# EEE3099S
# DESIGN REPORT


COMPOSED BY:

NATHAN SIEBERHAGEN – SBRNAT004

GATHUKU MATHERI – MTHGAT001

SIYABONGA NHLAPO – NHLSIY008

# Table of Contents

# INTRODUCTION

This report outlines the design process of an Arduino line maze follower which used a robotic system engineered to navigate through mazes. This design comprises of various components and subsystems, each playing a role in its operation. From the sensor subsystem, equipped with line and ultrasonic sensors, to the power management system relying on lithium-ion batteries, every facet has been carefully chosen and integrated. The controller, an Arduino Nano 33 IoT, serves as the control centre, facilitating communication and control across the system.

This report serves as a comprehensive documentation of the design, implementation, and performance evaluation of the line maze follower. It delves into the intricacies of each subsystem, discusses the algorithms governing motion control and line sensing, and details the power calculations needed to sustain operations.

The motion control, line following and treasure maze solving algorithm is discussed to justify and explain its logical reasoning with diagrams of its Simulink operations along with relevant pseudocode. The following sections offer an in-depth examination of each subsystem and algorithm, presenting a holistic view of the robot that navigates a maze and locates objects.

Git Repo: *https://github.com/MrLituation/EEE3099S.git*

# SYSTEM LEVEL DESIGN

The line maze follower comprises of a lot of different components and subsystems.

**Sensor subsystem**

**Line sensor sub-subsystem**

- Power supply: 3.3V
- Detecting range: 1-2cm
- Operating current is greater than 10mA
- Operating temperature range: 0-50$^{o}$C

**Ultrasonic sub-subsystem**

- Power supply: 5V
- Operating current: 15mA
- Ranging Distance: 2cm - 4m

**Motor subsystem**

Dual H – Bridge module

- Voltage level: 5V DC
- Drive voltage: 5-35V DC
- Drive current: 2A

Motor

- Gear Ratio 1:120
- No-load speed (6V): 200RPM
- No-load current (6V): 71mA

Wheel encoders

- Voltage: 5V
- Current < 20Ma

**Power Subsystem**

Batteries

- Output 3.7V each

**Controller subsystem**

Arduino Nano 33 IoT

- Input voltage: 5-21V
- I/O voltage: 3.3V
- Current per I/O pin: 7mA

Logic level converters

- Steps up 3.3V to 5V
- Steps down 5V to 3.3V

**Indicator subsystem**

- Red LED

**Layout subsystem**

- Veroboard max size: 10cm x 10cm
- Axle length: 13.6cm
- Wheel Diameter: 6.2cm
- Turtle dimensions: 170mm diameter base

The interactions of these subsystems can clearly be seen in the system block diagram below:

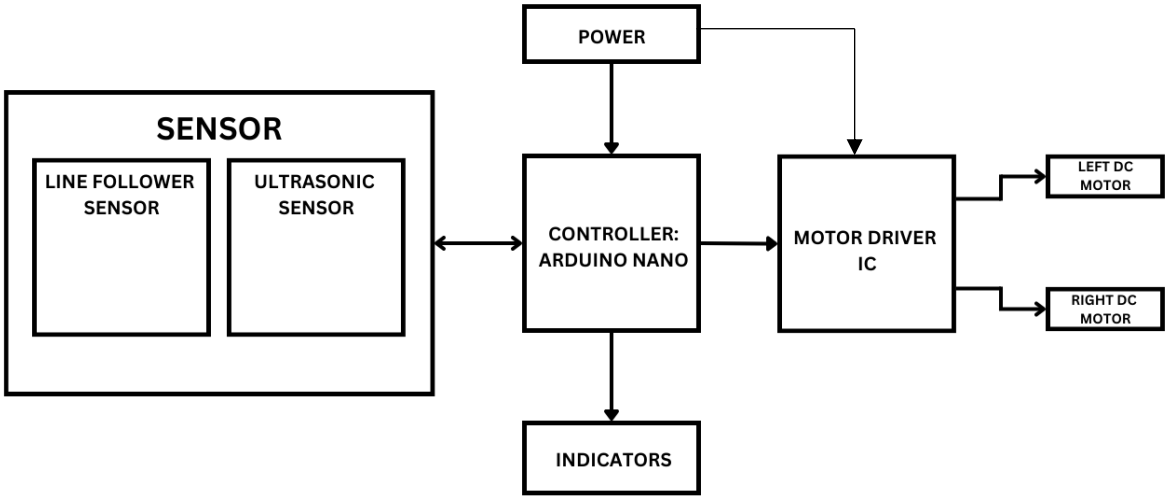


Figure 1: System block diagram

## System block diagram discussion

The system's whole circuitry is under the control of the controller (Arduino nano).

To enable the controller to deliver information to the motor subsystem, which will rotate the wheels to make the necessary modifications to follow the line, the line sensor will send data to the controller indicating the location of the line.

The controller and the motor driver are powered by the Power subsystem. The controller then has a 3.3V output which powers the sensor subsystem. The line sensor will communicate with the controller after it has located the "detect line" and the controller will then turn on the indicator (LED). The controller will then instruct the subsystem to begin detecting while also communicating with the ultrasonic sensor.

The following table lists the functions and justifications of all of the components that were used in the construction of the line maze follower:

| Component | Description / Function | Justification |
|---|---|---|
| **CONTROLLER SUBSYSTEM** | | |
| Arduino nano 33 IoT | Arduino's smallest board when relating to IoT functions. | The ultrasonic sensor and the line sensors used are both entry-level peripherals to the IoT world so this MCU is more than adequate for this application. |
| BS183 Logic level shifter | The level shifters are responsible for interfacing the communication of components that operate at different voltage levels. | In this case it is the microcontroller which requires 3.3V on its digital pins when the provided voltage is 5V. |
| **SENSOR SUBSYSTEM** | | |
| HC-SR04 Ultrasonic sensor | This sensor is responsible for detecting the distance of an arbitrary object from a detect line. | Low-cost, moderate accuracy distance detection sensor. The two communication pin configuration also makes it very easy to use for the underlying application. |
| ITR20001/T Line sensor for Arduino | Infrared line sensor that goes LOW for a black line being detected. | Low-cost, high sensitivity with a high response time. The mounting holes of the sensors also make it work well with the Arduino turtle platform. |
| **POWER SUBSYSTEM** | | |
| 2 x 18650 3.7V lithium-Ion batteries | 7.4V voltage system voltage supply | Lithium-ion batteries tend to have a higher power density for more battery life per unit size. |
| **MOTOR SUBSYSTEM** | | |
| L298N H-Bridge | Low-cost, small mobile two wheel drive platform that is used | The components and logic levels must have a maximum operation voltage of 5V |
| DC Motors | Take in required voltage provided by H-Bridge. The magnitude of | DC motors needed for physical motion of robot |

| | | |
|---|---|---|
| | the voltage provided is directly proportional to the motor's rotational speed. | |
| Wheel Encoders | Used to measure the rotational speed of a motor. | They capture the travelled distance which can be used in algorithms for certain applications. |
| **INDICATOR SUBSYSTEM** | | |
| LED (RED) | Responsible for turning on when an object has been detected by the ultrasonic sensor. | RED LEDs tend to have a lower volage drop than most LEDs such as blue or green LEDs. This is a low voltage application, hence, red was used. |

# ELECTRICAL DESIGN

## Power calculations

It is worth noting that most of the values in some of the power budgets could not be determined simply because of the underlying datasheets of the components in question were not comprehensive enough. Also, most of the maximum values that were not in the datasheets could not be physically determined as stress tests were not done on the components as it was a necessity to ensure that the components do not break down.

**Table tabulating the power submodule power budget:**

| Component | Supply Voltage (nominal V) | Max Voltage (V) | Typical Current (mA) | Maximum Current (mA) | Typical Power (mW) | Maximum Power (mW) |
|---|---|---|---|---|---|---|
| Battery | 7.37 | 7.4 | 330 | - | 2432.1 | - |
| **Totals** | 7.37 | 7.4 | 330 | - | 2432.1 | - |

The power figure that matters the most is the one above. These values were attained by simply measuring the voltage across the battery and then also getting the current flowing through the battery. Therefore, the 2432.1mW (2.4W) represents the power that is consumed by the whole system during operation. Therefore, since the capacity of the batteries are 2200 mAh, so on a full charge the batteries provide approximately 6.67 hours of continuous operation before the batteries die. ($\frac{2200mAh}{330mA} = 6.67\ Hours$)

**Table tabulating the controller submodule power budget:**

| Component | Supply Voltage (nominal V) | Max Voltage (V) | Typical Current (mA) | Maximum Current (mA) | Typical Power (mW) | Maximum Power (mW) |
|---|---|---|---|---|---|---|
| Arduino Nano 33 IoT (One I/O pin) | 3.3V | - | 7 | - | 23.1 | - |
| Logic Level Converters | 5 | - | 1 | - | 5 | 360 |
| **Totals** | 8.3 | - | 7 | - | 28.1 | 360 |

According to the Arduino documentation, each I/O pin consumes 7mA of current when in operation. Seven logic shifter lines were used, therefore, 35mW of total power was consumed through both of the two logic shifters. Furthermore, there are 14 I/0 pins on the microcontroller that are being used. Each of them have a typical power rating of 23.1mW, therefore, 323.4mW ($23.1 \times 14$) of power is being consumed by the microcontroller during operation. Therefore, this controller submodule accounts for $\frac{35+323.4}{2432.1} = 0.147 = 14.7\%$ of the total power consumption.

**Table tabulating the sensor submodule power budget:**

| Component | Supply Voltage (nominal V) | Max Voltage (V) | Typical Current (mA) | Maximum Current (mA) | Typical Power (mW) | Maximum Power (mW) |
|---|---|---|---|---|---|---|
| Ultrasonic Sensor [HC-SR04] | 5 | - | 15 | - | 75 | - |
| Line Sensor [ITR20001/T] | 3.3 | 5 | 18 | 20 | 59.4 | 100 |
| **Totals** | 8.3 | 20V | 33 | 20 | 134.4 | 100 |

The total system power dissipation is calculated in the power submodule power budget as 2432.1mW. Four line sensors were used, therefore, $59.4 \times 4 = 237.6mW$. Therefore, whilst in operation the sensors accounts for $\frac{75+237.6}{2432.1} = 0.1285 = 12.85\%$ of the total power consumption.

**Table tabulating the indicator submodule power budget:**

| Component | Supply Voltage (nominal V) | Typical Current (mA) | Typical Power (mW) |
|---|---|---|---|
| RED LED | 2.1 | 20 | 42 |
| **Totals** | 2.1 | 20 | 42 |

The total system power dissipation is calculated in the power submodule power budget as 2432.1mW. Therefore, whilst in operation the LED indicator accounts for $\frac{42}{2432.1} = 0.017 = 1.7\%$ of the total power consumption.

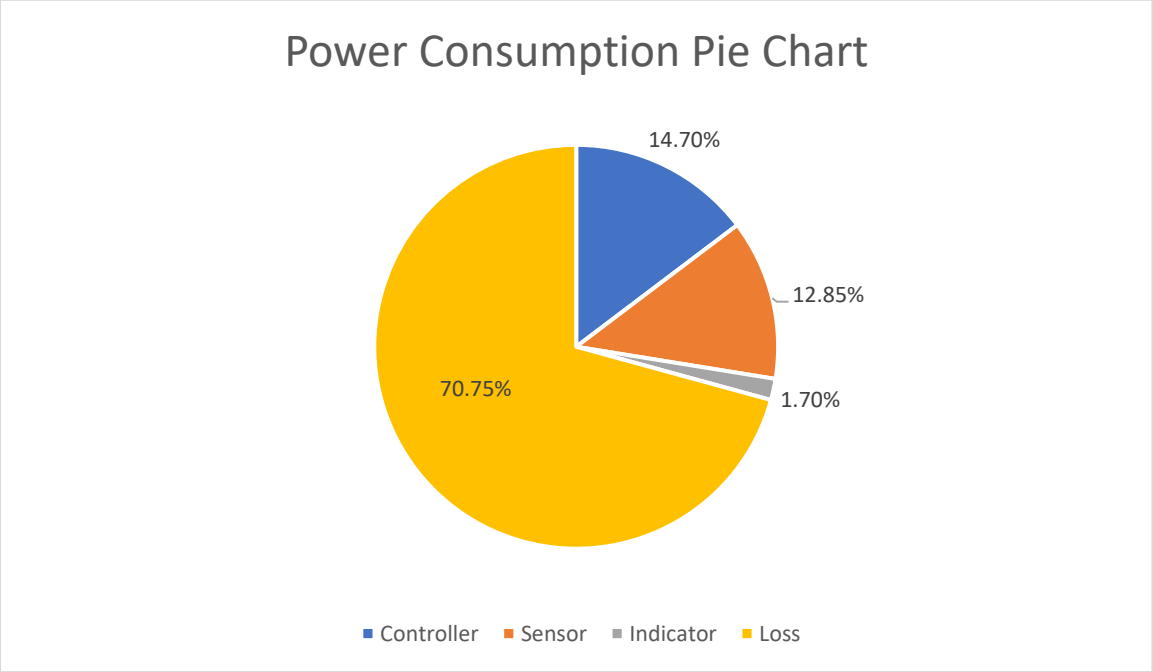


Figure 2: Pie chart illustrating system power consumption

The rough pie chart above showcases a very known phenomenon. This is that nearly 70% - 80% of the total power in a mechanical system is lost. The underlying system is indeed a mechanical system as we are dealing with motors. This loss, thus, comes in many forms particularly friction and heat.

## Power supply design and safety

Two simple 3.7 2200mAh lithium-ion batteries. The 2200mAh simply implies that the battery can supply 2.2A of continuous current for one hour. The 7.4V was fed directly into the Arduino Nano IoT as the microcontroller has an internal voltage regulator and can take voltages up to a maximum of 21V. Furthermore, the H-Bridge that was used has an internal voltage regulator as well as protection diodes. This H-Bridge can take in a maximum of 35V and it ideally has a port where a regulated 5V line can be taken. This regulated 5V line was the 5V line that was used in our circuit to power any component that needed 5V such as the motors. Similarly, the microcontroller has a regulated 3.3V output voltage which was used to power all 3.3 components such as the sensors used.

To add a redundant power safety aspect we routed our circuit in such a way that a 5V to 3.3V 3-pin volage regulator can be used in a case where the user would not want to use the 3.3V line from the microcontroller.

Note that the use of the logic-level shifters are crucial. This is because the Arduino nano 33 IoT cannot take in voltages greater than 3.3V on its digital pins. If this is the case the microcontroller stands a good chance of being permanently damaged. Therefore, the voltages coming from the H-Bridge are 5V lines so the level shifters are responsible for shifting these lines down to 3.3V before they reach the terminals of the microcontroller.

## Microcontroller resource allocation

| PIN | DESCRIPTION | TYPE | CIRCUIT FUNCTION |
|---|---|---|---|
| 1 | D13 | Digital | Connected to Input 4 on the H-bridge which is one of the two pins responsible for the rotational direction of motor 2 (right motor) |
| 2 | +3V3 | Pwr Out | 3.3V output line of MCU that was uses as the circuit 3.3V line. |
| 3 | AREF | Analog | UNCONNECTED |
| 4 | A0/DAC0 | Analog | LED connected to this pin. This pin is then converted to a digital pin and then set high whenever an object is detected by the ultrasonic sensor. |
| 5 | A1 | Analog | Analog line converted to digital and used as 3.3V digital data line of the furthest left line sensor. |
| 6 | A2 | Analog | UNCONNECTED |
| 7 | A3 | Analog | UNCONNECTED |
| 8 | A4/SDA | Analog | Analog line converted to digital and used as 3.3V digital data line of the nearest left line sensor. |
| 9 | A5/SCL | Analog | UNCONNECTED |
| 10 | A6 | Analog | Analog line converted to digital and used as 3.3V digital data line of the nearest right sensor. |
| 11 | A7 | Analog | Analog line converted to digital and used as 3.3V digital data line of the furthest right sensor. |
| 12 | VUSB | Pwr In/Out | UNCONNECTED |
| 13 | RST | Digital | UNCONNECTED (pin 18 duplicate) |
| 14 | GND | Power | Power ground |
| 15 | VIN | Power | +7.4V input from the lithium-ion batteries |
| 16 | TX | Digital | UNCONNECTED |
| 17 | RX | Digital | UNCONNECTED |
| 18 | RST | Digital | Connected to a switch which simply resets the microcontroller to its initial/default state. |
| 19 | GND | Power | Power Ground |

| | | | |
|---|---|---|---|
| 20 | D2 | Digital | UNCONNECTED |
| 21 | D3/PWM | Digital | UNCONNECTED |
| 22 | D4 | Digital | UNCONNECTED |
| 23 | D5/PWM | Digital | UNCONNECTED |
| 24 | D6/PWM | Digital | Connected to trigger pin of the ultrasonic sensor. This line is also a 5V line so it passes through the level shifter as well before getting to this Arduino pin. |
| 25 | D7 | Digital | Connected to echo pin of the ultrasonic sensor. This line is also a 5V line so it passes through the level shifter as well before getting to this Arduino pin. Alternatively this pin can also be passed through a 2/3 voltage divider to take the voltage to 3.3V from 5 as seen in the circuit schematic. |
| 26 | D8 | Digital | Connected to Input 2 on the H-bridge which is one of the two pins responsible for the rotational direction of motor 1 (left motor) |
| 27 | D9/PWM | Digital | Connected to the enable pin of motor 2 (right motor). This pin has to be a PWM enable pin as it is responsible for the speed of motor 2. |
| 28 | D10/PWM | Digital | Connected to the enable pin of motor 1 (left motor). This pin has to be a PWM enable pin as it is responsible for the speed of motor 1. |
| 29 | D11 | Digital | Connected to Input 1 on the H-bridge which is one of the two pins responsible for the rotational direction of motor 1 (left motor) |
| 30 | D12 | Digital | Connected to Input 3 on the H-bridge which is one of the two pins responsible for the rotational direction of motor 2 (right motor) |

## System Schematics

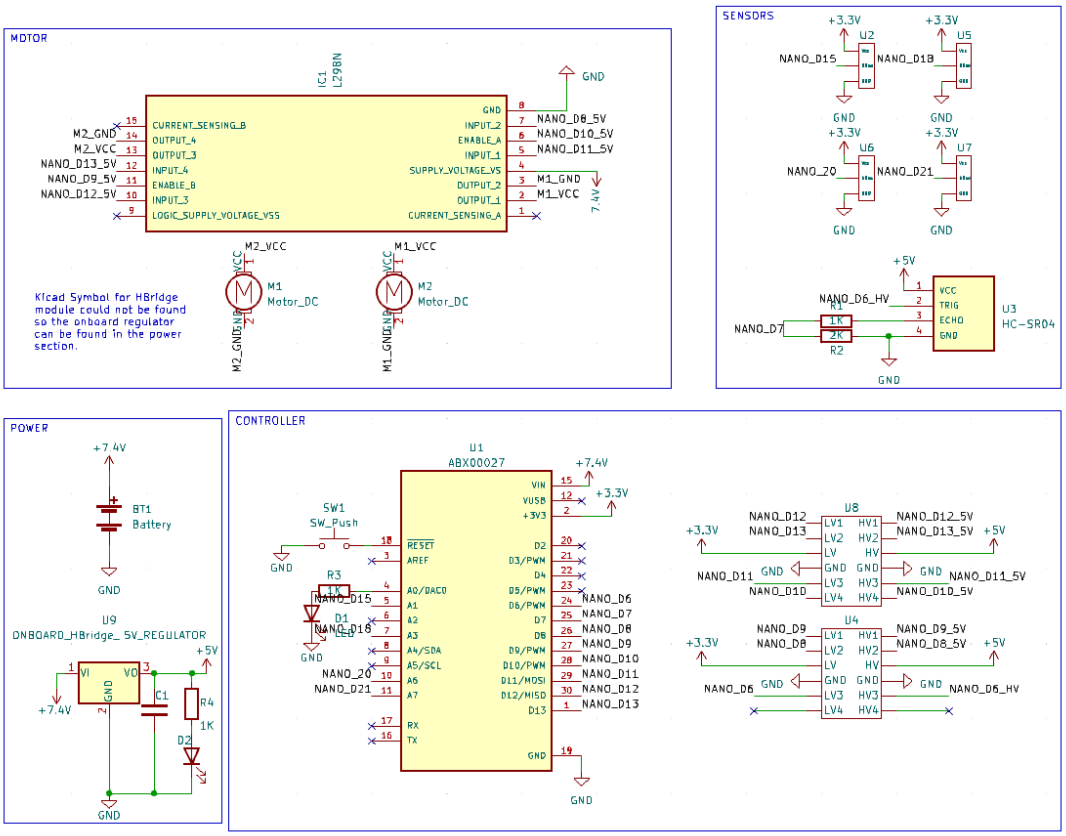


Figure 3: Circuit design of the various system submodules

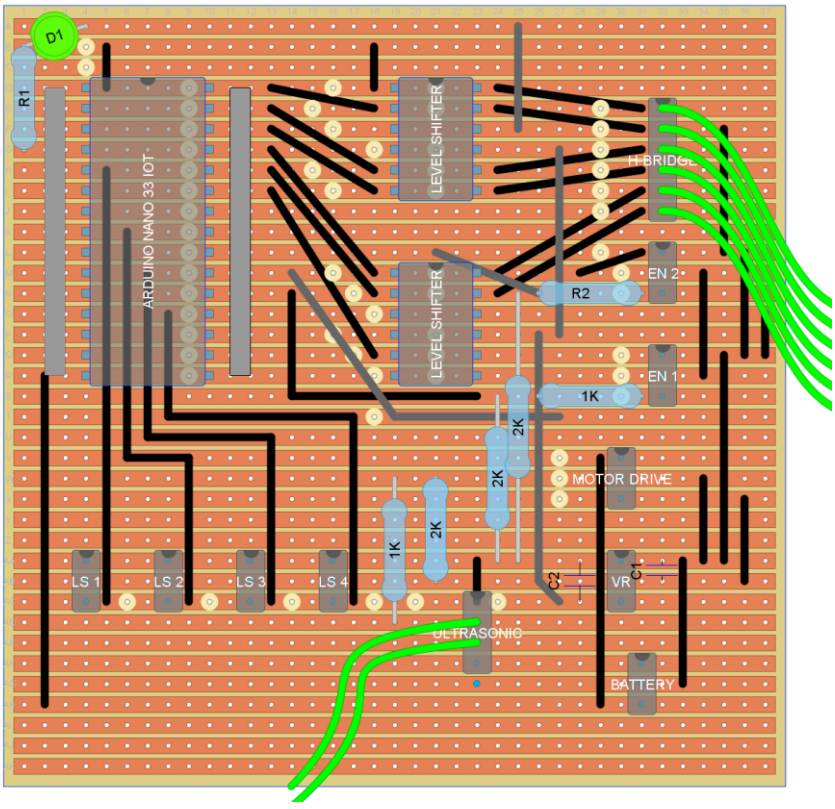


Figure 4: Veroboard system design and layout

# MOTION CONTROL DESIGN, IMPLEMENTATION AND RESULTS

| PSEUDOCODE: Motion Control | |
|---|---|
| | |
| 1 | There are two variables that we use to control motion of the robot: velocity (v) and angular velocity (w). |
| 2 | Velocity is the linear forward motion of the robot whereas angular velocity is the rotational movement of the robot. |
| 3 | A velocity value of 0.1 was used as input to our robot for all standard linear motion. This value was specified as variable setV. |
| 4 | An angular velocity of 1.2 was used as our standard angular velocity and specified as a variable. |
| 5 | During line following, a combination of linear and angular velocity is used to keep the robots centre on the line. |
| 6 | During branch turns, a sharp turn correction, and 180$^{o}$ turns, only angular velocity is used with linear velocity set to 0. The angular velocity is increased at these points to increase the speed of the turn. |
| 7 | Both angular and linear velocity are set to zero when the object reaches a detect line, completes a pure rotation task (as specified in step 6) and when it reaches the end of the maze. |
| 8 | After stopping at all the points mentioned in step 7 the robot performs a 180$^{o}$ turn and then returns to line following. |

## MATLAB Simulink layout:

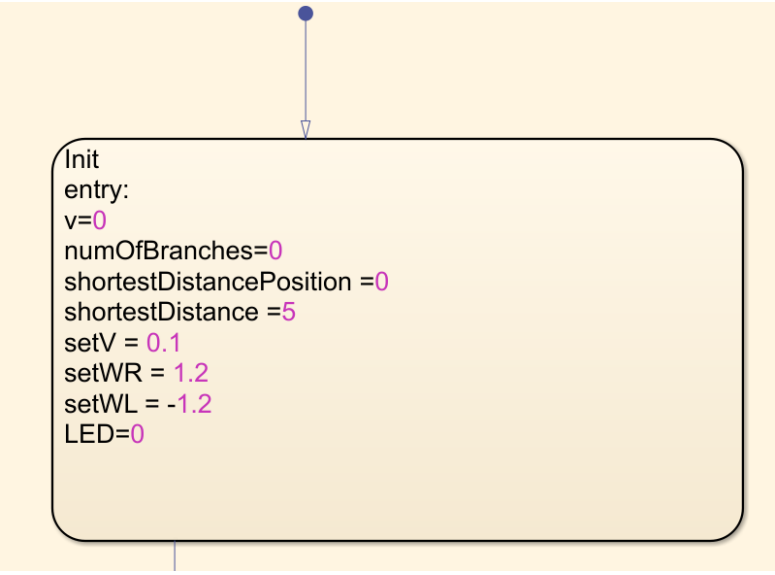


*Figure 5: All the initialisations that occur at the beginning of the code.*

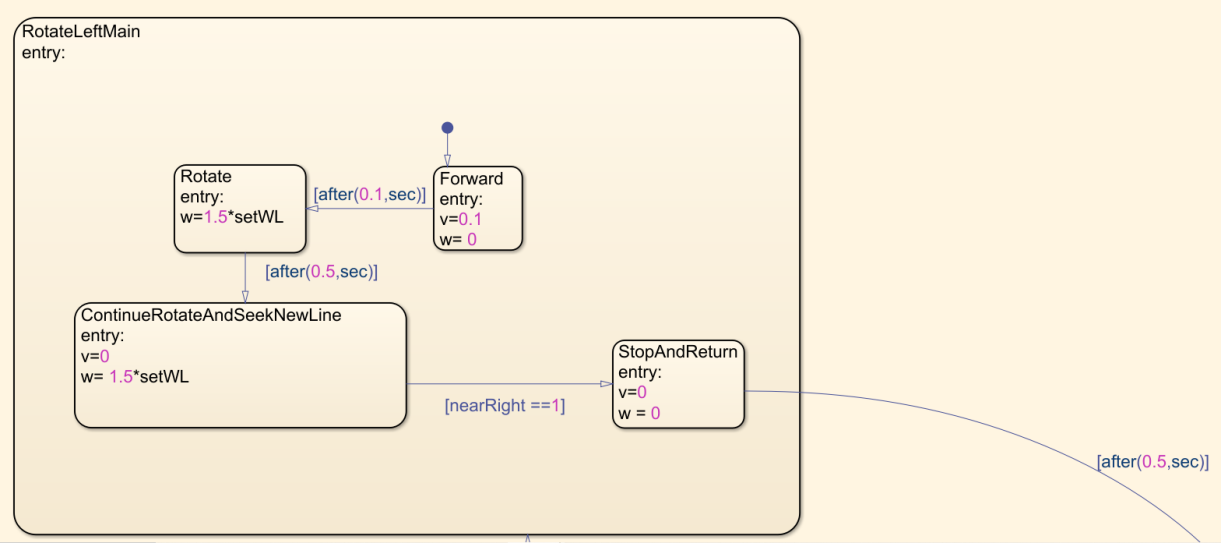


*Figure 6: Simulink code showing an increased angular velocity when making significant rotation.*

## Results and justification:

A velocity of 0.1 was chosen to be the most ideal velocity as this was found to move sufficiently fast enough to complete the maze in a reasonable time while also not moving so fast that the quality of line and branch detection degrades as there is insufficient time to process the relevant detection algorithms.

An angular velocity of 1.2 for left rotation and -1.2 for right rotation was used as this provided the sufficient amount of angular correction for line following when navigating turns.

An increased angular velocity was found to be needed when significant rotation needed to occur such as at branches, sharp corners, or performing a full 180$^{o}$ rotation. Thus, the angular velocity was 50% greater than the set angular velocity as seen in Figure 2.

The robot comes to a complete standstill for a brief moment when completing a pure rotation task as the subsequent linear velocity direction should not be impacted by the remaining angular velocity during its deceleration.

The motion of the robot was kept simple and repetitive to remove room for error and make it easier to troubleshoot and adjust if needed.

# LINE SENSING DESIGN, IMPLEMENTATION AND RESULTS

| PSEUDOCODE: Line Sensing | |
|---|---|
| 1 | Four line sensors are used. Two are placed in the front middle and two are placed on the front left and right side of the robot. |
| 2 | All line sensors are connected to analogue pins of the Arduino Nano. |
| 3 | When the line sensors detect white, it will give an analogue value of 0 but there is noise as well so it will not be perfectly zero. When they sense black, it will give an analogue value around 4000 again accounting for noise. |
| 4 | A switch block is then used so that when the analogue value is above 2000 (sensing black) a value of 1 is output. This was to make the implementation and Simulink logic easier to work with as then we only deal with 1s (sense black) and 0s (sense white). This step can be seen in Figure 7. |
| 5 | The middle two sensors (labelled nearLeft and nearRight in Simulink) are used for line following. When they sense white, angular velocity is induced in the motors causing the robot to rotate slightly so that the sensors go back to sensing the black line. So, the robot's centre is constantly on the line therefore following the line when moving. This can be seen in Figure 8. |
| 6 | One purpose of the outer two sensors (labelled farLeft and farRight in Simulink) are to sense turns. When the outer left senses black and the outer right senses white it means there is a left turn. In the same way if the outer right senses black and outer left senses white it means there is a right turn. |
| 7 | There are two possible turns that the robot can take. A turn that is on the main branch and a turn that will lead to side branches. To distinguish we let the robot check what is after the turn. If the middle two sensors sense white it means that it is a turn on the main branch as there is no line continued after it so it cannot be a side branch. Whereas if they sense black it means there is a line past the turn so the turn is a side branch. This can be seen in Figure 9. |
| 8 | On the side branches there is either a dead end or a detect line. |
| 9 | A dead end can be determined when all four sensors go white. This can be seen in Figure 10. |
| 10 | A detect line can be determined when the outer two sensors sense black. This can be seen in Figure 11. |
| 11 | After all the side branches are taken and the robot reaches the end of the main branch there is a black box. So, for the robot to sense the end all line sensors go black. This can be seen in Figure 12. |

## MATLAB Simulink layout:

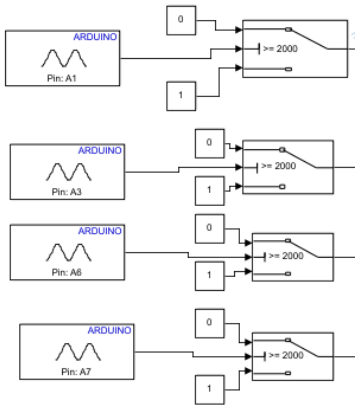


*Figure 7: Line sensor analogue pin outputs*

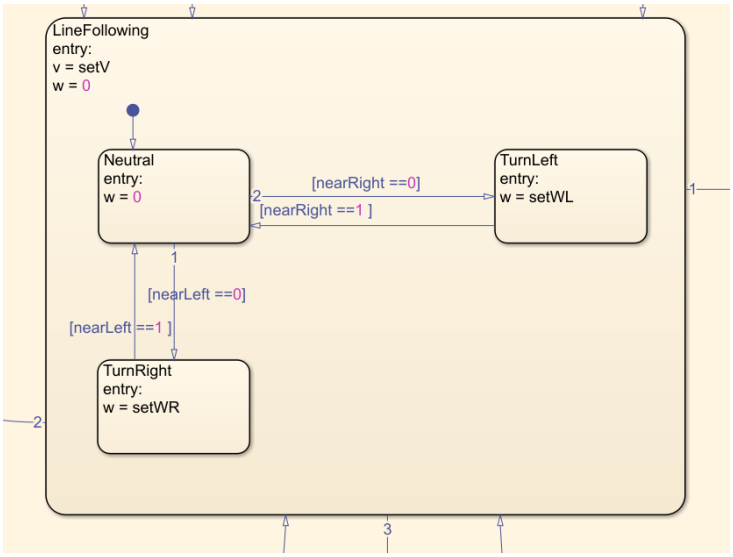


*Figure 8: Middle line sensors being used in line following algorithm.*

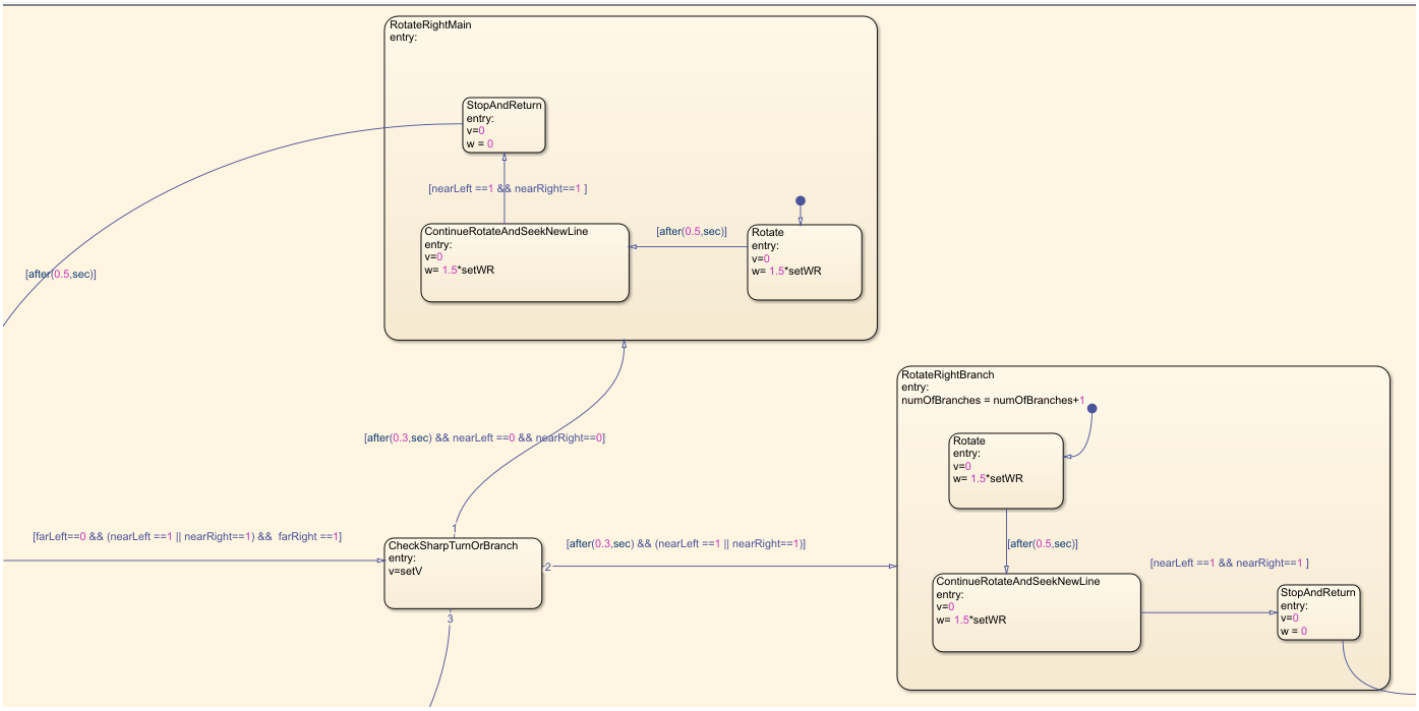


*Figure 9: Detecting a turn and determining what kind of turn it is.*

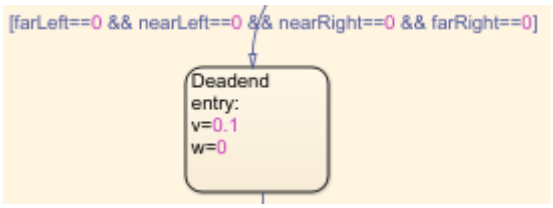


*Figure 10: Sensors determining a dead end.*

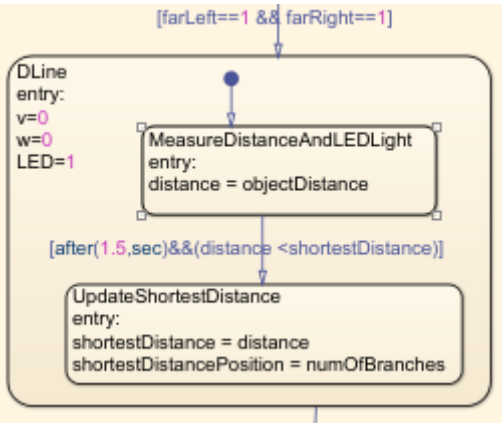


*Figure 11: The outer sensors detect a detect line.*

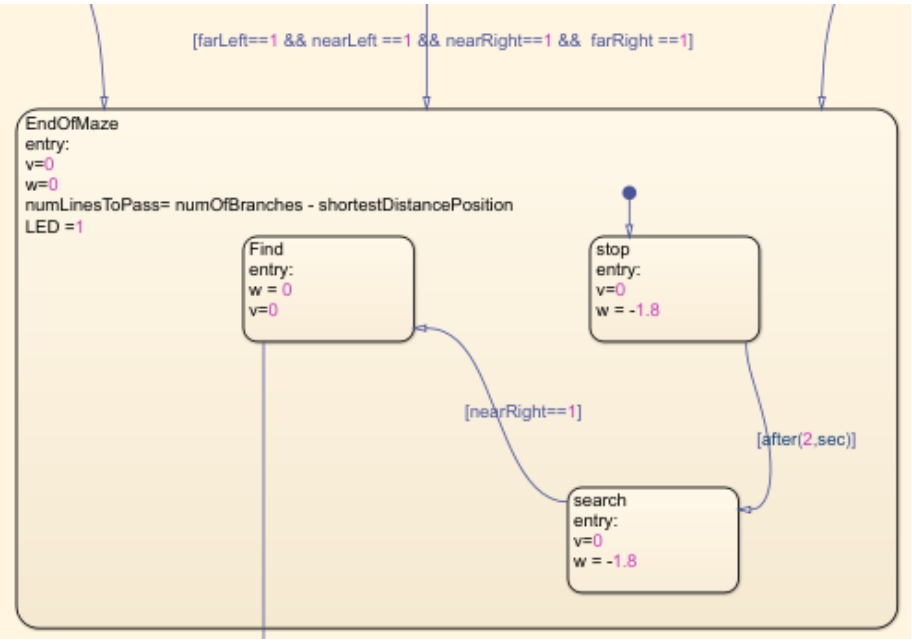


*Figure 12: All sensors detect the end of the maze.*

## Results and justification:

Four line sensors were chosen as that is the number necessary to satisfy all line sensing requirements. Three would not have been enough to satisfy all requirements. Five could have been used however that would be unnecessary and add additional risk of hardware problems and unneeded complexity.

The line sensors were very effective in carrying out their jobs. The reaction time of the sensors was instantaneous as expected. The accuracy and precision of the sensors caused us to utilize delays and more specific instructions as to not trigger an event when unwanted. The distinguishing between a main line turns and a branch turns worked as expected and the detection of dead ends, detect lines and the end of the maze was very smooth.

It was also determined that the line sensors preferred complete white and complete black surfaces and not ground that does not have uniform colouring (such as the labs floor which had chunks of lighter and darker bits). It must be said that sensors still functioned as wanted on nonperfect surfaces just not as effectively due to their sensitivity.

# TREASURE MAZE SOLVING ALGORITHM DESIGN, IMPLEMENTATION AND RESULTS

| | |
|---|---|
| **PSEUDOCODE: Treasure Maze Solving Algorithm** | |
| | **Section 1: Branch Detection and Measuring Algorithm** |
| | **Initialisations** |
| 1 | At the beginning of the code three variables are initialised:<br>- numberOfBranches: This is set to 0 and increases for each of the navigated branches<br>- shortestDistancePosition: This is set to 0 and keeps the position of the nearest object amongst the navigated branches<br>- shortestDistance: This is set to 10m (greater than any of the potential object distances). This maintains the shortest distance detected by the robot |
| | **Branch Detection and Line Following** |
| 2. | When a branch (and not a sharp corner) is detected using the line sensing algorithm, the object stops its linear velocity and rotates in the direction of the triggered outer sensor (i.e. Turns lefts if outer left sensor in triggered and right for outer right sensor). The numberOfBranches variable is incremented by 1. |
| 3. | The robot rotates for 0.5 seconds in the specified direction, after which it will continue rotating while seeking the new branch line as the inner sensor seeks a black line. |
| 4. | When a black line is detected, the object stops rotating and waits 1 second until it begins moving again. |
| 5. | The line following algorithm is triggered once again until either all the line sensors detect white (indicating a dead end) or until both outer sensors detect black (indicating a detect line) |
| | **Dead End Detection** |
| 6. 1 | If a dead end is detected, the robot continues moving forward for 0.5 seconds, until it starts rotating back around. The reason it continues moving forward on a dead end is to allow enough space for the robot to rotate and sense the branch line again when it rotates |
| | **Detect Line Detection and Object Distance Measurement** |
| 6.2.1 | If a detect line is detected, the robot stops moving and measures the object distance and assigns it to a variable. The LED light is also triggered to go on. |
| 6.2.2 | The distance measured is compared to shortestDistance variable and **if** the distance is shorter, the shortestDistance variable is updated to the current distance. The shortestDistancePosition is also updated to the current value of numberOfBranches to track which branch has the current shortest object. The object then waits 3 seconds. |
| | **Return to Main Line** |
| 7 | The object rotates for 1.8 seconds until it begins seeking the branch line again. When the inner black sensor detects black the object stops rotating and begins using the line following algorithm again |
| 8. | When the outer sensors detect black again, the robot continues to move forward for 0.3 seconds so that it aligns with the main line branch. |
| 9. | The robot begins rotating again in the direction of the main line until the inner sensors detect black again. The robot subsequently stops and begins using the line following algorithm again. |
| | **Section 2: End Of Maze Navigation To Shortest Object** |
| 10. | The process in Step 1-9 is repeated until the robot reaches the end of the maze. |
| 11. | When on the main branch, and all four sensors detect black. The robot detects it has reached the end of the maze and the robot stops moving. |
| | **Find Shortest Object and Return To Main Line** |
| 12. | The variable numberOfBranchesToPass is calculated by taking the total number of branches passed and subtracting where the shortest branch position is located. |

| | |
|---|---|
| 13. | The robot begins rotating for 1.8 seconds until it starts searching for the main navigation line again. The robot stops rotating once the inner sensors detect the main black line again. The robot then stops rotating |
| 14. | The robot begins following the line again using the line following algorithm |
| | **Determine if shortest object branch** |
| 15. | If the robot detects a would be branch using the line following algorithm, the following is checked:<br>-If numberOfLinesToPass is greater than 0, the robot carries on moving forward with the same line following algorithm. The numberOfLinesToPass is decremented by 1.<br>-If numberOfLinesToPass is equal to 0, the robot triggers its branch rotation algorithm. See Steps 4-6. |
| 16. | When the object reaches the detect line, the robot comes to a stop and halts its operation. |

## MATLAB Simulink layout:

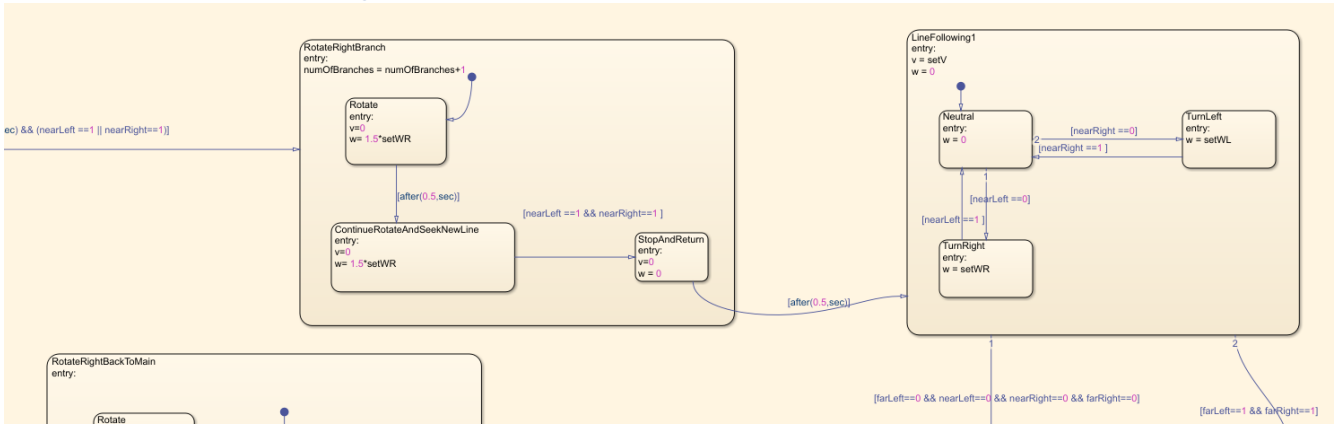


*Figure 13: Diagram showing how the robot rotates from the main line to the side branch along with subsequent line following*

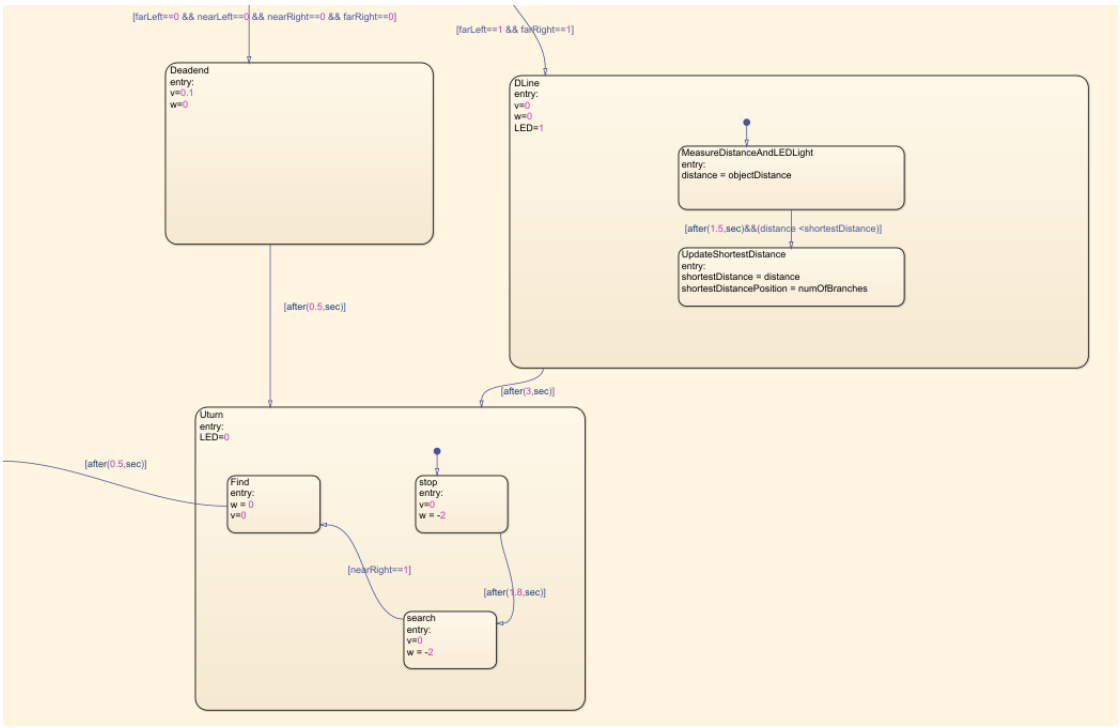


*Figure 14: Diagram showing how to handle detect lines and dead ends along with navigating back to the main line*

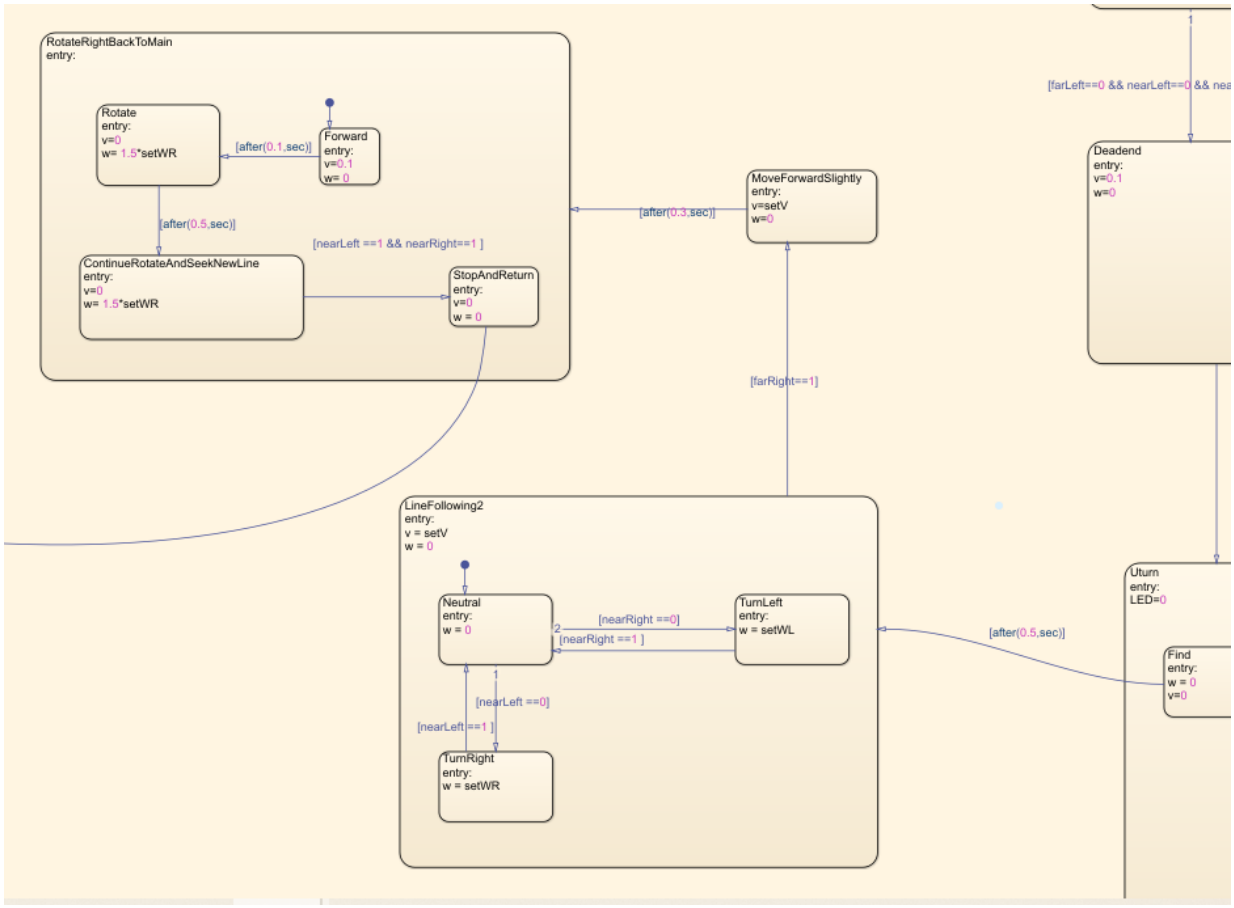


*Figure 15: Diagram showing how the robot navigates back to the main line using line following and rotation*

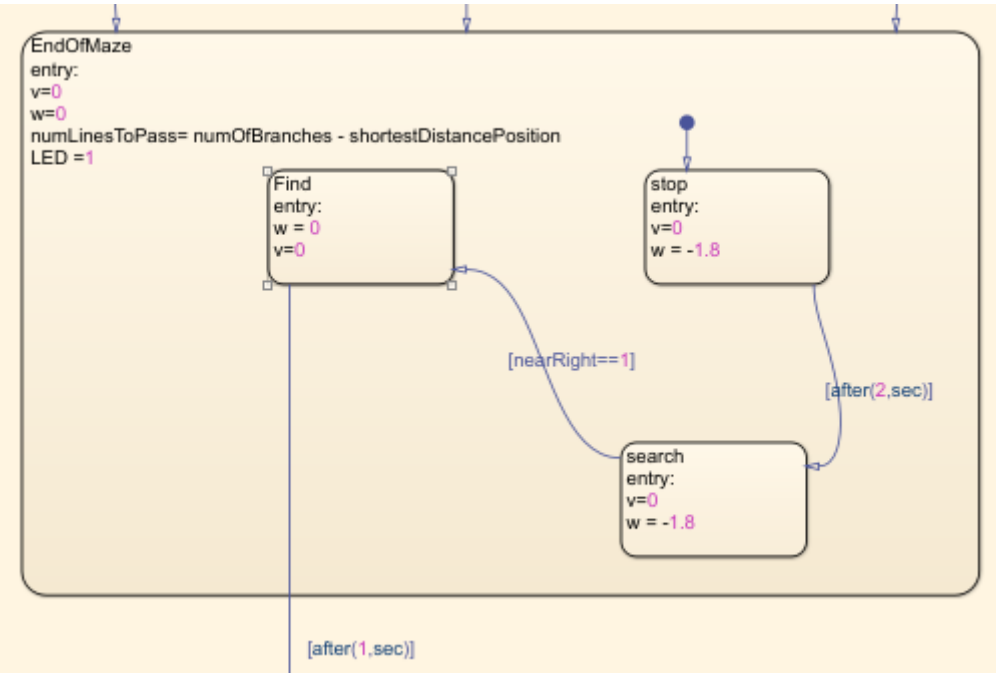


*Figure 16: Diagram showing relevant logic to find shortest object along with navigating back to main line*

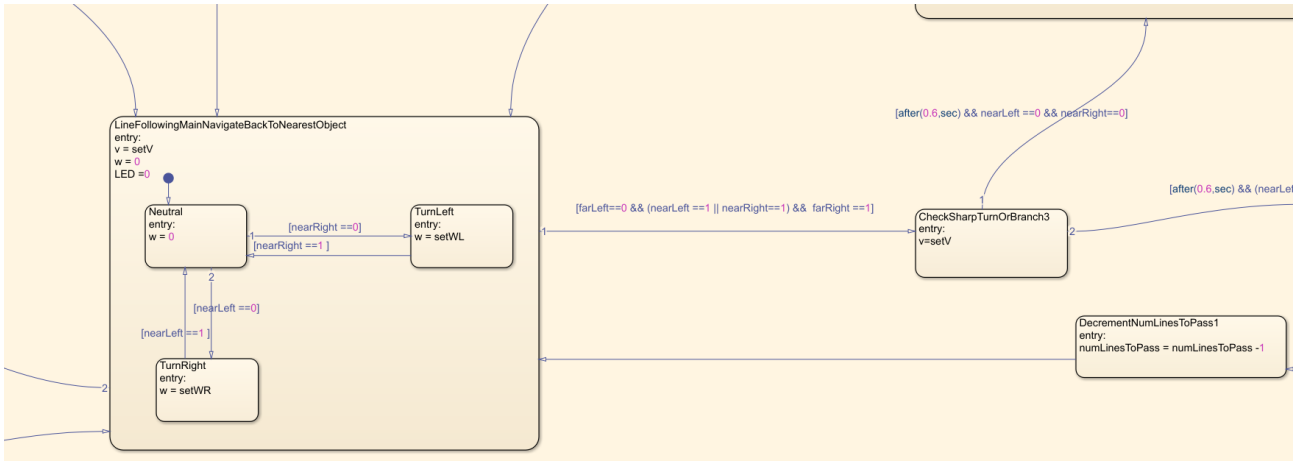


*Figure 17: Diagram showing identical line following algorithm as discussed prior*

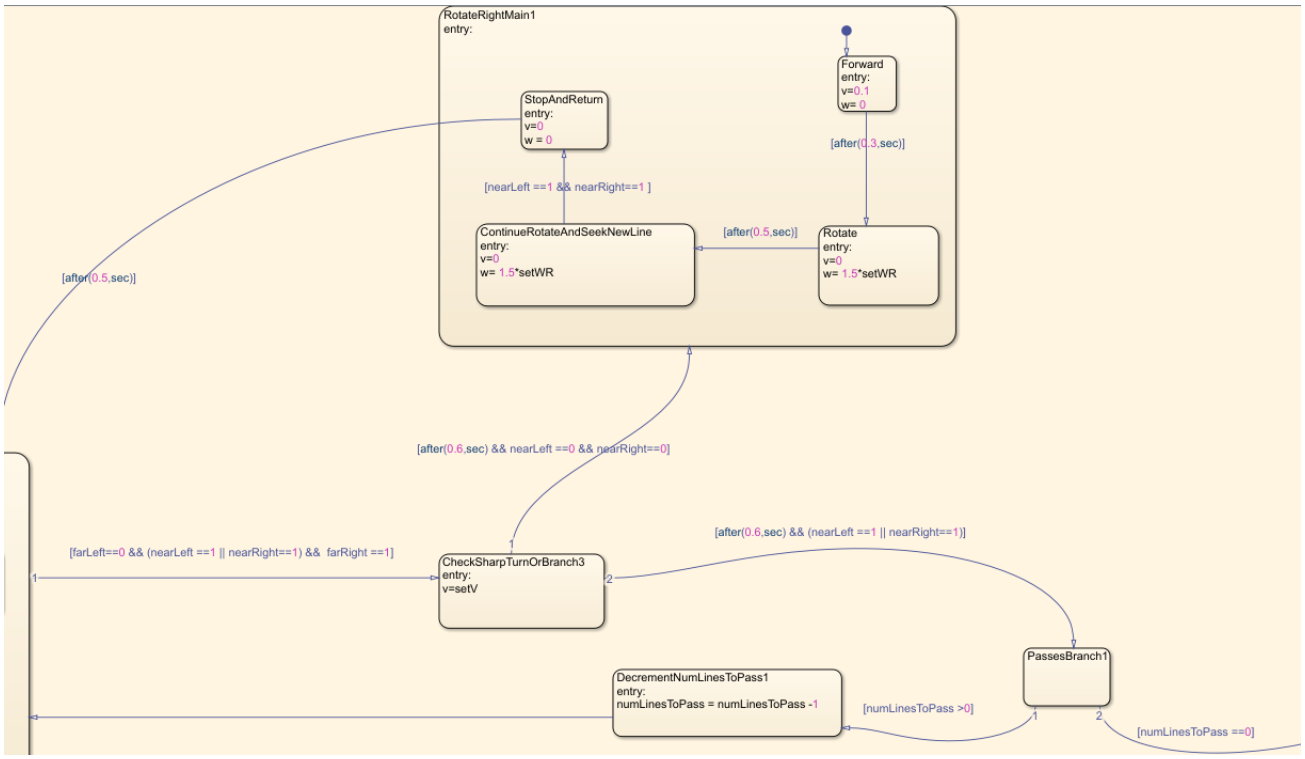


*Figure 18: Diagram showing the determination of a sharp corner or branch and the appropriate logic for branches*

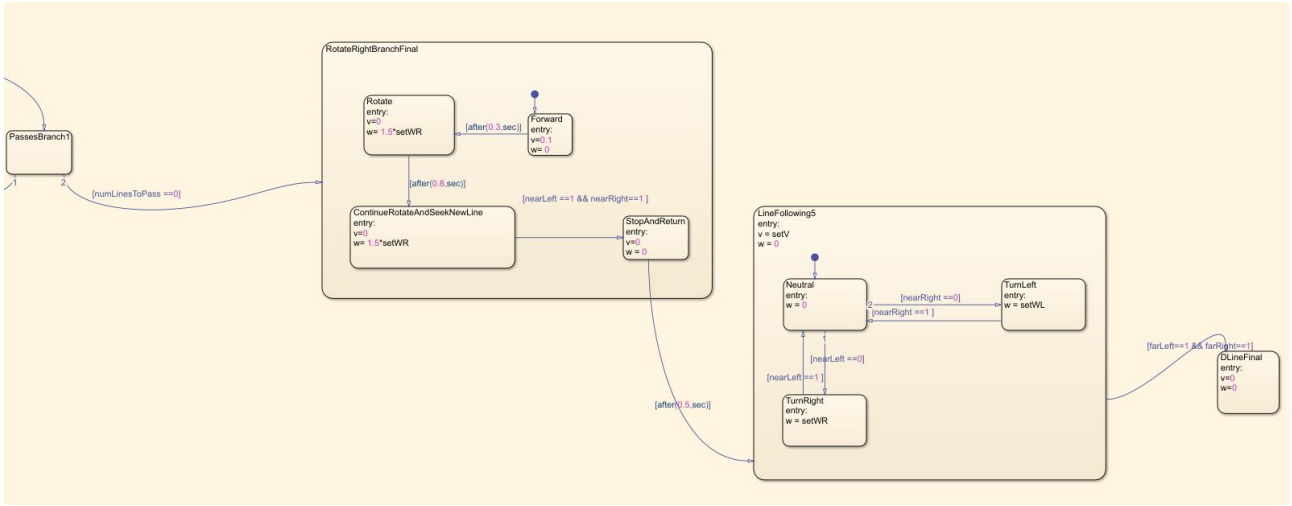


*Figure 19: Diagram showing how to navigate back to shortest object detect line*

## Results and justification:

The use of the number of branches variable and the shortest distance variable along with its corresponding logic allowed for the detection of objects while storing the shortest object without the need to utilise complex data types such as arrays. This successfully worked to store where the closest object was.

A waiting time during object detection is used to allow the robot to correctly measure the distance to different objects. This worked as intended.

Note, during rotations such as branches turns or $180^{o}$ there are two states of rotation where there is a waiting period between the periods. This allows the robots inner sensors to navigate of the previous branch it was on before it starts seeking the new branch. This prevents false triggers of finding the new line during rotation which successfully worked for all branches except during the demonstration for the last branch where the robot was meant to navigate back to the shortest object however, did not due to a false trigger.

The state MoveForwardSlightly allows the robot to move forward after detecting the main branch such that the robot remains aligned with the main line when it rotates again. This correctly helped align the robot.

# CONCLUSION

In conclusion, the design and implementation of the Arduino-based line maze follower has been detailed in this report. The system is tasked with navigating a maze, detecting branches, and locating objects of interest. Through careful consideration of subsystems, components, and power requirements, a robust system was assembled.

The sensor subsystem, comprising of a line and ultrasonic sensors, provided crucial data for navigation. The Arduino Nano 33 IoT served as the controller, facilitating communication and control between various components. Logic level converters ensured compatibility between different voltage levels, safeguarding the microcontroller.

Power management was a critical aspect, and a pair of 3.7V lithium-ion batteries was selected to power the system. The calculations demonstrated that the batteries could sustain approximately 6.67 hours of continuous operation, ensuring ample runtime for maze traversal.

The motion control algorithm, consisting of linear velocity and angular velocity variables, allowed precise control of the robot's movements. By adjusting these parameters, the robot followed lines, executed turns, and halted at detection points.

The line sensing system, with four strategically placed sensors, proved highly effective in distinguishing between black and white surfaces. This information was essential for making decisions at intersections, identifying dead ends, and detecting objects.

The treasure maze solving algorithm effectively managed branch detection, object distance measurement, and navigation back to the main line. Through careful logic and state management, the system was able to identify the shortest object and return to it reliably.

The implementation results demonstrated the effectiveness of the system. The line sensors exhibited rapid response times, while the motion control algorithm facilitated smooth and accurate movement. The algorithm for branch detection and object distance measurement performed consistently, with the waiting period during rotations ensuring precise measurements.

In summary, the line maze follower successfully combines sensor data processing, motion control, and decision-making algorithms to navigate mazes and locate objects. The integration of subsystems, power management, and effective algorithms culminate in a reliable and capable robot system.

# REFERENCES


Arduino (2023) *Nano 33 IOT, Arduino Documentation*. Available at: https://docs.arduino.cc/hardware/nano-33-iot (Accessed: 21 October 2023).

DFROBOT (no date a) *Line_tracking_sensor_for_arduino_v4_sku_sen0017, DFRobot*. Available at: https://wiki.dfrobot.com/Line_Tracking_Sensor_for_Arduino_V4_SKU_SEN0017 (Accessed: 22 October 2023).

DFROBOT (no date b) *Turtle: 2wd Arduino Mobile Robot Platform Wiki, DFRobot*. Available at: https://wiki.dfrobot.com/2WD_Mobile_Platform_for_Arduino__SKU_ROB0005_ (Accessed: 22 October 2023).

DFROBOT (no date b) *Wheel_encoders_for_dfrobot_3pa_and_4wd_rovers__sku_sen0038_* (no date) *DFRobot*. Available at: https://www.dfrobot.com/wiki/index.php/Wheel_Encoders_for_DFRobot_3PA_and_4WD_Rovers_(SKU:SEN0038) (Accessed: 22 October 2023).

DIYElectronics (2021) *Motor driver dual H-bridge module L298N, DIYElectronics*. Available at: https://www.diyelectronics.co.za/store/brushed-motor-drivers/34-motor-driverdual-h-bridge-module-l298n.html (Accessed: 21 October 2023).

Hullen, L.S. and N., Conradie, J. and Jc (2022) *Centurion Stellenbosch - Arduino importer, Micro Robotics*. Available at: https://www.robotics.org.za/ (Accessed: 22 October 2023).

PiHut (2013) *HC-SR04 ultrasonic range sensor on the Raspberry Pi, The Pi Hut*. Available at: https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi (Accessed: 20 October 2023).