CG1111A Engineering Principles and Practice I

The A-Maze-ing Race
Project Report 2022

Studio Group/Section/Team: 03-52

Team Members:
Shan YuXuan, A0264661B
Shen James, A0259217Y
Sim Jing Jie Ryan A0252590E
Siew Shui Hon A0258613

Table of Contents

1. Introduction

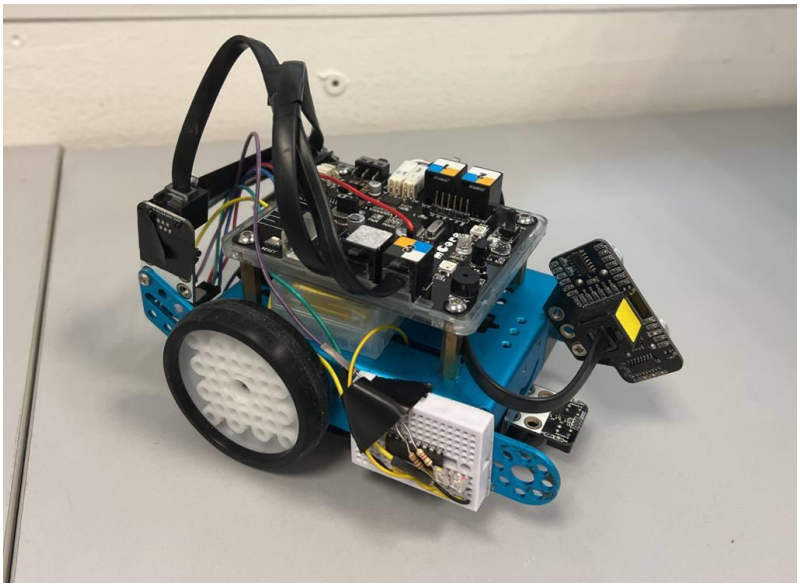

Figure 1: Our team's humble mBot, xiao Yan Ming

In this project, we are tasked to code and wire a robot that is capable of auto-navigating a random maze. We need to use ultrasonic sensors and an infra-red sensor to ensure that the robot maintains a straight course, using a line sensor to stop the robot when reaching a black line, and using LED & LDR to detect colours of the paper below the robot, executing different turns based on the colour. Lastly, the robot needs to play a victory tune at the end of the maze. This report details the components of the mBot, the difficulties we faced in this project and how we overcame them.

2. Robot Algorithm

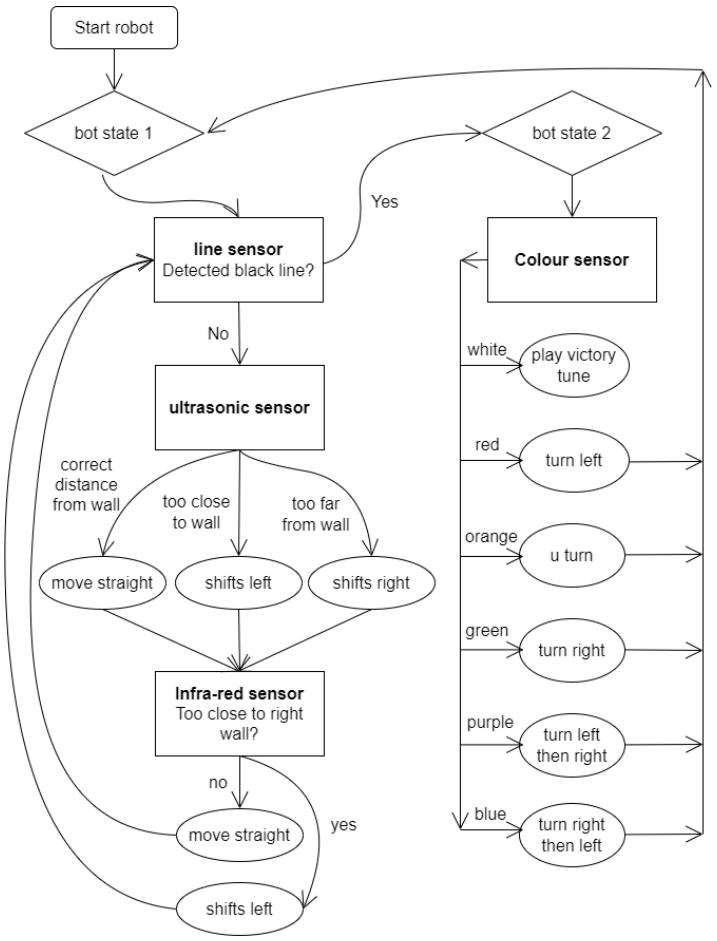

Figure 2: Algorithm used by the robot to solve the maze.

Our robot's overall operation can be divided into 2 states - movement and colour detection state. This chapter aims to introduce the algorithm and briefly cover subsystems' function.

During movement state, the robot constantly moves forward. During forward movement, the line sensor detects for coloured grids' black line while both the ultrasonic and IR sensor works to correct the robot's straight movement path.

Upon detection of black line by the line sensor, the robot transitions into the colour detection state. Identification of the grid's colour triggers the respective action dictated by the colour's requirement. After the specific action, the robot transitions back to the first state to continue solving the maze. This is with exception of detection of the white grid which signifies the end of the maze.

3. Robot Subsystems

Our mBot primarily consists of 3 crucial subsystems driving its operation in specific areas, namely - movement, grid detection and path correction. The subsystems are connected to the mBot's mCore and controlled using the HD74LS139P 2-to-4 Decoder IC Chip and the L293D motor chip. The following details each subsystem and its function specifically.

3.1 Basic Movement Operation

Basic movement operation is powered by 2 PMDC motors mounted under the chassis. The robot travels on 2 rear wheels, driven by the motors, and 1 unpowered front pivot wheel. The completion of the maze comprises the following movement operations: travel in the forward direction, left turns, right turns, and u-turns. The movements are also affected by the ultrasonic sensor, infra-red sensor and line sensor which ensures that the course of movement is straight or to stop the mBot for colour detection. Sensor operation will be detailed in the following section.

```
void move_forward()
{
leftMotor.run(-motorSpeed); // Negative: wheel turns anti-clockwise
rightMotor.run(motorSpeed-20); // Positive: wheel turns clockwise
}
void move_backward()
{
leftMotor.run(motorSpeed); // Negative: wheel turns clockwise
rightMotor.run(-motorSpeed); // Positive: wheel turns anti-clockwise
}
void turn_left()
{
leftMotor.run(motorSpeed); // Positive: wheel turns clockwise
rightMotor.run(motorSpeed); // Positive: wheel turns clockwise
}
void turn_right()
{
leftMotor.run(-motorSpeed); // Positive: wheel turns anti-clockwise
rightMotor.run(-motorSpeed); // Positive: wheel turns anti-clockwise
}
void vehicle_stop()
{
leftMotor.stop(); // Stop left motor
rightMotor.stop(); // Stop right motor
}
```

Figure 3: Code controlling PMDC motor

3.1.1 Forward Movement

To achieve forward movement, the two motors are provided equal power. The wheels are rotated in the opposite rotational direction since their orientations are opposite from each other. In theory, each motor should spin at equal speed and the movement will be perfectly straight. However, there was a noticeable difference in the speed of the 2 motors possibly due to slight difference in internal resistance or other hardware differences. The right motor was slightly faster than the left, causing a slight veer to the left. To account for this, we asymmetrically calibrated the power motors to ensure the wheels have equal RPM for movement in a straight line, nullifying the effects of manufacturing imperfections in the motors.

```
uint8_t motorSpeed = 255;
// Setting motor speed to an integer between 1 and 255
// The larger the number, the faster the speed\

void move_forward()
{
leftMotor.run(-motorSpeed); // Negative: wheel turns anti-clockwise
rightMotor.run(motorSpeed-20); // Positive: wheel turns clockwise
}
```

Figure 4: Default motor speed

Ultimately, our left motor operates at full power whilst the right operates at a slight negative offset, as seen from the figure above.

3.1.2 Turns

To execute simple right-angled turns, we programmed the mBot to only power 1 motor for the duration of the turn. For a left turn, the right motor is activated whilst the left motor is on standby. The inverse is true for a right turn. A right-angled turn is similar to a 180 degree turn, only differing in the duration of the turn itself. Hence, the main challenge was to calibrate the time for which the single motor is powered such that the robot rotates 90 or 180 degrees. Our team overcame this task through trial and error, incrementally adjusting the time the motor is powered to make sure that the robot is facing the right orientation after each turn.

```
turn_left();
delay(TURNING_TIME_MS);
vehicle_stop();
delay(50);
move_forward();
delay(750);
vehicle_stop();
delay(50);
turn_left();
delay(TURNING_TIME_MS);
vehicle_stop();
delay(50);
```

Figure 5: Code snippet for successive left turns

Our robot is also required to execute successive left turns. The operation involves a simple 90 degree left-turn, slight travel forward and another 90 degree left-turn. The left-turns utilise the same simple turn function above. The slight forward travel involves a straight frontwards movement for a short duration of time.

3.1.3 Difficulties faced with motor operation

In general, motor functions were relatively straightforward to implement, both physically into the robot's circuit and digitally via code.

However, the calibration for the successive left turn operation was slightly harder due to the small margin of error allowed. As the function was hard-coded, there is high probability of the robot colliding with the wall due to there being no path correction during the execution of the turn and hence very slight differences in starting position can result in the robot travelling or rotating too much. Therefore, not only does the operation need to be precise, our robot needs to also be in a relatively precise position before the turn itself. Ultimately, the challenge was not only to achieve the successive turns itself, but also to ensure that our path correction was reliable enough to provide this necessary initial conditions.

3.2 Grid Detection

The grid detection by our robot consists of the line detector (Me Line Follower) and a colour sensor circuit.

3.2.1 Line Detector

```
//// line sensor
int sensorState = lineFinder.readSensors(); // read the line sensor's state
if (sensorState == S1_IN_S2_IN) //when black line is reached
{
  vehicle_stop();
  bot_state = 2;
  delay(50);
}
```

Figure 6: Code for stopping upon black line detection

The line detector's function is to identify coloured grids by detecting the black horizontal line on each grid. After black line is detected by reading the line sensor's output, we then set the robot to the colour detection mode and movement stops. From here, the colour detection algorithm runs.

3.2.2 Colour Sensor

The colour sensor makes use of a RGB lamp and a light dependent resistor (LDR). The RGB lamp turns on in the order of red, green and blue, where the light is reflected from the ground below into the LDR. The lamps turn on through the use of the decoder IC where Y_1, Y_2, Y_3 are set to a low output in order. The sensor then inputs an analog value to the arduino based on the voltage from the LDR circuit. We used a 100kΩ resistor for the LDR circuit, a 1kΩ resistor for the red LED, and 2 560Ω resistors for the blue and green LEDs. We obtained these resistor readings based on the data sheets of the respective LEDs such that the current flowing through the LEDs when it is on is at a suitable level for operation in our colour sensor.

```
else if (state == 0) {
  //red
  analogWrite(G_enable, 0);
  analogWrite(A_input, 255);
  analogWrite(B_input, 0);
}
else if (state == 1) {
  //green
  analogWrite(G_enable, 0);
  analogWrite(A_input, 0);
  analogWrite(B_input, 255);
}
else if (state == 2) {
  //blue
  analogWrite(G_enable, 0);
  analogWrite(A_input, 255);
  analogWrite(B_input, 255);
}
```

Figure 7 : Code snippet for turning on RGB lamps

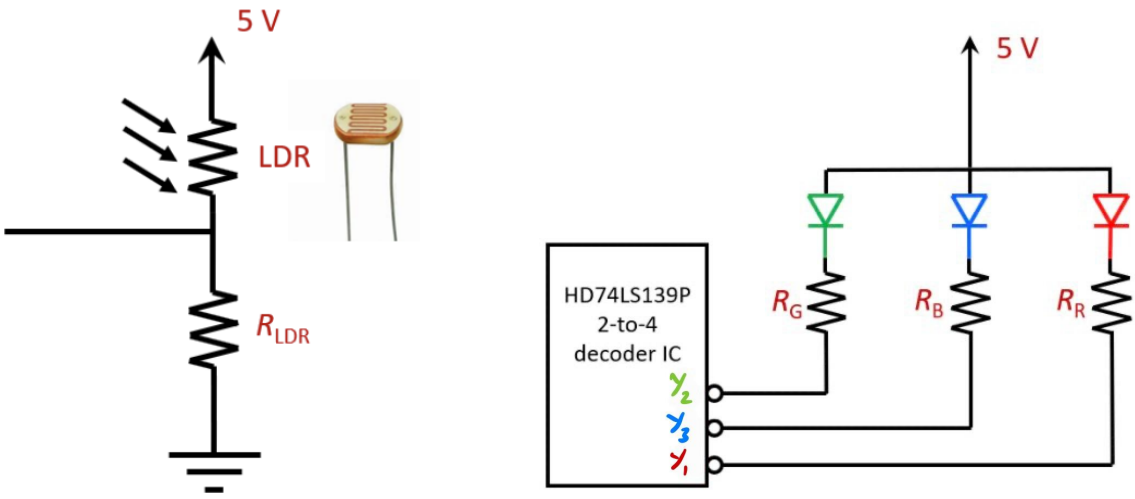

Figure 8 : Figure of the LDR circuit and RGB lamp

3.2.3 Calibration of RGB values

We first obtained the corresponding analog inputs of RGB for the colours white and black and input these values into an array.

```
void setBalance()
{
  //set white balance
  Serial.println("Put White Sample For Calibration ...");
  delay(5000);
  //delay for five seconds for getting sample ready
  //scan the white sample.
  //go through one colour at a time, set the maximum reading for each colour -- red, green and blue to the white array
  for(int i = 0;i<=2;i++){
     decoder_state(i);
     delay(RGBWait);
     whiteArray[i] = getAvgReading(5);
     //scan 5 times and return the average,
     Serial.println(whiteArray[i]);
     decoder_state(3);
     delay(RGBWait);
  }
  //done scanning white, time for the black sample.
  //set black balance
  Serial.println("Put Black Sample For Calibration ...");
  delay(5000);     //delay for five seconds for getting sample ready
  //go through one colour at a time, set the minimum reading for red, green and blue to the black array
  for(int i = 0;i<=2;i++){
     decoder_state(i);
     delay(RGBWait);
     blackArray[i] = getAvgReading(5);
     Serial.println(blackArray[i]);
     decoder_state(3);
     delay(RGBWait);
  //the differnce between the maximum and the minimum gives the range
     greyDiff[i] = whiteArray[i] - blackArray[i];
  }
```

Figure 9 : Code snippet used for obtaining white and black

	Red	Green	Blue
White	945.00	985.00	981.00
Black	805.00	881.00	862.00
GreyDiff	140.00	104.00	119.00

Figure 10 : Values obtained for white and black, stored in an array

We thus made use of these values to obtain corresponding RGB colour codes for the colour grid detection through our arduino code, and inputting the colours into the serial monitor to help us identify the readings obtained by the sensor.

```
for(int c = 0;c<=2;c++){
  Serial.print(colourStr[c]);
  decoder_state(c); //turn ON the LED, red, green or blue, one colour at a time.
  delay(RGBWait);
  //get the average of 5 consecutive readings for the current colour and return an average
  colourArray[c] = getAvgReading(5);
  //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 (i.e. the colour LDR is exposed to)
  colourArray[c] = (colourArray[c] - blackArray[c])/(greyDiff[c])*255;
  decoder_state(3);  //turn off the current LED colour
  delay(RGBWait);
  Serial.println(int(colourArray[c]));
  //show the value for the current colour LED, which corresponds to either the R, G or B of the RGB code
}
```

Figure 11: Code snippet used for colour detection

3.2.4 Colour recognition

From fig above, the colour codes are stored in an array colourArray. The index 0, 1 and 2 thus corresponds to R, G and B. We first tested the colour sensor on the grid colours to obtain the range of RGB values of its corresponding colours.

Colour	R	G	B
Red	225 - 240	120 -135	140 - 150
Orange	220 - 240	160 -170	130 - 140
Blue	165 - 175	220 - 235	235 - 245
Purple	180 - 190	140 -160	225 - 240
Green	75 - 90	165 - 180	125 -140
White	210 - 240	220 - 245	220 - 255

Figure 12 : Table of obtained RGB ranges

From the values we obtained, we noticed that white can be identified when all RGB values are more than 210, as it is the only colour among all 6 to have such properties. Red and orange can be identified when the R value is maximum, blue and purple can be identified when the B value is maximum, and green can be identified when G is maximum. These maximum RGB codes always hold true for each colour from the numerous testings we made.

We faced some difficulties in distinguishing colours with the same maximum RGB value, such as red and orange, and blue and purple. It was unreliable to only use the RGB colour values to

distinguish such cases as the RGB values might deviate slightly from the values we assign such to distinguish blue from purple, and red from orange. Thus, we decided to implement the calculation of hue to distinguish the similar colours from one another.

```
////calculating hue
int determine_colour(int R, int G, int B){
  double red = (double)R/255;
  double green = (double)G/255;
  double blue = (double)B/255;
  double newArray[3] = {red,green,blue};
  for(long i = 0; i < 2;i++){
    long flag = 0;
    for(long j = 0 ; j < 3 - (i + 1); j++){
      if (newArray[j+1] < newArray[j]){
          double temp = newArray[j + 1];
          newArray[j + 1] = newArray[j];
          newArray[j] = temp;
          flag += 1;
     }
      }
    if(flag == 0){
      break;
    }
  }

  double hue = 0;
  if(newArray[2] == red){
    hue = (green - blue)/(newArray[2] - newArray[0]);
  }
  else if(newArray[2] == green){
    hue = 2.0 + (blue - red)/(newArray[2] - newArray[0]);
  }
  else{
    hue = 4.0 + (red - green)/(newArray[2] - newArray[0]);
  }
  /*if(hue < 0){
    hue *= 60;
    return hue + 360;
  }*/
  return hue * 60;
}
```

Figure 13: Code snippet of hue calculation

Through the calculation of hues, we could easily identify red and orange as its hue values were easily distinguishable (red was 1 - 10, orange was 200 - 300). Blue and purple were still distinguishable (blue around 190, purple around 210), but the hues were a little too close for us to reliably solely depend on, hence we used the condition that the G value would be less than 200 to differentiate purple from blue as a fallback should the hue input deviate slightly.

```
if (colourArray[0] > 220 && colourArray[1] > 220 && colourArray[2] > 220)
//white
{
  Serial.println("white");
  vehicle_stop();
  // Each of the following "function calls" plays a single tone.
  // The numbers in the bracket specify the frequency and the duration (ms)
  buzzer.tone(392, 200);
  buzzer.tone(523, 200);
  buzzer.tone(659, 200);
  buzzer.tone(784, 200);
  buzzer.tone(659, 150);
  buzzer.tone(784, 400);
  buzzer.noTone();
}
else if (colourArray[1] > colourArray[0] && colourArray[1] > colourArray[2]) //when G is max
//green
{
  turn_right();
  delay(TURNING_TIME_MS);
  vehicle_stop();
  delay(1000);
  Serial.println("green");
  bot_state = 1;
}
else if(colourArray[0] > colourArray[1] && colourArray[0] > colourArray[2]) //when R is max
{
  if (Hue < 10)
  //red
  {
    turn_left();
    delay(TURNING_TIME_MS);
    vehicle_stop();
    delay(1000);
    Serial.println("red");
    bot_state = 1;
  }
  else
  //orange
  {
    turn_left();
    delay(2 * TURNING_TIME_MS);
    vehicle_stop();
    delay(1000);
    Serial.println("orange");
    bot_state = 1;
  }
}
```

Fig 14: Code snippet showing identification of white, green, red and orange

```
else if(colourArray[2] > colourArray[1] && colourArray[2] > colourArray[0]) //when blue is max
{
  if (Hue > 205 || colourArray[1] < 200)
  //purple
  {
    turn_left();
    delay(TURNING_TIME_MS);
    vehicle_stop();
    delay(1000);
    move_forward();
    delay(750);
    vehicle_stop();
    delay(1000);
    turn_left();
    delay(TURNING_TIME_MS);
    vehicle_stop();
    delay(1000);
    Serial.println("purple");
    bot_state = 1;
  }
  else //blue
  {
    turn_right();
    delay(TURNING_TIME_MS);
    vehicle_stop();
    delay(1000);
    move_forward();
    delay(750);
    vehicle_stop();
    delay(1000);
    turn_right();
    delay(TURNING_TIME_MS+20);
    //move_forward();
    //delay(750);
    vehicle_stop();
    delay(1000);
    Serial.println("blue");
    bot_state = 1;
```

Fig 15: Code snippet showing identification of blue and purple

3.2.5 Difficulties faced with colour recognition

On top of calibration of the light sensor itself, we found that surrounding ambient light was affecting reading of our light sensor which made our sensor unreliable in accurately differentiating colours from one another. We resolved this through the assembly of a chimney using black masking tape and paper to cover the LDR from ambient light.

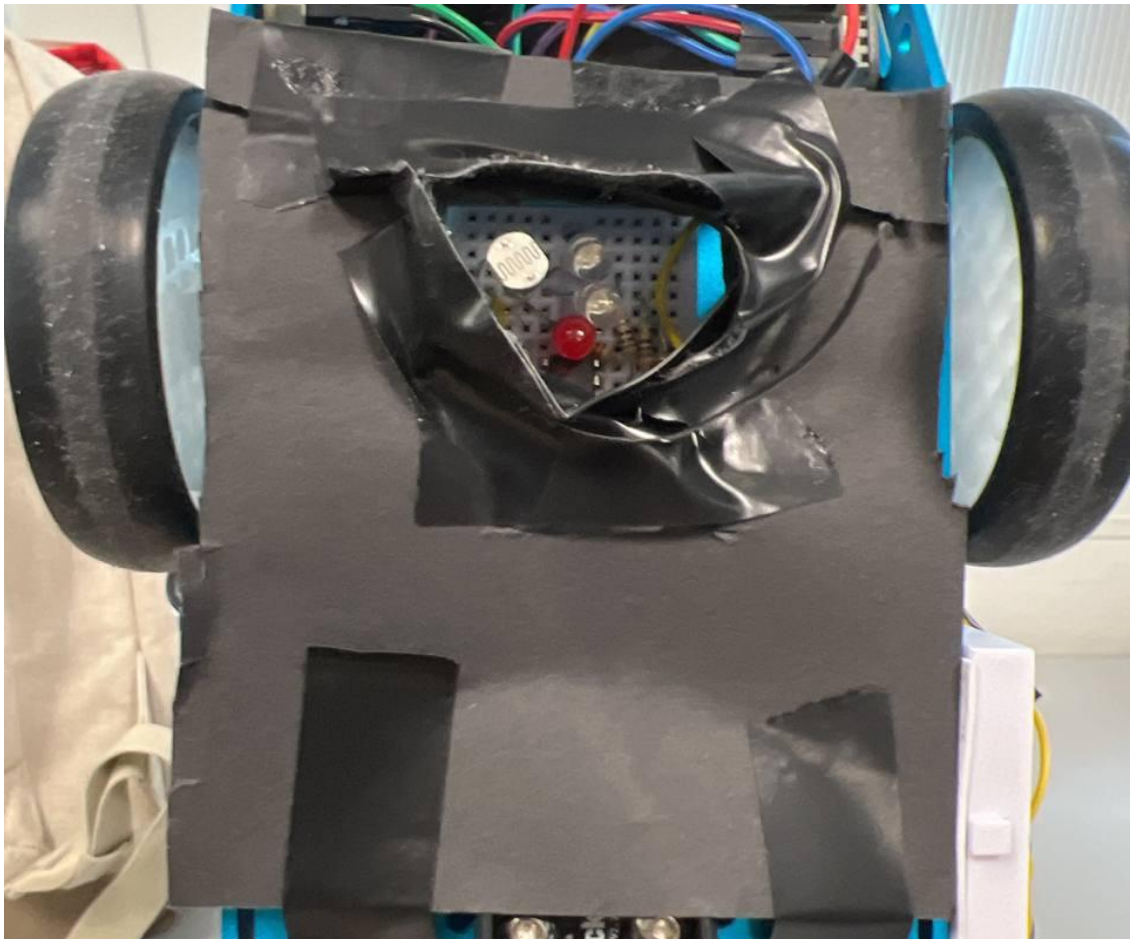

Fig 16: Chimney design

Additionally, as seen from the figure 13, the if condition where hue < 0 had been commented out as we noticed from our testings that the colour sensor occasionally returns RGB values such that the hue calculated would be a small negative value. From that, the function would return hue that exceeds the actual hue of the colour, and as such cause identification of red to identify orange instead.

To counter this issue, we removed the overflow part of the hue calculation so that it can return a negative value and we can just set the condition for colour red to be hue < 10.

3.3 Path Correction Mechanism

On top of basic movement, precise path adjustment in the maze was achieved by a combination of the ultrasonic sensor, the IR sensor and the code that constantly adjusts the robot based on the values measured.

3.3.1 Ultrasonic Sensor

The ultrasonic sensor is a component provided alongside the mBot kit. It is able to detect distances between the sensor and the object directly in front of it. The ultrasonic sensor was mounted on the front of the robot, facing left, and used to detect the distance between the robot and the left wall.
The value obtained from the ultrasonic sensor was through port 2 of the mCore.

```
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 dist_cm = duration / 2.0 / 1000000 * SPEED_OF_SOUND * 100;
```

Figure 17: Code snippet for ultrasonic sensor

The code snippet above in Figure 17 shows how we calculated distance with the ultrasonic sensor.

3.3.2 Infrared Sensor

The infrared sensor(IR) was a circuit constructed by the team on a breadboard. The diagrams of the circuits are shown below in Figures 18 and 19

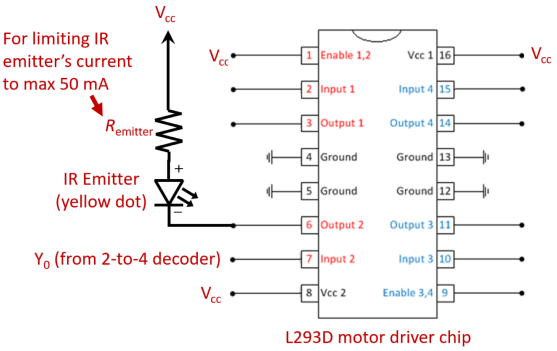

Figure 18

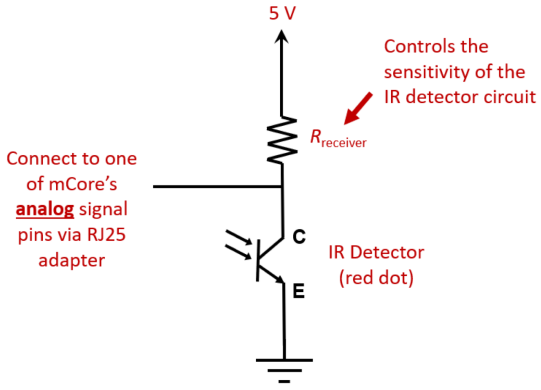

Figure 19

The $R_{reciever}$ value that we chose was 8200Ω, and the $R_{emitter}$ value we chose was 150Ω. This combination of resistances gave us the best possible effect of the IR sensor after multiple tests and allowed it to work both effectively and reliably.

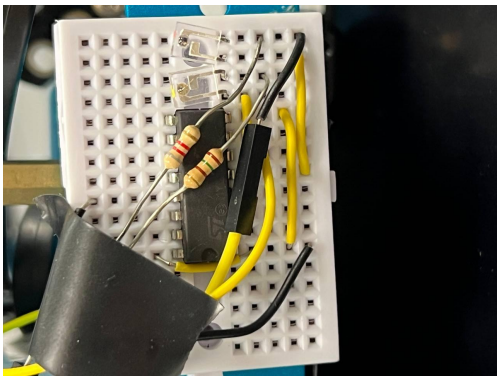

Figure 20: Set up of breadboard for IR sensor

The image above showcases our actual IR sensor circuit. The negative polarity of the IR emitter was connected to the Y_0 pin of the decoder IC chip. The breadboard that holds our IR sensor also has the L293D motor chip. To ensure that the entire circuit is flush with the body of the robot, we cut the wires, the IR emitter and detector as well as the resistors. We then used black tape to hold all the loose wirings in one place and in so doing, reduce the risk of one of the wires or components hitting the wall of the course.

The IR sensor was mounted on the right side of the robot and used to detect distances between the right wall and the robot.

3.3.3 Implementation of Ultrasonic and IR sensor together

Using the two sensors described above, we would be able to constantly detect the values of the distance of the left and right walls from our mBot and we would be able to calculate the position of the mBot on the course.

Due to the ultrasonic sensor having a larger range than the IR sensor and being more accurate, our mBot would adjust its course based on the reading from the ultrasonic sensor and only when the IR sensor detects a value that signified that the bot was too close to the right wall, there would be a leftwards adjustment to the course of the robot.

```
int val = analogRead(IR_receiver);  // read the input pin
Serial.println(val);
if (val < 700) {
  leftMotor.run(-motorSpeed+200);
  rightMotor.run(motorSpeed);
  delay(500);
}
```

Figure 21: Code snippet for IR sensor

The code snippet in Figure 21 shows how we made use of the values of the IR sensor. The IR sensor will input analog values based on its distance with the right wall. A higher value indicates that the distance is further and a lower value indicates that it is closer to the wall. If the value is too low, the robot will steer left sharply to prevent collision with the wall.

```
float dx = dist_cm - 11.0;
//Serial.println(dist_cm);
if (sensorState == S1_IN_S2_IN) //when black line is reached
 {
  vehicle_stop();
  bot_state = 2;
  delay(50);

}
else if (dist_cm == 0) {
  move_forward();
}
else if (dx <= 0) {
  leftMotor.run(-motorSpeed); // Negative: wheel turns anti-clockwise
  rightMotor.run(motorSpeed + dx*15 - 8); // Positive: wheel turns clockwise
}
else {
  leftMotor.run(-motorSpeed + dx*15); // Negative: wheel turns anti-clockwise
  rightMotor.run(motorSpeed - 8); // Positive: wheel turns clockwise
}
```

Figure 22: Code snippet for course adjustment base on ultrasonic sensor

As shown in the code snippet in FIgure 22, we programmed our ultrasonic sensor around a distance of 11cm (rightward positive). This is because we found that having the mBot 11cm away from the left wall allowed us to make the turns at the challenge squares without having any bumps against the course.

As for the movement, when the mBot is perfectly 11cm away from the left wall, it will be moving straight ahead.

However, when the mBot is more than 11cm away from the left wall, the left motor will spin at a slower speed compared to the right motor, thus causing a leftward correction.

Similarly, when the distance is less than 11cm away from the left wall, the right motor will spin at a slower speed compared to the right motor, causing a rightward correction.

In general, we tried not to vary the motor speeds by a huge amount as we wanted the path correction to occur as smoothly as possible.

3.3.4 Difficulties faced with Path Correction

One of the biggest difficulties we faced in our mBot design was that when the ultrasonic sensor was too close to the wall, it would not be able to correctly measure the distance and no reading would be returned. This means that there will be no course correction in those cases.

To combat this issue, we mounted the ultrasonic sensor a distance from the side of the robot so there is a buffer distance between the ultrasonic sensor and the wall. This allowed our ultrasonic sensor to detect distances more reliably.

3.4 Miscellaneous

3.4.1 Victory Tune

```
if (colourArray[0] > 200 && colourArray[1] > 220 && colourArray[2] > 220) //white
{
  Serial.println("white");
  vehicle_stop();
  // Each of the following "function calls" plays a single tone.
  // The numbers in the bracket specify the frequency and the duration (ms)
  buzzer.tone(392, 200);
  buzzer.tone(523, 200);
  buzzer.tone(659, 200);
  buzzer.tone(784, 200);
  buzzer.tone(659, 150);
  buzzer.tone(784, 400);
  buzzer.noTone();
}
```

Figure 23: Code snippet of victory tune

Once the bot detects white colour in colour detection mode, it will stop and play a victory tune. Since the code does not set it back to bot state 1, the robot will stay on the white colour while play the victory tune continuously. This signifies the completion of the maze and the end of the robot's operation.

3.4.2 Minor components

Other components necessary for the robot's functions include the battery, wires and cables, such as RJ25 connectors.

4. Work division

At the start of the project, Yu Xuan and James were working on the code while Ryan and Shui Hon tinkered with the hardware, making sure that the IR sensor and colour sensor worked. Throughout the lab sessions, team members gathered at different timings outside of our studio sessions to work on the bot. Each member played their respective part well.

5. TA Yan Ming

Figure 24: TA Yan Ming holding xiao Yan Ming

Without TA Yan Ming's help it would not have been possible for us to complete our project. After naming our robot and our arduino code file after him, our robot works perfectly.

End