CG1111A mBot Report

Project Group: B01-S1-T1

Group Members:

Aaron Chan Zhi	A0230354R
Amit Rahman	A0244348B
Bi Yuying	A0240016Y
Aung Phone Naing	A0234363J

Table of Contents

1. Overall Algorithm of mBot

This project aims to design an mBot that navigates a maze in the shortest time possible. Within the maze there are obstacles that the mBot must clear autonomously in order to complete the course. These challenges involve taking in IR, ultrasonic and RGB readings from the environment and making decisions based on them.

Figure 1 demonstrates the overall algorithm of the mBot. When the mBot enters the maze, it attempts to move forward in a straight line, maintaining a distance away from the walls to avoid collision. It does so with the help of IR sensors and ultrasonic sensors on either side of the mBot, constantly taking readings of distance from each wall and attempts to maintain a minimum distance away from the walls. Moving too close to either wall results in a negative feedback, where the mBot's algorithm is designed to incrementally make small turns away from the wall till a linear path of travel is re-established.

Upon detecting a black strip of paper on the maze track via the mBot's line sensor module, the mBot stops. It will detect & interpret the colour of the piece of paper below it and then executes one of the five different turns. At the end of the maze, when the mBot detects a black strip with white paper, it is designed to play a celebratory tune.

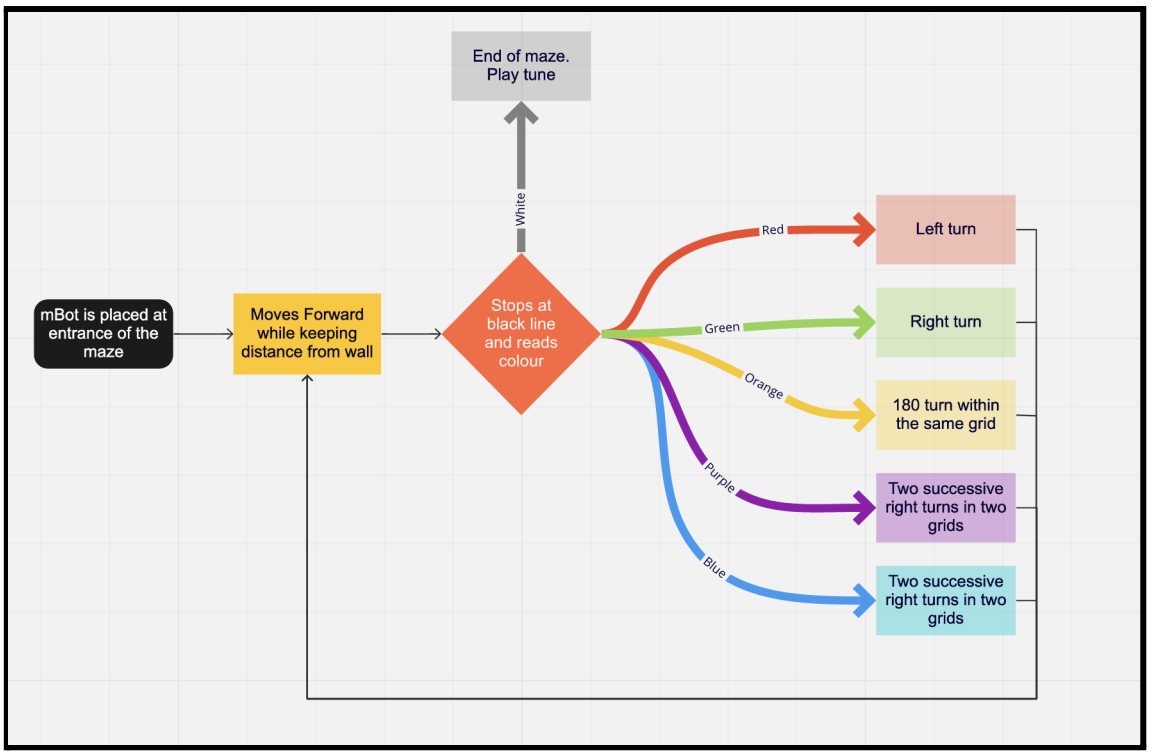


Fig 1

2. Implementation of subsystems in mBot

i. Wall avoidance

To ensure that the mBot will not crash into the walls, we attached IR and ultrasonic sensors to both sides of the mBot. The threshold for the ultrasonic sensors were set to a minimum of 7cm away, and that of the IR sensors were set to 150. When the threshold is crossed on either side, the mBot incrementally turns away until the reading is above the threshold by using a while loop.

For the ultrasonic sensor, while (distance < 7) is proven to be true, the right wheel closer to the wall will be sped up, causing the mBot to turn away from the wall.

For the IR sensor, while (IR_diff >= 150) is proven to be true, the left wheel closer to the wall will be sped up, causing the mBot to turn away from the wall.

ii. Waypoint Challenge: Colour sensing

When the mBot reaches a black strip and colour test, it will detect the colour above the black strip. To do this, the Light Emitting Diode (LED) at the bottom of the mBot will be turned on to display a set of red, green and blue light. The light is shone on the coloured paper below and the amount of light reflected by the coloured paper is then measured by the light sensor of the mBot, enabling the mBot to determine the colour of the paper (Fig 2). In order for the light detected by the light sensor to be accurate, we taped a black box on the 4 sides of the light sensor to ensure that it will not detect light from other sources (Fig 3 & Fig 4).

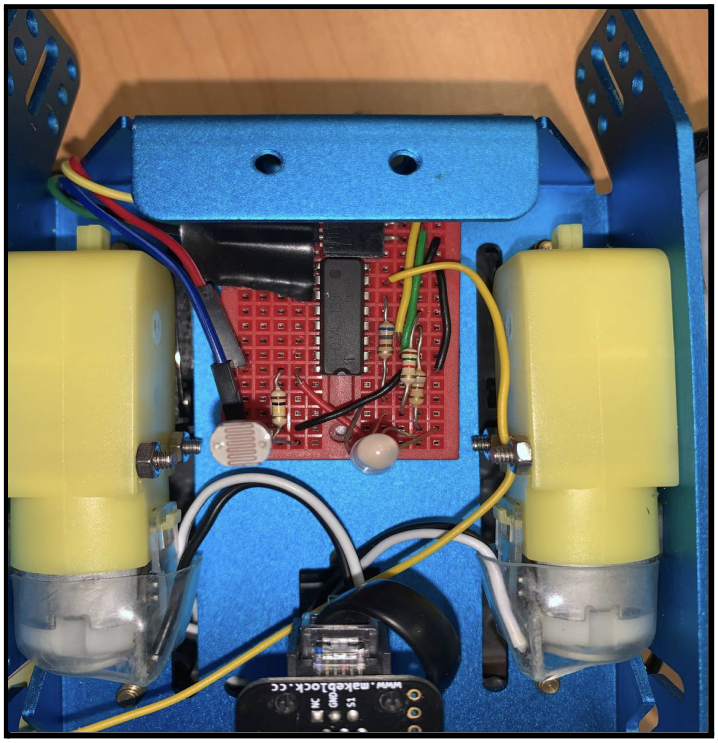


Fig 2

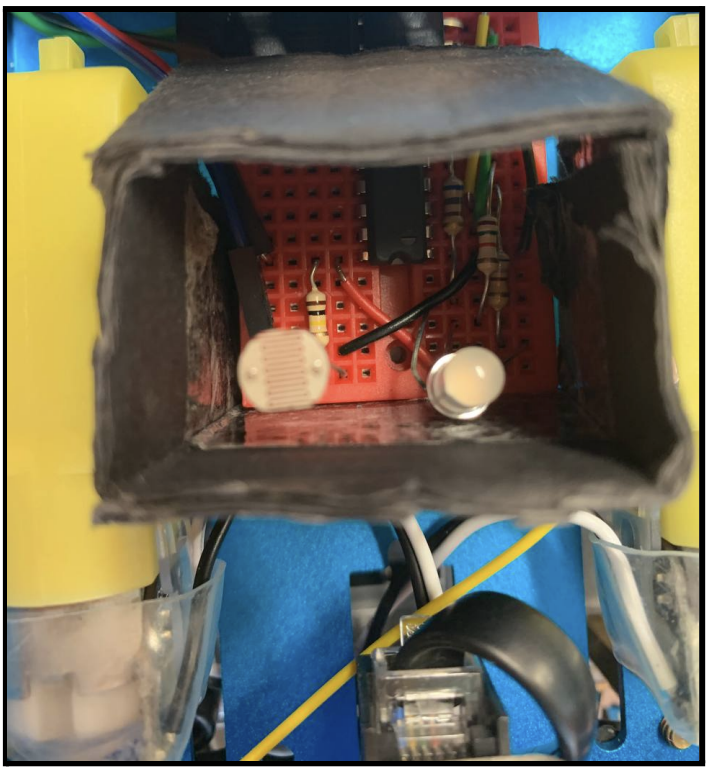


Fig 3

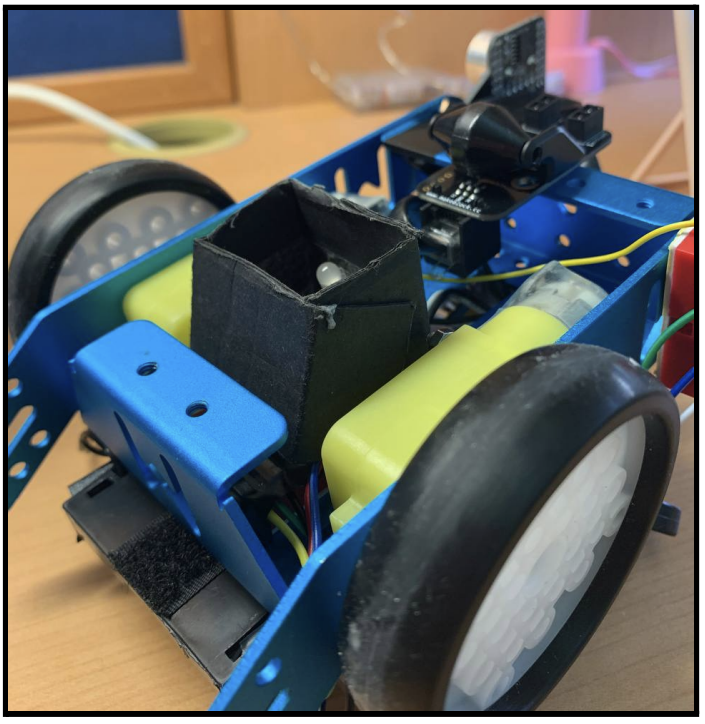


Fig 4

iii. Turning algorithm

At each colour challenge, depending on the colour detected by the colour detection algorithm mentioned above, a turn will be executed. The 5 different types of turns that will be executed can be found in Fig 5, and the corresponding colour interpretation can be found below in Table 1.

Table I: Colour Interpretation for the Colour-sensing Challenge

Colour	Interpretation
Red	Left-turn
Green	Right turn
Orange	180° turn within the same grid
Purple	Two successive left-turns in two grids
Light Blue	Two successive right-turns in two grids

Table 1

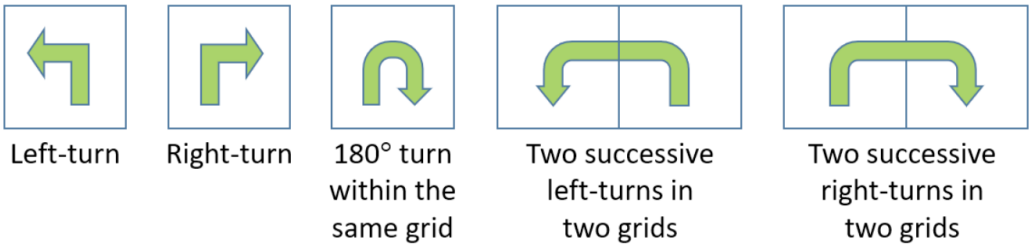


Fig 5

Red signals a 90° left turn while green signals a 90° right turn. For the 90° left turn on red, we set the left motorSpeed to 150, and the right motorSpeed to 100, such that the left wheel moves faster than the right wheel. Similarly, for the 90° right turn on Green, we set the left motorSpeed to -150, and the right motorSpeed to -100, such that the right wheel moves faster than the left wheel. The delay() for the 90° turn is 50ms.

Orange signals a 180° turn within the same grid. The motors would execute 2 consecutive 90° turns in the same direction. We did not add any delay between the 2 consecutive turns in order to ensure that the mBot would turn smoothly. In addition, to give the mBot more space to turn, if the ultrasonic distance sensors detect that the mBot is closer to the right wall, it will perform a 180° left turn to avoid collision. Conversely, if it is closer to the left wall, it will perform a 180° right turn.

Purple signals 2 successive left turns in 2 grids. The motors would run a left turn, followed by a delay of 875ms, followed by another left turn. Light blue signals 2 successive right turns in 2 grids. The motors would run a right turn, followed by a delay of 875ms, followed by another right turn. The delay for both left and right successive turns is set to 50ms.

Additionally, to stabilise the mBot after each turn, we also added a delay of 50ms after the turn is completed.

iv. End-of-maze

At the end of the maze, the colour sensor will detect a piece of white paper. When this happens, the motor will stop running and the celebratory music will be played, and the algorithm will exit the loop.

3. Steps taken for calibration

i. Colour Calibration

Shown below is the algorithm we used to obtain the maximum and minimum RGB values corresponding to the black and white colour samples provided.

```
void setBalance(){
//set white balance
 Serial.println("Put White Sample For Calibration ...");
 delay(5000);           //delay for five seconds for getting sample ready
 LED(3); //Check Indicator OFF during Calibration
//scan the white sample.
//go through one colour at a time, set the maximum reading for each colour -- red, green and blue to the white array
 for(int i = 0;i<=2;i++){
    LED(i);
    delay(RGBWait);
    whiteArray[i] = getAvgReading(5);    //scan 5 times and return the average,
    LED(3);
    delay(RGBWait);
 }
//done scanning white, time for the black sample.
//set black balance
 Serial.println("Put Black Sample For Calibration ...");
 delay(5000);     //delay for five seconds for getting sample ready
//go through one colour at a time, set the minimum reading for red, green and blue to the black array
 for(int i = 0;i<=2;i++){
    LED(i);
    delay(RGBWait);
    blackArray[i] = getAvgReading(5);
    LED(3);
    delay(RGBWait);
//the differnce between the maximum and the minimum gives the range
    greyDiff[i] = whiteArray[i] - blackArray[i];
 }
 //delay another 5 seconds for getting ready colour objects
 Serial.println("Colour Sensor Is Ready.");
 delay(5000);
 }
```

With the white sample placed under the mbot, we coded the LED to flash red, blue, and green to obtain the LDR voltage values corresponding to each colour for the white sample. These values were stored in whiteArray and likewise for the black sample under blackArray. greyDiff was calculated by obtaining the difference in whiteArray and blackArray.

$$currentReading = \frac{analogRead(LED\_sensor) - blackArray[c]}{greyDiff[c] \times 255}$$

Where 'c' is a variable in a loop corresponding to the value for either red, green, or blue

By using this formula, we were then able to obtain RGB readings from the LDR voltage readings. After obtaining the RGB values, we then converted them into Long type using the following equation:

$$LONG = (Blue) + (Green \times 256) + (Red \times 256^2)$$

We then obtained LONG readings for each of the 5 colour samples provided to us as well as for the white sample. We discovered that the LONG readings were different enough to distinguish between samples based on the first 2 digits alone. By taking multiple readings for each colour sample, we obtained a corresponding range of 2 digit values as shown below.

Colour	First 2 digits of LONG reading
Red	<16
Orange	16 - 35
Purple	35 - 49
Blue	50 - 62
Green	50 - 62
White	>62

However, different ambient lighting resulted in readings for certain colours. Hence, we added in additional checks to ensure that the final colour reading was accurate regardless of ambient lighting. This can be seen in the code snippet below:

```
  long LONG = (colourArray[2] + (colourArray[1] * 256) + (256 * 256 * colourArray[0])) / 1000;
  Serial.print("LONG reading : ");
  Serial.println(LONG);
   if (LONG > 62) {
    Serial.println("White");
    return 0;
  } else if (LONG <= 15) {
    Serial.println("Red");
    return 1;
  } else if ((((20 < LONG)  && (LONG <= 49)) && colourArray[2] > blue_threshold) { // Blue is more than threshold point (100)
    Serial.println("Purple");
    return 2;
  } else if ((LONG >= 50) && (colourArray[2] < colourArray[1])) { // Green is more than Blue
    Serial.println("Green");
    return 3;
  } else if ((LONG >= 50) && (colourArray[2] > colourArray[1])) { // Blue is more than Green
    Serial.println("Blue");
    return 4;
  } else if (((16 <= LONG) && (LONG < 40)) && colourArray[2] < blue_threshold) { // Blue is less than threshold point (100)
    Serial.println("Orange");
    return 5;
  } else { // if no colour detected
    Serial.println("???");
    return -1;
  }
}
```

For the confusion between blue and green, there was an overlap between the 2 ranges. Hence to differentiate between the 2 colours, if the blue component of the RGB reading was higher than the green component, the code will read the colour as blue and vice versa. For the confusion between orange and purple, a similar concept was used but instead of comparing blue and green components of RGB readings, we merely checked that the blue component was higher than a certain threshold value. We can do so as orange has a very low blue component while purple has a high blue component. We conducted further tests within the lab at different ambient lightings to ensure that the correct colour was detected consistently.

ii. IR calibration

By taking in 2 readings and the ambient IR and the pulsed IR within a short period and then taking the difference, we were able to reduce the effect of differing ambient IR values on our IR sensor. To ensure that this method would work regardless of location or orientation, we calibrated the IR sensor at different locations throughout the lab until we arrived at a value that enabled the mBot to adjust its course smoothly at varying locations in the lab.

iii. Turning calibration - based on battery

Using trial and error, we obtained the appropriate time required for the mBot to perform a 90 degree turn. However, we realised that the voltage of the batteries greatly affected the accuracy of our turning algorithm. The higher the voltage, the greater the speed of the motors and hence, angle of rotation during each turn increases despite the time spent turning remaining constant. Hence, we made sure to measure the battery voltage after each calibration and performed our calibration within a specific range of battery voltages.

4. Work Division

A. Wall avoidance - Amit
B. Waypoint Challenge: Colour sensing - Aaron
C. Turning algorithm - Phone
D. Calibration and body compactness - Yuying

5. Challenges and Actions taken

i. Avoiding collision with walls during 180° turns

The 180° turns were challenging as the mBot may enter the challenge grid too close to one of the walls due to the insensitivity of the IR sensor not correcting its movement. This may cause the mBot to hit the wall when executing the turn. We attempted a variety of solutions, such as adding a pause while turning instead of simply making a 180° turn straight and partial reversing of the mBot. However, we felt that these methods would greatly increase our timing and make the movements discontinuous. Our final solution was to program the mBot to turn left if it was too close to the right wall, and turn right if it is too close to the left wall based on the ultrasonic sensor’s distance inputs. This is because there would be more space for it to turn to the side it was further from.

ii. Overcoming inconsistent motor performance

Another challenge we faced was the inconsistency in motor output. We observed that the performance of each motor was different, one was turning at a higher speed than the other for the same PWM value. We found out that this was due to the fact that every motor has a different motor constant as we learnt earlier in the course and hence resulting in different parameters for PWM required to match both motors' speeds. Hence we did much trial and error to determine the PWM and the various delays required to execute turns and moving in a straight line in order to get both motors to be in sync.

iii. Improving accuracy of colour detection

In Section 3.3 above, we explained how we calibrated the light sensor values. However, the colour detection was still inaccurate at times. The issue was that for certain colours either the R, G or B values were slightly off which resulted in overlap in colours or another colour entirely. Hence, we decided to use the method of converting the various R, G and B components of a given colour to a 32-bit integer value in which a maximum of 2^{24} (16777216) shades of colours can be represented. The equation is similar to that of calculating the hex value of a colour, RGB LONG = 256 * 256 * R + 256 * G + B. We then conducted further calibration and determined that we are able to use the first two digits of the integer value as the range to separate the 5 colours from one another. This decreased our complexity in colour calibration as we no longer needed to work with comparing 3 different scales and ranges to separate the colour but a single scale was sufficient. Additionally, to ensure consistency in the readings of R, G and B values of any given colour pasted on the floor, we constructed a black “chimney” to block out ambient lighting from influencing the light emitted by the LED lamp when the LDR is reading the values. This can be seen from Figure 6 below.

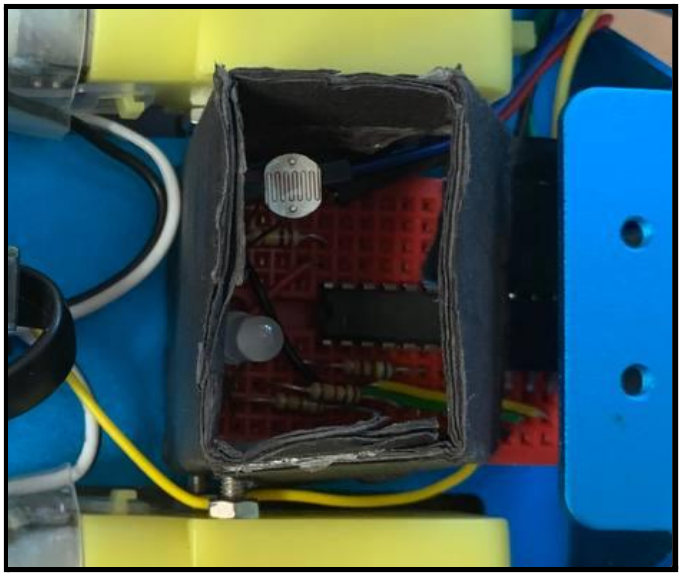


Figure 6: “Chimney” feature over colour sensors to prevent ambient light from affecting readings

iv. Reducing inconsistency in IR readings

As for the IR component of the circuit, we realized that the ambient conditions do greatly affect the readings on the IR sensor affecting its ability to accurately determine distance. Hence we implemented 2 solutions to overcome this issue. First is to use an aluminium shield to make a cover around the IR sensors (Figure 7) to prevent the IR detector from sensing ambient IR from all sides except its frontal area that it is facing. This worked in greatly reducing the inconsistency of the IR readings however, we then realized that the change in IR readings for when the IR emitter is constantly emitting seems to be insignificant for us to calibrate a range. Hence, we attempted to take the difference between the ambient IR reading (i.e. reading from the IR detector when the IR emitter is off) and the IR reading when the IR detector is on. This gave us a more reliable set of readings to calibrate a range out of.

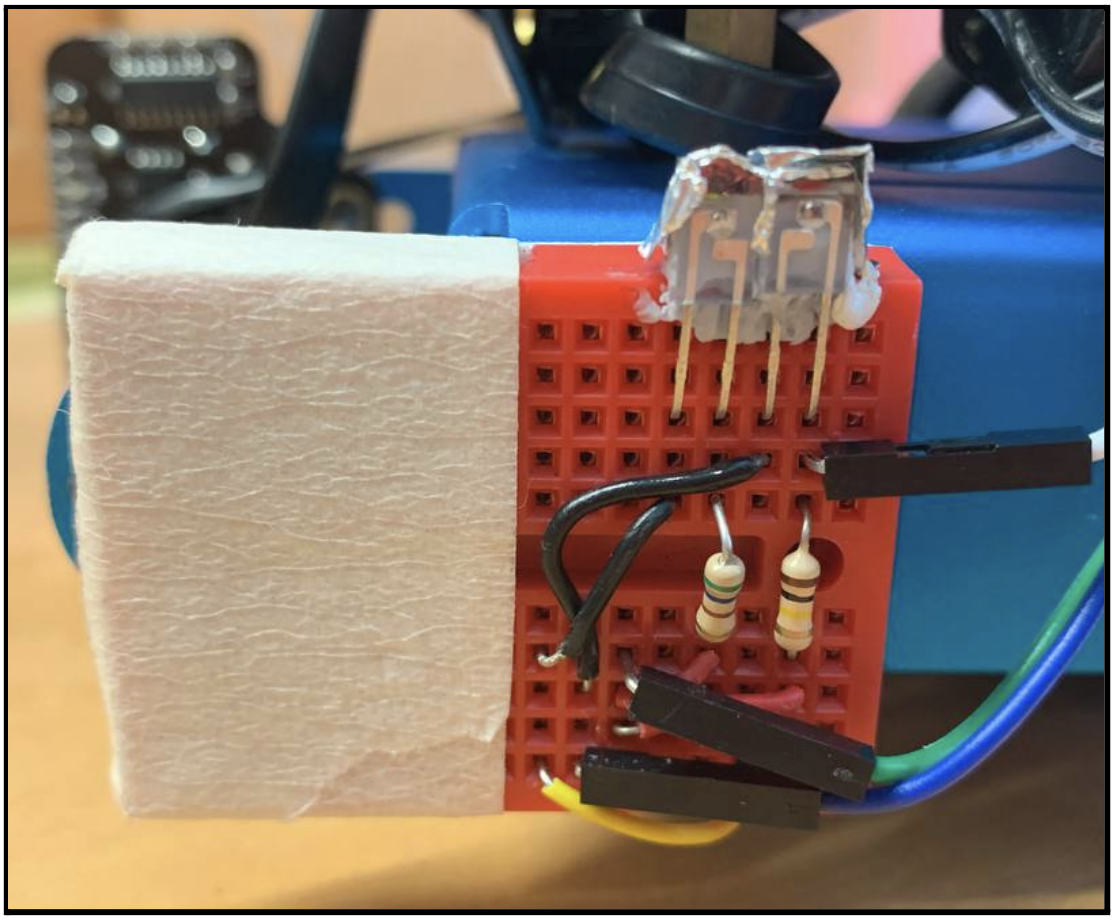
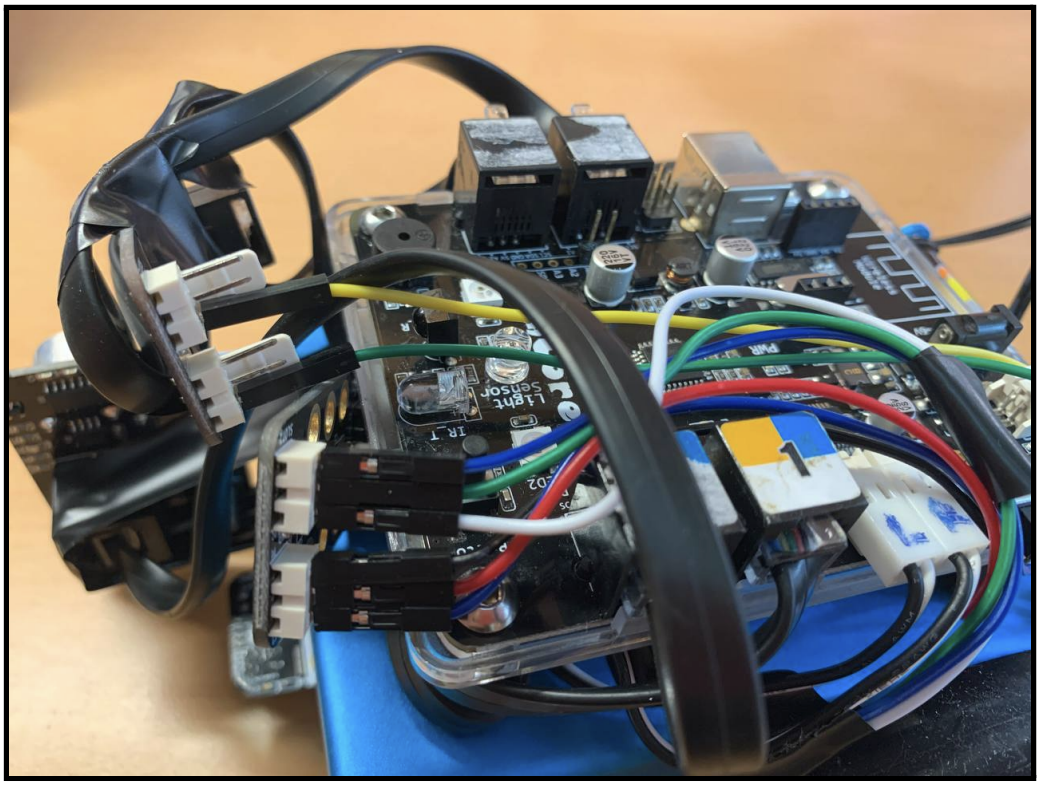
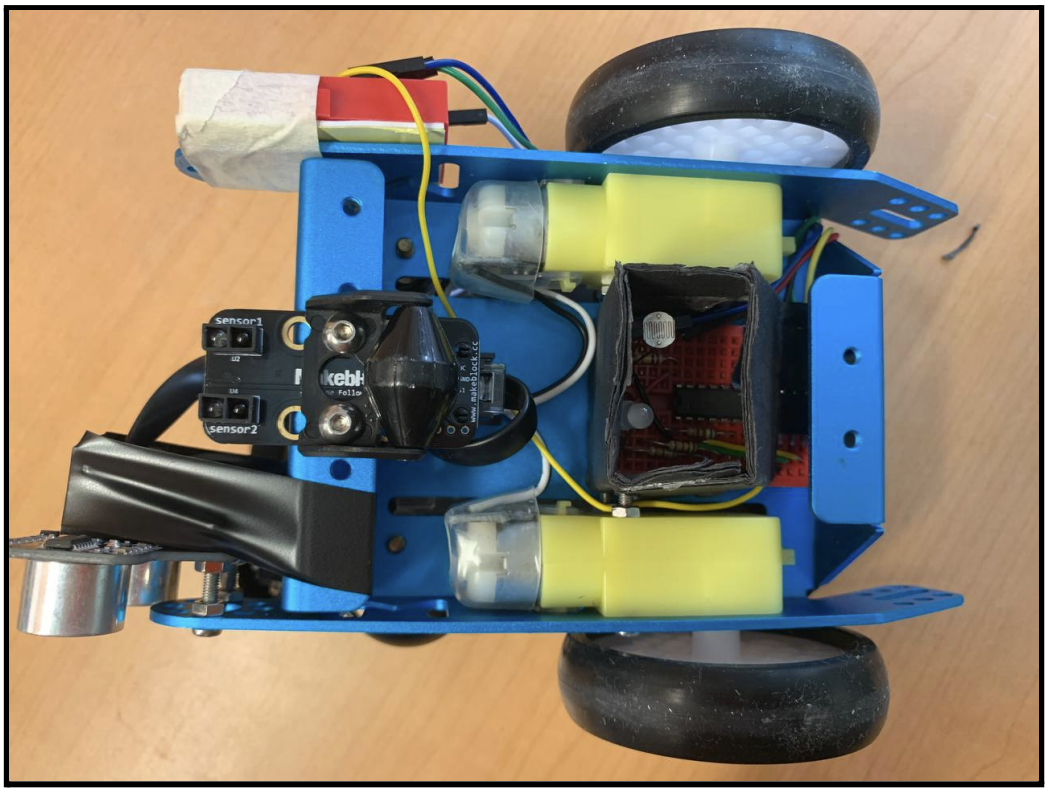


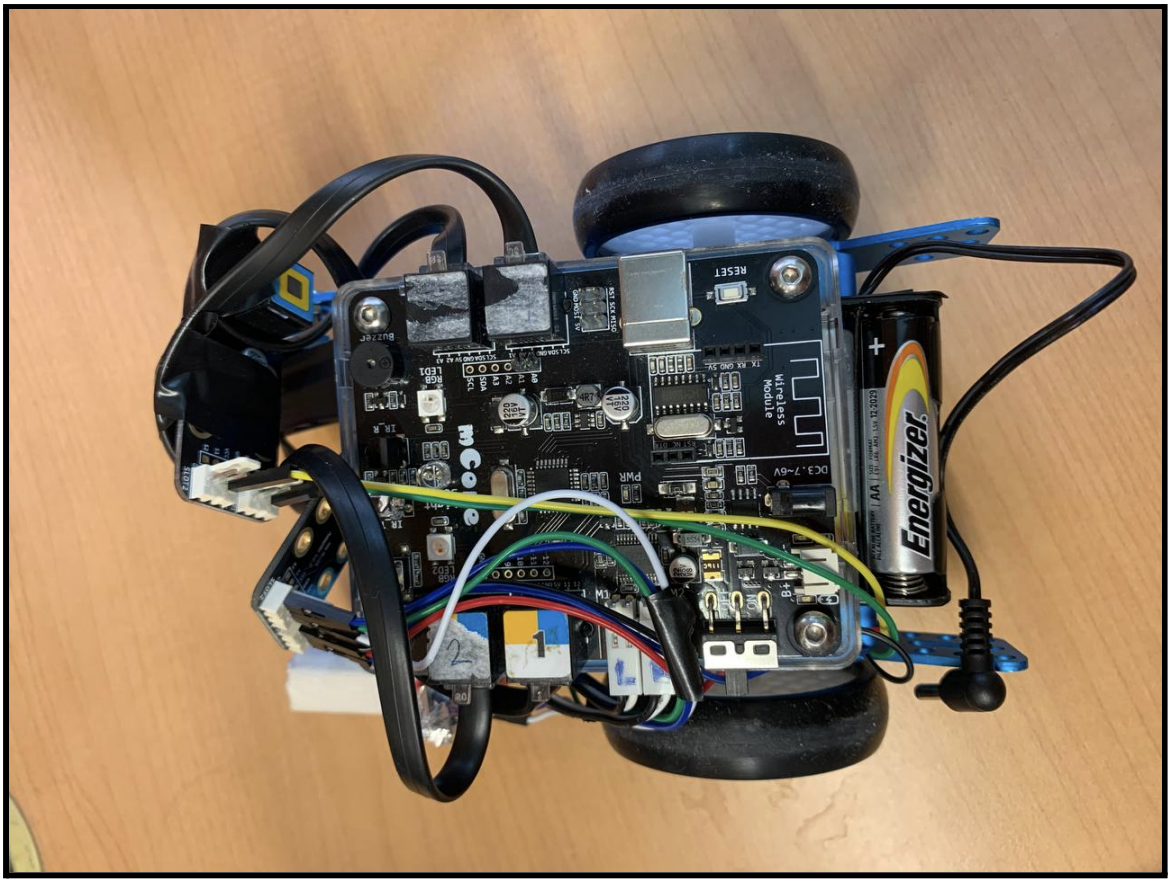
Figure 7: Shielding feature over IR sensors to prevent it from detecting ambient IR as much as possible

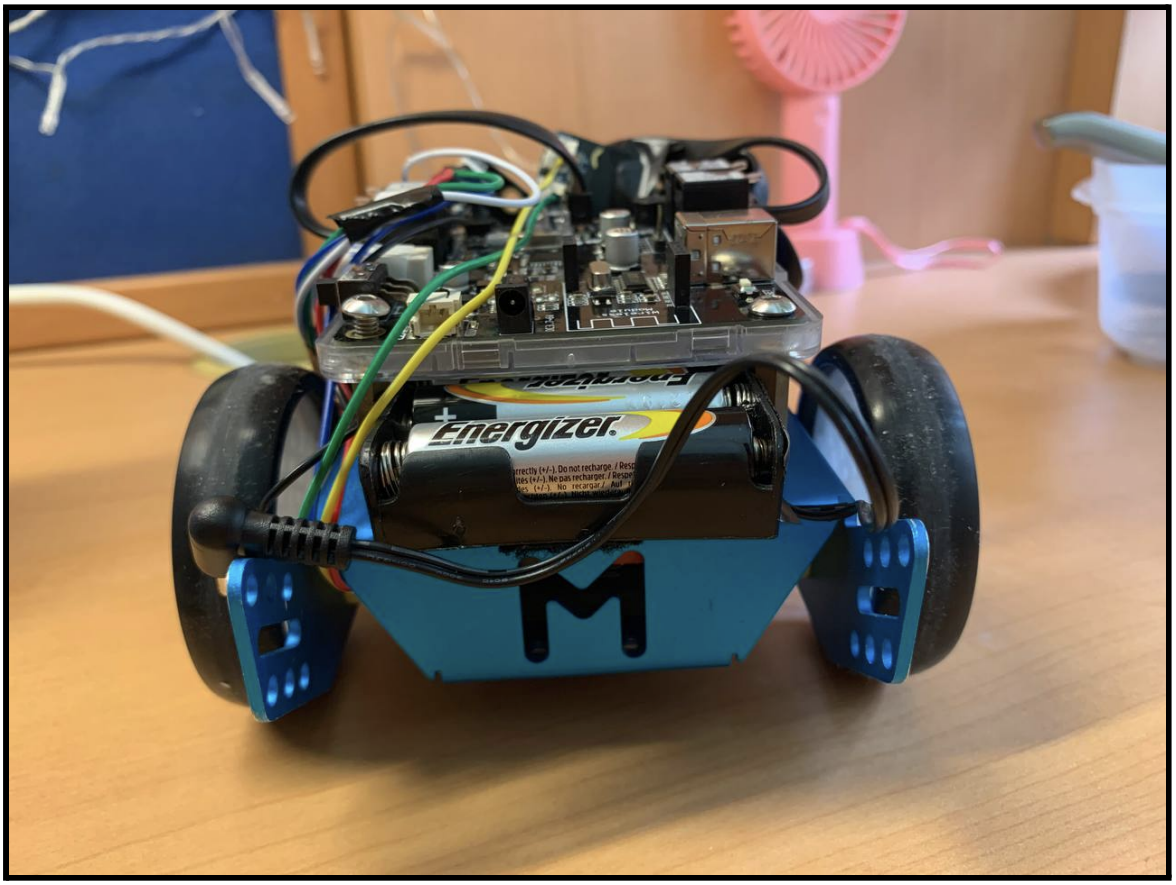
6. Appendix

i. General Pictures of Mbot






RESET
Wireless Module
Energizer


Energizer