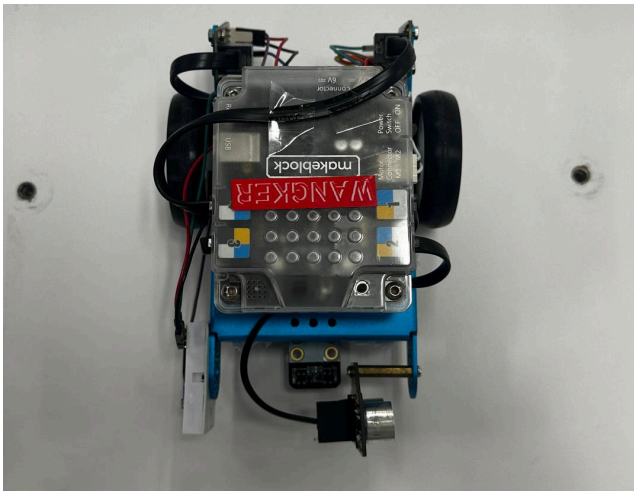
National University of Singapore

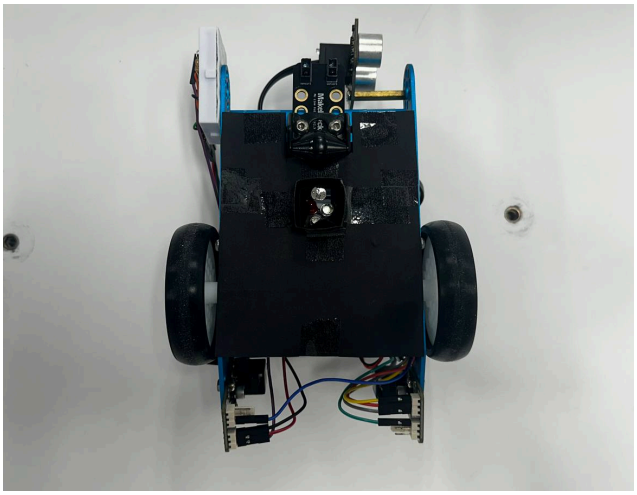
CG1111A B06-S1-T2 Project Report

1. Product Appearance

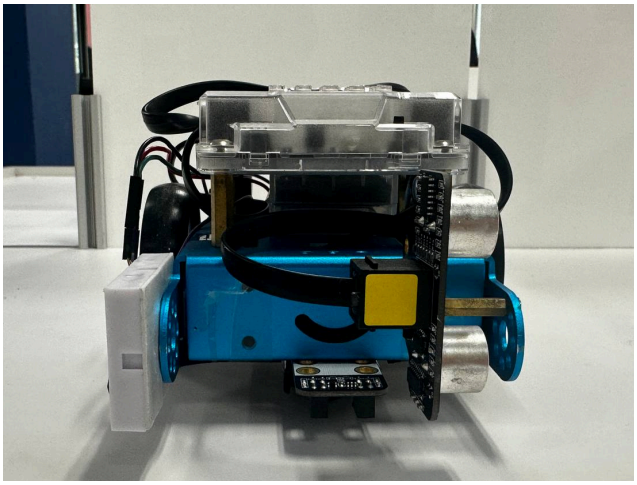
1.1 Main Body of the mBot



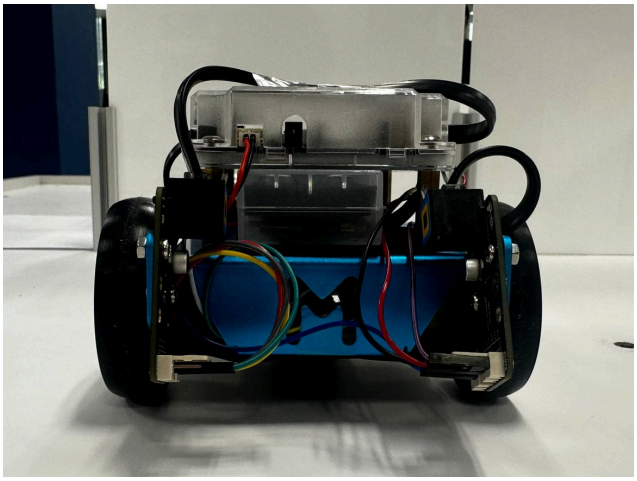
Top View



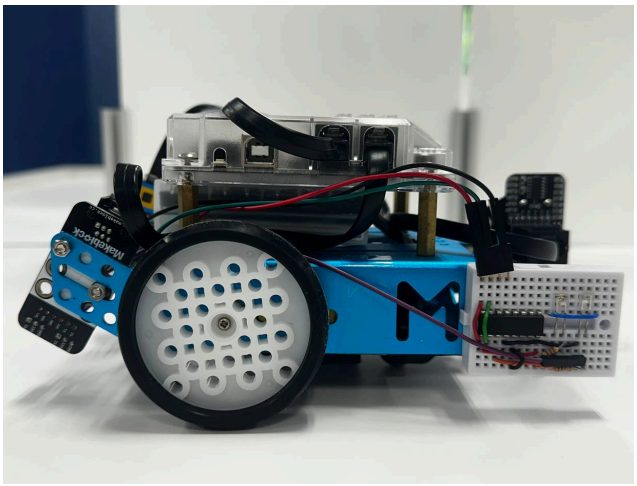
Bottom View



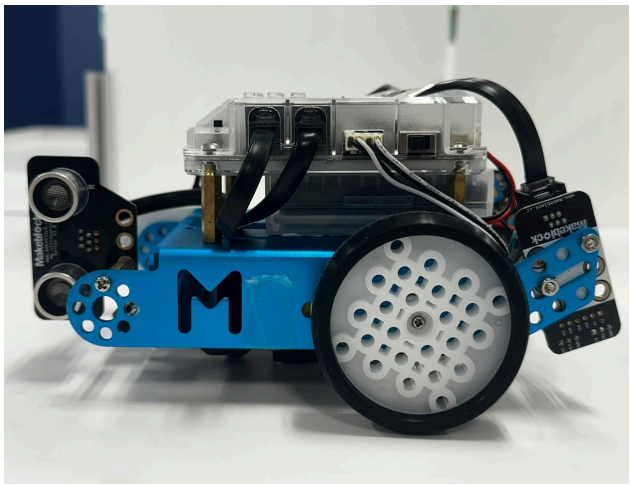
Front View



Back View

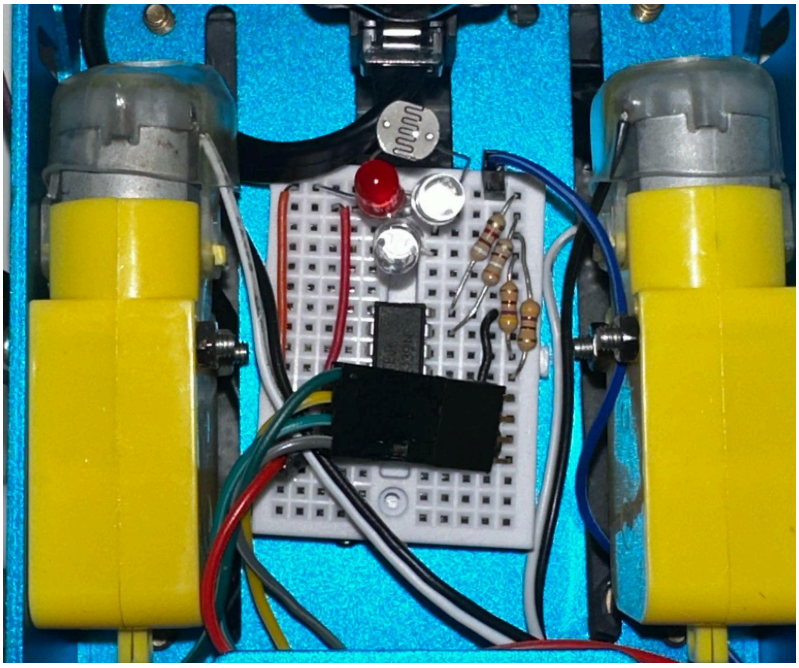


Right View



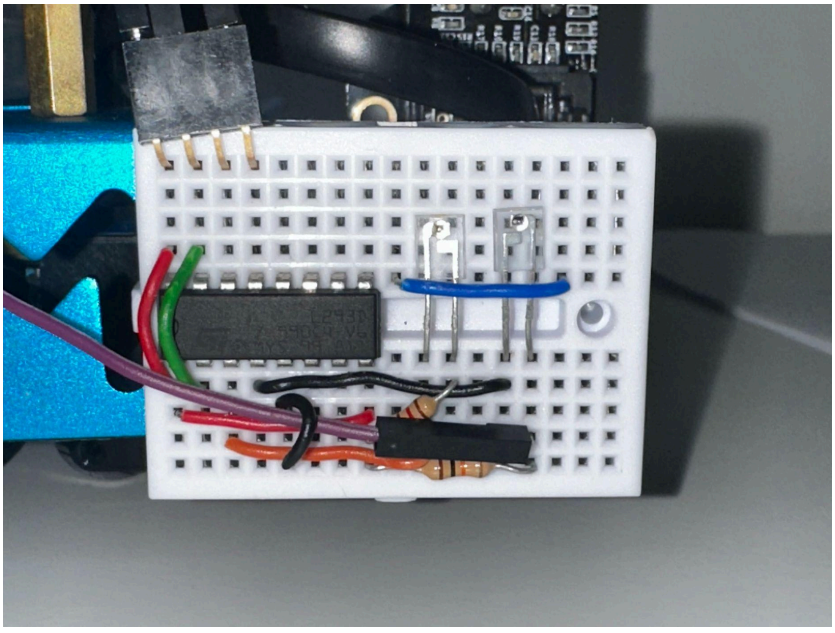
Left View

1.2 LDR Circuit View



LDR Sensor Circuit

1.3 Infrared Sensor Circuit View



IR Sensor Circuit

2. Overall Algorithm

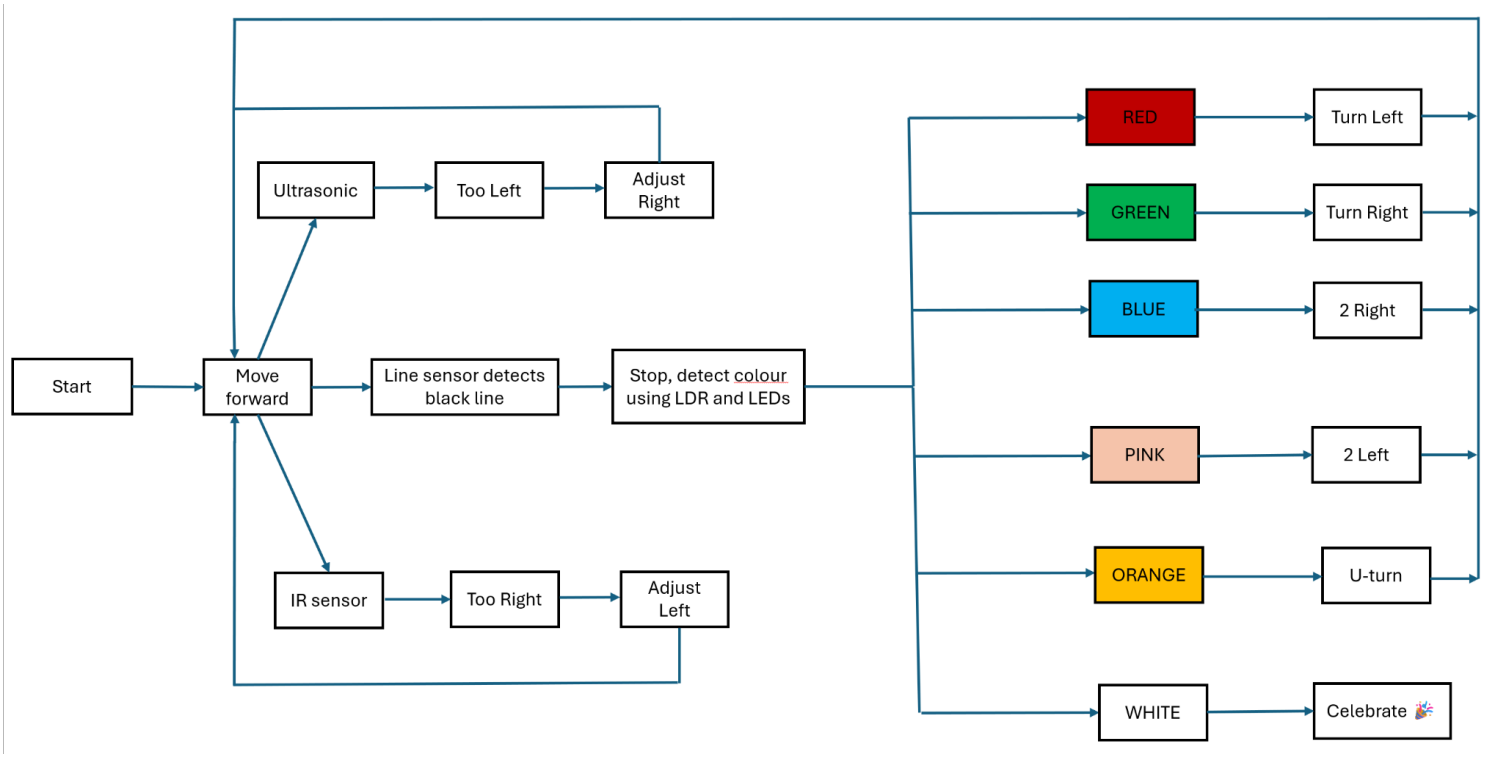


Fig. 1: Program Flowchart

In this chapter, we will briefly explain the algorithm and program flow behind our mBot Maze Solver. The main program of our mBot is constantly running in a loop located inside the void loop function. According to the Figure 1 flowchart, the main program can be categorised into three phases – Obstacle Avoidance, Stop Line Detection and Colour Detection phase. We will explain each phase in detail in the following section.

2.1 Obstacle Avoidance

When the switch of the mBot is turned on, the mBot will enter the loop of the program. Both the infrared (IR) Sensor and ultrasonic Sensor will be turned on. The ultrasonic sensor is located on the left side of the mBot and it is used to measure the distance to the left wall. The IR sensor is located on the right side of the mBot and it is used to detect the presence of the right wall. We set a certain range for the distance measured. If the mBot moves too close to the left side of the wall based on the ultrasonic sensor reading, the mBot will adjust slightly to the right to counter this leftward movement. If the mBot moves too close to the right side of the wall based on the IR sensor reading, the mBot will adjust slightly to the right. Otherwise, the mBot will just move forward. The left and right adjustment ensures that the mBot will travel almost in a straight line and not collide with the left and right walls of the maze. We will explain how each sensor works in detail in the later chapters.

2.2 Line Detection

For this project, we use the default Line follower sensor of the mBot package for black line detection. This line follower sensor is mounted at the front of the mBot and is facing downwards. As the mBot travels, it will eventually reach a colour paper. At the front of the colour paper, there will be a black line. As the line follower sensor reads that a black line is present, the mBot will stop and the colour sensor circuit will be turned on for colour detection.

2.3 Colour Detection

When the colour sensor circuit is turned on, three Light Emitting Diodes (LED) namely Red, Green and Blue will turn on one after another. The change in resistance and thus potential difference across the Light Dependent Resistor (LDR) will be measured, which corresponds to the intensity of the reflected light. The program will then interpret and decode which colour the paper is. Once decoded, an action will be executed based on the colour detected as shown in Table 2. The instructions for all the actions are given to us inside the Amazing Project document. Finally, when the LDR detects the white colour paper, the mBot has completed the maze and the motors will be turned off indefinitely. We chose to play the Star Wars theme song as the celebratory tone with the in-built LED displaying colours corresponding to each unique note (Refer to Appendix A).

Colour	Interpretation
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

Table 2: Interpretation of Colour Paper

3. Overall Circuitry

In this chapter, we will briefly explain all the port connections on the mCore and their designated functionality. The mCore has 4 RJ25 ports. Ports 1 and 2 are digital input and output ports. We connected the Line Follower Module to Port 1 and the Ultrasonic Sensor Module to Port 2 respectively as these two components only require digital input and output signals. Ports 3 and 4 are analog input and output ports. These ports are used for analog signal transmissions. The detailed RJ25 Port Circuit layout is shown in Figure 3.1 as follows.

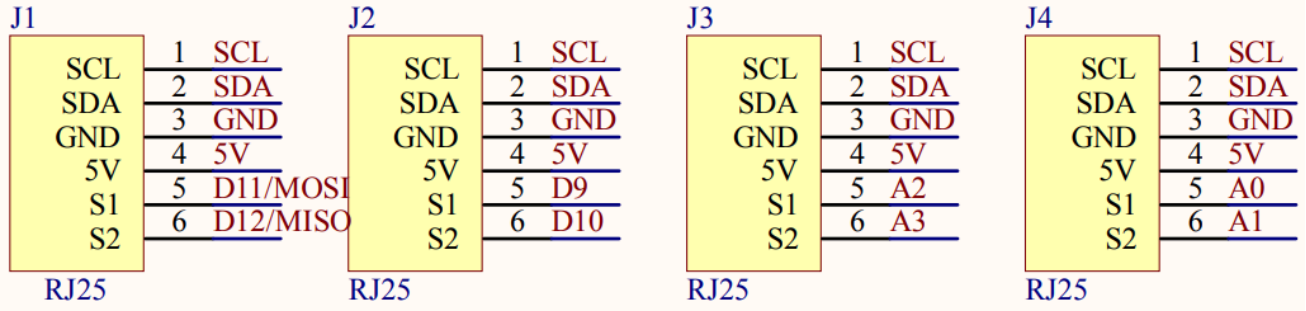


Fig. 3.1: RJ25 Ports Circuit Layout

Examining our circuitry needs, we require analog pins for the three LEDs, IR emitter and detector and LDR voltage reading. If we use one pin to control one component, it is not enough since three pins are needed for three LEDs, one pin is needed for an IR emitter, one pin is needed for an IR detector, and one pin is needed for LDR voltage reading. Due to the limited number of Analog Input/Output pins available on the RJ25 adapters, we will need to use an HD74LS139P 2-to-4 decoder IC chip to expand our circuit as shown in Figure 3.2.

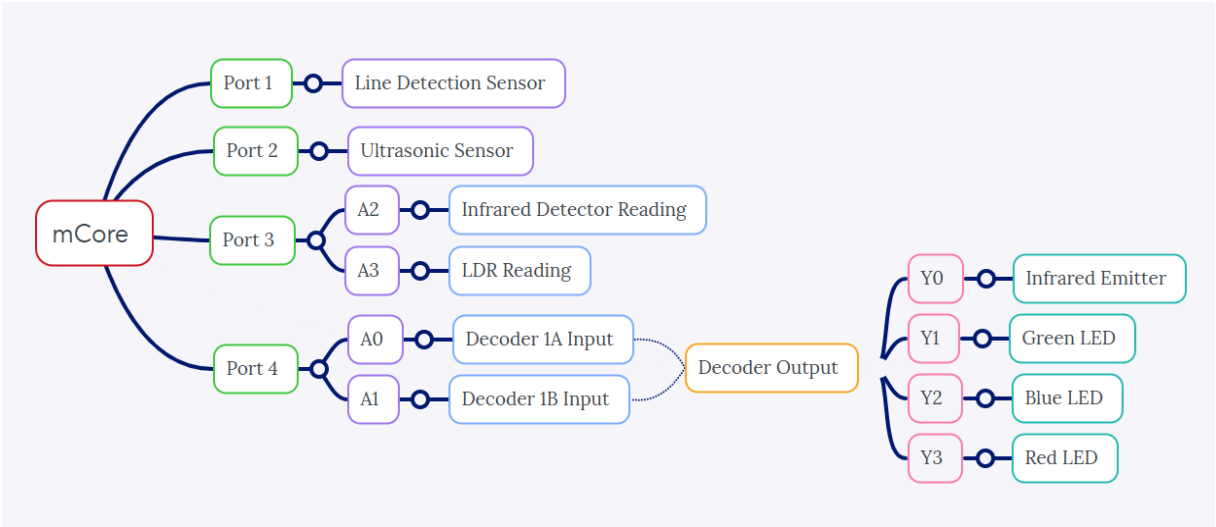


Fig. 3.2: Port Connection of mCore

4. Line Detection Sensor

The Me Line Follower inside the mBot package consists of two sensors, each with an IR transmitting LED and an IR static induction phototransistor. Since the Line Follower Sensor only requires a digital pin, we connected the Line Follower Sensor to Port 1 using the RJ25 cable. Based on the mBot Introduction document, the Line Follower Sensor is specifically used to make the mBot follow a black line. However, in this case, we don't want the Line Sensor to follow the black line but instead, we want it to stop at the black line. Hence, we will need to adjust the action of the code as illustrated in Figure 4 below.

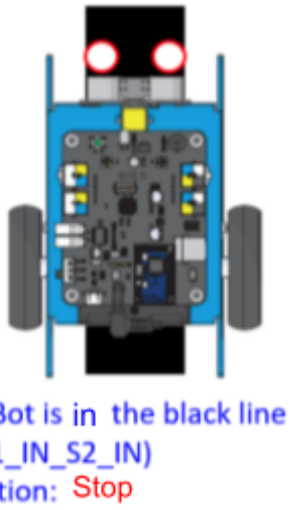


Fig. 4: Illustration of Line Follower

```
#include <MeMCore.h>

MeLineFollower lineFinder(PORT_1);
MeDCMotor leftMotor(M1);
MeDCMotor rightMotor(M2);

void setup() {
  Serial.begin(9600); //begin serial communication
}
void loop() {
  int sensorState = lineFinder.readSensors();
  if (sensorState == S1_IN_S2_IN) { //if inside the black line
  leftMotor.run(0); // mBot will stop
  rightMotor.run(0);
  } else { //if not inside the black line
  leftMotor.run(-200); // mBot will move forward
  rightMotor.run(200);
  }
}
```

5. Ultrasonic Sensors

In this chapter, we will explain the principle of ultrasonic sensors used onboard the mBot. The project information mentioned that the ultrasonic sensor only requires digital pins. Therefore, we connected the ultrasonic sensor to Port 2 using the RJ25 cable.

5.1 Principle of Ultrasonic Sensor

The sensor has a transmitter that emits ultrasonic waves of 42kHz in the form of pulses. These waves travel through the air at the speed of sound. When the emitted ultrasonic waves encounter the wall, they are reflected back toward the sensor. The sensor's receiver detects the reflected waves also known as echo. The sensor measures the time taken for the ultrasonic waves to travel to the wall and back to the sensor. This is known as the time-of-flight. Using the speed of sound travelled in air (approximately 340 m/s) and the total time taken for the ultrasonic waves to be reflected back from the wall, we can calculate the distance between the wall and the mBot. The division by 2 takes into account the round trips.

$$Distance\ Between\ the\ Wall\ and\ the\ Bot\ = \frac{Speed\ of\ Sound \times Time\ Of\ Flight}{2}$$

5.2 Initial Code for Ultrasonic Sensor Testing

We are using the default `ultraSensor.distanceCm()` function that is already included inside the MeMCore library to measure the distance between the mBot and the wall. Using the serial monitor, we can observe the distance between the Bot and the wall. After testing, we conclude that any distance below 8 cm is too close to the left wall. When the ultrasonic sensor detects the mBot to be too close to the left wall (i.e. `ultradistance <= 8`), the mBot will adjust right. The implementation code can be found in Chapter 6.4.

```
#include <MeMCore.h>

MeUltrasonicSensor ultraSensor(PORT_2);

void setup() {
  Serial.begin(9600); //begin serial communication
}
void loop() {
  float ultradistance = ultraSensor.distanceCm(); //Default Function in the library
  Serial.print("Distance: "); //Using Serial Monitor to print out the distance
  Serial.print(ultradistance);
  Serial.print("cm");
  delay(100);
}
```

6. IR Proximity Sensor

Other than using the Ultrasonic sensor to prevent the mBot from bumping into the maze walls, we implemented the IR emitter and detector on the right side of the mBot to detect the presence of the right wall.

6.1 Circuit Implementation

Similar to the studio 12 handout, we used the same circuit but modified it to include the L293D motor driver chip. This allowed us to control the IR circuit, switching it on and off based on the voltage input to the motor driver. We connected 5V (Vcc) to pins 1 and 8 via pin 16 of the motor driver chip. Because of the limited analog output and input pins availability of the mCore as mentioned in Chapter 3, we connected the 2-to-4 decoder Y0 pin to pin 2 (Input 1) via pin 15 of the motor driver chip to control the IR circuit. Lastly, we connected the IR emitter to pin 3 (Output 1) of the L293D chip. To get the analog reading from the IR detector, we connected the purple wire to Analog Pin A2 on the RJ25 adapter connected to Port 3 of the mCore, which reads the variation in voltage. The illustration of the circuit layout can be seen in Figure 6.1.

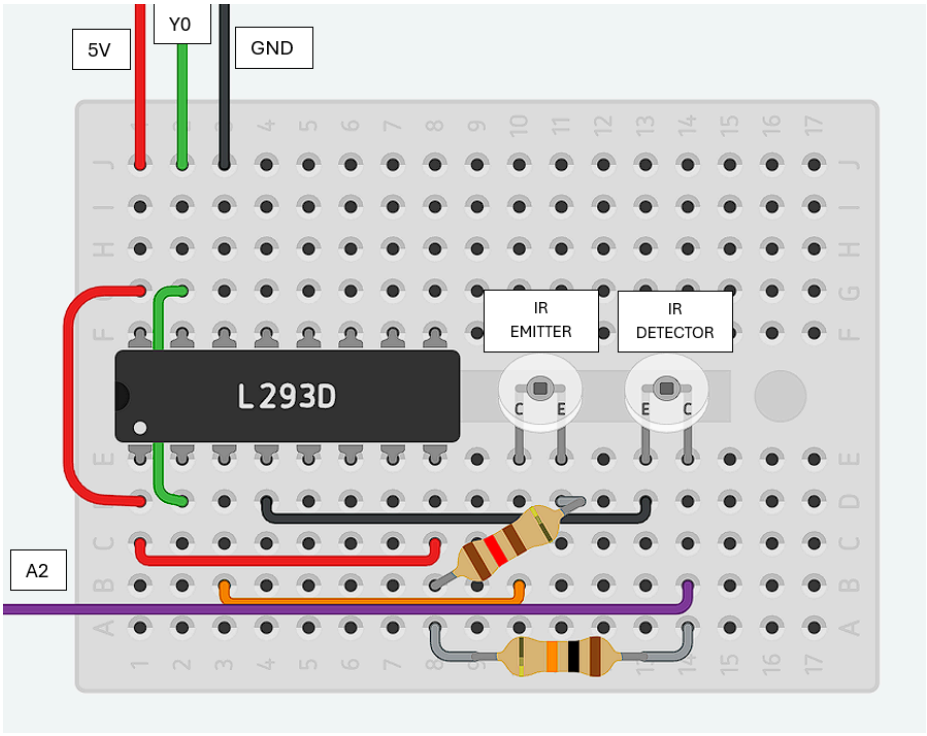


Fig. 6.1: TinkerCAD Circuit for IR Sensor

6.2 Choice of Resistors

IR Receiver: Based on the IR emitter datasheet and project information, the ideal maximum current for the receiver should be limited to 50mA. To achieve this, we chose to use a 120Ω resistor instead of the 150Ω resistor suggested in the handout. This adjustment allowed us to successfully keep the current below 50mA.

IR Detector: As noted in the project information, the 8.2kΩ resistor used in the studio may not be optimal, as a resistor with larger resistance can make the receiver voltage more responsive. However, the resistance value cannot be increased too much, as it would make the IR receiver overly sensitive. Fig. 6.2 shows the results from our studio report. As the gradient of the graph reflects the responsiveness of the IR sensor to changes in distance, we can observe that after 4cm, the voltage increase slowed down. Therefore, with the 8.2kΩ resistor, the optimal working range of the IR receiver was limited to 4 cm. To extend the working range of our IR receiver while maintaining an appropriate level of sensitivity, we opted to use a 10kΩ resistor in the IR detector circuit.

After choosing the resistors, we carried out several rounds of testing to check the optimal working range of the IR sensors with the new set of resistor values. Fig. 6.3 shows the result of our testing. From the graph, we can see that the slope of the graph only starts to decrease after 8 cm. This shows that with the new set of resistors, the optimal working range of our IR has increased from 4 cm to 8 cm. This allowed our mBot to be more responsive when travelling through the maze.

Emitter resistance /Ω	Receiver resistance /Ω	distance/cm	Vout/V			
			1	2	3	AVG
150	8.2k	1	1.81	1.79	1.82	1.80667
		2	3.12	3.1	3.08	3.1
		3	3.74	3.72	3.7	3.72
		4	4	4.02	4.01	4.01
		5	4.02	4.04	4.02	4.02667
		6	4.1	4.11	4.12	4.11
		7	4.11	4.11	4.09	4.10333
		8	4.09	4.1	4.1	4.09667
		9	4.2	4.22	4.21	4.21
		10	4.21	4.2	4.23	4.21333
		11	4.21	4.22	4.22	4.21667

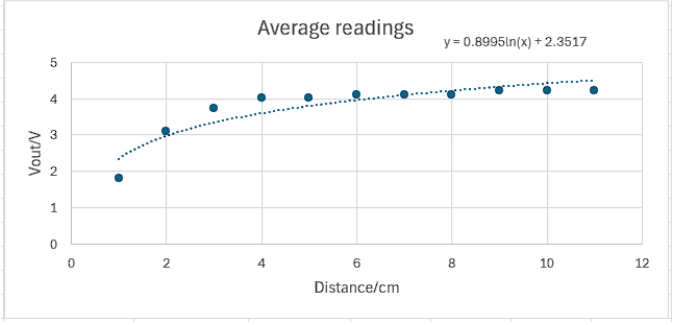


Fig. 6.2: Readings of IR Sensors Output Voltage vs Distance from Studio 12

Emitter resistance /Ω	Receiver resistance /Ω	distance/cm	Vout/V			
			1	2	3	AVG
120	10k	1	0.29	0.28	0.3	0.29
		2	0.7	0.69	0.78	0.723333
		3	1.07	1.08	1.1	1.083333
		4	1.87	1.81	1.78	1.82
		5	2	2.01	1.98	1.996667
		6	2.13	2.14	2.12	2.13
		7	2.21	2.2	2.2	2.203333
		8	2.2	2.21	2.23	2.213333
		9	2.21	2.21	2.22	2.213333
		10	2.21	2.2	2.21	2.206667
		11	2.21	2.21	2.22	2.213333

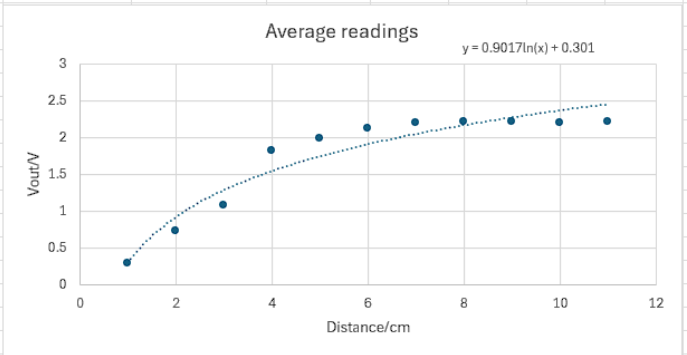


Fig. 6.3: Readings of IR Sensors Output Voltage vs Distance with New Sets of Resistors

6.3 Mounting of the Sensor

The IR sensor was placed on the right side of the mBot, enabling it to adjust to the left when it detected proximity to the right wall. Initially, the circuit board was positioned in a way that partially covered the hollow 'M' on the main body. However, after several trial runs, we observed that the mBot could not adjust in time when a wall was detected on the front right. To resolve this issue, we repositioned the breadboard, as shown in Fig. 6.4, allowing it to protrude outward and align with the ultrasonic sensor on the opposite side. This adjustment significantly improved the mBot's responsiveness when it was too close to the right wall.

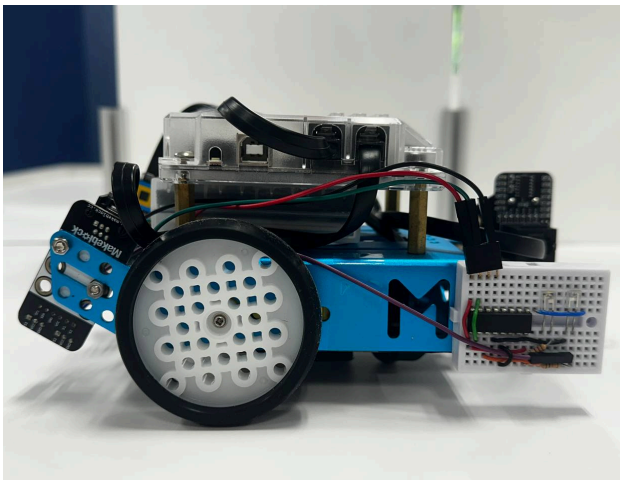


Fig. 6.4: Finalised Position of the IR Sensor

6.4 Implementation of Code

```
175 void loop() {
176   ir_value = analogRead(A2);
177   ultradistance = ultraSensor.distanceCm();
178   forward();
179   Serial.println(ir_value);
180   if (ultradistance <= 8) // too close to the left wall. Before Callibration: 10
181   {
182     adjustRight();
183   }
184   if (ir_value <= 225) // too close to the right wall. Before Callibration: 150, 170,180,200,250,275,
185   {
186     adjustLeft();
187   }
188   else{
189     forward();
190   }
```

We used `int analogRead(uint8_t pin)`, which is a built-in function provided by the Arduino core library, to directly read the value from the IR detector. After multiple calibrations, we found out that when `ir_value <= 225`, the mBot will be too close to the right wall and need to adjust left.

6.5 Challenges Faced with IR sensor and Improvement

6.5.1 Tackling Ambient IR Variation

Even though the IR sensors worked as expected during the graded runs, we identified a critical factor that we had initially overlooked: the influence of ambient IR variations in the environment on the sensor's performance. IR sensors are highly sensitive to the IR spectrum of light, which is not only emitted by the IR transmitter but also present in the environment due to sources such as sunlight, artificial lighting, or even heat-emitting objects. These external IR sources can interfere with the sensor's ability to accurately detect objects and measure distances. As shown in Figure 6.5 below, which is a graph in project information, we can observe that as the ambient IR increases, the IR detector voltage drops significantly for the same distance.

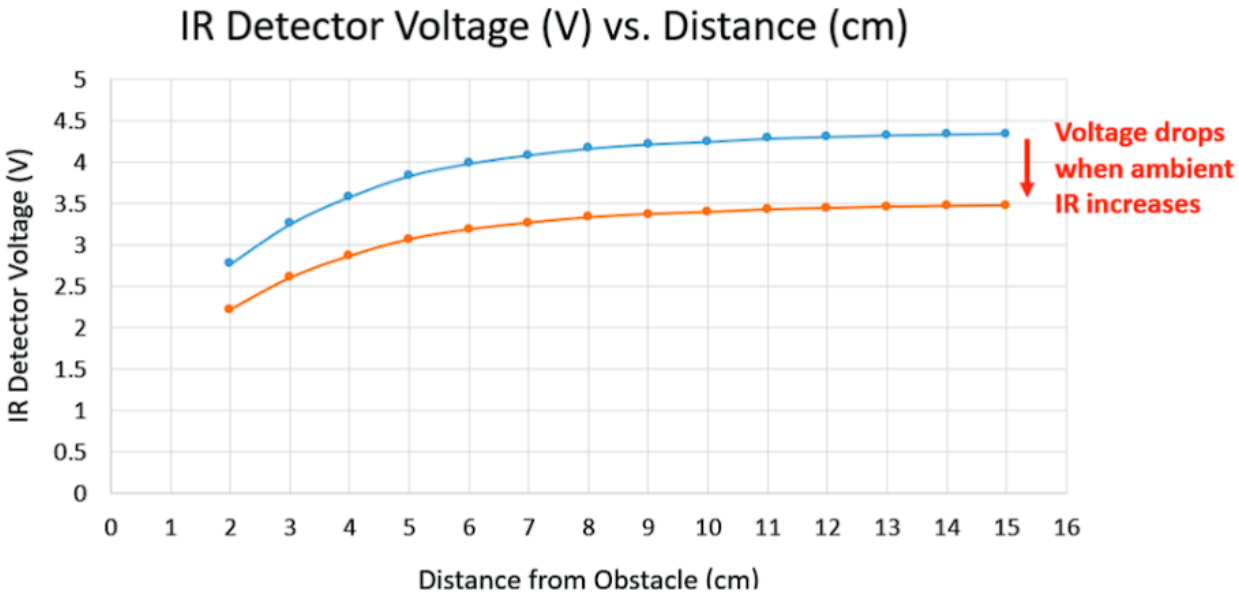


Fig. 6.5: Changes in IR detector voltage when ambient IR increases (from project information)

From the multiple test results we obtained, we also observed noticeable fluctuations in the voltage output of the IR sensors, even when the object was kept at a constant distance. This inconsistency indicated that the sensor was affected by changes in ambient light levels in the surrounding environment.

To address this issue, one approach is to momentarily turn off the IR emitter to measure the baseline IR value caused by ambient light. Once the ambient value is recorded, the IR emitter is turned back on to measure the IR reading reflected from the emitter. The final IR value is then calculated by subtracting the emitter value from the ambient baseline value. If the final IR value exceeds a certain threshold, it indicates that the IR reflection from the emitter is too low, suggesting that the mBot is too close to the right wall and needs to adjust to the left. This process is demonstrated in the code below.

```
#include <MeMCore.h>
#define IR 2 // IR input PIN at 2
#define DECODER_PIN_A A0  // Decoder control pin A
#define DECODER_PIN_B A1 // Decoder control pin B
int ir_emitter;
int count = 0;
int ir_base;
int ir_final;
uint8_t turnSpeed = 100;
MeDCMotor leftMotor(M1);
MeDCMotor rightMotor(M2);
void setup() {
  digitalWrite(A0,LOW);
  digitalWrite(A1,LOW);
  Serial.begin(9600);
}
void record_baseline() {
  if (count == 0){
  digitalWrite(A1,HIGH); // this turns off the IR emitter
  delay(50); // Short delay to stabilize reading
  ir_base = analogRead(A2); // Measure the baseline value (ambient light)
  digitalWrite(A1,LOW);// Turn the IR emitter back on
  Serial.print("Updated IR Baseline Value: ");
  Serial.println(ir_base);
  }
  else if (count == 10){
    count = 0;
  }
  else{
    count ++;
  }
}
```

```
33  void adjustLeft(){
34    rightMotor.run(200);
35    leftMotor.run(-turnSpeed+20);
36    delay(120);
37  }
38  void loop() {
39    record_baseline();
40    ir_emitter = analogRead(A2);
41    ir_final = ir_base - ir_emitter;
42    if (ir_final >=380) // too close to the right wall
43    {
44      adjustLeft();
45    }
46  }
```

The function `record_baseline()` above records the base IR value by turning off the IR emitter. After every 10 iterations, the base IR value will be updated to account for the variation in ambient light. Therefore, constantly updating the baseline IR value will effectively improve the IR sensor's ability to detect objects.

6.5.2 Radius of Turning

As we shifted the breadboard closer to the front of the mBot body to ensure proper adjustment, this inevitably increased the turning radius, making it more likely for the mBot to hit the wall, especially during U-turns. During our trial runs, the mBot collided with the left wall multiple times while executing a U-turn. One possible solution is to change the orientation of the breadboard. Instead of mounting it horizontally, we could position it vertically, which would reduce the turning radius and significantly decrease the chances of the mBot hitting the wall during a U-turn. Another solution is that instead of shifting the breadboard, we can shift the position of the IR sensors forward. This would allow for more accurate and timely corrections during the turns, potentially mitigating the risk of collisions without altering the breadboard's placement.

7. Colour Sensors

In this chapter, we will explain the principles behind our colour sensors on the mBot. Colours can be defined to be a combination of the three primary colours: Red, Green, and Blue (RGB). Each RGB is denoted by a number ranging from 0 to 255. Varying the values of RGB gives us different colours, allowing us to perceive the visible light spectrum.

It turns out the mBot can ‘see’ colours by using a circuit containing one LDR and three LEDs corresponding to RGB. When RGB light is shone sequentially on a coloured paper, the paper absorbs and reflects light. The reflected light affects the resistance of the LDR, changing the voltage across the LDR component. This change is measured and translated into RGB values, allowing the mBot to determine the colour of the paper. A corresponding action will be taken based on the colour.

7.1 Circuit Implementation

The colour sensor circuit performs two functions: The mBot core will output a 5V voltage to light up each RGB LED and the analog input will measure the intensity of each light being reflected onto the LDR. To perform these functions, we require analog pins to control LEDs and measure reflected light intensities.

Refer to Figure 7.1 below for the Tinkercad circuit diagram of the colour sensor circuit. The circuit consists of an HD74LS139P 2-to-4 decoder IC chip, one 5-100kΩ LDR, Red/Blue/Green LEDs and three corresponding resistors. The Red LED is connected in series to a 390Ω Resistor, the Green LED is connected in series to a 470Ω Resistor and the Blue LED is connected in series to a 470Ω Resistor. The LDR is connected in series to a 9.1kΩ Resistor. The four branches are connected in parallel to each other. The 3 RGB LEDs are mounted equidistant from the LDR. The positive terminals of the 3 LEDs and LDR are connected to a 5V power supply. We used a CD4511 chip to represent the 2-4 decoder IC chip. The A0, A1, and A3 terminals correspond to the respective terminals on the RJ25 adaptors. The LDR is connected to A3 terminal, which reads the voltage against the ground and converts it into a raw RGB value.

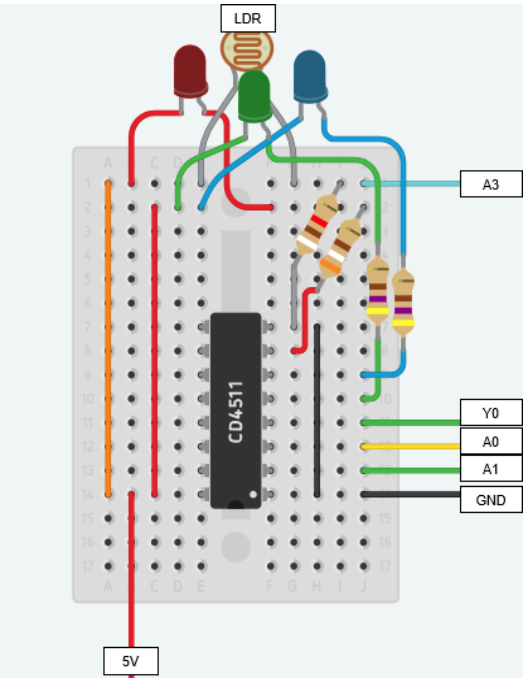


Fig. 7.1: TinkerCAD Circuit Diagram of LDR

RGB LED control is achieved via the 2-4 decoder IC chip (Refer to Table 7.2 and Figure 7.3 below). The logic behind the 2-4 decoder IC chip makes use of two binary inputs (LOW, HIGH) and returns one of 4 possible output scenarios. By varying the two inputs, the 3 RGB LEDs and the IR emitter can be turned on and off. The chip is connected to the analog output (A0, A1), 5V power supply (Vcc) and ground (GND) pins via an RJ25 adapter connected to Port 4 of the mBot Core.

Pins 1 through 8 and 16 are used. Vcc (Pin 16) is connected to a 5V power supply to power the chip. Enable 1G (Pin 1) is connected to GND to set it to logic LOW. The four data outputs Y0 - Y3 (Pins 4 through 7) are connected to the negative terminals of IR Emitter, Green, Blue and Red LEDs respectively. Select Inputs 1A and 1B (Pins 2 and 3) are respectively connected to analog output pins A1 and A0 from mBot Core.

Inputs			Outputs				
Enable	Select						
1G	1A	1B	Y0	Y1	Y2	Y3	Effect
L	L	L	L	H	H	H	Infrared Emitter turns on
L	H	H	H	H	H	L	Red LED turns on
L	L	H	H	L	H	H	Green LED turns on
L	H	L	H	H	L	H	Blue LED turns on

L: Low H: High

Table 7.2: Logic of the 2-to-4 Decoder

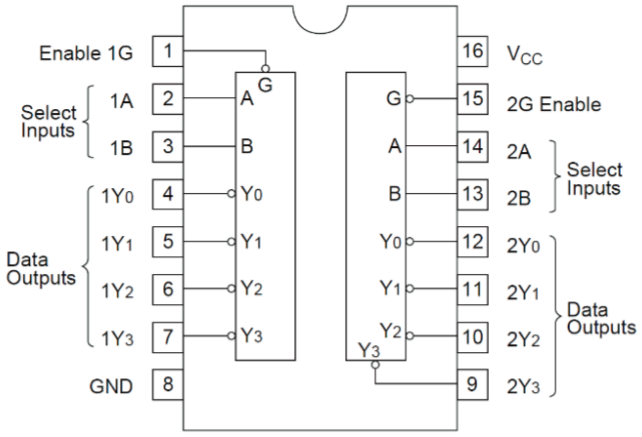


Fig. 7.3: Pin Layout of HD74LS139P 2-to-4 Decoder IC Chip

7.2 Calibration of Colour Sensors

To start the calibration of colour sensors, we first need to convert each of the raw RGB values into the standard range of 0 to 255. The raw values range from 377 to 915, which is unusable in its current state. We used the normalisation formula taught in Studio 12. Below is the code implementation of the formula:

```
colourArray[c] = (colourArray[c] - blackArray[c]) / greyDiff[c] * 255;
```

We need a set of values to use the formula. To start, we obtain a set of whiteArray values and blackArray values (which corresponds to the raw RGB values when a white and a black piece of paper is used respectively). greyDiff is obtained by using whiteArray - blackArray for each RGB.

Code implementation is as such:

```
1  //for calibration of LDR at the start(Disabled for Final Run)
2  void setBalance() {
3      Serial.println("Place White Sample for Calibration...");
4      delay(5000); // Wait for white sample
5
6      // Scan white sample
7      for (int i = 0; i <= 2; i++) {
8          selectLED(i);
9          delay(RGBWait);
10         whiteArray[i] = getAvgReading(5);
11         Serial.println(whiteArray[i]);
12         delay(RGBWait);
13     }
14     offLED();
15
16     Serial.println("Place Black Sample for Calibration...");
17     delay(5000); // Wait for black sample
18
19     // Scan black sample
20     for (int i = 0; i <= 2; i++) {
21         selectLED(i);
22         delay(RGBWait);
23         blackArray[i] = getAvgReading(5);
24         Serial.println(blackArray[i]);
25         greyDiff[i] = whiteArray[i] - blackArray[i];
26         delay(RGBWait);
27     }
28     offLED();
29     Serial.println("Calibration Completed. Ready for Color Detection.");
30     delay(5000);
31 }
```

We implemented it using a `setBalance()` function to obtain the raw RGB of whiteArray and blackArray, which is used to calculate greyDiff.

We also chose to get an average reading of the colour array values using the `getAvgReading(int times)` function. This helps to smooth variations and reduce the effect of random noise. Components like the LDR can sometimes be noisy, especially when the LDR's resistance is sensitive to fluctuating light conditions (e.g. ambient light changes). Taking an average over multiple readings minimises this by averaging out the effects of random fluctuations, giving a more precise and stable output. We can thus obtain reliable values of RGB. Below is the implementation of the code.

```
33 //LDR gets analog reading of the amount of light reflected
34 int getAvgReading(int times) {
35     int total = 0;
36     for (int i = 0; i < times; i++) {
37         total += analogRead(LDR);
38         delay(LDRWait);
39     }
40     return total / times;
41 }
```

We will then store the values in their respective arrays. The reason for storing the raw values in an array is because, with the current implementation, every time we turn on the mBot, the `setBalance()` function will run and we will have to re-obtain the White, Black, and GreyDiff values. This is not feasible during the graded run as our mBot has to be able to run immediately after it is turned on. Thus, before the start of every lab/graded, we will first calibrate the values using our `setBalance()` function, and then store these calibrated values for future calculations, which is also why as shown in the previous picture, our `setBalance()` function will then be commented out of the code. Below is the calibrated whiteArray, blackArray and greyDiff array used for our Final Run.

```
45  // Placeholders for color detection (Final Test Run values)
46  float colourArray[] = {0, 0, 0}; // Detected color intensities
47  float whiteArray[] = {907, 906, 768}; //{887, 915, 763};  // White calibration values
48  float blackArray[] = {771,672,430};//{791, 729, 377};  // Black calibration values
49  float greyDiff[] = {136, 234, 338} ; //{96, 186, 386};    // Range (white - black)
```

Now, with the black, white Array and greyDiff values obtained, we can finally start calibrating the actual colours that our mBot will need to eventually detect. We used a function called `detectColor()` to aid with the detection and calibration of the colours. The first part of the code is as such.

```
67  //detect color of paper and return index of color detected
68  int detectColor() {
69      for (int c = 0; c <= 2; c++) {
70          selectLED(c); // on the selected LED
71          delay(RGBWait); // Stabilize LDR
72          // Average reading for current LED color
73          colourArray[c] = getAvgReading(5);
74          colourArray[c] = (colourArray[c] - blackArray[c]) / greyDiff[c] * 255;
75          //debugging purposes
76          Serial.print(blackArray[c]);
77          Serial.print(whiteArray[c]);
78          Serial.print(greyDiff[c]);
79          Serial.print("Intensity for ");
80          Serial.print(colorNames[c]);
81          Serial.print(": ");
82          Serial.println(int(colourArray[c]));
83          delay(RGBWait);
84          offLED();
85      }
```

We will first loop through the 3 primary colours, Red, Green, and Blue with a delay of 200ms between each colour. For each RGB value, we will get their individual averaged value and store it inside the variable `colourArray[c]`, we will then calculate its normalised value using the normalisation formula and store it back inside `colourArray[c]`, we will then print out the value of

this normalised `colourArray[c]` value for each of the R, G, B for debugging and calibration purposes using the `Serial.print()` function.
After we calibrate the RGB values for each of the colours we need to detect, we then manually store these values into a `calibratedColors[6][3]` array which will be used in our detection algorithm later.

```
// check calibration on the test day (Final Test Run values)
int calibratedColors[6][3] = {
  {132, 82, 77},//Before Callibration: {155, 84, 103},  // Red
  {55, 198, 136},//{45, 196, 151},   // Green
  {150, 152, 95}, //{168, 145, 108},  // Orange
  {152, 209, 198}, //{168, 209, 220},  //{360, 235, 236},  // Pink
  {51, 191, 201}, //{45, 191, 218}, //Blue
  {162, 245, 245}, //white
};
```

7.3 Implementation of Colour Sensors

Now, after we have calibrated the colours, we can move on to the colour detection part of the code. Below is the implementation of the first part of the code, which employs the same `detectColor()` function.

```
//detect color of paper and return index of color detected
int detectColor() {
    for (int c = 0; c <= 2; c++) {
        selectLED(c); // on the selected LED
        delay(RGBWait); // Stabilize LDR
        // Average reading for current LED color
        colourArray[c] = getAvgReading(5);
        colourArray[c] = (colourArray[c] - blackArray[c]) / greyDiff[c] * 255;
        //debugging purposes
        Serial.print(blackArray[c]);
        Serial.print(whiteArray[c]);
        Serial.print(greyDiff[c]);
        Serial.print("Intensity for ");
        Serial.print(colorNames[c]);
        Serial.print(": ");
        Serial.println(int(colourArray[c]));
        delay(RGBWait);
        offLED();
    }
```

First, for reading in the RGB values. When our mBot reaches a coloured paper, our line sensor will detect the black line and our mBot will stop. It will then use the `selectLED()` function to activate each of the RGB LEDs in sequence. Once the light is shone, some light will be absorbed while others will be reflected by the paper. The intensity of the reflected wavelength is dependent on the colour of the paper. The reflected wavelength is incident to the LDR, affecting the LDR resistance. This change in resistance will change the voltage across the LDR component. The voltage measured at the A3

terminal (see Figure 7.1) is with respect to the ground, which changes due to the reflected light. The mCore reads the analog input and returns a raw RGB value ranging from 0 to 1023, representing 0V to 5V. These RGB values will be stored in the colourArray array, we will use the getAvgReading function to get an average of the detected RGB values as explained above. These RGB will then be normalised to the standard range of 0 to 255 using the normalisation formula as explained above.

Now. moving on to the part of the code to identify the colour detected.

```
    int minDifference = 1000;  // Arbitrary large number
    int colorIndex = -1;
   //Manhattan Distance: double squaredDetectColor =
   //sqrt(colourArray[0]*colourArray[0] + colourArray[1]*colourArray[1] + colourArray[2]*colourArray[2]);

    // Find closest match
    for (int i = 0; i < 6; i++) {
        //Manhattan Distance: double diff = fabs(squaredDetectColor - squaredColors[i]);
        int diff = abs(colourArray[0] - calibratedColors[i][0]) +
                   abs(colourArray[1] - calibratedColors[i][1]) +
                   abs(colourArray[2] - calibratedColors[i][2]);

        if (diff < minDifference) {
            minDifference = diff;
            colorIndex = i;
        }
    }
    //Index of the closest matching color
    if (colorIndex != -1) {
        Serial.print("Detected Color: ");//Debug on Serial Monitor
        Serial.println(int(colorIndex));
        Serial.println(colorNames[colorIndex]);
        led.setColor(displayLED[colorIndex][0], displayLED[colorIndex][1], displayLED[colorIndex][2]);//Visual debug
        led.show();
    }
    return colorIndex;
    //Return the index of the color detected
}
```

We chose to use a simple yet effective algorithm known as the Manhattan distance to aid us with colour detection. Manhattan distance is a metric used to determine the distance between two points in a grid-like path. Manhattan distance measures the sum of the absolute differences between the coordinates of the points. Mathematically, the Manhattan distance between two points in an n-dimensional space is the sum of the absolute differences of their Cartesian coordinates. In the case of a 3-D dimension, like in our case, the formula is Distance = $|x2-x1| + |y2-y1| + |z2-z1|$. The variables x2, y2, and z2 are the RGB values of the coloured paper stored in colourArray, while x1, y1, and z1 are RGB values respectively of each of the 6 colours stored in the `calibratedColors[6][3]` array. We will then loop through and calculate the Manhattan distance between the RGB values of the coloured paper and the calibrated RGB values of each of the 6 colours. By finding the Manhattan distances with reference to the 6 colours, the colour with the smallest Manhattan distance will be determined as the actual colour of the coloured paper. The action associated with the colour will then be executed by the mBot.

We then use the `led.setColor()` and `led.show()` functions built into the mBot to display the colour detected by our sensors. This helps us debug the sensor in situations where it detects the wrong colour. By knowing which incorrect colour was detected, we can more effectively identify and resolve issues with colour recognition, especially when colours are being mixed up. Additionally, it adds a touch of style.

7.4 Mounting of Colour Sensor

We mounted the colour sensors in the empty space underneath the mBot. To minimise ambient light interference, we constructed a black skirting around our colour sensor breadboard with a chimney for the LDR and colour sensors to protrude out. We subsequently added 4 flaps at the chimney area to further block out ambient light. We also ensured that all our LEDs were below the LDR so that the light from the LED would not be captured by the LDR.

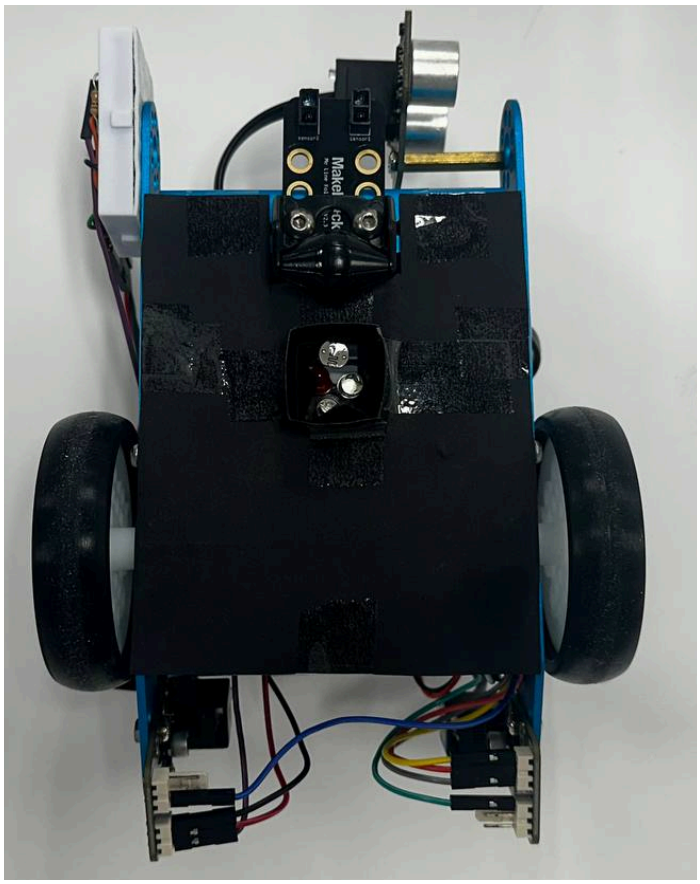


Fig. 7.4: Finalised Position of the LDR Colour Sensor

7.5 Challenges Faced with Colour Sensor

The greatest challenge that we faced while working with our colour sensor, as one would expect is with the consistency of our colour sensor. Actually, in fact we are very lucky that our colour sensor worked most of the time. However, there were still several instances where it failed and we want to minimise that. After going through the debugging process, i.e. looking through the values of the detected RGB values and which corresponding colour our algorithm detects. We found 2 main problems.

Firstly, our raw RGB range was too small, causing the calibrated RGB values to be quite close to each other for some colours, thus causing our algorithm to detect the wrong colour.

To expand the raw RGB range of our colour sensor, we selected a resistor with a lower value to connect in series with the LDR. This causes the voltage changes across the LDR to become more significant, making the sensor more responsive to variations in colour. As a result, the difference between the detected RGB values and the calibrated values for each colour is larger. Since our algorithm works by calculating the Manhattan distance between the detected and calibrated RGB values, a larger difference will lead to a larger difference in Manhattan distance between the colours. This reduces the chances of ambiguous cases, where the Manhattan distance of 2 colours is too close to each other, which may lead to a situation where the wrong colour is detected. This will then minimise errors in colour detection. We changed our resistor value from 100kΩ which was the one used in the studio, to 9.1kΩ.

Second is that the ambient condition is always changing, e.g brightness of the light, and shadow. There was even one time when our mBot kept detecting a specific green paper wrongly, after closer inspection, we realised that there was a little white spot on the paper(probably from a piece of paper) which affected our colour detection. After removing that piece of paper, our colour detection proceeded to work nicely.

There is nothing we can do about the ambient condition, the only thing we can do is to minimise the impact of it. Thus, our solution is to do a round of calibration of colours before the start of every lab session and the graded course, this is to calibrate our colour sensor to the conditions of that day. This can be seen from the various commented RGB values beside our colour array.

8. Work Distribution

Name	Work
Tan Chun Liang	Director of LED and LDR Director of Design for Black Skirting
Wang Chuhao	Director of LED and LDR Director of Heavy-Ass Laptop and Code Modification
Wang Zaixi	Director of Ultrasonic and IR Director of Calibration Code Director of Safekeeping the mBot
Wang Jiawei	Director of Ultrasonic and IR Director of Project Report

Table 8: Work Distribution

APPENDIX

Appendix A (mBot_Main Code)

```
#include <MeMCore.h>

//Define time delays for LDR, values are used in LDR.ino
#define RGBWait 200 //in ms
#define LDRWait 10  //in ms

//Define time delays for Motor, values are used in motor.ino
#define SPEED 255 //speed between 0 - 255
#define FORWARD_DELAY 100 //in ms
#define TURN_DELAY_right 320//in ms //Before callibration: 335
#define TURN_DELAY_left 310 //in ms //Before callibration: 300, 320
#define ONEBOX_DELAY_blue 640//in ms //Before callibration: 610
#define ONEBOX_DELAY_pink 660 //in ms
#define UTURN_DELAY 290 //in ms

//Define pins
#define DECODER_PIN_A A0  //Decoder control pin A
#define DECODER_PIN_B A1 //Decoder control pin B
#define IR A2 //IR input PIN at A2
#define LDR A3 //LDR sensor pin at A3
#define PUSH_BUTTON A7

//Define some global variables, variables are used inside the main loop
int ir_value;
float ultradistance;

//Define some motor speed because the mBot doesn't move in a straightline
uint8_t speed_right = 215; //Before callibration: 215, 245,255 //235, 225, 215
uint8_t speed_left = 235;  //Before callibration: 200 original 235
uint8_t turnSpeed = 100; //Before callibration: 100, 80,75

//Default mBot pin declaration
MeDCMotor leftMotor(M1);
MeDCMotor rightMotor(M2);
MeRGBLed led(0, 30); // set up LED pin
MeLineFollower lineFinder(PORT_1); // assigning Line Follower to port 1
MeUltrasonicSensor ultraSensor(PORT_2); // assigning UR to port 2
MeBuzzer buzzer;

// Placeholders for color detection (Final Test Run values)
float colourArray[] = {0, 0, 0}; // Detected color intensities
float whiteArray[] = {907, 906, 768}; //{887, 915, 763};  // White calibration values
float blackArray[] = {771,672,430};//{791, 729, 377};  // Black calibration values
```

```
float greyDiff[] = {136, 234, 338} ; //{96, 186, 386};    // Range (white - black)

// check calibration on the test day (Final Test Run values)
int calibratedColors[6][3] = {
  {132, 82, 77},//Before Callibration: {155, 84, 103},  // Red
  {55, 198, 136},//{45, 196, 151},   // Green
  {150, 152, 95}, //{168, 145, 108},  // Orange
  {152, 209, 198}, //{168, 209, 220},  //{360, 235, 236},  // Pink
  {51, 191, 201}, //{45, 191, 218}, //Blue
  {162, 245, 245}, //white
};

/*squared RGB, sqrt. (Manhattan Distance for improvement)
//Did not Implement this for Final Test Run
double squaredColors[6] = {
    195.369,//{155, 84, 103},  // Red
    266.207,//{45, 196, 151},   // Green
    256.144, //{168, 145, 108},  // Orange
    359.369, //{168, 209, 220},  //{360, 235, 236},  // Pink
    308.146, //{45, 191, 218}, //Blue
    382.484, //white
};*/

// Display colors for RGB LED, for debugging purposes
int displayLED[6][3] = {
    {255, 0, 0},     // Red
    {0, 255, 0},     // Green
    {255, 50, 0},    // Orange
    {128, 0, 128},   // Pink
    {0, 0, 255},      // Blue
    {255,255,255},  //white
};

// Color names for serial output for debug
char* colorNames[] = {"Red", "Green", "Orange", "Pink", "Blue","white"};

//Action executed after LDR decodes the colour of the paper, function inside motor.ino
void action() {
  int color = detectColor(); // Each LDR number corresponds to a specific colour detected.
  if (color == 0) { // Red
    turnLeft();
  } else if (color == 1) { // Green
    turnRight();
```

```
  } else if (color == 2) { // Orange
    uTurn();
  } else if (color == 3) { // Purple
    twoLeftTurn();
  } else if (color == 4) { // Blue
    twoRightTurn();
  }
   else { //White
    celebrate();
    stop_indef();
  }
}

void setup() {
    pinMode(PUSH_BUTTON, INPUT);
    pinMode(DECODER_PIN_A, OUTPUT);
    pinMode(DECODER_PIN_B, OUTPUT);
    led.setpin(13); // Set the LED pin
    pinMode(A2, INPUT);
    pinMode(A0, OUTPUT);
    pinMode(A1, OUTPUT);
    Serial.begin(9600); // Begin serial communication
    //setBalance(); // Calibrate LDR with black and white(Disabled after callibration)
    digitalWrite(A0,LOW);
    digitalWrite(A1,LOW);
}

void celebrate()
{
 // Star Wars Main Theme (simplified)
  led.setColor(displayLED[0][0], displayLED[0][1], displayLED[0][2]);
  led.show();
  // Note: G
  buzzer.tone(392, 500);  // G4 for 500 ms
  delay(100);
  led.setColor(displayLED[0][0], displayLED[0][1], displayLED[0][2]);
  led.show();

  // Note: G
  buzzer.tone(392, 500);  // G4 for 500 ms
  delay(100);
  led.setColor(displayLED[0][0], displayLED[0][1], displayLED[0][2]);
  led.show();
```

```
135
136     // Note: G
137     buzzer.tone(392, 500);  // G4 for 500 ms
138     delay(100);
139     led.setColor(displayLED[0][0], displayLED[0][1], displayLED[0][2]);
140     led.show();
141
142     // Note: Eb
143     buzzer.tone(311, 350);  // Eb4 for 350 ms
144     led.setColor(displayLED[1][0], displayLED[1][1], displayLED[1][2]);
145     led.show();
146
147     // Note: Bb
148     buzzer.tone(466, 150);  // Bb4 for 150 ms
149     led.setColor(displayLED[2][0], displayLED[2][1], displayLED[2][2]);
150     led.show();
151
152     // Note: G
153     buzzer.tone(392, 500);  // G4 for 500 ms
154     led.setColor(displayLED[0][0], displayLED[0][1], displayLED[0][2]);
155     led.show();
156
157     // Note: Eb
158     buzzer.tone(311, 350);  // Eb4 for 350 ms
159     led.setColor(displayLED[1][0], displayLED[1][1], displayLED[1][2]);
160     led.show();
161
162     // Note: Bb
163     buzzer.tone(466, 150);  // Bb4 for 150 ms
164     led.setColor(displayLED[2][0], displayLED[2][1], displayLED[2][2]);
165     led.show();
166
167     // Note: G
168     buzzer.tone(392, 1000); // G4 for 1000 ms
169     led.setColor(displayLED[0][0], displayLED[0][1], displayLED[0][2]);
170     led.show();
171     delay(1100);
172   }
173
174
175   void loop() {
176     ir_value = analogRead(A2);
177     ultradistance = ultraSensor.distanceCm();
```

```
178     forward();
179     Serial.println(ir_value);
180     if (ultradistance <= 8) // too close to the left wall. Before Callibration: 10
181     {
182       adjustRight();
183     }
184     if (ir_value <= 225) // too close to the right wall. Before Callibration: 150, 170,180,200,250,275,
185     {
186       adjustLeft();
187     }
188     else{
189       forward();
190     }
191     int sensorState = lineFinder.readSensors();
192     if (sensorState == S1_IN_S2_IN) { //inside black line, LDR operates and starts action
193     stop(10);
194     action();
195     }
196   }
```

Appendix B (LDR Code)

```
//for calibration of LDR at the start(Disabled for Final Run)
void setBalance() {
    Serial.println("Place White Sample for Calibration...");
    delay(5000); // Wait for white sample

    // Scan white sample
    for (int i = 0; i <= 2; i++) {
        selectLED(i);
        delay(RGBWait);
        whiteArray[i] = getAvgReading(5);
        Serial.println(whiteArray[i]);
        delay(RGBWait);
    }
    offLED();

    Serial.println("Place Black Sample for Calibration...");
    delay(5000); // Wait for black sample

    // Scan black sample
    for (int i = 0; i <= 2; i++) {
        selectLED(i);
        delay(RGBWait);
        blackArray[i] = getAvgReading(5);
        Serial.println(blackArray[i]);
        greyDiff[i] = whiteArray[i] - blackArray[i];
        delay(RGBWait);
    }
    offLED();
    Serial.println("Calibration Completed. Ready for Color Detection.");
    delay(5000);
}

//LDR gets analog reading of the amount of light reflected
int getAvgReading(int times) {
    int total = 0;
    for (int i = 0; i < times; i++) {
        total += analogRead(LDR);
        delay(LDRWait);
    }
    return total / times;
}

//function that turns on selected LED color
```

```
44  void selectLED(int color) {
45      switch (color) {
46          case 0: // Red
47              digitalWrite(DECODER_PIN_A, HIGH);
48              digitalWrite(DECODER_PIN_B, HIGH);
49              break;
50          case 1: // Green
51              digitalWrite(DECODER_PIN_A, LOW);
52              digitalWrite(DECODER_PIN_B, HIGH);
53              break;
54          case 2: // Blue
55              digitalWrite(DECODER_PIN_A, HIGH);
56              digitalWrite(DECODER_PIN_B, LOW);
57              break;
58      }
59  }
60
61  //turn off LED and on IR
62  void offLED() {
63    digitalWrite(DECODER_PIN_A, LOW);
64    digitalWrite(DECODER_PIN_B, LOW);
65  }
66
67  //detect color of paper and return index of color detected
68  int detectColor() {
69      for (int c = 0; c <= 2; c++) {
70          selectLED(c); // on the selected LED
71          delay(RGBWait); // Stabilize LDR
72          // Average reading for current LED color
73          colourArray[c] = getAvgReading(5);
74          colourArray[c] = (colourArray[c] - blackArray[c]) / greyDiff[c] * 255;
75          //debugging purposes
76          Serial.print(blackArray[c]);
77          Serial.print(whiteArray[c]);
78          Serial.print(greyDiff[c]);
79          Serial.print("Intensity for ");
80          Serial.print(colorNames[c]);
81          Serial.print(": ");
82          Serial.println(int(colourArray[c]));
83          delay(RGBWait);
84          offLED();
85      }
86
```

```
87      int minDifference = 1000;  // Arbitrary large number
88      int colorIndex = -1;
89     //Manhattan Distance: double squaredDetectColor =
90     //sqrt(colourArray[0]*colourArray[0] + colourArray[1]*colourArray[1] + colourArray[2]*colourArray[2]);
91
92      // Find closest match
93      for (int i = 0; i < 6; i++) {
94          //Manhattan Distance: double diff = fabs(squaredDetectColor - squaredColors[i]);
95          int diff = abs(colourArray[0] - calibratedColors[i][0]) +
96                     abs(colourArray[1] - calibratedColors[i][1]) +
97                     abs(colourArray[2] - calibratedColors[i][2]);
98
99          if (diff < minDifference) {
100             minDifference = diff;
101             colorIndex = i;
102         }
103     }
104     //Index of the closest matching color
105     if (colorIndex != -1) {
106         Serial.print("Detected Color: ");//Debug on Serial Monitor
107         Serial.println(int(colorIndex));
108         Serial.println(colorNames[colorIndex]);
109         led.setColor(displayLED[colorIndex][0], displayLED[colorIndex][1], displayLED[colorIndex][2]);//Vis
110         led.show();
111     }
112     return colorIndex;
113     //Return the index of the color detected
114 }
```

Appendix C (Motor Code)

```
void forward(){
  rightMotor.run(speed_right);
  leftMotor.run(-speed_left);
}
//Turn Functions for Obstacle Avoidance
void adjustRight(){
  leftMotor.run(-200);
  rightMotor.run(turnSpeed);
  delay(165); //200 175
}
void adjustLeft(){
  rightMotor.run(200);
  leftMotor.run(-turnSpeed+20);
  delay(120); //200 175 165 150,145
}
void stop(int time) {
  leftMotor.stop();
  rightMotor.stop();
  delay(time);
}
void moveOneBoxblue() {
  stop(FORWARD_DELAY);
  leftMotor.run(-SPEED);
  rightMotor.run(SPEED);
  delay(ONEBOX_DELAY_blue);
  stop(FORWARD_DELAY);
}

void moveOneBoxpink() {
  stop(FORWARD_DELAY);
  leftMotor.run(-SPEED);
  rightMotor.run(SPEED);
  delay(ONEBOX_DELAY_pink);
  stop(FORWARD_DELAY);
}

void turnRight() {
  leftMotor.run(-SPEED); // Left wheel goes anti-clockwise
  rightMotor.run(-SPEED); //right wheel goes anti-clockwise
  delay(TURN_DELAY_right);
}
void turnLeft() {
  leftMotor.run(SPEED);
```

```
    rightMotor.run(SPEED);
    delay(TURN_DELAY_left);
  }
void uTurn() {
    turnLeft();
    delay(UTURN_DELAY);
  }
void twoLeftTurn() {
    turnLeft();
    moveOneBoxpink();
    turnLeft();
  }
void twoRightTurn() {
    turnRight();
    moveOneBoxblue();
    turnRight();
  }

  //Indefinite loop
  void stop_indef()
{
    leftMotor.stop();
    rightMotor.stop();
    while(true)
    {
      delay(10);
    }
  }
```