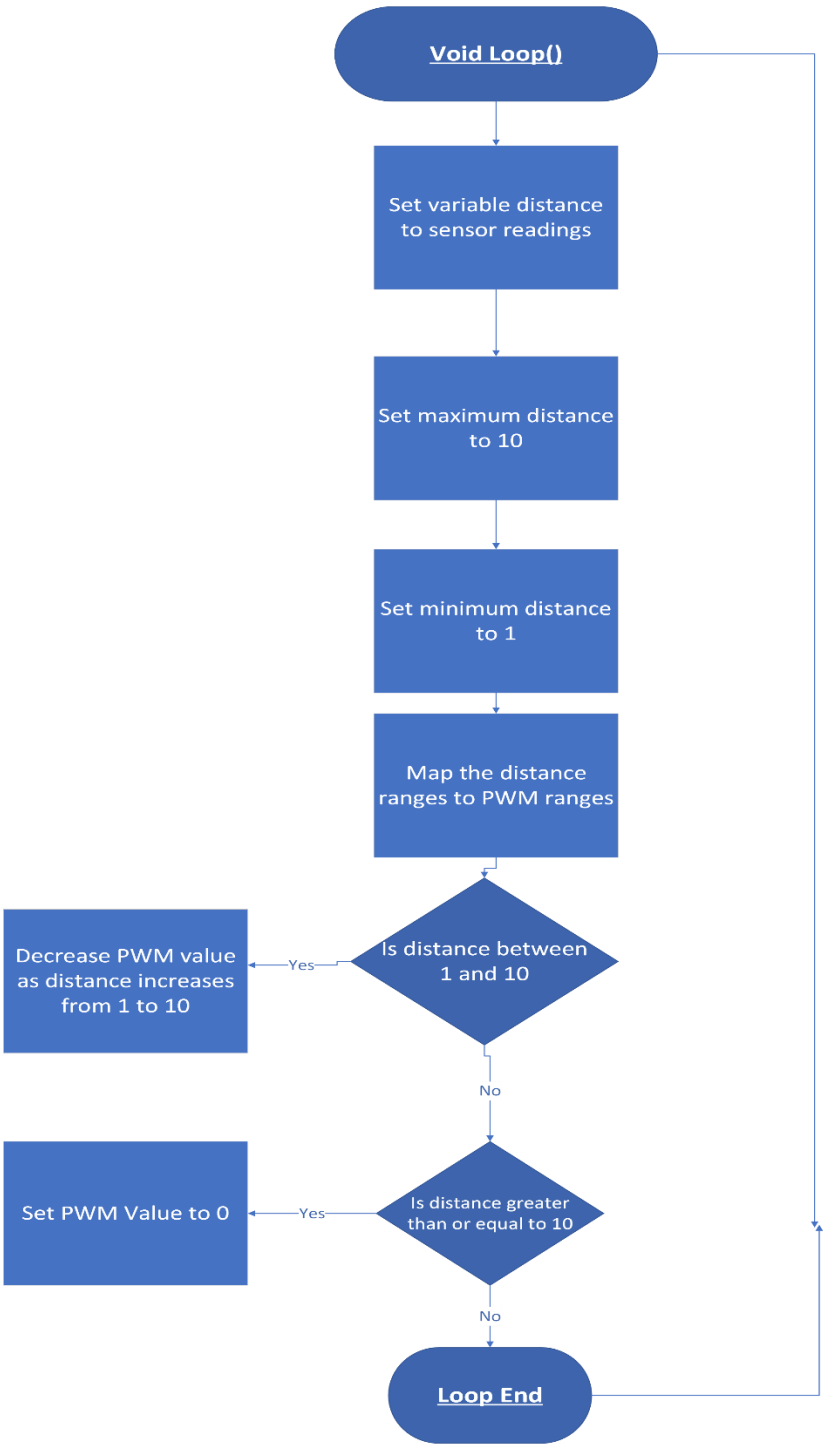


Figure 9:LED Dim Code Flowchart

The flowchart above details the software loop required to dim the LED depending on the distance measured by the LED. By using an if statement to check whether the distance is between 1 and 10 the ranges are then mapped so that as the distance measured increases from 1 to 10 the PWM value decreases in a similar increment to where once the sensor sends back a value greater than or equal to 10 the LED switches OFF.

1.2 IMU Subsystem

1.2.1 Principle of Operation

An IMU (Inertial Measurement Unit) is a device that measures the force, angular movement and the orientation of an object. In specific IMUs the magnetic field surrounding a body can also be measured. This is done through a variety of sensors including accelerometers, gyroscopes and even magnetometers. The most common technology used as a basis for IMUs is MEMS (micro-electromagnetic systems).MEMS are extremely tiny devices (down to micrometres in size) that are a combination of electrical and mechanical elements made from various materials (such as silicon, polymers, metal) that are designed to provide a specific function[5]. The primary benefits of MEMS are its extremely small form factor allowing for its suitability in applications that require low payload sizes and weight. In addition, the lower power consumption makes it exceptionally useful for battery-powered devices. The IMU which will be discussed in this report is the MPU-6050 which contains a 3-axis gyroscope and a 3-axis accelerometer. A 3-axis accelerometer works through the use of capacitive sensing, in which the change in capacitance between the fixed electrodes and proof mass is used to determine body acceleration[6]. Since the distance between the plates of a capacitor is inversely proportional to capacitance a change in the distance between the plates would then lead to a change in capacitance. Due to capacitance also being proportional to voltage a change in voltage would be recognised by the system and then processed into a readable output. A 3-axis accelerometer measures movement in the X, Y, Z axis which allows for accurate measurement of an object's tilt and or movement as an objects state of motion can be determined in all 3 dimensions. A 3-axis gyroscope measures works through the Coriolis effect. "The Coriolis effect is the apparent deflection caused to a moving object caused by the Earth's rotation. In a Coriolis effect gyroscope, an oscillating proof mass is used that creates a secondary vibration when the gyro is rotated due to the Coriolis effect on the proof mass. The secondary vibration is measured to determine the rate of rotation and resulting angular change."[7] During motion the Coriolis effect causes movement of the proof mass against the direction of rotation thus resulting in a change in capacitance.

1.2.2 Schematic Design & Simulation

Contrary to the ultra-sound sensor, The MPU-6050 is a 3.3V device therefore there is no need for any voltage corrections. As such, the MPU-6050 can connect directly to the ESP32 without any issues with excess voltage potentially damaging the micro-processor. As done for the previous sensor, in order to test whether the sensor is working as expected an LED has been added to the circuit in order to respond to specific outputs from the sensor. In this case, the LED will turn on when the MPU-6050 detects a rotational angle of 90˚.

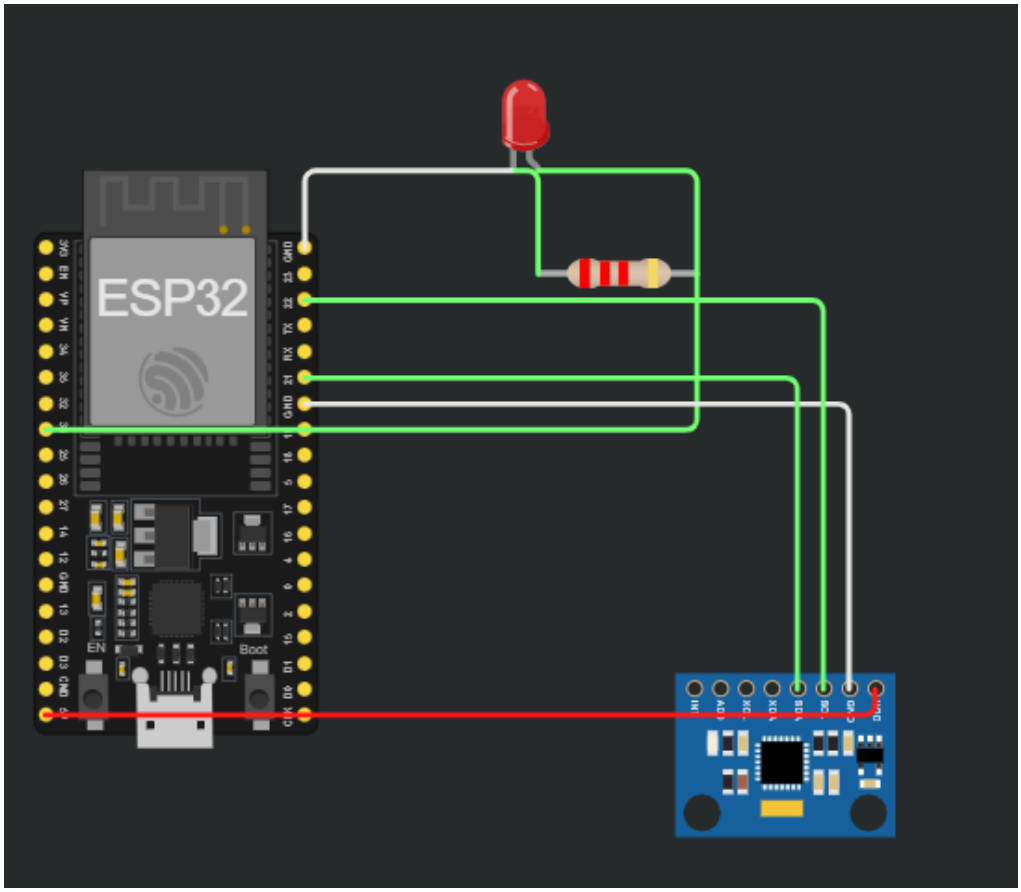


Figure 10:MPU Initialisation Circuit

Figure 10 displays the schematic for the MPU test circuit. The sensor being connected to the GPIO pins of the ESP32 enables software integration. This will allow the ESP32 to output a HIGH signal if the sensor detects a rotational angle of 90˚. This voltage output would then be sent to the LED in order to turn it on thus confirming the sensor works.

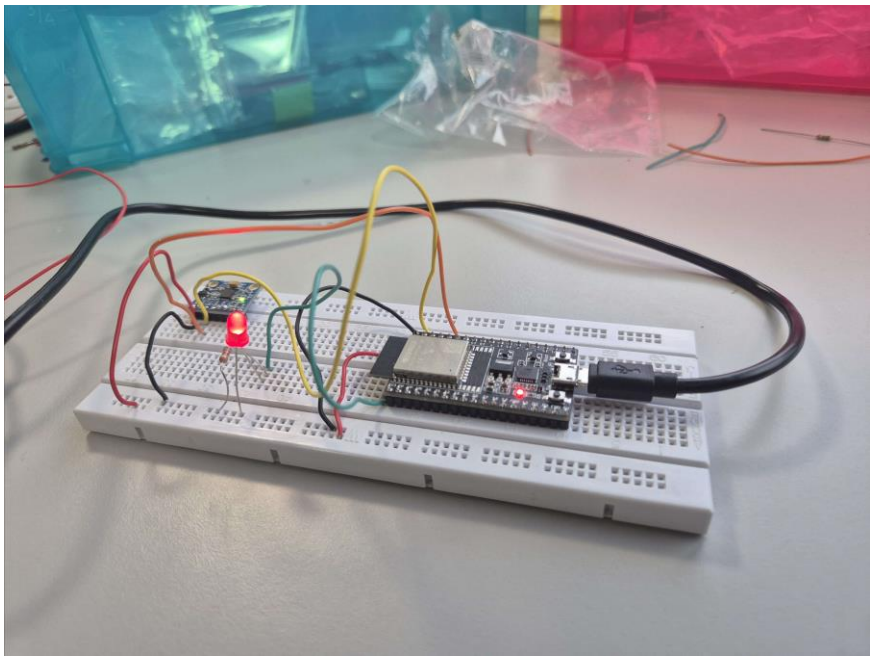


Figure 11:Breadboard MPU Test Circuit

1.2.2 *Software Integration for Sensor Responses*

The software design for the basic implementation is rather simple comprising of a simple check to see whether or not the sensor has detected an angle of 90 degrees. If true, the LED will receive a HIGH output signal.

Figure 12:Software flowchart

The flowchart details the basic premise of the software design in order to test the sensor. Due to the angle measurements for the sensor being in radians in order to simplify the process of checking whether the sensor is detecting an angle of 90˚ the measurements had to be converted to degrees first using the equation:

$$Angle\ in\ degrees = Measured\ Angle \times \frac{180}{\pi}$$

The reason as to why the Z axis is checked is due to the z axis being the principal axis. The principal axis is defined as the axis of any rigid body are those for which the products of inertia

are all zero[8]. In simpler terms means the axis of a rigid body in which rotation occurs with the least resistance due to the distribution of mass.

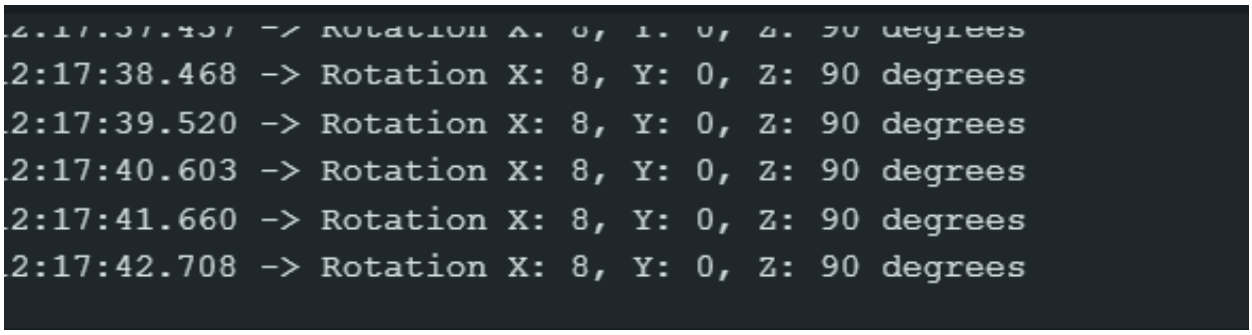


Figure 13:Serial Monitor Sensor Output

As depicted in Figure 13 the sensor does indeed work and according to the code the LED should be turned on. However, the issue with my initial software was not considering the absolute angle. The measuring of an absolute angle is imperative as it increases the precision of the reading. The absolute angle is the measurement of the angular orientation from a fixed reference point. These prior readings have had varying reference points therefore making the readings inaccurate as there is no confirmation on whether or not the sensor has measured true 90˚. By creating a fixed reference point no matter where the sensor starts from 90˚is always measured at the same exact position every time.

```
angleX += correctedGyroX * dt;

 angleY += correctedGyroY * dt;

 angleZ += correctedGyroZ * dt;

const float alpha = 0.96; // Weight for gyroscope (tunable)

 angleX = alpha * angleX + (1 - alpha) * accelAngleX;

 angleY = alpha * angleY + (1 - alpha) * accelAngleY;
```

One of the changes to the code to implement the absolute angle was to add in variable “dt” which tracks the objects rotation over time. By multiplying this by the “weight” of the gyroscope and the angle of acceleration to compute the absolute angle. The angle of acceleration is used as the fixed reference point as variables accelAngleX and accelAngleY are the degree of tilting from the object. The movement of the object outside of rotation becomes the reference point therefore the angular rotation readings are not interfered with by the linear motion of the object

2. Maze Navigation Challenge

The maze navigation challenge involves the use of both sensors in order to complete a maze autonomously. The maze in question is a simple “Default Left Rule Maze”. The default left rule is a maze solving algorithm in which the challenger moves along the walls of the maze and if confronted with a barrier will then move to left. This system guarantees the challenger does not get lost as they can always go back to the entrance by tracing their steps backwards. In addition, this method of maze solving forces the challenger to traverse every “corridor” connected to the walls at least once therefore increasing the likelihood of an exit being found.

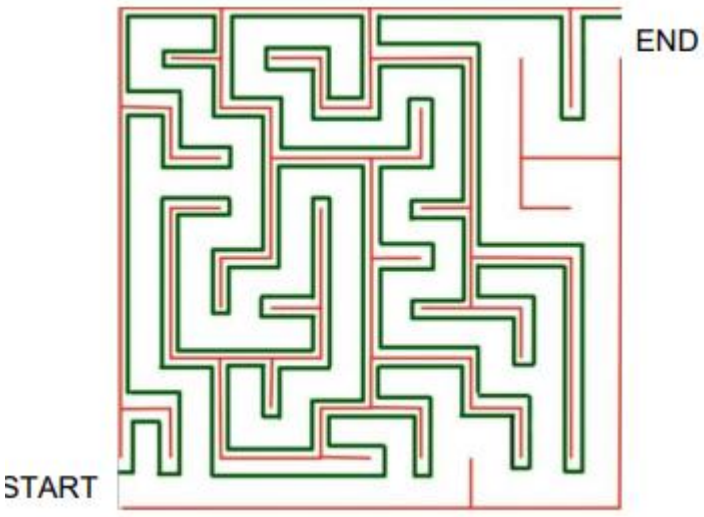


Figure 14:Visualisation of the Left-Hand Rule

The challenger in this case is the EEEbot and the goal is to traverse the maze autonomously. The way in which the EEEbot should complete this challenge is through the use of the two sensors mentioned in Section 1. Theoretically, once the EEEbot is at least 10cm away from the wall the bot should move 90˚ clockwise. The ultrasound sensor is used to gauge the distance between the bot and any obstacles and the IMU is used to measure the angular rotation of the bot and once the bot has rotated 90˚ it should then move forward. This loop continues until either the bot returns to the entrance or finds the exit. In order to have the sensors communicate with the part of the bot controlling motion two ESP32 micro controllers are required. The 'master' ESP32 controls the sensors and the 'slave' ESP32 controls the motors. These two ESP32 communicate via the master-slave wired connection

2.1 Master-Slave Communication

The master-slave communication between multiple ESP32 micro controllers is done via the I2C protocol. I2C (Inter Integrated Circuit) allows for synchronous, multi master and multi slave communications. As per the ESP32 specifications the speed of data transfer is typically 100kbits/s and can go up to a maximum of 400kbits/s [9]. In order to connect the master and slave together we must connect the two micro-processors to a common ground and connect the SDL and SDA pins to each other. On an ESP32 the SDL pin is the GPIO 21 pin, and the SDA pin is the GPIO 22 pin.

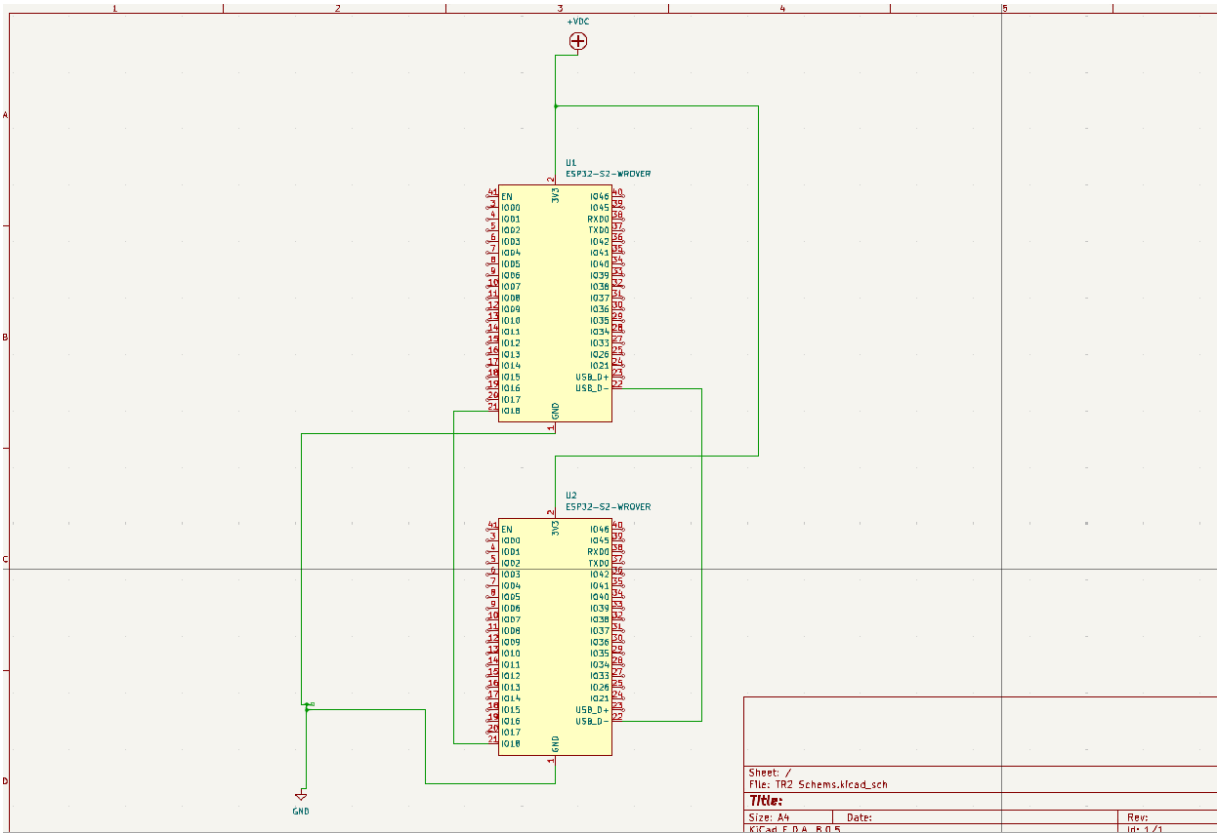


Figure 15:ESP32 Master-Slave Schematic

Figure 15 displays the required connection in order to have the master (top ESP32) and the slave (bottom ESP32) communicate with one another. The next step to implement the I2C communication is via software. Software will have to be written separately for each ESP32. The main function used for the I2C communication is the 'Wire.beginTransmission()' and the 'Wire.begin()' functions. 'Wire.beginTransmission()' takes the address of the slave ESP32 and initiates a connection to it. 'Wire.begin()' initialises the ESP32 as a slave to receive data.

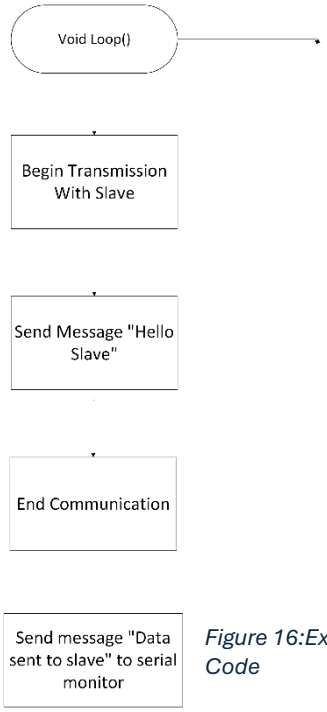
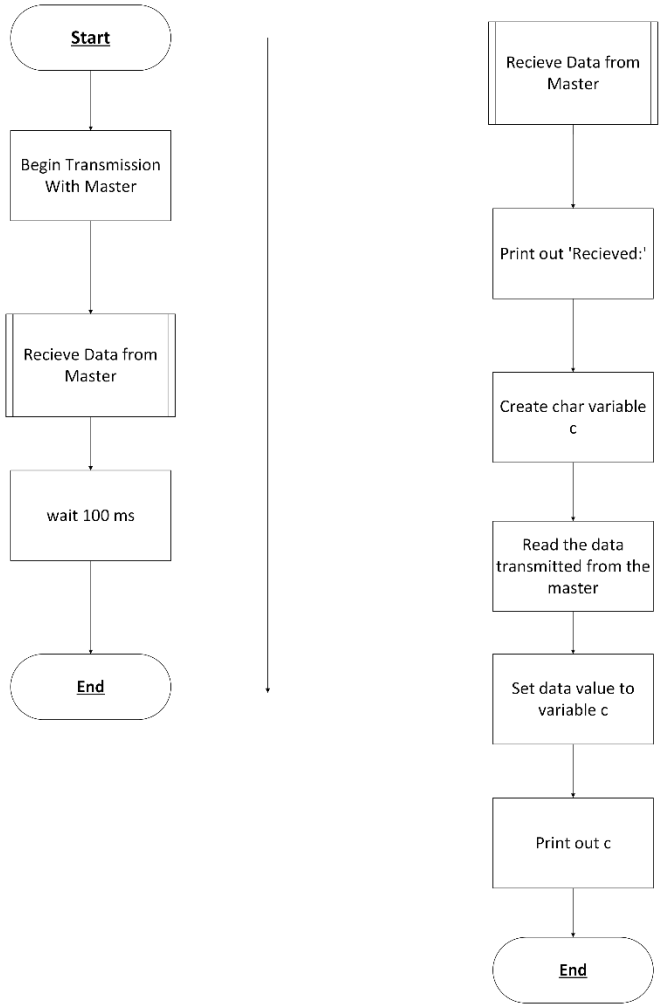


Figure 16:Example of Master Code

```
#include <Wire.h>

#define SLAVE_ADDRESS 0x08

void setup() {

  Wire.begin();

  Serial.begin(115200);

}

void loop() {

  Wire.beginTransmission(SLAVE_ADDRESS

  Wire.write("Hello Slave!");

  Wire.endTransmission();

  Serial.println("Data sent to Slave.");

  delay(1000);

}
```

The flowchart above simplifies the example master code. This simple code displays the necessary software integration required to make the master-slave interaction work. The code for the slave will deviate from the master code as instead of transmitting data it is receiving the data.




```
#include <Wire.h>

#define SLAVE_ADDRESS 0x08

void receiveEvent(int bytes) {

  Serial.print("Received: ");

  while (Wire.available()) {

    char c = Wire.read();  // Read
received byte

    Serial.print(c);

  }

  Serial.println();

}

void setup() {

  Wire.begin(SLAVE_ADDRESS);

  Wire.onReceive(receiveEvent);  data

  Serial.begin(115200);

}


void loop() {

  delay(100);

}
```

Figure 17:Slave Example Code

Similarly to the master code, this simple example of the slave code displays the fundamentals required to get the I2C communication to work correctly. The use of the function 'Recieveevent()' was to increase the efficiency of the code and also allow said function to be used in other pieces of code if the situation requires it to. In the context of the maze navigation challenge, this communication between two micro controllers is vital. For instance, if the HC-SR04 detects an object that's 10cm away then the master ESP32 would transmit this to the slave ESP32 which would stop the motors and start turning clockwise to avoid the obstacle and move further down the maze.

2.2 Software for Maze-Navigation

As explained in the previous section the main aspect of the software used for the maze navigation is the wired communication between two ESP32 microprocessors. However, in order to use this in the context of the maze navigation challenge software would have to be developed so the master ESP32 transmits data to the Slave ESP32 in order to react to any environmental changes

```
if (angle == 0 && distance > 10) {
   state = 0;
} else {
   state = 1;
}

Serial.print("distance: ");
Serial.print(distance);
Serial.print("  ");
Serial.println(state);

Wire.write((byte)(state & 0xFF));
delay(10);
```

Figure 19:Master-Code Snippet

This code snippet displays the contents of the data being sent from the master ESP32 to the slave ESP32. The variable 'state' is used to indicate the next action for the bot. If state is equal to 0 then the bot should continue moving forward. However, is state is equal to 1 then the bot should stop and rotate 90˚degrees. The main environmental change that dictates whether state is equal to 0 or 1 is the distance measured by the ultrasound sensor. As discussed in (1.1.2.3 Testing Sensor Responses to specific Ranges), the initial test was to check if an LED will light up if the ultrasound sensor measures a distance under 10cm. In this practical situation, when the sensor measures a distance of under 10cm the variable state is set to 1. As such this variable will be sent to the slave ESP32 to command it to stop and turn around. The function Wire.write() deals with the transmission of the 'state' variable, the data in this case being transmitted as a byte to ensure compatibility with the Wire.write() function. This is done via the AND operation between the variable 'state' and 0xFF which is 255 in decimal. This result of this operation is the lowest 8-bit binary possible for the 'state' variable thus preventing out of range variables. The benefit of transmitting data as a singular byte is the efficiency of data transfer. Transmitting the 'state' variable as an integer would require multiple transmissions thus leading to slower data transfers. As continued from the master code, The variable state decides whether the bot moves forwards or moves anticlockwise. The use of a switch case here is to increase the efficiency of the code as the repeated use of if and if else statements tend to slow down the compilation of software.

```
switch (state) {
  case 0:
    steeringServo.write(91);
    goForwards();
    motors(95, 254);
    break;

  case 1:
    stopMotors();
    delay(10);
    steeringServo.write(0);
    goAntiClockwise();
    break;
`
```

Figure 18:Slave-Code Snippet

3. Line Following Challenge

The Line following challenge consists of getting the bot to follow a path set by a solid black line. The methodology used to achieve this is the use of photodiodes and infrared LEDs. The basic concept of this challenge is for the infrared LED to emit light and said light being reflected and detected by the photodiode. Due to the difference in how much light is reflected from black and white surfaces the photodiode would measure this difference and thus send data to the bot informing the bot where the black line is. This section details the hardware and software designs needed to complete this challenge.

3.1 Hardware Design

3.1.1 Principle of Operation

Photodiodes are an example of optical sensors. Optical sensors are devices that measure the changes in light intensity in a given environment. Since black objects reflect far less light than white light an optical sensor would be able to measure the change in light intensity between the reflected light from both surfaces. The reasoning as to why black objects reflect significantly less light is due to the higher rate of EM radiation from black objects. Since the preponderance of EM radiation is absorbed, less radiation is reflected thus leading to a lower light intensity relative to white objects that have a lower rate of EM radiation absorption. This relationship is based on the physics of a black body in which an object that's perfectly black will absorb any radiation incident upon it regardless of wavelength or temperature[10]. Since a perfect black body is merely theoretical, all black objects will reflect some sort of light which would be measurable by an optical sensor and also be a lot smaller compared to a white object. The optical sensor that will be used for this challenge is a photodiode.

3.1.1.2 Principle of Photodiodes

Photodiodes are optical sensors that convert light into a small current known as a photocurrent. The photocurrent produced by a photodiode is directly proportional to the light intensity of the source. The photocurrent produced is also inversely proportional to the distance between the photodiode and the source or in this case the reflecting object gets smaller the light intensity increases as more light is reflected thus leading to a higher photocurrent.

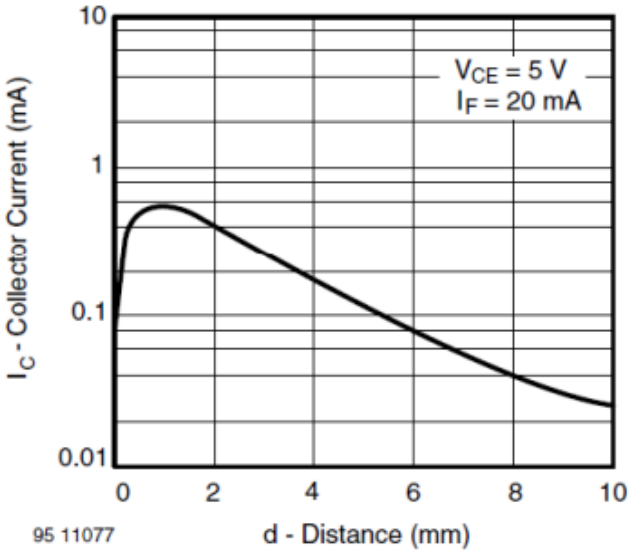


Figure 20:Relationship between Photocurrent and Distance

The collector's current is the maximum photocurrent produced by a specific sensor. As seen on the graph, the collectors current and the distance have a direct relationship up until the peak collector's current is reached. Once the peak current is reached distance and collectors current seem to have an inversely proportional relationship. The ideal scenario is to have the peak collectors current be reached at the operating distance. For instance, within the context of the line following example the operating distance would be approximately 5cm. Therefore, in order to measure the photocurrent more easily the peak collectors current must be reached in order to measure the maximum possible photocurrent.

3.1.1.3 Photodiode Operating Conditions

Photodiodes have two operating conditions called photoconductive and photovoltaic. The main differences between photoconductive and photovoltaic operating conditions are the voltage bias involved and the existence of dark current. Dark current is the flow of electrons while there is no illumination present[11]. The existence of dark current specifically in the photoconductive operating condition provides a challenge when measuring photocurrent as dark current will have to be take into consideration. For this line following exercise all photodiodes used will be in the photoconductive operating condition due to the reverse bias aspect of the operating condition. Photoconductive sensors have a reverse bias which increases the response time and linearity of the diode due to the larger depletion region width and decrease in junction capacitance [12]. This aspect of photoconductive sensors makes it very suitable for high-speed applications. Photovoltaic sensors have no bias which means the flow of current is unrestricted thus leading to a build-up of voltage due to the greater junction capacitance[12]. Junction capacitance being defined as the capacitance existing between a p-n junction. Due to the nature of p-n junctions, the p and n side of the junction acts as pseudo metal plates of a capacitor thus the unrestricted movement of charge will lead to a build up of voltage stored within the p-n junction[13].

3.1.2 Testing and Simulations

In order to utilise the photodiode and be able to extract a useful output from it an amplifier circuit is required. Since the magnitude of the photocurrent produced by the photodiode is in the milliamps, an amplifier circuit is needed in order to increase the magnitude of the photocurrent in order to make it easier to measure.

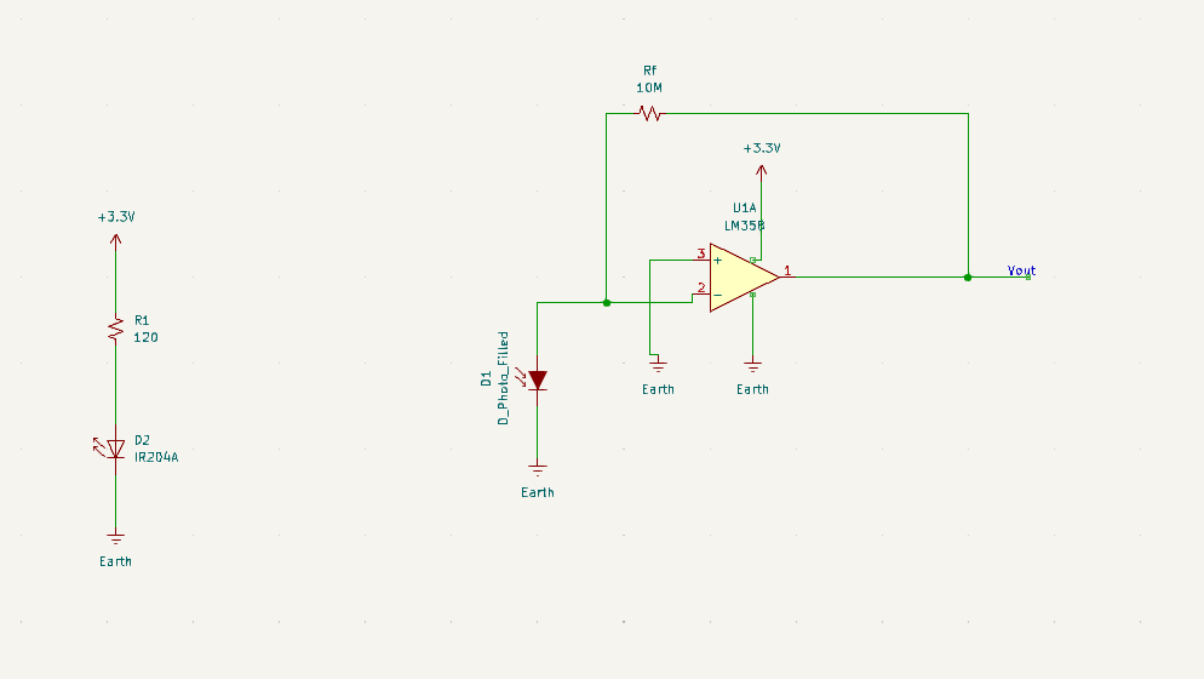


Figure 21:Amplifier Test Circuit Schematic

This is the basic schematic for the circuit used to test out the amplified output of the photodiode. In this case the light source will be an Infrared LED mirroring the actual use case for the line following exercise. The photodiode is connected to a transimpedance amplifier circuit. A transimpedance amplifier circuit takes a current source and amplifies it but also converts it to a voltage output. A voltage output is more useful than a current output due to its faster response time and higher sensitivity compared to a current output [14].The gain of this amplifier circuit is dictated by the value of Rf which in this case is 10MΩ. The expected gain for this circuit should also be 10 million or 140dB.

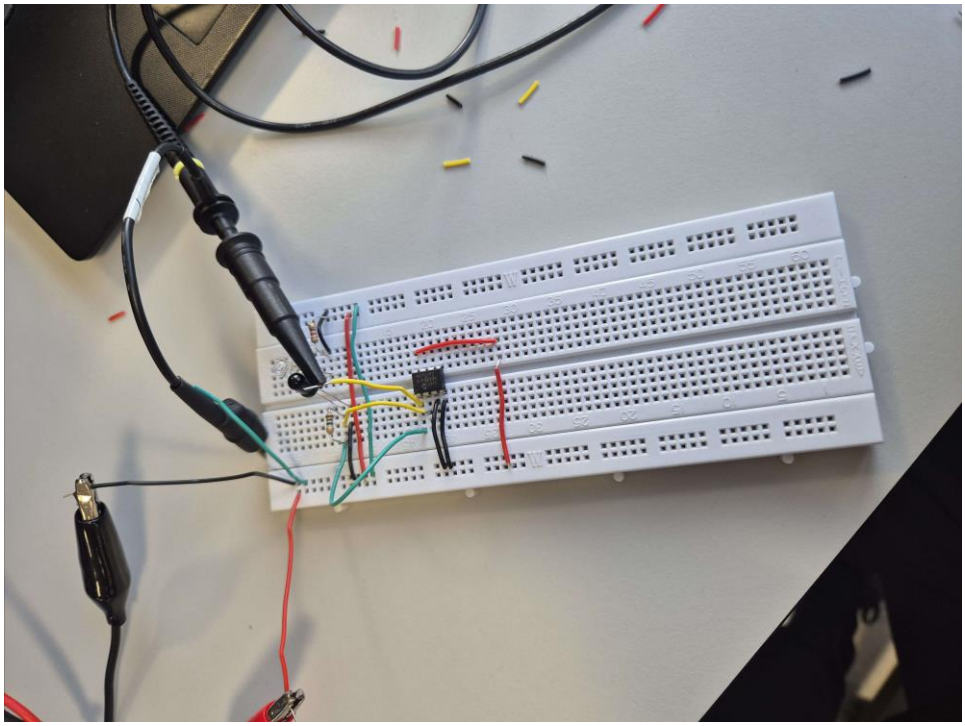


Figure 22:Amplifier Breadboard Test Circuit

The resistors used in this breadboard circuit is 120Ω for the current limiting resistor if the IR LED and 200kΩ for the reference resistor which should theoretically be the gain of the amplifier circuit. According to the photodiode datasheet, the typical photocurrent produced by the photodiode is around 10μA. Therefore, the expected voltage output should be current multiplied by the reference resistor. As such, the voltage output should theoretically be 2V on the condition that the IR LED is constantly outputting light of wavelength 940nm.

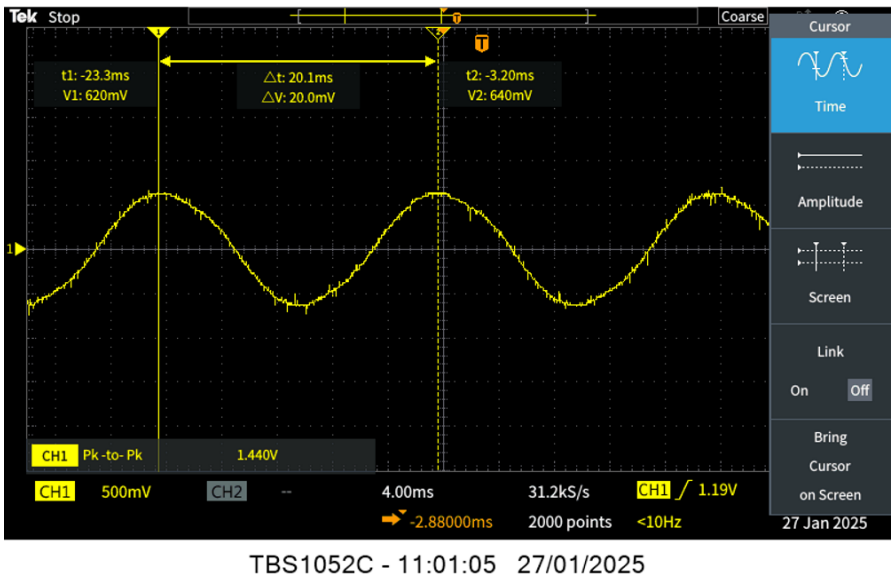


Figure 23:Oscilloscope Output

As shown the by the oscilloscope the actual output is 1.44V, the reasoning behind this could either be the voltage input at 3.3V. A higher voltage input would increase the gain of the amplifier circuit as the voltage output of any amplifier circuit is limited to the voltage input.

3.1.2 Stripboard Designs

While attempting to convert the schematic shown in Figure 21:Amplifier Test Circuit Schematic multiple stripboard designs were made each with their own flaws until the final design was constructed and used for the line following challenge.

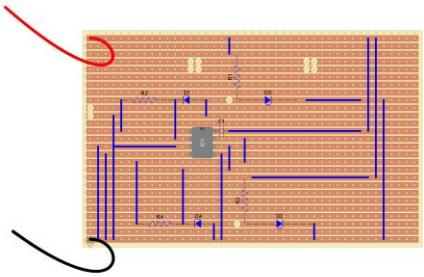


Figure 24:Stripboard Design 1

This first stripboard design contains a couple of issues, and the most glaring one is the lack of sensors used. As seen, the number of sensors used in this design is two which causes a severe lack of accuracy and precision in measurements due to the small number of readings available. In addition, the lack of output wires makes it impossible for the readings from the sensors to be directed to the EEEbot thus making this design incapable of performing the necessary functions required for this challenge.

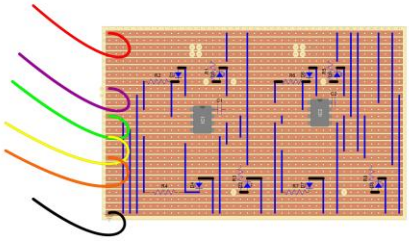


Figure 25:Stripboard Design 2

This stripboard design improves upon the first one via the addition of output wires and the increase in the number of sensors from two to four. However, this design still has its issues mainly with the placement of the sensors. This design consists of two photodiodes on the top and two on the bottom, the issue with such placement is the distance between the sensors are too large. As such, this creates a weak signal thereby leading to a decrease in accuracy which would in practice make this stripboard unable to properly follow the line. A possible improvement would be to place all the sensors on a single line thus decreasing the distances between them and increasing the signal thereby increasing the accuracy of the measurements.

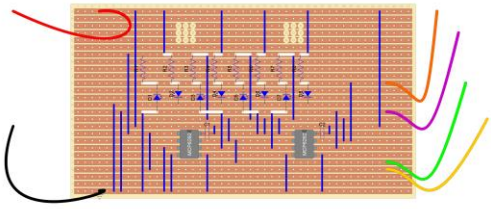


Figure 26:Stripboard Design 3

Though close to a perfect design the issue with this design isn't more so any complications with the sensors as the sensor placement is correct. By placing them in a line and aligning them horizontally the total surface area able to be measured increases thus leading to an increase in accuracy. The alignment also increases the total photocurrent which would mean the voltage output will also be larger thus it would be measured more easily. The issue with this design is

more so the difficulty in soldering. By placing the LED and photodiode back-to-back like this would require a lot of tracks being finely cut in order to prevent any short circuits as such another final design was made in order to make soldering much easier.

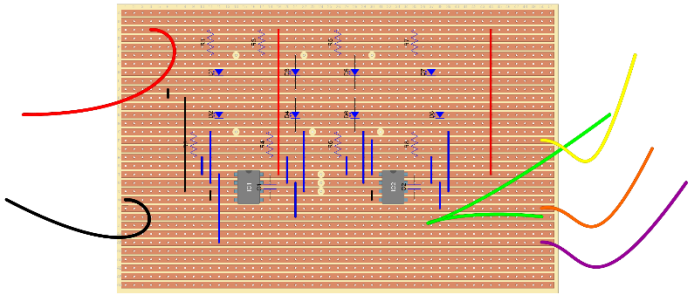


Figure 27:Final Stripboard Design

This is the design that was ultimately implemented for the line following challenge. This final design incorporates all the previous improvements such as the alignment of the sensors and the use of output wires and makes it easier to solder which also makes it less prone to short circuits. The reason why the LEDs were placed above the photodiodes was to make it easier to connect them all to ground and decrease the number of wires but to also remove the need to cut as many tracks as possible.

3.2 Software Design

The software design in this challenge utilises the same master-slave I2C communication described in 2.1 . As such, this section will be split into two sub sections describing the software design for the master ESP32 which is connected to the sensor subsystems and the software design for the slave ESP32 which is connected to the motor systems

3.2.1 *Master Code Design*

The premise for the master code is to obtain readings from the multiple photodiodes and depending on what each individual sensor is reading send a command to the slave to dictate the direction of the bot. The figure 28 shows the data from each sensor being set to different variables. This makes it easier to compare the values measured from the sensors. This especially applies to the mid-left and mid-right sensors as those two together sense the front of the bot which dictates whether the bot should be moving forwards or not. The reasoning behind measuring the data from the sensors as an analogue output rather than a digital one is due to the characteristics of each type of output. An analogue output provides a continuous signal which means that the output changes over time. This is more beneficial compared to a digital output which is discrete as analogue outputs have faster response times as there is no need for a DAC to convert the initial analogue output into a digital one [15]. In high-speed applications such as this one any sort of decrease in response times could lead to issues with the bot's ability to follow the line.

```
int leftValue =
analogRead(LEFT_SENSOR);

  int rightValue =
analogRead(RIGHT_SENSOR);

  int midLeftValue =
analogRead(MID_LEFT_SENSOR);

  int midRightValue =
analogRead(MID_RIGHT_SENSOR);
```

Figure 28:Master-Code Snippet 1

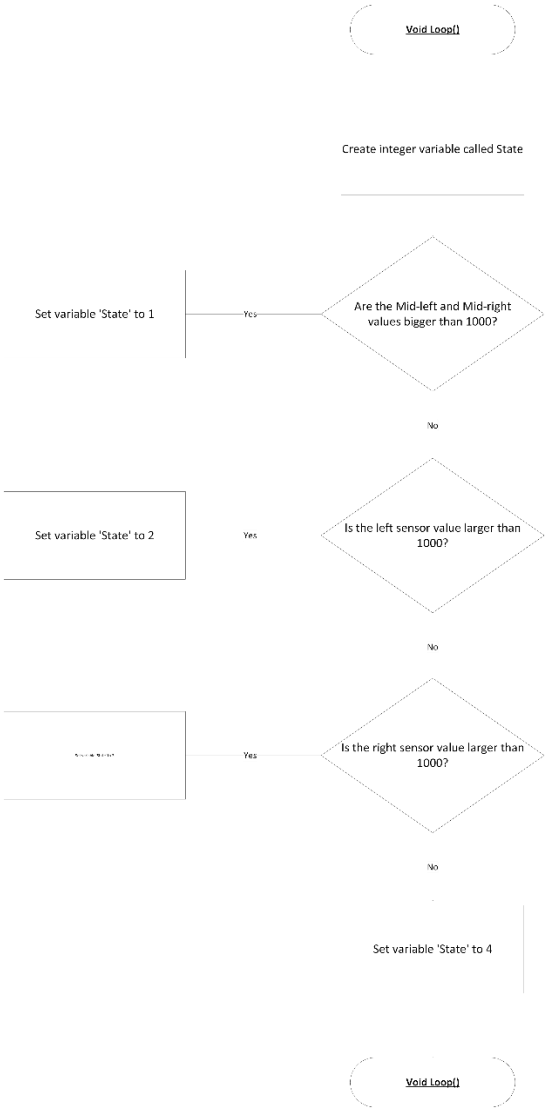


Figure 29:Master-Code flowchart

The figure 29 flowchart describes the process in which the software decides which direction the bot should go depending on the reading from the photodiodes. The value of 1000 is defined as the minimum photocurrent the photodiode should produce before the software deems that it is sensing the black line. Once the sensors reading exceeds that threshold the variable 'State' will change. A 'State' value of 1 would cause the bot to move forwards, a value of 2 would cause the bot to move right, a value of 3 would cause the bot to move left and if none of the sensors exceed the 1000 threshold then a value of 4 would stop the motors.

3.2.2 Slave Code Design

The premise behind the slave code design is similar to the master code design however instead of transmitting the 'state' variable the slave code receives the slave variable then moves the motors in accordance with the value 'State' variable dictated by the master ESP32. The function described in figure 30 receives the data sent by the master ESP32 and applies it to the variable 'State'. Unlike the approach used in 2.2 a function is used in this case to receive data from the master ESP32. The reason behind this change in approach is to improve upon the efficiency of the code as within a function checks can be made to make sure the data has been received properly. The do-while loop within the function checks whether or not there is a connected. The function "Wire.available()" returns the number of bytes sent from the master ESP32. If the function reads 0 bytes, then the while loop breaks thus the variable 'State' doesn't get updated and prints out an error message. Checks like these are important in order to make troubleshooting far easier, if for instance the motors aren't moving its possible to check whether its an issue with the commands being sent from the ESP32 or whether it's a hardware problem with the motors themselves.

```
void onReceive(int howMany)
{
  do
{
    if (Wire.available())
{
      int State = Wire.read();
      Serial.println(State);
    } else
 {
      Serial.println("Error: No data
received");
    }
  } while (Wire.available());
}
```

Figure 30:Slave code receive function

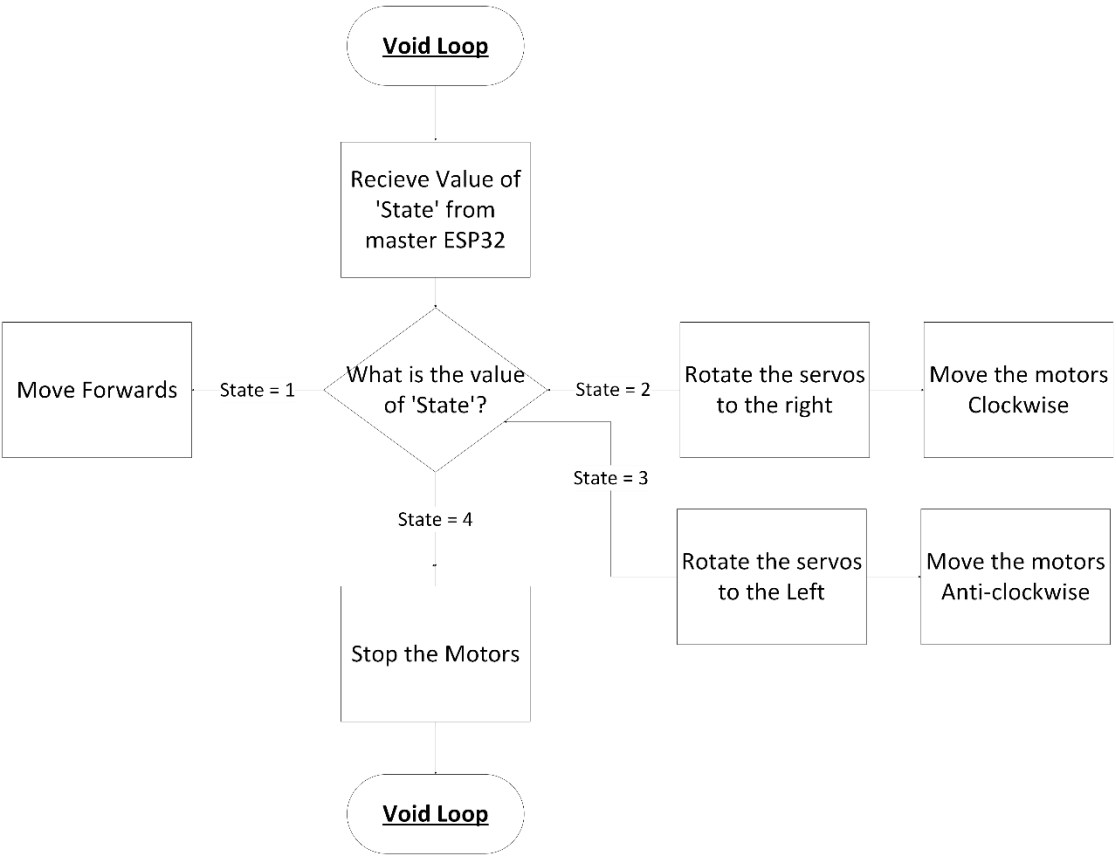


Figure 31:Slave Code Flowchart

Figure 31 displays the main aspect of the slave code which is determining the motor direction depending on the value of the variable 'State' received. In the event the motor would need to move clockwise or anticlockwise the servos would have to be initialised first in order for the bot to move left or right. Since the value of 'State' is defined earlier outside of the setup or loop functions it is being treated as a global variable. The function described in Figure 30 updates whenever new data is received. As such, both the variable 'State' and the function receiving data does not need to be initialised again in the loop function. The reasoning behind the inclusion of the first process which is receiving the value of 'State' is to make the flowchart easier to follow and understand.

Conclusion

As for my own results regarding these challenges both were unfortunately met with some issues. The main issue which affected both challenges was the communication between the master and slave ESP32. During my attempts at the maze navigation and line following challenge I was unable to initialise a proper connection between the two microcontrollers. Due to this issue, my motors did not receive any commands thus leading to my bot staying stationary despite there being no glaring hardware issues. A possible improvement to my initial methods would be to add error checks in the code as I displayed in Figure 30 so I would know whether or not there was a hardware issue or software issue.

References

[1] ‘Sensor Terminology’. Accessed: Feb. 06, 2025. [Online]. Available: https://www.ni.com/en/shop/data-acquisition/sensor-fundamentals/sensor-terminology.html

[2] K. L. Meadmore, S. P. Liversedge, M. J. Wenger, and N. Donnelly, ‘Exploring the relationship between response time, sensitivity and bias in categorical and coordinate visuospatial processes: Evidence for hemispheric specialisation’, *J. Cogn. Psychol.*, vol. 26, no. 4, pp. 423–432, May 2014, doi: 10.1080/20445911.2014.903255.

[3] ‘map() | Arduino Documentation’. Accessed: Feb. 07, 2025. [Online]. Available: https://docs.arduino.cc/language-reference/en/functions/math/map/

[4] ‘Pulse wide modulation (PWM) explained’, LTECH Europe. Accessed: Feb. 07, 2025. [Online]. Available: https://www.ltech-led.eu/it-it/blog/pulse-width-modulation-pwm-explained.html

[5] ‘Inertial Measurement Unit (IMU) – An Introduction’, Advanced Navigation. Accessed: Feb. 12, 2025. [Online]. Available: https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/

[6] ‘How do Accelerometers and Gyroscopes work?’, CircuitBread. Accessed: Feb. 12, 2025. [Online]. Available: https://www.circuitbread.com/ee-faq/how-do-accelerometers-and-gyroscopes-work

[7] ‘Coriolis Effect’, Advanced Navigation. Accessed: Feb. 12, 2025. [Online]. Available: https://www.advancednavigation.com/glossary/coriolis-effect/

[8] ‘Principal Axis - an overview | ScienceDirect Topics’. Accessed: Feb. 13, 2025. [Online]. Available: https://www.sciencedirect.com/topics/engineering/principal-axis

[9] S. Santos, ‘ESP32 I2C Master and Slave Communication (Arduino) | Random Nerd Tutorials’. Accessed: Feb. 17, 2025. [Online]. Available: https://randomnerdtutorials.com/esp32-i2c-master-slave-arduino/

[10] ‘Black Body Radiation - an overview | ScienceDirect Topics’. Accessed: Feb. 24, 2025. [Online]. Available: https://www.sciencedirect.com/topics/physics-and-astronomy/black-body-radiation

[11] ‘Understanding Photovoltaic and Photoconductive Modes of Photodiode Operation - Technical Articles’. Accessed: Feb. 24, 2025. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/understanding-photovoltaic-and-photoconductive-modes-of-photodiode-operation/

[12] tavotech.com, ‘Photodiode biasing’, TavoTech. Accessed: Feb. 24, 2025. [Online]. Available: https://tavotech.com/photodiode-biasing-photoconductive-or-photovoltaic-mode/

[13] ‘Diode junction capacitance | Transition capacitance and diffusion capacitance’. Accessed: Feb. 24, 2025. [Online]. Available: https://www.physics-and-radio-electronics.com/electronic-devices-and-circuits/semiconductor-diodes/junctioncapacitance-transitioncapacitance-diffusioncapacitance.html#google_vignette

[14] ‘Sensor Output Types’. Accessed: Feb. 24, 2025. [Online]. Available: https://www.monolithicpower.com/en/learning/mpscholar/sensors/basics-of-sensor-operation/sensor-output-types

[15] ‘Analog vs. Digital Sensors - Unpacking the Cost Factors’. Accessed: Feb. 24, 2025. [Online]. Available: https://www.processsensing.com/en-us/blog/digital-versus-analog-sensors.htm

Appendix

[1] IMU MPU-6050 Adafruit Library - https://github.com/adafruit/Adafruit_MPU6050

[2] HC-SR04 Newping Library - https://bitbucket.org/teckel12/arduino-new-ping/wiki/Home

[3] Photodiode Datasheet - https://item.szlcsc.com/datasheet/ZSPD053B-S40/6103837.html?lcsc_vid=QVFaU1AEFlZWBFxeQQQMV1cARgJXA1NSQAVfBFEHEgMxVlNSR1RdVVVeT1VdUDsOAxUeFF5JWBYZEEoBGA4JCwFIFA4DSA%3D%3D

[4] MCP6292 Datasheet - https://ww1.microchip.com/downloads/aemDocuments/documents/MSLD/ProductDocuments/DataSheets/MCP6291-Family-Data-Sheet-DS20001812G.pdf