



University of  
**Nottingham**  
UK | CHINA | MALAYSIA

# EEEE1002: Applied Electrical and Electronic Engineering: Construction Project

## Individual Lab Report 1

Covering Weeks 1, 2 and 3

Student ID Number: 20645251

Report Due Date: 09/01/2025

## Abstract

EEEE1002 lab weeks 1 and 2 covers fundamental skills for designing, building, testing and understanding simple circuits that will help transfer over the course of the program. The work will largely consist of using and understanding basic tools and components then applying the knowledge to build simple circuits to demonstrate comprehension of the material. Week 3 will involve the start of construction of the EEEBot that will span the duration of the module.

The work under taken throughout the week was completed mostly without any issues due to good circuit design however any problem that aroused were mainly due to poor soldering and an incorrect understanding of the components used. This was easily rectified by going through the data sheets or by seeking assistance.

## Contents

Abstract .....	2
1.0 Introduction .....	4
2.0 OP-AMP Circuit.....	5
2.1 Non inverting amplifier .....	5
2.2 Breadboard Prototyping.....	5
3.0 555 Timer Monostable.....	7
3.1 Simulation .....	7
3.2 Stripboard Plan.....	8
3.3 Circuit Construction on Stripboard .....	8
3.4 Testing and Questions .....	9
4.0 H-Bridge .....	10
4.1 MOSFETs.....	10
4.2 Full Half Bridge Prototyping .....	12
4.3 Stripboard Construction.....	13
5 Control Logic and Working Out .....	13
5.1 Control Logic Implementation.....	15
6.0 EEEBot .....	16
6.1 Introduction .....	16
6.2 Soldering Components to PCB .....	16
6.3 Front Chassis Assembly .....	17
6.4 Rear Chassis Assembly .....	18
6.5 Final Steps .....	18
7 Programming.....	19
7.1 Initial Testing .....	19
7.2 10 Meter Challenge.....	19
8.0 Conclusion .....	22
References.....	22

## 1.0 Introduction

The purpose of this lab report is to consolidate and record the activities of the first 3 lab weeks of module 1002. The first two weeks mainly covered the basic skills and components used in the lab that will carry over to the rest of the module and beyond. The last week covers the start of construction of the EEEBot that will span the rest of the module.

For the first 2 weeks the material covered is tested in stages throughout the week with various task and simple circuits. The main circuits completed at the end of each week is covered here in chapters 2 to 5. Each of these chapters will cover one circuit each with details regarding them. For week 3, the building of the EEEBot is covered in chapter 6 while the programming of the bot and 10-meter challenge results are covered in chapter 7.

Over all, there were not many difficulties faced throughout the lab weeks with the main issues being my lack of knowledge which led to some mistakes being made when designing the circuit layout as well as when to use a current limiting resistor. These issues were easily solved through testing and tweaking as I go as well as being more thorough when reading through the data sheet.

## 2.0 OP-AMP Circuit

### 2.1 Non inverting amplifier

A non-inverting amplifier is an example of an operation amplifier – OP AMP. The OP AMP I used is an example of an IC. The non inverting OP AMP is a component that can amplify the input signal and output it without inverting the output. The OP AMP model used is the MCP6292

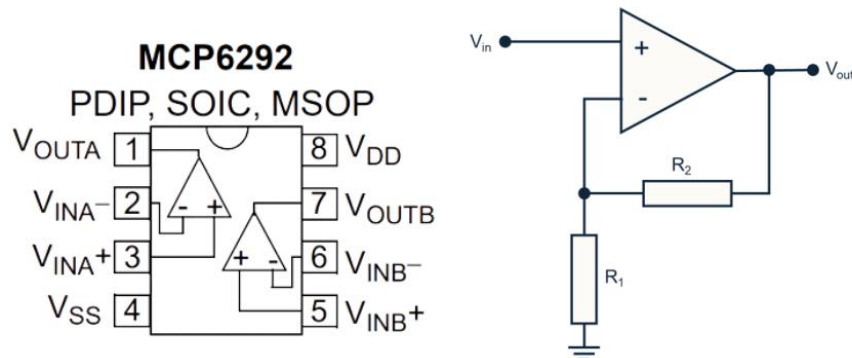


Figure 1 - MCP6292 Pinout and Circuit Diagram from [4]

### 2.2 Breadboard Prototyping

The MCP6292 contains 2 separate amplifiers that can be connected. It also requires power to be connected to  $V_{DD}$  and negative or inverted power to be connected to  $V_{SS}$ .

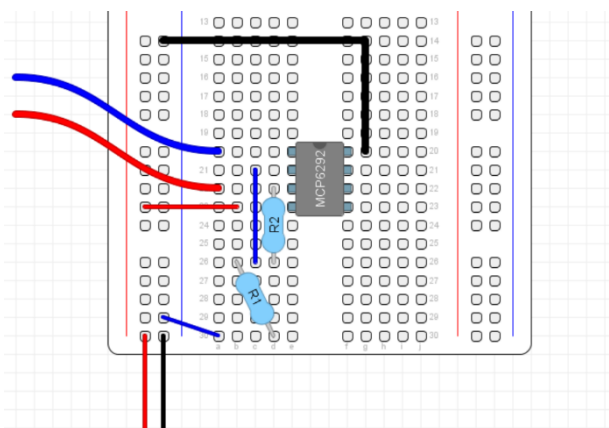


Figure 5 - OP-AMP Circuit Breadboard Schematic

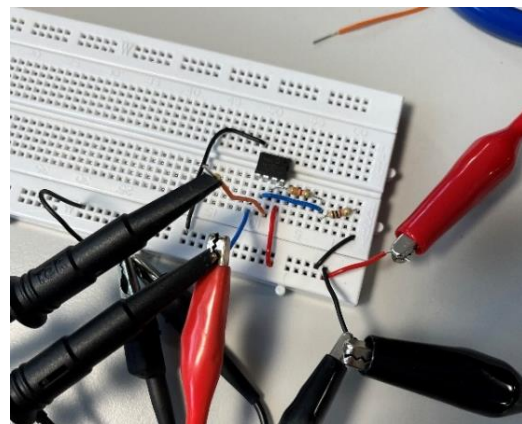


Figure 2 - OP-AMP Circuit Breadboard

The circuit was set up and prototyped on the breadboard following the layout on the right side of Figure 1 - MCP6292 Pinout and Circuit Diagram from which produced the schematic shown in Figure 3 - OP-AMP Circuit Breadboard Schematic

Figure 1 Figure 3. The set up was then measured using an oscilloscope displaying the outputs of 2 waves in Figure 4 - OP-AMP Oscilloscope Output

Figure 4 with the yellow being the input and the blue being the output signal. The oscilloscope output shows that the amplitude of the positive output is greater than that of the input while still maintaining the same frequency of 1KHz that was passed into it by a signal generator. The negative portion of the signal gets clipped can be explained by the gain equation of the non-inverting OP AMP where  $\text{Gain} = 1 + \frac{V_{\text{Out}}}{V_{\text{In}}}$ . This means that the gain can never be less than 1 and so output will always be positive, causing the amplifier to clip the negative values.

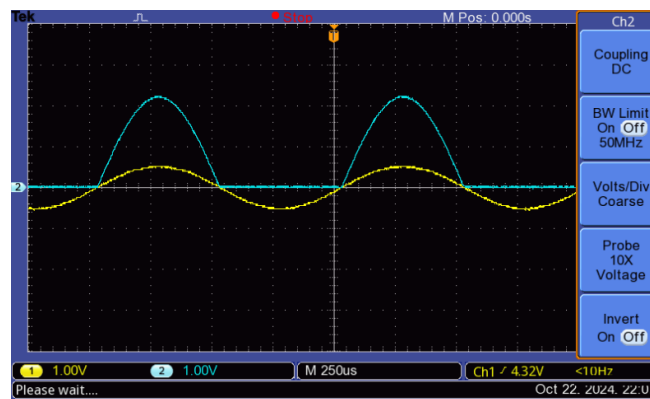


Figure 8 - OP-AMP Oscilloscope Output

## 3.0 555 Timer Monostable

The 555-timer monostable circuit is a circuit that uses a capacitor as a clock. The timer is then connected to an LED to visualize the output. The 555-timer monostable uses the NE555 chip as a key component.

### 3.1 Simulation

First, the circuit is simulated in the LT Spice program to help understand the connection and the expected outputs. Using the equation for capacitance charging we can derive the time constant or the time it takes to charge the capacitor to be  $T = 1.1RC$ . By setting the time we want to be 500ms and knowing that capacitance values tend to fall in the range of micro farads, we can pick a commonly used capacitance value of  $1.5\mu\text{F}$  and so derive the value of resistance to be  $300\text{K}\Omega$ .

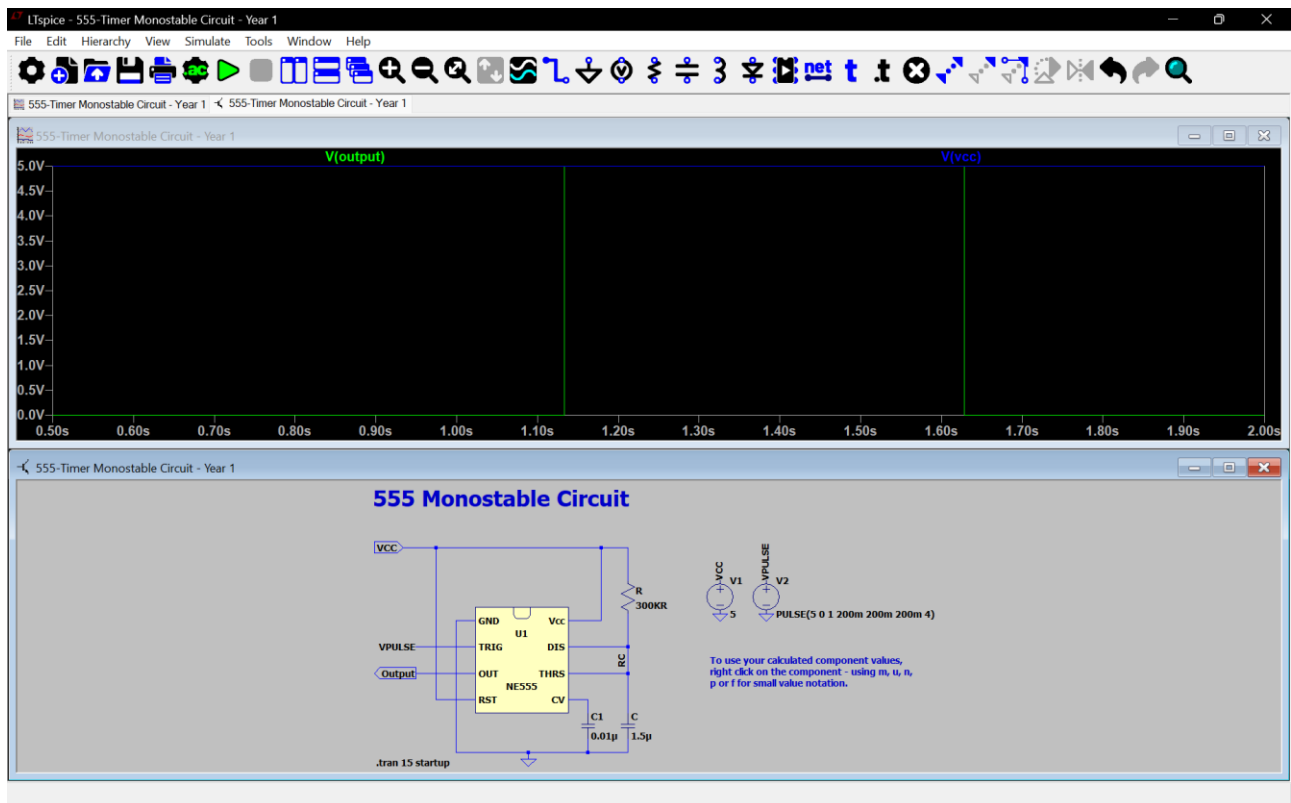
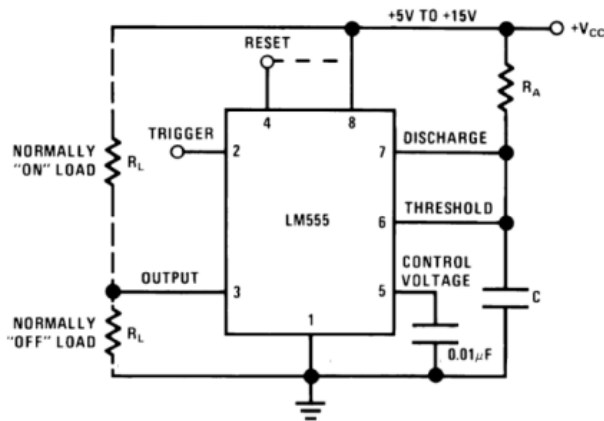


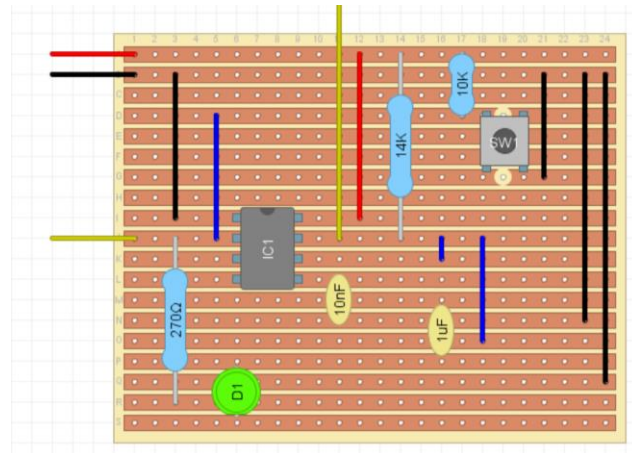
Figure 10 - Simulation Page

### 3.2 Stripboard Plan

A stripboard plan was made based on the layout provided from the NE555 data sheet shown on the left in Figure 6 – Monostable Circuit Layout. This produced the layout shown below in Figure 7 - Timer



Stripboard Layout.



### 3.3 Circuit Construction on Stripboard

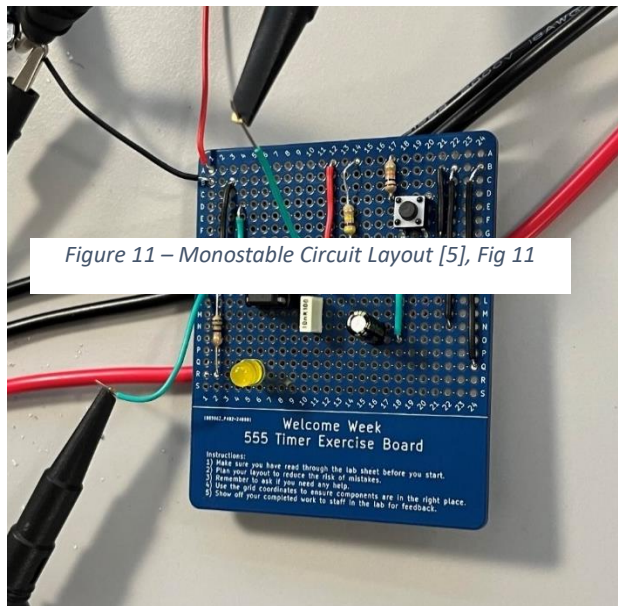


Figure 11 – Monostable Circuit Layout [5], Fig 11

The circuit was constructed on a stripboard with the layout design above.

There were changes in values of components used due to the limited selection of parts available therefore instead of 1.5µF capacitor I used 1µF and

Figure 12 - Timer Stripboard Layout

instead of 300kΩ resistor I used 450KΩ. These values were calculated by dividing and multiplying the values by 2/3 respectively.

There are some poor practices used such as the long-stretched resistor which should be replaced by a shorted resistor and a wire. These design flaws will be improved on in future circuits.

Overall, the construction went smoothly with

minimal obstacles.

Figure 13 - Timer Circuit Stripboard



### 3.4 Testing and Questions

First the circuit was connected to an oscilloscope like in Figure 10 - Timer Stripboard Oscilloscope Testing

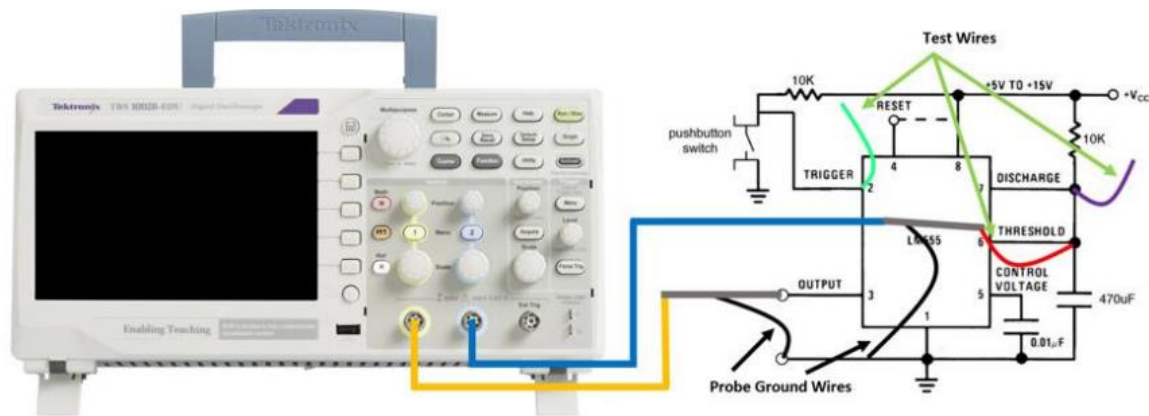


Figure 15 - Timer Stripboard Oscilloscope Testing Setup [4, p. 81], Fig 74.

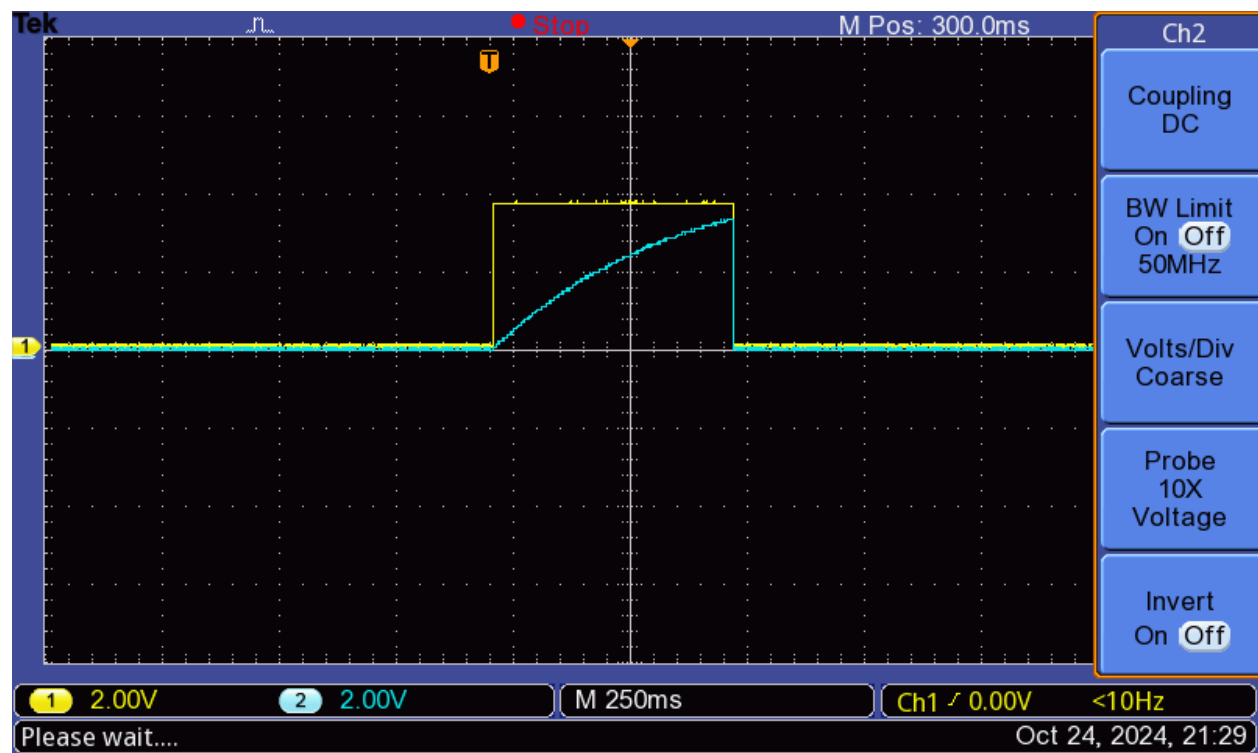


Figure 14 - Timer Oscilloscope Reading

Setup below.

In Figure 9 - Timer Oscilloscope Reading above is the oscilloscope reading produced when the button is pressed. The yellow line shows the NE555 activating as the capacitor charges and the blue line shows the charging of the capacitor. The activation time – time for capacitor to charge – during experimentation was 510ms which was off by 10ms from the calculated value. This could be due to the values of the components not being exactly what it says and so causing the error. However, the percentage error of the result is only 2% which could be considered acceptable.



Question : Observing the waveforms from the simulation or experiment, explain how the 555 timer operates.

The 555 timer operates by using the time the capacitor takes to charge as the timer. When a pulse triggers the 555 timer circuit, it starts to charge the capacitor. While the capacitor is charging a constant signal is output which shows us as the LED lighting up on the board as well as the yellow line on the oscilloscope rising vertically and sustaining a constant pulse while the blue line rises.

Question : How would you adapt the circuit so that the frequency of the output can be varied?

The output frequency can be varied by using a potential divider where one of the resistors is a variable resistor instead of a constant resistor. This will allow the output voltage and current to be easily manipulated and applied to the necessary use case.

## 4.0 H-Bridge

### 4.1 MOSFETs

The Metal Oxide Semiconductor Field Effect Transistors or MOSFETs are a type of transistor. A transistor can be thought of as a small switch that can be controlled using a signal. In week 2 of lab work we will be using 2 types of MOSFETs – NMOS and PMOS – with their – respective – circuit diagrams below.

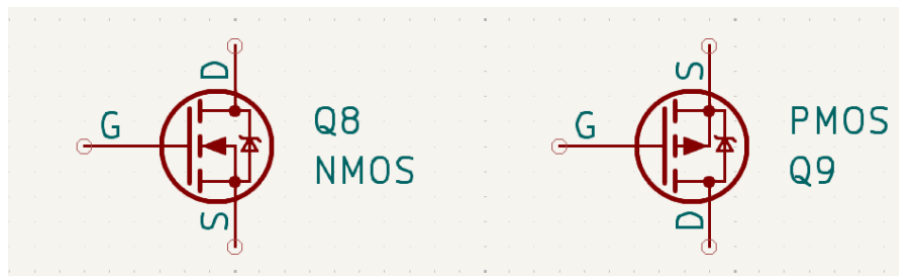


Figure 16 - MOSFET Diagrams [1, p. 5], Fig 1

The MOSFETS each have 3 pins:

- G / Gate: Controls the resistance between the source and drain pin. [1, p. 5]
- D / Drain: Is the output pin of the MOSFET as well as making up the sides of the MOSFET and connecting to ground. [1, p. 5]
- S / Source: Is the power pin that creates a potential difference between Drain pin and ground by making up the sides of the MOSFET. [1, p. 5]

The potential difference between the gate and source pins determines if there is conduction between the drain and source pins or not. The graphs below show the output of both the NMOS (left) and PMOS (right) types.

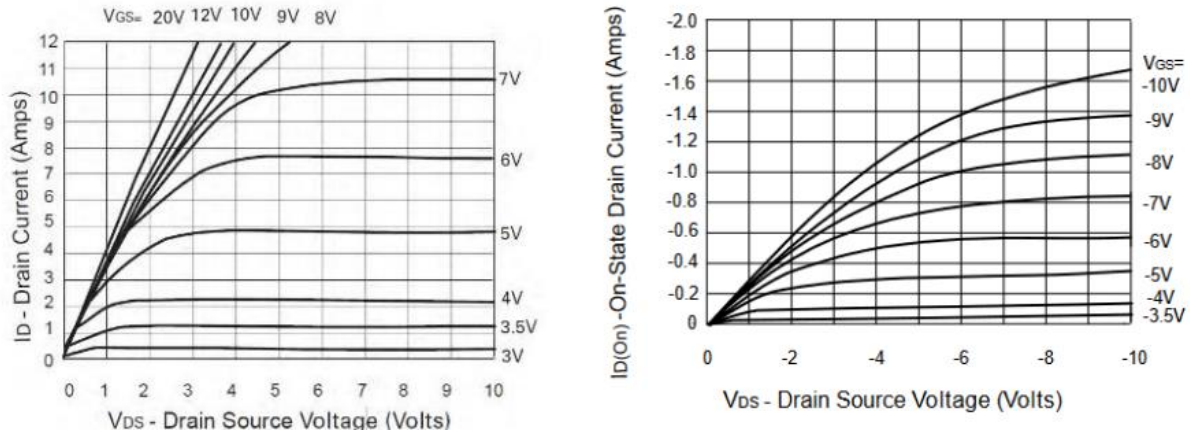


Figure 17 - NMOS and PMOS Activation Graph [1, p. 6], Fig. 2

The PMOS used is the [ZVP2106A](#)

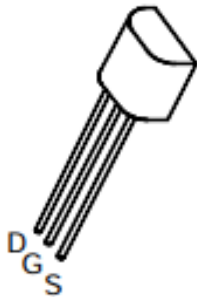


Figure 18 - PMOS Pinout

$V_{ds(max)}$ : -60V  
 $I_{ds(max)}$ : -280mA  
 $R_{ds(on)}$ : 5Ω  
 $P_D$ : 700mW  
 $V_{gs(th)}$ : -1.5V – -3.5V  
 Power dissipation motor no load: 3.125 mW  
 Power dissipation motor stall: 98 mW

Information and diagram from [2]

The NMOS used is the [ZVN4306A](#)

Top-down view:

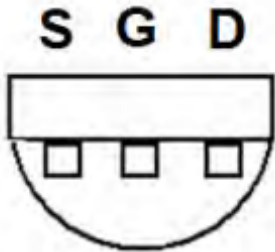


Figure 19 - NMOS Pinout

$V_{ds(max)}$ : 60V  
 $I_{ds(max)}$ : 1.3A  
 $R_{ds(on)}$ : 0.22Ω  
 $P_D$ : 850mW  
 $V_{gs(th)}$ : 1.3V – 3V  
 Power dissipation motor no load: 0.1375 mW  
 Power dissipation motor stall: 3.412 mW

Information and diagram from [3]

## 4.2 Full Half Bridge Prototyping

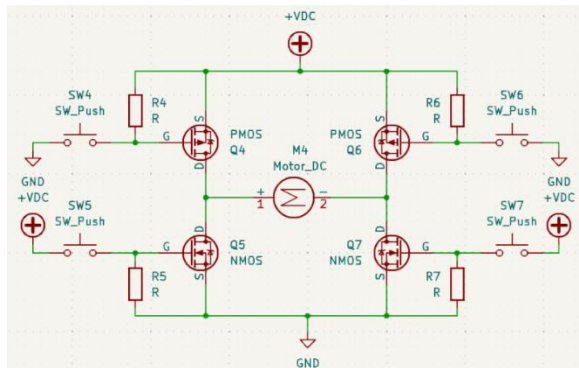


Figure 20 - H Bridge Circuit Diagram [1, p. 14], Fig. 7

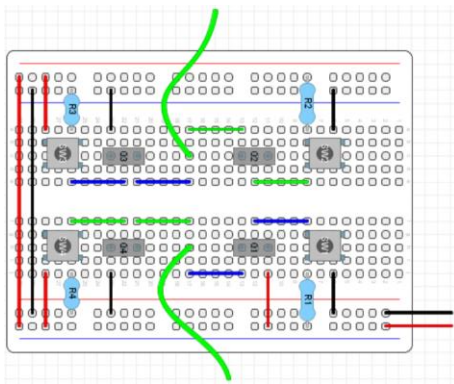


Figure 21 - H Bridge Breadboard Schematic

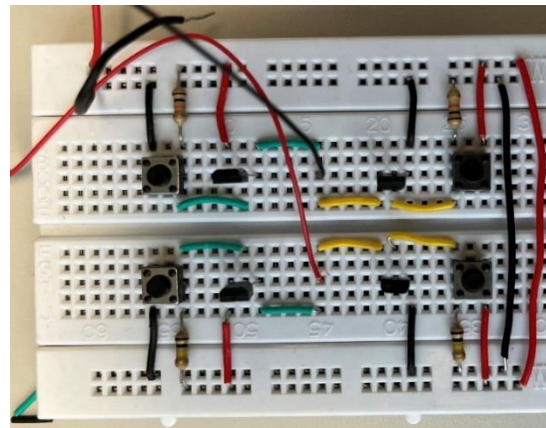


Figure 22 - H Bridge Breadboard Circuit

The H bridge is first planned and tested on a breadboard to ensure that I understood the components and how the circuit works before it is fixed onto a stripboard. The full H-bridge circuit contains 4 buttons with one per MOSFET; to complete the circuit and allow current to flow through the component – the motor or bi directional LED – two diagonal buttons – one for the PMOS and one for the NMOS – must be pressed simultaneously.

### 4.3 Stripboard Construction

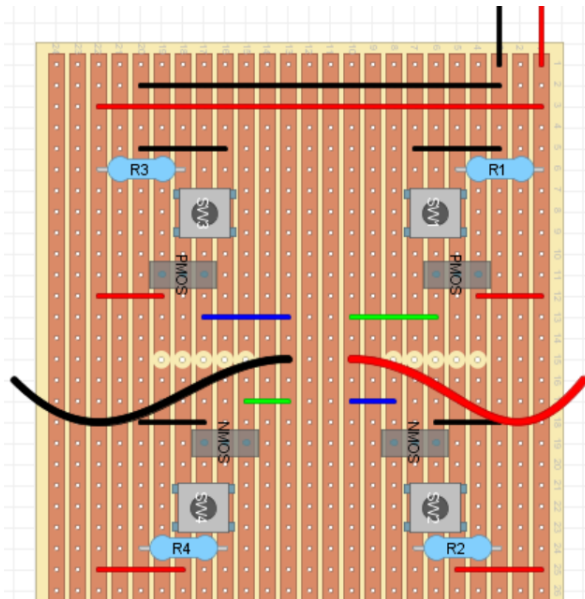


Figure 24 - H Bridge Stripboard Schematic

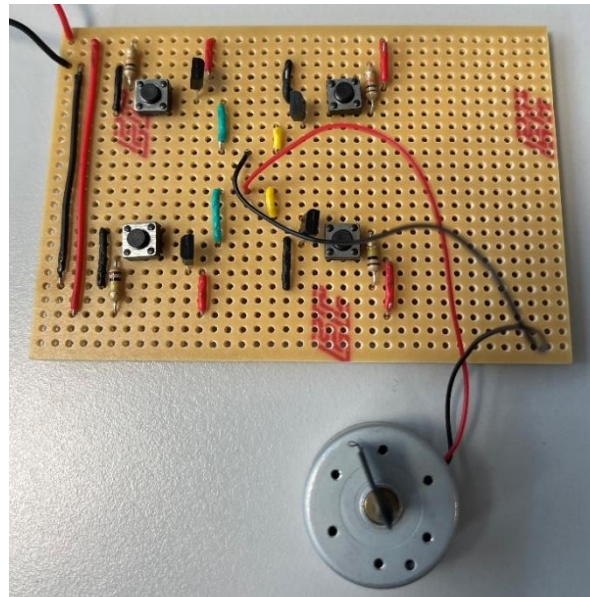


Figure 23 - H Bridge Stripboard Construction

The H bridge circuit is designed and constructed on the stripboard and tested to ensure proper working. The design is based on the one produced for the breadboard with minor adjustments to ensure it works on a stripboard. There were minimal difficulties during construction due to appropriate planning done beforehand.

## 5 Control Logic and Working Out

Controlling the H-bridge manually comes with the chance of accidentally pressing the wrong combination of buttons that will result in a short circuit. By implementing a logic circuit with two inputs, one representing direction and the other representing state – on or off – it will allow us to easily control the circuit without human error. Note that the PMOS (Q4 & Q6) require a low signal to activate while the NMOS (Q5 & Q7) needs a high signal to activate. This is important as we can represent a high signal as a binary 1 and a low signal as a binary 0. This will then allow us to use Boolean logic with physical logic gates. The working out is shown in the

## Working out

			P	N	P	N
Motor state	Direction	ON/OFF	Q4	Q5	Q6	Q7
OFF	0	0	1	0	1	0
on clockwise	0	1	0	0	1	1
OFF	1	0	1	0	1	0
on anti clockwise	1	1	1	1	0	0

Q4 & Q6 needs LOW to turn ON

Q5 & Q7 needs HIGH to turn ON

Q4 :  $\overline{B \cdot \bar{A}}$       Q6 =  $\overline{A \cdot B}$

Q5 =  $A \cdot B$       Q7 =  $\bar{A} \cdot B$  OR B

A = Direction (yellow)  
B = ON/OFF (blue)

Q4 =  $\begin{array}{c|c|c} A & B & Y \\ \hline 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array} \checkmark$       Q5 =  $\begin{array}{c|c|c} A & B & Y \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array} \checkmark$       Q6 =  $\begin{array}{c|c|c} A & B & Y \\ \hline 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array} \checkmark$       Q7 =  $\begin{array}{c|c|c} A & B & Y \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array} \checkmark$

OR

		P	N	P	N
Direction	ON/OFF	Q4	Q5	Q6	Q7
0	0	0 on	0 off	0 on	0 off
0	1	0 on	0 off	1 off	1 on
1	0	1 off	1 on	1 off	1 on
1	1	1 off	1 on	0 on	0 off

Q4 = D    Q5 = P    Q6 = D ⊕ P    Q7 = D ⊕ S

Figure 25 - H Bridge Circuit Control Logic Working Out



## 5.1 Control Logic Implementation

in the end I used the second – bottom – logic table as it required only 4 NAND gates to implement instead of 8. This means I would only need to use of chip instead of 2 and so conserving space. The logic used for the second table requires only one XOR gate which can be made using 4 NAND gates. NAND gates are used as they are a universal gate that can be used to make up any other gate – like shown in Figure 22.

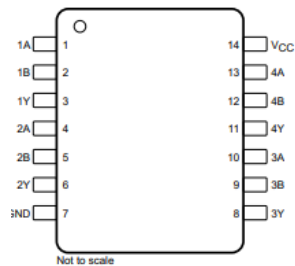


Figure 27 - NAND Chip Pinout [6, p. 3]

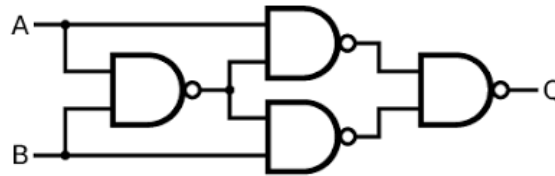


Figure 26 - XOR from NAND [7]

The chip used is the SN741LS00N chip that contains 4 NAND gates each with 2 inputs and 1 output.

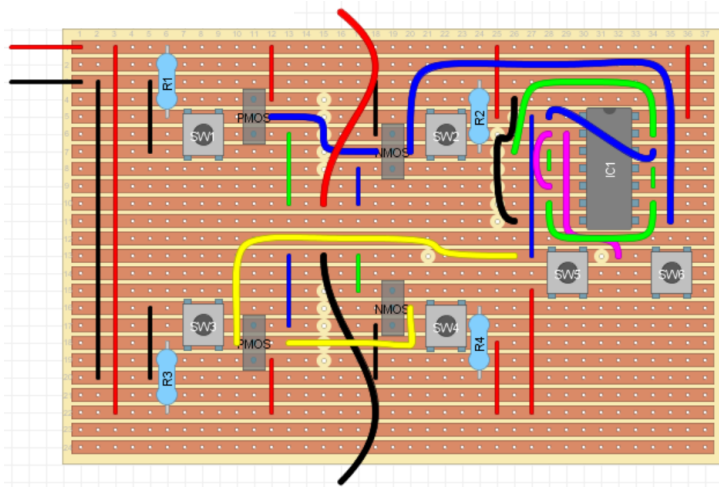


Figure 28 - Control Logic Stripboard Schematic

The logic was tested separately on a breadboard the soldered onto the existing stripboard using the plan in Figure 23.

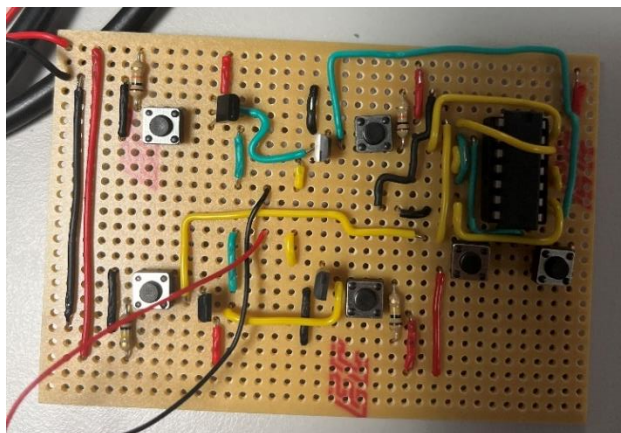


Figure 30 - H Bridge Control Logic Stripboard

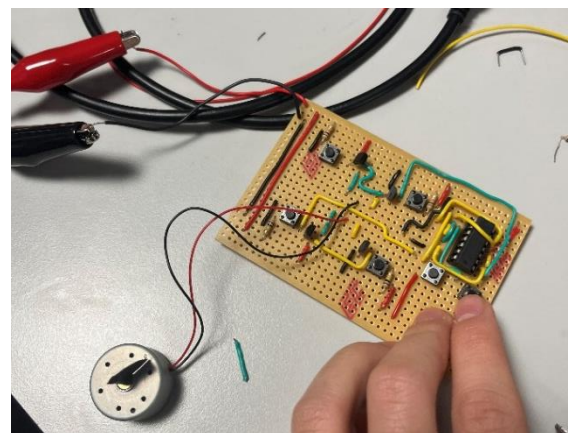


Figure 29 - H Bridge Control Testing



## 6.0 EEEBot

### 6.1 Introduction

The EEEBot is a robot vehicle based around the ESP32 micro controller. The robot consists of two rear driving wheels and two front steering wheels. It can be programmed in C and C++ using the Arduino IDE. A kit is provided which includes all necessary parts for constructing the EEEBot in week 3 of lab work.

### 6.2 Soldering Components to PCB

First part of assembly was to solder on the SMD components. It is better to solder on shorter components first as it can be held in place more easily. To do this, I referred to the EEEBot schematic that outlines how the components on the PCB is connected. On the schematic there are labels which corresponds to the pins / solder pads on the PCB, helping us know where each component is intended.

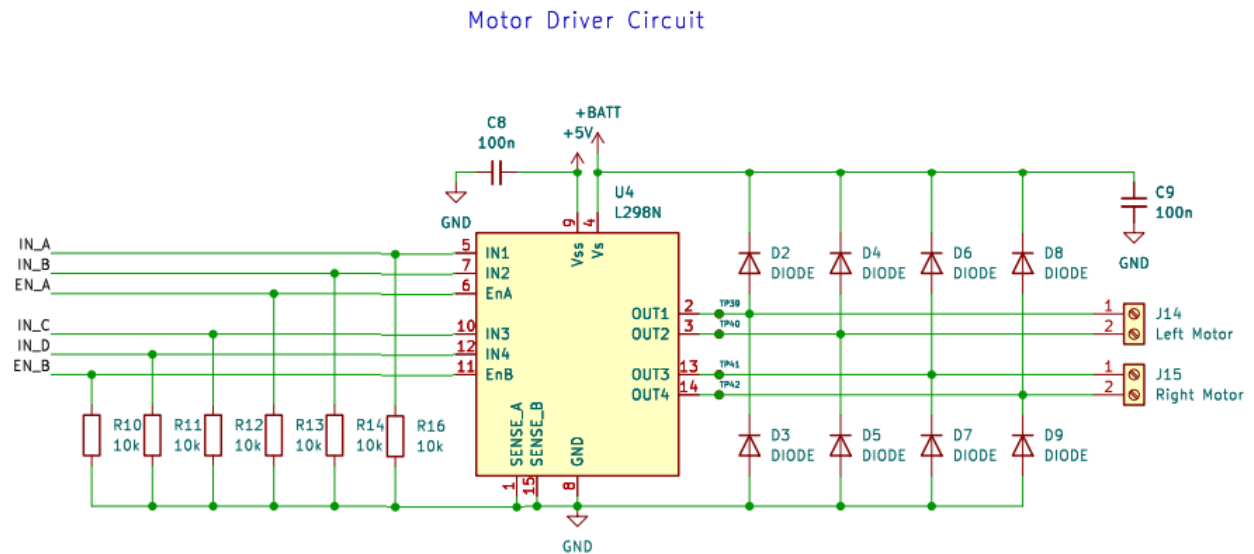


Figure 31 - Motor Driver Circuit Schematic [8]

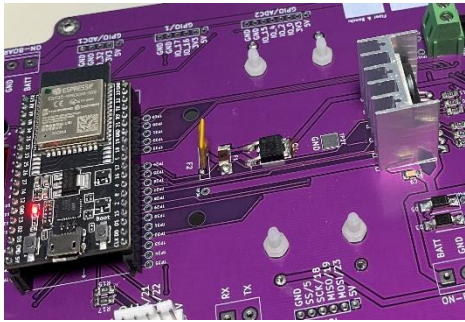


Figure 33 - EEEBot Progress Photo 1

After soldering all SMD components, I soldered all through hole components as well as the headers and the motor driver and installing the heat sink. This allowed me to connect the ESP32 microcontroller to the board.

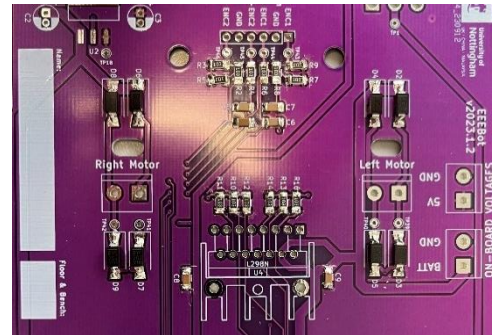


Figure 32 - EEEBot Progress Photo 2

### 6.3 Front Chassis Assembly

First, I connected the servo kit to the front chassis ensuring that the component is aligned correctly. There were some difficulties I ran into regarding this later as the screw hole was not aligned correctly. However, as the component as made by 3D printing, I used a soldering iron to expand the hole as solved the issue. Then, I attached the chassis to the board as shown in the diagram provided in the construction guide document – Figure 29.

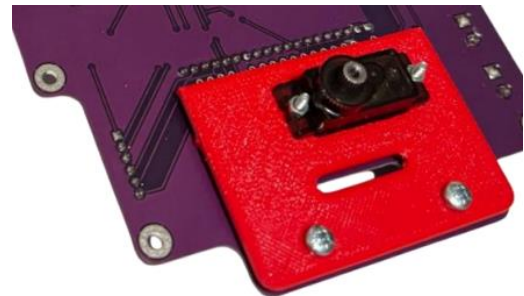


Figure 34 - Front Chassis Diagram [9, p. 13], Fig.16



Figure 35 - Front Steering Chassis Diagram [9, p. 14], Fig. 18

Next, I attached the front steering chassis to the bracket provided in servo kit using 2 M2.2 6.5mm self-tapping screws. Then, I connected the bracket to the servo kit using a M2.5 3.5mm panhead screw which was provided in the servo kit. Lastly, insert the front wheels into the side of the front steering chassis and use the M3 locknut to hold in place. Use the allen key to tighten the locknut and secure the wheel in the front steering chassis. Once finished, it looks like the image provided in the construction guide – in Figure 30.

## 6.4 Rear Chassis Assembly

First I soldered cables onto the encoder PCB using the 6 center pins. Then I attached the rotary encoder to the encoder PCB by soldering on the other side. Next, I attached the rear chassis to the main PCB using 2 M3 12mm panhead screws, 2 M3 full nuts and 2 rear chassis spacers by placing the spacer between the main PCB and rear chassis. This will then allow us to attach the encoder PCB to the rear chassis and plug the wires into the 6-way female header connected on the end of the main PCB.

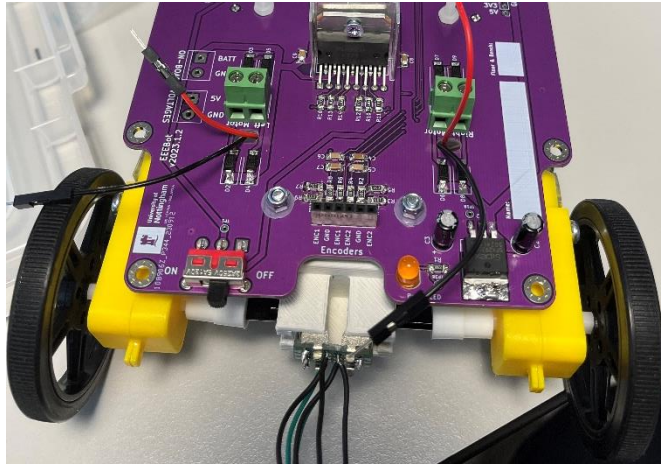


Figure 36 - EEEBot Progress Photo 3

Then connect the encoder bush to the ends of the rotary encoders. This works as an adapter to allow the rotary encoder to the motor. Next, I connected the motor to the rear chassis using 2 M3 25mm panhead screws and 2 M3 full nut for each motor. Once finished, the motor should align with the rotary encoder push and so the rotary encoder can get information from the motor. Route the motor wires through the holes in the main PCB and plug into the 2 way terminal block on each side and the result is shown in Figure 31.

## 6.5 Final Steps

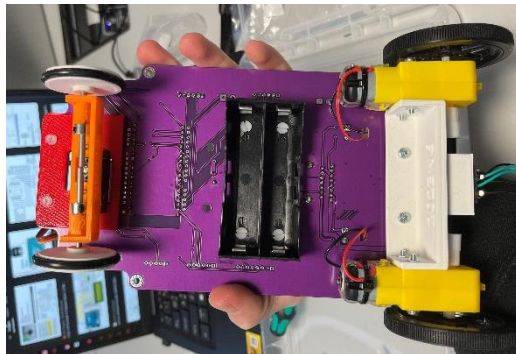


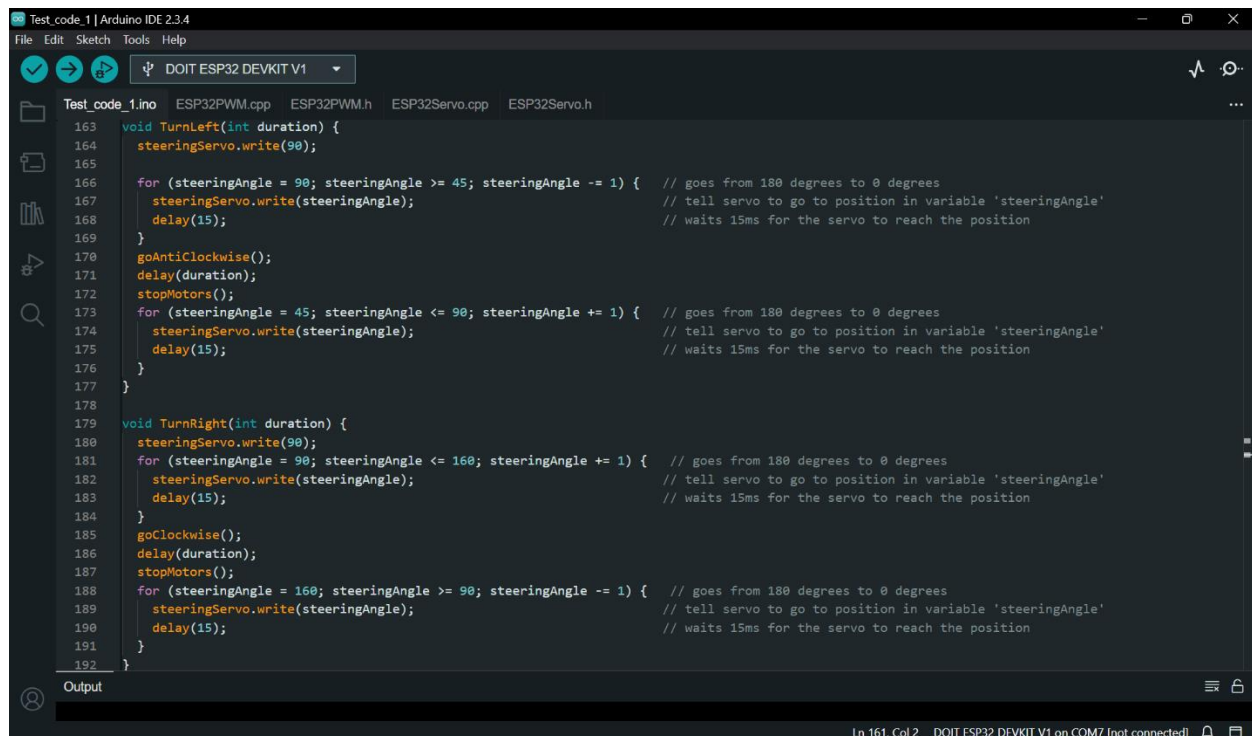
Figure 37 - EEEBot Progress Photo 4

Lastly, attach the battery holder to the main PCB using 4 M3 10mm panhead screws and 4 M3 full nuts. Then, insert the 2 batteries ensuring they are oriented correctly. To check, turn on the EEEBot by flipping the power switch and the LED on the board and on the ESP32 should light up. To further check connections, use a multimeter to check voltage across all of the components by touching one probe to the point and the other to ground and if there is voltage, a reading will show on the multimeter.

## 7 Programming

### 7.1 Initial Testing

The EEEBot is controlled using the ESP32 Devkit microcontroller that can be programmed using C in the arduino IDE. To program the ESP32 it, first connect it to a laptop so that we can upload code using a USB A to micro USB cable. Then, upload the test code to that it works as intended and that the EEEBot is connected properly. During this stage I ran into problems with missing drivers however, that was solved very swiftly as the driver can be found easily online. Adding the file to my project folder allowed the program to run without issues.

The image is a screenshot of the Arduino IDE 2.3.4 interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu bar, there are icons for checking, uploading, and erasing code, followed by a dropdown menu showing 'DOIT ESP32 DEVKIT V1'. The main workspace displays a file named 'Test\_code\_1.ino' with the following code:

```
163 void TurnLeft(int duration) {
164   steeringServo.write(90);
165
166   for (steeringAngle = 90; steeringAngle >= 45; steeringAngle -= 1) { // goes from 180 degrees to 0 degrees
167     steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
168     delay(15); // waits 15ms for the servo to reach the position
169   }
170   goAntiClockwise();
171   delay(duration);
172   stopMotors();
173   for (steeringAngle = 45; steeringAngle <= 90; steeringAngle += 1) { // goes from 180 degrees to 0 degrees
174     steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
175     delay(15); // waits 15ms for the servo to reach the position
176   }
177 }
178
179 void TurnRight(int duration) {
180   steeringServo.write(90);
181   for (steeringAngle = 90; steeringAngle <= 160; steeringAngle += 1) { // goes from 180 degrees to 0 degrees
182     steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
183     delay(15); // waits 15ms for the servo to reach the position
184   }
185   goClockwise();
186   delay(duration);
187   stopMotors();
188   for (steeringAngle = 160; steeringAngle >= 90; steeringAngle -= 1) { // goes from 180 degrees to 0 degrees
189     steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
190     delay(15); // waits 15ms for the servo to reach the position
191   }
192 }
```

The bottom status bar indicates 'Ln 161, Col 2 DOIT ESP32 DEVKIT V1 on COM7 [not connected]'.

Figure 38 - TurnLeft and TurnRight Function

To help understand how the code interacts with the EEEBot, I first wrote simple programs such as turn left, turn right and turn around shown in Figure 33 above.

### 7.2 10 Meter Challenge

The main task for this week was for the car to travel as close as possible to exactly 10 meters. To do this, the EEEBot can be programmed to travel for a set amount of time so that it would stop at exact 10 meters. The main hurdle to this challenge however was not finding out the time taken to travel 10 meters but keeping the EEEBot in a straight line. This is because the front wheels are not perpendicular to the ground and so causes the EEEBot to slowly veer to the side – predominantly left. This meant that the car needed to correct its own course at certain intervals to try and maintain a straight path overall.

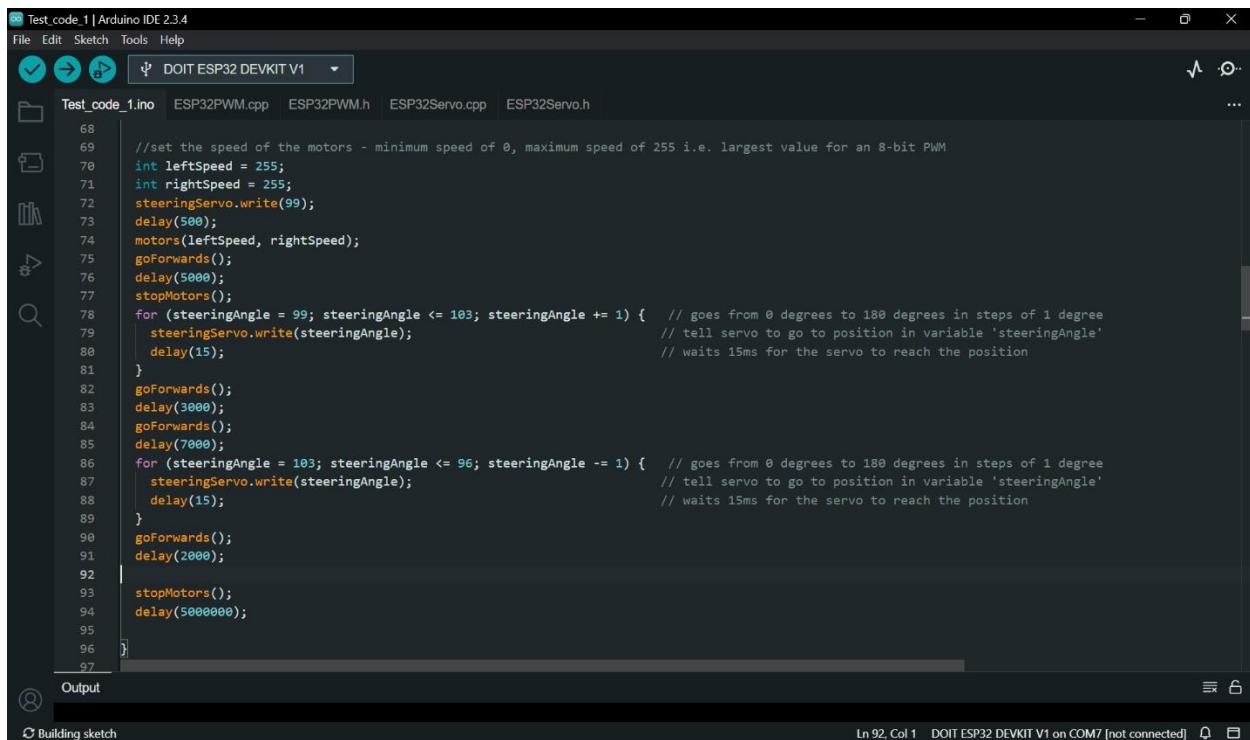
My solution to this was to create a function to reset the steering, getting the front wheels as close as mechanically possible to being straight.

```
121 void ResetSteering() {
122     steeringServo.write(90);
123     delay(200);
124
125     for (steeringAngle = 90; steeringAngle > 0; steeringAngle -= 1) { // goes from 180 degrees to 0 degrees
126         steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
127         delay(15); // waits 15ms for the servo to reach the position
128     }
129     delay(500);
130     for (steeringAngle = 0; steeringAngle <= 200; steeringAngle += 1) { // goes from 0 degrees to 180 degrees in steps of 1 degree
131         steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
132         delay(15); // waits 15ms for the servo to reach the position
133     }
134     delay(500);
135     for (steeringAngle = 200; steeringAngle >= 100; steeringAngle -= 1) { // goes from 180 degrees to 0 degrees
136         steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
137         delay(15); // waits 15ms for the servo to reach the position
138     }
139 }
140
```

*Figure 39 - ResetSteering Function*

This function is called once each time the EEEBot is turned on. The idea is that by getting as close to straight as possible it will continue in a straight line for longer. However, it would continue to deviate from its intended path.

My second solution was that about half way, the EEEBot will turn slightly left to correct its course. This is because during testing I noticed that the EEEBot will always veer left. When tested in the lab, this solution worked well however there was a problem of consistency as how much the EEEBot will deviate is not the same every test. In the end, I produced a result I felt satisfactory while testing in the lab using the program seen below in Figure 34.



```
Test_code_1.ino ESP32PWM.cpp ESP32PWM.h ESP32Servo.cpp ESP32Servo.h
68
69 //set the speed of the motors - minimum speed of 0, maximum speed of 255 i.e. largest value for an 8-bit PWM
70 int leftSpeed = 255;
71 int rightSpeed = 255;
72 steeringServo.write(99);
73 delay(500);
74 motors(leftSpeed, rightSpeed);
75 goForwards();
76 delay(5000);
77 stopMotors();
78 for (steeringAngle = 99; steeringAngle <= 103; steeringAngle += 1) { // goes from 0 degrees to 180 degrees in steps of 1 degree
79   steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
80   delay(15); // waits 15ms for the servo to reach the position
81 }
82 goForwards();
83 delay(3000);
84 goForwards();
85 delay(7000);
86 for (steeringAngle = 103; steeringAngle <= 96; steeringAngle -= 1) { // goes from 0 degrees to 180 degrees in steps of 1 degree
87   steeringServo.write(steeringAngle); // tell servo to go to position in variable 'steeringAngle'
88   delay(15); // waits 15ms for the servo to reach the position
89 }
90 goForwards();
91 delay(2000);
92
93 stopMotors();
94 delay(5000000);
95
96
97
Output
Building sketch Ln 92, Col 1 DOIT ESP32 DEVKIT V1 on COM7 [not connected]
```

Figure 40 - 10 Meters Main Code

During the final day, the venue had uneven floor which was not taken into account for when programming as well as other EEEBots which would interfere with my own. During my 10 meters test, the EEEBot was obstructed by another bot that left its lane and crashed into mine resulting in the EEEBot stopping roughly 2 meters before the 10 meters mark. While the result was not as expected, the EEEBot did perform its function of moving forward in a straight line which can be counted as a success to some degree.



## 8.0 Conclusion

Through the lab weeks, the circuits I tried to produce all worked however not all exactly as intended. The OP-AMP circuit produced an unexpected output as the negative values clipped however after I found out why this occurred. The 555-timer monostable worked as intended and helped me learn to adapt on the go with the available resources. The H-bridge was relatively simple as during the week there were tasks to build simpler circuits that worked up to the H-bridge, guiding me through its construction in a very step by step manner. This led to minimal issues during this section. The control logic for the H-bridge however required more time and planning as the initial logic that was derived was much more complicated and difficult to implement. This led to me changing the design and using the much simpler logic equation as shown. The logic circuit implementation was also difficult as it there were difficulties during testing due to poor breadboard connections. Lastly, the construction of the EEEBot in week 3 was straight forward due to the labeling of the main PCB as well as the schematic provided meaning the main skills test – for me personally – was my ability to solder. The programming side of this was also simple as a template was provided with the main issue being the skewing of the front wheels. Overall, the lab weeks were a success as all assigned work was completed on time and to the best of my abilities.

## References

- [1] Department of Electrical and Electronic Engineering, "EEEE1002: Project Week 2 H-Bridge," University of Nottingham, Nottingham, 2024.
- [2] Diodes Incorporated, "ZVP2106A P-CHANNEL ENHANCEMENT MODE VERTICAL DMOS FET," 1994.
- [3] Diodes Incorporated, "ZVN4306A N-CHANNEL ENHANCEMENT MODE VERTICAL DMOS FET IN E-LINE," 2012.
- [4] Department of Electrical and Electronic Engineering, "EEEE1002: Project Week 1 Skills," University of Nottingham, Nottingham, 2024.
- [5] Texas Instruments, "LM555 Timer Datasheet (Rev. D)," 2000.
- [6] Texas Incorporated, "SNx400, SNx4LS00, and SNx4S00 Quadruple 2-Input Positive-NAND Gates," 2017.
- [7] U. Inductiveload, "File:XOR from NAND.svg," WIKIPEDIA, 2009. [Online]. Available: [https://en.m.wikipedia.org/wiki/File:XOR\\_from\\_NAND.svg](https://en.m.wikipedia.org/wiki/File:XOR_from_NAND.svg). [Accessed 12 2024].
- [8] N. Dacombe, "EEEBot Mainboard Schematic," University of Nottingham, 2023.
- [9] Department of Electronic and Electrical Engineering, "EEEE1002: EEEBot Construction and Testing Guide," University of Nottingham, 2024.