National University of Singapore

CG1111A GROUP PROJECT

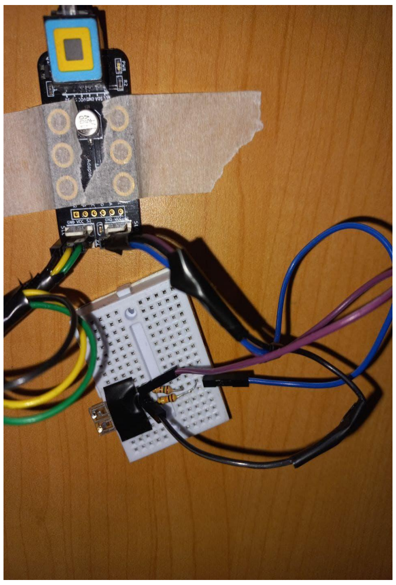
Studio Group Number: B03

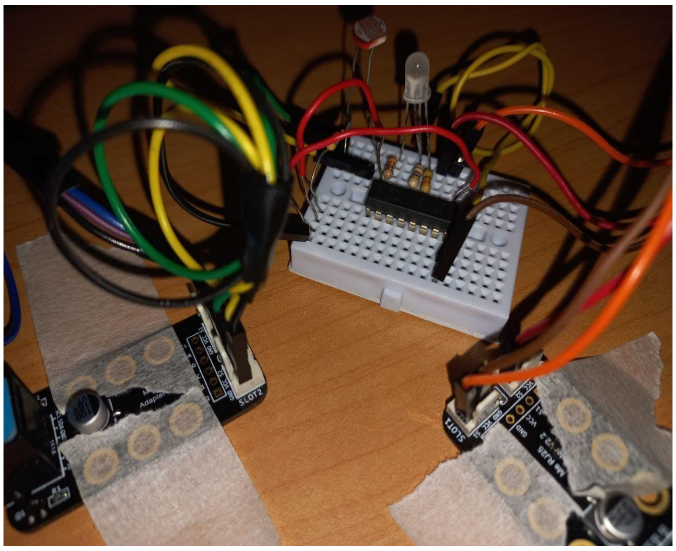
Section Number: 3

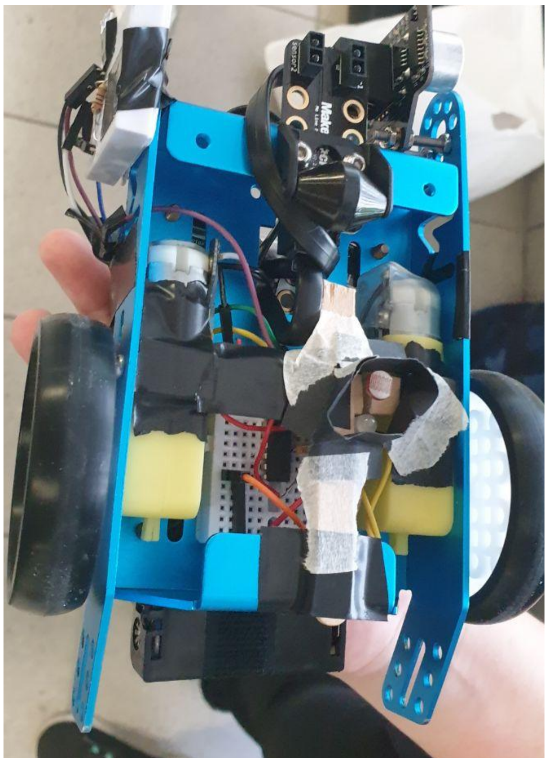
Team: 2

Name	Student ID
Ng Sihan, Ian	A0231053X
Ng Yan Zhen	A0238311N
Ng Yu Fei	A0240054W
Nguyen Quang Anh	A0240314X
Steven Antya Orvala Waskito	A0244129H

Pictures of mBot and Sensor Breadboard Circuits







Overall Algorithm

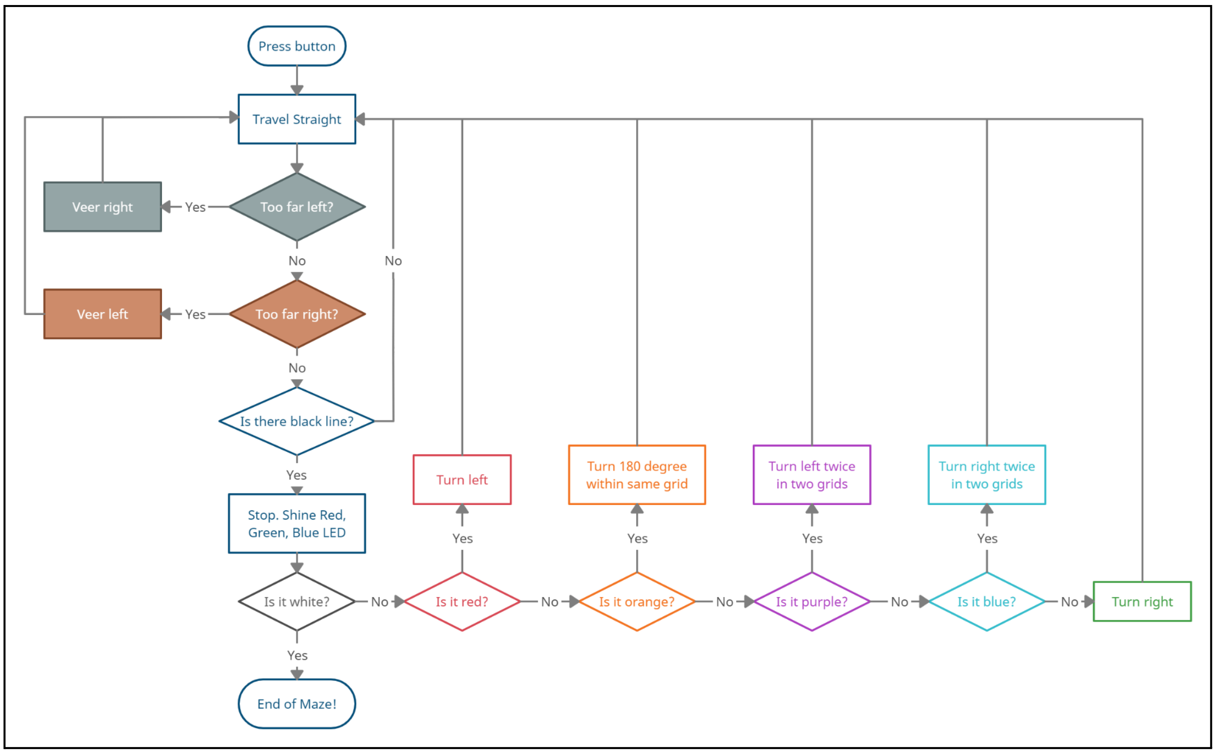


Figure 1: Flowchart of Overall Algorithm

At the start of the maze, the mBot waits for the onboard button to be clicked on before attempting to decipher the maze. This is initialised using a Boolean variable.

After the button is pressed, the mBot travels straight whilst reading and calculating the distances between the left (Ultrasonic Sensor) and right walls (Infrared Proximity Sensor). Should the mBot detect that it is displaced too far to the left, it will correct its course by adjusting its left wheel motor speed to be higher than that of the right. Similarly, if the mBot detects that it is displaced to the right, it corrects its course by adjusting its right wheel motor speed to be higher than that of the left. The corresponding values for the threshold of ultrasonic and IR distances as well as both motor speeds were decided upon through meticulous trial and error, ensuring the mBot is able to move sufficiently quickly through the maze without colliding with walls at turns and corners.

Along with calculating the US/IR distances and veering left/right, the mBot is concurrently searching for a black line through the Makeblock line sensor. Upon detection of a black line, the mBot stops, then attempts to decipher the coloured paper that is underneath it. The mBot runs a function to flash the Red, Green and Blue lights through the LED to obtain the corresponding RGB values. Based on the sets of lower and upper bounds for the RGB values of the various colours which were obtained through trial and experimentation, the mBot then accurately identifies the colour that is underneath it. Based on this colour, the mBot executes the corresponding function (e.g. turn_left or turn_right etc.) to manoeuvre towards the next section of the maze.

All of the above keeps running in a loop until the mBot detects that there is a white paper underneath it which would represent that it has reached the end of the maze. At this juncture, the Boolean variable which was initialised to TRUE at the beginning is now set to FALSE and the victory tune is played along with a randomised light show using the onboard LED to simulate a disco party.

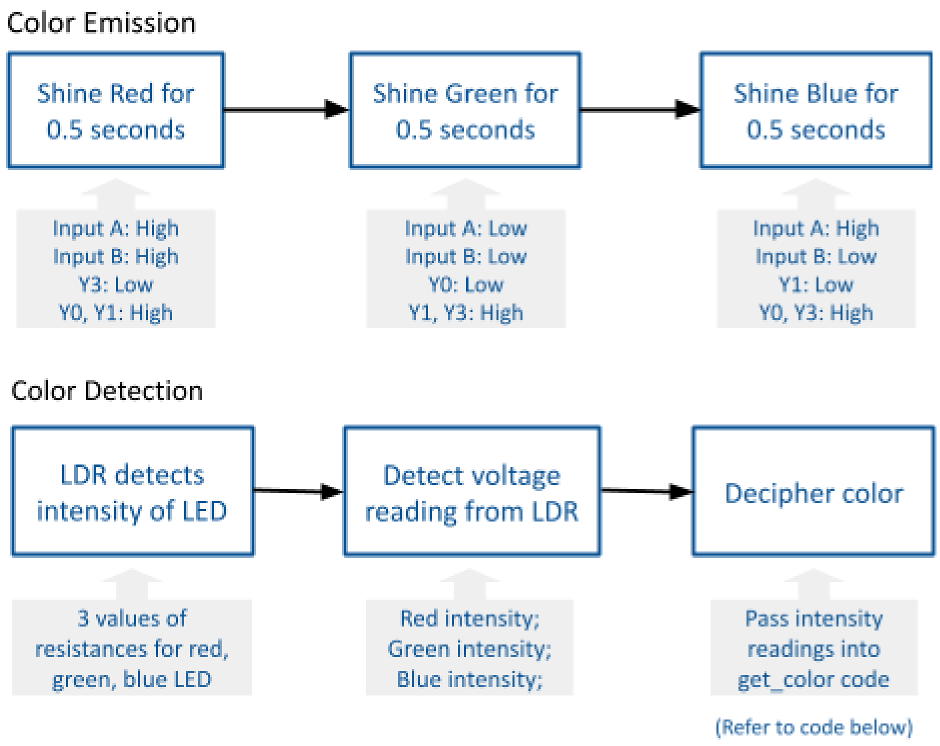
Subsystems

Colour Sensing:

Use the Red, Green, and Blue LED as emitter, and Light Dependent Resistor as the receiver.
In order to maximize the intensity of each light, we calculated the resistor that corresponds to the highest amperes with regards to the 8 mA limit. The LED was connected to the 2-to-4 decoder which was used to vary the signal (from input pins A and B) to light up each color of the LED individually (refer to Color Emission chart below). Detect the amount of light reflected by the surface and use the RGB values of the reflected intensity to decipher the color of the surface (refer to Color Detection chart below).

Color of LED	Resistance of Resistor used (Ω)	Connected to data outpin pin
Red	460	Y3
Blue	324	Y1
Green	323	Y0

Remarks: The Y2 pin was not used for better pin/cable management as the Y2 pin is directly underneath the positive pin of the RGB LED. Enable Pin was set to LOW.



Color Emission
Shine Red for 0.5 seconds
Shine Green for 0.5 seconds
Shine Blue for 0.5 seconds
Input A: High
Input B: High
Y3: Low
Y0, Y1: High
Input A: Low
Input B: Low
Y0: Low
Y1, Y3: High
Input A: High
Input B: Low
Y1: Low
Y0, Y3: High
Color Detection
LDR detects intensity of LED
Detect voltage reading from LDR
Decipher color
3 values of resistances for red, green, blue LED
Red intensity; Green intensity; Blue intensity;
Pass intensity readings into get_color code
(Refer to code below)

```
int get_intensity(int color) {
  // shine one LED color and return the reflected intensity value
  shine(color); // emit the specified color by changing signal to input A and B
  delay(RGB_WAIT);
  int intensity = getAveLDR(5);
  shine(BLACK);
  delay(RGB_WAIT);
  return intensity;
}

int get_color(int red, int green, int blue) {
  // decipher the color of the paper
  // determined through experimentation, the order of these checks ensure
  // greater consistencies as well
  if (green + red + blue > 2450) {
    return WHITE;
  }
  if (green + red + blue < 2100) {
    return BLACK;
  }
  if (red > 840) {
   if (green < 680) {
     return RED;
   }
```

```
    return ORANGE;
  }
  if (green < 750) {
    return PURPLE;
  }
  if (blue > 800) {
     return BLUE;
  }
 return GREEN;
}

int detect_color() {
  // obtain intensity of LED reflected and returns the color of the paper detected
  int red_intensity = get_intensity(RED);
  int green_intensity = get_intensity(GREEN);
  int blue_intensity = get_intensity(BLUE);
  return get_color(red_intensity, green_intensity, blue_intensity);
}
```

<u>Infrared (IR) Proximity Sensing (mounted on the right):</u>

- Use of the IR emitter and IR detector to determine the approximate distance between the IR and the reflected surface.

We chose to connect the IR circuit to ground instead of the 2-to-4 decoder as we deemed it more useful for the IR to be turned on at all times. We used a 148Ω resistor for the IR emitter and a 3430Ω resistor for the receiver. This was calculated to ensure the effective IR range is around the 4cm mark. The S1 pin in the RJ45 adapter is connected to the A0 Pin on the mBot board.

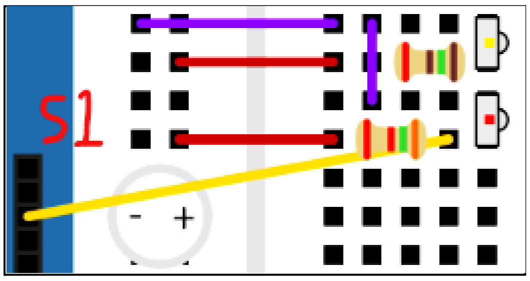


Figure 2: Illustration of IR Circuit

The IR voltage was recorded when it is 4cm, and the code will detect the close proximity when the voltage is less than that of 4cm.

Ultrasonic Sensor (mounted on the left):

- Use of the emitter to emit ultrasonic frequency and the receiver detects an echo of the frequency bounced back from a surface.

We attached the sensor using nuts and bolts such that the sensor will always be more than 3cm away from the walls. We connect the Ultrasonic directly to the RJ25 port 3, analog port of the mBot board. We used the inbuilt library of the mBot MeUltrasonicSensor sonic(PORT_3); and the function double dist = sonic.distanceCm(); to detect the distance of the mBot to the wall.

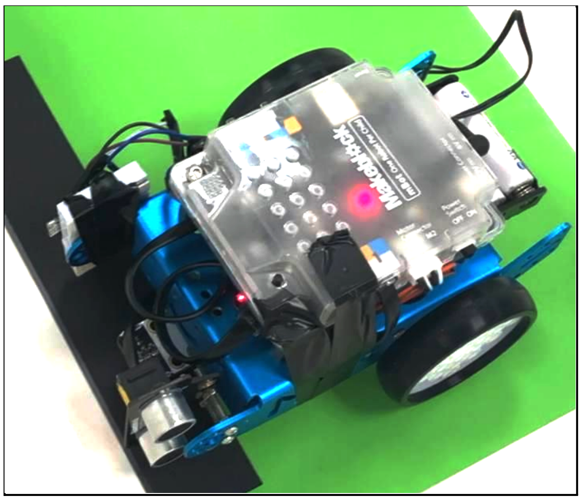


Figure 3: Top-right front view of the mBot

Algorithm for Going Straight:

- Use of IR mounted on the right and Ultrasonic Sensor mounted on the left of the mBot.

If the mBot is travelling towards one side of the walls, the code below will update and veer the bot away from the wall accordingly.

```
if (Ultrasonic_Distance < 6cm)
{
    // make sure it veers rightwards, away from wall
    veer_right();
} else if ((IR_Distance < 4cm) ||
        (Ultrasonic_Distance > 12cm && Ultrasonic_Distance < 25cm)) {
    // Ultrasonic_Distance < 25 because there is DEFINITELY no wall beyond 25cm,
    // make sure it veers leftwards, away from wall
    veer_left();
} else {   // continue to move forward
    straight();
}
```

Algorithm for Veering Left or Right:

```
// For Proximity Sensor
Serial.println("Veering left");
if (CURR_DIR != V_LEFT) { // if just started veering left
  VEER_EXTENT -= 1;
  CURR_DIR = V_LEFT;
  motor2.run(motorSpeedL);
  motor1.run(-motorRedR);
} else if (VEER_EXTENT <= -MAX_VEER_EXTENT) { // just keep going straight
  motor2.run(motorSpeedL);
  motor1.run(-motorSpeedR);
} else {
  VEER_EXTENT -= 1;
}
```

```
// For Ultrasonic sensor
Serial.println("Veering right");
if (CURR_DIR != V_RIGHT) { // if just started veering right
  VEER_EXTENT += 1;
  CURR_DIR = V_RIGHT;
  motor2.run(motorRedL);
  motor1.run(-motorSpeedR);
} else if (VEER_EXTENT >= MAX_VEER_EXTENT) { // just keep going straight
  motor2.run(motorSpeedL);
  motor1.run(-motorSpeedR);
} else {
  VEER_EXTENT += 1;
}
```

Algorithm for Turns (after color detection):

```
void turn_right() {
  Serial.println("Turning Left");
  stop(); // makes a stop, followed by full turn right
  motor2.run(-motorSpeedL);
  motor1.run(-motorSpeedR);
  delay(PERP_TIME);
  stop();
}
```

```
void turn_left() {
 Serial.println("Turning Left");
 stop();  // Makes a stop, followed by full turn left
 motor2.run(motorSpeedL);
 motor1.run(motorSpeedR);
 delay(PERP_TIME);
 stop();
}

void turn_left_twice() {
 Serial.println("Turning Left Twice");
 turn_left();
 straight();
 delay(TURN_DELAY);
 turn_left();
}

void turn_right_twice() {
 Serial.println("Turning Right Twice");
 turn_right();
 straight();
 delay(TURN_DELAY + 30);
 turn_right();
}

/**
 * Do a 180 degrees turn
 */
void kebelakang_pusing() {
 Serial.println("Turning Left");
 // Makes a stop, followed by full turn left
 CURR_DIR = LEFT;
 motor2.run(-motorSpeedL);
 motor1.run(-motorSpeedR);
 delay(1.9 * PERP_TIME); // 1.9 instead of 2 because for some reason, it keeps
 overshooting
 stop();
}
```

Calibration and Improvements

Calibrating the Color Sensor

- To ensure consistency of RGB voltage values obtained
  - Setting a longer duration of each color LED lighting up (adjusting the delay to be 200ms) to give the LDR time to respond to the changes in colour
- To reduce the effect of ambient light on LDR
  - Enclosing both the LDR and LED with black paper to ensure majority of the incident light on the LDR is from the LED
- To improve accuracy/reliability of LDR
  - Fixing the position of the LDR and LED in place using ice cream sticks to prevent lateral and diagonal displacement which allowed for consistent readings
- To ensure detected RGB range of each color results in correct color deciphered
  - For each calibration done, a minimum of three readings were taken for each coloured paper
  - Estimate a reasonable value of the three readings, check and modify the threshold values of each RGB condition accordingly
  - Calibrate the threshold values in the setting most closely resembling the test location for greatest accuracy.

We also met early on test day before the final maze run in order to calibrate and confirm the color detection values to fit the lab's environment and the day's lighting conditions.

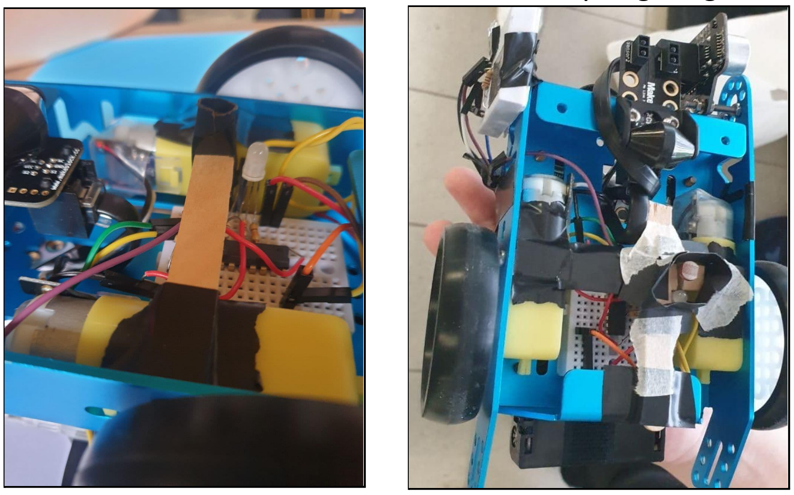


Figure 4: Iterations of Modifications made

Calibrating the IR Proximity Sensor

- Obtained lower and upper bounds of IR readings through the Serial Monitor
  - Placed mBot at various positions within the maze (e.g. displaced all the way to the left)
  - Tilted mBot at different angles to simulate it turning around a corner

- Through the above trial and experimentation, we decided on the threshold to be about 4-5cm as it can still be used in conjunction with the US sensor in the logical if-else statements

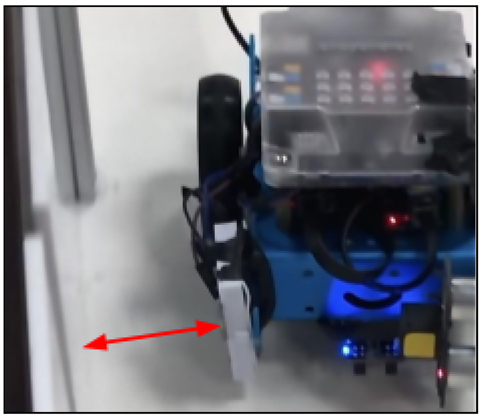


Figure 5: Illustration of Checking the Threshold Value

<u>Calibrating the Ultrasonic Sensor</u>

- Prioritised as main sensor to calculate distance from the walls as it gave more consistent and accurate readings than the IR sensor
- Modified the distance thresholds through multiple iterations of test runs to arrive at the most ideal lower bound of about 6 to indicate that the mBot should start veering towards the right, and a reading of about 12 to 25 to indicate that the mBot should start veering left to correct its course
- Decreased delay in loop function to increase frequency of calculation of distance from walls, which proved to be especially important after the mBot has executed a turn

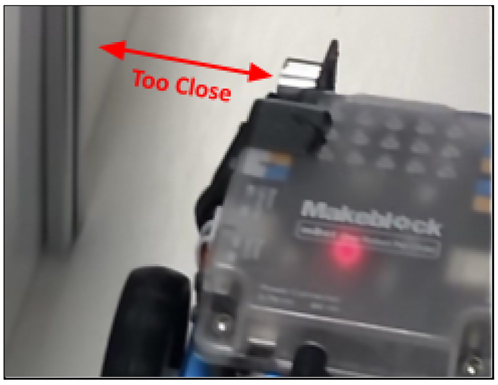


Figure 6: Illustration of Checking Distance Threshold

Work Division

Hosea and Ian were mainly in charge of doing the code and algorithm for the whole mBot while Andrew, Steven and Yan Zhen took charge of the building of the mBot and assembly of the circuits for the various subsystems onto the mBot. Whenever our bot encounters some issues, like inconsistent color values detected, the hardware team will firstly check for any changes on the circuits or loosened parts, and make the necessary adjustments, before handing over to the software team to fine-tune or re-calibrate the mBot depending on our hypothesised error found.

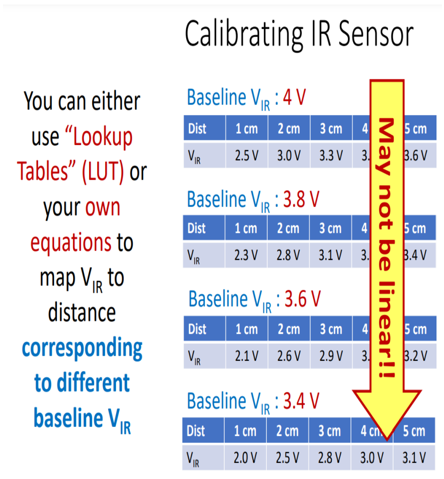
Difficulties faced

Motor variance

At some point, the motors started to rotate at slightly different RPMs despite the same potential difference and current, causing the car to veer to the right without reason. As the batteries discharge further, the severity of this problem increases. In order to mitigate this, through trial and error, we set different potential differences across the 2 motors which would create the least deviation from a straight path.

Another problem faced was that sometimes when turning, the mBot would be left slightly off-centre. In the initial stages of the code, the mBot would continue to veer left/ right if it found itself to be too close to the wall on the opposite side, causing it to turn at a very high angle. Subsequently, it would oscillate constantly while moving forward, rather than go straight, or in the worst case even head straight into a wall. Hence, we added a max veer extent, which tracks and limits the angle it will veer off from.

Flaws of the Proximity Sensor



Calibrating IR Sensor

You can either use "Lookup Tables" (LUT) or your own equations to map V_{IR} to distance corresponding to different baseline V_{IR}

Baseline V_{IR} : 4 V

Dist	1 cm	2 cm	3 cm	4	5 cm
V_{IR}	2.5 V	3.0 V	3.3 V	3.	3.6 V

Baseline V_{IR} : 3.8 V

Dist	1 cm	2 cm	3 cm	4	5 cm
V_{IR}	2.3 V	2.8 V	3.1 V	3.	3.4 V

Baseline V_{IR} : 3.6 V

Dist	1 cm	2 cm	3 cm	4	5 cm
V_{IR}	2.1 V	2.6 V	2.9 V	3.	3.2 V

Baseline V_{IR} : 3.4 V

Dist	1 cm	2 cm	3 cm	4 cm	5 cm
V_{IR}	2.0 V	2.5 V	2.8 V	3.0 V	3.1 V

May not be linear!!

Initially, we noticed that the proximity sensor could not determine its distance from the wall with the same consistency and accuracy as the ultrasonic sensor. Hence, we just used it to detect the presence of a wall at a distance which was within the IR sensor's sensitive range and less than or equal to the distance we wanted the mBot to be at.

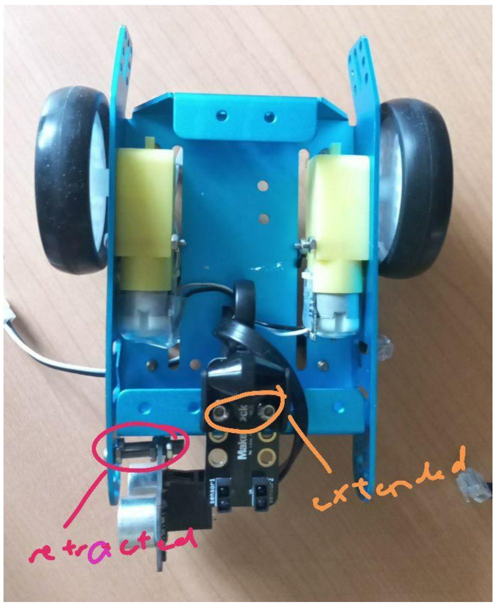
However, we realised that the proximity sensor's effective range was estimated to be about 1-3cm, based on previous studios. This was a lot shorter than the distance (~6-10cm) the mBot was supposed to be from either side of the hallway in the maze. To mitigate this issue, we increased the resistance of the resistor in series with the IR receiver, to increase the effective range of the proximity sensor to about 1-5cm, (any more than that would make the values less distinct). We also placed it on the outer side of the mBot, making it closer to the right wall, bringing its sensitive range closer to what was meaningful to us.

However, we also found out through experimentation with the proximity sensor that the readings of the proximity sensor would surge when its angle slightly deviates from the wall.

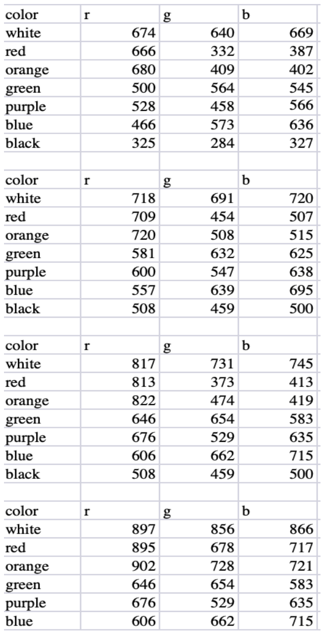
In order to mitigate this, the IR sensor was used to support the ultrasonic sensor which was much more reliable. Through experimentation, we found that the ultrasonic sensor would only detect a maximum distance of 25cm unless there is no wall on that side. Since the length of the hallway was constant, this allowed us to have an estimate of the distance from the right to cross-reference the IR proximity sensing with.

<u>Black line detection</u>

Initially, we faced some difficulty getting the robot to stop consistently at the black line as it would occasionally move past and ignore the line. We attributed this to a delay() that was longer than necessary, causing the mBot to not detect the black line in time to stop. We rectified this by reducing the duration of delay in our loop function to allow the mBot to look for the black line more frequently and also reducing the motor speed slightly to give it more leeway for detection of the black line.



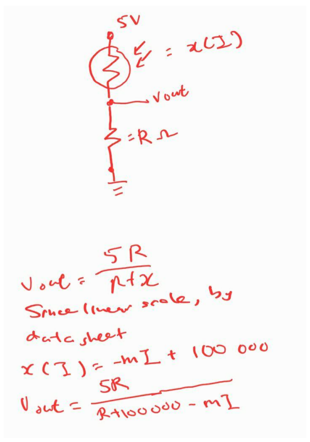
When we finally set the bot onto the course, we realised that the hallways of the maze were narrower than we expected. When turning, the bot would bump into the walls, especially when the two sensors which were protruding out of the main body would make contact with the adjacent walls. Eventually, we found that shifting the line sensor forward and retracting the 2 sensors ensured that it would not bump into the wall. We also coded the bot to move with greater precision, as close to the centre and parallel to the walls as possible by modifying the distance thresholds.



color	r	g	b
white	674	640	669
red	666	332	387
orange	680	409	402
green	500	564	545
purple	528	458	566
blue	466	573	636
black	325	284	327
color	r	g	b
white	718	691	720
red	709	454	507
orange	720	508	515
green	581	632	625
purple	600	547	638
blue	557	639	695
black	508	459	500
color	r	g	b
white	817	731	745
red	813	373	413
orange	822	474	419
green	646	654	583
purple	676	529	635
blue	606	662	715
black	508	459	500
color	r	g	b
white	897	856	866
red	895	678	717
orange	902	728	721
green	646	654	583
purple	676	529	635
blue	606	662	715

Colour detection

We also faced a significant obstacle in getting the LDR and LED to return consistent readings of RGB values when we were calibrating our colour detection. The RGB readings would vary widely in the magnitude of hundreds (based on analogRead output). This made calibration of the color detection nearly impossible because the variance was larger than the difference between each colour. Initially we thought that this could be solved by applying a function over the RGB readings. I.e. create a function of the R, G and B readings in order to find some correlation which can display consistent distinction between the colors. We tried everything from a constant offset, normalising, using ratio R/G & R/B, using more complex functions of polynomials, reciprocals of polynomials, and even HSV/HSL. However, there was no function that could provide a consistent distinction between the colors we needed. Through some research, we also found that although there was a positive relationship between the voltage readings and the intensity of light for the LDR, it was not a linear relationship, hence, most computer vision functions like HSV would not help find the issue.

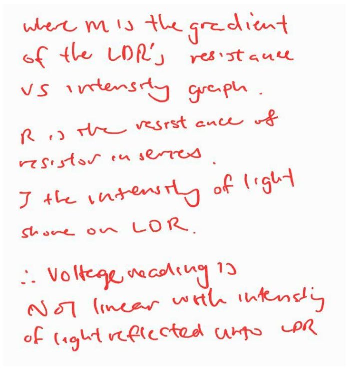


$V_{out} = \frac{5R}{R+x}$

Since linear scale, by data sheet

$x(I) = -mI + 100\,000$

$V_{out} = \frac{5R}{R+100000-mI}$



where m is the gradient of the LDR's resistance vs intensity graph.

R is the resistance of resistor in series.

I the intensity of light shone on LDR.

∴ Voltage reading is NOT linear with intensity of light reflected unto LDR

Our next hypothesis was that the problem was due to the LED and LDR. We swapped the LDR for a new one, but it still showed the same inconsistency. Then we added a tunnel around the LDR, to mitigate the effect of ambient light on the LDR. However, this did not solve the issue. We finally rectified this problem by clamping down both the LDR and LED in place using ice cream sticks, which were then taped to the motors (which is screwed to the chassis) to ensure no lateral and diagonal displacement when the mBot turns and shifts. This helped to ensure that the emitting light from the LED and incident light on the LED would be fairly constant and at approximately the same angle of incidence, allowing for more consistent RGB readings. A makeshift “tunnel” using black paper was also placed around the

LDR and LED, rather than just the LDR, to reduce the effect of ambient light on the color detection readings. We also altered the resistance in series with the LDR to maximise the range of colour detection. However, since the range varied based on environmental conditions, we settled for the resistance value that was overall most useful across all environmental conditions tested.

On top of this, we considered comparing the HSV values (rather than RGB), calibrated against black and white paper, but eventually decided against it since we knew this was not meaningful. This option assumes that there was a linear correlation between intensity of the colour and the RGB readings (which are voltage readings). Moreover, the RGB values of each of the 5 colours had to be hard coded anyway. All of these just added increased complexity to the processing which was counter-intuitive for our IoT device. Ultimately, we found these modifications to the hardware to be consistent enough to meet the project requirements.

<u>Protruding wires</u>

At some point, we found that our bot could meet the requirements of the project. However, the wires were untidy and protruding out, occasionally catching onto the wheels or making contact with the surrounding walls, causing the bot to veer off course. A simple inspection revealed that there was a major overhaul of wires required to achieve proper cable management. We planned how the wires were to be positioned, tucking most of the circuits under the mBot or between the arduino board and the main base of the mBot. In the end, we had only 2 thick black wires protruding slightly from the front and 2 wires on the side with the IR sensor. These were taped down during the test run to ensure they would not catch onto any object.