Final Design Report

Micromouse Maze-Solving Project

Prepared by:

Chris Scheepers - SCHCHR077
Nang'alelwa Sitwala - STWNAN001

Prepared for:
EEE3097/8/9S
Department of Electrical Engineering
University of Cape Town

October 22, 2024

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.

October 22, 2024

Chris Scheepers, Nang'alelwa Sitwala — Date

Contents

Chapter 1

Introduction

1.1 Problem Description

This report refers to the third and fourth milestones in the overarching project from the UCT engineering design course, EEE3097/8/9S, that requires designing a solution for a robotic micromouse to solve a maze autonomously. Milestone 3 refers to taking the project's accomplishments from Milestone 2, which relied on the initial calibration and localisation aspect using IR sensors and the basic use of the motors to create a solution that takes this data to make informed turning decisions around the maze. Milestone 4 refers to and builds on the progress of Milestone 3 by creating a solution that optimises the path around the maze, based on the decisions made in Milestone 3. Simulink Stateflow was exclusively used throughout the project to generate the necessary control logic to achieve Milestones 2,3 and 4 due to its streamlined integrated development environment and capable debugging capabilities. The report will only refer to the software design and not refer to the design or production of the modulated subsystems that make up the micromouse system. The report will however, refer to an **Executive Summary** that will provide an overarching explanation of the solution chosen and designed for the project for this design course.

1.2 Scope and Limitations

As these were the final two milestones, the scope of this report is limited to and focuses on the following tasks:

- Understanding external and internal factors that affect the navigation and implementing a solution to mitigate these factors.
- Creating a solution that will allow the micromouse to traverse the maze's multiple pixels through the data gained from the IR sensor subsystem.
- Implementing decision-making methods for the micromouse.
- Using the decision-making methods to effectively turn and detect crossroads within the maze (the middle of the maze's pixel).
- Achieving adaptability for our solution to work in any configuration of the maze.
- Implementing contingencies in case of failure management for our solution is needed.
- Creating an optimisation method that is original and non-elementary.

Milestone 2 achievements - such as identifying and holding a line, calibration, and surrounding awareness are not a part of the scope of this report but will be discussed in the executive summary.

There were a multitude of types of limitations that affected the micromouse:

Design Limitations

- Size, weight, and power consumption constraints
- Battery capacity
- Quality of components such as the IR sensor chosen
- Processing power of STM microcontroller
- MatLab debugging bottlenecks due to underpowered hardware

Testing Limitations

- Edge case handling
- Continual calibration and tuning

Environmental Limitations

- Performance affected by environmental factors such as reflective surfaces and ambient light

1.3 Executive Summary

The design of this solution will be outlined, while referring to the various requirements of previous milestones of this project. Namely:

- Understanding the hardware, software and subsystems that make up the micromouse
- Calibrating IR sensor and motor subsystems
- Implementing line following
- Implementing localisation and surrounding awareness
- Turning and decision-making for maze traversal
- Maze-mapping and optimisation

1.3.1 Micromouse Hardware

The micromouse has been designed in such a way that integrated various subsystems into one maze-solving robot - namely, **motherboard**, **IR sensor**, **power** and **processing** subsystems. This allowed freedom of modulation and debugging while allowing the convenience of customisation if needed. An STM32 board was used alongside MatLab's Simulink/Stateflow to program and implement the solution to this project.

1.3.2 Calibration

The Micromouse is equipped with eight sensors, out of which only the two forward-facing and two down-facing sensors were utilized for the final navigation system. The decision to omit the side and motor sensors was due to difficulties in calibration and the environmental uncertainty that plagued our solution's efficacy. Furthermore, we saw the motor sensors as redundant to our solution's implementation.

Calibration was simple as we collected multiple samples of front sensor data and then found the average of our sample count to gain an average threshold value at the distance we chose to calibrate our micromouse.

The down-facing sensors required a more elegant approach as we would place our micromouse in the middle of a pixel of the maze and then force the micromouse to rotate a full rotation. While rotating, the micromouse would continuously gather the minimum (meaning black line) and maximum value (meaning no black line) for each down-facing sensor. These values were averaged and used for our line-following thresholds for the left and right down-facing sensors. The average of these thresholds was taken and used as the crossroad detection. To increase the reliability of these values, we placed a piece of cardboard over the IR sensor module to decrease the effect ambient light and external IR light had on our calibration process.

1.3.3 Line Following

Line following was simple to implement as we had thresholds for both left and right down-facing sensors - so whenever the ADC value was above the threshold for a given side, we knew that the mouse was straying to one side off of the black line and adjustments to the motors' speed could be implemented accordingly to correct the micromouse's course in realtime.

1.3.4 Surrounding Awareness

As we had successfully calibrated our front and down sensors, the micromouse's awareness relied upon staying in the middle of the maze's pixels and rotating at every crossroad to detect walls to its left, front and right. As we were given an untimed run to map and gain surrounding awareness and then an optimised run, we saw no need to implement a risky solution that depended on the external IR light. The micromouse was successfully able to gain surrounding awareness by stopping at every crossroad and making decisions based on IR sensor data.

1.3.5 Turning and Decision-Making

Our micromouse follows a **right-handed algorithm**. This means that based on a hierarchy, the mouse will attempt to turn right first, go straight second, left third and backward if all other options are not possible. Although slow, this allows an effective way of mapping due to time not being a requirement for the optimisation run. This was done by using crossroads to initiate the turning process. The front sensors turn to face each direction and make a decision as to whether there was a wall present or not.

1.3.6 Maze-Mapping and Optimisation

Following the previous subsection, the micromouse was given one run of the maze to map the maze and a second to complete the maze in the most optimal manner. Our implementation of maze mapping follows from the turning and decision-making logic where these choices (left, right, straight or backwards) were logged into a matrix until the end of the maze was found. At the beginning of the second run, an algorithm that will be explained further in the design section of this report, which can be found in the appendix, was implemented to find the shortest path. The algorithm takes predefined rules and continuously looks for ways to reduce the size of the 'turning/decision matrix' until it is the smallest that it could possibly be. On the second run, every time a crossroad is met, the micromouse scans the matrix and looks for the next direction that it needs to go - eliminating the need to turn its front sensors at every crossroad and possibly going in a non-optimal direction, greatly reducing the time the micromouse took to navigate the maze.

Chapter 2

Requirements Analysis

2.1 Requirements

The requirements for the milestone 3 and 4 micromouse maze-solving project are described in Table 2.1.

Table 2.1: Functional requirements for milestone 3 and 4.

Requirement ID	Description
R01	Movement of Micromouse
R02	Decision Making
R03	Turning
R04	Mapping of Maze
R05	Optimisation of Path

2.2 Specifications

The specifications, refined from the requirements in Table 2.1, for the micromouse infrared module are described in Table 2.2.

Table 2.2: Specifications of the milestone 2 project derived from the requirements in Table 2.1.

Specification ID	Type	Details
SP01	Sensors	Infrared (IR) Sensors
SP02	Detection Coverage	Front, Down
SP03	Real-time processing	Yes
SP04	Customisation	Adjustable calibration for different maze environments
SP05	Motors	Left and Right
SP06	Battery	3.7V Li-ion
SP07	Efficiency	Must be able to optimise path through maze
SP08	Maze-Solving	Must be able to first find the end of the maze then optimise its path through the maze on its second run.
SP09	Speed	Must be able to solve the maze in under 10 minutes.
SP10	Autonomy	Must be able to solve the maze without manual interference.

2.3 Testing Procedures

A summary of the testing procedures detailed in chapter 4 is given in Table 2.3.

Table 2.3: Testing Procedures

Acceptance Test ID	**Description**
AT01	Test **left**, **right**, **straight** and **backward** decision-making
AT02	Ambient light rejection to reduce instability
AT03	Test crossroad detection
AT04	Test turning
AT05	Test maze is solved in the required timeframe of 10 minutes
AT06	Test path is optimised following AT06

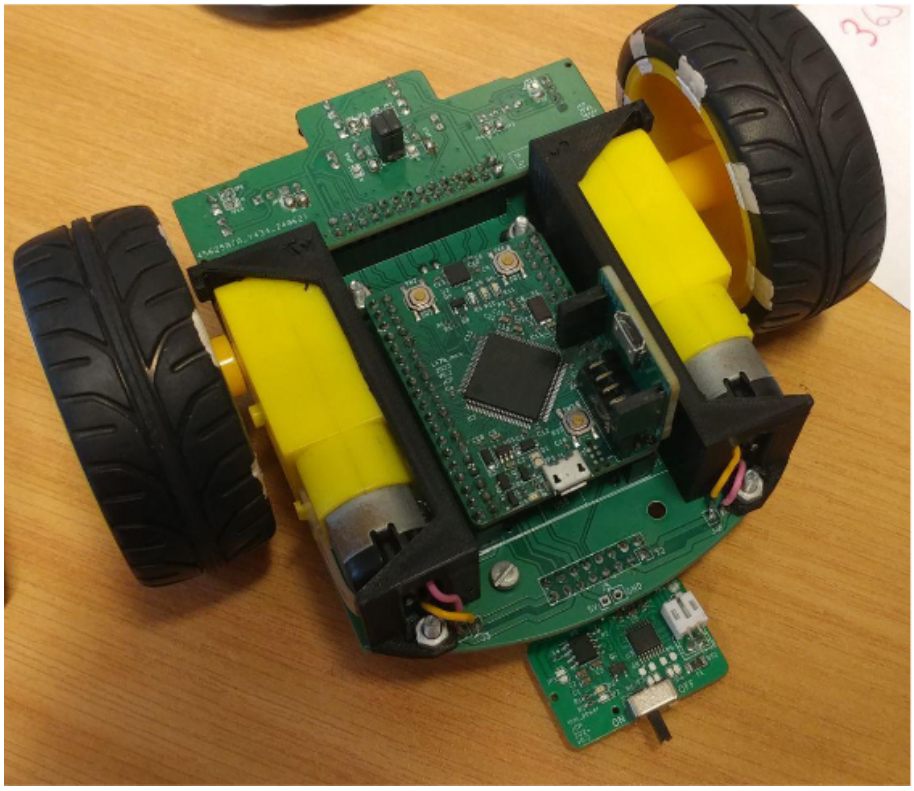

Figure 2.1: Unmodified Micromouse [1]

Chapter 3

Micromouse Design

3.1 Design and Implementation of Navigation

3.1.1 Calibration

The Micromouse is equipped with eight sensors, out of which only the two forward-facing and two down-facing sensors were utilized for the final navigation system. The decision to omit the side and motor sensors was due to difficulties in calibration and redundancy in the motor sensors.

Forward Sensors: These sensors are crucial for detecting walls in front of the Micromouse. Calibration is performed by placing the mouse near a wall and averaging multiple readings to set a threshold for wall detection. This threshold ensures that the mouse accurately identifies when it is near a wall and adjusts its movement accordingly.

Down Sensors: The down sensors are primarily used for detecting crossroads and intersections in the maze. Calibration involves rotating the mouse 360 degrees at a crossroad to capture the minimum and maximum sensor readings. The minimum reading corresponds to the black lines on the maze floor, and the system is configured to detect a crossroad when the sensor values drop below this minimum threshold.

3.1.2 Additional Design Choices

IR Sensor Module Cover: Due to external factors and varying IR lighting conditions due to position within the maze or time of day, a cardboard cover was placed over the IR sensor module in order to mitigate the effect that these factors would have on our calibration and data collection. Furthermore, by doing so, the reliability of our solution increased and became almost flawless regardless of the time of day.

Sled: The micromouse had one design flaw at the beginning of the semester. The back of the power module would drag on the floor when the micromouse moved and would sometimes catch on the black tape of the maze. To mitigate this issue, a raised sled was attached to the back of the micromouse to increase the height of the back of the micromouse. This eliminated the problem of catching the tape while also causing the sensing module to face a more downward angle, further reducing the effect of ambient IR light.

3.1.3 Line Detection

The maze floor is marked with black lines to help with navigation, and the downward sensors are essential for detecting these lines. The micromouse is equipped with two downward sensors: down-

left and down-right, which compare Analog-to-Digital Converter (ADC) values to determine if the micromouse is on the right path or deviating from it.

- **Line Detection Mechanism**: When the micromouse begins to veer off the intended path, one of the down sensors detects an increase in ADC value, indicating that it has gone off course. The micromouse then adjusts its wheel speeds to bring itself back on course.
  - If the down-left sensor detects an increase in ADC value, it means the micromouse has moved slightly left, and the right wheel speed is increased to correct the course.
  - If the down-right sensor detects an increase in ADC value, it indicates a rightward deviation, and the left wheel speed is increased to guide the micromouse back to the centre.
- **Continuous Adjustment**: By continuously comparing the readings from the down-left and down-right sensors, the micromouse can make fine adjustments to its wheel speeds, ensuring that it stays centred on the intended path. This dynamic correction helps the micromouse follow lines and navigate the maze without veering off course.

3.1.4 Crossroad Detection

For crossroad detection, the micromouse uses the down sensors to identify when it encounters the black lines at an intersection. During calibration, the micromouse records the minimum and maximum values of the down sensors as it spins 360 degrees at a crossroad. An average between the left and right down sensors is taken and used as the crossroad threshold. When navigating, a decrease in the ADC value below the minimum threshold indicates that the micromouse has detected a black line and reached a crossroad.

3.1.5 Navigation Strategy

The micromouse employs a right wall follower algorithm as its primary navigation strategy. This method is a classic approach for maze solving where the micromouse continuously keeps a wall on its right side and makes decisions at every intersection based on the presence or absence of walls. The right wall follower is particularly useful in simply connected mazes (without loops) and ensures that the micromouse can eventually reach the maze's centre by following the outer wall.

The algorithm can be broken down into the following key steps and can be found in Figure 1:

1. **Move Forward**: The micromouse always attempts to move forward until it encounters an intersection. The line detection is used to keep it on its path and detect the crossroads.
2. **Turn Right at Intersections**: Whenever the micromouse reaches an intersection, its first preference is to turn right. Using the forward sensors, the micromouse checks if it can make a right turn by detecting the presence or absence of a wall to its right. The process is as follows:
   - If no wall is detected to the right: The micromouse will make a right turn and continue forward.
   - If a wall is detected to the right: The micromouse will not turn and will continue checking other directions.

3. **Check Forward After a Failed Right Turn**:
    - After determining that it cannot turn right (because a wall is present), the micromouse checks if it can continue moving straight by sensing the wall in front.
    - If no wall is detected ahead: The micromouse will move forward.
    - If a wall is detected ahead: The micromouse will attempt the next possible direction, which is to turn left.
4. **Turn Left When Both Right and Forward Are Blocked**:
    - If the micromouse cannot turn right and also finds a wall directly in front, it will then turn left and attempt to move forward again.
5. **Turn Around When All Directions Are Blocked**:
    - In cases where the micromouse is blocked in all three directions (right, forward, and left), it will perform a 180-degree turn to retrace its steps.

The micromouse implements this strategy continuously throughout the maze until it reaches its destination. Example of the Right Wall Follower in Action:

- **Initial Position**: The micromouse starts at the edge of the maze, with a wall to its right.
- **First Move**: The micromouse moves forward, keeping the wall on its right, and checks if it can turn right at every intersection.
- **Intersection**: Upon reaching an intersection, the micromouse attempts to turn right. If the right path is open (no wall detected), it turns and continues forward.
- **Dead-End**: When a dead-end is encountered (with walls on the right, forward, and left), the micromouse turns around and retraces its path until it finds another open direction.
- **Crossroads and Adjustments**: Using the forward sensors for wall detection and the down sensors for black line detection (discussed previously), the micromouse continues making adjustments, ensuring that it follows the wall to its right while navigating through the maze.

3.2 Design and Implementation of Optimisation

3.2.1 Maze-Mapping

The decisions made using the **Right-Hand Algorithm** at crossroads were logged into a matrix using a number system but for the purpose of simplicity and readability, this report will refer to these decisions as letters:

- 1 - Micromouse turned **right** (R)
- 2 - Micromouse remained **straight** (S)
- 3 - Micromouse turned **left** (L)
- 4 - Micromouse turned **backwards** (B)

As the maze end was a 2x2 square, the robot knew that it had reached the end of the maze in its first run when one of two situations occurred:

- RLLL
- SLLL

As this combination of consecutive moves was impossible anywhere else in the maze. Once the micromouse knows that it has finished its first run, the last 4 turning decisions are removed from the path matrix and the micromouse sits in a 'ready state' until the second run begins.

3.2.2 Optimisation

Using an algorithm with the following predefined rules:

- LBR = B
- RBL = B
- SBS = B
- RBS = L
- SBR = L
- LBL = S
- RBR = S
- SBL = R
- LBS = R

The algorithm utilises the original path, the newly optimised path and the temporary previous path matrices (initially the original path). The algorithm is applied to the original path until it reaches the end of the matrix. The original path is then assigned to the optimised matrix and then it compares this optimised matrix to the temporary previous path matrix and continues the algorithm if they are

different or breaks the check if they are the same (no change could be applied). This loop continues until no more changes can be made.

The algorithm will repeatedly apply these rules to the path matrix to remove decisions from the matrix until no more reductions can occur - resulting in the most optimal path. It is important to note that this algorithm does not rely on the micromouse having been at positions in the maze where the optimal path would occur as the hierarchal nature of the **Right-Hand Algorithm** only needs to reach the end of the maze. This means that the micromouse could take a completely different route if the maze is laid out in such a way that this is possible. A MatLab script was written and implemented as a function in Stateflow to achieve this as Stateflow lacks the functionality to achieve this. The function is called in Stateflow and the output at any given index of the decision state results in a transition that outputs the most optimal decision to complete the maze. The MatLab script can be seen in Listing 1.

This solution is infinitely scalable and does not require prior knowledge of the maze before the optimisation run. This algorithm can also be implemented in any variation of a maze as the number of elements in the matrix will always equal the number of crossroads that the micromouse has to make a decision at. Furthermore, this solution allows the micromouse to immediately make decisions at crossroads and eliminated the need to check its right, left or front before making decisions - greatly increasing the maze-solving speed.

For example, Figure 3.1 shows a maze that follows the path: [R, L, R, R, B, S, B, R, L, R, R]

The algorithm would continuously optimise until all that was left in the optimised matrix was: [S]

This algorithm was adapted from a **Left-hand based algorithm**, however this algorithm could not keep data of turns where only one direction was available nor function properly if there was a loop in the maze [2]. The Stateflow Diagram for this algorithm can be found in Figure 2.

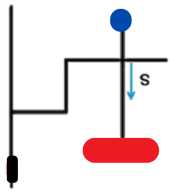

Figure 3.1: Solved Example Maze

3.3 Failure Management

Failure management was an integral part of the design process, as there were many limitations and challenges throughout the solution's design. Measures taken to mitigate and eliminate failure are presented in table form below:

Table 3.1: Failure Management for the Micromouse System

Name	**Description**
Redundancy	Multiple sensors for front and down direction detection are used.
Loop Detection	Logic has been implemented to detect repetitive patterns and break if needed.
Hardware Modulation	IR sensor and motor modules have been split in case of failure of one module and a replacement is needed.
Software Modulation	Code has been subdivided into manageable and independent subsystems for easy debugging.
Data Noise Reduction	Calculations and means of collecting sensor data were done in ways to minimise noise
Sensor Threshold Consistency	• Calibration was done in real-time using large data samples as opposed to predefined thresholds. • Thresholds were coded to adjust to acknowledge noise
External Factor Mitigation	• A cover was placed over the IR sensor module to protect and reduce IR ambient light. • A sled was placed at the back of the micrmouse to eliminate the possibility of the micromouse catching on the black tape and reducing the effect of ambient IR light through the change of micromouse angle.

Chapter 4

Acceptance Testing

4.1 Tests

Table 4.1: Milestone 3 and 4 acceptance tests

Test ID	Description	Testing Procedure	Pass/Fail Criteria
AT01	Test **left**, **right**, **straight** and **backward** decision-making	• Place the micromouse at different crossroads with multiple options to turn	• Micromouse must follow the hierarchy of turning (right, straight, left, backward)
AT02	Ambient light rejection to reduce instability	• Run AT01 multiple times throughout different times of the day in different parts of the maze	• Must not fail to follow turning decision hierarchy
AT03	Test crossroad detection	• Repeat AT1 and AT2 multiple times	• Micromouse must stop at a crossroad
AT04	Test turning	• Create a maze that forces all 4 types of turning consecutively and put the micromouse in it	• The micromouse cannot hit a wall • The micromouse must not stray from the black line
AT05	Test maze is solved in the required timeframe of 10 minutes	• Make the micromouse do a full maze-solving run with the optimisation run included from different positions of the maze	• The time to complete is less than 10 minutes.
AT06	Test path is optimised following AT06	• Complete AT06 and check whether the path is the most optimised by running the decisions through the optimisation MatLab script	• The path cannot to the centre of the maze cannot be optimised any further.

4.2 Critical Analysis of Testing

4.2.1 AT01

The turning decision-making worked well and passed with flying colours. As the logic for the decision-making was based in Stateflow, the implementation of the hierarchical control logic was simple. The micromouse correctly followed the hierarchy of:

1. Right
2. Straight

Table 4.2: Milestone 3 and 4 acceptance test results

Test ID	Description	Result
AT01	Test **left**, **right**, **straight** and **backward** decision-making	Pass
AT02	Ambient light rejection to reduce instability	Pass
AT03	Test crossroad detection	Pass
AT04	Test turning	Pass
AT05	Test maze is solved in the required timeframe of 10 minutes	Pass
AT06	Test path is optimised following AT06	Pass

3. Left
4. Backwards

everytime and proved not to cause any issues.

4.2.2 AT02

The implementation of a cardboard cover over the IR sensor module and the placement of the raised sled to decrease the sensors' exposure to ambient IR light significantly improved the consistency and reliability of the micromouse's sensor data. Testing the micromouse throughout various times of the day proved to yield the same result, leading to almost no false detections and reduced the instability of the data.

4.2.3 AT03

The micromouse passed AT03 every time that it was tested. The solidity of this result served as the backbone of our remaining design implementation as the crossroad detection was so reliable. If this was not the case, the optimisation algorithm implemented for this solution would not have been possible.

4.2.4 AT04

Through tinkering, this test was passed in a very reliable fashion. The design choice to stop at the crossroads before turning made the reliability of the micromouse's turning far greater. The mouse did not hit adjacent walls and if it slightly strayed from the black line, the line following algorithm was designed well enough to make the adjustment and never fully allow the micromouse to lose the black line.

4.2.5 AT05

The micromouse passed this crucial test with flying colours, with the average maze-solve taking roughly 2 minutes, including both mapping and optimisation runs. While not the fastest, the micromouse spent

most of those 2 minutes mapping the maze while the second run was significantly faster.

4.2.6 AT06

The micromouse passed AT06 but this is expected as the algorithm is designed to find the most optimal path. The only possible issue that could arise from this test would be if a previous acceptance test failed or the implementation of the algorithm was logically flawed but fortunately, they did not and the implementation was sound.

4.3 Milestone 3 and 4 Milestone Achievements

As discussed in Table 4.1, which outlines the requirements and the tests to determine whether these requirements were successfully filled, below is a simplified table of the solution's achievements with regards to the requirements laid out for milestone 3 and 4:

Table 4.3: Requirements Achieved for Milestone 3 and 4.

Requirement ID	**Description**	**Result**
R01	Movement of Micromouse	Pass
R02	Decision Making	Pass
R03	Turning	Pass
R04	Mapping of Maze	Pass
R05	Optimisation of Path	Pass

Chapter 5

Conclusion

The overarching focus of this project was designing and implementing a micromouse that could accurately navigate maze environments using a right-wall follower algorithm. The mouse was equipped with forward and downward infrared (IR) sensors to detect walls and identify crossroads. The downward sensors were calibrated using a 360-degree rotation. The forward sensors effectively detected obstacles, but calibration was challenging due to ambient light interference. Despite several attempts, the side sensors were difficult to fine-tune and were eventually not included in the final navigation system. Additionally, motor sensors were considered unnecessary and were not integrated due to their redundancy. The micromouse consistently navigated the maze with precise wall detection and reliable line following in most test scenarios. The mouse effectively navigated crossroads and responded well to different maze configurations, showcasing the success of the design. This project lays a solid groundwork for future improvements in sensor performance and navigational speed.

The reliability of the micromouse's navigation and decision-making allowed the ease of implementing the optimisation algorithm which worked flawlessly and showcased the effectiveness and efficiency of the algorithm - however, the speed of the micromouse could be seen as lacking.

5.1 Recommendations

1. **Improved Sensor Calibration**: Future iterations should explore advanced calibration techniques to mitigate ambient light interference, particularly for side sensors. Introducing adaptive calibration algorithms that account for changes in environmental conditions could improve overall sensor accuracy.

2. **Incorporating Side Sensors**: Although side sensors were excluded due to calibration challenges, they could enhance the micromouse's navigation capabilities, especially in more complex maze configurations. Fine-tuning their sensitivity and integrating them into the navigation algorithm could further refine the micromouse's performance.

3. **Optimization of the Right-Wall Follower Algorithm**: While the right-wall follower algorithm worked well, it was slow and there is room for optimisation. Future work could involve exploring alternative navigation strategies, such as dynamic path planning or using machine learning to optimize the micromouse's decision-making process.

4. **Sensor Fusion**: Combining data from multiple sensors, such as integrating IMU (Inertial Measurement Unit) readings with IR sensor data, could provide more robust environmental awareness. This would allow the micromouse to handle more complex mazes with fewer recalibration needs.

Chapter 6

Appendix

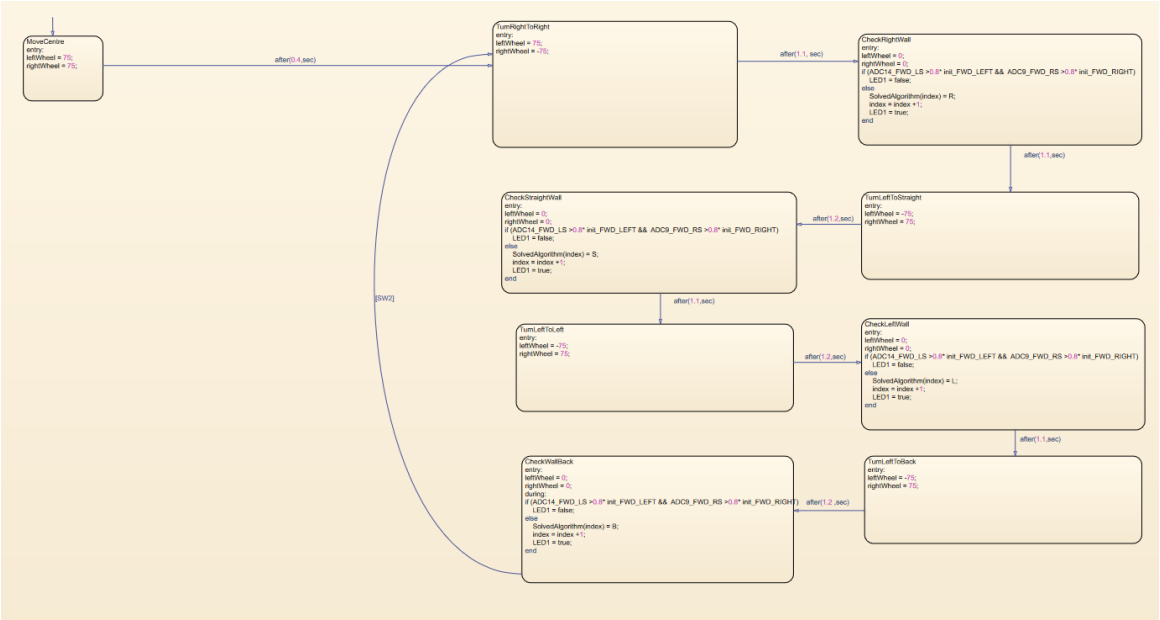

Figure 1: Turning Logic for Right-Hand Algorithm

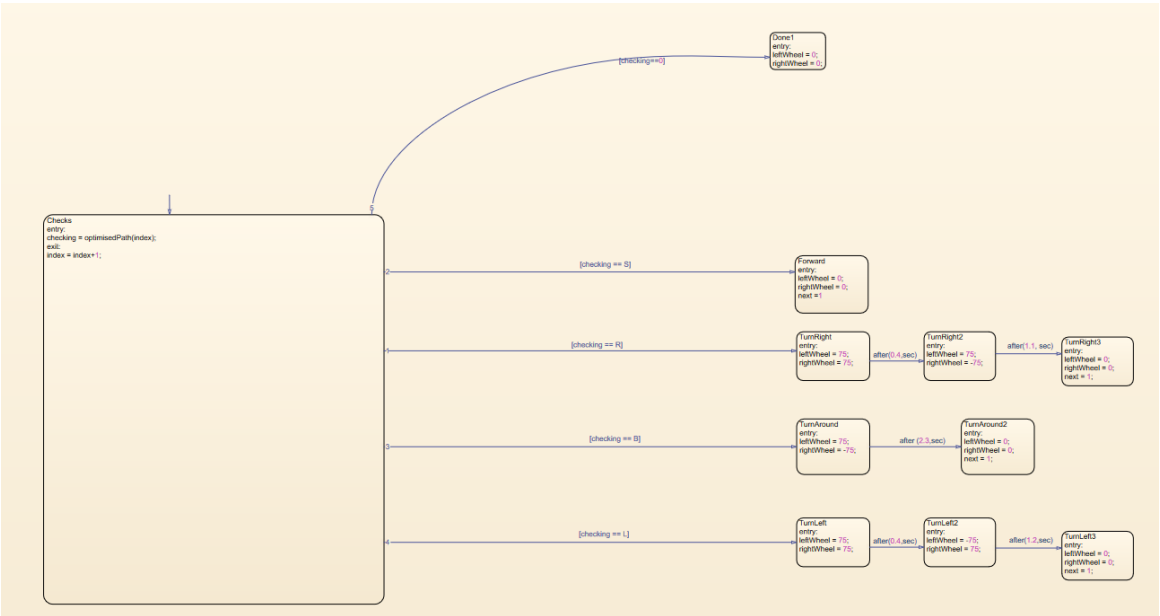

Figure 2: Maze-Solving Logic for Run 2 (next continues driving state)

```
function optimized_path = optimize_path(input_path)
    % This function takes a 1D array of integers as input and applies
    % optimization rules repeatedly until no further optimizations can be applied.

    % input_path is a 1D array of integers: 1=R, 2=S, 3=L, 4=B
    % optimized_path will be the same 1D array after applying rules, filled with zeros initially

    % Initial setup
    previous_path = input_path;
    optimized_path = zeros(size(input_path)); % Preallocate the optimized path

    % Keep applying optimization until the path no longer changes
    while true
        % Index for the optimized path
        opt_index = 1;
        i = 1;

        % Traverse the path and apply the replacement rules
        while i <= length(previous_path)
            % Check if there are enough elements left to apply a rule (3 elements)
            if i <= length(previous_path) - 2
                % Check for LBR -> B (3 4 1 -> 4)
                if isequal(previous_path(i:i+2), [3, 4, 1])
                    optimized_path(opt_index) = 4;  % Replace with B
                    i = i + 3;  % Move forward by 3 elements in the input path
                % Check for RBL -> B (1 4 3 -> 4)
                elseif isequal(previous_path(i:i+2), [1, 4, 3])
                    optimized_path(opt_index) = 4;  % Replace with B
                    i = i + 3;
                % Check for SBS -> B (2 4 2 -> 4)
                elseif isequal(previous_path(i:i+2), [2, 4, 2])
                    optimized_path(opt_index) = 4;  % Replace with B
                    i = i + 3;
                % Check for RBS -> L (1 4 2 -> 3)
                elseif isequal(previous_path(i:i+2), [1, 4, 2])
                    optimized_path(opt_index) = 3;  % Replace with L
                    i = i + 3;
                % Check for SBR -> L (2 4 1 -> 3)
                elseif isequal(previous_path(i:i+2), [2, 4, 1])
                    optimized_path(opt_index) = 3;  % Replace with L
                    i = i + 3;
                % Check for LBL -> S (3 4 3 -> 2)
                elseif isequal(previous_path(i:i+2), [3, 4, 3])
                    optimized_path(opt_index) = 2;  % Replace with S
                    i = i + 3;
                % Check for RBR -> S (1 4 1 -> 2)
                elseif isequal(previous_path(i:i+2), [1, 4, 1])
                    optimized_path(opt_index) = 2;  % Replace with S
                    i = i + 3;
                else
                    % If no rule is applied, just copy the current element
                    optimized_path(opt_index) = previous_path(i);
```

```
                    i = i + 1;  % Move to the next element in input path
                end
            else
                % For the last 2 or fewer elements, just copy them directly
                optimized_path(opt_index) = previous_path(i);
                i = i + 1;  % Move to the next element in input path
            end

            % Increment the index for the optimized path
            opt_index = opt_index + 1;
        end

        % If no changes were made, break the loop
        if isequal(previous_path, optimized_path)
            break;
        end

        % Update previous_path for the next iteration
        previous_path = optimized_path;
        optimized_path = zeros(size(previous_path)); % Reset optimized path
    end
end
```

Listing 1: MatLab Optimisation Script

Bibliography

[1] U. of Cape Town Electrical Engineering Department, "Complete micromouse example," Available online, August 2024. [Online]. Available: https://www.uct.ac.za/

[2] I. N. Gabriel Harja, "Autonomous maze solving robot," Available online, May 2020. [Online]. Available: https://www.researchgate.net/publication/342614026_Autonomous_Maze_Solving_Robot