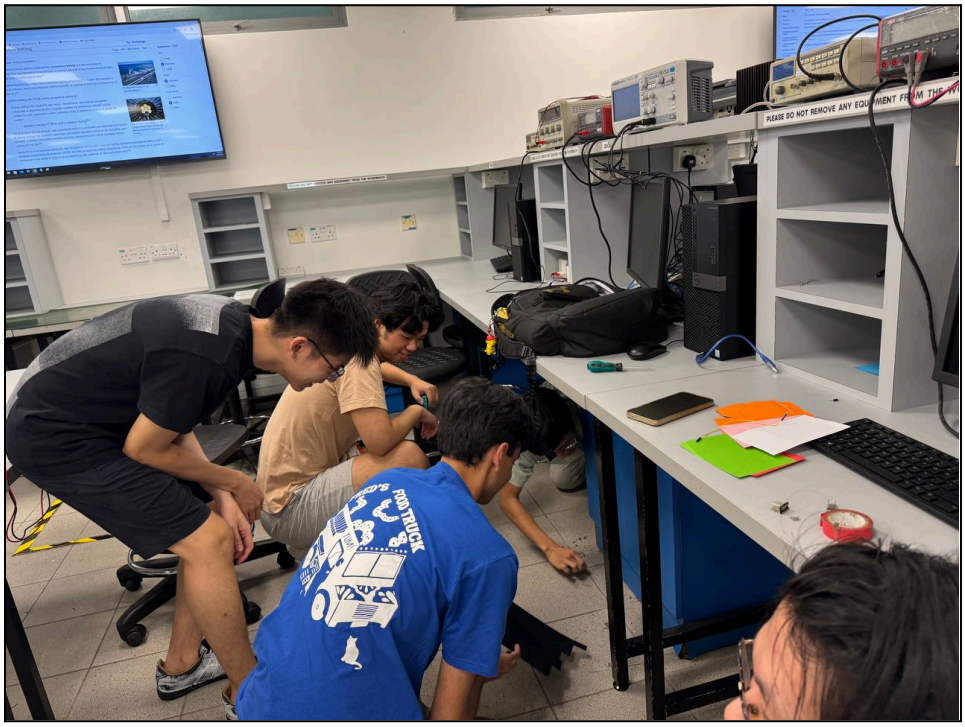
CG1111A Engineering Principles & Practice I
The A-maze-ing Race Project 2024



Group Numbers

Studio Group Number	B04
Section Number	Section 2
Team Number	Team 2

Team Members

Member 1	Tirodkar Om Milind
Member 2	Toh Ee Sen, Izen
Member 3	Thia Yang Han
Member 4	Tjhin Brian

Table of Contents

1. MBot Design..... 3

    1.1 Top and Bottom View..... 4

    1.2 Sensor Breadboard Circuits..... 5

    1.3 Overall Circuit Diagram..... 6

2. Description of Overall Algorithm..... 7

3. Specific Implementation Details..... 9

    3.1 Wall Avoidance..... 9

    3.2 Colour Sensing..... 10

    3.3 Ultrasonic Sensing..... 12

    3.4 Infrared Proximity Sensing..... 13

    3.5 Rotation..... 15

4. Calibration Process..... 16

    4.1 Colour Sensor..... 16

    4.2 Infrared Sensor..... 16

    4.3 PID Tuning..... 16

5. Improving Robustness of Algorithms..... 17

    5.1 Colour Classifying Algorithm..... 17

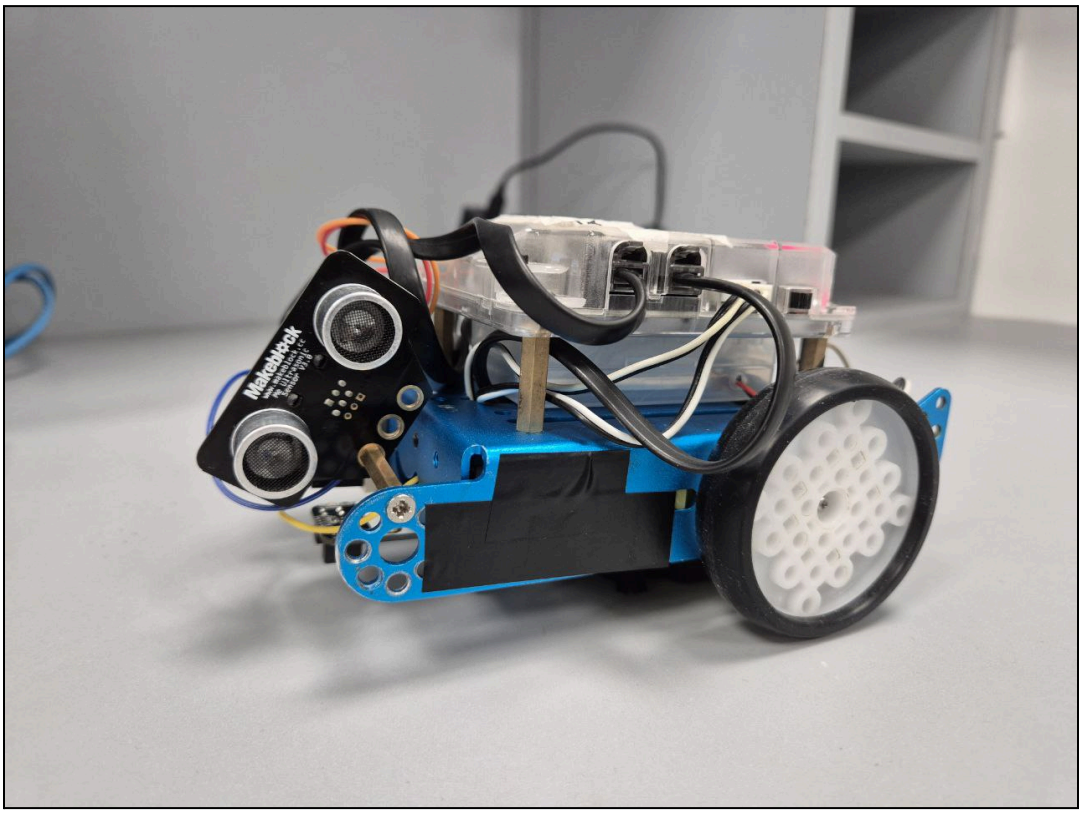
    5.2 Path Correction Algorithm..... 18

6. Challenges Faced..... 19

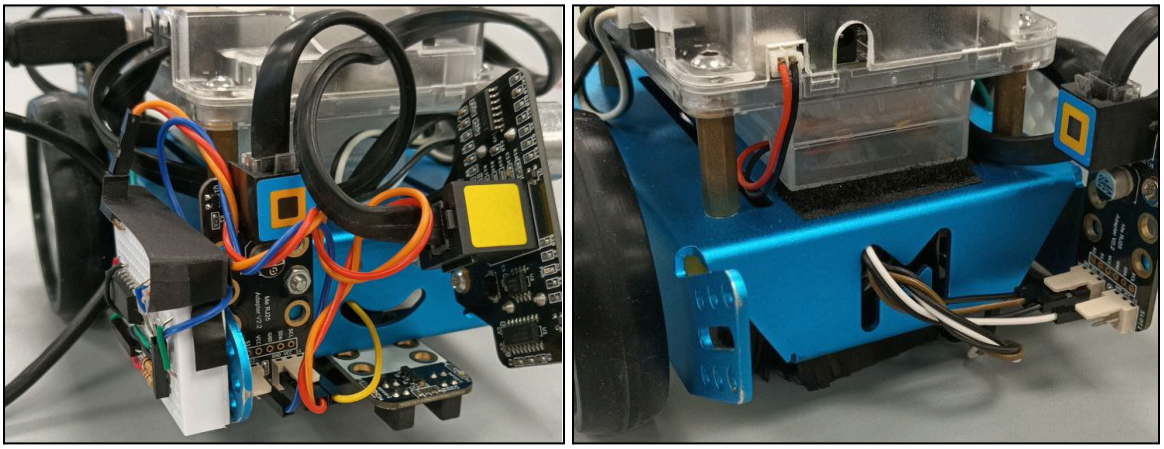
7. Work Division..... 22

8. Conclusion..... 23

1. MBot Design

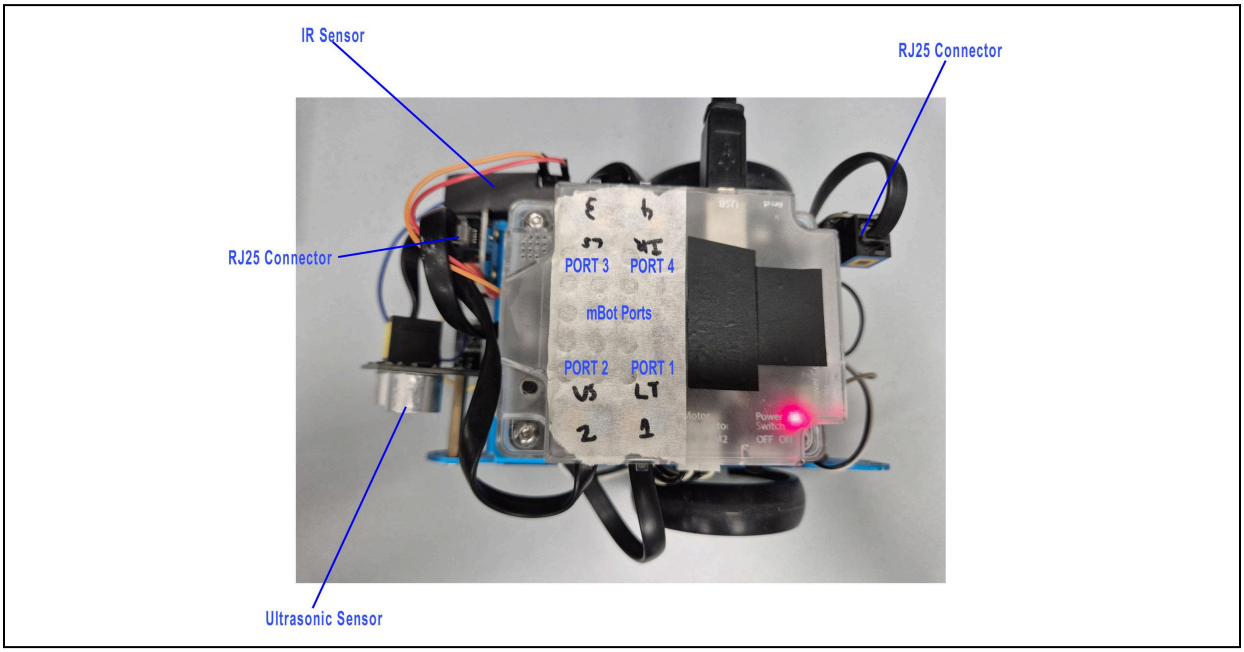


The Finished Product.

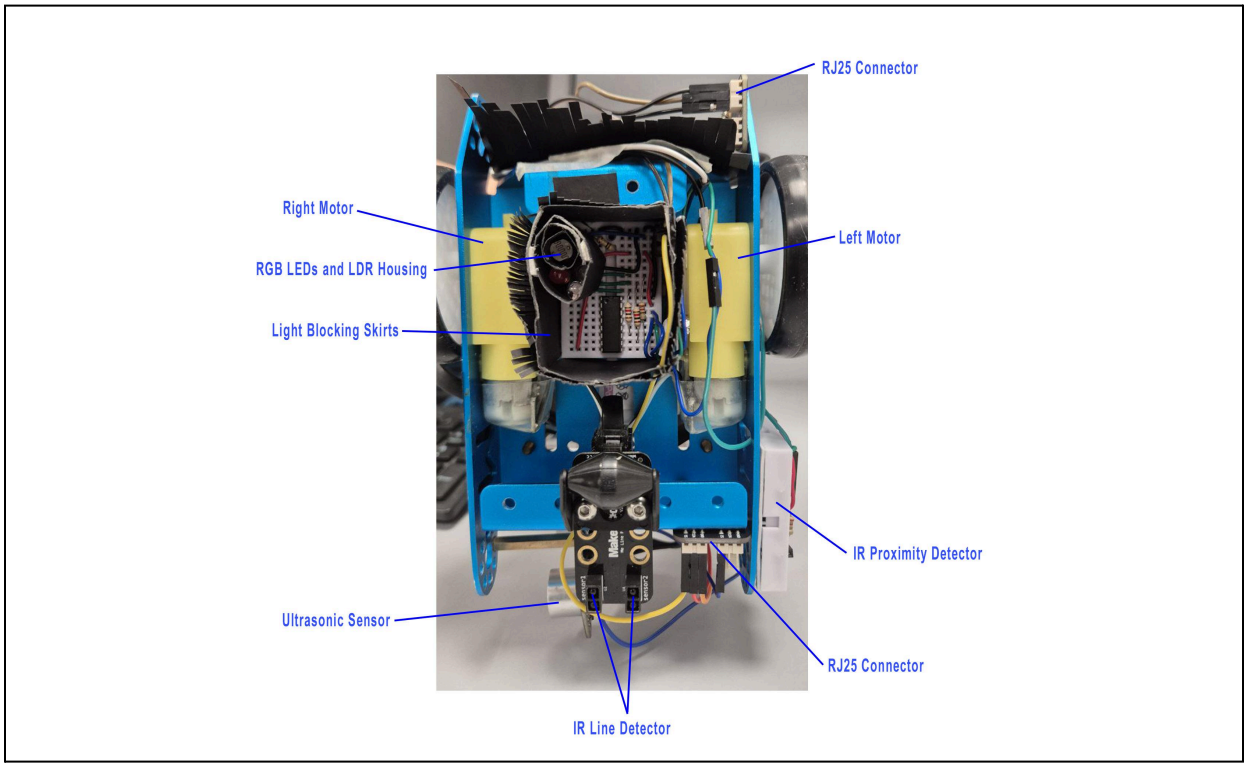


Neat Wiring to Improve Circuit Readability.

1.1 Top and Bottom View

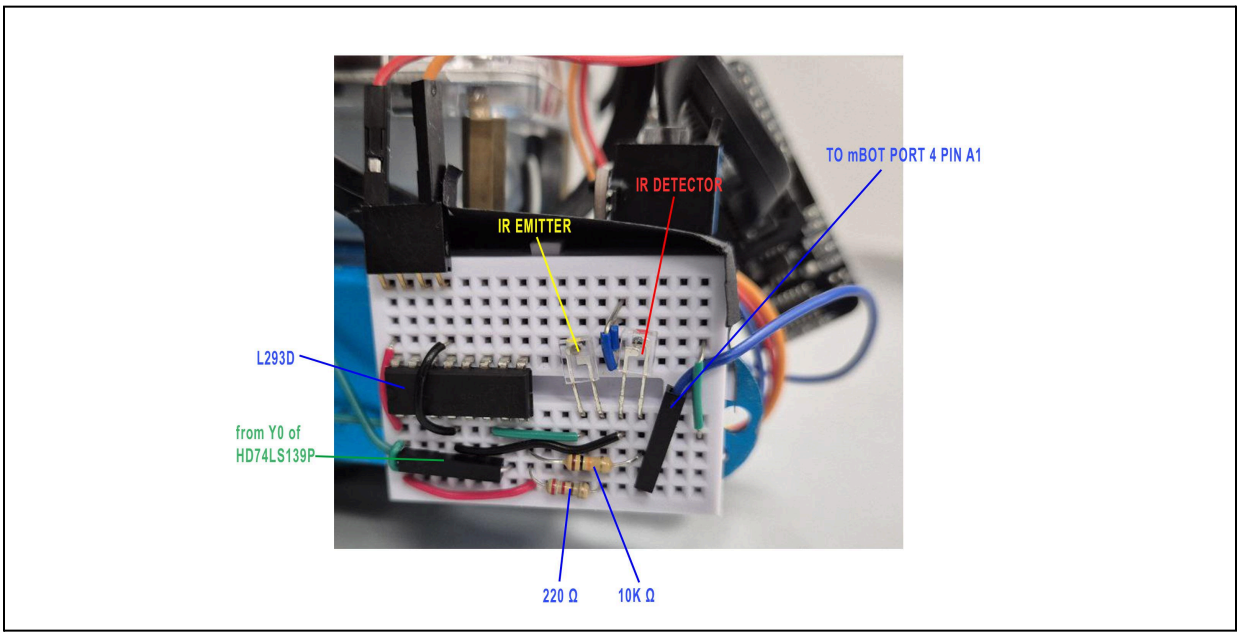


Top View

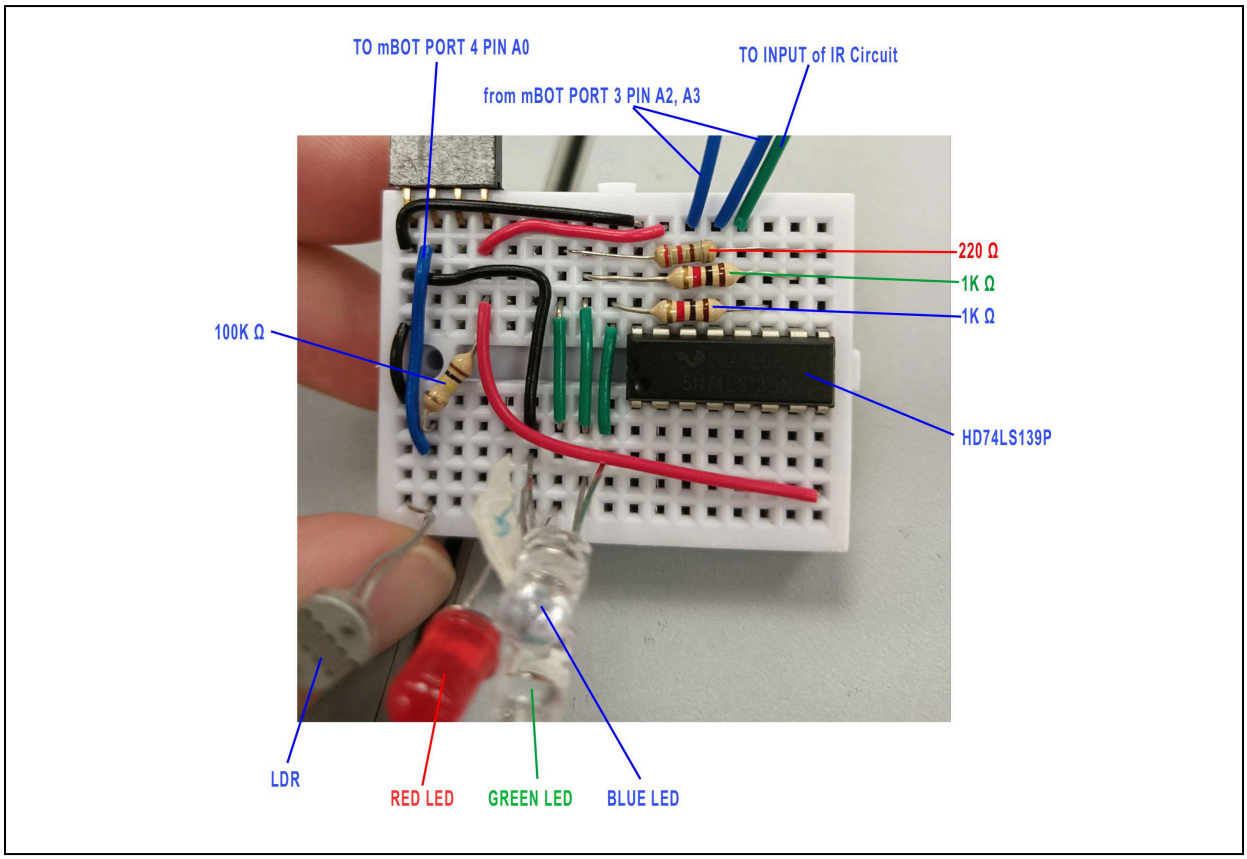


Bottom View

1.2 Sensor Breadboard Circuits

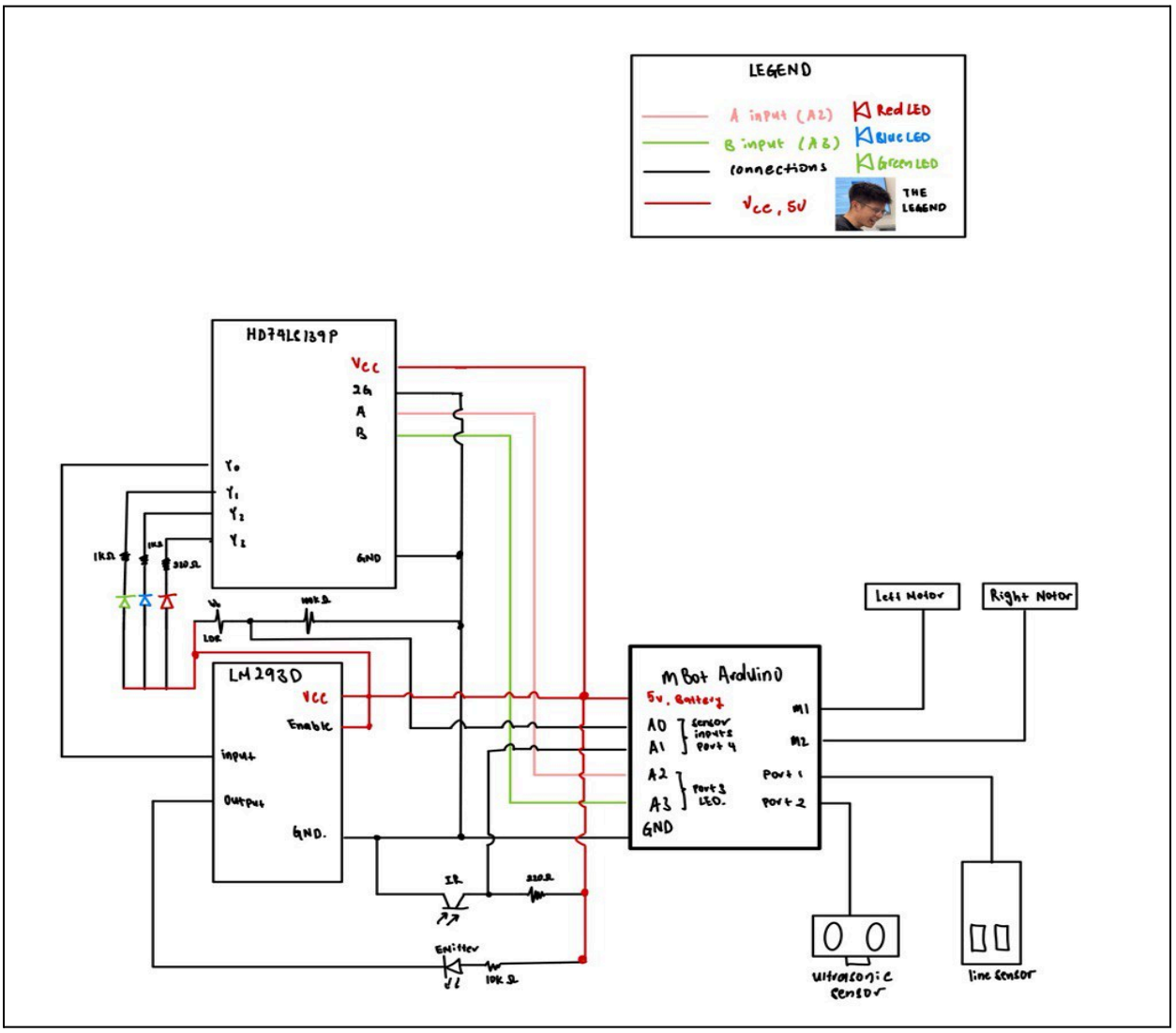


IR Proximity Sensor



Colour Sensor

1.3 Overall Circuit Diagram



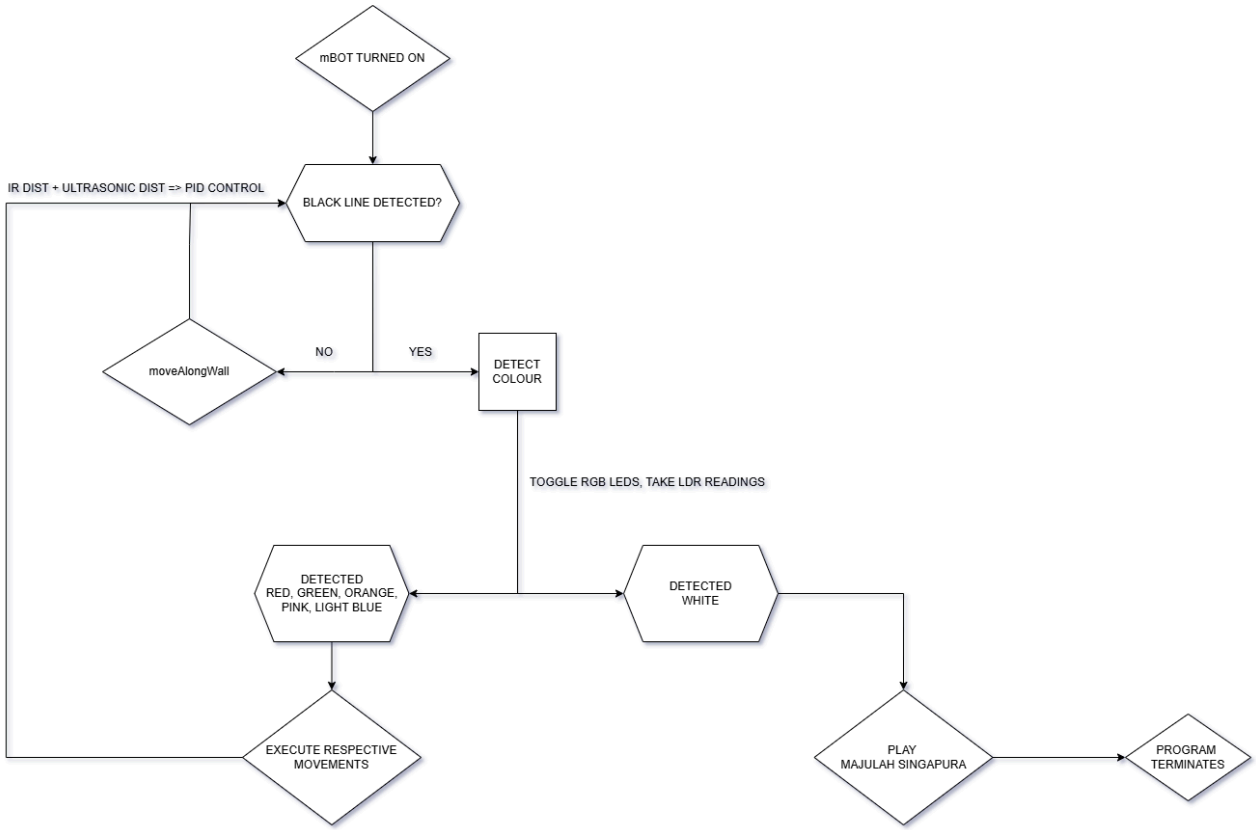
For the wiring schematic of our mBot robot, we decided to use Port 1 for the line sensor, Port 2 for the ultrasonic sensor, Port 3 for LED controls and Port 4 for sensor inputs (LDR and IR).

Inputs			Outputs			
Enable	Select					
G	B	A	Y_0	Y_1	Y_2	Y_3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

H ; high level, L ; low level, X ; irrelevant

According to the table of logic outputs of the HD74LS139P chip shown above, when the output of the chip is low, the LED will be turned on. This is why the LED's bias is towards the V_{cc} of 5V.

2. Description of Overall Algorithm



Upon startup, the mBot initialises its sensors and motors. It begins by using the line-following sensor to check for the presence of a black line. If no black line is detected, the robot transitions into wall-following mode. In this mode, it employs an ultrasonic and infrared (IR) sensor to measure its distance from the left and right wall respectively. The robot maintains a right-biassed position between the walls using a Proportional-Integral-Derivative (PID) control algorithm.

The target distances for centering are approximately 117mm from the left wall and 60mm from the right wall. The PID controller uses constants of K_p = 2.0, K_d = 0.4, and K_i = 0.05. If the left ultrasonic distance exceeds 165mm or the right IR sensor distance exceeds 130mm, indicating no nearby wall, the robot substitutes base distance values of 117mm for the left side and 60mm for the right side.

If the IR sensor detects that the robot is too close to the right wall, indicated by a reflected voltage dip below the threshold, it adjusts its trajectory to nudge left. The robot dynamically adjusts motor speeds within a range of 0-255, compensating for motor bias using a factor of 0.98 for the right motor. It continues moving forward while iteratively correcting its path until the line sensor detects a black line.

Upon detecting a black line, the robot halts, resets its integral error for PID control, and activates the RGB colour sensor. The robot reads red, green, and blue light intensities using an LDR circuit and calculates the closest match to predefined colours using a Euclidean distance algorithm with an initial error threshold of 20.

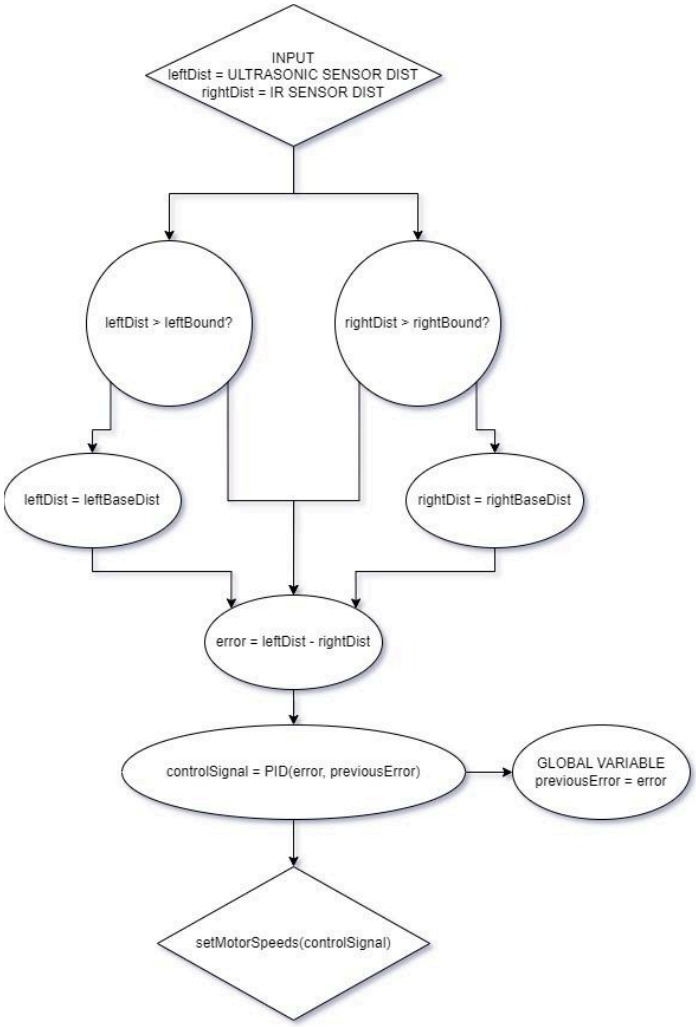
Below is a summary of the respective colours and its corresponding movement:

Respective Colour	*Corresponding Movement*
Red	Left turn.
Green	Right turn.
Orange	180° turn within the same grid.
Pink	Two successive left-turns in two grids.
Light Blue	Two successive right-turns in two grids.

Upon detecting WHITE, the robot stops and plays a celebratory tune consisting of predefined notes and durations of *'Majulah Singapura'*, signalling the end of the maze.

3. Specific Implementation Details

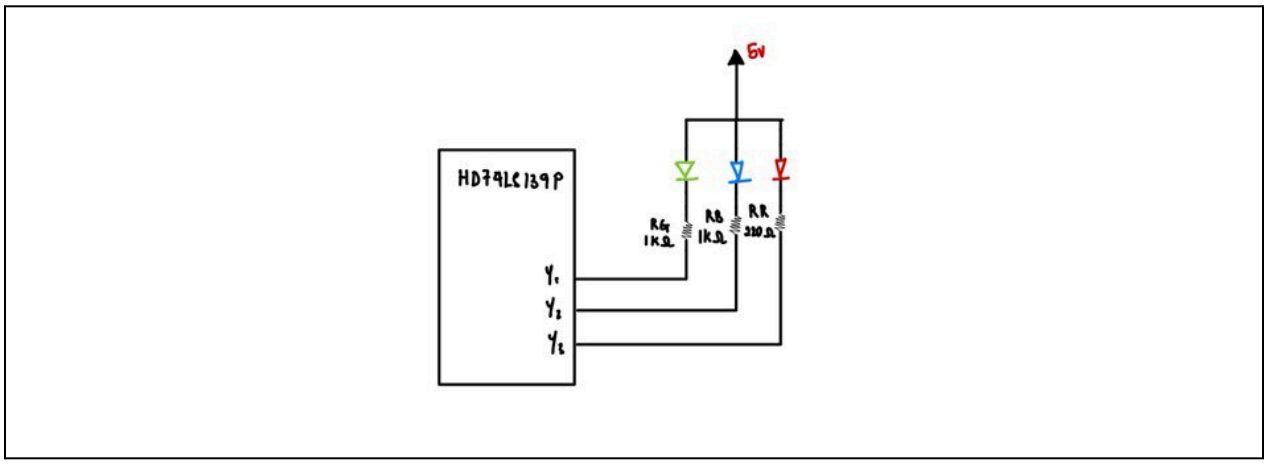
3.1 Wall Avoidance



While the mBot searches for a black line, it maintains a straight path by utilising a proportion, integral, derivative (PID) controller that takes in inputs from both the ultrasonic and IR sensors to determine its distance from the left and right wall respectively. However, if either or both of the left and right distances exceed a certain bound, we revert it back to the fixed base value.

This is to allow the robot to move straight even when there is no wall detected on either side. Additionally, these base values have been set such that the ideal position of the robot is closer towards the right wall, to give it sufficient space to rotate 180° (at the orange square) without hitting the wall.

3.2 Colour Sensing



LED Control using HD74LS139P.

The 3 LEDs are controlled using only 2 input pins (A2 and A3 of the J3 RJ25 port) through the use of a HD74LS139P 2-to-4 decoder IC chip. The circuitry is set up (as shown above) such that setting both pins to HIGH switches on the red LED, and setting only pin A2 or A3 to HIGH will switch on only the blue and green LED respectively.

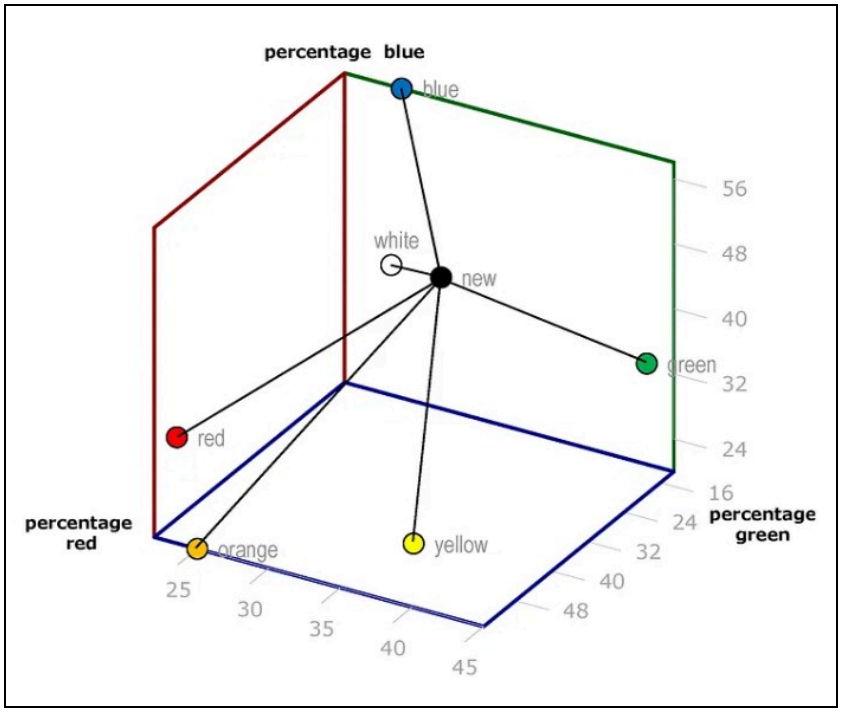
The LDR reading essentially gives us the intensity of the light reflected by the coloured paper. So, we must first establish a baseline intensity, such that we can normalise the LDR readings, which are given as values between 0 and 1023, to an 8-bit RGB value between 0 and 255. To achieve this, we first read the LDR value for red, green and blue light reflecting off a black surface one-by-one, then repeat the process using a white surface. The first 3 values will be stored in our blackArray, while the second 3 is stored in our whiteArray. Afterwards, we can find the differences between white and black for each of the 3 colours, which we call greyDiff. This gives us the following equation to normalise the LDR reading into an 8-bit RGB value, where blackArray and greyDiff vary for each of the 3 RGB colours:

```
int curr_reading = (analogRead(LDR) - blackArray[COLOR])/(greyDiff[COLOR])*255;
```

To calibrate our colour sensor, we utilised the above formula to determine the baseline red, green and blue values for each colour paper. We then set these calibrated values into our colour sensing function. When the robot is operating, it reads the red, green and blue values of the colour paper and calculates the delta in readings to each calibrated value. It then determines which colour gives the lowest delta value and performs the actions associated with each colour.

The Euclidean algorithm determines the closest match from a set of predefined colours by first applying a threshold filter, checking if the detected colour (rgbReading) is within a tolerance (errorshift) of each colour's RGB values.

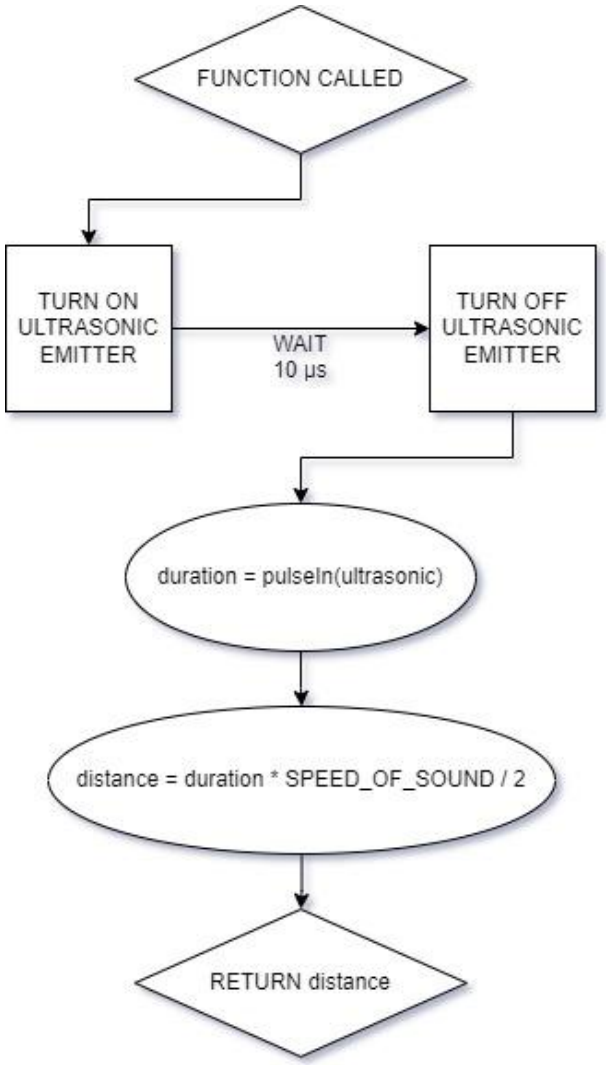
If only one colour matches, it is returned directly. If multiple colours remain, and the tolerance is relatively small (≤ 5), the Euclidean distance between rgbReading and each colour is calculated, with the closest match selected. If multiple colours still match, the function recursively reduces the tolerance to refine the match.



Vectorising colours with Red, Green and Blue values.

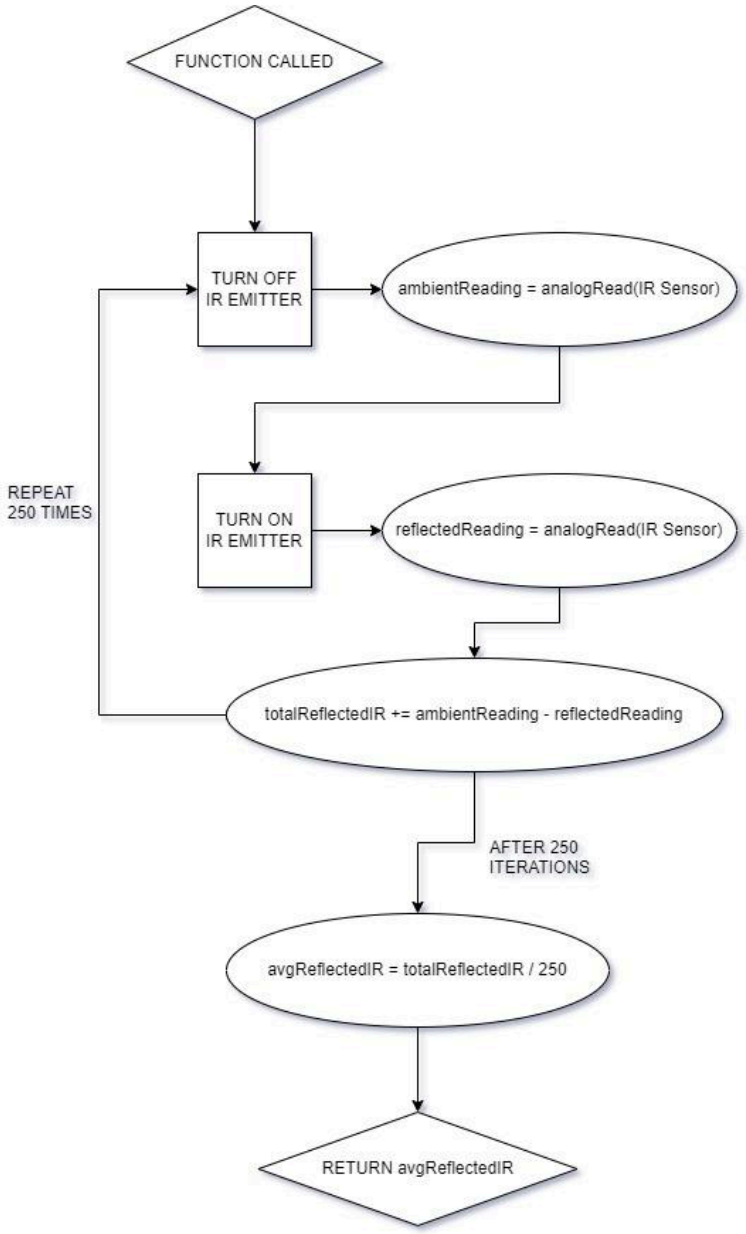
The Euclidean distance calculates the geometric proximity between colours in RGB space, by vectorising each colour using its Red, Green and Blue values, ensuring accurate matching even with slight variations. This combined approach of threshold filtering and distance calculation allows the algorithm to identify the closest colour efficiently.

3.3 Ultrasonic Sensing

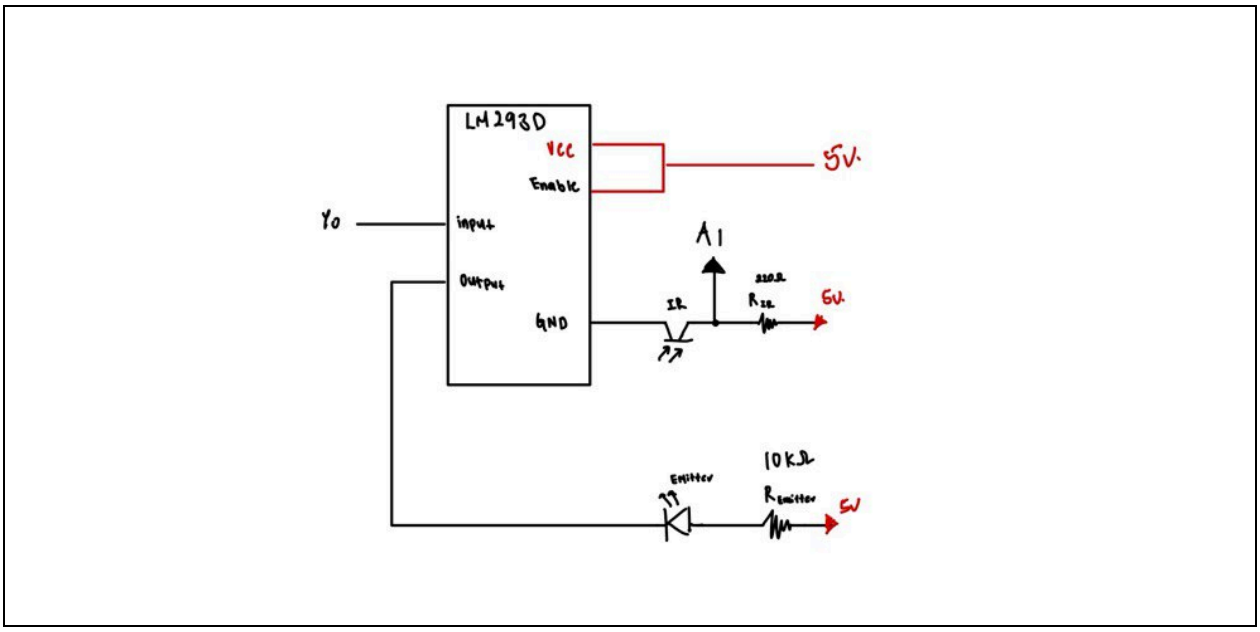


While our mBot moves through the maze, it continuously measures its distance from the left wall by using the ultrasonic sensor. The process of measuring the instantaneous distance is described in the flowchart above.

3.4 Infrared Proximity Sensing



While our mBot navigates the maze, it uses an IR Proximity Sensor to determine its distance from the right wall. Similar to the LEDs used for colour sensing, to toggle the IR emitter to turn on and off, we send an appropriate signal to the HD74LS139 2-4 decoder. When the IR Emitter is on, we use the IR sensor to sense the reflected IR light, when it is off, we detect the ambient IR light. Hence when we take the ambient reading and minus the reflected reading, we get the value of the reflected IR light, mitigating the effect of ambient lighting conditions. We repeat the above process 250 times and take an average reflected IR light value, which is summarised in the flowchart above.



IR Emitter and Detector using L293D.

Using the above process, we conducted a calibration experiment to find a correlation between distance from wall to IR sensor, and the avgReflectedIR detected, which will be detailed in “Calibration Process”.

3.5 Rotation

```
// Code for a 90 deg left, forward and 90 deg left.
void doubleLeftTurn() {
  rotateLeft();
  delay(100);
  moveForward(860);
  delay(100);
  rotateLeft();
}
```

Code block for the 2-grid rotation, using delays to alter the angle of rotation.

Based on the colour detected and classified from above, we created 6 different functions for each of the 6 colours. Within each function, we specify the necessary timings to complete the required actions for each colour challenge. Using this approach, it is convenient to individually tweak each movement to fine tune them up to standard. As seen in the code snippet above for pink colour challenge “two successive left turns in two grids”, we are able to tune the delays between movements and the duration in which the mBot moves for each of the Colour Challenges.

4. Calibration Process

4.1 Colour Sensor

We first collected sets of RGB LDR values for black and white paper. This finds the theoretical maximum and minimum values that our LDR should be outputting, allowing us to convert the LDR output from a scale of 0-1023 to the RGB scale of 0-255, thereby finding the exact RGB code for easier colour verification.

We then collected sets of sRGB LDR values for each coloured paper and converted the LDR values into an 8-bit RGB value. This gave us a baseline set of values to compare against when running the maze.

4.2 Infrared Sensor

Using the serial monitor, we measured the output of the IR sensor when a white obstacle is placed at regular intervals from it, and recorded the values in a table. We then plotted a graph of IR output against distance using excel, and found the equation of the best-fit line. This allows us to use the LDR output value to determine the IR sensor's distance from the wall.

Finally, we determined the distance between the wall and the IR sensor when the mBot is in its ideal position. This allows us to find the ideal distance between the IR sensor and the wall, giving our PID controller a target value.

4.3 PID Tuning

Our method of PID tuning utilised systematic trial-and-error to find the ideal K_P, K_i and K_D values for the purposes of this mission. This is essential as there are no predetermined ideal K_P, K_i and K_D values to be used, and they have to be experimentally determined according to our specific use case conditions, hence iterative testing was our approach.

When our mBot does not seem to be sufficiently responsive to the stimuli (decreasing ultrasonic/IR output), K_P-value has to be increased and vice versa. This is because K_P is a multiplier which determines the magnitude of the feedback action used to correct the error. When our mBot appears to be oscillating when attempting to achieve the ideal distance from the wall, K_i-value has to be decreased and vice versa. This is because K_i is an add-on term which increases as the duration and magnitude of error increases. This leads to our mBot overshooting the target value and creating an error in the opposite direction. When our mBot needs to be more responsive only immediately after a turn, i.e. an inaccurate 90° turn leading to a huge error in its position, K_D-value has to be increased.

Conversely, if the 90° turns are more reliable, K_D-value can be decreased to smoothen out the mBot's path. This is because K_D is an add-on term that increases according to the rate-of-change of the error. If it is instantaneous, it boosts the overall feedback action to quickly correct the error.

5. Improving Robustness of Algorithms

5.1 Colour Classifying Algorithm

Initially, we had utilised the method of boundary values for classifying colour, which uses predetermined RGB bounds for each of the coloured papers. However, we finally settled on the Euclidean method instead, which involves calculating the Euclidean distance between the detected RGB values, and that of the calibrated values for each coloured paper. This is due to the following 2 reasons:

1. We are only allowed to calibrate the mBot based on our 'home' mazes, which could have differing ambient light conditions and slightly different shades of coloured papers compared to the 'away' maze. Thus, the RGB red value of the red-coloured paper in the 'home' and 'away' maze might differ significantly. If the bounds method is utilised, the mBot has a high chance of failure; whereas if the Euclidean method is used, as long as the RGB values lie the closest to red, the mBot will not fail.

2. The Euclidean method ensures that the mBot will always make a decision, giving it a chance of making the right decision rather than remaining stationary in the maze.

Hence, our final colour classifying program consists of the 3 following steps:

Steps	*Justification*
1. When mBot detects a black line, it switches on its red, green and blue LED successively and takes 40 LDR readings each. 2. The algorithm discards the first 20 values and proceeds to take the average of the second set of 20 values. 3. The Euclidean algorithm has a maximum error of 20 for each RGB value.	1. Taking 40 readings each allows us to have a larger sample size, increasing the precision of our colour sensor. 2. From prior testing, we realised that the LDR has an inherent response time of about 20 readings, whereby it outputs values which are inaccurate or even possibly negative. This is a hardware issue which could not be avoided. Hence, we discarded the first 20 readings to increase the accuracy of our colour sensor. 3. From our testing, we determined that it is near impossible for different ambient lighting settings and different shades of colours to produce an error of more than 20. As such, the starting point of the Euclidean algorithm was determined to be 20, ensuring its efficiency.

5.2 Path Correction Algorithm

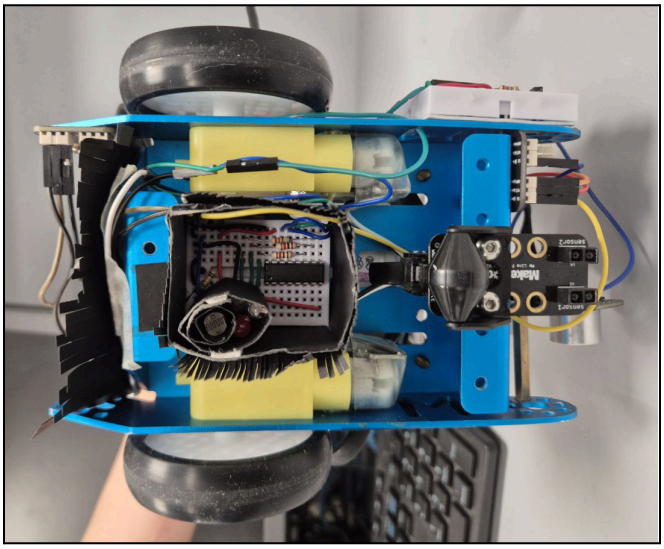
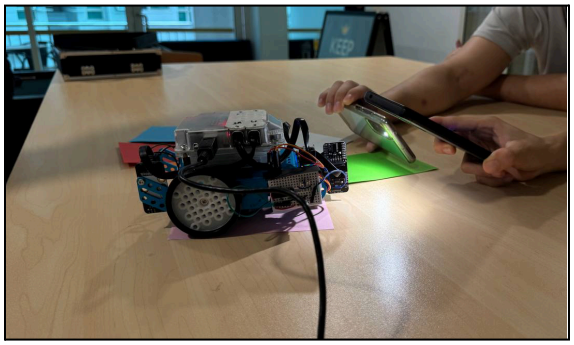
We considered 2 primary methods in path correction; namely, the nudging and PID method. While the nudging method uses predetermined bounds that elicit a preprogrammed response from the mBot, the PID method is a continuous process that constantly increases and decreases the speed of each geared motor depending on how far the mBot is from the ideal distance from each wall. We identified 4 main advantages that the PID method has over the nudging method, which ultimately led to us utilising it. They are as follows:

1. Utilising the PID allows the mBot to complete the maze in a shorter period of time. This is because it prevents zig-zagging during its travel, decreasing the overall distance travelled by the mBot and hence reducing overall travel time.

2. For the orange coloured challenge, our mBot requires sufficient clearance on the left to execute its 180° turn. As such, using PID allowed us to set the ideal position of the mBot slightly closer to the right wall, preventing it from colliding with the left wall while executing the u-turn. On the other hand, nudging does not guarantee the position of the mBot when it stops at the black line, increasing the chance of collisions.

3. PID corrects the direction of movement of the mBot. This is especially important for the colour challenges, where the robot ideally approaches the coloured paper perpendicular to the black line, and leaves the challenge without collision. PID enables the mBot to approach the challenge at the correct angle, while also correcting the angle when leaving, should there be inaccuracies in the hard coded movements.

6. Challenges Faced

Challenge Faced	***How We Overcame Them***
Unreliable Colour Sensing - Ambient light	During our calibration, our values deviated heavily from the pre-calibrated values between multiple runs. We isolated the issue and found that our sensor was heavily sensitive to ambient light. To solve this, we lined our colour sensor with black paper skirts to block out any ambient light that could enter the chamber to affect the readings. The skirts were further refined with a 'sweeper' like design to minimise friction during movement.  *The side skirts of the Color Sensor chamber to minimise ambient light.* We also tested its resistance to ambient light by varying it during testing and checking its delta to the pre-calibrated values, tweaking the delay between the red, green and blue LEDs to improve reading accuracy.  *We altered the ambient light by flashing light at different angles.* This approach ensured a good balance of light blocking, improving the

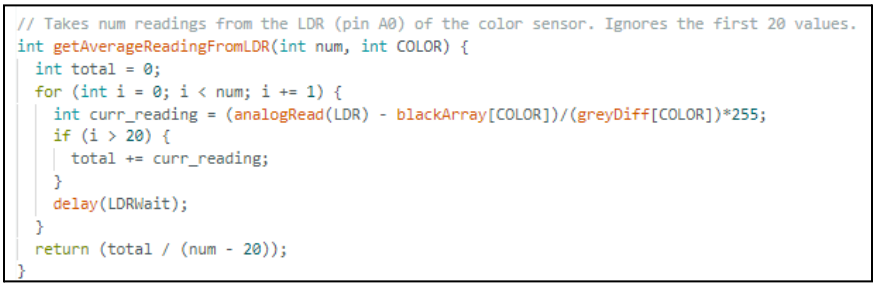
<table>
<tr><td></td><td>accuracy of readings taken, whilst reducing any impedance in its movement.</td></tr>
<tr><td>Unreliable Colour Sensing
- Insufficiently robust algorithm</td><td>The initial approach taken was to use a boundary checking algorithm, using numerous if-else statements to check if the current value falls within the range of our pre-calibrated readings.

However, this made our effective range of detection static, resulting in many instances of incorrect colour classification.

As such, we refactored and moved to an Euclidean approach, using a combination of threshold filtering and Euclidean distance calculation, which made this accepted range mutable. As a result, it significantly improved our ability to distinguish between similar colours, such as red, orange, and pink. (functionality explained above).

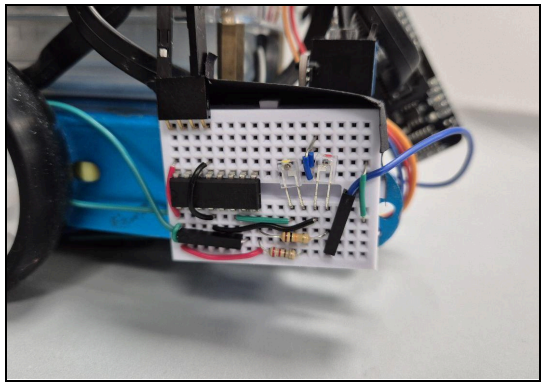
This approach also increased the efficiency of calibration, as we only required 1 set of RGB channel colours, compared to 2 sets (minimum and maximum for the bounds) for the previous iteration.</td></tr>
<tr><td>Unreliable Colour Sensing
- Dirty data in LDR output</td><td>When taking colour readings from the LDR, we took an average to increase overall precision.

However, we noticed that the first 20 readings taken from the LDR were heavily inaccurate, due to the LDR’s response time. As such, we refactored the function that took the colour readings, removing the first 20 values from the average calculated in software.

The refactored LDR average read function.

This improved the overall accuracy of the readings, improving colour identification and classification.</td></tr>
<tr><td>Limited IR Working Range</td><td>During our testing, we identified that our IR sensor’s effective range was too limited. This led to the robot not having enough time and distance to correct itself when it was too close to the wall.</td></tr>
</table>

	 *Our IR sensor, placed at the front of the robot.* To address this issue, we isolated various components and experimented with different resistors to adjust our calibration equation in software. We then identified that the IR receiver was faulty and replaced this component. This change, along with tweaking the resistor values, enhanced the response time and extended its effective range.
Proportional–Integral–Derivative (PID) Controller Tuning	Initially, our robot would move erratically, nudging left or right whenever it detected that it was too close to a wall. To address this issue, we decided to implement a PID controller to smooth out its responses. When tuning the PID, the main challenge we faced was the sensitivity of the turning for the robot. We needed to carefully adjust the values of K_p (the proportional gain) and K_d (the derivative gain) to minimise any oscillation around its path. This required an iterative tuning process, where we isolated each variable and conducted multiple test runs to identify the optimised values.  *Our iterative testing platform for wall avoidance.* With the PID controller enabled, the robot was able to effectively realign itself to the centre of the maze, even after exiting a colour challenge at an angle.

7. Work Division

Member	***Overall Responsibilities***
Member 1: Tirodkar Om Milind	Software - Overall movement (rotation, etc). - Colour sensing and response.
Member 2: Toh Ee Sen, Izen	Software - IR circuitry and design. - PID for movement.
Member 3: Thia Yang Han	Hardware - Color Sensor circuitry and design. Software - PID for movement.
Member 4: Tjhin Brian	Hardware - Color Sensor circuitry and design. - Color Sensor light-shielding measures.

Our Thoughts

Om:
This project was truly enriching, it was rewarding to put both the software and hardware knowledge we had gained over the past semester to build something with real-world applications.

Izen:
A very fun and educational introduction to the world of engineering. Was highly satisfying to see the integration of our hardware knowledge from the studios into a robot with ‘real’ applications.

Yang Han:
This project was an educational and interactive first step into the world of Computer Engineering. While it was challenging, I had fun interacting with my lab and group mates, helping each other throughout the course.

Brian:
This hands-on project was an eye-opening foray into the world of Computer Engineering, and it has certainly deepened my understanding of integrating hardware components with software development to solve real-world problems.

8. Conclusion

This project provided our entire team with an invaluable learning experience that went far beyond just mastering engineering fundamentals. We also honed essential skills like collaboration and iterative development, giving us our first taste of owning an engineering project from start to end.

It was a daunting task in the beginning, with many technical requirements that we had to implement, especially in a short time frame of just a few weeks. We quickly realised that leveraging our strengths was crucial in optimising workflows and ensuring that we stuck to our timeline. It was truly fulfilling to interface with both hardware and software components to tackle obstacles.

This project also taught us a key lesson; setbacks often occur when you least expect them.

While such setbacks may seem inexpensive to resolve in the bounds of a classroom, they become significantly costlier in real-world applications. Our key takeaway is that no plan is bulletproof—especially so in engineering projects. It is all about isolating the issue and adapting the approach when faced with unexpected challenges.

We also want to express our gratitude to the teaching staff and lab technicians for the wealth of knowledge and guidance they have provided us throughout the course.

In conclusion, this project encapsulated the complexities of real-world engineering challenges. It highlighted the importance of teamwork, applied our theoretical knowledge to practical situations, and taught us how to respond to setbacks. This experience has significantly enhanced our skills and better prepared us for our future academic and professional journeys.