CG1111 Lab Group 2-4-B
The A-maze-ing Race Written Report

Ling Si Hui, Shiho A0206391J

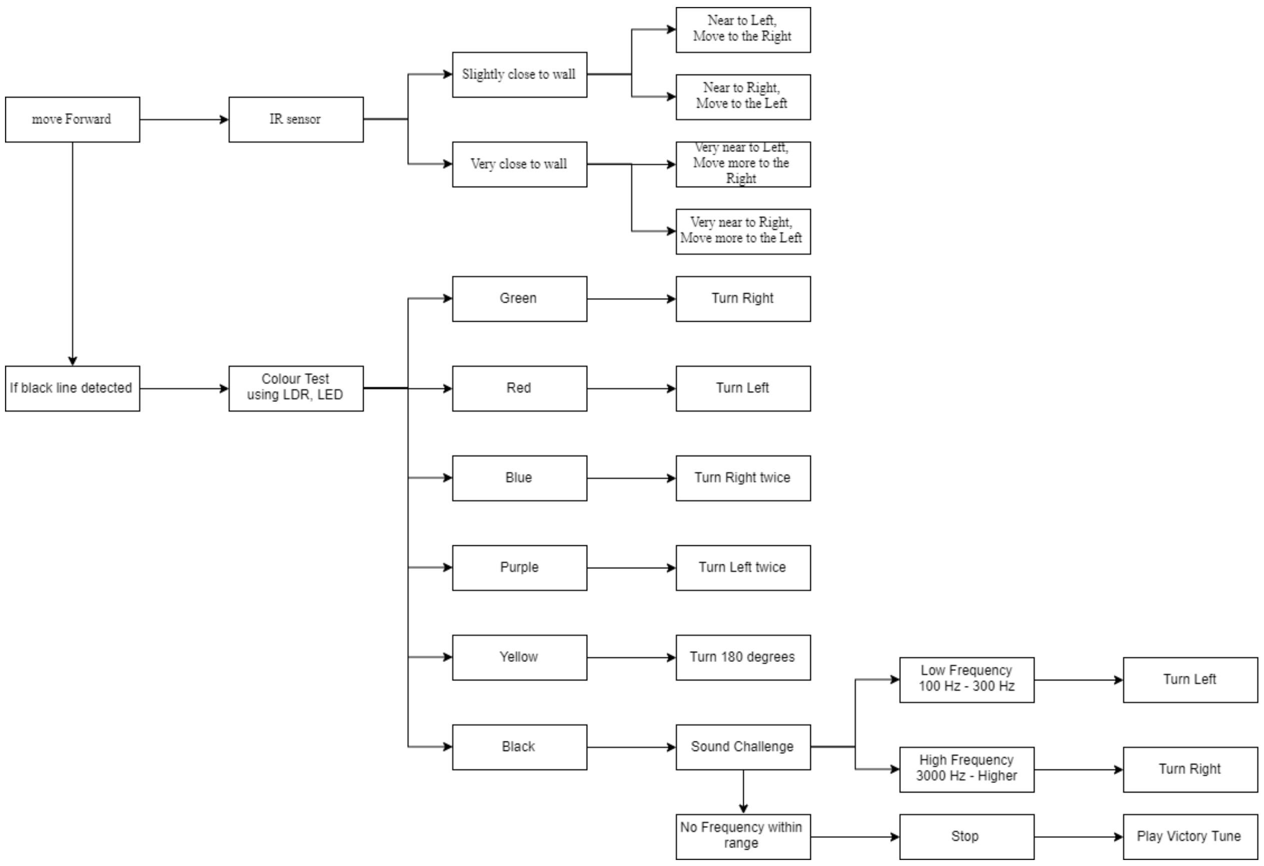
Liu Yifeng A0206037N

Lu Ziyi A0197282U

Tan Zheng Chong, Shawn A0199891A

Tay Wee Teng A0206066L

1. Overall Algorithm



move Forward
IR sensor
Slightly close to wall
Near to Left, Move to the Right
Near to Right, Move to the Left
Very close to wall
Very near to Left, Move more to the Right
Very near to Right, Move more to the Left
If black line detected
Colour Test using LDR, LED
Green
Turn Right
Red
Turn Left
Blue
Turn Right twice
Purple
Turn Left twice
Yellow
Turn 180 degrees
Black
Sound Challenge
Low Frequency 100 Hz - 300 Hz
Turn Left
High Frequency 3000 Hz - Higher
Turn Right
No Frequency within range
Stop
Play Victory Tune

Figure 1. Overall algorithm of our mBot code

The loop begins with an if-condition that detects the presence of blackline using the line follower function. If there is no black line detected, the mBot will continue to move forward with the “moveForward” function.

In the “moveForward” function, the in-build left and right Infrared (IR) sensors are constantly utilized via an if-condition with variations that account for changes in distance between the bot and the left and right walls respectively.

Whereas, if a black line is detected, a “Stop” function is applied to stop the rotation of the left and right motors. Following, a “colour_test” function allows the mBot to detect the different colours using the LDR and LED on the mBot. As soon as a certain colour is sensed by the LDR, it will enter a unique action function. For example, if green is observed, the mBot’s motor will operate such that the mBot turns to the right for a preset duration and the loop resets and continues to move forward.

For the case where yellow is detected, it is required to turn 180 degrees within the same tile. The left and right IR sensor is once again utilized to eliminate the likelihood of the mBot colliding with the walls. It is observed that since the wheels are placed at the back of the mBot, it should turn away from the wall to prevent any collision. Hence, if IR sensors detect that the mBot is closer to the right wall, it will turn anticlockwise, away from the wall and vice versa.

The mBot is required to turn twice either to the left and right across 2 tiles when purple and blue is detected respectively.

When black is detected, the mBot will undergo “SoundChallenge” function which senses for high or low frequency via the installed microphone setup. If the low frequency is detected, it turns left and if the high frequency is detected, it turns right. In the case where neither of the frequencies is picked up, the bot will become stationary and play the victory tune with the buzzer.

2.Implementation details of various subsystems

2.1 Algorithm for movement (ultrasonic sensor)

The movement is determined by the speed of rotation of the left and right motor. We initialized the motor as follows and set the original moving speed to 250:

```
MeDCMotor MotorL(M1);               //MOVE MBOT (LEFT)
MeDCMotor MotorR(M2);               //MOVE MBOT (RIGHT)
```

Figure 2.1.1 Code for moving speed of mBot

Since the motors are placed on the left and right side of the mBot, in order for the wheels to rotate in the same direction, one motor has to turn clockwise while the other motor has to turn counter-clockwise as shown below:

```
void Forward()
{
  MotorL.run(-moveSpeed);
  MotorR.run(moveSpeed);
}
```

Figure 2.1.2 Code for moving mBot forward

For the mBot to make a left or right turn, the motor speed is reduced by 0.65 with both wheels turning in opposite directions. This allows the mBot to turn smoothly. The delay added right after determines the duration of turn.

```
void TurnLeft()
{
  MotorL.run(moveSpeed * 0.65);
  MotorR.run(moveSpeed * 0.65);
  delay(450);
}
```

```
void TurnRight()
{
  MotorL.run(-moveSpeed * 0.65);
  MotorR.run(-moveSpeed * 0.65);
  delay(450);
}
```

Figure 2.1.3 Code for left and right turn

Another variation involves the bot to turn left or right twice, across two tiles. This requires the mBot to turn 90 degrees, move forward for a certain duration and turn another 90 degrees in the same direction. To ensure that the bot does not collide with the wall when moving forward, a front ultrasonic sensor is put in place. The ultrasonic sensor measures the distance between the front wall and itself, allowing the bot to move forward as long as it stays out of a certain range using a while loop. Once the bot exceeds this distance, it proceeds to turn in the direction of the command (Figure 2.1.5).

```
void TurnLeft2()
{
  TurnLeft();
  while(ultraSensor.distanceCm() >9){
  Forward();
  }
  TurnLeft();
}
```

```
void TurnRight2()
{
  TurnRight();
  while(ultraSensor.distanceCm() >9){
  Forward();
  }
  TurnRight();
}
```

Figure 2.1.4 Code for left and right twice turn

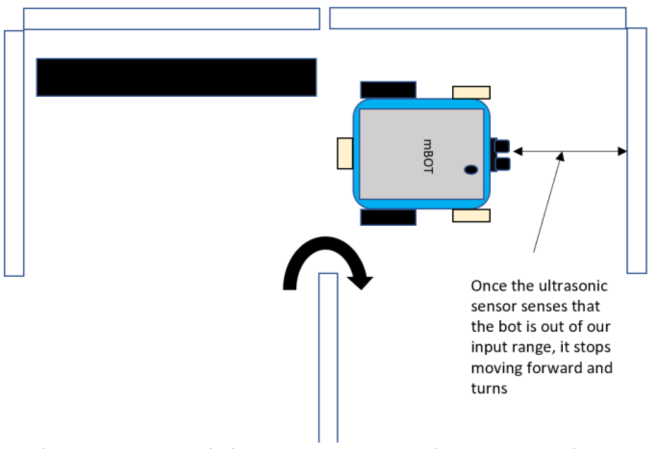

mBOT
Once the ultrasonic sensor senses that the bot is out of our input range, it stops moving forward and turns

Figure 2.1.5 Diagram explaining our ultrasonic usage when turning twice

The last movement requires the bot to turn 180 degrees within the same area without hitting the walls. We made use of IR sensors to decide where the bot should turn, i.e. clockwise or anti-clockwise. If the bot appears to be closer to the left, it will turn clockwise away from the wall (Fig 2.1.7).

```
// Take in IR values when mbot does 180 turn, to avoid crashing on the walls
void Turn180IR()
{
  int inputLeft2, inputRight2;
  inputLeft2 = analogRead(IRLeft);
  inputRight2 = analogRead(IRRight);
  // if mbot is too close to the right, it turns anticlockwise
  if (inputLeft2 > inputRight2){
    MotorL.run(moveSpeed);
    MotorR.run(moveSpeed);
    delay(555);
  }
  // if mbot is too close to the left, it turns clockwise
  else {
    MotorL.run(-moveSpeed);
    MotorR.run(-moveSpeed);
    delay(555);
  }
}
```

Figure 2.1.6 Code for turning 180 degrees with IR

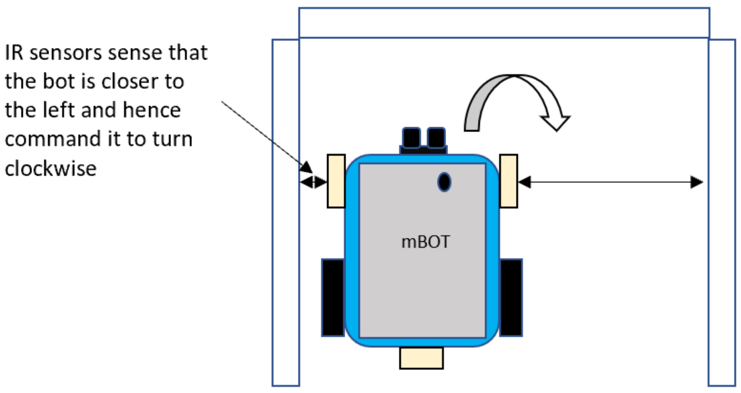

IR sensors sense that the bot is closer to the left and hence command it to turn clockwise
mBOT

Figure 2.1.7 Diagram explaining our IR usage when the bot turns 180 degrees

Lastly, we provided a Stop function that will be executed during the end of the maze to reduce the motor speed to 0 so that the bot will remain in its position.

```
void Stop()
{
  MotorL.run(0);
  MotorR.run(0);
}
```

Figure 2.1.8 Code for stopping the mBot

2.2 Infrared (IR) Sensor Algorithm to prevent mBot from hitting the walls

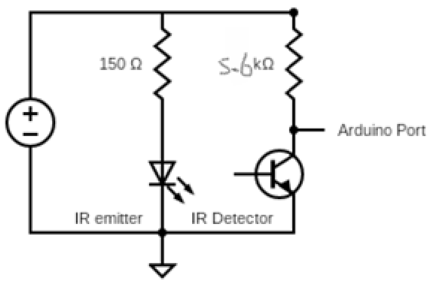
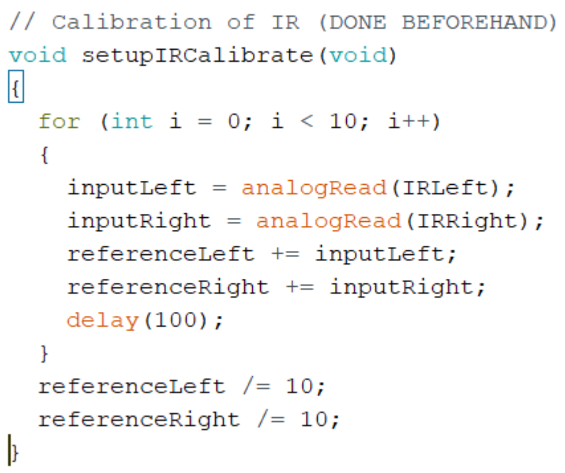


Figure 2.2.1 IR Circuit Diagram

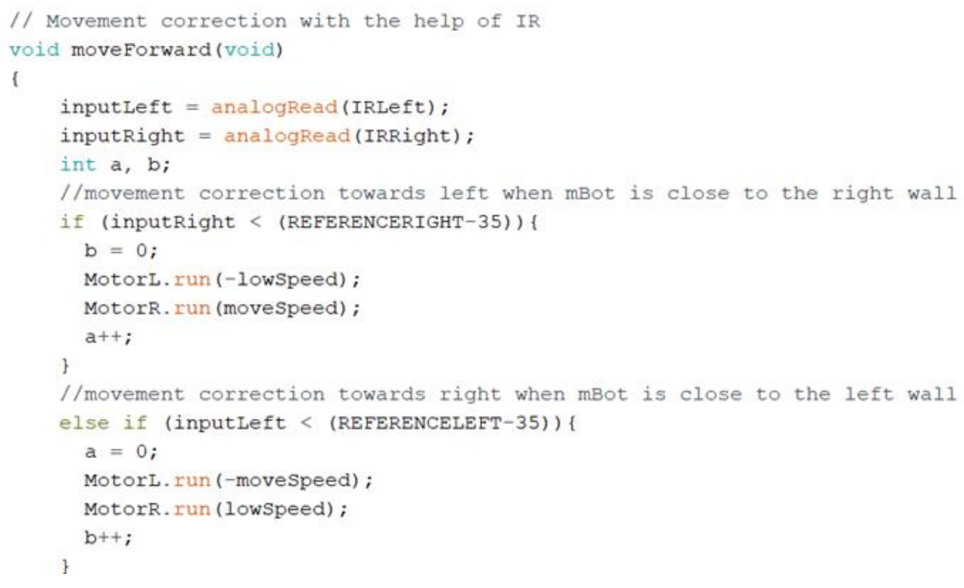
The position of the mBot with respect to the left and right walls of the maze can be determined by readings obtained from two IR sensors attached to the left and right of the mBot respectively. After setting the left and right IR sensors to pin A3 and A2 of the Arduino, we then proceeded to calibrate the ideal position of the mBot from the walls of the maze. The code for IR calibration is as shown below:



```
// Calibration of IR (DONE BEFOREHAND)
void setupIRCalibrate(void)
{
  for (int i = 0; i < 10; i++)
  {
    inputLeft = analogRead(IRLeft);
    inputRight = analogRead(IRRight);
    referenceLeft += inputLeft;
    referenceRight += inputRight;
    delay(100);
  }
  referenceLeft /= 10;
  referenceRight /= 10;
}
```

Figure 2.2.2 Code for IR Calibration

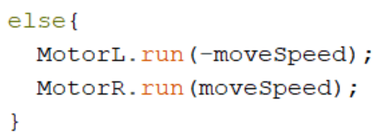
Based on our calibration, the left IR and right IR readings are 651 and 653 respectively when the mBot is placed in the ideal position. These two values are used as our reference point. We then proceed to determine the IR readings when a movement correction is needed – when the mBot is either too close to the left or to the right of the walls. After multiple trial and errors, we have determined the movement correction reference point. Whenever the IR reading is less than the movement correction reference point, the mBot will adjust its motor speed and move back to its ideal position.



```
// Movement correction with the help of IR
void moveForward(void)
{
    inputLeft = analogRead(IRLeft);
    inputRight = analogRead(IRRight);
    int a, b;
    //movement correction towards left when mBot is close to the right wall
    if (inputRight < (REFERENCERIGHT-35)){
      b = 0;
      MotorL.run(-lowSpeed);
      MotorR.run(moveSpeed);
      a++;
    }
    //movement correction towards right when mBot is close to the left wall
    else if (inputLeft < (REFERENCELEFT-35)){
      a = 0;
      MotorL.run(-moveSpeed);
      MotorR.run(lowSpeed);
      b++;
    }
```

Figure 2.2.3 Code for mBot Movement Correction

As demonstrated in Figure 2.2.3, when the right IR reading is less than the movement correction reference point, which is (REFERENCERIGHT – 35), the mBot is too close to the right wall. The speed of the left motor will decrease while the right motor will continue to function at a normal speed. Thus, this causes the mBot to move away from the right wall.
If both the left and right IR readings are within the movement correction reference point, the mBot will move straight as both motors will run at normal speed but of opposite direction, as shown in Figure 2.2.4.



```
else{
  MotorL.run(-moveSpeed);
  MotorR.run(moveSpeed);
}
```

Figure 2.2.4 Code to make mBot in a straight line

2.3 Audio Processing Circuit Design

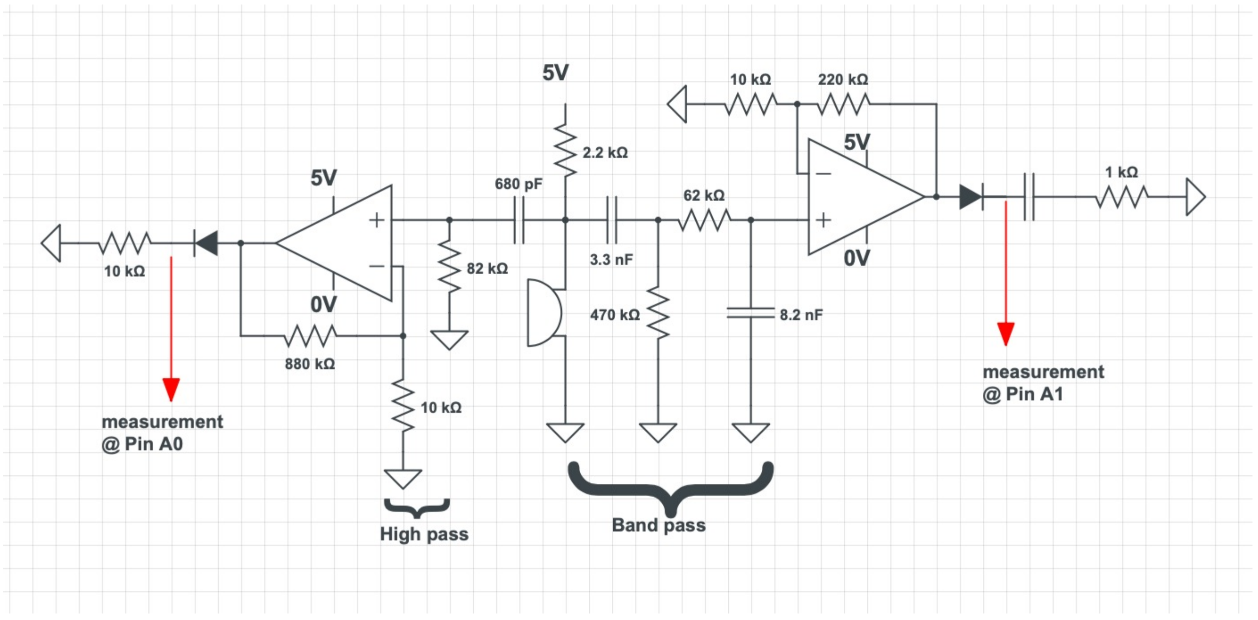


Figure 2.3.1 Audio Sound Processing Circuit Design

The circuit consists of several parts, namely band-pass filter, high-pass filter, and microphone. The microphone is connected to the 5V power supply via a 2.2kΩ resistor, to ensure that its operating voltage is of its optimum of 2.5V. The purpose of having a band-pass filter and high-pass filter is to accurately capture the sounds of the required frequency. For the 100-300 Hz signal, a band-pass filter is needed, whereas a high-pass filter is required for the 3000 Hz signal. To configure the circuit, appropriate resistance (R) and capacitance (C) values were calculated using Equation 1 as follows:

$$cut - off\ frequency, f = \frac{1}{2\pi RC} \tag{1}$$

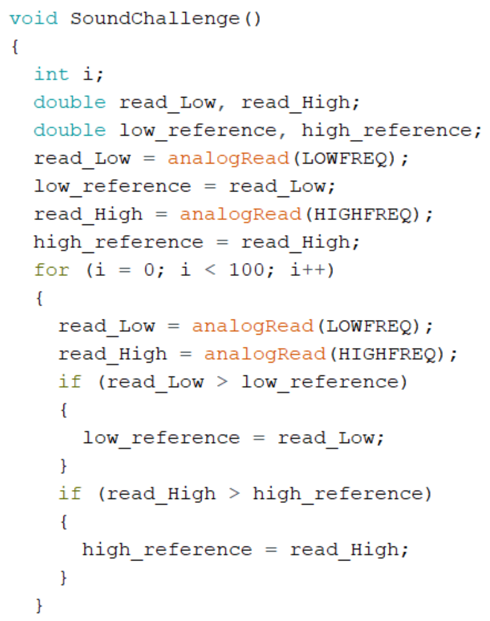
By calculating the respective RC values, we then proceed to choose the appropriate resistor and capacitor. The resistors with the approximate resistance values were used, and the actual cut-off frequencies were calculated to ensure that they were of the appropriate values.

Filter type	Cut-off Frequency/Hz	RC Value /ΩF	Resistance Value Used/kΩ	Capacitance Value Used/pF	Calculated Cut-off Frequency/Hz
Band-Pass	100 (High-Pass)	0.0015915	470	3300	102.61
	300 (Low-Pass)	0.00053052	62	8200	313.05
High-Pass	3000	0.000053052	82	680	2854.29

Figure 2.3.2 RC values for accurate calculation

The processed signals from the filters were then passed into a non-inverting op-amp for amplification, with Ri value of 10kΩ and Rf value of 880kΩ for high-pass filter; with Ri value of 10kΩ and Rf value of 220kΩ for band-pass filter, which amplifies the output voltage to a suitable value significant enough for processing and analysis. The output from the op-amp is then connected to an envelope detector with a resistor-capacitor network, to produce an envelope of the signal to be read by the Arduino.

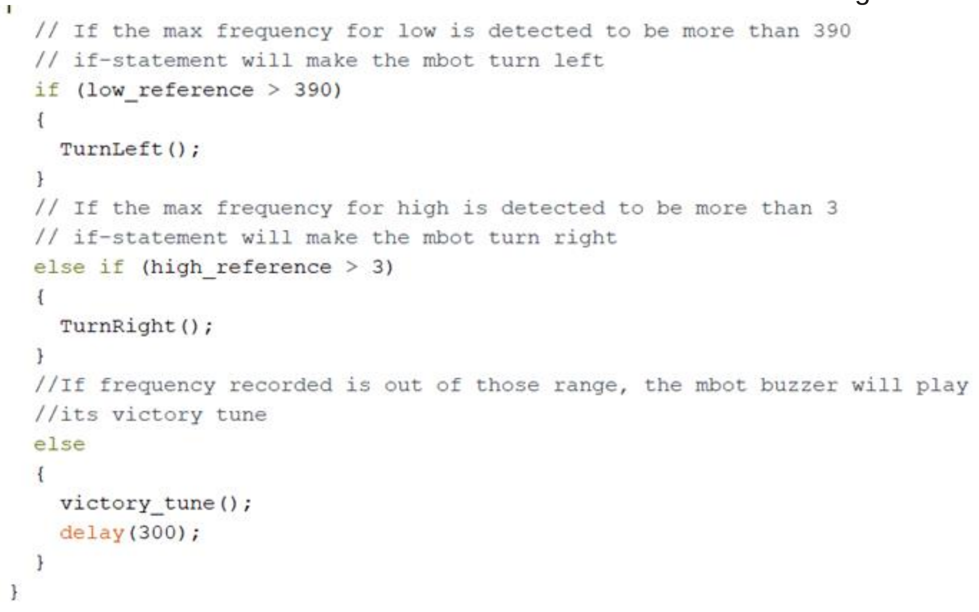
2.4 Audio Processing Circuit Algorithm



```
void SoundChallenge()
{
  int i;
  double read_Low, read_High;
  double low_reference, high_reference;
  read_Low = analogRead(LOWFREQ);
  low_reference = read_Low;
  read_High = analogRead(HIGHFREQ);
  high_reference = read_High;
  for (i = 0; i < 100; i++)
  {
    read_Low = analogRead(LOWFREQ);
    read_High = analogRead(HIGHFREQ);
    if (read_Low > low_reference)
    {
      low_reference = read_Low;
    }
    if (read_High > high_reference)
    {
      high_reference = read_High;
    }
  }
```

Figure 2.4.1 Code to determine the maximum value for low_reference and high_reference

100 readings are taken for both low and high frequency challenge, and the maximum reading of each challenge is being stored as the low_reference and high_reference respectively. The low_reference and high_reference values are used to determine the calibration for the sound challenge.



```
  // If the max frequency for low is detected to be more than 390
  // if-statement will make the mbot turn left
  if (low_reference > 390)
  {
    TurnLeft();
  }
  // If the max frequency for high is detected to be more than 3
  // if-statement will make the mbot turn right
  else if (high_reference > 3)
  {
    TurnRight();
  }
  //If frequency recorded is out of those range, the mbot buzzer will play
  //its victory tune
  else
  {
    victory_tune();
    delay(300);
  }
}
```

Figure 2.4.2 Code for mBot action after detecting black color board on top

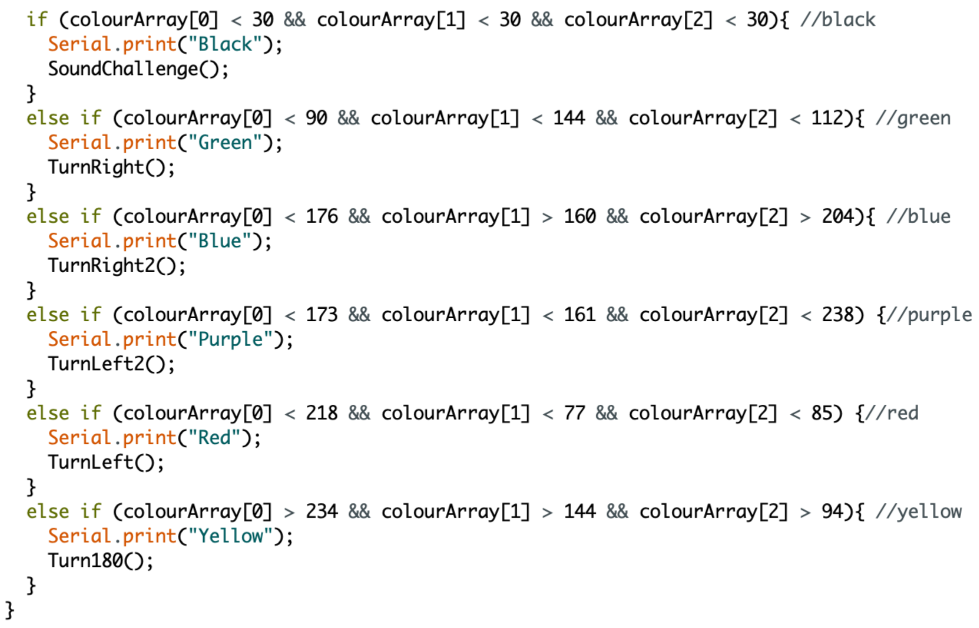
If the low_reference is larger than 390 and the high_reference is larger than 3, the mBot will turn left and right respectively. Should neither the above conditions are met, the mBot will determine that it is the end of the maze and play the victory tune as there is no sound challenge detected and the color of the board on top of the mBot is black.

2.5 Algorithm for colour sensing
2.5.1 Calibrations for colour sensing

For the calibrations for our colour sensor, we modelled the code setBalance() given to us during one of our lab sessions. In this function setBalance(), we took down the RGB values for the white and black and input the greyDiff array with the difference between the white and black RGB values. We ensured that the white colour and black colour samples used for calibration is purely white and purely black to ensure the light intensity range is accurate, hence giving more accurate readings of the subsequent RGB values of other colours. We also ensure that we calibrated our mBot under the test environment to provide an accurate reference point which increases our accuracy and minimizes uncertainty. In addition, we included a black hat which enclosed the 2 LED and the LDR on the mbot to minimize uncertainty provided by incident light from the environment.

2.5.2 Obtaining RGB values of colours

After calibrations, we recorded the RGB value of the respective colours. When measuring the RGB value for the colours, on top of including the black hat, we ensure that the mBot is under similar conditions when detecting the colours during the actual test run. This means that we recorded the values with the mBot surrounded by at least two panels at all times. We recorded multiple values of the RGB values until we are able to determine the range where we are able to get the correct output for each colour detected. Once we determine the range of the respective colours, we integrate the movement functions and the sound challenge functions with the colour test function.



```
  if (colourArray[0] < 30 && colourArray[1] < 30 && colourArray[2] < 30){ //black
    Serial.print("Black");
    SoundChallenge();
  }
  else if (colourArray[0] < 90 && colourArray[1] < 144 && colourArray[2] < 112){ //green
    Serial.print("Green");
    TurnRight();
  }
  else if (colourArray[0] < 176 && colourArray[1] > 160 && colourArray[2] > 204){ //blue
    Serial.print("Blue");
    TurnRight2();
  }
  else if (colourArray[0] < 173 && colourArray[1] < 161 && colourArray[2] < 238) {//purple
    Serial.print("Purple");
    TurnLeft2();
  }
  else if (colourArray[0] < 218 && colourArray[1] < 77 && colourArray[2] < 85) {//red
    Serial.print("Red");
    TurnLeft();
  }
  else if (colourArray[0] > 234 && colourArray[1] > 144 && colourArray[2] > 94){ //yellow
    Serial.print("Yellow");
    Turn180();
  }
}
```

Figure 2.4.3 Code for RGB values of the respective colours

2.6 Algorithm for end of maze and victory music

Upon detection of Sound challenge, if there is no high frequency sound or low frequency sound being detected within the next 100 samples taken, the mBot will recognize that as a signal for the end of the maze. When that happens, the victory_tune function will be executed to play the programmed music using the mbot's inbuilt buzzer and tone function. The function will run through an array containing different notes to be played and delays that determine the length of the notes. For example [NOTE_C3] indicates a C note on the 3rd octave with the length of one quaver, while [NOTE_A3, NOTE_R3] indicates a A note on the 3rd octave with the length of a crotchet. A note is played using the inbuilt tone() and noTone() function, by providing the frequency of the note to be played, the pin (8 for mBot by default), and the duration of the note (250ms for quaver at 60BPM). After completing four bars of the victory tune, the array ends and the program will go back to detecting for a black line. If the mbot is not moved and no sound challenge is detected, the mbot will continue to loop the victory_tune function, marking the end of the maze.

3.Work division within the team

Job description	**Name**
Hardware: IR Sensor	Yifeng & Ziyi
Hardware: Microphone	Yifeng & Ziyi
Hardware: Setting up mBot	Si Hui, Yifeng, Shawn, Wee Teng & Ziyi
Code for IR Sensor	Yifeng
Code for victory tune	Ziyi
Code for colour sensing	Si Hui
Code for stopping at black tape	Si Hui
Code for frequency detection/microphone	Shawn
Calibrating colour sensor	Si Hui, Wee Teng
Code for mbot movement	Si Hui, Wee Teng & Shawn

For our group, we split the workload by assigning each challenge to each members. At first, we completed our task separately. After everyone completed their assigned work, we came together and integrate our circuits and codes one by one, while debugging and modifying our codes at the same time. Everyone contributed as much as they could and took charge for the challenges they were assigned to, making the process of integrating all the functions easier.

4.Any significant difficulties and the steps taken to overcome them

Difficulties faced	***Correctives measures taken***
Conflicts of pins used when combining different pieces of codes together, resulting in the IR sensor to take in wrong values and not function as expected.	We first isolated the section of codes for IR sensor and tested it. After confirming that the problem does not lie within the IR code or the IR hardwares, we added in other pieces of code section by section and tested the code to see where the problem is at. Eventually we found out that both the IR sensor and the color sensor were using PIN3, and we managed to correct the pins and get the code working.
The battery level for the mbot is not kept constant during the experiments. This is especially the case when the battery is fully charged as the speed of the motor is significantly faster. This caused all our movement values to be off as we took measurements for the value when the battery level is around 80~90%.	After fully charging the battery, we will first let the battery discharge for around 10 minutes first such that the power of the battery is close to the level at which we take the measurements before we conduct the experiments. This made sure that the speed of our motor is always around a fixed value.
The values measured by the microphone were too small, and sometimes the values for high frequency cannot be detected.	We first tried to increase the gain of the op-amp. However, by doing that we realized that the microphone will catch in unwanted frequencies as well and this will cause the mBot to make wrong action. After trial and erroring different Rf and Ri values and still not getting an optimal measurement for the frequencies, we eventually found out that this

	problem can be resolved by moving the microphone as close to the speaker as possible. As a result, we secured the microphone to the front of the mBot and kept it up straight using blu-tack to amplify the frequency detected.
We faced tremendous difficulties when taking measurements for the frequency due to a variety of factors. Firstly, there is a lot of environmental noises and this made is especially difficult to test for 100~300Hz, as the mBot often picks up unwanted noises and treats it as low frequency being detected. This is made worse by the fact that there are always multiple groups testing for the frequency values inside the lab at all times, which created further confusion to our measured values.	We tried to measure the reference values for the mic outside the lab so that the noises in the lab won’t affect the values we take. We also learnt of the electronic noises produced in the lab and that more accurate results will be taken when the mBot is lifted off the table.