# CG1111 Lab Section B Group 2

# The A-maze-ing Race Written Report

Cynthia Lee Weng Yan A0190363H

Danish B Eddie A0183040U

Janel Ang Yee Huey A0187706U

Jerry Zhang Zhuoran A0187394M

Jess Teo Xi Zhi A0182668N

Nguyen Ngoc Linh Chi A0170767W

# 1. Implementation Details of Subsystems

## 1.1 Algorithm for Keeping mBot Straight

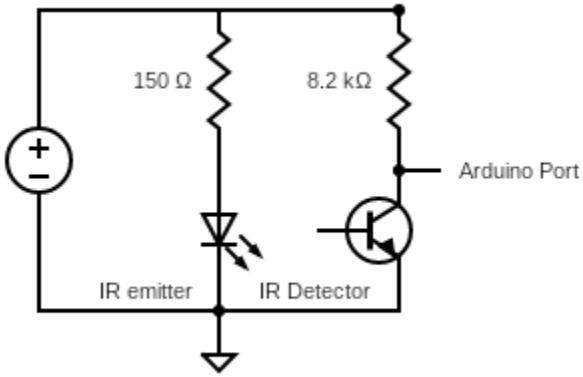


Figure 1.1.0 IR Circuit diagram

We implemented an external Arduino Library called PID Library, which we found when researching online on ways to let our mBot move in a straight line. The Arduino library was created by Brett Beauregard (https://playground.arduino.cc/Code/PIDLibrary). Utilizing 3 tuning parameters called Proportional, Integral and Derivative, the PID function aims to reduce any differences between the set point. We are unsure of the mathematical calculations that goes into the calculation but we were able to arrive at our values through trial and error.

```
PID leftPID(&inputLeft, &outputLeft, &setpointLeft, 0.5, 0.01, 0, DIRECT);
PID rightPID(&inputRight, &outputRight, &setpointRight, 0.5, 0.01, 0, DIRECT);
```

Figure 1.1.1 PID function

Based on our research online, the Proportional variable accounts for present deviations from the set point. The Integral variable accounts for past deviations from the set point and the Derivative variable accounts for future deviations from the set point. Through experimentation, we feel that the values P = 0.5, I = 0.01, D = 0 were the most optimal for our robot.

```
inputLeft = analogRead(LEFT_IR);
```

```
 inputRight = analogRead(RIGHT_IR);
 if ((inputLeft < 280) || (inputRight < 280)) {
   extremeIR();
 } else {
   leftPID.Compute();
   rightPID.Compute();
   speedLeft = -(outputRight * 2.2) + 250;
   speedRight = -(outputLeft * 2.2) + 250;
   move(1, speedLeft, speedRight);
 }
```

Figure 1.1.2 Decremental approach used to keep mbot in the centre

We implemented a decremental approach by setting the base speed at 250 and any corrections will be made by decrementing the motor speed of the opposite motor. For example, should the mBot be too close to the right wall, the PID function will compute the correctional value based on the difference between the current right IR readings and the base right IR value. This correctional value will then be used to decrease the speed of the left motor, causing the mBot to correct itself by turning to the left and avoid hitting the right wall.

```
void extremeIR() {
 if (inputLeft < 280) {
   move(1, 255, 165);
   delay(100);
 }
 else if (inputRight < 280) {
   move(1, 165, 255);
   delay(100);
 }
}
```

FIgure 1.1.3 Extreme IR function

However, we soon realised that relying on PID is not good enough as our mBot will swerve and react too slowly when it is too near a wall. Therefore, we implemented another function called extremeIR() that will steer the mBot away from the wall should the readings from the IR sensors reach a certain threshold. We arrived at the threshold and speed at which the motors should turn based on experimentation and calibration.

## 1.2 Audio Processing Circuit Design

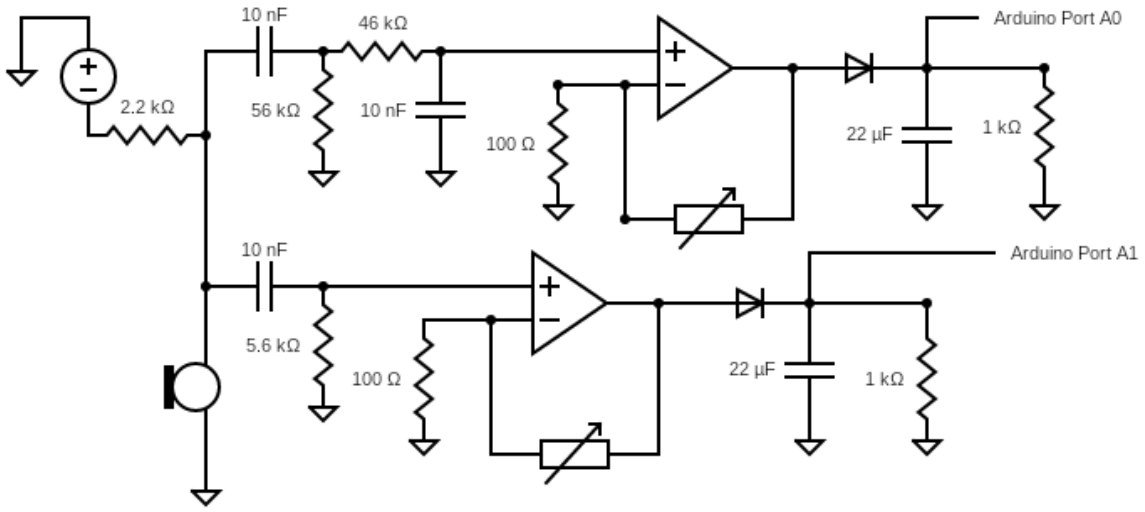


Figure 1.2.1 Audio Processing Circuit

The circuit consists of several parts. Firstly, the microphone is connected to the 5V power source via a 2.2kΩ resistor, to ensure that it is operating at an optimum of 2.5V.
Filters were built into the audio processing circuit in order to accurately capture sounds of the appropriate frequencies. For the 300 Hz signal, a band-pass filter consisting of a high-pass filter and a low-pass filter was required. For the 3000 Hz signal, a high-pass filter was required.
To obtain the necessary cut-off frequencies, the resistance (R) and capacitance (C) values were calculated using the following formula:

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

Using 10nF capacitors by default, the necessary resistances were calculated for each filter. 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.

| Sound | Filter required | Cut-off frequency/ Hz | RC Value/ΩF | Resistance value/Ω | Resistance value used/Ω | Actual Cut-off frequency/Hz |
|---|---|---|---|---|---|---|
| 300 Hz | High-pass | 280 | $5.68\times10^{-4}$ | 56841 | 56000 | 284 |
| | Low-pass | 320 | $4.97\times10^{-4}$ | 49735 | 46000 | 345 |
| 3000 Hz | High-pass | 2800 | $5.68\times10^{-5}$ | 5684 | 5600 | 2842 |

Figure 1.2.2 Table of values required.

The processed signals from each filter were passed into op-amps configured as non-inverting amplifiers, with a $R_i$ value of 100Ω and a variable resistor of maximum 10kΩ for $R_f$. Adjusting the variable resistor allows for tuning the gain of the amplifier and thus adjusting the output voltage to suitable values for later processing and analysis. The output of the op-amp was then connected to an envelope detector with a resistor-capacitor network, to produce an envelope of the signal to be read by the arduino.

## 1.3 Audio Processing Circuit Algorithm

20 readings were taken for each frequency, 3000 Hz and 300 Hz, and the averages were calculated. The ratio of the average reading of 3000 Hz to the average reading of 300 Hz was used to determine the calibration for the sound challenge.

If the ratio is more than 1 and less than 5, the mBot would execute a left turn. If the ratio is between 5 and 9, the mBot would execute a 180° turn. And if the ratio is more than 9, the mBot would execute a right turn. These ratios were determined through trial and error during calibration.

Should the ratio be less than 1, the mBot will determine that its 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 bot is black in colour.

```
void soundChallenge() {
 float reading_low = 0;
 float reading_high = 0;
 for (long i = 0; i < 20; i ++) {
   reading_low += analogRead(Reader300Hz);
   reading_high += analogRead(Reader3000Hz);
   delay(50);
 }
 float avghigh = reading_high/20; //Gets average reading for 3000Hz
 float avglow = reading_low/20; //Gets average reading for 300Hz
 double ratio = avghigh/(avglow);
 if (ratio > 1) {
   if (ratio <= 5){ //300 Hz higher
     turnLeft(speedLeft, speedRight);
   } else if ((ratio > 5) && (ratio < 9)) { //300Hz and 3000Hz same level
     turn180(speedLeft, speedRight);
   } else if (ratio >= 9) { //3000 Hz higher
     turnRight(speedLeft, speedRight);
   }
 } else {
   play();
 }
 delay(100);
}
```

Figure 1.3.1 Sound challenge algorithm

## 1.4 Colour Sensing Algorithm

For colour sensing algorithm, we started off by calibrating the set of RGB values for the white and black colour in the function SetBalance. We had to differentiate the range of values that it would sense between the colour white and black therefore we created greyDiff by taking the difference between the set of values obtained by white and black. Moving on, we proceed by attaining the set of RGB values for the respective colour papers given for the colour challenge, Red, Green, Blue and Orange. These set of RGB values are read in the colourArray.

```
* Green: 50, 133, 50 (Right turn)
* Red: 208, 38, 30 (Left turn)
* Blue: 45, 130, 160 (Two successive right turns in two grids)
* Black: -5 -5 -5 (Check for Ultrasonic/ Victory)
* White: 275, 275, 280 (Uturn in one grid)
* Orange: 222, 63, 38 (Two successive left turns in two grids)
```

Figure 1.4.1 Set of RGB values for the respective colours

However, obtaining the set of RGB values for each colour was not enough to get the mbot to successfully turn at the correct colour. We had to experimentally test out the readings of each colours to get the range of RGB values of each colour for the mbot to read in order to make the correct turns at the repective colour. The toughest part was reading the orange colour as the values are very close to Red. After multiple attempts, we then successfully obtain the right values and use a long if function, colourChecker, to read the respective colours. For red and green, we we had to calibrate the wheels to turn right and left respectively at the right speed. For blue, we created a function turnUright such that the mbot will make two successive right turns in two grids. Similarly for orange, we created the turnUleft function such that the mbot will make two successive left turn in two grids. The mbot will make a u-turn, turn180(), when it sense black and finally play a victory song when it senses black.

## 1.5 End of Maze Algorithm

At the end of the maze, the colour of the board on top is black in colour. When the mBot detects a black colour and no sound challenges, the mBot will play a victory tune of our choice.

```
MeBuzzer buzzer;

int melody[] = {NOTE_DS3, NOTE_AS3, NOTE_AS3, NOTE_GS3, NOTE_AS3, NOTE_GS3,
NOTE_FS3, NOTE_GS3, NOTE_GS3, NOTE_FS3, NOTE_DS3, NOTE_FS3};
int noteDurations[] = {4, 4, 8, 8, 4, 8, 8, 4, 8, 8, 8, 8};

void play() {
 for (int thisNote = 0; thisNote < 12; thisNote++) {
   int noteDuration = 1000 / noteDurations[thisNote];
   buzzer.tone(8, melody[thisNote], noteDuration);
   int pauseBetweenNotes = noteDuration * 1.10;
   delay(pauseBetweenNotes);
   // End of music
   buzzer.noTone(8);
 }
}
```

Figure 1.5.1 End Maze Algorithm

In order to get the mBot to play a song, we first got a range of definition for different musical notes based on resources we found online. After choosing our desired tune, we matched the notes to the music through trial and error. Once we have obtained the required musical notes for our tune, we stored the note values in the correct order of the tune into an array, melody[]. We configured the duration of the note of the tune and the optimal pauses between each note to mimic the exact rhythm of the original tune. We did this by using two parameters, noteDuration, which is defined as 1 second divided by the duration of the musical note, and pauseBetweenNotes, which is noteDuration * 1.10, respectively to configure and calculate them. After which, we stored the duration of each musical note in chronological order into an array, noteDurations[ ]. These are all done by hard-coding and repeating the original tune many times and listening closely to try and pick up the correct keys and beats.

# 2. Work division within the team

| | |
|---|---|
| Hardware - Sound Detector | Cynthia, Jess |
| Hardware - Setting up mBot | Janel, Jess, Danish, Cynthia |
| Hardware - IR Sensor | Danish, Janel |
| Software - Sound Detector | Jerry, Linh Chi, Cynthia, Jess |
| Software - Light Detector & IR Sensor | Jerry, Linh Chi, Jess |
| Software - mBot Movement | Linh Chi, Jerry |

We mainly split our workload based on hardware and software. The software team comprises Linh Chi and Jerry while the hardware team comprises of Jess, Danish, Cynthia and Janel. However, we didn't draw clear lines between hardware and software and always strived to help one another.

Initially, there were limited things that the software team could do, therefore we combined effort and strived to complete the IR sensor and sound detector circuits. However, multiple complications came up with the sound detector circuits, which will be further discussed under difficulties and corrective steps taken.

Later on, near the date of evaluation, we have managed to accomplish all hardware components, we combined our effort to help with the testing and calibration of the mBot to avoid collisions and turning at at the right angles.

All in all, everyone played an equal and important role in accomplishing this project, be it in terms of software or hardware.

# 3. Difficulties and Corrective Steps Taken

| Difficulties faced | Corrective steps taken |
|---|---|
| Audio circuit had rusty components such as resistors that affected the entire circuit, causing strange fluctuations in voltage when there was no sound. | We changed a new set of resistors which worked fine. We didn't realise that the resistor was not working properly initially until we went through extensive testing and changing of components. |
| Expansion board had a broken pin, resulting in one of the analog pins to constantly measure 5V. | We swapped the expansion board for a new one and the new board worked perfectly. |
| IR detectors burnt out/broke while we were calibrating. | We swapped the IR sensors and took extra care to prevent the sensor from burning out again by letting it cool down in in between tests. We also built a cardboard support to prevent the parts from breaking when colliding into walls. |
| Making the colour challenge and sound challenge work together. There were multiple instances whereby the calibration for colour sensor was off after successfully calibrating the sound, vice versa. | We put in extra effort and checked everything after we have calibrated something so as to ensure that everything was working as intended. We also took down notes so everyone was clear of the necessary changes required to calibrate. |
| Making the mBot turn at the right angle to avoid collisions. | We had to calibrate the speed of each wheels multiple times to ensure the mbot does not turn too much and collide into the walls of the maze. |
| The light ambience affected our colour calibration | By calibrating the mbot in different parts of the lab, we ensured our light sensor worked under different lighting. We also built a small cover around the MCore using black coloured construction paper to prevent light from external sources from interfering with colour sensing. |
| Calibration of detecting the right frequency was inconsistent, might be due to the different background noises | We tackled this by measuring the background noise before calibration and ensure there was no noise picked up by the mBot which might affect the reading. |

# 4. Overall Algorithm Flow

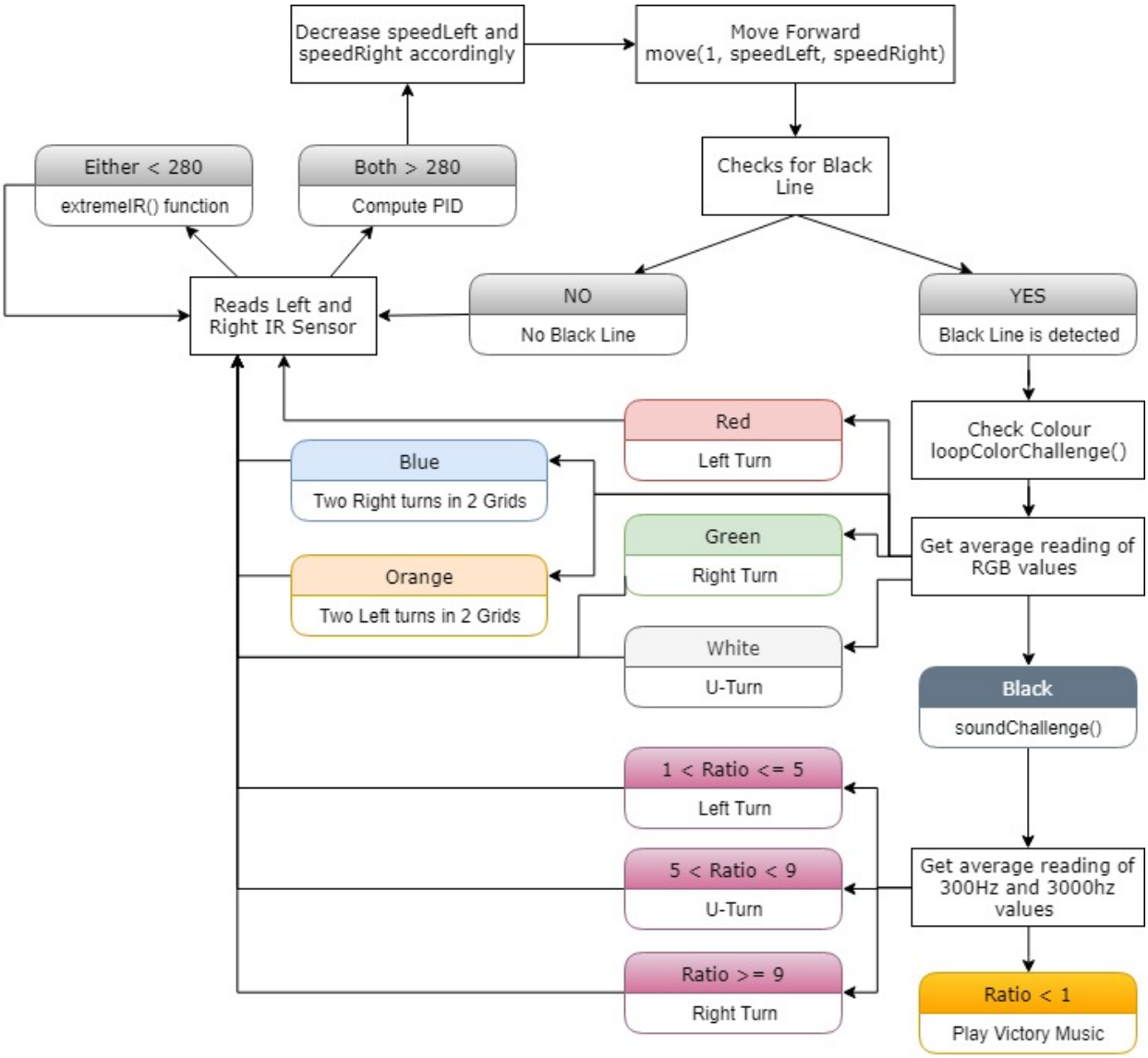


Figure 4 Algorithm Flowchart

```
void setup() {
 Serial.begin(9600);
 setupIRCalibrate();
 Serial.println("IR CALIBRATION DONE");
 //setup_Color_Challenge();
}
```

During setup(), the mBot will read values from the left and right IR sensor and set it as the setpoint for the PID function.

```
void loop() {
 if (isBlackLine() == 1) {
   Serial.println("BLACK LINE!!!");
   loopColorChallenge();
   delay(100);
 }
 inputLeft = analogRead(LEFT_IR);
 inputRight = analogRead(RIGHT_IR);
 if ((inputLeft < 280) || (inputRight < 280)) {
   extremeIR();
 } else {
   leftPID.Compute();
   rightPID.Compute();
   speedLeft = -(outputRight * 2.2) + 250;
   speedRight = -(outputLeft * 2.2) + 250;
   move(1, speedLeft, speedRight);
 }
}
```

Afterwards, it will check for a black line. Should no black line be found, the mBot will move forward, while constantly reading from the left and right IR sensor. If either values fall under the threshold of 280, the extremeIR function will kick in. If not, the PID function will continuously calculate outputRight and outputLeft, allowing the mBot to adjust its path accordingly in order to move in a straight line.

Upon reaching a black line, the mBot will start checking what is the colour of the board above it. Should the colour match Red, Green, Orange, Blue or White, the relevant actions will be taken by the mBot before continuing to move forward and checking IR values.

If the detected colour is black, the mBot will start soundChallenge() and attempt to detect for sounds. Based on the ratio that we have calibrated, it will determine its next course of action. Should there be no sound, the ratio detected will be lesser than 1 and the mBot will start playing the victory tune.

# 5. Appendix: Full Algorithm Used

```
#include "MeMCore.h"
#include <PID_v1.h>

/**
* Musical notes definition based on values found online
*/
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262

/**
* Ultrasonic Sensor definitions
* This is used to aid the colour sensor challenge when we are required to do
two
* successive left turns (orange) or two successive right turns (blue) in two
* grids. After turning the first time, mBot will rely on the ultrasonic sensor
* and detects if the wall is within a certain distance before turning again
*
* We have defined the sensing distance ULTRADISTANCE to be 12cm
* Initialised the ultrasonic sensor under the variable ultrasonicSensor reading
from PORT_1
*/
#define ULTRADISTANCE 12

MeUltrasonicSensor ultrasonicSensor(PORT_1);

/**
* Functions for Music
* Store musical notes in an array based on defined numbers found online
* Duration that each note is played is also stored in an array
* Chosen song: Thug Life, Snoop Dog LOL
*
```

```
* Initialise the onboard buzzer as variable buzzer
* melody[] is the array containing the notes of our victory tune
* noteDurations[] is the array holding the duration of each note
* noteDuration is defined as 1 second divided by the duration of that note.
* E.g. A quartet will be 1000/4
* Between each note there is a delay that we experimented to be *1.1 of the
noteDuration
* for best effect
*/
MeBuzzer buzzer;

int melody[] = {NOTE_DS3, NOTE_AS3, NOTE_AS3, NOTE_GS3, NOTE_AS3, NOTE_GS3,
NOTE_FS3, NOTE_GS3, NOTE_GS3, NOTE_FS3, NOTE_DS3, NOTE_FS3};
int noteDurations[] = {4, 4, 8, 8, 4, 8, 8, 4, 8, 8, 8, 8};

void play() {
 for (int thisNote = 0; thisNote < 12; thisNote++) {
   int noteDuration = 1000 / noteDurations[thisNote];
   buzzer.tone(8, melody[thisNote], noteDuration);
   int pauseBetweenNotes = noteDuration * 1.10;
   delay(pauseBetweenNotes);
   // End of music
   buzzer.noTone(8);
 }
}




/**
* Functions related to movement corrections
* Utilized for movement correction so that robot can stay in a straight line.
* Utilizes PID for minor motion adjustments.
* However, should IR sensor detect the wall to be too near (<300)
* we will utilize another function called extremeIR to correct our movement
back
* within safe regions
*/

#define LEFT_IR A2
#define RIGHT_IR A3

/**
* Definitions for the PID function to work
* setpointLeft and setpointRight is the initial values sensed by the
```

```
* left and right IR sensor respectively
*/
double setpointLeft, inputLeft, outputLeft;
double setpointRight, inputRight, outputRight;

// Values for Proportional, Integral and Derivative were found via trial and
error
// P = 0.5 accounts for present errors
// I = 0.01 accounts for past errors
// D = 0 accounts for future errors
PID leftPID(&inputLeft, &outputLeft, &setpointLeft, 0.5, 0.01, 0, DIRECT);
PID rightPID(&inputRight, &outputRight, &setpointRight, 0.5, 0.01, 0, DIRECT);

/**
* setupIRCalibrate function
* Used to calibrate the initial left and right distance of the mBot
* Takes 10 readings from left and right before calculating average
* Average value will be setpointLeft and setpointRight
* Upon calibration, PID function is set to AUTOMATIC
*
* @param[in] inputLeft is the readings from the left IR sensor
* @param[in] inputRight is the readings from the right IR sensor
* @param[out] setpointLeft is the calibrated base point of the left IR sensor
(to use with PID)
* @param[out] setpointRight is the calibrated base point of the right IR sensor
(to use with PID)
*/
void setupIRCalibrate() {
 for (int i = 0; i < 10; i++) {
   inputRight = analogRead(RIGHT_IR);
   inputLeft = analogRead(LEFT_IR);
   setpointLeft += inputLeft;
   setpointRight += inputRight;
   delay(100);
 }
 setpointLeft /= 10;
 setpointRight /= 10;
 Serial.println(setpointLeft);
 Serial.println(setpointRight);
 // turn PID on
 leftPID.SetMode(AUTOMATIC);
 rightPID.SetMode(AUTOMATIC);
}

/**
```

```
* extremeIR function
* Should readings from the left or right sensor be too low, mBot will
* manually move towards the right or left to avoid the wall
*
* @param[in] inputLeft is the readings from the left IR sensor
* @param[in] inputRight is the readings from the right IR sensor
*/
void extremeIR() {
 if (inputLeft < 280) {
   move(1, 255, 165);
   delay(100);
 }
 else if (inputRight < 280) {
   move(1, 165, 255);
   delay(100);
 }
}




/**
* Motor functions
* Responsible for changing the MeDCMotor values for movement
* Functions defined under here are used for any form of movement that is
determined by the color challenge or the sound challenge
*/

//time for turn left or right
#define TIME_TURN_MAX 240.0
#define SPEED_MAX 250.0

int speedLeft = SPEED_MAX;
int speedRight = SPEED_MAX;
MeDCMotor motor1(M1);
MeDCMotor motor2(M2);

/**
* move function
* We designed this function as we realised that in order to move forward or
backward
* the values for the left and right motor have to be inversed. Eg. Left is
positive
* and right is negative or vice versa.
* As such, utilizing an if else statement at the end, we are able to avoid
```

```
confusing ourselves
* when we wish our mBot to move in a certain direction
* We crunched some numbers and decided to change the delay of each turn
according to
* a constant and the maximum speed of our mBot.
*
* @param[in] direction is an indication of the direction in which we desire the
mBot to movement
* @param[in] speedLeft is the speed at which the left motor turns. Usually
determined via PID
* @param[in] speedRight is the speed at which the right motor turns. Usually
determined via PID
*/
void move(int direction, int speedLeft, int speedRight) {
 int leftSpeed = 0;
 int rightSpeed = 0;
 // 1 is move forward
 // 2 is move backward
 // 3 is turn left
 // 4 is turn right
 if (direction == 1) {
   leftSpeed = speedLeft;
   rightSpeed = speedRight;
 }
 else if (direction == 2) {
   leftSpeed = -speedLeft;
   rightSpeed = -speedRight;
 }
 else if (direction == 3) {
   leftSpeed = -speedLeft;
   rightSpeed = speedRight;
 }
 else if (direction == 4) {
   leftSpeed = speedLeft;
   rightSpeed = -speedRight;
 }
 motor1.run(M1 == M1 ? -(leftSpeed) : (leftSpeed));
 motor2.run(M2 == M1 ? -(rightSpeed) : (rightSpeed));
}

void turnLeft(int speedLeft, int speedRight) {
 move(3, speedLeft, speedRight);
 delay(TIME_TURN_MAX * SPEED_MAX / (speedLeft / 2 + speedRight / 2));
 move(1, speedLeft, speedRight);
 delay(200);
```

```
 stop();
}

void turnRight(int speedLeft, int speedRight) {
 move(4, speedLeft, speedRight);
 delay(TIME_TURN_MAX * SPEED_MAX / (speedLeft / 2 + speedRight / 2));
 move(1,speedLeft, speedRight);
 delay(100);
 stop();
}

void turn180(int speedLeft, int speedRight) {
 if (analogRead(LEFT_IR) > analogRead(RIGHT_IR)) {
   move(3, speedLeft, speedRight);
   delay(2 * TIME_TURN_MAX * SPEED_MAX / (speedLeft / 2 + speedRight / 2));
   stop();
 }
 else {
   move(4, speedLeft, speedRight);
   delay( 2 * TIME_TURN_MAX * SPEED_MAX / (speedLeft / 2 + speedRight / 2));
   stop();
 }
}

void turnULeft(int speedLeft, int speedRight) {
 turnLeft(speedLeft, speedRight);
 delay(50);
 while (ultrasonicSensor.distanceCm() > ULTRADISTANCE) {
   move(1, speedLeft, speedRight);
 }
 delay(50);
 move(3, speedLeft, speedRight);
 delay(50 + TIME_TURN_MAX * SPEED_MAX / (speedLeft / 2 + speedRight / 2));
 stop();
}

void turnURight(int speedLeft, int speedRight) {
 turnRight(speedLeft, speedRight);
 delay(50);
 while (ultrasonicSensor.distanceCm() > ULTRADISTANCE) {
   move(1, speedLeft, speedRight);
 }
 delay(50);
 move(4, speedLeft, speedRight);
 delay(50 + TIME_TURN_MAX * SPEED_MAX / (speedLeft / 2 + speedRight / 2));
```

```
 stop();
}

void stop() {
 motor1.run(0);
 motor2.run(0);
}

/**
* Black Line Sensing
* Sensor is attached to PORT_2 of the mBot expansion
* Senses if there is a black line
* Black line is an indication of a challenge, sound or colour
* We set it such that only when both sensor are inside a black line then
* the mBot will stop.
*
* @param[out] returns 1 if black line is sensed. Else it will return 0
*/
MeLineFollower linefollower_2(PORT_2);

int isBlackLine() {
 if ((linefollower_2.readSensors() == S1_IN_S2_IN) ) {
   stop();
   return 1;
 }
 return 0;
}

/**
* For Color Sensing Challenge
* whiteArray holds measured values for white paper
* blackArray holds measured values for black paper
* greyArray holds the difference between white and black
* colorArray will be used to hold measurements of measured
* color paper
*
*
* Green: 50, 133, 50 (Right turn)
* Red: 208, 38, 30 (Left turn)
* Blue: 45, 130, 160 (Two successive right turns in two grids)
* Black: -5 -5 -5 (Check for Ultrasonic/ Victory)
* White: 275, 275, 280 (Uturn in one grid)
* Orange: 222, 63, 38 (Two successive left turns in two grids)
*/
```

```
// Define time delay before the LED is ON
#define LED_RGBWait 20
// Define time delay before the next RGB colour turns ON to allow LDR to
stabilize
#define RGBWait 20
// Define time delay before taking another LDR reading
#define LDRWait 10
#define TIMES 20

MeLightSensor lightsensorTOP(PORT_6);
MeRGBLed rgbled(PORT_7, 2);

float colourArray[] = {0, 0, 0};
float whiteArray[] = {479.00, 334.00, 432.00};
float blackArray[] = {232.00, 160.00, 207.00};
float greyDiff[] = {246.00, 174.00, 225.00};
char colourStr[3][5] = {"R= ", "G= ", "B= "};

void setup_Color_Challenge() {
 turnOffLed(0);
 setBalance();
}

void setBalance() {
 // Calibrates for whiteArray
 Serial.println("Put White Sample For Calibration ...");
 buzzer.tone(8, NOTE_G4, 250);
 delay(5000);  // Delay 5 seconds to get white sample ready
 turnOffLed(0);
 for (int i = 0; i <= 2; i++) {
   // Turn on LED Red, Green, Blue at a time
   turnOnOffRGBLed(i, 0);
   whiteArray[i] = getAvgReading(TIMES); //Calibrates whiteArray based on
average reading
   turnOffLed(0);
   delay(LED_RGBWait);
 }
 Serial.println("Put Black Sample For Calibration ...");
 buzzer.tone(8, NOTE_C5, 250);
 delay(5000); // Delay 5 seconds to get black sample ready
 for (int i = 0; i <= 2; i++) {
   // Turn on LED Red, Green, Blue at a time
   turnOnOffRGBLed(i, 0);
   blackArray[i] = getAvgReading(TIMES); // Calibrates blackArray based on
average reading
```

```
   turnOffLed(0);
   delay(LED_RGBWait);
   // greDiff is the difference between the maximum possible and the minimum
possible values
   greyDiff[i] = whiteArray[i] - blackArray[i];
 }
 Serial.println("Colour Sensor Is Ready.");
 buzzer.tone(8, NOTE_E5, 250);
}

void turnOnOffRGBLed(int i, int light) {
 if (i == 0) {
   turnOnRedLed(light);
 }
 else if (i == 1) {
   turnOnGreenLed(light);
 }
 else {
   turnOnBlueLed(light);
 }
}

void turnOnRedLed(int light) {
 rgbled.setColor(light, 255, 0, 0);
 rgbled.show();
 delay(LED_RGBWait);
}

void turnOnGreenLed(int light) {
 rgbled.setColor(light, 0, 255, 0);
 rgbled.show();
 delay(LED_RGBWait);
}

void turnOnBlueLed(int light) {
 rgbled.setColor(light, 0, 0, 255);
 rgbled.show();
 delay(LED_RGBWait);
}

void turnOnWhite(int light) {
 rgbled.setColor(light, 255, 255, 255);
 rgbled.show();
 delay(LED_RGBWait);
}
```

```
void turnOffLed(int light) {
 rgbled.setColor(light, 0, 0, 0);
 rgbled.show();
 delay(LED_RGBWait);
}

void loopColorChallenge() {
 for (int c = 0; c <= 2; c++) {
   Serial.print(colourStr[c]);
   turnOnOffRGBLed(c, 0);
   colourArray[c] = getAvgReading(TIMES);
   // the average reading returned minus the lowest value divided by the
maximum possible range,
   // multiplied by 255 will give a value between 0-255, representing the value
for the current reflectivity
   colourArray[c] = (colourArray[c] - blackArray[c]) / (greyDiff[c]) * 255;
   turnOffLed(0);
   delay(10);
   Serial.println(int(colourArray[c]));
 }
 colorChecker();
}

/**
* Green: 50, 133, 50 (Right turn)
* Red: 208, 38, 30 (Left turn)
* Blue: 45, 130, 160 (Two successive right turns in two grids)
* Black: -5 -5 -5 (Check for Ultrasonic/ Victory)
* White: 275, 275, 280 (Uturn in one grid)
* Orange: 222, 63, 38 (Two successive left turns in two grids)
*/

void colorChecker() {
 // Red
 if ((colourArray[0] > 180) && (colourArray[1] < 45) && (colourArray[2] < 60))
{
   turnLeft(speedLeft, speedRight);
   return;
 }
 // Green
 if ((colourArray[0] < 70) && (colourArray[1] > 100) && (colourArray[2] < 70))
{
   turnRight(speedLeft, speedRight);
   return;
```

```
 }
 // Blue
 if ((colourArray[0] < 70) && (colourArray[1] > 100) && (colourArray[2] > 130))
{
   turnURight(speedLeft, speedRight);
   return;
 }
 // White
 if ((colourArray[0] > 200) && (colourArray[1] > 200) && (colourArray[2] >
200)) {
   turn180(speedLeft, speedRight);
   return;
 }
 // Optional Orange
 if ((colourArray[0] > 200) && (colourArray[1] > 45) && (colourArray[2] < 60))
{
   turnULeft(speedLeft, speedRight);
   return;
 }
 // Black
 if ((colourArray[0] < 40) && (colourArray[1] < 40) && (colourArray[2] < 40)) {
   soundChallenge();
   return;
 }
}

// This fucntion is used to get the average reading
int getAvgReading(int times) {
 int reading;
 int total = 0;
 for (int i = 0; i < times; i++) {
   reading = lightsensorTOP.read();
   total = reading + total;
   delay(LED_RGBWait);
 }
 return total / times;
}




/**
* For Sound Challenge
* Required to read in sound frequencies of 3000Hz and 300Hz from the mic
* S2 is 3000Hz
```

```
* S1 is 300Hz
* Read Voltage values from both analogPins and compare their Voltage
* If 300Hz louder, Left turn
* If 3000Hz louder, Right turn
* If both the same amplitude, U-turn same grid
*/

#define Reader3000Hz A1
#define Reader300Hz A0

int volt3000, volt300;

void soundChallenge() {
 float reading_low = 0;
 float reading_high = 0;
 for (long i = 0; i < 20; i ++) {
   reading_low += analogRead(Reader300Hz);
   reading_high += analogRead(Reader3000Hz);
   delay(50);
 }
 float avghigh = reading_high/20; //Gets average reading for 3000Hz
 float avglow = reading_low/20; //Gets average reading for 300Hz
 double ratio = avghigh/(avglow);
 Serial.print("300Hz:");
 Serial.print((avglow)/1023*5000);
 Serial.print("       3000Hz:");
 Serial.print(avghigh/1023*5000);
 Serial.print("       ");
 Serial.print(ratio,7);
 Serial.println("");
 // Numbers are determined through trial and error
 if (ratio > 1) {
   if (ratio <= 5){
     turnLeft(speedLeft, speedRight);
   } else if ((ratio > 5) && (ratio < 9)) {
     turn180(speedLeft, speedRight);
   } else if (ratio >= 9) {
     turnRight(speedLeft, speedRight);
   }
 } else {
   play();
 }
 delay(100);
}
```

```
void setup() {
 Serial.begin(9600);
 setupIRCalibrate();
 Serial.println("IR CALIBRATION DONE");
 //setup_Color_Challenge();
}

/**
* Checks for black line
* If no black line, continue moving forward. and correct itself using PID
functions
* If black line, checks for colour first
* If colour is black, checks for sound.
* If no sound, play victory music. Else, follow accordingly to certian sound or
colour
*/
void loop() {
 if (isBlackLine() == 1) {
   Serial.println("BLACK LINE!!!");
   loopColorChallenge();
   delay(100);
 }
 inputLeft = analogRead(LEFT_IR);
 inputRight = analogRead(RIGHT_IR);
 if ((inputLeft < 280) || (inputRight < 280)) {
   extremeIR();
 } else {
   leftPID.Compute();
   rightPID.Compute();
   speedLeft = -(outputRight * 2.2) + 250;
   speedRight = -(outputLeft * 2.2) + 250;
   move(1, speedLeft, speedRight);
 }
}
```