

# **Final Design Report**

## **Micro mouse Milestone 3 and 4**



**Prepared by:**

John Michael(MCHJOH015) and Justin Cross (CRSKAT005)

**Prepared for:**

EEE3099S

**Department of Electrical Engineering**

**University of Cape Town**

**October 19, 2024**

## Contents

Chapter 1 .....	3
Introduction: .....	3
1.1 Problem Description .....	3
1.2 Scope and Limitations .....	3
1.3 GitHub Link.....	3
Chapter 2 .....	4
Requirement Analysis .....	4
2.1. Requirements.....	4
2.2. Specifications.....	5
2.3 Testing Procedures .....	6
Chapter 3 .....	7
3. System Integration.....	7
Chapter 4 .....	9
4.1 Design decisions .....	9
Hardware Design Decisions:.....	9
Software Design Decisions:.....	10
4.2 Testing and Results.....	13
4.3 Observation, Challenges and Solutions.....	15
4.3.1 AT01: Motor Test.....	15
4.3.2 AT02 & AT05: Line Detection Test and Sensor Calibration Test:.....	16
4.3.3 AT03, AT06 & AT08: Wall Detection, Crosstalk Prevention and LED Indicator Tests....	17
4.3.4 AT04, AT07&AT08: Intersection Detection, Mapping and LED Indicator Tests .....	17
Chapter 5 .....	18
5.1 Conclusion and Recommendations .....	18
Chapter 6 .....	19
6.1 Bibliography.....	19
6.2 Plagiarism declaration .....	19

# Chapter 1

## Introduction:

### 1.1 Problem Description

The purpose of the micro mouse is to autonomously navigate its way through a maze by using infrared sensors that are integrated with motors to manipulate wheel behaviour. Equipped with an STM32 motherboard, eight Infra-red (IR) sensors at the front, and two motors - the robot is designed to find its way out of a maze by interpreting reflection signals off maze solving lines and surrounding walls. The system's goal is to continuously adapt to its surroundings, avoiding collisions, following lines accurately, and ultimately solving the maze through a combination of precise sensor readings, responsive motor actions and the incorporation of a basic algorithm.

### 1.2 Scope and Limitations

The final objective of this project is to ensure that the micro mouse robot can successfully navigate a maze by combining line-following, wall detection, and intersection recognition. The micro mouse will use downward-facing infrared (IR) sensors to follow the line on the maze's floor, while the forward- and side-facing IR sensors will detect walls and obstacles ahead and to the sides of the robot. Additionally, downward-facing side sensors will be used to identify intersections, enabling the robot to make informed decisions about when to turn left, right, or continue straight, which in conjunction with a basic algorithm, will facilitate maze-solving capabilities.

The micro mouse robot faces several limitations, particularly regarding sensor performance and environmental factors. Varied lighting conditions can affect sensors, leading to inconsistent line and wall detection, while the sensitivity of the IR sensors, especially makes accurate detection even more challenging. The side-facing sensors are positioned farther inward on the sensing board, reducing their efficiency and causing delayed obstacle detection. Additionally, the forward and side sensor interference, or crosstalk, can result in false readings, especially when the sensors are required to work simultaneously.

### 1.3 GitHub Link

[https://github.com/justincross10/TheMouse\\_group53](https://github.com/justincross10/TheMouse_group53)

# Chapter 2

## Requirement Analysis

### 2.1. Requirements

The requirements for the micro mouse maze-solving system are shown in the table below:

Table 1:

Requirement ID	Description
UR01	The system must be able to follow a black line directly below it.
UR02	The system must be able to detect nearby walls in front of it.
UR03	The system must be able to detect nearby walls on the left- and right-hand side of it.
UR04	The system must be able to calibrate itself to adapt to its surroundings in order to solve the maze.
UR05	The sensing module must be integrated with the motor module to create a response system to sensing obstacles and lines.
UR06	The system must detect crossroads in the black tape to identify which direction is best to move.
UR07	The micro mouse must be able to solve the maze structure provided.

## 2.2. Specifications

The specifications, refined from the requirements table, for the maze solving process are shown in the table below:

Table 2:

Specification ID	Description
SP01	The mouse must remain above the black line at all times and realign itself if it moves off of it.
SP02	The system must be able to detect a wall in front of it when it reaches the crossroad nearest to that wall.
SP03	The system must be able to detect a wall on the left- and right-hand side of it at every crossroad.
SP04	The system will spend 10 seconds, before starting the maze solving process, calibrating itself to its surroundings.
SP05	The motors must be able to turn the mouse right, left and rotate 180 degrees when at crossroads. They must also correct the system's line following behaviour if it strays off the black line.
SP06	The system must come to a stop and analyse and determine the best path to move in at every crossroad to avoid hitting walls.
SP07	Using an algorithm known as the right-hand rule, the mouse must solve the maze.

## 2.3 Testing Procedures

A summary of the testing procedures taken is given in table below

Table 3:

Acceptance Test ID	Description
AT01	Motor test: A motor test was conducted to ensure the motors were functioning correctly .This involved pressing the SW1 button to start the test, allowing the robot to move in a circular motion.
AT02	Line Detection Test: Tests if the micro mouse can accurately detect and follow maze lines using downward-facing IR sensors.
AT03	Wall Detection Test: Evaluates the ability of forward, left, and right IR sensors to detect nearby walls accurately.
AT04	Intersection Detection Test: Tests if the micro mouse correctly stops at intersections and makes proper navigation decisions.
AT05	Sensor Calibration Test: Ensures sensors are properly calibrated in various lighting conditions (bright, dim, ambient).
AT06	Crosstalk Prevention Test: Verifies front and side sensors don't interfere, using debugging mode for confirmation.
AT07	Mapping Test: Tests micro mouse's right-hand rule navigation, ensuring it turns right when possible while hugging the wall.
AT08	LED Indicator Test: Tests the micro mouse's use of LEDs to show state: LED0 (line following), LED1 (intersection detection), LED2 (decision-making).

# Chapter 3

## 3. System Integration

In order for the micro mouse to perform as desired, each subsystem needed to be integrated together onto one motherboard to ensure a harmonious relationship between the modules.

A flow diagram showing the system integration relationships is given below:

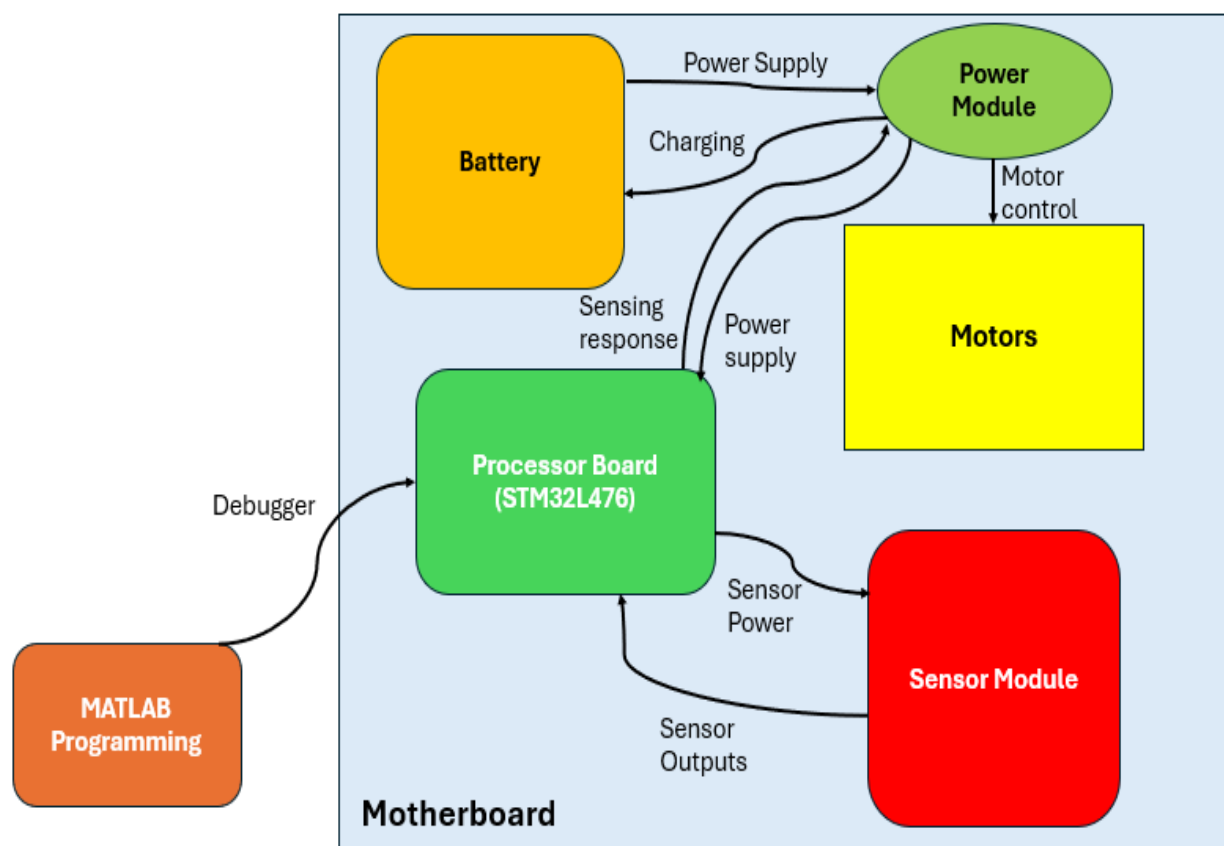


Diagram 1: Flow diagram of system integration.

The relationship between each module is further explained in the table below

Table 5: Table showing integrated relationships of modules.

Relationship	Description
MATLAB programming to Processor board	Connected through the debugger, this relationship stores desired programs on the STM controller using flash memory.
Battery to/from Power module	<ul style="list-style-type: none"> <li>Charging - The power module is responsible for charging the battery.</li> <li>Power supply - The battery supplies power to the power module.</li> </ul>
Power Module to Motors	The power module is responsible for supplying adequate amounts of power to the motors.
Processor Board to/from Sensor Module	<ul style="list-style-type: none"> <li>The Processor board is responsible for supplying the correct amounts of power to the desired sensing and handles sensor switching behaviour.</li> <li>The Sensor Module outputs the analogue voltage values from its surroundings to be analysed by the processor.</li> </ul>
Processor Board to Power Module	<ul style="list-style-type: none"> <li>The response to sensor readings is outputted from the Processor Board to the Power Module to alter motor behaviour.</li> <li>The Power Module supplies power to the processor board.</li> </ul>



# Chapter 4

## 4.1 Design decisions

### Hardware Design Decisions:

Table 6: Table showing hardware design decisions.

#	Hardware Design Decision	Description	Purpose	Outcome
1	Forward and Side-Facing IR Sensors Covered with Black Tape	Applied black tape over sensors to prevent interference.	Ensured sensors could operate simultaneously without crosstalk.	Simplified programming by eliminating sensor toggling.
2	Motor Facing IR Sensors Turned Downward	Side sensors faced the floor instead of detecting wheel movement.	Directly detected intersections and maze lines.	Improved navigation accuracy.
3	LED Indicators for Debugging	Incorporated three LEDs to indicate the current operational state.	Provided visual feedback for real-time debugging and monitoring.	Made troubleshooting easier during development.
4	Soldered for High Current Absorption	Adjusted infrared sensors for higher current absorption on Forward and side facing sensors.	Increased sensor range for better obstacle detection.	Improved sensor performance without excessive power usage.

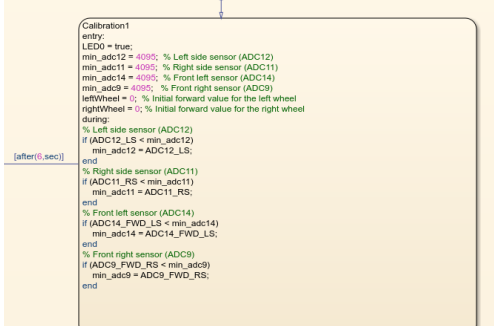
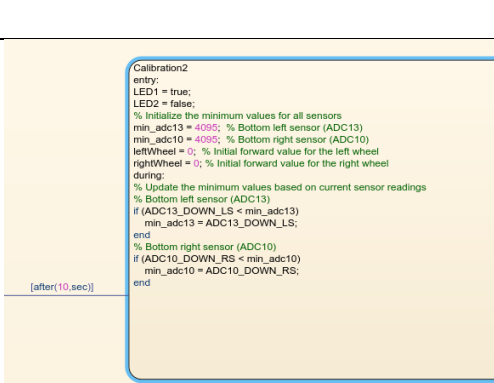
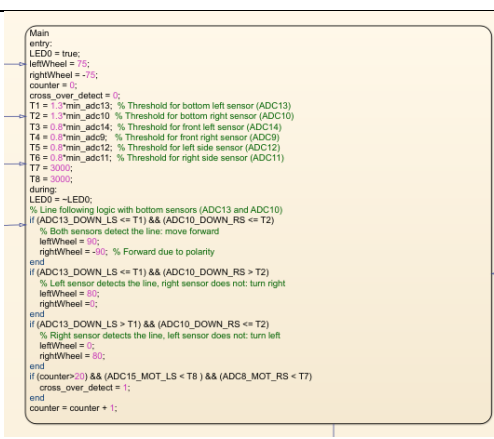
## Software Design Decisions:

Table 7: Table showing software design decisions.

#	Section	Description	Purpose	Outcome
1	Calibration Sate	Calibration for Lighting Conditions	Calibration adjusts sensor thresholds for different lighting environments.	Ensures accurate line/wall detection in varying conditions.
2	Crossover Detection State	LED Indicators for Debugging	LEDs indicate the current state (line-following, wall detection, etc.).	Provides visual feedback for real-time debugging.
3	Wall Detection State	Right-Hand Rule Implementation	Robot prioritizes right turns, following the right-hand rule for maze-solving.	Ensures consistent and predictable maze navigation.
4	Entire State flow	Timed State Transitions	Time-based delays (e.g., 0.8 sec) between state transitions.	Ensures the robot rotates the required rotation for the turns and provides delay for the robot to absorb value
5	Crossover Detection State	Crossover Detection Handling	Detects intersections or crossovers in the maze.	Prevents losing track of the line or maze path.
7	Straight State	Failsafe Conditions	Default behaviour which makes the robot go straight and then back into line following triggers if no wall or line is detected for a set time.	Ensures movement even without clear sensor input.
8	All States	Smooth Wheel Speed Control	Specific speed values for left and right wheels (e.g., 90 for left, 77 for right).	Ensures smooth transitions during turns and state changes.

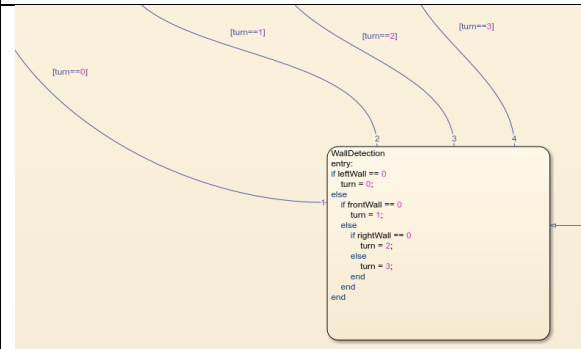
From the table above, the design decisions are further elaborated using the figures below:

Table 8: Table showing the Stateflow software design decisions from MATLAB.

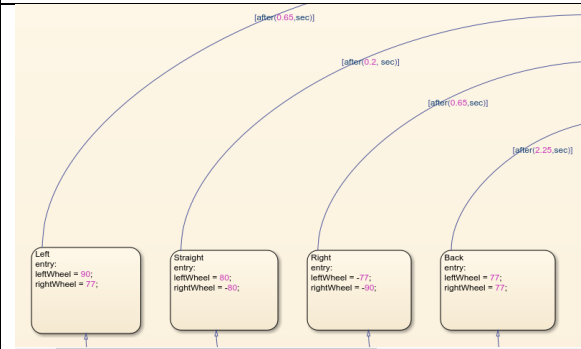
 <pre> Calibration1 entry: LED0 = true; min_adc12 = 4095; % Left side sensor (ADC12) min_adc11 = 4095; % Right side sensor (ADC11) min_adc14 = 4095; % Front left sensor (ADC14) min_adc9 = 4095; % Front right sensor (ADC9) leftWheel = 0; % Initial forward value for the left wheel rightWheel = 0; % Initial forward value for the right wheel during: % Left side sensor (ADC12) if (ADC12_DOWN_LS &lt; min_adc12) min_adc12 = ADC12_DOWN_LS; end % Right side sensor (ADC11) if (ADC11_DOWN_RS &lt; min_adc11) min_adc11 = ADC11_DOWN_RS; end % Front left sensor (ADC14) if (ADC14_FWD_LS &lt; min_adc14) min_adc14 = ADC14_FWD_LS; end % Front right sensor (ADC9) if (ADC9_FWD_RS &lt; min_adc9) min_adc9 = ADC9_FWD_RS; end end [after(0,sec)] </pre>	<p>Calibration1 sets threshold ADC values for the mouse that will be used for wall detection during the maze solving process. The mouse is placed in the presence of walls all around it for 6 seconds to accurately set threshold values.</p>
 <pre> Calibration2 entry: LED1 = true; LED2 = false; % Initialize the minimum values for all sensors min_adc13 = 4095; % Bottom left sensor (ADC13) min_adc10 = 4095; % Bottom right sensor (ADC10) leftWheel = 0; % Initial forward value for the left wheel rightWheel = 0; % Initial forward value for the right wheel during: % Update the minimum values based on current sensor readings % Bottom left sensor (ADC13) if (ADC13_DOWN_LS &lt; min_adc13) min_adc13 = ADC13_DOWN_LS; end % Bottom right sensor (ADC10) if (ADC10_DOWN_RS &lt; min_adc10) min_adc10 = ADC10_DOWN_RS; end end [after(10,sec)] </pre>	<p>Calibration2 sets the threshold ADC values for the line following system during the maze solving process. The mouse is manoeuvred over the black lines for 10 seconds to accurately set threshold values that will allow the mouse to successfully line follow.</p>
 <pre> Main entry: LED0 = true; leftWheel = 75; rightWheel = 75; counter = 0; cross_over_detect = 0; T1 = 1.0*min_adc13; % Threshold for bottom left sensor (ADC13) T2 = 1.0*min_adc10; % Threshold for bottom right sensor (ADC10) T3 = 0.0*min_adc14; % Threshold for front left sensor (ADC14) T4 = 0.0*min_adc9; % Threshold for front right sensor (ADC9) T5 = 0.0*min_adc12; % Threshold for left side sensor (ADC12) T6 = 0.0*min_adc11; % Threshold for right side sensor (ADC11) T7 = 3000; T8 = 3000; during: LED0 = ~LED0; % Line following logic with bottom sensors (ADC13 and ADC10) if (ADC13_DOWN_LS &lt;= T1) &amp;&amp; (ADC10_DOWN_RS &lt;= T2) % Both sensors detect the line: move forward leftWheel = 90; rightWheel = -90; % Forward due to polarity end if (ADC13_DOWN_LS &lt;= T1) &amp;&amp; (ADC10_DOWN_RS &gt; T2) % Left sensor detects the line, right sensor does not: turn right leftWheel = 80; rightWheel = -80; end if (ADC13_DOWN_LS &gt; T1) &amp;&amp; (ADC10_DOWN_RS &lt;= T2) % Right sensor detects the line, left sensor does not: turn left leftWheel = 80; rightWheel = 80; end if (counter&gt;20) &amp;&amp; (ADC15_MOT_LS &lt; T8) &amp;&amp; (ADC8_MOT_RS &lt; T7) cross_over_detect = 1; end counter = counter + 1; end </pre>	<p>The main block contains all line following programming for the mouse. The current ADC values measured by the mouse as it makes its way through the maze are compared with the threshold values to keep the mouse over the black lines. Wheel behaviour is adjusted for circumstances where the mouse moves off the black line.</p>



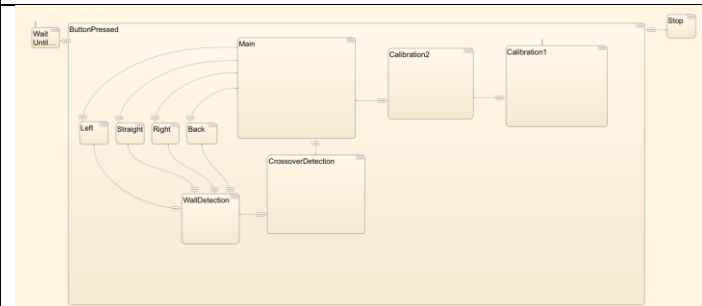
The CrossoverDetection state is achieved only if the mouse encounters a crossroad. The mouse spends one second analysing its wall surroundings, and different wall sensing variables are set at either 1 or 0, which will be used to determine where the system should move next.



WallDetection state is where the system applies the right-hand rule algorithm for maze solving. The mouse will favour turning right, if possible, if there is a right side wall, it will favour going straight, if there is also a front wall, the mouse will then turn left.



The different turning state blocks have timing mechanisms applied to them in order for the mouse to accurately turn 90 degrees or 180 degrees and remain on top of the black line.



The associated diagram depicts the overall relationship between all discussed software design decisions to create a unified State flow diagram for a successful maze solving micro mouse.

## 4.2 Testing and Results

Table 9: Table showing testing procedures and criteria.

Test ID	Description	Testing Procedure
AT01	Motor Test	<p>*Before programming begins this must be done</p> <ol style="list-style-type: none"><li>1. Press the SW1 button to initiate the motor test.</li><li>2. Observe the micro mouse as it moves in a circular motion for a predetermined duration.</li><li>3. Monitor the behaviour of the motors throughout the test, ensuring smooth operation.</li></ol>
AT02	Line Detection Test	<ol style="list-style-type: none"><li>1. Position the micro mouse at an intersection in the maze.</li><li>2. Check if LED 0 blinks, indicating that the micro mouse is in line-following mode.</li><li>3. Observe the micro mouse's performance as it attempts to follow the line.</li></ol>
AT03	Wall Detection Test:	<ol style="list-style-type: none"><li>1. Position the micro mouse near a front facing and 2 side facing wall in the maze.</li><li>2. Check the activation of the relevant LEDs:<ul style="list-style-type: none"><li>• LED 0 for right wall detection</li><li>• LED 1 for front wall detection</li><li>• LED 2 for left wall detection</li></ul></li><li>3. Observe the micro mouse's response as it approaches each wall.</li></ol>
AT04	Intersection detection test	<ol style="list-style-type: none"><li>1. Place the micro mouse at an intersection in the maze.</li><li>2. Check if LED 1 turns on, indicating that the program has moved into the intersection state.</li><li>3. Observe the micro mouse's behaviour:<ul style="list-style-type: none"><li>• The micro mouse should stop for 1 second at the intersection.</li><li>• After the pause, it should move smoothly onto the desired path.</li></ul></li></ol>

AT05	Sensor Calibration Test	<ol style="list-style-type: none"> <li>1. Place the micro mouse facing forward near three walls, with two side walls positioned accordingly.</li> <li>2. Allow the micro mouse to remain in this position for 6 seconds, during which LED 0 will illuminate, indicating that the forward and side sensors are being calibrated.</li> <li>3. After 6 seconds, LED 1 will turn on, indicating that manual intervention is required.</li> <li>4. Using your hands, gently move the micro mouse over the black lines to calibrate the bottom sensors.</li> </ol>
AT06	Crosstalk prevention test	<ol style="list-style-type: none"> <li>1. Place the micro mouse in the maze and enable debugging mode.</li> <li>2. Activate the front and side sensors simultaneously.</li> <li>3. Monitor the sensor readings to confirm that they operate independently without interference.</li> </ol>
AT07	Mapping Test	<ol style="list-style-type: none"> <li>1. Place the micro mouse at the starting point of the maze, oriented correctly.</li> <li>2. Calibrate the micro mouse to begin its navigation.</li> <li>3. Observe the micro mouse's behaviour: <ul style="list-style-type: none"> <li>• It should turn right whenever possible.</li> <li>• If it cannot turn right, it should go straight.</li> <li>• If it cannot go straight, it should turn left.</li> <li>• If it cannot turn left, it should perform a U-turn.</li> </ul> </li> </ol>
AT08	LED Indicator Test	<ol style="list-style-type: none"> <li>1. Activate the micro mouse and observe its behaviour in various scenarios: <p><b>LED 0:</b> Should light up when the micro mouse is in line-following mode.</p> <p><b>LED 1:</b> Should light up when the micro mouse detects an intersection.</p> <p><b>LED 2:</b> Should light up when the micro mouse is in decision-making mode(i.e turn right , left , straight or u turn )</p> </li> <li>2. Monitor the LED indicators as the micro mouse navigates the maze and responds to different situations.</li> </ol>

## 4.3 Observation, Challenges and Solutions

Below is a detailed analysis of the most crucial tests we conducted, exploring the challenges we encountered and the solutions we implemented.

### 4.3.1 AT01: Motor Test

\*Note: The speeds of the motors were measured on MATLAB using a duty cycle system, 100 being the fastest and 0 the slowest.

#### Observations when Testing

1. **Wiring Polarity:** The wheels had opposite polarity; the right wheel required a negative speed value to move forward, while the left wheel needed a positive value.
2. **Speed Variability:** Wheel speeds varied based on the assigned values, with a minimum of 75 needed for rotation and 80 being ideal for effective maze navigation.
3. **Inverted Variables:** The wheel variables were inverted, causing the left wheel to receive right wheel values and vice versa.
4. **Turning Mechanism:** By moving one wheel forward quickly while reversing the other slowly, we achieved sharp, quick turns.
5. **Rotation Efficiency:** At a speed of 90 forward and 77 backward, the robot could rotate 90 degrees in 0.65 seconds, while a speed of 77 on both wheels for 2.25 seconds allowed for a 180-degree U-turn.

#### Challenges

1. **Overshooting:** High speeds over 85 led to overshooting during line following, resulting in the robot losing the line.
2. **Sensor Reading Delays:** At high motor speeds, the robot didn't have enough time to accurately analyse and read surrounding values.
3. **Turn Detection Issues:** Overturning due to long rotation durations during left, right, or U-turns caused the robot to lose track of lines.

#### Solutions

1. **Optimal Speed Calibration:** We determined that a speed of 80 was optimal for line following to avoid overshooting.
2. **Fine-Tuning Turn Speeds:** Adjusted speeds to 90 for the forward moving wheel and 77 for the backwards moving wheel at for 0.65s to ensure consistent line detection after the turn, ensuring the robot maintained its path during manoeuvres.

3. **Testing and Adjustment:** Conducted multiple tests to refine exact speeds and timings for optimal performance.

#### 4.3.2 AT02 & AT05: Line Detection Test and Sensor Calibration Test:

##### Observations when Testing

1. **Speed Sensitivity:** Higher speeds (over 85) led to overshooting the line, causing the robot to lose track of the path.
2. **Time of Day Sensitivity:** The robot's sensors required different calibration values depending on the time of day, as varying light conditions affected their accuracy. As a result, we couldn't rely on fixed constants for sensor readings.
3. **Jittery Movement During Calibration:** At first, the robot attempted to calibrate itself by rotating in a circular motion whilst picking up threshold values, but it often displayed jittery movements instead of smooth line following, indicating issues with picking up the correct threshold values.

##### Challenges

1. **Loss of Line at High Speeds:** The robot was more likely to lose the line when the motors were running at speeds above 85.
2. **Inconsistent Line Following:** Under dim lighting, the sensors struggled to consistently detect lines and correct thresholds.
3. **Calibration Issues:** The robot occasionally failed to properly calibrate, which resulted in unreliable detection of line thresholds and jittery behaviour.

##### Solutions

1. **Optimized Speed:** The speed was reduced to 80, which provided a balance between smooth movement and accurate line following.
2. **Calibration Method:** The robot recorded the lowest sensor values from its surroundings and used these as thresholds. The forward and side sensors were calibrated in the presence of walls and the line following sensors were calibrated to a threshold based the ADC values obtained when the line was beneath it. Decisions were made based on whether sensor readings exceeded or fell below these values.
3. **Improved Calibration Method:** Rather than allowing the robot to rotate in circular motion for 6 seconds to detect threshold values, we manually moved the robot over the desired lines for 10s and placed the robot in the proximity of front and side walls. This ensured more accurate threshold detection and resolved the jittery movements during calibration. In order to improve the threshold values to account for different lighting conditions Wall detection thresholds were reduced by 20%, while line-following thresholds were increased by 30%, ensuring smoother and more reliable detection.



### 4.3.3 AT03, AT06 & AT08: Wall Detection, Crosstalk Prevention and LED Indicator Tests

#### Observations when Testing

1. **Sensor Positioning Issue:** The side-facing sensors were placed too far inward on the board, resulting in weak signals and inaccurate values when detecting nearby walls.
2. **Interference Between Sensors:** The forward and side sensors interfered with each other, causing inaccurate wall detection and random movements.

#### Challenges

1. **Weak Side Sensor Readings:** The placement of the side sensors caused weak readings, making it difficult to accurately detect walls.
2. **Sensor Crosstalk:** The forward and side sensors interfere with each other when operating simultaneously, leading to incorrect readings and unpredictable movements.
3. **Incorrect Turn Decisions:** The robot would sometimes turn in the wrong direction or make random movements due to faulty wall detection.

#### Solutions

1. **Soldering High current jumpers:** We soldered high-current jumpers to provide more current to both the side and front sensors, enabling them to detect walls at further distances.
2. **Physical Sensor Changes:** To resolve the interference between the sensors, we simplified the programming by placing black tubes over the forward and side sensors. This effectively eliminated crosstalk.
3. **LED Debugging:** Using LED0, LED1, and LED2 to visually show which walls were being detected at any given moment allowed us to identify issues in the state machine and correct them for more reliable wall detection.

### 4.3.4 AT04, AT07&AT08: Intersection Detection, Mapping and LED Indicator Tests

#### Observations when Testing

1. **Unsuccessful Navigation:** The robot navigated the maze randomly, failing to reach a designated endpoint.
2. **State Indication:** The robot showed unexpected random movements as it ended up in incorrect State flow blocks, resulting from faulty programming, calibration issues, and inaccurate sensor readings.

## Challenges

1. **Random Movements:** Initially, the robot displayed erratic movements, failing to solve the maze.
2. **Unclear Debugging:** Identifying the source of random behaviour was difficult due to the robot's unpredictable actions.
3. **Intersection Detection Failure:** The robot struggled to detect intersections, missing critical turns.

## Solutions

1. **Right-Hand Rule:** Implementing the right-hand rule allowed the robot to prioritize right turns, followed by straight, then left, and a U-turn if all directions were blocked, enabling it to navigate the maze effectively and reach a designated endpoint.
2. **LED Debugging:** Using LEDs to track states—LED0 for line following, LED1 for calibration, and LED2 for decision-making—helped pinpoint issues in the State flow logic, aiding in debugging.
3. **Improved Intersection Detection:** Initially, intersections weren't being detected effectively. We repositioned the side “motor” sensors downward, allowing the robot to stop every time these sensors picked up a black line. This greatly simplified the programming and improved the robot's ability to accurately detect and navigate intersections.

# Chapter 5

## 5.1 Conclusion and Recommendations

In terms of developing a micro mouse system that can solve a maze using line following and wall detection characteristics, this project was indeed a success. The mouse achieved the goal of autonomously navigating through a maze, with zero obstacle collisions using infrared sensing systems integrated with motor control. The calibration techniques used for the mouse to adapt itself to its surroundings and lighting conditions was successful, creating a flexible robot capable of handling a wider range of environmental scenarios. The right-hand rule algorithm that was applied to create a maze solving solution proved to be a simple and valuable solution to achieve the project's ultimate goal.

Overall, this project was a great success, but there are recommendations based on observations that were noted throughout the project duration. Although the maze solving goal was a success, the right-hand rule was far from the most optimal solution to solve the problem. Other techniques such as flood filling were significantly faster and

more efficient. The motors used were also very entry level motors, and many other stringer and faster motors could have been utilised to speed up the mouse. The design of the sensor board could also be adapted to reduce challenges regarding interference between sensors, whereby the side wall sensors could have been moved to the outer edges of the PCB to enhance side wall detection.

This project offered fantastic insight into the world of robotics and automation, providing interesting challenges that allowed for significant growth in the world of debugging, problem solving and optimisation.

## Chapter 6

### 6.1 Bibliography

matlabacademy.mathworks.com. (n.d.). Stateflow Onramp | Self-Paced Online Courses - MATLAB & Simulink. [online] Available at:  
<https://matlabacademy.mathworks.com/details/stateflow-onramp/stateflow>.

matlabacademy.mathworks.com. (n.d.). Self-Paced Online Courses. [online] Available at:  
<https://matlabacademy.mathworks.com/details/matlab-onramp/gettingstarted>.

matlabacademy.mathworks.com. (n.d.). Simulink Onramp | Self-Paced Online Courses - MATLAB & Simulink. [online] Available at:  
<https://matlabacademy.mathworks.com/details/simulink-onramp/simulink>.

### 6.2 Plagiarism declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.


---

Name Surname

19/10/2024

---

Date