CG1111A Report

Abstract. We describe the design and implementation of an mbot-based robot car that is capable of manoeuvring through a maze and discerning various colours. We discuss our initial implementations, calibrations; setbacks and how we overcame them.

AY 2023/24, Semester 1

CG1111A B01 Section 2 Team 3

Chan Wen Yee
Chan Xu Ming Ethan
Chan Zun Mun Terence
Chang Jia Jun

Table of Contents

Section 1. Algorithm

Below is the general logic of our robot. Figure 2 shows the logic of the function **colorAction**.

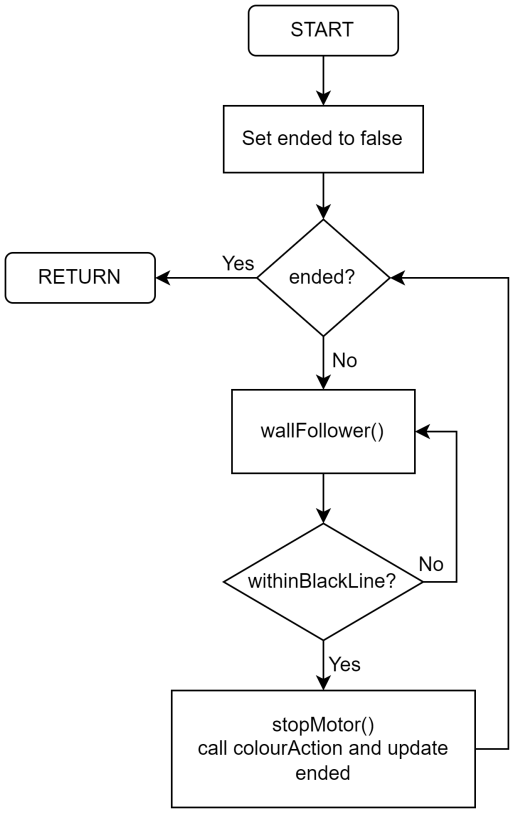


Figure 1

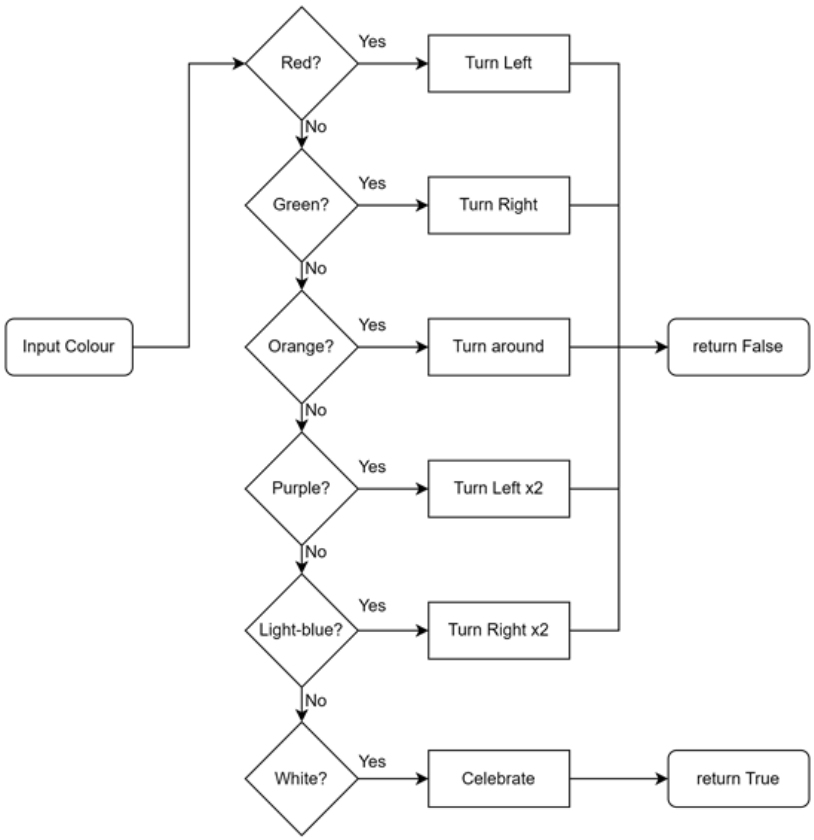


Figure 2

The logic follows a loop which continuously triggers as long as the end bool statement is set to false. Within the loop, the robot follows the wall continuously until a black line is found. This triggers the colour-sensing logic in Figure 2 which runs the appropriate action before returning to the main logic in Figure 1. If the robot detects white, it exits the loop and stops moving.

Section 2. Implementation of Various Subsystems.

2.1 Electrical Connections

The mCore's connectors/ ports were connected as such

Port	Device
M1	Left Motor
M2	Right Motor
PORT_1	Line Follower Sensor
PORT_2	Ultrasonic Sensor
PORT_3	LDR and IR Emitter (Breadboard)
PORT_4	Decoder (Breadboard)

Figure 3: Usage of the mCore's Ports

We used a decoder to increase the number of output pins, such that we can control 4 LEDs (Red, Green, Blue and IR). We chose resistor values for the LEDs such that the current flowing through the components would not exceed the maximum current rating of 8mA of the decoder (or the max current rating of 50mA IR LED as suggested to us). We wanted to supply our IR emitter with more than 8mA of current, such that the IR proximity sensor has a larger sensitivity range. Hence, we used a motor driver to supply more current to the IR emitter.

The full schematic can be found in the appendix.

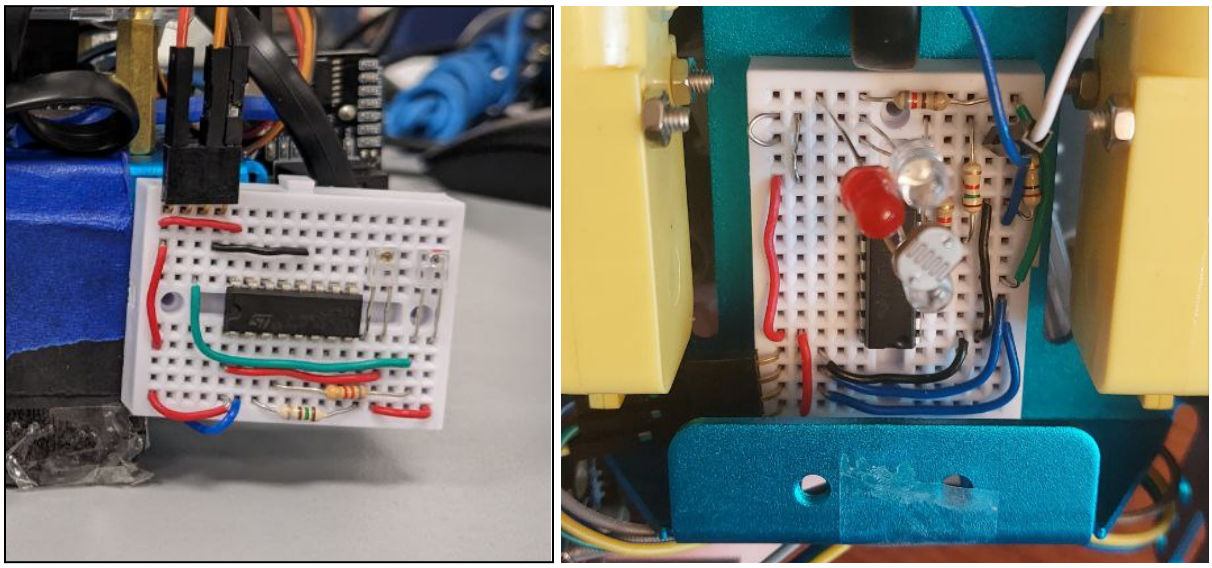


Figure 4: Breadboard circuits

2.2 Keeping the mBot Straight

Our robot does proportional wall following. In our code, we fix a distance away from the wall. This distance would be the distance from the wall to a centre line parallel to it. Our robot would aim to move straight in the centre line.

The ultrasonic sensor measures the distance between the robot and the wall. It then calculates the deviation of our robot from the centre line. We designed the adjustment to follow a non-linear reciprocal curve $s = \frac{1}{0.1x + 1}$, where s is the multiplier of the speed of one side of the motor, and x is the distance from the centre line. This allows the mBot to correct more extremely the further away it is from the centre line without compromising on its smoothness. If the ultrasonic does not detect any wall, and the IR sensor detects the robot is too close to the wall on the respective side of the robot, the robot will nudge away from the wall. If the ultrasonic sensor does not detect any wall and no detection of a wall by the IR sensor, the robot will keep moving straight.

This allows our robot to follow the wall with smoother motions, as well as take into account cases with missing walls. The robot would hence be able to move straight.

2.3 Colour Sensing

To measure the colour values, the robot turns on the respective Red, Green and Blue LEDs. On turning on each coloured LED, the robot measures the LDR voltage, to provide 3 integer readings. These are then processed to provide RGB values from 0 to 255 for ease of debugging.

We implemented a modified version of the KNN machine-learning algorithm. After collating a small dataset using our colour sensor, we took the following steps:

1. Determine the Euclidean distance from the measured colour to each data point
2. Determine the average deviation for each colour
3. The colour with the smallest average distance was chosen

This has a few advantages:

1. Matching colours based on hardcoded ranges would require us to tweak these colour ranges to work best in every maze environment. Environments may vary drastically, causing it to be very tedious and time-consuming to find the perfect colour ranges. In contrast, KNN achieves a higher accuracy with fewer parameters that need to be adjusted and tweaked, the only data needed are sample colour points which are easier to collect and require less tweaking.
2. We can increase accuracy with ease simply by adding more data points
3. As there is no need for sorting, the time complexity was reduced from O(n^2) to O(n). This improves general performance. Overfitting was not a big issue due to our relatively small dataset.

2.4 IR Proximity Sensing

We implemented a boolean function designed to estimate the proximity of the robot to a nearby wall based on the readings from the IR emitter and detector. The function will return true if the estimated distance between the body of the robot and the wall is less than 4 cm and false otherwise. We turn on and off the IR emitter at regular intervals so that the algorithm adapts to the change in ambient IR.

The algorithm operates as follows:

1. Turn off the IR emitter.
2. Read the ambient IR value, which is the baseline value.
3. Turn on the IR detector.
4. Read the total value of the ambient IR and the reflected IR.
5. Calculate the difference between the readings from step 2 and 4. The value calculated is the voltage drop of the IR detector's voltage that is represented by the analog output of Arduino Uno.
6. Check if the baseline value is within the predefined thresholds. If a suitable range is not found, the function returns false. Otherwise, it proceeds to the next step.
7. Check if the voltage drop falls within the specified range. If it does, the function returns true; otherwise, it returns false.

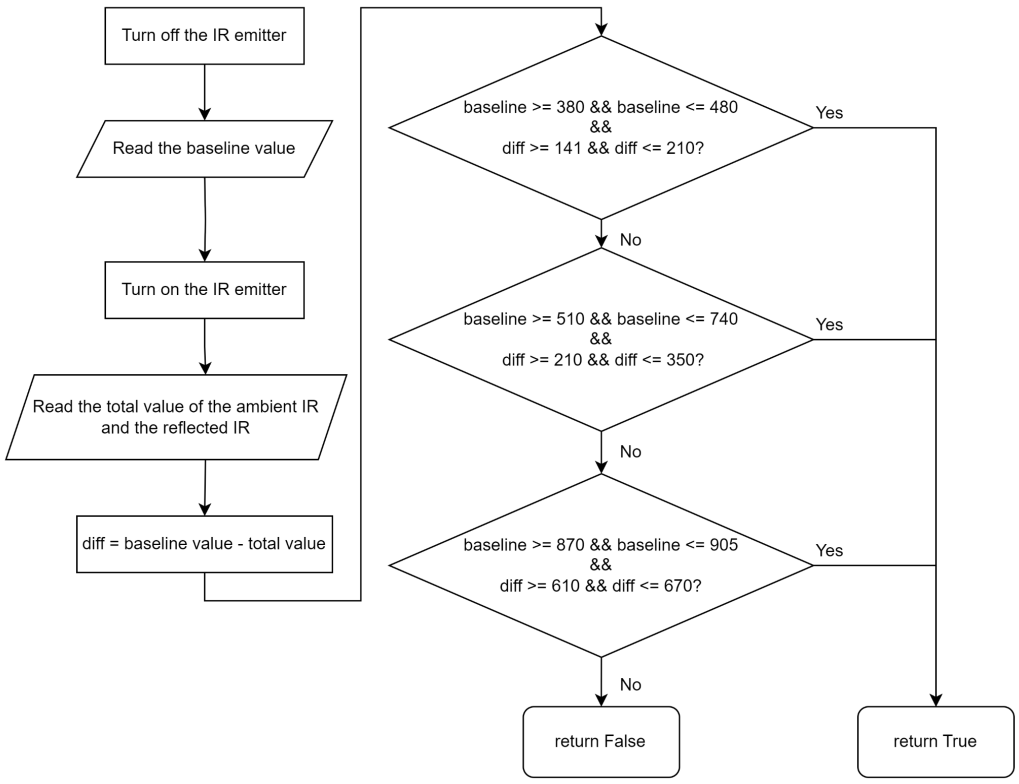


Figure 5: Logic of the function isNearIR

Section 3. Calibration/ Robustness

3.1 General Neatness of the Robot

Tape was used to secure the wiring and hanging components. Boards such as the adaptor and the Ultrasonic Sensor were tightly fastened onto the robot using screws and bolts.

Solid core wires were cut to a minimal length for use on the breadboard. Such is apt for permanent and short connections. Between the breadboard and adaptor, flexible jumper wires and female header pins were used. Their tips were bent as our team saw fit.

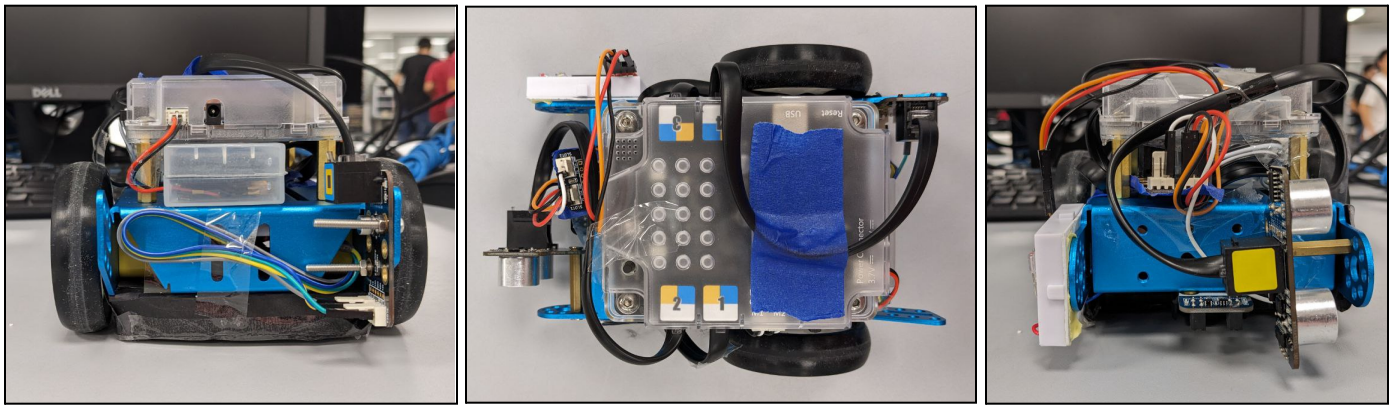


Figure 6: Back, Top and Front view of the robot

3.2 Tackling Ambient Light Variations

To reduce error due to light entering and escaping the system, a chimney and a layer of skirting were implemented. Multiple layers of paper were used to enhance the skirting. The guideline was that as little light as possible should be observed from the outside.

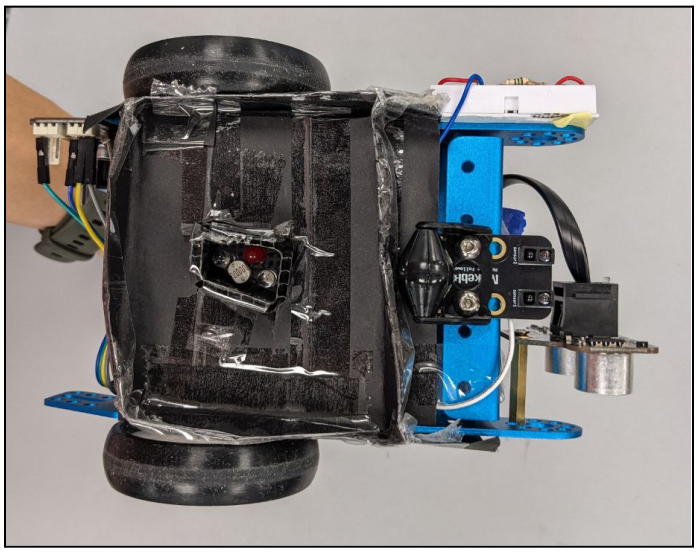


Figure 7: Skirting and Chimney of mBot

3.3 Improving Colour Sensor Reliability

We wanted to understand the characteristics of our LDR and adjust our colour sensing and detection algorithm to be more consistent and reliable. As such, we connected the LDR to the oscilloscope (Analog out from LDR to Oscilloscope). We observed that the readings (in orange) took 0.5s to stabilise.

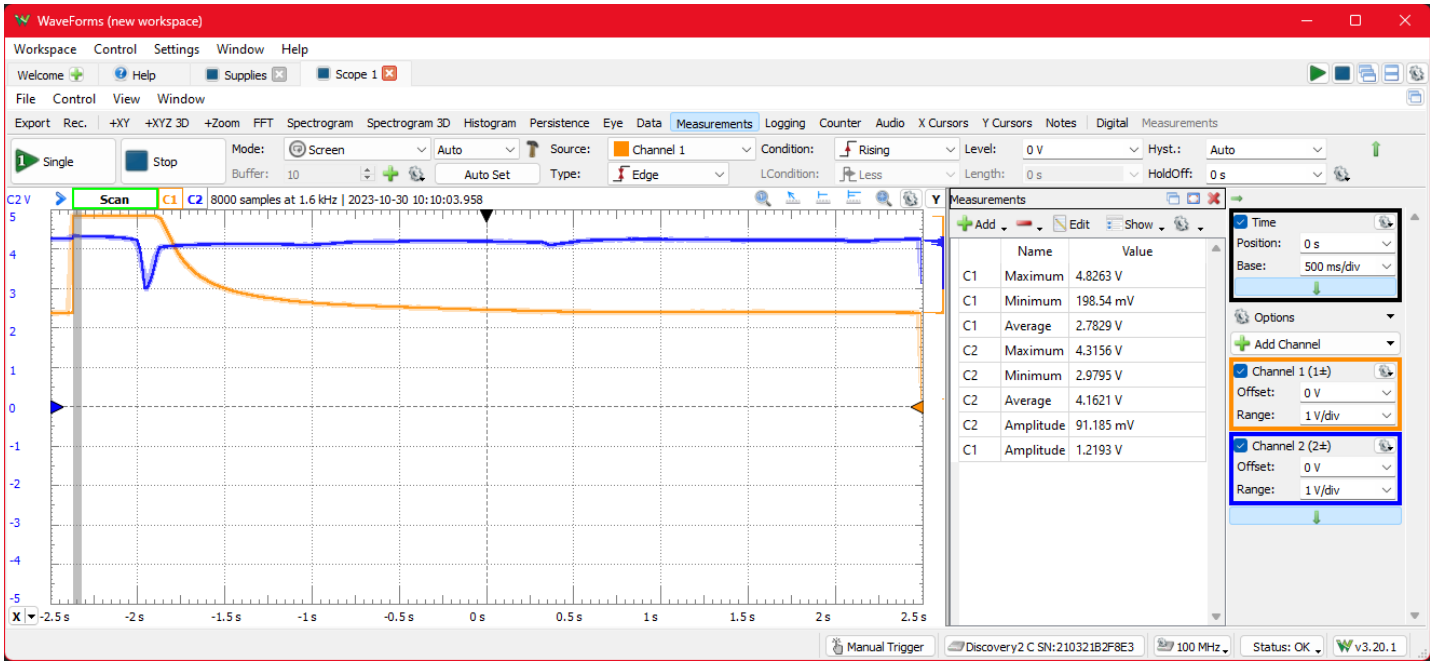


Figure 8: WaveForms Graph

We delayed the colour reading accordingly in our code. This would ideally reduce variation in the LDR readings and result in a more accurate colour sensor. From our testing and results, this led to a more reliable colour sensor with less variation in our readings.

We calibrated our colour sensor by measuring the maximum and minimum LDR readings when each coloured LED was turned on (by placing the robot on a black and white piece of paper). All other colour measurements were scaled to be within the range of 0 to 255. This allows colours to be easily debugged, as we can check what colour the robot is seeing by keying the processed values into a colour picker website.

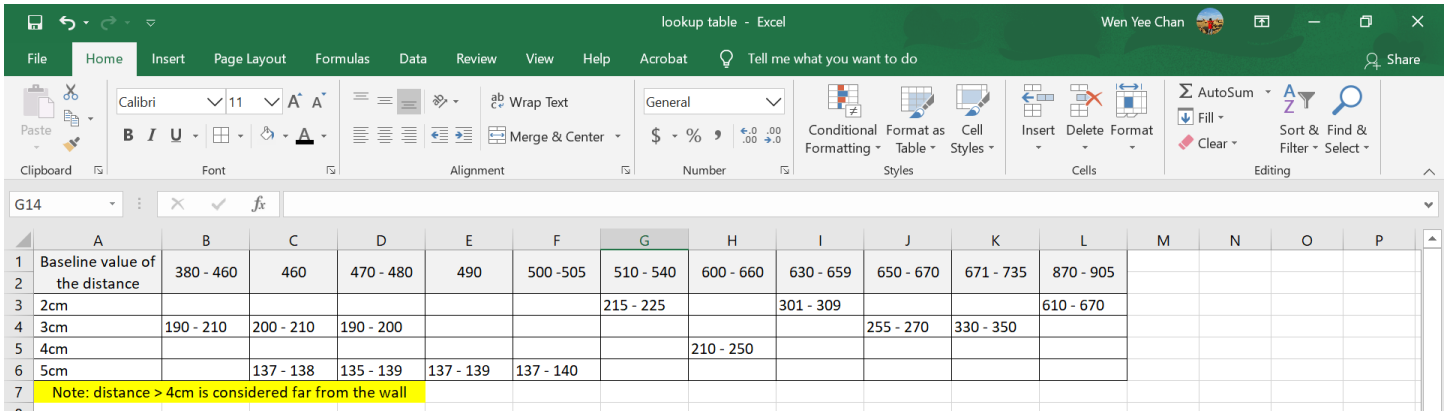
We took measurements of the colours on various mazes with our colour sensor. Furthermore, we augmented the dataset by adding minor variations (eg. +-10 to 1 of the RGB values). This provides a good variety of readings to allow for accurate colour detection on the various mazes and lighting conditions.

Our robot managed to pass all colour measurement tests in the mazes with no misdetection.

3.4 IR Proximity Sensor

To estimate the distance, during the calibration process, we did multiple measurements to determine the baseline value and its corresponding value (i.e. the voltage drop between the baseline value and the total value of the ambient IR and the reflected IR), for distances varying from 2 cm to 5 cm between the wall and the body of the robot.

As a result, a lookup table is created by using the data collected from these measurements. This table, which contains the baseline value and its corresponding voltage drop, serves as a key reference in the IR proximity-sensor algorithm. By observing the table, we established the thresholds of the baseline ambient IR and its voltage drop when the distance is less than 4 cm.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Baseline value of the distance	380 - 460	460	470 - 480	490	500 -505	510 - 540	600 - 660	630 - 659	650 - 670	671 - 735	870 - 905				
2	2cm						215 - 225		301 - 309			610 - 670				
3	3cm	190 - 210	200 - 210	190 - 200						255 - 270	330 - 350					
4	4cm							210 - 250								
5	5cm		137 - 138	135 - 139	137 - 139	137 - 140										
6	Note: distance > 4cm is considered far from the wall															

Figure 9: Lookup Table of the IR Proximity Sensor

Section 4. Difficulties and Solutions

4.1 Imprecise Turns

We spent a significant amount of time fine-tuning our turns. We manually adjusted our turning durations such that they were ideal. Furthermore, we also used our wall-following algorithm to correct any slight imprecision the robot might have made when turning. For example, if the robot overturns, the ultrasonic sensor will detect that the wall is too close or too far, causing the robot to adjust itself until it is straight.

We also faced an issue with the robot slipping on some of the obstacle courses and underturning. We figured that this could be due to dust on the wheels. Eventually, we settled on regularly cleaning the wheels, but also adjusting our turns with the dust taken into account. This would allow our turns to be more repeatable, working even when there is a moderate amount of dust instead of only working when the wheels are fully cleaned.

We left a tiny gap between our skirting and the floor the robot was on. This prevented the skirting from brushing against the floor and increasing friction. That friction slowed down our robot and affected the accuracy of our turns during our test runs.

4.2 Unreliable Ultrasonic Detections

We decided to use a function from the MakeBlockDrive library, `ultraSensor.distanceCm()`, to get readings from the ultrasonic sensor, as it is most likely refined and well tested by the official manufacturer of these components. This library helped us achieve more accurate ultrasonic readings without spending too much time on debugging.

There was an edge case, which was when the robot was near the edge of the table, and there was no wall for the ultrasonic to detect there. In this case, our ultrasonic sensor would have unreliable readings, where it might detect no wall at all, or it would detect the edge of the table.

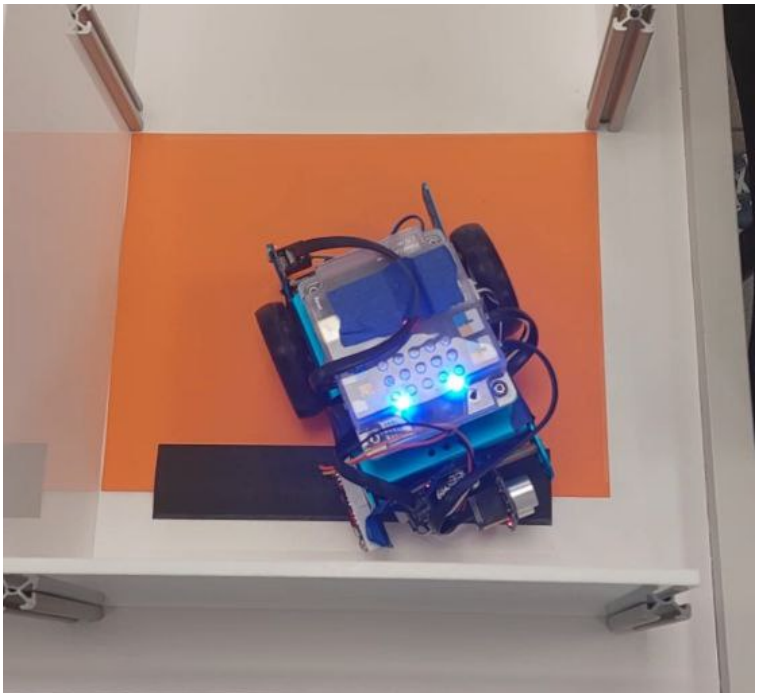


Figure 10: Picture of the Edge Case

To solve this, we mounted the ultrasonic sensor lower, closer to the maze floor. The ultrasonic sensor would hence have a more consistent reading. At this edge case, it would always measure the edge of the table. We can then handle this in code more reliably, for the robot to move straight with reference to the edge of the table.

4.3 Slow Reaction Time of the Robot

Occasionally at the colour challenges, our robot did not stop fast enough once it detected the black tape. It would end up hitting the wall in front of it. To solve this, we used C++ compiler directives in our code to enable and disable the Serial Logging and Debugging functionality in our code. This would remove unnecessary function calls in our code, ensuring a faster loop for our wall following and line detecting such that the robot can stop faster when detecting the black tape.

Compiler directives would allow us to easily toggle our Serial Functionality by changing just 1 line of code, helping smoothen debugging while still allowing us to get our robot in a usable state quickly. We would remove the need to manage and comment out several Serial Logging function calls.

Section 5. Division of labour

Person	General Tasks
Chan Wen Yee	IR Proximity Sensing
Chan Xu Ming Ethan	Ultrasonic Sensing and Movement Code
Chan Zun Mun Terence	Colour Sensing
Chang Jia Jun	Line Sensor and General Logic

Figure 11: Distribution of Labour Among Various Members

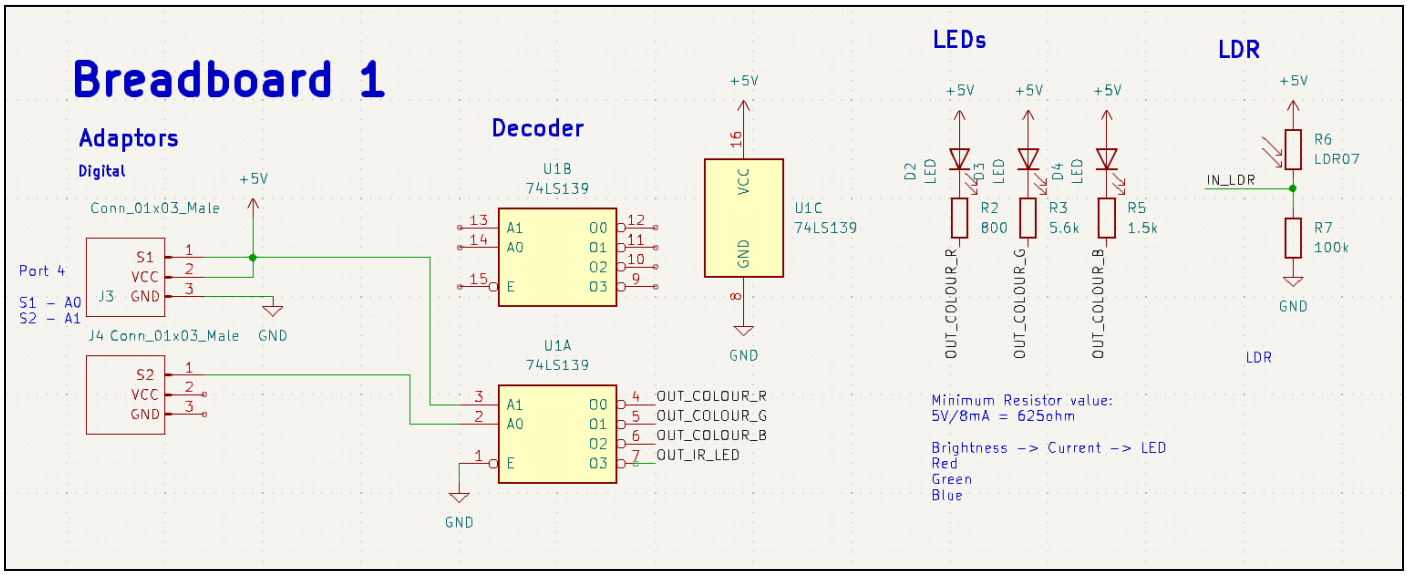
Firstly, the overall schematic of the robot was drafted based on the project requirements. The components were placed on the respective breadboard based on their proximity to the other components, and to minimise unnecessary wiring.

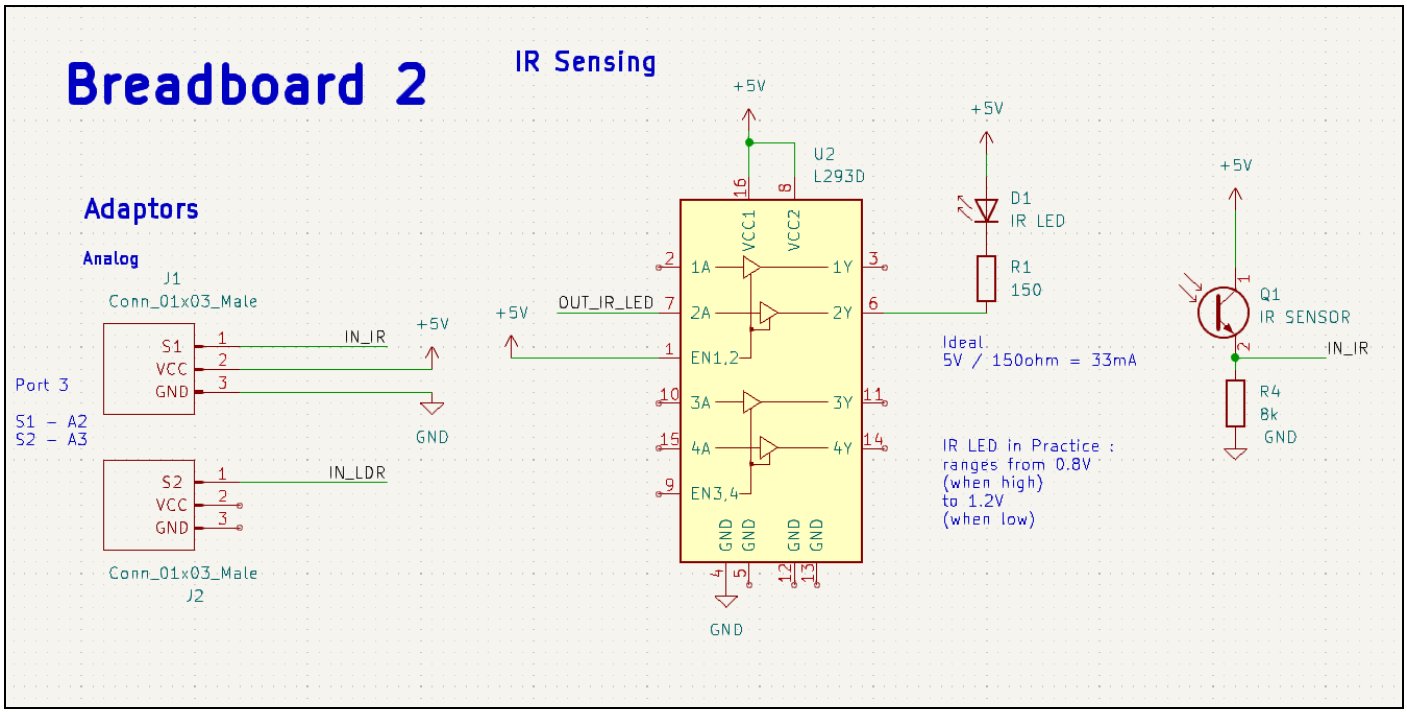
Each team member took charge of building a section of the schematic or a section of the robot (eg. Skirting of the robot, mounting of adaptors)

Collaboration of Code was done through Git. Each member implemented code on their various branches, which was then tested and merged into the main branch to be used for the final robot.

Appendix

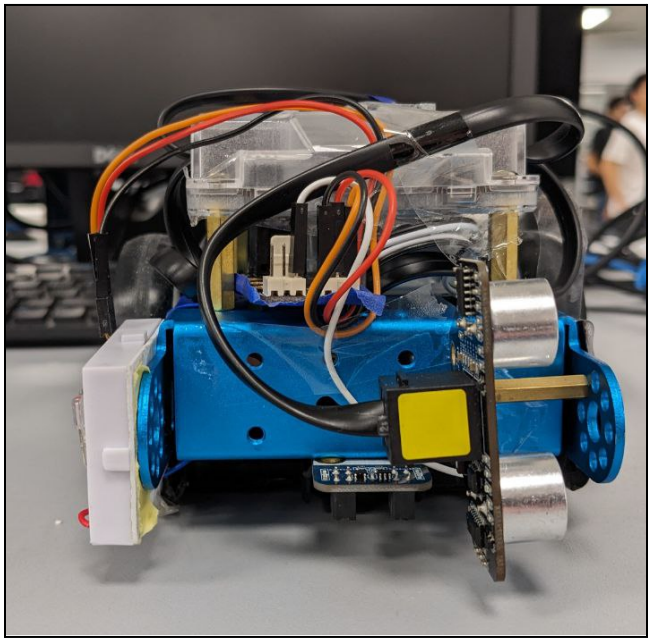
Overall Schematic of Breadboards



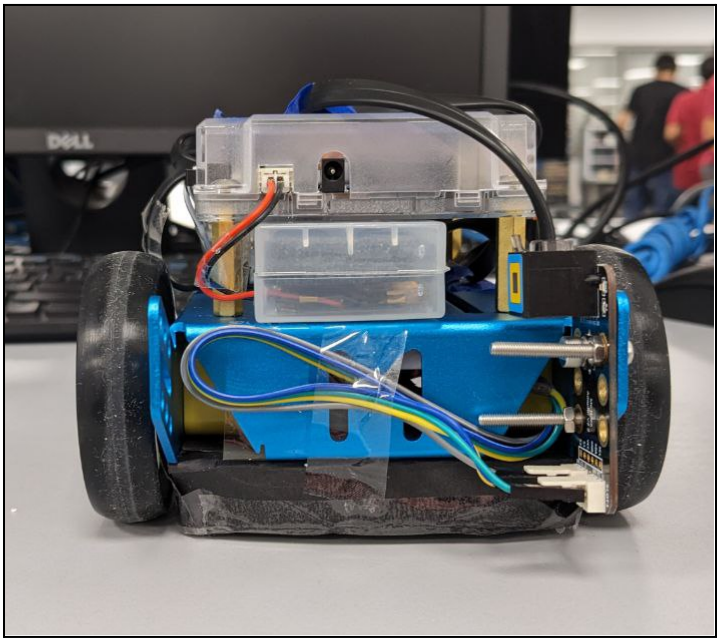


mBot and breadboard diagrams

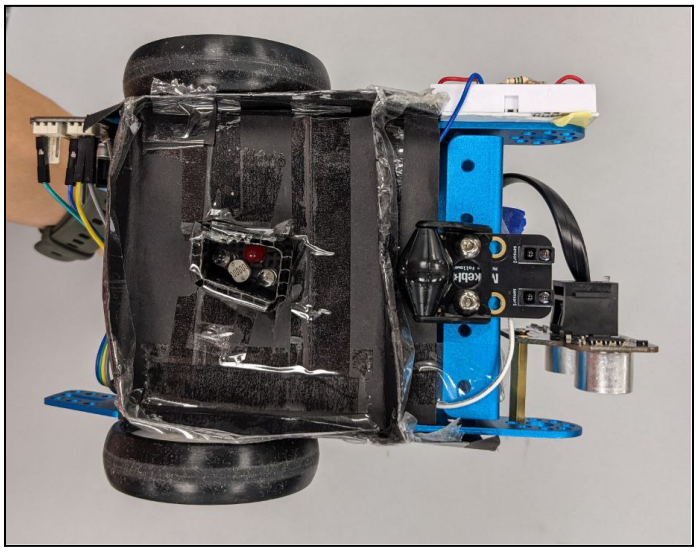
Front view of the mBot:



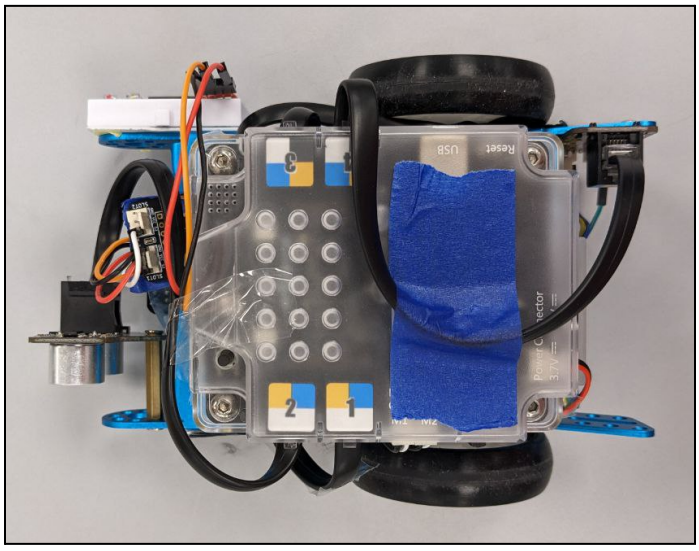
Back view of the mBot:



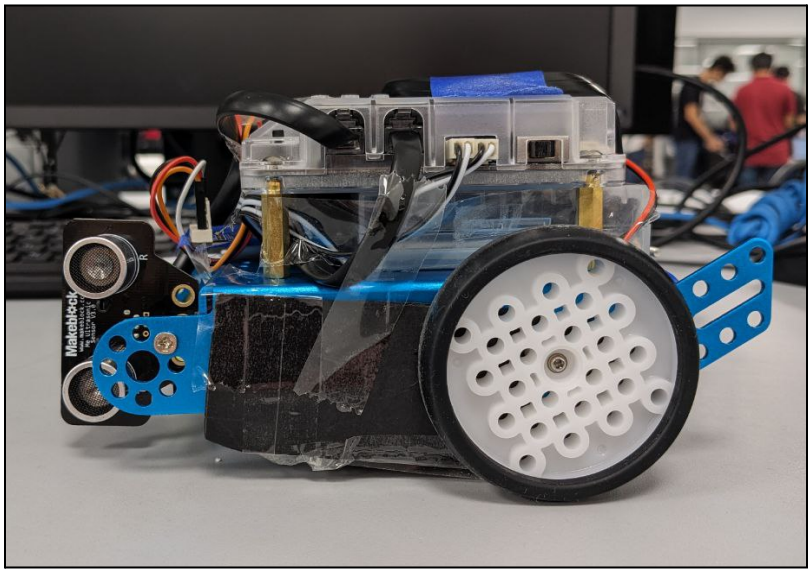
Bottom view of the mBot:



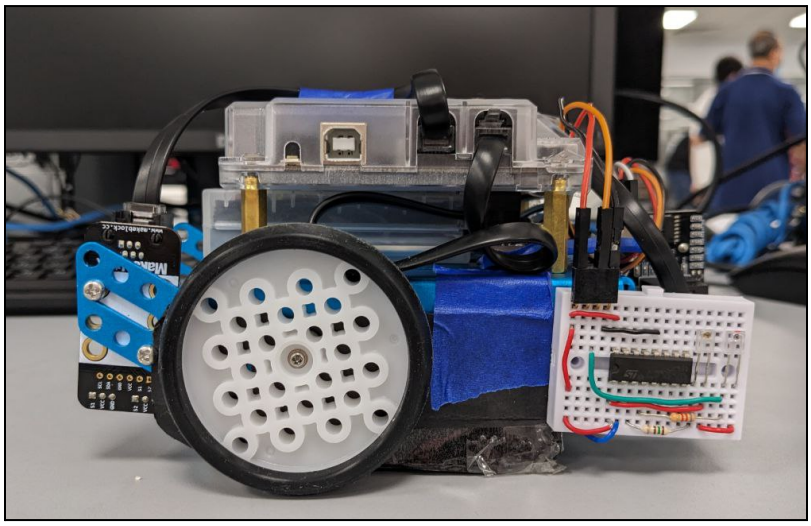
Top view of the mBot:



Side view of the mBot: (Left)



Side view of the mBot: (Right)



Bottom view of the mBot without skirting: