```
#include <MeMCore.h> //include Mbot library
#define TURNING_TIME_MS 410 // The time duration (ms) for turning
#define TIMEOUT 1400 // Max microseconds to wait before moving on
#define SPEED_OF_SOUND 340 // Speed of sound to be 340m/s
#define ULTRASONIC 12 // we use pin 12 (port 1) for the ultrasonic sensor
#define IR_SENSOR A2 //Pin S1 of Port 3 is used to read voltage readings from the IR
sensor
#define LDR_SENSOR A3 //Pin S2 of Port 3 is used to read voltage readings from the LDR
sensor
#define LED_INPUT_1 A0 //Pin S1 of Port 4 is used to control input 1A on LS139P chip
#define LED_INPUT_2 A1 //Pin S2 of Port 4 is used to control input 1B on LS139P chip
#define LDRWait 10 // Define time delay before taking another LDR reading (in ms)
#define RGBWait 100 // Define time delay (in ms) before changing to another LED colour

/**
  * Declaration of constant variables for different notes to be played in celebratory
tune
  * The numbers correspond to the frequency of each note
  */

const int c = 261;
const int d = 294;
const int e = 329;
const int f = 349;
const int g = 391;
const int gS = 415;
const int a = 440;
const int aS = 455;
const int b = 466;
const int cH = 523;
const int cSH = 554;
const int dH = 587;
const int dSH = 622;
const int eH = 659;
const int fH = 698;
const int fSH = 740;
const int gH = 784;
const int gSH = 830;
const int aH = 880;

MeBuzzer buzzer;// create the buzzer object
MeDCMotor leftMotor(M1); // assigning leftMotor to port M1
MeDCMotor rightMotor(M2); // assigning RightMotor to port M2
MeLineFollower lineFinder(PORT_2); // assigning lineFinder to RJ25 port 2

/**
  * setting up default motor speeds to 220/255
  */

uint8_t motorSpeed_left = 220;
uint8_t motorSpeed_right = 220;
```

```
/**
  * Values for colour sensor calibration and measurements are stored as ints
  * whiteArray, blackArray, greyDiff are used to calibrate the LDR and all subsequent
colour measurements will be based off the values stored in these arrays
  * colourArray holds the final RGB values measured by the LDR to be used for colour
identification
  */

int colourArray[] = {0,0,0};
int whiteArray[] = {952,963,938};
int blackArray[] = {864,926,917};
int greyDiff[] = {85,36,19};

/**
  * The numbers in the bracket specify the frequency (represented by note) and the
duration (ms)
  * This function calls the Mcore to play Star Wars Theme
  */

void play_tune() {
  buzzer.tone(a, 500);
  buzzer.tone(a, 500);
  buzzer.tone(a, 500);
  buzzer.tone(f, 350);
  buzzer.tone(cH, 150);
  buzzer.tone(a, 500);
  buzzer.tone(f, 350);
  buzzer.tone(cH, 150);
  buzzer.tone(a, 650);

  delay(500);

  buzzer.tone(eH, 500);
  buzzer.tone(eH, 500);
  buzzer.tone(eH, 500);
  buzzer.tone(fH, 350);
  buzzer.tone(cH, 150);
  buzzer.tone(gS, 500);
  buzzer.tone(f, 350);
  buzzer.tone(cH, 150);
  buzzer.tone(a, 650);

  delay(500);

  buzzer.tone(aH, 500);
  buzzer.tone(a, 300);
  buzzer.tone(a, 150);
  buzzer.tone(aH, 500);
  buzzer.tone(gSH, 325);
  buzzer.tone(gH, 175);
  buzzer.tone(fSH, 125);
  buzzer.tone(fH, 125);
  buzzer.tone(fSH, 250);

  delay(325);
```

```
  buzzer.tone(aS, 250);
  buzzer.tone(dSH, 500);
  buzzer.tone(dH, 325);
  buzzer.tone(cSH, 175);
  buzzer.tone(cH, 125);
  buzzer.tone(b, 125);
  buzzer.tone(cH, 250);

  delay(350);

  buzzer.tone(f, 250);
  buzzer.tone(gS, 500);
  buzzer.tone(f, 350);
  buzzer.tone(a, 125);
  buzzer.tone(cH, 500);
  buzzer.tone(a, 375);
  buzzer.tone(cH, 125);
  buzzer.tone(eH, 650);

  delay(500);

  buzzer.tone(f, 250);
  buzzer.tone(gS, 500);
  buzzer.tone(f, 375);
  buzzer.tone(cH, 125);
  buzzer.tone(a, 500);
  buzzer.tone(f, 375);
  buzzer.tone(cH, 125);
  buzzer.tone(a, 650);

  buzzer.noTone();
}

/**
  * Function to move the robot forward/straight
  */

void go_straight(){

  leftMotor.run(-motorSpeed_left); // Negative: wheel turns anti-clockwise
  rightMotor.run(motorSpeed_right); // Positive: wheel turns clockwise
}

/**
  * Function that tells robot to turn left
  */

void turn_left(){
  leftMotor.run(motorSpeed_left); // Positive: wheel turns clockwise
  rightMotor.run(motorSpeed_right); // Positive: wheel turns clockwise
}

/**
  * Function that tells robot to turn right
```

```
  */

void turn_right(){
  leftMotor.run(-motorSpeed_left); // Negative: wheel turns anti-clockwise
  rightMotor.run(-motorSpeed_right); // Negative: wheel turns anti-clockwise
}

/**
  * Function that tells robot to stop moving
  */

void stop_motors(){
  leftMotor.stop(); // Stop left motor
  rightMotor.stop(); // Stop right motor
}

/**
  * Function that tells motor to go slightly right
  */

void adjust_left(){
  leftMotor.run(-(motorSpeed_left+35)); // Negative: wheel turns anti-clockwise
  rightMotor.run(motorSpeed_right-35); // Positive: wheel turns clockwise
}

/**
  * Function that tells motor to go slightly left
  */

void adjust_right(){
  leftMotor.run(-(motorSpeed_left-35)); // Negative: wheel turns anti-clockwise
  rightMotor.run(motorSpeed_right+35); // Positive: wheel turns clockwise
}

/**
  *  Function to u-turn the robot on the spot (orange waypoint)
  */

void u_turn_on_spot() {
  turn_right();
  delay(2*TURNING_TIME_MS-130);
}

/**
  * Function to make 2 consecutive left turns (purple)
  */

void two_left_turns(){
      turn_left();
      delay(TURNING_TIME_MS-30);
      go_straight();
      delay(830);
      turn_left();
      delay(TURNING_TIME_MS+5);
}
```

```
/**
  *  Function to make 2 consecutive right turns (blue)
  */

void two_right_turns(){
      turn_right();
      delay(TURNING_TIME_MS-30);
      go_straight();
      delay(850);
      turn_right();
      delay(TURNING_TIME_MS+5);
}

/**
  * Controlling the digital pins of red, blue, green LEDs and the IR emitter
  * At any one time, only one of the LEDs or the IR emitter will be turned on with the
others turned off
  */

void turn_on_red(){
  digitalWrite(LED_INPUT_1, HIGH);
  digitalWrite(LED_INPUT_2, LOW);
}

void turn_on_blue(){
  digitalWrite(LED_INPUT_1, HIGH);
  digitalWrite(LED_INPUT_2, HIGH);
}

void turn_on_green(){
  digitalWrite(LED_INPUT_1, LOW);
  digitalWrite(LED_INPUT_2, HIGH);
}

void turn_on_IR(){
  digitalWrite(LED_INPUT_1, LOW);
  digitalWrite(LED_INPUT_2, LOW);
}

/**
  * This function is used to calibrate the LDR to adjust to ambient lighting, it is
used to determine the values to be stored in the respective arrays to be used for
colour measurements
  */

void setBalance(){
  //Set white balance
  Serial.println("Put White Sample For Calibration ...");
  delay(5000);
  //Delay for five seconds for getting white sample ready
  //Scan the white sample
  //Turn on each LED one at a time, wait for 100ms before scanning 5 times to take
average, and set the reading for each colour - red, green and blue - to whiteArray,
turn off all LEDs by turning on IR emitter at the end
```

```
    turn_on_red();
    delay(RGBWait);
    whiteArray[0] = getAvgReading(5);
    delay(RGBWait);
    turn_on_green();
    delay(RGBWait);
    whiteArray[1] = getAvgReading(5);
    delay(RGBWait);
    turn_on_blue();
    delay(RGBWait);
    whiteArray[2] = getAvgReading(5);
    delay(RGBWait);
    turn_on_IR();
    //Helps us to record the values stored to whiteArray
    for(int i=0; i<3; i++){
     Serial.println("White Indexes are:");
     Serial.println(whiteArray[i]);
    }

  //Set black balance
  Serial.println("Put Black Sample For Calibration ...");
  delay(5000);
  //Delay for five seconds for getting black sample ready
  //Scan the black sample
  //Turn on each LED one at a time, wait for 100ms before scanning 5 times to take
average, and set the reading for each colour - red, green and blue - to blackArray,
turn off all LEDs by turning on IR emitter at the end
    turn_on_red();
    delay(RGBWait);
    blackArray[0] = getAvgReading(5);
    delay(RGBWait);
    turn_on_green();
    delay(RGBWait);
    blackArray[1] = getAvgReading(5);
    delay(RGBWait);
    turn_on_blue();
    delay(RGBWait);
    blackArray[2] = getAvgReading(5);
    delay(RGBWait);
    turn_on_IR();

    //Helps us to record the values stored to blackArray and greyDiff
    for(int i=0; i<3; i++){
     Serial.println("Black Indexes are:");
     Serial.println(blackArray[i]);
     //The difference between the maximum and the minimum gives the range
     greyDiff[i] = whiteArray[i] - blackArray[i];
     Serial.println("Grey Ranges are:");
     Serial.println(greyDiff[i]);
    }

    delay(7000);
    //Delay 7 seconds for transferring of robot to maze for test run during our
trials, this is not used for the actual run
  }
```

```
/**
  * Find the average reading for the requested number of times of scanning LDR
  */

int getAvgReading(int times){
  int reading;
  int total =0;
  //Take the reading as many times as requested and add them up, delaying 10ms in
between taking each reading
  for(int i = 0;i < times;i++){
     reading = analogRead(LDR_SENSOR);
     total = reading + total;
     delay(LDRWait);
  }
//Calculate the average and return it
  return total/times;

}

/**
  * Function to tell the robot to scan for colour and take action depending on the
colour interpreted
  * The function takes in the previously declared colourArray as it needs to store the
final RGB readings in the array
  */
void color_sense(int colourArray[]){
  int red_readings[12];
  int green_readings[8];
  int blue_readings[8];
  int red_total=0;
  int green_total=0;
  int blue_total=0;
  int red_average;
  int green_average;
  int blue_average;

  //Taking one sample at a time, total of 12 samples for R
  turn_on_red();
  for(int i = 0; i< 12; i++){
    red_readings[i] = (analogRead(LDR_SENSOR) - blackArray[0])/(greyDiff[0])*255;

  Serial.println("Red readings are:");
  Serial.println(red_readings[i]);
  }

//Taking one sample at a time, total of 8 samples for G
  turn_on_green();
  for(int i = 0; i< 8; i++){
    green_readings[i] = (analogRead(LDR_SENSOR) - blackArray[1])/(greyDiff[1])*255;

  Serial.println("Green readings are:");
  Serial.println(green_readings[i]);
  }
```

```
  //Taking one sample at a time, total of 8 samples for B
  turn_on_blue();
  for(int i = 0; i< 8; i++){
    blue_readings[i] = (analogRead(LDR_SENSOR) - blackArray[2])/(greyDiff[2])*255;

  Serial.println("Blue readings are:");
  Serial.println(blue_readings[i]);
  }

  //Take average for the last 6 readings for R
  for (int i=6; i<12; i++){
    red_total += red_readings[i];
    red_average = red_total/6;
  }

  //Take average for the last 4 readings for B
  for (int i=4; i<8; i++){
    blue_total += blue_readings[i];
    blue_average = blue_total/4;
  }

  //Take average for the last 4 readings for G
  for (int i=4; i<8; i++){
    green_total += green_readings[i];
    green_average = green_total/4;
  }

  //Store the 3 average values at their respective positions in colourArray
  colourArray[0] = red_average;
  colourArray[1] = green_average;
  colourArray[2] = blue_average;

  //To check the final RGB values recorded
  for(int i=0; i<3; i++){
    Serial.println(colourArray[i]);
  }

}

/**
  * Setting up the digital pins and begin serial communication with computer
  */

void setup() {
  Serial.begin(9600);
  pinMode(LED_INPUT_1, OUTPUT);
  pinMode(LED_INPUT_2, OUTPUT);
  pinMode(ULTRASONIC, OUTPUT);
  //setBalance();
  //setBalance() is commented out as we no longer use it in the actual run, it is only
used for calibration purposes during trials
}

void loop() {
  uint8_t sensorState = lineFinder.readSensors();
```

```
  //Declare sensor state of line sensor

  /**
  * If black line detected, stop the robot and begin colour sensing
  * If not, keep the robot going and adjust when too close to wall
  */

  if(sensorState == S1_IN_S2_IN){
    stop_motors();

    //read the RGB values of the current colour into colourArray
    color_sense(colourArray);

    //if red, turn left
    if(colourArray[0]>240 && colourArray[1]>0 && colourArray[1]<110 &&
colourArray[2]>0 && colourArray[2] < 135){
      turn_left();
      delay(TURNING_TIME_MS-50);
    }

    //if green, turn right
    else if(colourArray[0] > 50 && colourArray[0] <160   && colourArray[1]>120 &&
colourArray[1]<245 && colourArray[2] > 50 && colourArray[2] < 175){
      turn_right();
      delay(TURNING_TIME_MS-40);
    }

    //if orange, turn 180 degrees in same grid
    else if(colourArray[0]>230 && colourArray[1]>110 && colourArray[1]<190 &&
colourArray[2] >0 && colourArray[2] < 130){
      u_turn_on_spot();
    }

    //if purple, 2 left turns in 2 grids
    else if(colourArray[0]>190 && colourArray[0]<240 && colourArray[1]> 100 &&
colourArray[1]<200 && colourArray[2] >150){
      two_left_turns();
    }

    //if light blue, 2 right turns in 2 grids
    else if(colourArray[0]>120 && colourArray[0]<190 && colourArray[1]>160 &&
colourArray[1]<240 && colourArray[2] >190){
      two_right_turns();

    }

    //if white, play star wars theme
    else if (colourArray[0]>250 && colourArray[1]>250 && colourArray[2]>200){
      play_tune();
    }

    //if unable to recognise colour,delay 50ms and try again
    else {
      delay(50);
    }
```

```
  } else {
    //The ultrasonic sensor sends out a pulse and returns the time taken to receive
the reflected pulse, if timeout duration is exceeded, 0 is returned
    digitalWrite(ULTRASONIC, LOW);
    delayMicroseconds(2);
    digitalWrite(ULTRASONIC, HIGH);

    delayMicroseconds(10);
    digitalWrite(ULTRASONIC, LOW);

    pinMode(ULTRASONIC, INPUT);
    long duration = pulseIn(ULTRASONIC, HIGH, TIMEOUT);
    //distance of wall in cm from ultrasonic sensor is calculated using the duration
    double distance_cm = duration / 2.0 / 1000000 * SPEED_OF_SOUND * 100;

    /**
    * Take voltage reading of IR detector with IR emitter off (blue is lit by default)
    * Turn on IR emitter, let it stabilise for 20ms, take new voltage reading of IR
detector and find the difference between this value and the previous one
    * Emitter is then turned off by turning blue LED back on
    */

    int vb = analogRead(IR_SENSOR);
    turn_on_IR();
    delay(20);
    int vd = analogRead(IR_SENSOR);
    turn_on_blue();

    /**
    * If the difference between the 2 IR detector readings is above 30, it suggests
the mBot is dangerously close to the right wall and need to turn slightly left
    * If not in danger of hitting right wall, check if the ultrasonic sensor is
deducing the mBot is within 9cm of left wall
    * If within 9cm of left wall, turn slightly right, if not, the mBot is safe on
both sides and can proceed straight
    */

  if((vb-vd)>30){
      adjust_right();
      delay(10);

    } else if (distance_cm > 0 && distance_cm < 9){
        adjust_left();
        delay(5);

    } else {
        go_straight();
    }
  }

}
```