National University of Singapore

Computer Engineering

CG1111 Engineering Principles and Practice I

The A-maze-ing Race Project Report

Studio Group: 3

Group number: 3a - 5

Team members

Name	Student ID
Nguyen Van Binh	A0219795L
Ngoi Hui Wen, Vanessa	A0221939Y
Nguyen Minh Tuan	A0219731E

1. Introduction

This report describes our development of the mBot of The A-maze-ing Race. It includes the algorithms, and how we worked together to solve the challenges and complete the maze.

2. Overall Algorithm

Figure 1 describes the overall algorithm used to help the mBot complete the different challenges in the maze.

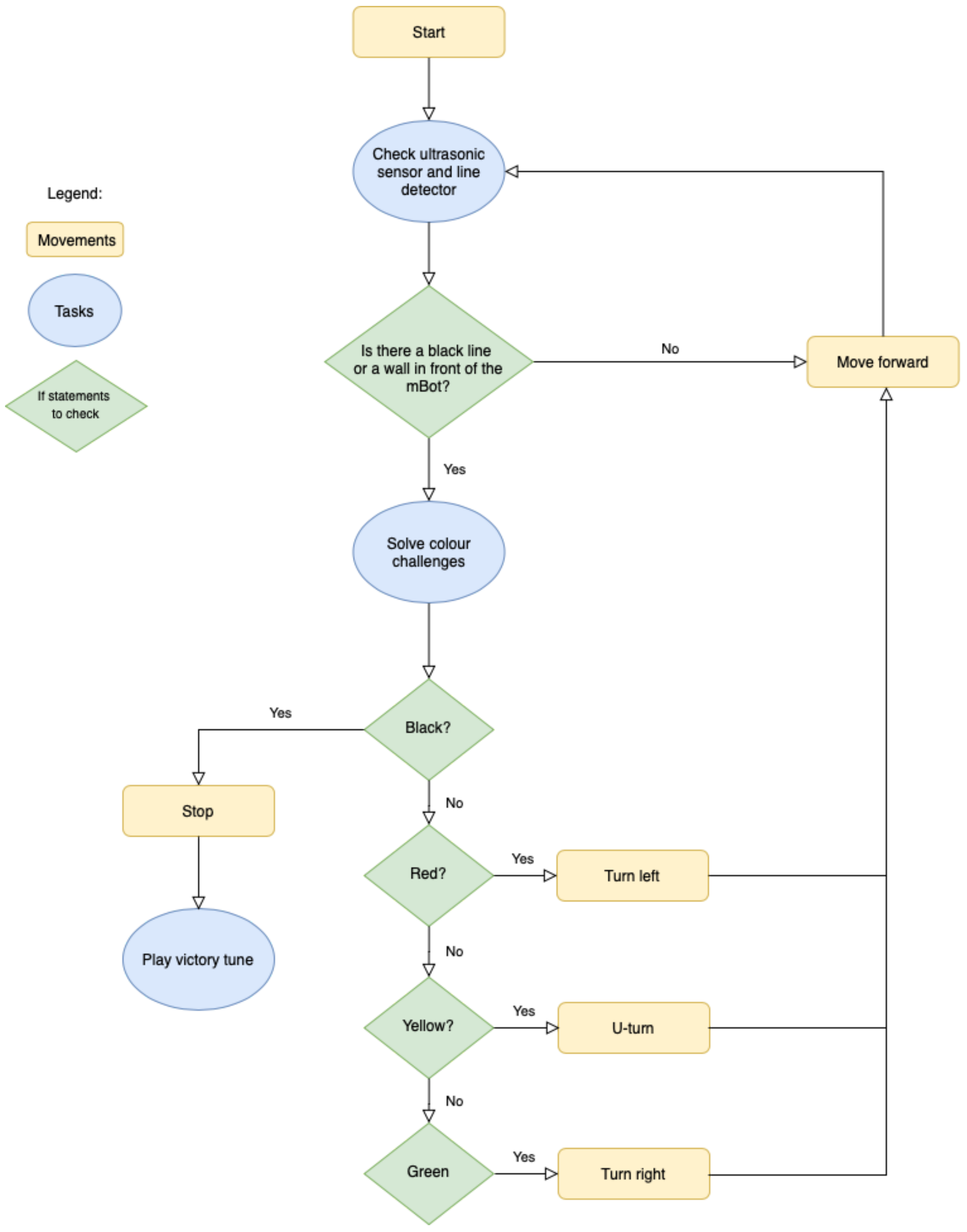


Figure 1.. Overall algorithms

3. Solving of Challenges

3.1. Movements

3.1.1. Overview

There were three types of movements the mBot was required to perform: moving forward, turning, and stopping. The speed of the left and right motors of the mBot was adjusted based on the output returned by the IR and colour sensors to ensure it would run smoothly in the maze.

3.1.2. Moving forward

From the start, we realized that the two motors did not behave identically. Although we initialized the speed of these two motors to the same value, the left motor spun slightly faster than the right motor. Hence, the proximity IR sensors were further required to ensure that the mBot would travel in a straight line in the center of the path.

For the calibration process using IR sensors, the mBot was first positioned centrally between the 2 walls of the maze and the initial values from the IR sensors were recorded. The 2 motors were then initialized to the same speed and allowed to move forward. Through observation of the position of the mBot, we determined a "safe range" of IR voltage values where the mBot was far enough from the walls. If the mBot exceeded the safe range, the speed of the specific motors would be adjusted accordingly to help the mBot return to the center, hence helping the mBot to travel in a straight line. This will be explained in more detail in Section 3.3.4.

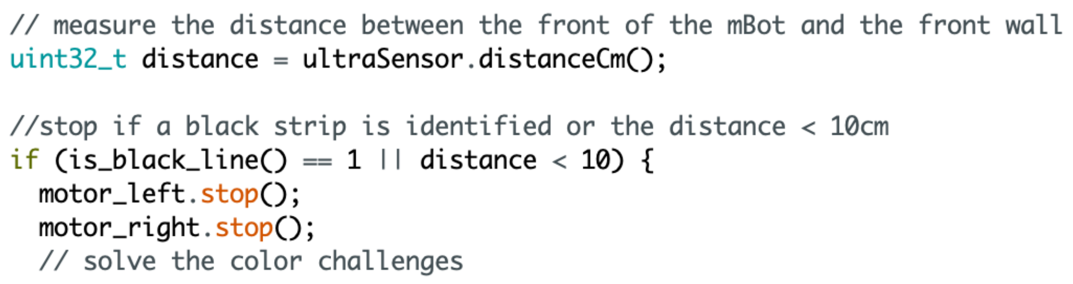
3.1.3. Turning

There were three different types of turns: right turn, left turn, and 180 degree U-turn within the same grid.

We initially considered two different methods to perform these movements: (1) Change the speed of the two motors but keep both motors moving forward or (2) Change one motor to rotate backward. After trial runs of both methods, we realized that the second method was safer as the mBot was less likely to bump into the wall at the side.

3.1.3. Stopping

Both the line and ultrasonic sensors were used. The line sensor was used to detect the black strip, while the ultrasonic sensor was used to measure the distance between the mBot and the wall in front of it. Both sensors were used to better ensure that the mBot would be able to stop at junctions and not bump into any walls. Through observation of the position of the mBot, the condition we decided to stop the motors was when the distance sensed by the ultrasonic sensor was less than 10cm. The implementation of this can be seen in Figure 2 below.



```
// measure the distance between the front of the mBot and the front wall
uint32_t distance = ultraSensor.distanceCm();

//stop if a black strip is identified or the distance < 10cm
if (is_black_line() == 1 || distance < 10) {
  motor_left.stop();
  motor_right.stop();
  // solve the color challenges
```

Figure 2. Code snippet for the stopping of the mBot

3.1.4. Difficulties encountered and steps to overcome them

Problem 1: Turning of the mBot varied as time progressed.

We observed that the mBot seemed to turn at different angles despite not changing our code over a few hours. However, after recharging the batteries, the mBot could run smoothly. Hence, we realized that the charge of the batteries affected the movement of the mBot. Depending on the current charge level of the batteries, they would supply different amounts of power to the motors.

To fix this problem, we kept the batteries fully charged. Although there might still be small differences that could reduce the accuracy of turnings, the mBot would be able to adjust its position based on the readings from IR sensors, hence eliminating this problem.

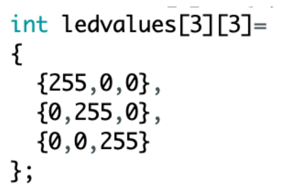
3.2. Colour Sensing

3.2.1 Overview

As the mCore has 2 RGB LEDs and 1 LDR, the colour sensing challenge could be accomplished by using these built-in components. Firstly, the 2 LEDs shine red, green, and blue lights onto the coloured paper above the mBot. The LDR then detects the light that is reflected off the paper and outputs the intensity of each colour of light in the form of digital voltage values ranging from 0-1024. Thus, each time the mBot stops at a junction to sense a colour, 3 readings corresponding to the values of red, green, and blue are produced.

3.2.2 Detection of RGB Values

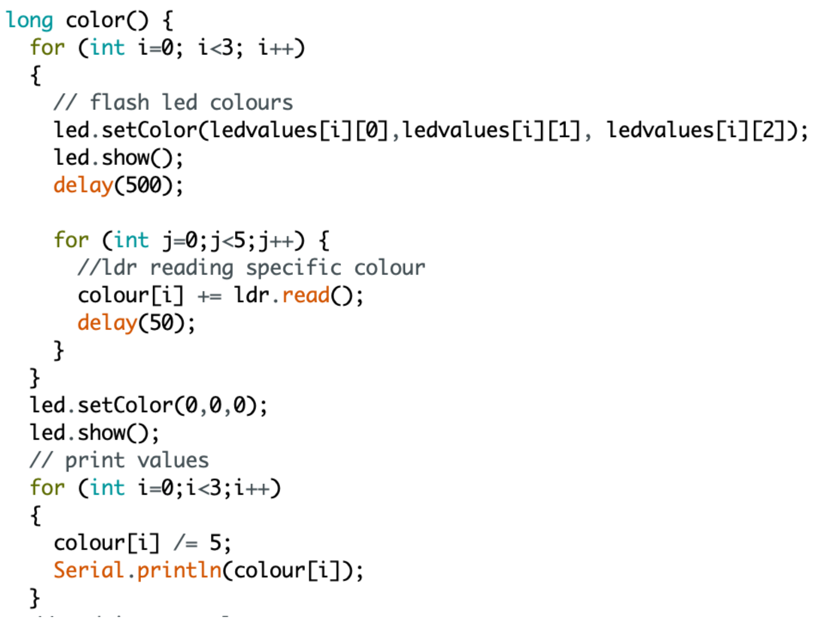
Firstly, for the LEDs to shine red, blue, and green lights, the respective RGB values of each colour are stored in a 2D array as shown in Figure 3.2.1 below. The 3 rows correspond to the RGB values of red, green, and blue respectively. For each colour, the RGB values for the other 2 colours are set to 0 to ensure that a pure colour is produced, while the value of the target colour is set to the maximum value of 255. This is to ensure that the highest intensity of the target colour is produced, which makes it easier to obtain accurate results in the later stage. For example, in the first row where the target colour to be shone by the LED is red, the RGB value of red is set to 255, while the values of green and blue are set to 0. The different colours are then produced through a loop as shown in Figure 3.



```
int ledvalues[3][3]=
{
  {255,0,0},
  {0,255,0},
  {0,0,255}
};
```

Figure 3. Code snippet for producing LED colours

Following the code, as shown in Figure 4, after each individual colour is flashed, the LDR takes 5 readings with a delay of 50 milliseconds between each reading. The 5 values are added up and the total value is stored in an array called colour. This array stores the RGB values of the colour that the mBot is sensing. After this process has been repeated for all 3 colours, each value in the colour array is divided by 5 to produce the average RGB values. Thus, this array now stores the final RGB values that will be checked against the different conditions to determine the final colour.



```
long color() {
  for (int i=0; i<3; i++)
  {
    // flash led colours
    led.setColor(ledvalues[i][0],ledvalues[i][1], ledvalues[i][2]);
    led.show();
    delay(500);

    for (int j=0;j<5;j++) {
      //ldr reading specific colour
      colour[i] += ldr.read();
      delay(50);
    }
  }
  led.setColor(0,0,0);
  led.show();
  // print values
  for (int i=0;i<3;i++)
  {
    colour[i] /= 5;
    Serial.println(colour[i]);
  }
```

Figure 4. Code snippet for flashing LED colours

3.2.3 Sensing of Colours

Based on observing the raw RGB values of the different colours at the start, our group realized that there were clear and distinct relationships between the RGB values for each colour that could help separate the different colours. Hence, the calibration method of using black and white as seen in Week 10 Studio 1 was not used and we instead relied on the relationships between the RGB values for each colour to differentiate between colours. These relationships were also independent of environmental conditions (eg. brightness of room) as all values would be similarly affected, hence it was an accurate way of detecting colours. Table 3.2.3 shows some samples of raw data of each colour that was collected and hence the conditions used to differentiate between the different colours.

To differentiate between red, green, and black, it was very clear what conditions to use as there was no chance for overlap which would result in the mBot detecting the wrong colour. However, between red and yellow, it was slightly more complicated. As both colours had red as the highest value, green as the second-highest, and blue as the lowest, the difference between red and green values was used to further differentiate between the colours. As red is a pure colour while yellow is a combination of red and green, hence the value of green would be much higher in yellow compared to green. As such, the difference between red and green values in yellow would be much smaller. After sampling many raw values of both yellow and red and observing the readings, the difference of 300 was decided as the boundary condition between the two colours, with some allowance given for fluctuation of values across different readings as shown in Table 1 below.

Colour mBot was sensing	Sample RGB values for each colour			Conditions used to differentiate between colours
	Red	Green	Blue	
Red	920.60	540.10	480.30	(R > G) && (R > B) && (R - G > 300)
Green	589.40	645.00	423.70	(G > R) && (G > B)
Yellow	895.60	770.20	791.40	(R > G) && (R > B) && (R - G < 300)
Black	361.40	304.00	313.20	R + G + B < 1500

Table 1. Boundary conditions of colours

3.2.4 Difficulties encountered and steps to overcome them

Problem 1: Fluctuation in values across readings
Despite the exact same environmental conditions, the RGB values across successive readings taken immediately one after the other had non-negligible fluctuations ranging from 10-50. This resulted in very inaccurate values, which would significantly affect the differentiation of colours.

To counter this, 5 readings of each value was taken. (i.e 5 readings of red, 5 readings of green, and 5 readings of blue). The values first added up into the colour array that stored the RGB values, then divided by 5 after all 3 colours had finished flashing. Hence the colour array stores the average values of each RGB value, which would increase the precision of the values that were used to check against the conditions for the different colours.

Problem 2: Difference in environmental conditions
We initially hardcoded the individual RGB values to determine the different colours. (eg. R > 900 && G > 600 && B > 400). However, we realized that different areas of the lab had different levels of brightness, and hence the difference in ambient lighting affected our readings significantly and these hardcoded condition checks no longer held true all the time.

To counter this, the difference between the individual RGB values were used instead as all 3 values would be similarly affected by the difference in environmental conditions and hence would be accounted for. Thus, our conditions mainly relied on relationships such as maximum value or difference between 2 values to detect colours. For black, hardcoded values were still used as regardless of environmental conditions, the difference between the RGB values of black and other colours was very significant, hence hardcoded values were still accurate in determining black.

3.3. Infrared Sensors

3.3.1 Overview

To prevent the mBot from bumping into the wall, two proximity IR sensors were used. They were attached to both sides of the mBot to detect the change in distances between the mBot and the walls of the maze. When the bot was too close to a wall, the speed of the motors was adjusted to help the mBot recentralize its path.

3.3.2 Hardware

Two IR sensors were used on each side of the mBot. The circuit design for the sensors followed the design in Week 10 Studio 1 as seen Figure 5 below. The mCore was used as the voltage source of the sensors, and the values obtained from the left sensor was through pin A0 while the right sensor was connected to A1.

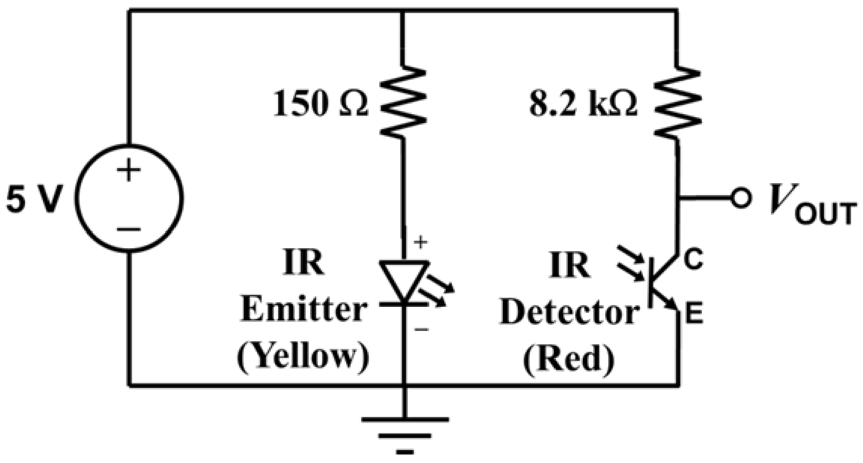
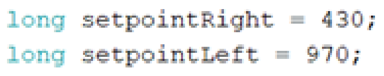


Figure 5. Circuit diagram of IR sensors

3.3.3 Calibration on mBot

As the distance between the IR sensor and the wall changes, the IR detector outputs different values. For the left IR sensor, the value increases as the mBot travelled closer to the wall. For the right IR sensor, the value decreases as it travels closer to the wall. Through experimentation and observation of raw values, the boundary values obtained by two sensors for the mBot to be deemed as safe and in the middle of the path are shown below in Figure 6. For the left sensor, any value larger than 970 meant that the mBot was too close to the wall, while for the right sensor it was any value smaller than 430.



```
long setpointRight = 430;
long setpointLeft = 970;
```

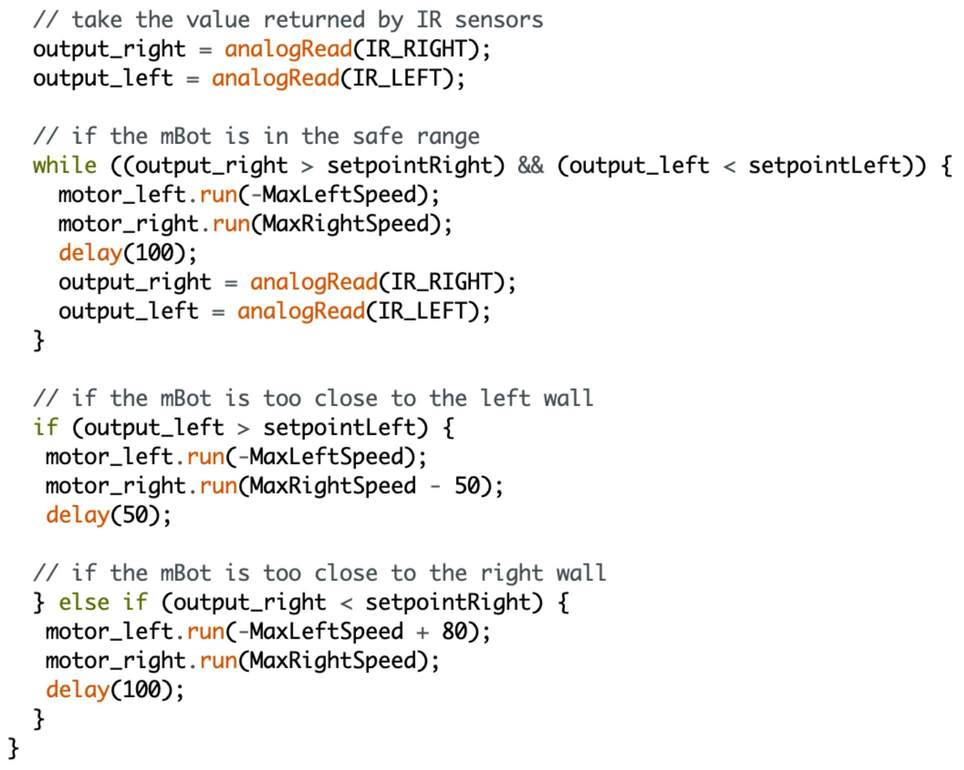
Figure 6. Setpoints of the IR sensors

3.3.4 Algorithm for keeping the mBot straight

When the mBot travelled too close to a wall, the speed was adjusted using the following code in Figure 7. Since the values of the left IR sensor gives increases while the values of the right IR sensor decreases as the mBot travels closer to the wall, the mBot is considered to be at the center when the values from the left sensor is smaller than the setpointLeft and the values from the right sensor are larger than the setpointRight.

When output_left is larger than the setpointLeft, the mBot is too close to the left wall. Consequently, the speed of the right motor is decreased to allow the mBot to curve away from the left wall. When output_right is smaller than the setpointRight, the mBot is too close to the right wall. Consequently, the speed of the left motor is decreased to allow the mBot to curve away from the right wall.

Since the two motors did not behave identically, the decrease in speed is different for each motor in order to ensure the mBot can return to the middle. Furthermore, there is also a delay of 100 milliseconds between each adjustment to prevent the mBot from travelling in a zigzag manner.



```
  // take the value returned by IR sensors
  output_right = analogRead(IR_RIGHT);
  output_left = analogRead(IR_LEFT);

  // if the mBot is in the safe range
  while ((output_right > setpointRight) && (output_left < setpointLeft)) {
    motor_left.run(-MaxLeftSpeed);
    motor_right.run(MaxRightSpeed);
    delay(100);
    output_right = analogRead(IR_RIGHT);
    output_left = analogRead(IR_LEFT);
  }

  // if the mBot is too close to the left wall
  if (output_left > setpointLeft) {
   motor_left.run(-MaxLeftSpeed);
   motor_right.run(MaxRightSpeed - 50);
   delay(50);

  // if the mBot is too close to the right wall
  } else if (output_right < setpointRight) {
   motor_left.run(-MaxLeftSpeed + 80);
   motor_right.run(MaxRightSpeed);
   delay(100);
  }
}
```

Figure 7. Code snippet for adjusting motors' speed based on the data returned by IR sensors

3.3.5 Difficulties encountered and steps to overcome them

Problem 1: Output from the IR sensors were inconsistent.

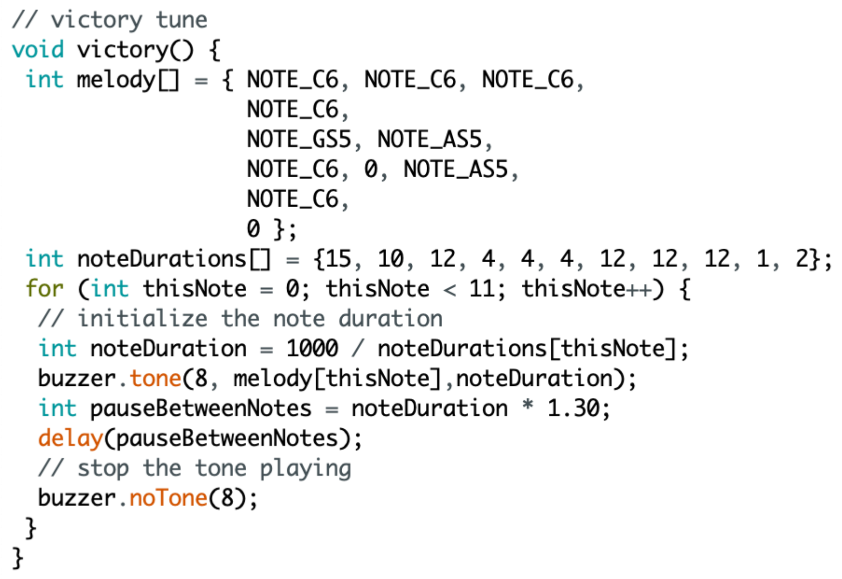
The values from IR sensors fluctuated significantly even when the mBot was stationary and not moving. We suspected that it might have been due to loose connections in our circuit, or the tape was not sticky enough to completely fix the position of the IR sensors.

First, we tried to change the wires and rebuilt the circuit to check if the problem was with the circuit(e.g loose wires, connected circuit wrongly). However, after the same issue persisted, we decided to tape the sensors to the breadboard and also tape the breadboard to the mBot. This improved the precision of our readings and our values did not fluctuate as much as before.

Problem 2: The values from the left IR sensor increased as the mBot travelled closer to the wall. In theory, the value detected by the IR detector should decrease when distance between the sensor and wall decreases. However, even after rebuilding our circuit and checking our code, the same trend was still observed. To account for this difference, we decided to reverse the inequality sign of our condition. Hence it can be seen from Figure 3.3.4 that the condition for the left and right motors are different.

3.4. Victory Tune

After stopping, if black is detected (i.e. the mBot reaches the end of the maze), we call the victory() function (as shown in figure 8) to play the victory tune.



```
// victory tune
void victory() {
 int melody[] = { NOTE_C6, NOTE_C6, NOTE_C6,
                  NOTE_C6,
                  NOTE_GS5, NOTE_AS5,
                  NOTE_C6, 0, NOTE_AS5,
                  NOTE_C6,
                  0 };
 int noteDurations[] = {15, 10, 12, 4, 4, 4, 12, 12, 12, 1, 2};
 for (int thisNote = 0; thisNote < 11; thisNote++) {
  // initialize the note duration
  int noteDuration = 1000 / noteDurations[thisNote];
  buzzer.tone(8, melody[thisNote],noteDuration);
  int pauseBetweenNotes = noteDuration * 1.30;
  delay(pauseBetweenNotes);
  // stop the tone playing
  buzzer.noTone(8);
 }
}
```

Figure 8. Code snippet for the victory tune

4. Work Division

The workload was divided equally among our group members, with each student focusing on a different challenge to increase the efficiency of work done given the limited amount of time. Table 2 summarises the main responsibility of each member. After completing our individual components, we then combined our code together and worked together to do the final testing and fine tuning of the mBot. All group members generally contributed equally and were focused during our group discussions.

Task	**Student-in-charge**
Movements (motors, turns, line follower sensor, and ultrasonic sensor)	Nguyen Van Binh
Colour sensor	Ngoi Hui Wen Vanessa
Infrared sensor	Nguyen Minh Tuan

Table 2. Work division

5. References

[1] Available: https://github.com/Makeblock-official/Makeblock-Libraries

[Access: 25-Oct-2020]