CG1111A - Engineering Principles and Practices I

A-maze-ing Race Project Report

Lab B02 - Team 3A

Kuek Yeau Hao, Jonathan - A0258485M
Leong Deng Jun - A0254459U
Jeffinson Darmawan - A0259229U
Leong Xuan Wei, Max - A0254473B

13 November 2022

Content Page

Introduction

In this race, our team is challenged to solve a maze in the shortest possible time. To do so, our team has developed an mBot capable of traversing the maze by ensuring that our mBot moves consistently straight, accurately detecting a series of coloured papers and turning successfully according to the programme we have created. This report provides detailed explanation on the various electrical subsystems utilised, some difficulties faced and how we overcome them to arrive at our final implementation.

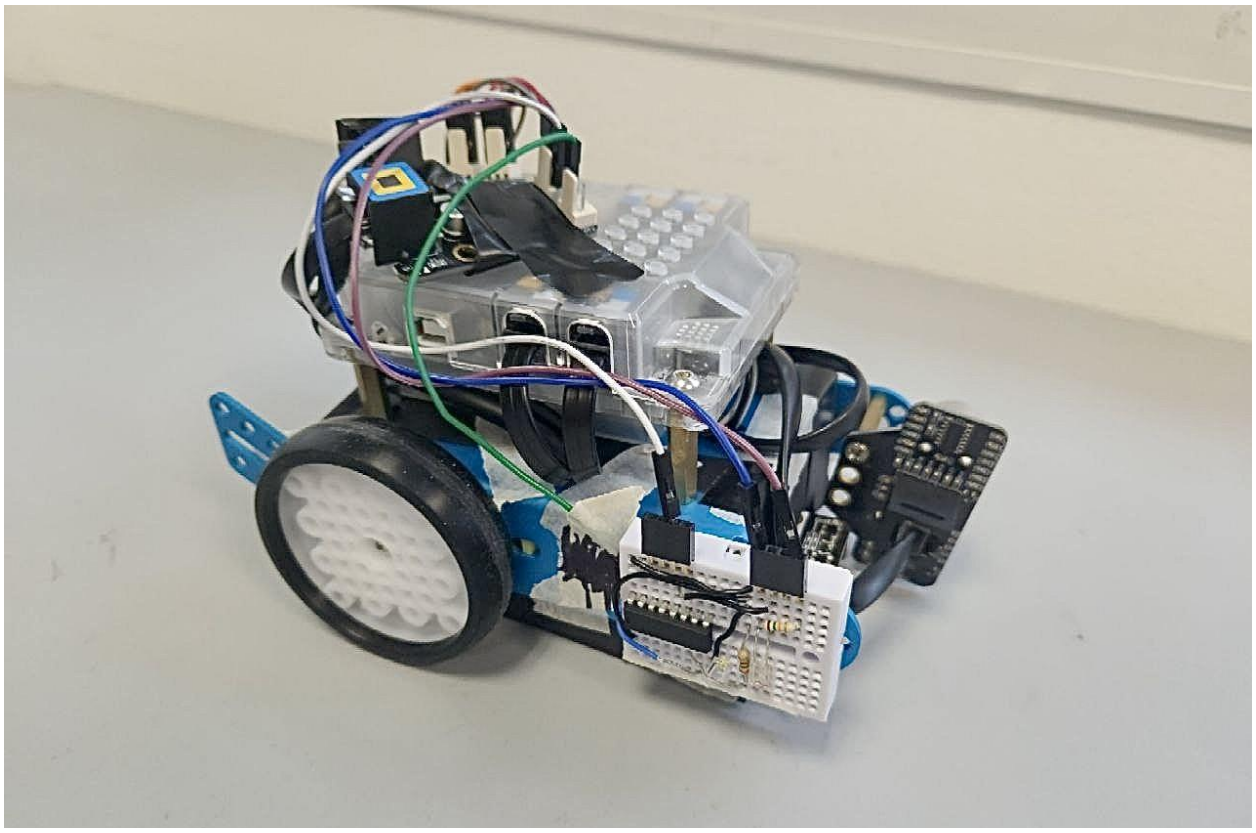


Figure 1: mBot Overview

Chapter 1 Overall Algorithm

The flowchart below shows how our mBot goes about solving the maze.

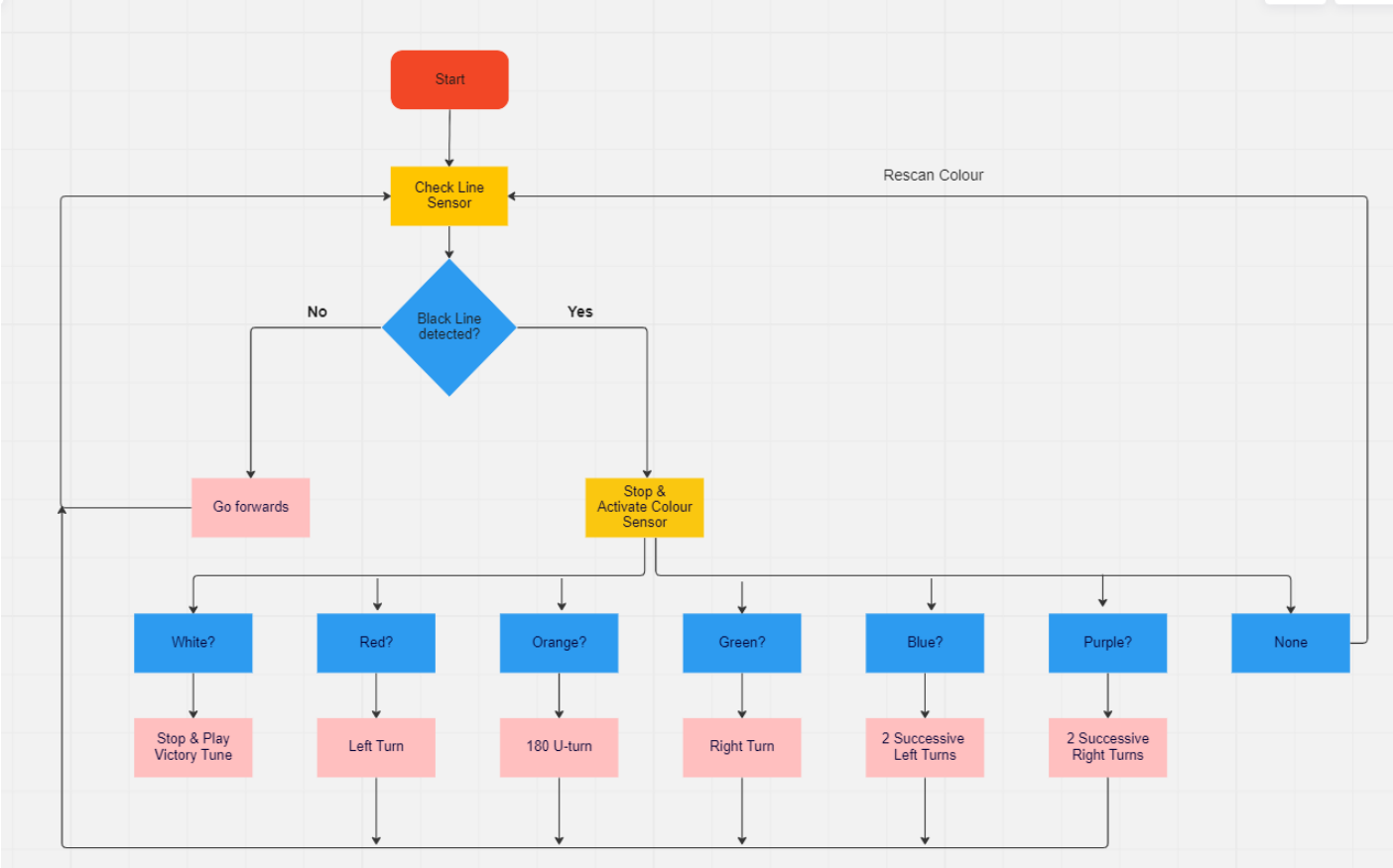


Figure 2: Flow chart of the general algorithm

```
void loop()
{
  int sensorState = lineFinder.readSensors();
  if (sensorState == S1_IN_S2_IN)
  {
    stop(300);
    int colour = detect_colour();
    setColor(colour);
    turn(colour);
  }
  else
  {
    move_maze(targetSpeed, last_error);
  }
}
```

Figure 3: Code snippet for the main loop

Our code also makes use of constants and various helper functions (included in the appendix) which helps improve neatness and readability of the code.

1.1 Movement

Upon experimenting, we realised that the motor speeds of both motors differ slightly which causes the mBot to significantly veer off to one side when travelling for a long distance within the maze. Thus, we had to look for a solution which keeps the robot straight with the help of the ultrasonic and infrared sensors.

One basic solution would be an on-off controller to adjust the movement based on the distance of the robot from the wall as depicted below.

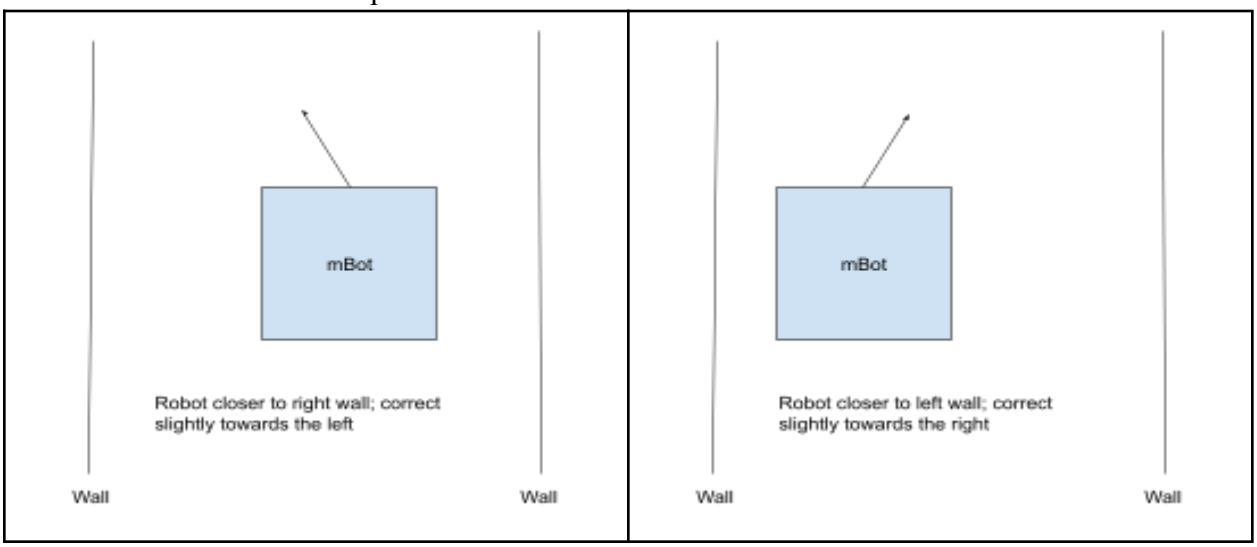


Figure 4: Illustration of how a basic On-Off controller works based on robot distance to wall

However, this does not fulfil the mission objective of having the robot move as straight as possible as the robot will move in a zig-zag manner. Such methods of corrections may also lead it to arriving at the colour checkpoints at awkward angles, which may cause the robot to be unable to execute its turns properly and thus, bump into walls after turning.

1.2 PID Control System

To solve the problem above, we settled on the PID closed-loop control system. PID stands for Proportion, Integral and Derivative controller, and it is a control-loop mechanism that employs sensor feedback to help the mBot move as straight as possible within the maze.

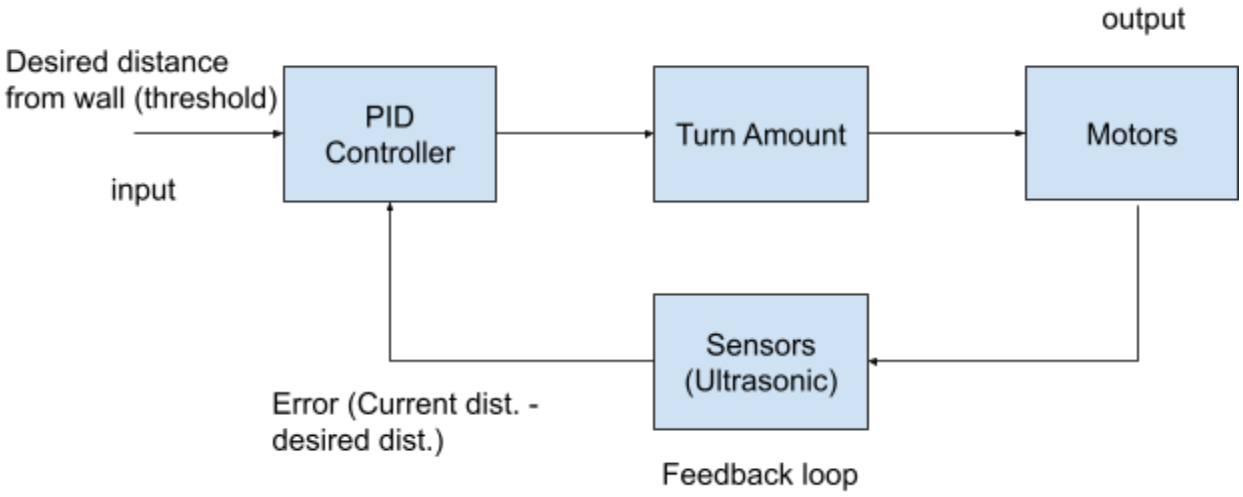


Figure 5: Flowchart depicting the closed-loop feedback control system

```
float ultra = get_ultra_distance(); // check if u
if (ultra >= 3.0 && ultra <= 17.0)
{
  float error = ultra - THRESHOLD;
  float derivative = error - last_error;
  float turn = (error * kp) + (derivative * kd);
  leftMotor.run(-(targetSpeed - turn));
  rightMotor.run(targetSpeed + turn);
  last_error = error;
}
```

Figure 6: Code snippet of the PD Controller system

Essentially, the Proportionality part of the control system controls the motor speed such that the speed is directly proportional to the distance that the robot has deviated from the threshold distance, which is the desired distance that we want the robot to travel from the wall (ideally in the middle).

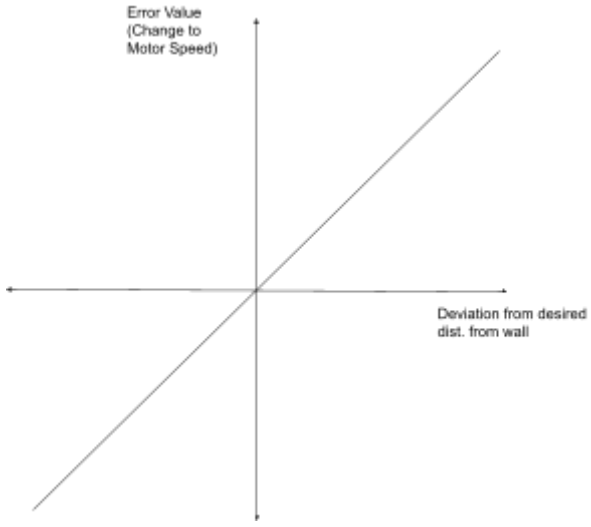


Figure 7: Graph representation depicting how the error output changes with deviation

This error is multiplied by a proportionality constant, Kp, which controls the how large the robot will attempt to correct. The greater the multiplier, the bigger the changes and the robot will 'stabilise' faster. However, it may make more violent changes that causes it to veer off course.

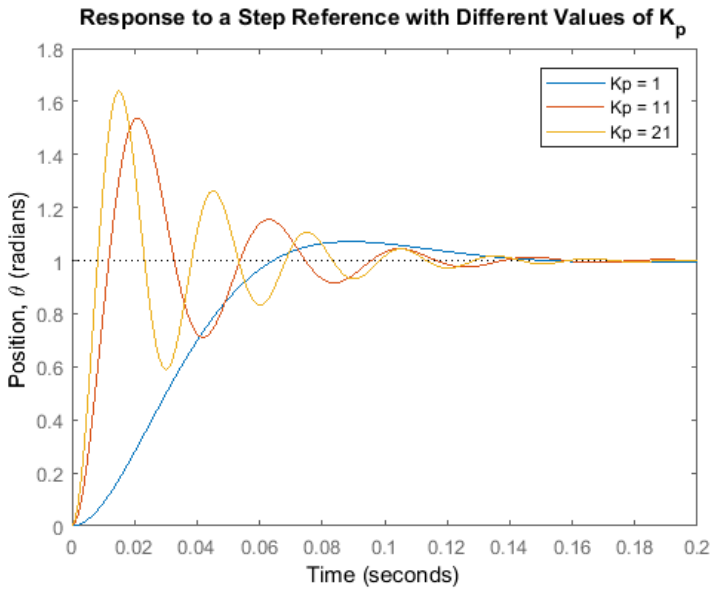


Figure 8: Graph depicting how different constant values affect the output error

In summary, the Derivative part of the system is the difference between the current and previous error. It aids in correcting the robot by assuming that the previous change in error is equal to the current change in value, thereby predicting the next change in error. This aids in smoothening out the Proportionality control as it decreases the overshoot amount and the time needed to stabilise. This is also controlled by a Derivative constant, Kd.

As for the Integral part of the system, it aids in helping the mBot to stabilise at the exact desired threshold as the Proportionality control may be unable to do so due to steady-state error. This occurs when the calculated error is so small that it is unable to give the feedback needed for the final 'boost' towards the threshold.

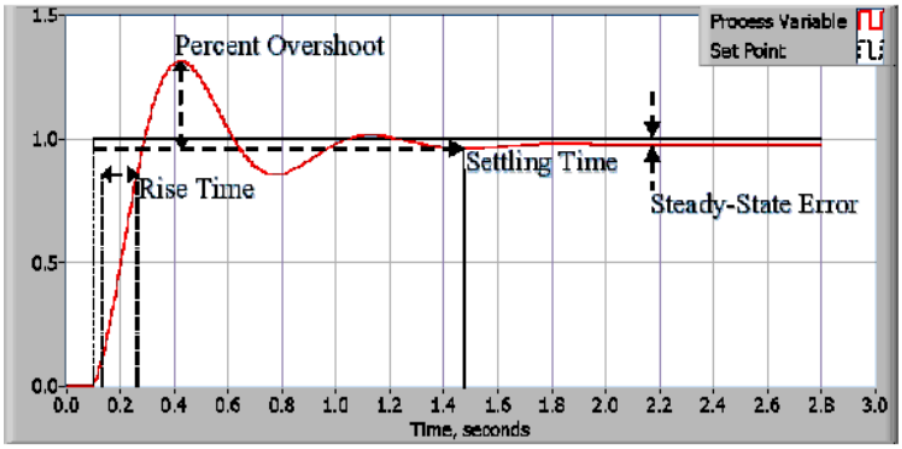


Figure 9: Graph of a standard P-controller with steady-state error

In practice, we decided not to implement the integral part and stick to a 'PD' controller as we felt that the robot is already moving forward smoothly enough and we did not need to be super precise making sure the robot stays exactly in the middle. This also reduces the complexity and time taken to tune our PID system.

1.3 Turning

Colour	Movements
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
White	Stop and play victory tune

```
// colour sensor logic for turning based on decoded colour
void turn(int colour)
{
  if (colour == 1) // RED - left turn
  {
    move(-targetSpeed, targetSpeed);
    delay(TURNING_TIME_MS);
    stop(200);
  }
  if (colour == 2) // GREEN - right turn
  {
    move(targetSpeed, -targetSpeed);
    delay(TURNING_TIME_MS);
    stop(200);
  }
  if (colour == 3)// ORANGE - UTURN
  {
    move(-targetSpeed, targetSpeed);
    delay((TURNING_TIME_MS * 2) - 50);
    stop(200);
  }
  if (colour == 4)// PURPLE 2 successive lefts
  {
    move(-targetSpeed, targetSpeed);
    delay(TURNING_TIME_MS);
    stop(200);
    move(targetSpeed, targetSpeed);
    delay(1035); // go forward for 1 tile
    //2nd left
    stop(300);
    move(-targetSpeed, targetSpeed);
    delay(TURNING_TIME_MS - 10);
    stop(200);
  }
  if (colour == 5) // BLUE - 2 successive rights
  {
    move(targetSpeed, -targetSpeed);
    delay(TURNING_TIME_MS);
    stop(200);
    move(targetSpeed, targetSpeed);
    delay(1035); // go forward for 1 tile
    // 2nd right
    stop(300);
    move(targetSpeed, -targetSpeed);
    delay(TURNING_TIME_MS - 10);
    stop(200);
  }
  if (colour == 6) // WHITE
  {
    play_tune();
  }
}
```

Figure 10: Code snippet of turning based off colour detected

Chapter 2 Infra-red Detector

2.1 Implementation details

We were also tasked to build an Infra-Red (IR) detector circuit to compliment our ultrasonic sensor in helping the mBot avoid the walls in the maze. It was installed on the right side of our mBot as shown below.

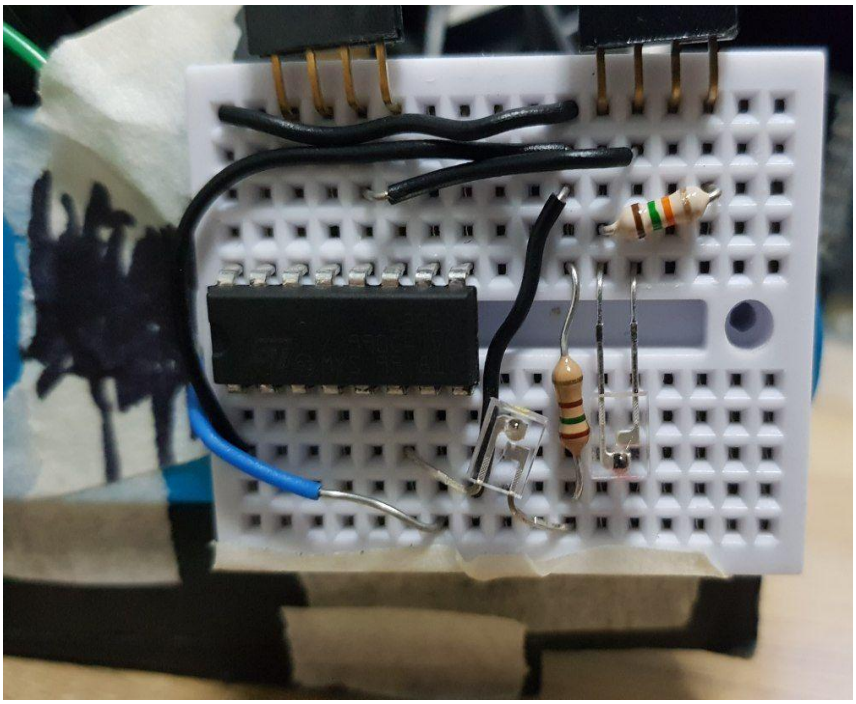


Figure 11: Actual IR circuit

When characterising our IR sensor, we realised that we are unable to get a clear linear relationship between the analog readings and the distance from the maze wall as shown below. Furthermore, the ambient IR from the surroundings as well as from our bodies also caused the readings taken to range inconsistently each time we tried to re-calibrate as we had to be in close proximity with the sensor.

	IR VALUE (0-1024)	Corresponding Distance (cm)
2	966	3
3	958	4
4	952	5
5	947	6
6	943	7
7	940	8
8	937	9
9	935	10
10	933	11
11	932	12
12	931	13

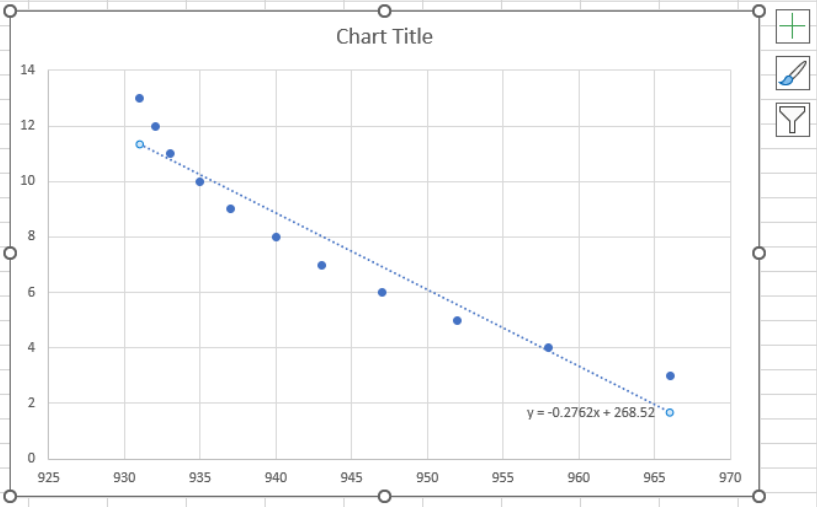


Figure 12: Characterisation of IR sensor done on Excel

2.2 Coding

As such, we decided to make the ultrasonic sensor the primary method for wall detection as it is much more reliable. The IR sensor would only be used when the ultrasonic is unable to detect a wall within range successfully on its side. Given that we cannot obtain a clear linear relationship and that fact that the readings tend to be inconsistent, we chose not to implement the PID control system for the IR sensor, opting to do minor corrections like the on-off controller to avoid the wall. In the event that it returns readings that are out of the pre-programmed range, the mBot will move straight by default (both motors running at the same speed).

```
int getIRAverage(int times){
//find the average reading for the requested number of times of reading IR sensor
  int reading;
  int total = 0;
//take the reading as many times as requested and add them up
  for(int i = 0;i < times; i++){
     reading = analogRead(A2);
     total = reading + total;
     delay(LDRWait);
  }
  return total/times;
}
float get_ir_distance()
{
  float values = getIRAverage(5);
  Serial.println(values);
  if (values >= 950 && values <= 980) // too close to right side
  {
    return 5;
  }
  else if (values <= 935 && values >= 900)
  {
    return 10; // too close to left side
  }
  return 8; // robot is roughly in the middle OR readings are out of range
}
```

Figure 13: Code snippet of the IR detection

2.3 Circuitry

We constructed the circuit according to the design shown below. The value obtained from the IR sensor is read from pin A2 on the mCore.

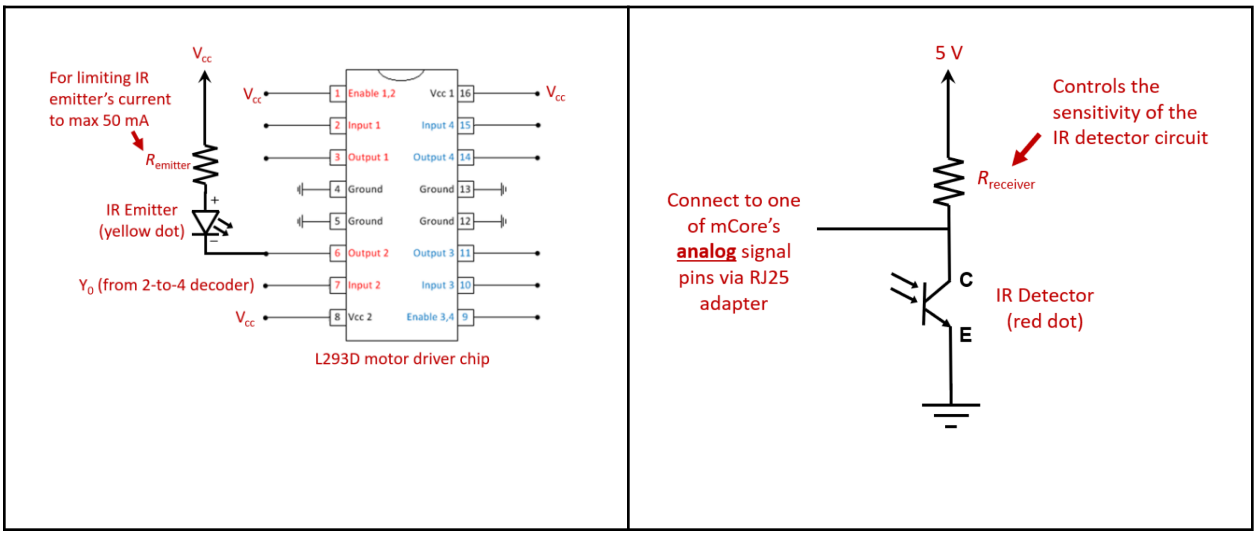


Figure 14: IR-detector circuit

Remmiter: 150kΩ , RReceiver : 15kΩ

In order to limit the current to 50mA, using Ohm's Law we calculated that the Remmiter had to be approximately 150kΩ, as the supply voltage was 5V.
To increase the IR sensor's responsiveness, we chose the resistance of the RReceiver to be 15kΩ, as this would give us a more spread out reading to calibrate the IR sensor for it to work reliably.

Chapter 3: Colour sensor

3.1 Implementation Details

We used 2 main components to construct our colour sensor: 3 RGB LEDs lamps and a Light-dependent Resistor (LDR). We placed the breadboard holding the components underneath our mBot. When the mBot line follower detects the black strip of paper, the robot will stop and activate the colour sensor. We cycle through and turn on each RGB LED separately via the truth table logic of the HD74LS139P 2-to-4 decoder, taking the average readings of the reflected light with each colour before passing the data through to the function that determines the colour detected. Based on the colour detected, the mBot will execute the appropriate turns or play the victory tune if white is detected, thereby ending our run.

3.2 Circuitry

The HD74LS139P 2-to-4 decoder IC input has a limit of maximum 8mA. We used Ohm's law to calculate the resistor values required for R_G, R_B and R_R. As the intensity of the red LED is higher than the other colours, we used a higher resistor value to reduce the current.

R_G: 680Ω R_B: 680Ω R_R: 800 Ω

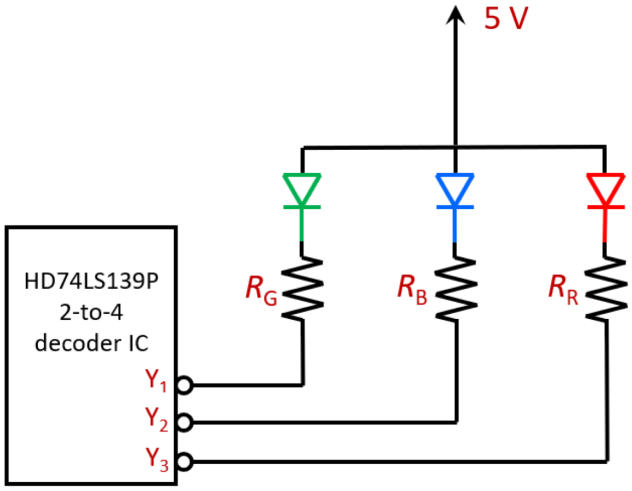


Figure 15: Connecting the red, green and blue LEDs

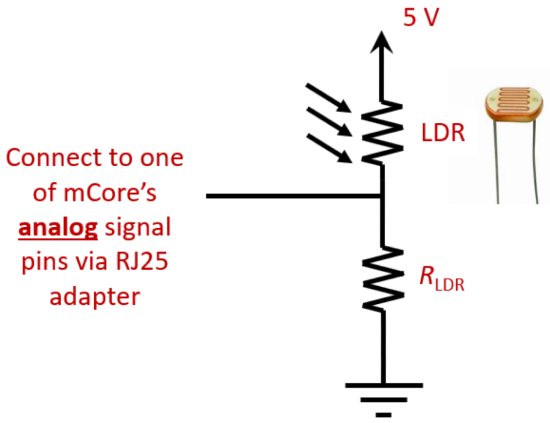


Figure 16: Connecting the LDR to the circuit

In previous studios, the R_{LDR} used was 100kΩ. To adapt it to work with our circuit, we experimented with different values or RLD to find a resistor value which would provide us with workable values to differentiate between the colour papers in the maze. We had much difficulty

trying to calibrate the Colour sensor. We started of with resistor values greater than 100kΩ but the Red, Green and Blue reading was too similar on the coloured papers for the sensor to work reliably. Instead of trying resistors less than 100kΩ, we kept going higher and spent a long time before realising that we should try a lower resistor value. When we finally experimented with lower resistor values and the R, G, B was more distinct. We finally decided on a R_{LDR} value of 10kΩ as it gives us R, G and B values which are distinct for different colour papers, enabling us to reliably detect the correct colour

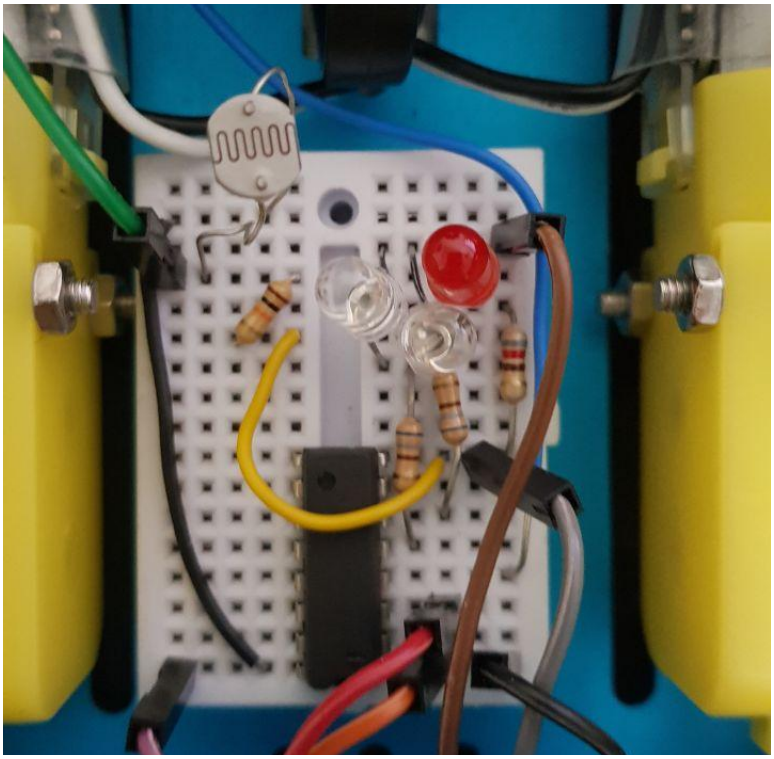


Figure 17: Actual colour sensor circuit

3.3 Coding

Prior to the run, we calibrate the colour sensor by obtaining the RGB values sensed by the colour sensor from the serial monitor on each colour paper. As different colours give different ranges of readings of RGB values, we can use this distinct characteristic to differentiate between the different colours. After finding an appropriate range of values for all the colours, we then implement in the code, whereby when the sensor detects this specific range of values, it identifies the colour correctly, and turns in the correct direction.

We also made use of the on-board LED of the mBOT and coded it to light up according to which colour it detects. This aids in debugging the colour sensor without needing to open up the serial monitor to observe the range of values that the LDR is detecting.

```
// debug function that prints the avg readings read from LDR
// 4-2 adaptor truth table logic to get respective LEDs to light up
// L/H BLUE
// H/L GREEN
// H/H RED
int detect_colour()
{
   analogWrite(A0, 255);
   analogWrite(A1, 255);
   delay(RGBWait);
   int red = getAvgReading(10);
   delay(RGBWait);

   analogWrite(A0, 255);
   analogWrite(A1, 0);
   delay(RGBWait);
   int green = getAvgReading(10);
   delay(RGBWait);

   analogWrite(A0, 0);
   analogWrite(A1, 255);
   delay(RGBWait);
   int blue = getAvgReading(10);
   delay(RGBWait);
   Serial.print("R: ");
   Serial.println(red);
   Serial.print("G: ");
   Serial.println(green);
   Serial.print("B: ");
   Serial.println(blue);
   int colour = get_colour(red, green, blue);
   return colour;
}
```

Figure 18: Code snippet of the program used to obtain RGB values via manipulating truth table of the decoder

	A	B	C	D	E	F
1	RED	ORANGE	GREEN	BLUE	PURPLE	WHITE
2	R: 553	R: 555	R: 255	R: 348	R: 431	R: 566
3	G: 684	G: 759	G: 769	G: 816	G: 762	G: 873
4	B: 340	B: 347	B: 419	B: 620	B: 534	B: 655
5	1	3	2	5	4	6
6	R: 552	R: 555	R: 255	R: 348	R: 432	R: 566
7	G: 684	G: 759	G: 769	G: 816	G: 762	G: 873
8	B: 340	B: 346	B: 419	B: 620	B: 534	B: 655
9	1	3	2	5	4	6
10						
11						

Figure 19: Screenshot of readings of RGB values obtained during calibration

```
// determine colour detected based on raw LDR value detected
int get_colour(int R, int G, int B)
{
  if (R > 500 && G > 810 && B > 600)
  {
    return 6; //white
  }
  if (R > 500 && (G > 640 && G < 730) && B > 315)
  {
    return 1; //red
  }
  if (R > 500 && G >= 730 && B < 400)
  {
    return 3; //orange
  }
  if (R < 300 && G >= 700 && B > 385)
  {
    return 2; //green
  }
  if (R < 400 && G >= 750 && B > 590)
  {
    return 5; //blue
  }
  if (R > 405 && G > 680 && B > 500)
  {
    return 4; //purple
  }
  return 7; //error :(
}
```

Figure 20: Code snippet of the logic used to determine colour detected

3.4 Calibration Improvements

However, we realised that the readings we got were very inconsistent, and we could not come up with the correct values to consistently differentiate between the colours. Hence to ensure that the RGB readings are constant, we created a “curtain” around the bottom of the robot to reduce ambient light from reaching our system. With minimal ambient light, we can find a consistent range of values from the LDR to determine which direction our robot will turn. We also colored the white tape black so as to further minimise reflected white light.

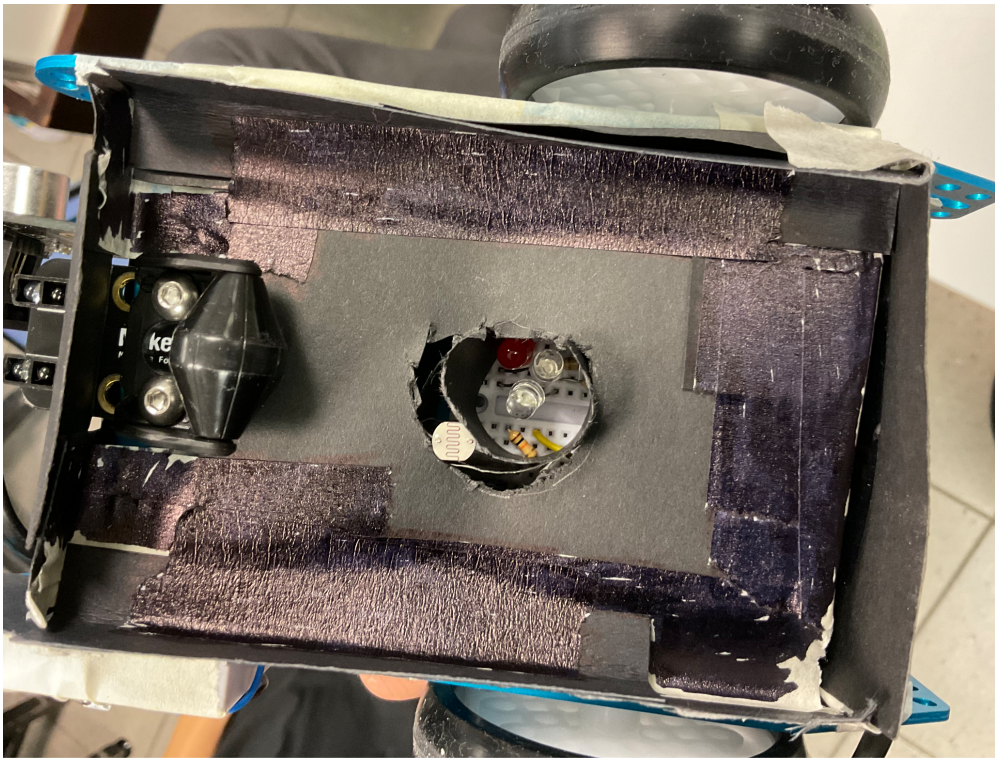


Figure 21: The “Curtain” around our colour sensor

Chapter 4: Ultrasonic Sensor

4.1 Position

We have mounted the ultrasonic sensor at the front part of the mBot, further inward towards the centre of the mBot and facing the left side of the mBot. Since the ultrasonic sensor operates within a range of 3 cm - 400 cm, the position of the ultrasonic sensor ensures that the distance between the ultrasonic sensor is not less than 3 cm from the left wall. We positioned the sensor closer to the body of the mBot so as to reduce the size of the robot to prevent it from coming into contact with the wall when turning.

To prevent short-circuiting, we used a bolt so that the solder points on the Makeblock ultrasonic sensor are not in contact with the metal chassis of the mBot.

During our test runs, we noticed that the ultrasonic sensor of the robot would occasionally hit the wall as it was making a u-turn. To resolve this, we mounted the ultrasonic sensor closer to the body of the robot and made some minor adjustments to the code to increase the distance between the robot and the wall as it moved. With this the robot no longer hit the wall during u-turns.

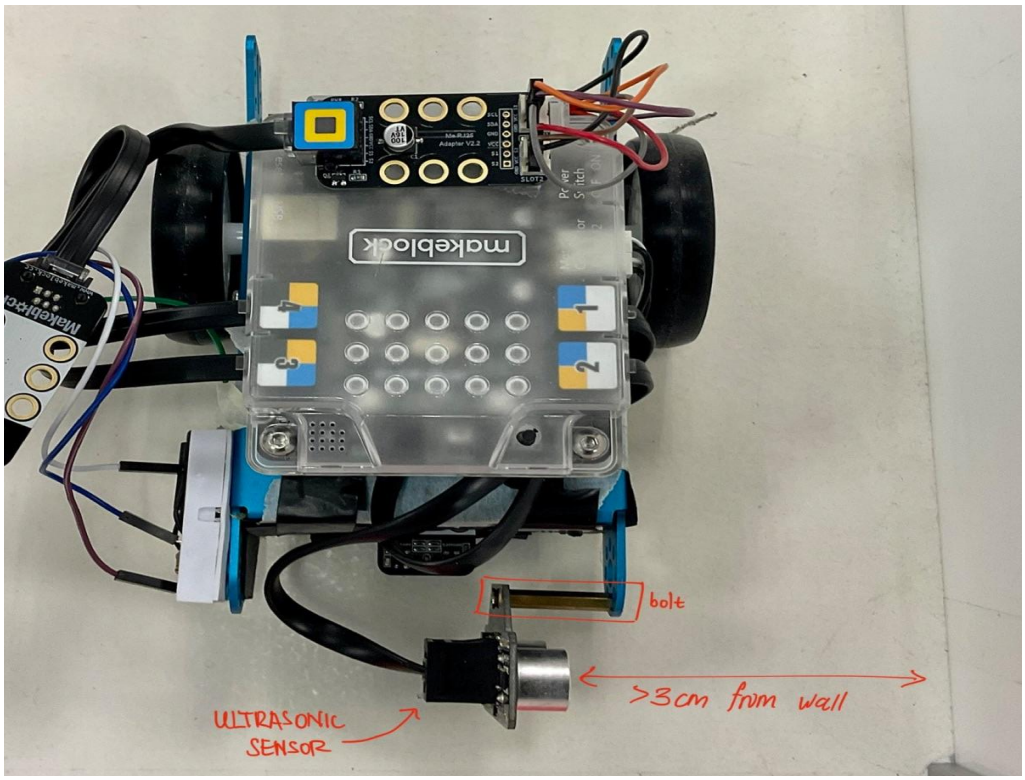


Figure 22: Position of the ultrasonic sensor

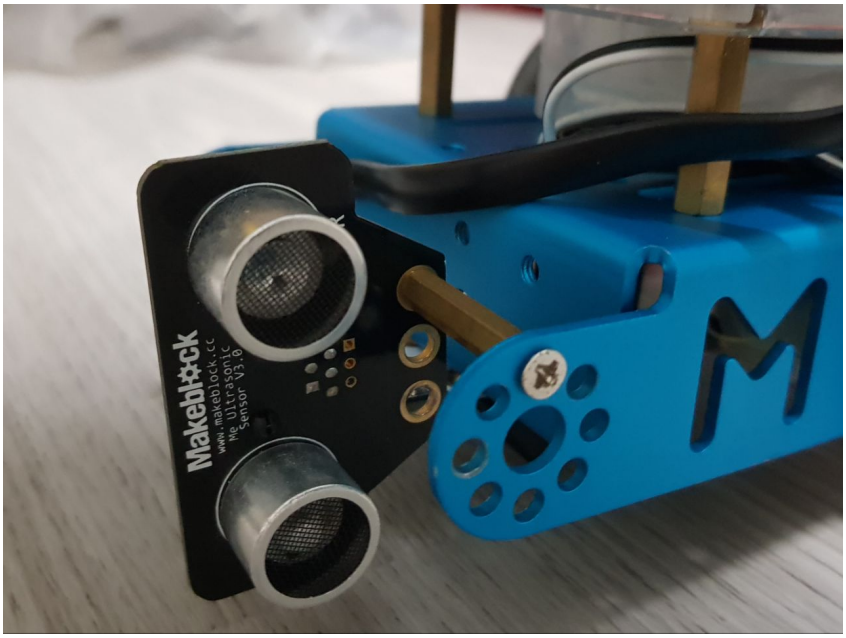


Figure 23: Ultrasonic Sensor

4.2 Circuitry

We connected the ultrasonic sensor to port 2 of the mCore, which corresponds to digital pin 10.

4.3 Coding

Our ultrasonic sensor implements the Proportional Integral Derivative (PID) control system. PID control provides a continuous variation of output within a control loop feedback mechanism to accurately control the process as described above in Section 1.2. The code attached below shows how the ultrasonic values are processed to obtain the distance from the wall accurately.

```
// function that processes and returns distance sensed from ultra
float get_ultra_distance()
{
 pinMode(ULTRASONIC, OUTPUT);
 digitalWrite(ULTRASONIC, LOW);
 delayMicroseconds(2);
 digitalWrite(ULTRASONIC, HIGH);
 delayMicroseconds(10);
 digitalWrite(ULTRASONIC, LOW);

 pinMode(ULTRASONIC, INPUT);
 long duration = pulseIn(ULTRASONIC, HIGH, TIMEOUT);
 float distance = duration / 2.0 / 1000000 * SPEED_OF_SOUND * 100;
 Serial.println(distance);
 return distance;
}
```

Figure 24: Code snippet of obtaining the distance read from the ultrasonic

Chapter 5 Victory Tune

5.1 Implementation details

Upon completion of the maze, the mbot would play a victory tune. For the tune, we decided to use the chorus of the song “Never gonna give you up” by Rick Astley.

```
void play_tune()
{
  buzzer.tone(466, 200);
  buzzer.tone(466, 100);
  buzzer.tone(415, 100);
  buzzer.tone(415, 100);
  buzzer.tone(698, 300);
  buzzer.tone(698, 300);
  buzzer.tone(622, 600);
  buzzer.tone(466, 100);
  buzzer.tone(466, 100);
  buzzer.tone(415, 100);
  buzzer.tone(415, 100);
  buzzer.tone(622, 300);
  buzzer.tone(622, 300);
  buzzer.tone(554, 300);
  buzzer.tone(523, 100);
  buzzer.tone(466, 200);
  buzzer.tone(554, 100);
  buzzer.tone(554, 100);
  buzzer.tone(554, 100);
  buzzer.tone(554, 100);
  buzzer.tone(554, 300);
  buzzer.tone(622, 300);
  buzzer.tone(523, 300);
  buzzer.tone(466, 100);
  buzzer.tone(415, 200);
  buzzer.tone(415, 200);
  buzzer.tone(415, 200);
  buzzer.tone(622, 400);
  buzzer.tone(554, 800);
  buzzer.tone(466, 100);
  buzzer.tone(466, 100);
  buzzer.tone(415, 100);
  buzzer.tone(415, 100);
  buzzer.tone(698, 300);
  buzzer.tone(698, 300);
  buzzer.tone(622, 600);
  buzzer.tone(466, 100);
  buzzer.tone(466, 100);
  buzzer.tone(415, 100);
  buzzer.tone(415, 100);
  buzzer.tone(831, 300);
  buzzer.tone(523, 300);
  buzzer.tone(554, 300);
  buzzer.tone(523, 100);
  buzzer.tone(466, 200);

  buzzer.tone(554, 100);
  buzzer.tone(554, 100);
  buzzer.tone(554, 100);
  buzzer.tone(554, 100);

  buzzer.tone(554, 300);
  buzzer.tone(622, 300);
  buzzer.tone(523, 300);
  buzzer.tone(466, 100);
  buzzer.tone(415, 200);
  buzzer.tone(0, 200);
  buzzer.tone(415, 200);
  buzzer.tone(622, 400);
  buzzer.tone(554, 800);

  buzzer.noTone();
}
```

Figure 25: Victory tune code

Chapter 6: Work Distribution

Contribution	**Name**
Program Code	Leong Deng Jun, Kuek Yeau Hao, Jonathan, Jeffinson Darmawan, Leong Xuan Wei, Max
Movements (Moving straight & turning based on colour)	Leong Deng Jun, Kuek Yeau Hao, Jonathan, Jeffinson Darmawan, Leong Xuan Wei, Max
Colour Sensor & IR Sensor (Hardware)	Leong Deng Jun, Kuek Yeau Hao, Jonathan, Jeffinson Darmawan, Leong Xuan Wei, Max
Victory Tune	Leong Xuan Wei, Max

Appendix

Complete Code

Constants Declaration

```
#include "MeMCore.h"

/* MBOT LIBRARY FUNCTION DECLARATIONS */
MeBuzzer buzzer; // create the buzzer object to play the sound
MeLineFollower lineFinder(PORT_1); // assigning the black lineFinder to RJ25 port 1
MeDCMotor leftMotor(M1); // assigning leftMotor to port M1
MeDCMotor rightMotor(M2); // assigning RightMotor to port M2
MeRGBLed led(0,30);


/* PID CONTROL CONSTANTS */
#define kp 15
#define kd 3
float last_error = 0;


/* COLOUR SENSOR CONSTANTS */
#define RGBWait 250
#define LDRWait 10
#define LDR A3 // reading LDR from port 4 (A3 pin)


/* ULTRASONIC CONSTANTS */
#define TIMEOUT 2000
#define SPEED_OF_SOUND 340 // Update according to your own experiment
#define ULTRASONIC 10 // ultrasonic assigned to pin 10 as we are using port 2
#define THRESHOLD 11; // ultrasonic threshold for wall tracking


/* MOVEMENT CONSTANTS */
uint8_t targetSpeed = 225; // max target speed = 255, lowered so PD corrections are easier to execute
#define TURNING_TIME_MS 440 // adjust time taken to turn here
```

Setup and Main Loop

```
void setup() {
 Serial.begin(9600);
 delay(1000);
 pinMode(ULTRASONIC, OUTPUT);
 led.setpin(13);
}

void loop()
{
  int sensorState = lineFinder.readSensors();
  if (sensorState == S1_IN_S2_IN)
  {
    stop(300);
    int colour = detect_colour();
    setColor(colour);
    turn(colour);
  }
  else
  {
    move_maze(targetSpeed, last_error);
  }
}
```

Movement Helper Functions

```
// combine movement into 1 function, note opp motor dir for left
void move(int left, int right)
{
  leftMotor.run(-left);
  rightMotor.run(right);
}

// stop function for specified duration
void stop(int duration)
{
  leftMotor.stop();
  rightMotor.stop();
  delay(duration);
}
```

Movement Algorithm

```
// general movement function based on PD controller
void move_maze(int targetSpeed, float last_error)
{
  float ultra = get_ultra_distance(); // check if ultrasonic is within a workable range (able to detect wall on left)
  if (ultra >= 3.0 && ultra <= 17.0)
  {
    float error = ultra - THRESHOLD;
    float derivative = error - last_error;
    float turn = (error * kp) + (derivative * kd);
    leftMotor.run(-(targetSpeed - turn));
    rightMotor.run(targetSpeed + turn);
    last_error = error;
  }
 /* if no wall detected on ultrasonic side (left), switch to ir sensor, but given its potentially
 unreliable reading, just do minor corrections to each motor instead of PID system */
  else
  {
    float ir = get_ir_distance();
    if (ir <= 5.0) // too close to right
    {
      move(targetSpeed - 20, targetSpeed + 20);
    }
    else if (ir >= 10.0) // too close to left
    {
      move(targetSpeed + 20, targetSpeed - 20);
    }
    else
    {
      move(targetSpeed, targetSpeed);
    }
  }
}
```

Logic For Turning

```
// colour sensor logic for turning
void turn(int colour)
{
  if (colour == 1) //RED - left turn
  {
    move(-targetSpeed, targetSpeed);
    delay(TURNING_TIME_MS);
    stop(200);
  }
  if (colour == 2) //GREEN - right turn
  {
    move(targetSpeed, -targetSpeed);
    delay(TURNING_TIME_MS);
    stop(200);
  }
  if (colour == 3)//ORANGE - UTURN
  {
    move(-targetSpeed, targetSpeed);
    delay((TURNING_TIME_MS * 2) - 50);
    stop(200);
  }
  if (colour == 4)// PURPLE 2 successive lefts
  {
    move(-targetSpeed, targetSpeed);
    delay(TURNING_TIME_MS);
    stop(200);
    move(targetSpeed, targetSpeed);
    delay(1035); // go forward for 1 tile
    //2nd left
    stop(300);
    move(-targetSpeed, targetSpeed);
    delay(TURNING_TIME_MS - 10);
    stop(200);
  }
  if (colour == 5) // BLUE - 2 successive rights
  {
    move(targetSpeed, -targetSpeed);
    delay(TURNING_TIME_MS);
    stop(200);
    move(targetSpeed, targetSpeed);
    delay(1035); // go forward for 1 tile
    // 2nd right
    stop(300);
    move(targetSpeed, -targetSpeed);
    delay(TURNING_TIME_MS - 10);
    stop(200);
  }
  if (colour == 6) // WHITE
  {
    play_tune();
  }
}
```

Colour Sensing

```
// debug function that prints the avg readings read from LDR
// 4-2 adaptor truth table logic to get respective LEDs to light up
// L/H BLUE
// H/L GREEN
// H/H RED
int detect_colour()
{
   analogWrite(A0, 255);
   analogWrite(A1, 255);
   delay(RGBWait);
   int red = getAvgReading(10);
   delay(RGBWait);

   analogWrite(A0, 255);
   analogWrite(A1, 0);
   delay(RGBWait);
   int green = getAvgReading(10);
   delay(RGBWait);

   analogWrite(A0, 0);
   analogWrite(A1, 255);
   delay(RGBWait);
   int blue = getAvgReading(10);
   delay(RGBWait);
   Serial.print("R: ");
   Serial.println(red);
   Serial.print("G: ");
   Serial.println(green);
   Serial.print("B: ");
   Serial.println(blue);
   int colour = get_colour(red, green, blue);
   return colour;
}
```

```
// determine colour detected based on raw LDR value detected
int get_colour(int R, int G, int B)
{
  if (R > 500 && G > 810 && B > 600)
  {
    return 6; //white
  }
  if (R > 500 && (G > 640 && G < 730) && B > 315)
  {
    return 1; //red
  }
  if (R > 500 && G >= 730 && B < 400)
  {
    return 3; //orange
  }
  if (R < 300 && G >= 700 && B > 385)
  {
    return 2; //green
  }
  if (R < 400 && G >= 750 && B > 590)
  {
    return 5; //blue
  }
  if (R > 405 && G > 680 && B > 500)
  {
    return 4; //purple
  }
  return 7; //error :(
}

// find the average reading for the requested number of times of scanning LDR
int getAvgReading(int times){
  int reading;
  int total = 0;
  for(int i = 0;i < times;i++){
     reading = analogRead(LDR);
     total = reading + total;
     delay(LDRWait);
  }
//calculate the average and return it
  return total/times;
}
```

MBot On-board LED Indicator (lights up according to colour detected)

```
void setColor(int colour)
{
  if (colour == 1)
  {
    led.setColor(255, 0, 0); // RED
    led.show();
  }
  if (colour == 2)
  {
    led.setColor(0, 255, 0); // GREEN
    led.show();
  }
  if (colour == 3)
  {
    led.setColor(255, 138, 0); // ORANGE
    led.show();
  }
  if (colour == 4)
  {
    led.setColor(255, 0, 255); // PURPLE
    led.show();
  }
  if (colour == 5)
  {
    led.setColor(0, 0, 255); // BLUE
    led.show();
  }
  if (colour == 6)
  {
    led.setColor(255, 255, 255); // WHITE
    led.show();
  }
  if (colour == 7) // if unable to identify colour
  {
    led.setColor(0, 0, 0);
    led.show();
  }
}
```

Ultrasonic Sensor

```
// function that processes and returns distance sensed from ultra
float get_ultra_distance()
{
 pinMode(ULTRASONIC, OUTPUT);
 digitalWrite(ULTRASONIC, LOW);
 delayMicroseconds(2);
 digitalWrite(ULTRASONIC, HIGH);
 delayMicroseconds(10);
 digitalWrite(ULTRASONIC, LOW);

 pinMode(ULTRASONIC, INPUT);
 long duration = pulseIn(ULTRASONIC, HIGH, TIMEOUT);
 float distance = duration / 2.0 / 1000000 * SPEED_OF_SOUND * 100;
 Serial.println(distance);
 return distance;
}
```

Infrared Sensor

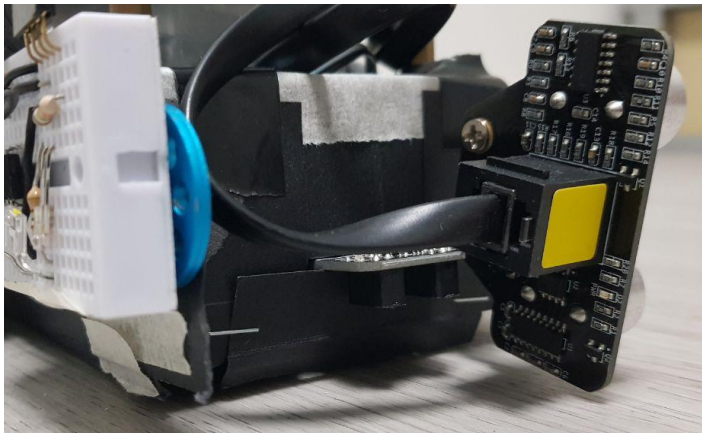
```
int getIRAverage(int times){
//find the average reading for the requested number of times of reading IR sensor
  int reading;
  int total = 0;
//take the reading as many times as requested and add them up
  for(int i = 0;i < times; i++){
     reading = analogRead(A2);
     total = reading + total;
     delay(LDRWait);
  }
  return total/times;
}
float get_ir_distance()
{
  float values = getIRAverage(5);
  Serial.println(values);
  if (values >= 950 && values <= 980) // too close to right side
  {
    return 5;
  }
  else if (values <= 935 && values >= 900)
  {
    return 10; // too close to left side
  }
  return 8; // robot is roughly in the middle OR readings are out of range
}
```

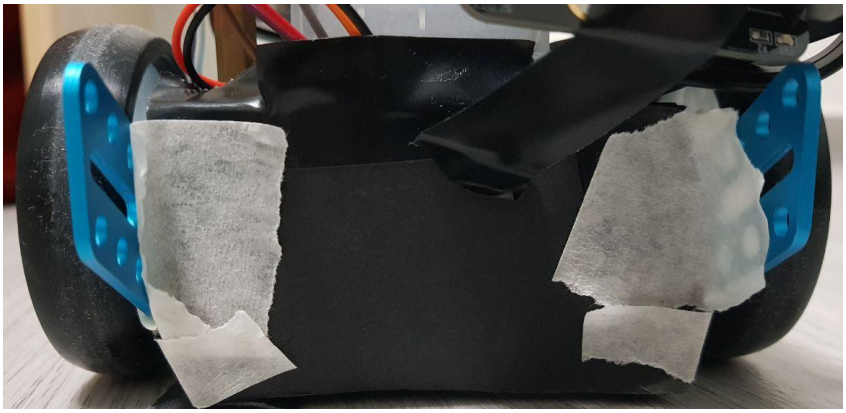
Victory Tune

Credits: Rick Astley - Never Gonna Give You Up chorus

```
void play_tune()
{
   buzzer.tone(466, 200);
   buzzer.tone(466, 100);
   buzzer.tone(415, 100);
   buzzer.tone(415, 100);
   buzzer.tone(698, 300);
   buzzer.tone(698, 300);
   buzzer.tone(622, 600);
   buzzer.tone(466, 100);
   buzzer.tone(466, 100);
   buzzer.tone(415, 100);
   buzzer.tone(415, 100);
   buzzer.tone(622, 300);
   buzzer.tone(622, 300);
   buzzer.tone(554, 300);
   buzzer.tone(523, 100);
   buzzer.tone(466, 200);
   buzzer.tone(554, 100);
   buzzer.tone(554, 100);
   buzzer.tone(554, 100);
   buzzer.tone(554, 100);
   buzzer.tone(554, 300);
   buzzer.tone(622, 300);
   buzzer.tone(523, 300);
   buzzer.tone(466, 100);
   buzzer.tone(415, 200);
   buzzer.tone(415, 200);
   buzzer.tone(415, 200);
   buzzer.tone(622, 400);
   buzzer.tone(554, 800);
   buzzer.tone(466, 100);
   buzzer.tone(466, 100);
   buzzer.tone(415, 100);
   buzzer.tone(415, 100);
   buzzer.tone(698, 300);
   buzzer.tone(698, 300);
   buzzer.tone(622, 600);
   buzzer.tone(466, 100);
   buzzer.tone(466, 100);
   buzzer.tone(415, 100);
   buzzer.tone(415, 100);
   buzzer.tone(831, 300);
   buzzer.tone(523, 300);
   buzzer.tone(554, 300);
   buzzer.tone(523, 100);
   buzzer.tone(466, 200);
   buzzer.tone(554, 100);
   buzzer.tone(554, 100);
   buzzer.tone(554, 100);
   buzzer.tone(554, 100);
   buzzer.tone(554, 300);
   buzzer.tone(622, 300);
   buzzer.tone(523, 300);
   buzzer.tone(466, 100);
   buzzer.tone(415, 200);
   buzzer.tone(0, 200);
   buzzer.tone(415, 200);
   buzzer.tone(622, 400);
   buzzer.tone(554, 800);
   buzzer.noTone();
}
```

Mbot colour sensor curtain





References

PID Control System:

- http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html
- https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Book%3A_Chemical_Process_Dynamics_and_Controls_(Woolf)/09%3A_Proportional-Integral-Derivative_(PID)_Control/9.02%3A_P%2C_I%2C_D%2C_PI%2C_PD%2C_and_PID_control

Arduino Programming:

- https://www.arduino.cc/reference/en/