

# Milestone 4 Report

Authors: Kian Frassek, Cameron Clark and Thiyashan Pillay

## Table of Contents

1	Introduction: .....	3
2	Summary of Acceptance Test Procedure: .....	4
3	System level design: Overview of design concepts and justification for choices.....	5
4	Electrical design: .....	10
4.1	Power calculations and Power supply design and schematics (including safety) .....	10
4.2	Microcontroller resource allocation.....	12
4.3	Design and Schematics of circuits with Vero/PCB layout.....	13
4.3.1	Schematics .....	13
4.3.2	PCB Layout.....	16
5	Motion control design, implementation, and results Include.....	17
5.1	Motion control design to move forward 1m.....	17
5.1.1	Pseudocode functions .....	17
5.1.2	UML or flow diagrams of code or MATLAB Simulink layout.....	17
5.1.3	Results and justifications.....	18
5.2	Motion control design for rotation .....	18
5.2.1	Pseudocode functions .....	18
5.2.2	UML or flow diagrams of code or MATLAB Simulink layout.....	19
5.2.3	Results and justifications.....	19
6	Line sensing design, implementation, and results Include. ....	20
6.1	UML or flow diagrams of code or MATLAB Simulink layout.....	20
6.2	Results and justifications.....	20
7	Treasure Maze solving algorithm design, implementation, and results. ....	21
7.1	UML diagrams (flow diagram of code).....	21
7.2	Algorithm explanation.....	21
7.3	Pseudocode functions .....	22
8	Conclusion.....	23

## 1 Introduction:

A line follower robot is a common robotic task where a robot with a simple drive system, sensors and a microcontroller are tasked to follow a line and sometimes complete tasks along the way. In this iteration of a line follower robot design challenge, there is a treasure hunt taking place. The goal is for the robot to take a twisting path with intersections and dead ends, stopping at all the specified measurement lines to measure the distance to the treasure objects using an ultrasonic sensor, and finally stopping at a specified black rectangle as fast as possible.

The provided equipment includes the chassis (the structure, wheels, motors, and motor drivers), the sensors (five-line sensors, two motor rotation sensors, one ultrasonic sensor), the batteries (with power switches) and finally the microcontroller. Our goal is to regulate the two 18650 batteries to a stable 5V to power the onboard equipment as well as the coding the brain of the operation: the Arduino Nano.

This document is an in-depth analysis of the evaluation of the performance of the robot and whether or not it has met the required objectives set out in the beginning.

## 2 Summary of Acceptance Test Procedure:

In milestone 1, there was a list of merits on which the final design was tested against. The table is shown below:

<i>Table 1: Acceptance Test Procedure</i>	
Power On	Validation that the robot powers on and initialises with no errors
Interconnectivity	Make sure the subsystems and components interact with one another efficiently.
Movement	The robot must move without any human intervention.
Speed and turning control	Turning and correcting within 10% error. There must be minimal overshoot when navigating through turns.
Line following	Robot can follow a line of varying winding without completely driving off.
Intersection resolution	Robot must identify all intersection correctly and document all possible paths
Path mapping	Robot must map the path accurately, documenting the routes during navigation.
Stop and measure distances	The robot should stop at measuring points and record the distance to objects.
Termination	Movement stopped when reaching the stop zone only if all paths have been explored.
Party mode	This is a mode at the end that is executed once the maze is completed.
Size of Veroboard	Should not exceed $10cm \times 10cm$ .

This table helps drive design decisions in the right direction, to achieve these objectives. This is explained in more detail in the next section.

One important factor is slip. The friction of the surface upon which the robot is moving would dictate the success of the project. If the robot is too quick, there will be an increased risk of slipping on the surface, which may cause the robot to overshoot its marks. If the robot is too slow, significant time will be sacrificed despite the robot completing the maze correctly.

### 3 System level design: Overview of design concepts and justification for choices.

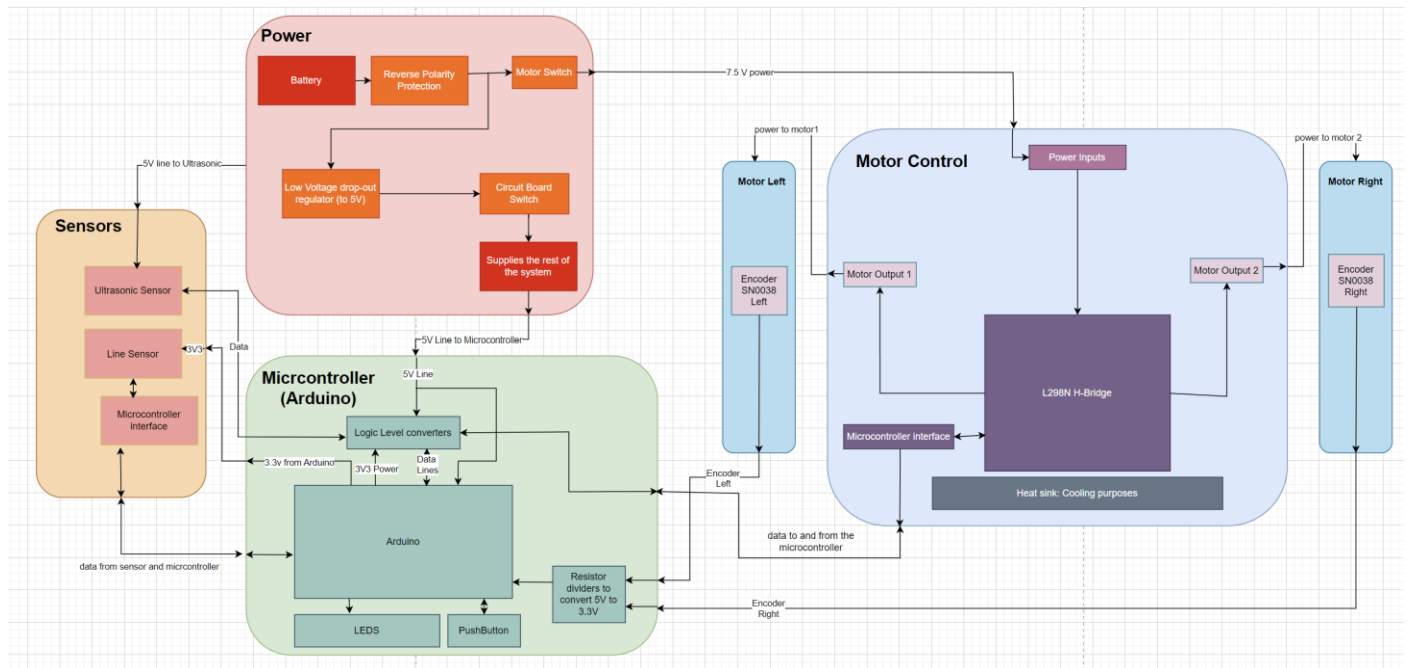


Figure 1: Block diagram of Robot

Table 2: Design Decisions associated with the block diagram

Design Decisions:	
Power	<p>Reverse Polarity Protection: Ensures that the batteries are connected in the correct orientation. If they are reversed, no power will be drawn.</p> <p>Motor Switch: Controls power to the motors. Saves power.</p> <p>Low Voltage Dropout Regulator (LDO): Provides a more stable 5V source for the Arduino, as well as other circuit components.</p> <p>Circuit Board Switch: Allows us to flash code onto the Arduino safely.</p>
Microcontroller	<p>Logic Level Converters: Connected to the H-Bridge and the Ultrasonic Sensors.</p> <p>Resistor Dividers: Due to a shortage of ports on the Logic Level converters, more resistor dividers were created for the encoders. This is a single direction connection, so resistor dividers work well.</p> <p>Arduino Nano 33 LoT: Constraint of the system was to use this microcontroller.</p> <p>LEDS: Used as the detect LED for when the Ultrasonic sensor is active.</p> <p>Switch: Helps with debugging.</p>
Sensors	<p>Ultrasonic Sensor: Supplied component.</p> <p>Line Sensor: Supplied component. Limited to 4 sensors.</p>
Motor Control	<p>H-Bridge: Supplied component.</p> <p>Encoders: Supplied component.</p> <p>Motor (Right and Left): Supplied component.</p>

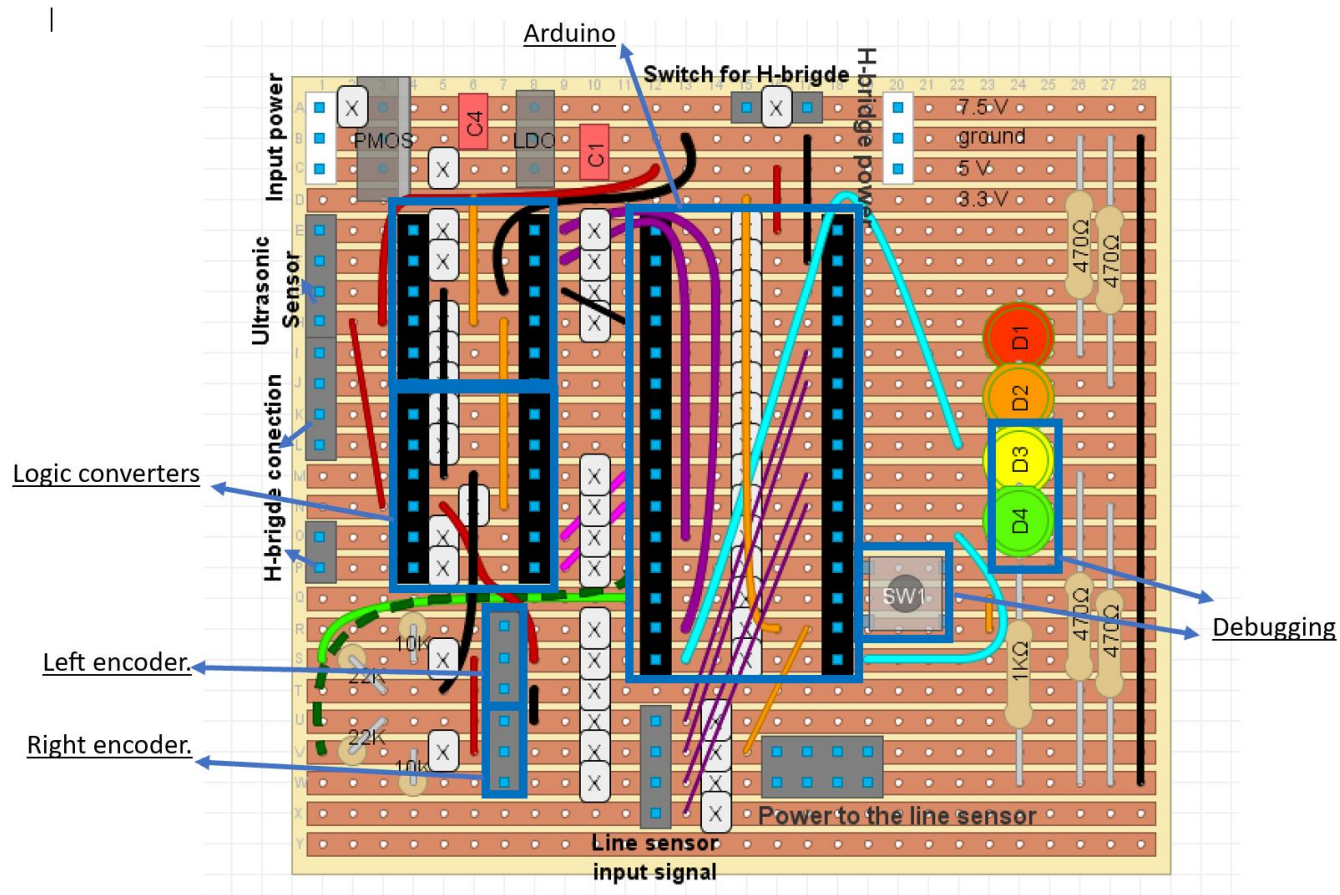


Figure 2: Schematic of the PCB layout (Key below)

Light blue	LED to Arduino connection
purple	Line sensor to Arduino and ultrasonic
pink	Arduino to logic level converters (for H-bridge connection)
Green (light and dark)	Use to connect the Arduino to the encoders.
Black	Ground or 0V
Dark red	5V from LDO
Orange	3.3V from Arduino

Figure 3: Key for figure 2

Table 3: Design decisions associated with the PCB layout.

Design Decisions:	
Power	<ul style="list-style-type: none"> <li>Unidirectional power connector used for a stable connection.</li> <li>PMOS used for reverse polarity protection.</li> <li>C1 and C4 capacitors used as a stabiliser for power signals.</li> <li>LDO used to regulate input power from batteries to 5V. The voltage from the battery is a maximum of 7.4V, so this does need to be stepped down.</li> <li>Connection for the switch to the H-Bridge: Unidirectional power connector to motor subunit.</li> <li>Power to the line sensor: 2 × 4 male connectors used to supply power to the line sensors.</li> </ul>
Microcontroller	<ul style="list-style-type: none"> <li>X2 1 × 15 female headers are used to mount the Arduino.</li> <li>Each logic level converter has X2 1 × 6 female headers.</li> <li>Power lines run beneath the logic level converters. Helps improve neatness.</li> <li>Yellow LED indicates that the ultrasonic sensor is active.</li> <li>Green LED indicates when we are within 15cm of the object.</li> <li>Pushbutton used for debugging purposes.</li> </ul>

Sensors	<ul style="list-style-type: none"><li>• The line sensors are connected via a <math>4 \times 1</math> male headers that is connected to the digital pins of the Arduino. Allows for seamless connection of the data lines of the line sensors.</li><li>• Left and Right encoder are connected via <math>3 \times 1</math> male headers, with the bottom and top lines being power lines and the middle going through a resistor divider circuit which then connects to the Arduino. Allow for easy connection to the Arduino.</li><li>• Ultrasonic sensor is connected via a <math>4 \times 1</math> male connectors. Done to allow easy data transfer through the logic level converters.</li></ul>
Motor Control	<ul style="list-style-type: none"><li>• H-Bridge controls the motors. Connected via a <math>4 \times 1</math> and <math>2 \times 1</math> male connector. For the <math>4 \times 1</math> male connector, Pin 1: Input for direction control for motor left. Pin 2: Input for direction control for motor left. Pin 3: Enable used to send PWM signal for motor left. Pin 4: Enable used to send PWM signal for motor right. For the <math>2 \times 1</math> male connector, Pin 1: Input for direction control for motor right. Pin 2: Input for direction control for motor right.</li></ul>



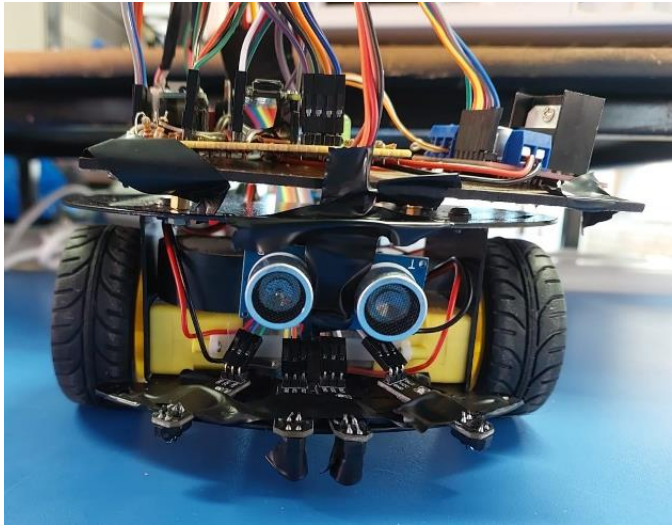
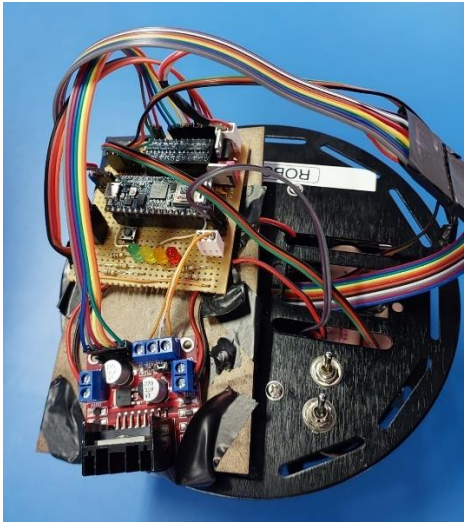
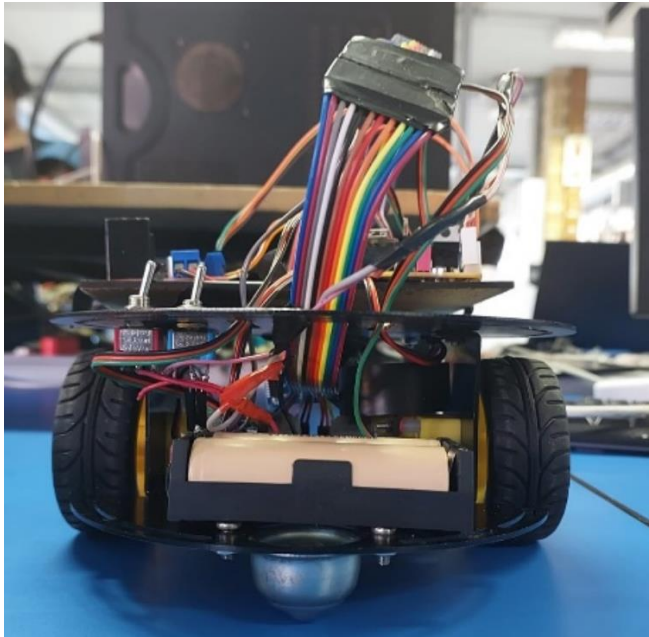
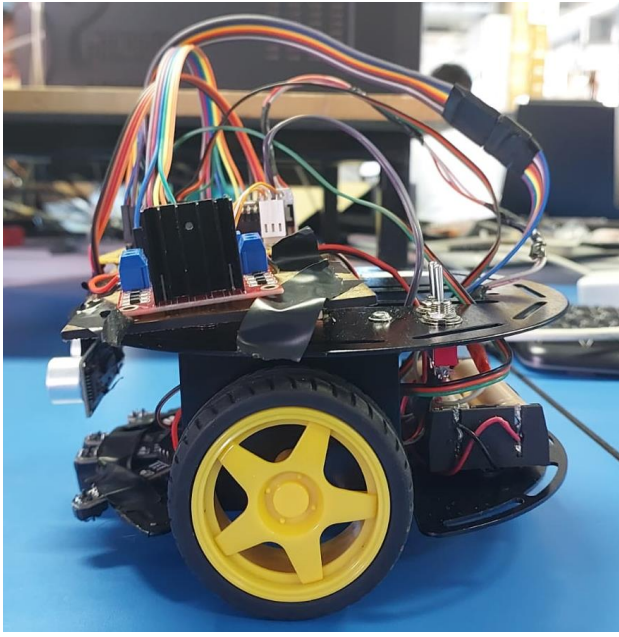
Table 4: Images of the robot from different perspectives.	
Front View: 	Top View: 
Rear View: 	Side View 
Bottom View:	

Figure 4: Front View

Figure 5: Top View

Figure 6: Rear View

Figure 7: Side View



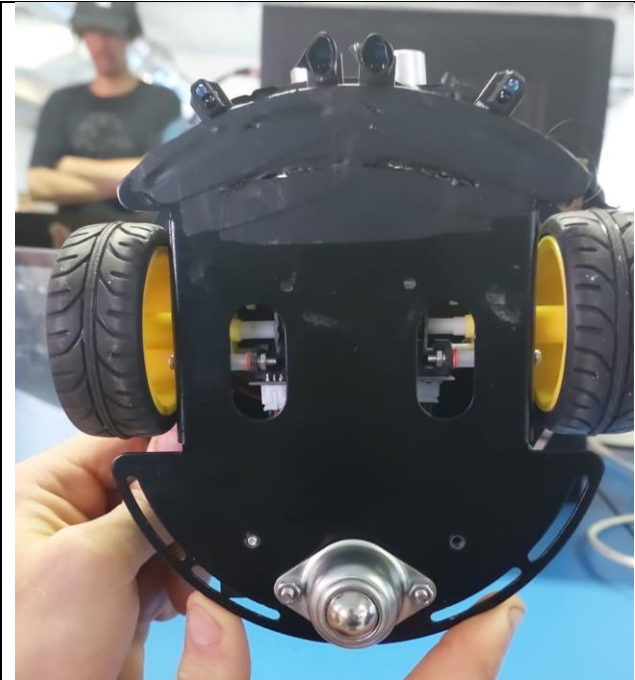


Figure 8: Bottom View

Table 5: Design decisions associated with the placement of components on the robot.	
Design Decisions:	
Front View	Ultrasonic sensor and line sensors placed facing forward for optimal driving control.
Top View	Wooden board used to isolate circuitry from the metal chassis. Insulation tape used to cover any exposed metal that can potentially short the board. All wires seen are used to connect motor drivers, power circuitry and sensors to the main circuit.
All other views show prefabricated parts that were supplied. These were pre-built.	

## 4 Electrical design:

### 4.1 Power calculations and Power supply design and schematics (including safety)

Power subsystem:

Component	Supply Voltage (Nominal V)	Minimum Current (mA)	Typical Current (mA)	Maximum Current (mA)	Minimum Power (mW)	Typical Power (mW)	Maximum Power (mW)
PMOSFET IRf9530n	0,083	270	415	600	22,41	34,445	49,8
LDO LM3940	3	270	316	330	810	948	990
C1	8	N/A	0	N/A	N/A	0	N/A
C2	8	N/A	0	N/A	N/A	0	N/A
Total					832,41	982,445	1039,8

Figure 9: Power calculations for the power subsystem

Microcontroller subsystem:

Component	Supply Voltage (Nominal V) min	Supply Voltage (Nominal V) mid	Supply Voltage (Nominal V) max	Minimum Current (mA)	Typical Current (mA)	Maximum Current (mA)	Minimum Power (mW)	Typical Power (mW)	Maximum Power (mW)
Arduino Nano 33 IoT	5	5	5	120	131	200	600	655	1000
Logic Level Converter max	0,003	3,3	5	0,5	2	4	0,0015	6,6	20
Total		8,3	10	120,5	133	204	600,0015	661,6	1020

Figure 10: Power calculations for microcontroller subsystem

H-bridged subsystem:

Component	Supply Voltage (Nominal V) min	Minimum Current (mA)	Typical Current (mA)	Maximum Current (mA)	Minimum Power (mW)	Typical Power (mW)	Maximum Power (mW)
H-bridge	4,9	8	112	200	39,2	548,8	980
Total	4,9	8	112	200	39,2	548,8	980

Figure 11: Power calculations for H-Bridge subsystem

Component	Supply Voltage (Nominal V) min	Minimum Current (mA)	Typical Current (mA)	Maximum Current (mA)	Minimum Power (mW)	Typical Power (mW)	Maximum Power (mW)
motor left	5	20	40	200	100	200	1000
motor right	5	20	40	200	100	200	1000
Total	5	40	80	400	200	400	2000

Figure 12: Power calculations for motors subsystem

Sensor subsystem:

Component	Supply Voltage (Nominal V) min	Minimum Current (mA)	Typical Current (mA)	Maximum Current (mA)	Minimum Power (mW)	Typical Power (mW)	Maximum Power (mW)
Encoder ×2	5	5	36	50	25	180	250
line senser × 4	3,3	5	25	40	16,5	82,5	132
ultrasonic	5	2	6	25	10	30	125
total	5	12	67	115	51,5	292,5	507

Figure 13: Power calculations for sensor subsystem

Total system:

Component	Supply Voltage (Nominal V) min	Minimum Current (mA)	Typical Current (mA)	Maximum Current (mA)	Minimum Power (mW)	Typical Power (mW)	Maximum Power (mW)
Power subsystem	8	540	731	930	832,41	982,445	1039,8
Microcontroller subsystem	5	120,5	133	204	600,0015	661,6	1020
H-bridged subsystem	4,9	8	112	200	39,2	548,8	980
Motors subsystem	5	40	80	400	200	400	2000
Sensor's subsystem	5	12	67	115	51,5	292,5	507
Total		660,5	864	1134	1432,412	1644,045	2059,8

Figure 14: Power calculations for total subsystem

## 4.2 Microcontroller resource allocation



**ARDUINO**  
**NANO 33 IoT**

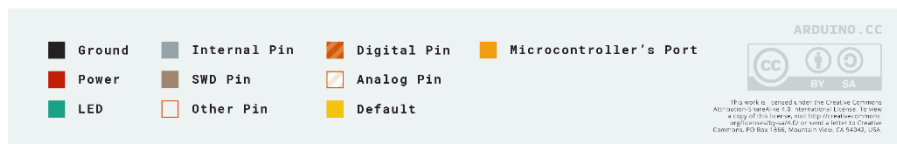
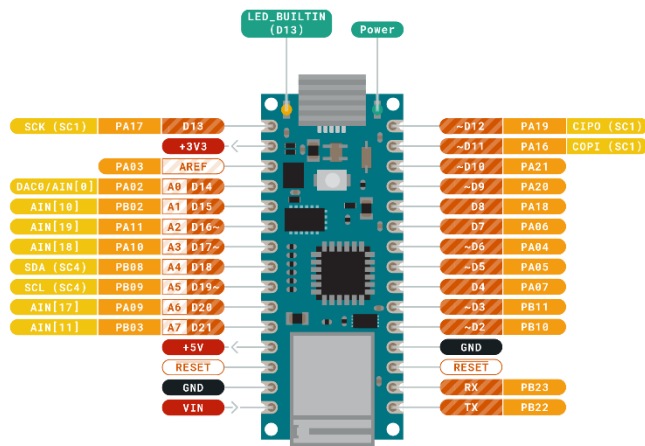


Figure 15: Schematic of the Arduino Nano 33 LoT

(Arduino, 2023)

Pin left	Pin map	pin right	Pin map
Pin 1 (Tx)	Not connected	pin 16 V in	Power
Pin 2 (Rx)	Not connected	pin 17 GRN	GRN
Pin 3 reset	Not connected	pin 18 Reset	Not connected
Pin 4 GRN	GRN	pin 19 5V	Not connected
Pin 5 D2	H-Bridge	pin 20 A7	Line Sensor
pin 6 D3	H-Bridge	pin 21 A6	Line Sensor
pin 7 D4	H-Bridge	pin 22 A5	Not connected
pin 8 D5	H-Bridge	pin 23 A4	Not connected
pin 9 D6	H-Bridge	pin 24 A3	Line Sensor
pin 10 D7	H-Bridge	pin 25 A2	Line Sensor
pin 11 D8	Ultrasonic	pin 26 A1	Not connected

pin 12 D9	Rotary Encoders	pin 27 A0	Not connected
pin 13 D10	Rotary Encoders	pin 28 AREF	3.3V
D13 D11	Ultrasonic	pin 29 3.3v	3.3V to line sensor and encoders
pin 15 D12	Yellow LED	pin 30 D13	Green LED

Figure 16: Pin Allocation to Arduino Nano 33 LoT

### 4.3 Design and Schematics of circuits with Vero/PCB layout

#### 4.3.1 Schematics

Main schematics:

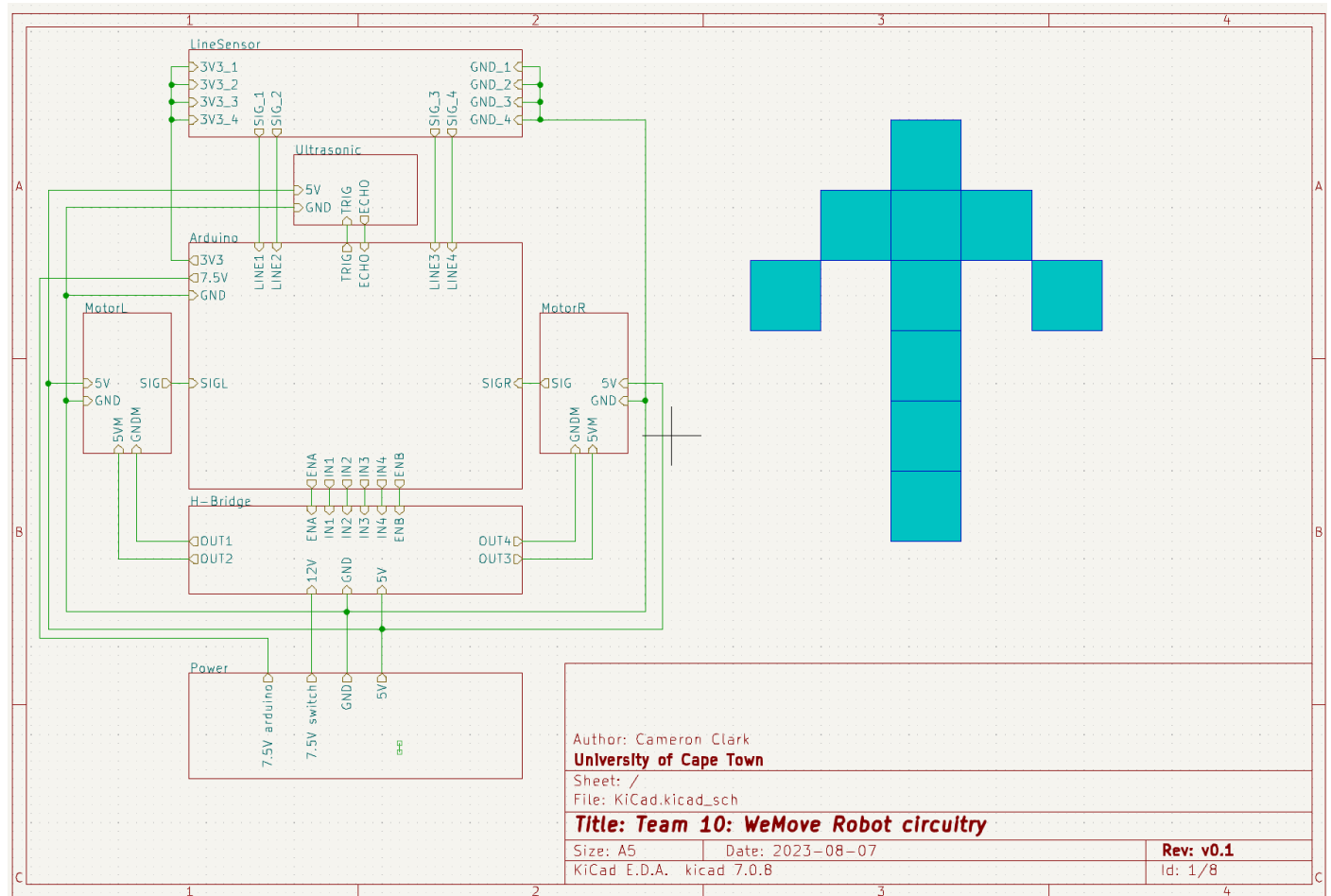


Figure 17: Main Schematic

Power subsystem:

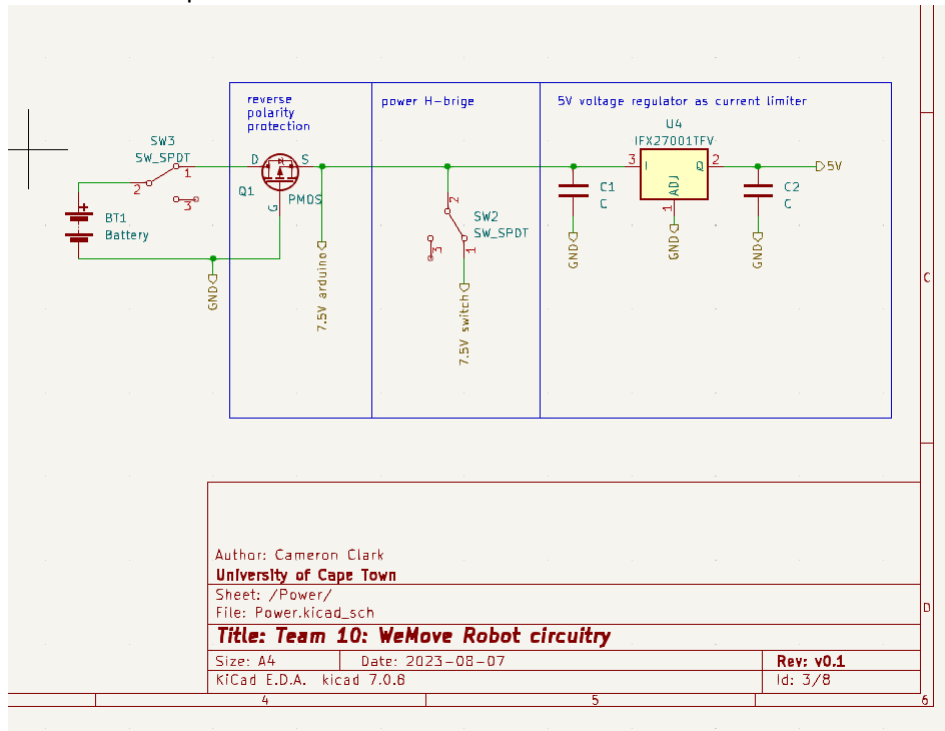


Figure 18: Power subsystem schematic

Microcontroller subsystem:

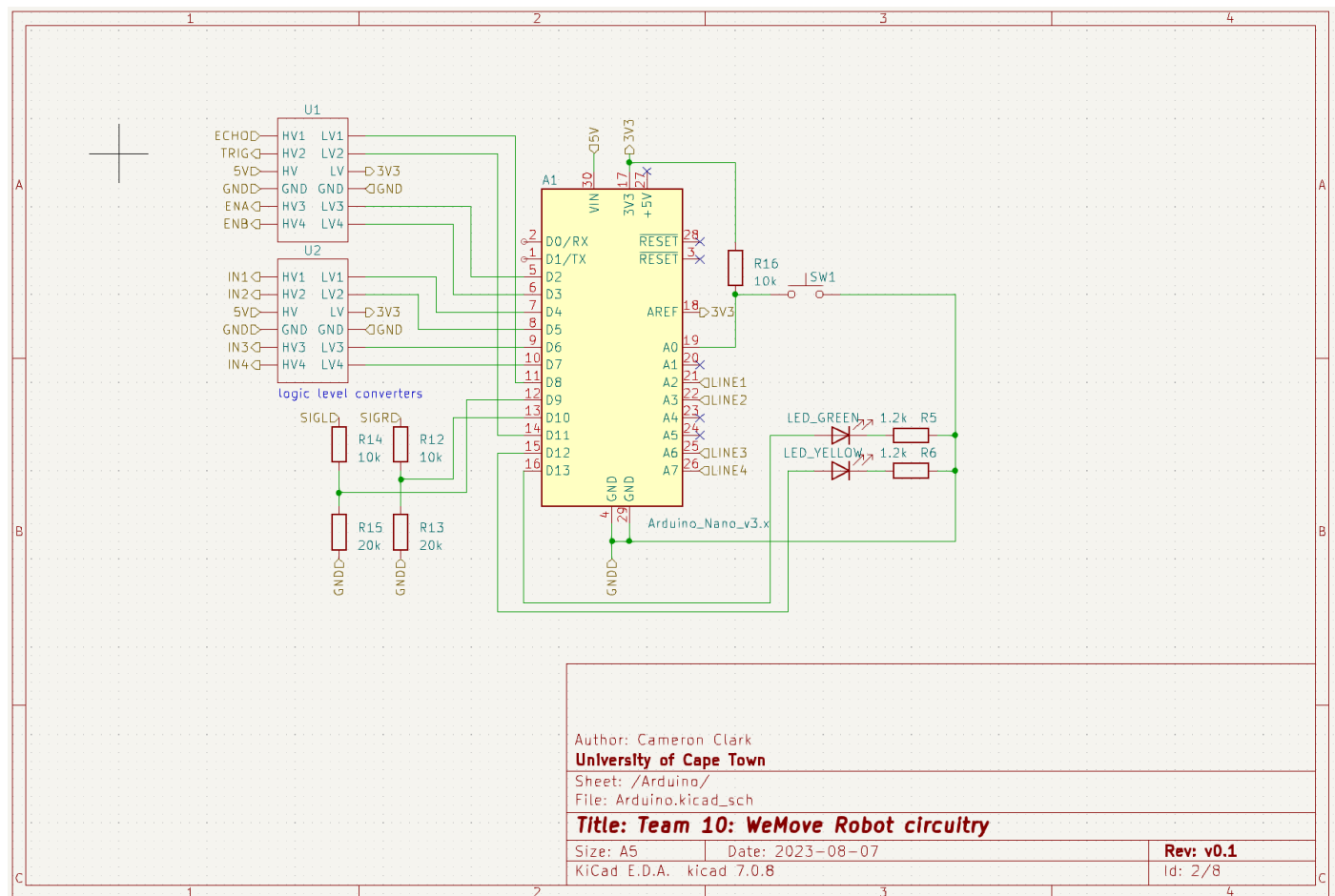


Figure 19: Microcontroller subsystem schematic

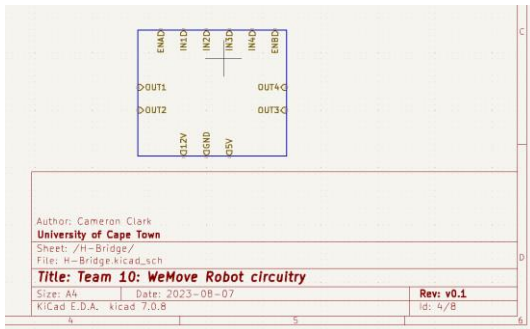


Figure 20: H-Bridge subsystem schematic

## Motors subsystem:

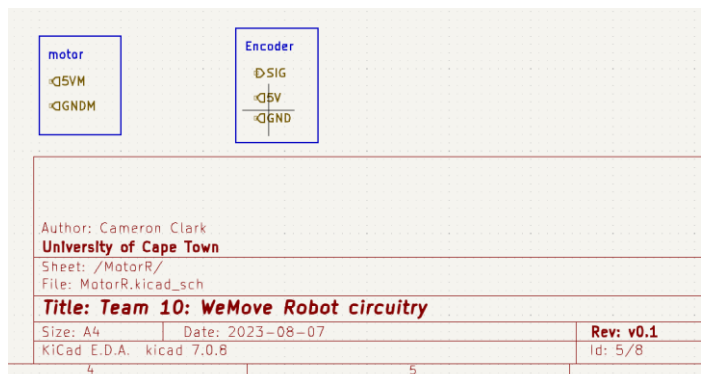


Figure 21: Motor subsystem schematic (1)

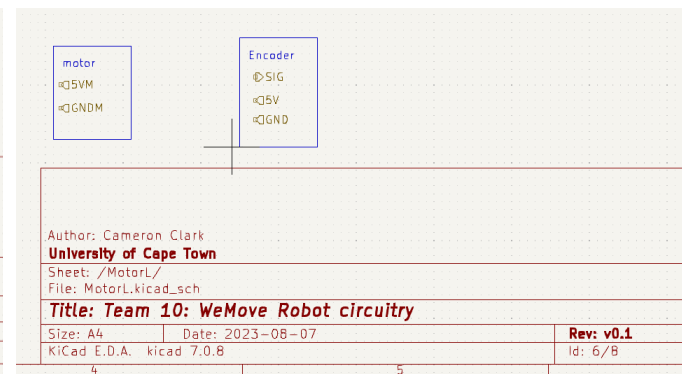


Figure 22: Motor subsystem schematic (2)

## Sensor subsystem:

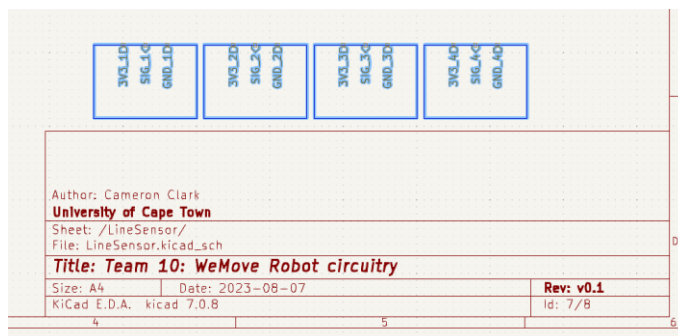


Figure 23: Sensor subsystem schematic (1)

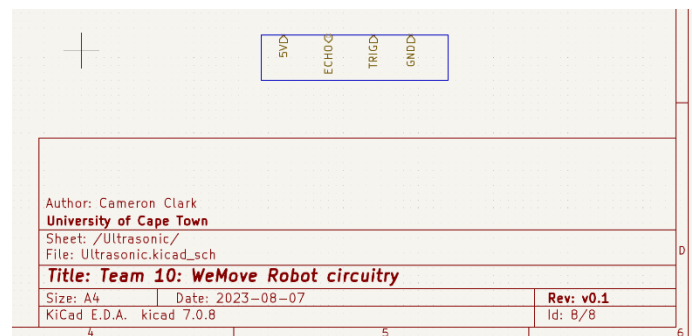


Figure 24: Sensor subsystem schematic (2)



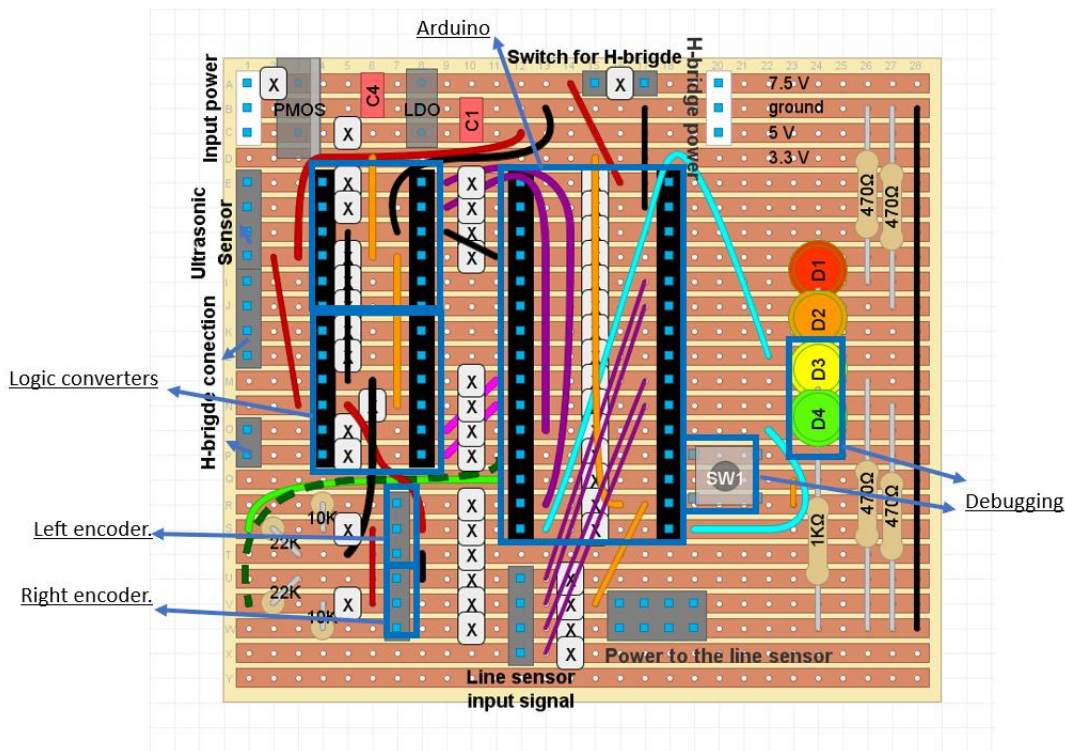


Figure 25: Layout of the PCB

Light blue	Arduio to LED
purple	Arduio to Line sensor and Ultrasonic logic level converters
pink	Arduio to H-Bridge logic level converters
Green (light and dark)	Arduio to Rotary encoders
Black	Ground
Dark red	5V Power
Orange	3V3 Power

Figure 26: Key for figure 17

## 5 Motion control design, implementation, and results Include.

### 5.1 Motion control design to move forward 1m.

#### 5.1.1 Pseudocode functions

Not applicable.

#### 5.1.2 UML or flow diagrams of code or MATLAB Simulink layout

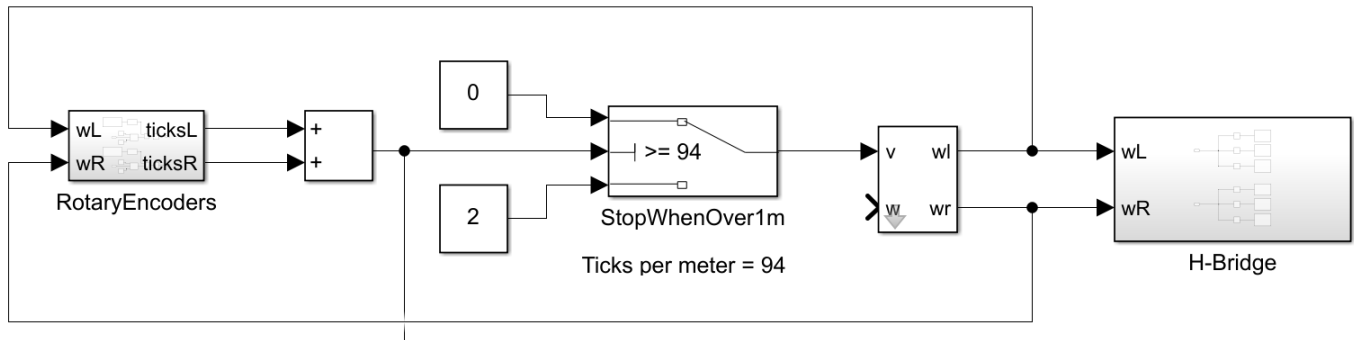


Figure 27: Overview of design used to move forward on meter.

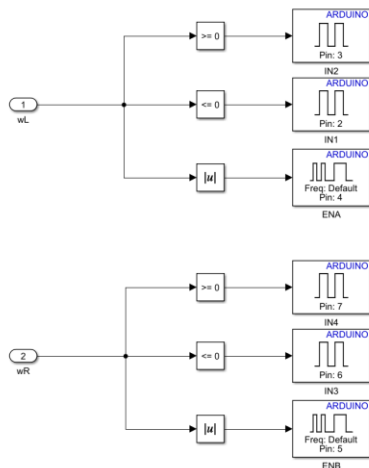


Figure 28: internals of the H-Bridge block

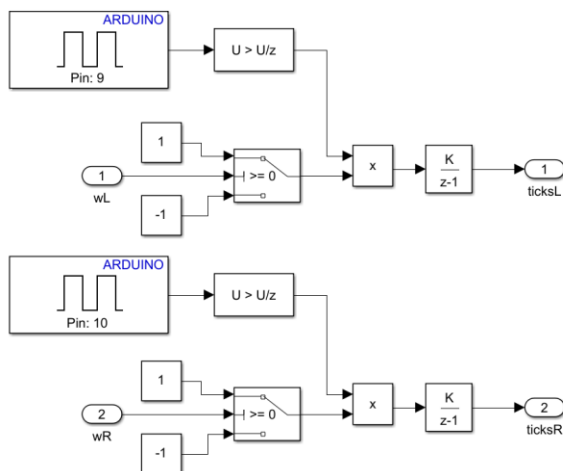


Figure 29: internals of the H-Bridge block

### 5.1.3 Results and justifications

For the rotary encoders, a rising edge was used to detect a tick as opposed to a changing edge because it used less processing power, allowing for a higher tick rate, which allows for a faster robot. A discrete time accumulator then counted the ticks every time a rising edge occurred. The ticks added would be positive or negative depending on the individual wheel velocity.

The H-Bridge required a PWM pin and two logic pins per wheel. The PWM pin takes a value between 0 and 255 which is converted into a duty cycle and sent to the H-Bridge. The two logic pins dictated direction, thus a simple 'compare to zero' block will set one pin high and the other low while the motor is moving, and both high when the motor is stopped which matches the functionality described by the H-Bridge datasheet.

The easiest way to compare measurement was to stay in ticks, thus it was calculated that there are 94 total ticks between the wheels when traveling a metre.

The encoders are set up to have 10 tics per rotation.

The wheel radius (R) is 0.033m.

The following calculation were used to work out how many tics in on meter.

$$\text{Circumference} = 2 \times \pi \times R = 2 \times \pi \times 0.033 = 0.2073m$$

$$\text{Number of rotations in one meter} = \frac{1}{0.2073} = 4.823$$

$$\text{Number of tics required in one meter} = \text{Number of rotations in one meter} \times \text{No. of tics per rev} \times \text{no. of wheels} = 4.823 \times 10 \times 2 = 94 \text{ tics.}$$

*Table 6: Results for the straight-line motion of the robot over 1m.*

Test 1:	1.09 m
Test 2:	0.99 m
Test 3:	1.01 m
Test 4:	0.97 m
Test 5:	0.93 m

$$\text{Result Average} = \frac{1.09+0.99+1.01+0.97+0.93}{5} = 0.998m$$

The for the Average from these results from the test above = 0.998 m. All test where within the 10% error margin. There seem to be a decrease in the amount rotate this is most liked decrease in battery life.

## 5.2 Motion control design for rotation

### 5.2.1 Pseudocode functions

Not applicable.

## 5.2.2 UML or flow diagrams of code or MATLAB Simulink layout

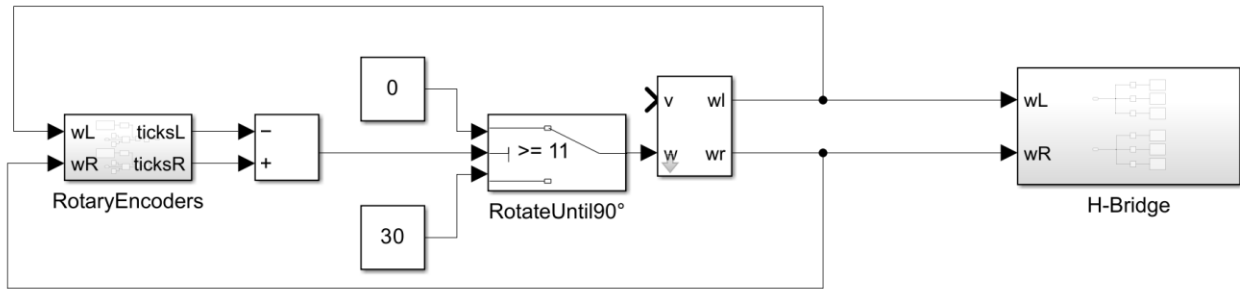


Figure 30: Overview of design used to move forward on meter.

## 5.2.3 Results and justifications

For the rotary encoders, a rising edge was used to detect a tick as opposed to a changing edge because it used less processing power, allowing for a higher tick rate, which allows for a faster robot. A discrete time accumulator then counted the ticks every time a rising edge occurred. The ticks added would be positive or negative depending on the individual wheel velocity.

The H-Bridge required a PWM pin and two logic pins per wheel. The PWM pin takes a value between 0 and 255 which is converted into a duty cycle and sent to the H-Bridge. The two logic pins dictated direction, thus a simple 'compare to zero' block will set one pin high and the other low while the motor is moving, and both high when the motor is stopped which matches the functionality described by the H-Bridge datasheet.

The easiest way to compare measurement was to stay in ticks, thus it was calculated that there is a 42 tick difference between the wheels when turning full 360°

The axel length (AL) is 0.14m

The difference in distance each wheel would have to travel to do a complete circle is  $C = 2\pi \cdot AL = 0.88m$

And each wheel has 47 ticks per metre. Thus, there needs to be a difference of  $0.88 \cdot 47 = 42$  ticks between the wheels to travel the full circumference.

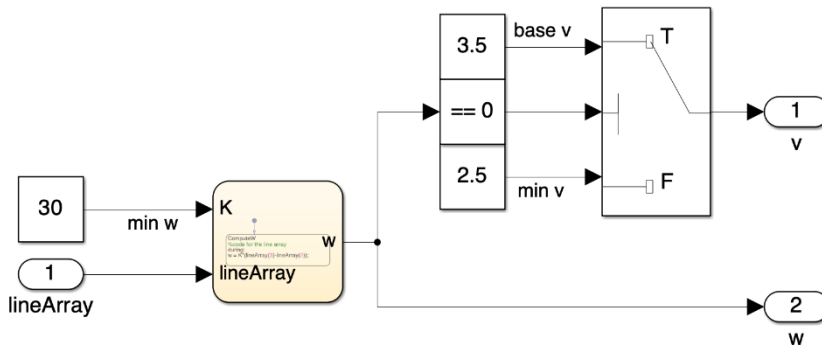
Table 7: Results for the rotation of the robot through 90 degrees.	
Test 1:	98°
Test 2:	86°
Test 3:	95°
Test 4:	91°
Test 5:	92°

$$\text{Result Average} = \frac{98+86+95+91+92}{5} = 92.4^\circ$$

The for the Average from these results from the test above = 92.4°. All the test where within the 10% error margin. There seem to be a decrease in the amount rotate this is most liked decrease in battery life.

## 6 Line sensing design, implementation, and results Include.

### 6.1 UML or flow diagrams of code or MATLAB Simulink layout



### 6.2 Results and justifications

The idea of the line following algorithm is to create smooth, but consistent line following. The method is uses is by comparing the middle two-line sensors. If both sensors are on the line, it does not turn and drives at a higher speed of 3,5. If either line sensor if off the line, it drives forwards at a lower speed with an added angular velocity which causes one wheel to travel significantly faster than the other.

This knowledge can be used to calculate the smallest radius of curvature that the robot can follow. The velocities were obtained using the simulation and the axel length was measured on the robot.

$$Velocity = \frac{d}{dt} Circumference = 2\pi \cdot \frac{d}{dt} Radius$$

$$\therefore \frac{v_{outer}}{v_{inner}} = \frac{R_{outer}}{R_{inner}} = \frac{R + \frac{Axel\ length}{2}}{R - \frac{Axel\ length}{2}}$$

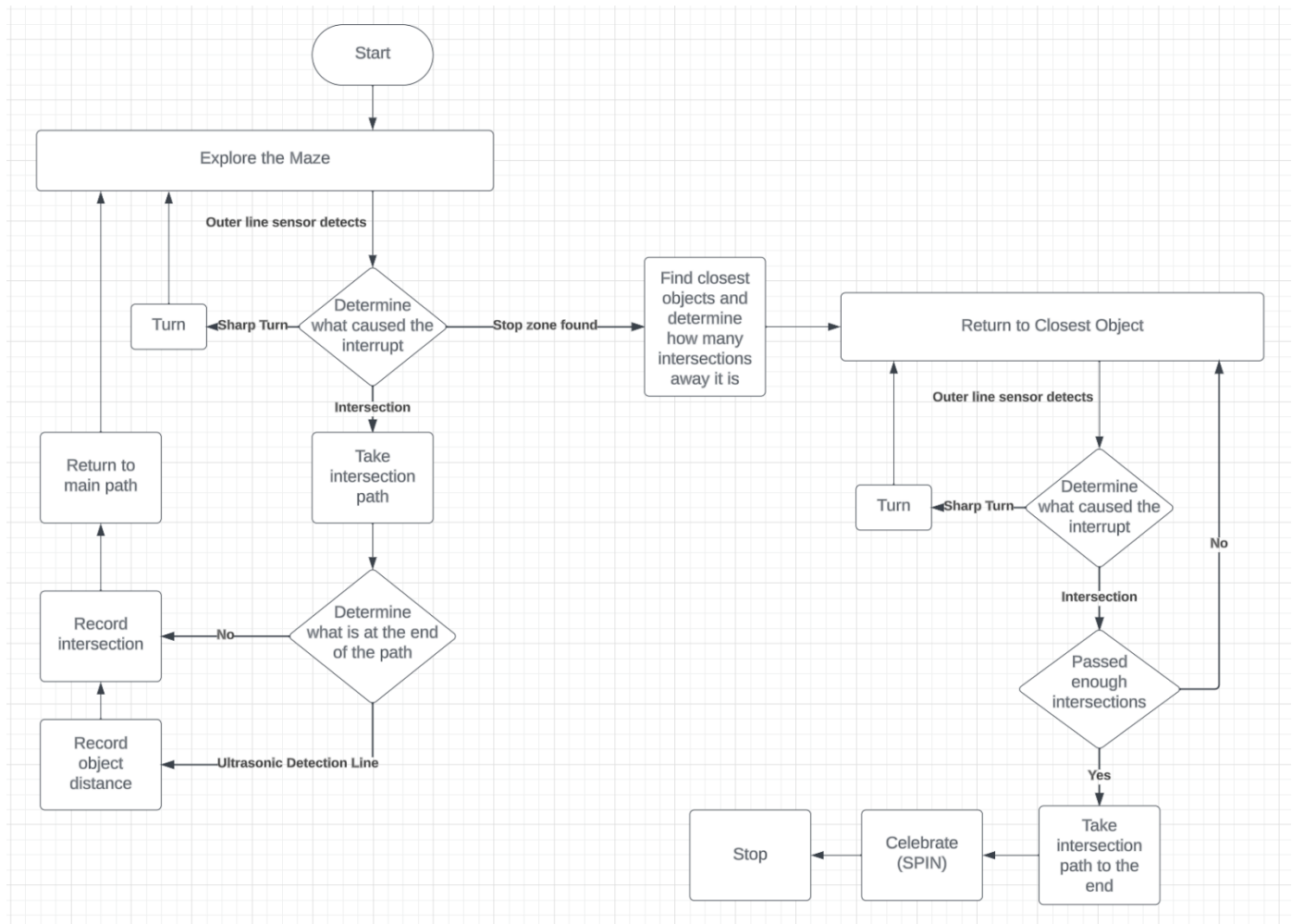
$$\frac{139,4}{12,12} = \frac{R + 0,07}{R - 0,07}$$

solve for R:  $R = 83\ mm$

In the map, the robot completed a 90° turn with a radius of curvature of 77mm, however the calculations show that it would not have been able to make a 360° turn with the same radius of curvature.

## 7 Treasure Maze solving algorithm design, implementation, and results.

### 7.1 UML diagrams (flow diagram of code)



### 7.2 Algorithm explanation

There are two major sections to the code: solving the maze and returning to the closest object. The solving the maze objects was built on three given principles of the maze: there is a single, straight path between the start and stop zone, there are no compound intersections where one intersection leads to another, and a measurement line is denoted by a T junction.

While the robot is solving the maze, it is mostly doing line following using the middle two sensors. When either of the outer two line sensors detect a line, the robot will then move forwards to centre itself on intersection. If the middle two line sensors lose the line, the robot assumes it is at a sharp turn and turns according to which outer line sensor detected the line. Otherwise, the robot turns down the intersection path to discover what is at the end of it, where the turning direction depends on which outer line sensor found a line.

At the end of the intersection there are two possibilities. The one is that the robot find a dead end where all the line sensors lose the line; here the robot will record the intersection as a 2 in the array of intersections, turn around and return to the intersection. The other is that there is an object measurement line where both outer line sensors find a line; here the robot will record the intersection as the distance to the object in metres in the array of intersections, light up one LED for the detected object and another if the object is closer than 15cm, and finally turn around and return to the intersection. When the robot returns to the intersection, it will turn off the LEDs then turn the same direction that it turned to go down that intersection which will return it to the main path.

Finally, if, while on the main path, both outer line sensors find a line, it is assumed that the stop zone has been found. Here, the robot calculates how many intersections away the closest object which is always displayed on the LEDs as a 2-bit binary number. This function is explained in pseudo code below. It then turns around and follows the line back down the main path. Every time either of the outer line sensors finds a line then loses it, the intersection counter decrements, if the counter is zero, the robot takes that path to the end where it pauses, celebrates by spinning fast, and finally stops.

### 7.3 Pseudocode functions

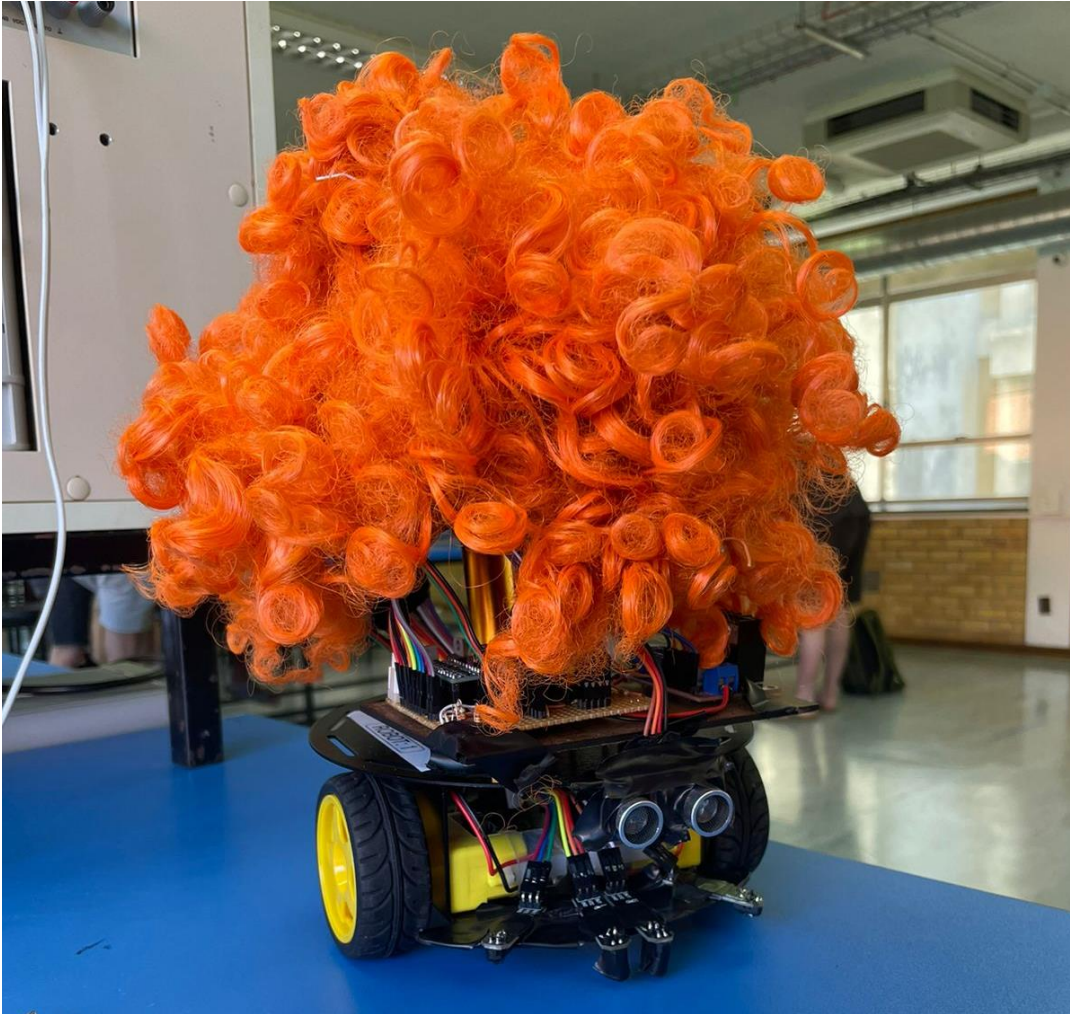
Determining how many intersections away the closest object is

```
#Input: "intersections" which is an array of the distances of the objects at each intersection; dead ends are stored as 2m.
#Output: "target" the number of intersections to pass before arriving at the closest object intersection
INITIALISE closest_object  $\leftarrow$  2
FOR EACH distance IN intersections
    IF distance < closest_object
        closest_object  $\leftarrow$  distance
    END IF
END FOR
#Closest object found, now loop through and find how many intersections from the start it is.
FOR target FROM 1 TO length(intersections)
    IF intersections[target] = closest_object
        BREAK
    END IF
END FOR
# "target" is now how many intersections from the start to the closest object.
target  $\leftarrow$  length(intersections) - target
# "target" is now how many intersections from the end to the closest object.
```



## 8 Conclusion and Recommendations

The robot completed the maze in a time of 2 minutes and 59 seconds, meeting all the criterion outlined in the acceptance test procedure. Below is an image of the robot, Greg, in final form,



*Figure 31: Final image of Greg*

Despite having 2 fried Arduino pins, the robot was able to meet all specifications perfectly. One major concern was slip. This didn't have an impact on the test since the surface provided enough friction to prevent the robot from slipping. The speed at which the robot navigated the maze was also adequate that there was no risk of slipping.

For next time,

- Have a fully working Arduino. All Arduino pins should be inspected prior to code development.
- Use rotary encoders that can count the number of ticks faster.
- Perhaps insulate the chassis of the robot much better. There were instances where the body was shorting the Veroboard and sensors.
- Add additional cooling to the 5V regulator.
- Use a dial with smaller ticks so that measurements are more precise.