Report
of
The A-maze-ing Race Project 2020

Studio Group 4B Team 6

Zhao Luoyuang (A0211196L)
Zhong Xinghan (A0211198H)
Zhou Chengxu (A0211222E)
Zhuang Jianning (A0214561M)

Section 1: Overall Algorithm of mBot

The purpose of the project was to build up our mBot along with two proximity sensors and program it so that it can clear a maze in the shortest time without any collisions.

Fig. 1. illustrates the overall algorithm of our mBot that allows it to overcome the colour challenges to complete the maze. At the start of each loop, we read in input from the MeLineFollower, MeUltrasonicSensor and the voltage (analogRead()) from the left and right IR sensors.

We check if there is a black line under the mBot, and as an additional level of safety, we also make sure the mBot is not too close to the wall in front. This is done so through the conditional statement if (sensor_state < 3 || front_distance <= 7). If this returns True, the mBot will Stop(), and call the detect_colour() function which returns the corresponding integer mapping of the 6 colours. The mBot then carries out the corresponding action, turn or celebratory tune.

If there is no black line or wall, the mBot will move Forward() and check if the left or right distance to the wall is too close. Beyond a certain threshold, the mBot will adjust itself using TurnRight(); delay(15); or TurnLeft(); delay(15); slightly so as not to crash into the walls.

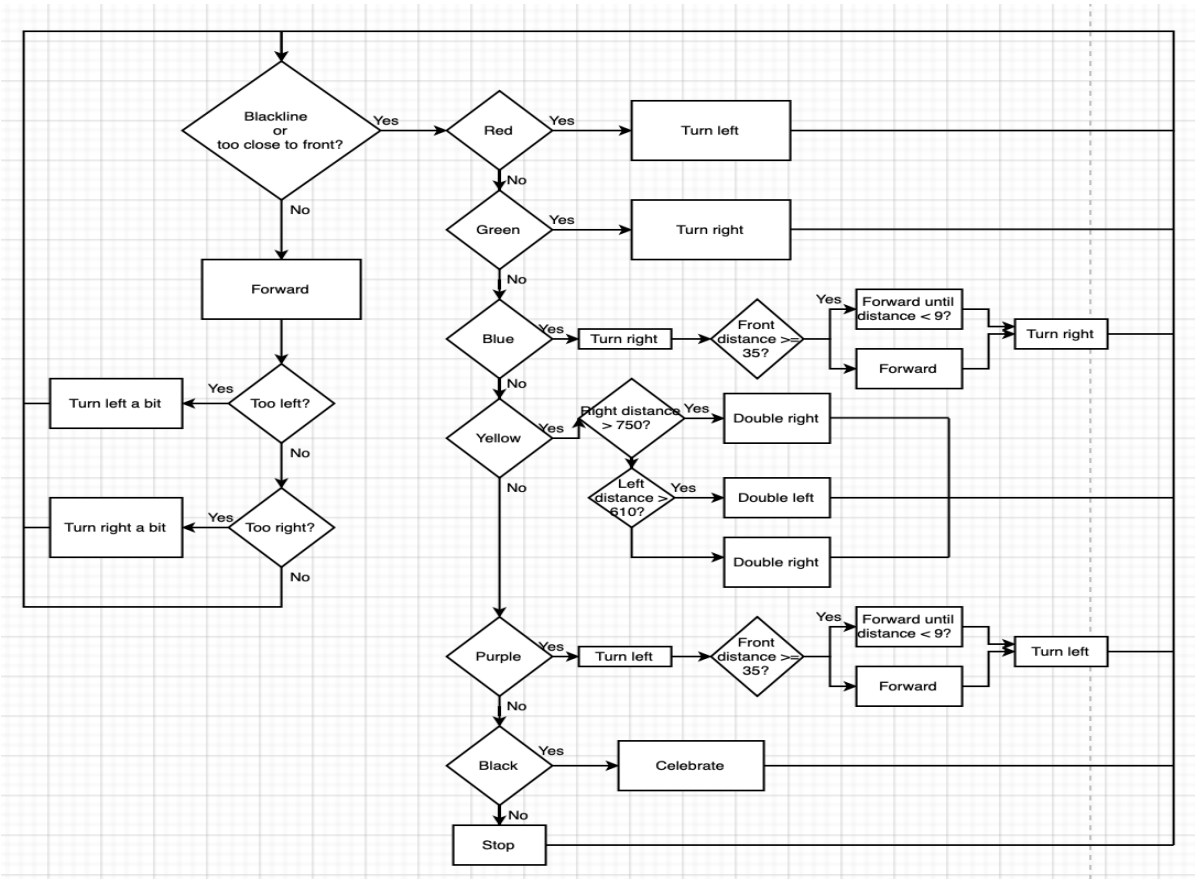


Fig. 1. Overall Algorithm of mBot

Section 2: Implementation of Subsystems in mBot

This section describes the 4 subsystems in our algorithm which allow the mBot to complete the maze. They include the Motor / Turning, Straight Line Movement / Wall Avoidance, Colour Detection, and Music Playing subsystems.

2.1. Motor / Turning Subsystem

We initiated a variable for the motor's speed, and then wrote the functions for moving forward, turning left, turning right, and stopping by giving commands to the mBot's 2 motors. However, we noticed that given the same motor speed, the left wheel moves slower than the right one, so we assigned the speed of the left wheel to be (-moveSpeed * 1.09) to match the speed of the right wheel. The MotorL.run and MotorR.run functions are from the <MeMCore.h> library.

```
void Forward(){
  MotorL.run(-moveSpeed*1.09);
  MotorR.run(moveSpeed);
}

void TurnLeft(){
  MotorL.run(moveSpeed);
  MotorR.run(moveSpeed);
}

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

Fig. 2. Motor / Turning Functions

2.2. Straight Line Movement/ Wall Avoidance Subsystem

As for the algorithm for keeping mBot straight, we used the IR sensors at both sides of the car to detect whether there's a wall near the sides of the mBot. If there is a wall 3 cm away from the car, we make the car turn in the opposite direction for slightly (delay(15)). The left_distance = analogRead(IR1), and the right_distance = analogRead(IR2). After our calibration of the IR sensors, we got that the reading of the left IR sensor is about 590 while the car is around 3 cm away from wall, and the reading of the right IR sensor is about 600 while the car is around 3 cm away from wall.

```
    Forward();
    if (left_distance < 590){ //around 3 cm away from wall?
      Serial.println(left_distance);
      Serial.println("Turn right");
      TurnRight();
      delay(15);
      //Stop();
    }
    if (right_distance < 600){ //around 3 cm away from wall?
      Serial.println(right_distance);
      Serial.println("Turn left");
      TurnLeft();
      delay(15);
      //Stop();
    }
  }
}
```

Fig. 3. Straight Line / Wall Avoidance Functions

2.3.Colour Detection and Execution Subsystem

The setColor () function makes sure that only red/green/blue light at the same time to get the accurate rgb value of that card.
colourArray[i] = (LightSensor.read() - blackArray[i]) / (greyDiff[i]) * 255;
This sentence adjusts the value inside the range of white and black RGB value to make the RGB value reasonable. “delay(10);” is used to meet the response time otherwise the sensor cannot work correctly. “LED.setColor(0,0,0)” ensures that the LED is off after each one value detecting.

```
int detect_colour(){
  for(int i = 0; i < 3; i++){
    LED.setColor(rgbArr[i][0],rgbArr[i][1],rgbArr[i][2]);
    LED.show();
    for(int j=0; j < 20; j++){
      colourArray[i] = (LightSensor.read()-blackArray[i])/(greyDiff[i])*255;
      delay(10);
    }
    LED.setColor(0,0,0);
    LED.show();
    Serial.println(colourArray[i]);
  }
```

```
if (colourArray[0] > 140 && colourArray[1] < 70 && colourArray[2] < 70){
  Serial.println("RED DETECTED!");
  return 0;
}
else if (colourArray[0] < 70 && colourArray[1] >100 && colourArray[2] < 70){
  Serial.println("GREEN DETECTED!");
  return 1;
}
else if (colourArray[0] > 200 && colourArray[1] > 200 && colourArray[2] > 200){
  return 6;
}
else if (colourArray[0] > 100 && colourArray[1] >150 && colourArray[2] > 150){
  Serial.println("BLUE DETECTED!");
  return 2;
}
else if (colourArray[0] > 200 && colourArray[1] > 130 && colourArray[2] < 100){
  Serial.println("YELLOW DETECTED!");
  return 3;
}
else if (colourArray[0] > 100 && colourArray[1] > 100 && colourArray[2] > 150){
  Serial.println("PURPLE DETECTED!");
  return 4;
}
else if (colourArray[0] < 50 && colourArray[1] < 50 && colourArray[2] < 50){
  Serial.println("BLACK DETECTED!");
  return 5;
}
else{
  return 6;
}
```

Fig. 4. Colour Detection Function

These conditional statements are to return the color type of that RGB value, when return 6, it means not detected any color.

Then, as for the execution of different color's commands (including the end of maze command), we have the below conditions to make the car execute different commands after the color sensing returns values related to different colors,

a) When there is a black line detected, the car will stop to detect the color:
b) When cmd == 0 (red color detected), the car will turn single left;
c) When cmd == 1 (green color detected), the car will turn single right;
d) When cmd == 2 (blue color detected), the car will turn double right;
e) When cmd == 3 (yellow color detected), the car will turn 180 degrees;
f) When cmd == 4 (purple color detected), the car will turn double left;
g) When cmd == 5 (black color detected), the car will stop and play music;
h) When cmd == 6 (nothing detected), the car will simply stop.

```
if (sensor_state < 3 || front_distance <= 7){
  Stop();
  int cmd = detect_colour();
  if (cmd == 0){
    TurnLeft();
    delay(300);
    Stop();
    delay(500);
  }
  else if (cmd == 1){
    TurnRight();
    delay(300);
    Stop();
    delay(500);
  }
  else if (cmd == 2){
    TurnRight();
    delay(300);
    Stop();
    delay(500);
    if(UltrasonicSensor.distanceCm() <= 35) {
      while(1) {
        if (UltrasonicSensor.distanceCm() < 9)
          Stop();
          break;
        } else {
          Forward();
        }
       }
      } else {
      Forward();
      delay(700);
    }
    Stop();
    delay(500);
    TurnRight();
    delay(300);
    Stop();
    delay(500);
  }
  else if (cmd == 3){
    Stop();
    delay(300);
    Serial.println(analogRead(IR2));
    if (analogRead(IR2) > 750){
      TurnRight();
      delay(300);
      Stop();
      delay(500);
      TurnRight();
      delay(300);
      Stop();
      delay(500);
    }
    else if (analogRead(IR1) > 610){
      TurnLeft();
      delay(300);
      Stop();
      delay(500);
      TurnLeft();
      delay(300);
      Stop();
      delay(500);
    }
    else{
      TurnRight();
      delay(300);
      Stop();
      delay(500);
      TurnRight();
      delay(300);
      Stop();
      delay(500);
      }
  }
  else if (cmd == 4){
    TurnLeft();
    delay(300);
    Stop();
    delay(500);
    if(UltrasonicSensor.distanceCm() <= 35) {
      while(1) {
        if (UltrasonicSensor.distanceCm() < 9) {
          Stop();
          break;
        } else {
          Forward();
        }
       }
      } else {
      Forward();
      delay(700);
    }
    Stop();
    delay(500);
    TurnLeft();
    delay(300);
    Stop();
    delay(500);
  }
  else if (cmd == 5){
    Stop();
    celebrate();
    delay(5000);
  }
  else if (cmd == 6){
    Stop();
    delay(500);
  }
}
else{
  Forward();
  if (left_distance < 590){ //around 3 cm away from wall?
    Serial.println(left_distance);
    Serial.println("Turn right");
    TurnRight();
    delay(15);
    //Stop();
  }
  if (right_distance < 600){ //around 3 cm away from wall?
    Serial.println(right_distance);
    Serial.println("Turn left");
    TurnLeft();
    delay(15);
    //Stop();
  }
}
```

Fig. 5. Movements Execution commands

2.4. Music Playing Subsystem

Music is generated through the MeBuzzer on the mBot. To play a tone of specific frequency and duration, we call Buzzer.tone(pin number, frequency, duration) with the appropriate parameters.

We first need to define the various notes and their corresponding frequencies that will be used in our celebratory tune. We decided to go with Take On Me by a-ha as it has a catchy tune and only uses 7 notes. Within the celebrate() function, we also created arrays called melody and durations which stores each note and their respective duration. We then iterate through the arrays which play the notes in order according to the array with the correct duration and pause. The celebrate() function is called when the mBot detects the colour Black in subsystem 2.3, which signals the end of the maze.

```
// frequencies for required notes
#define NOTE_B4  494
#define NOTE_D5  587
#define NOTE_E5  659
#define NOTE_FS5 740
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_B5  988
#define REST      0

void celebrate(){ //TAKE ON ME melody
  int melody[] = {
    NOTE_FS5, NOTE_FS5, NOTE_D5, NOTE_B4, NOTE_B4, NOTE_E5,
    NOTE_E5, NOTE_E5, NOTE_GS5, NOTE_GS5, NOTE_A5, NOTE_B5,
    NOTE_A5, NOTE_A5, NOTE_A5, NOTE_E5, NOTE_D5, NOTE_FS5,
    NOTE_FS5, NOTE_FS5, NOTE_E5, NOTE_E5, NOTE_FS5, NOTE_E5
  };
  int durations[] = {
    8, 8, 8, 4, 4, 4,
    4, 5, 8, 8, 8, 8,
    8, 8, 8, 4, 4, 4,
    4, 5, 8, 8, 8, 8
  };
  int songLength = sizeof(melody)/sizeof(melody[0]);
  for (int thisNote = 0; thisNote < songLength; thisNote++){
    int duration = 1000/ durations[thisNote];
    Buzzer.tone(8, melody[thisNote], duration);
    int pause = duration * 1.3;
    delay(pause);
    Buzzer.noTone(8);
  }
}
```

Fig. 6. Music Playing Function

Section 3: Steps Taken for Calibration

Calibration for IR sensor:
We first put the IR sensor equipped mbot in the middle of the maze with two walls on left and right. Then we got the value around 700. And when we put the mbot in the position that we want it to adjust its direction we found the value is around 590 for left, 600 for right. Thus we used these two values in the code. When the right wall distance detected is less than 590, the bot will turn left a bit. When the left wall distance detected is less than 600, the bot will turn right a bit.

Calibration for Color sensor:
The major calibration of the color sensor was the range of the Red, Green, Blue(RGB) value of all the colorful papers and the RGB value of the white paper and black paper. We first calibrated the RGB value of the black paper and white paper, these values were used as ranges, without this range the RGB value of other colorful papers will not be accurate. We used the arduino to output the average RGB value of black paper and white paper and stored them in whitearray and blackarray so that the grey difference is whitearray[i] - blackarray[i]. For calibration of the colorful papers, in order to get the most accurate RGB value of the colorful papers we put the car inside the maze to calibrate the value at that certain height(the height may influence would influence the RGB value), we output the values and expanded the range appropriately by making sure that they are not overlapped, for example

```
if (colourArray[0] > 140 && colourArray[1] < 70 && colourArray[2] < 70){
  Serial.println("RED DETECTED!");
  return 0;
}
else if (colourArray[0] < 70 && colourArray[1] >100 && colourArray[2] < 70){
  Serial.println("GREEN DETECTED!");
  return 1;
}
```

The Blue value for red and green were both less than 70, because the Red and Green values were different so the 2 color's value will not overlap.

Section 4: Work Division

Work Division

Zhao Luoyuang:
Built the IR sensor circuit and completed the wire connection of the mbot.
Helped adjust the turning angles.

Wrote Section 1 and Section 3 IR sensor's part.

Zhong XInghan:
Wrote the Motor / Turning subsystem code, then adjusted the value of the left / right motor to make the wheels on both sides generally have the same speed.
Adjusted the delay values in the execution subsystem to make the turning angle generally accurate.
Built up the circuit of the IR sensor and then connected and fixed it to the mBot with a beautified version of circuit.

Wrote the Motor / Turning, Straight Line Movement / Wall Avoidance, Execution Subsystems in Section 2 and drew the 2 schematic diagrams that describe the problem we encountered during double turns in Section 5.

Zhuang Jianning:
Designed the overall control flow of the mBot program, including the detect_colour() function, the corresponding action for each of the 6 colours, left and right adjustments when moving straight, and the celebrate() function.

Wrote Section 5 and the Music Playing subsection in Section 2 of the report.

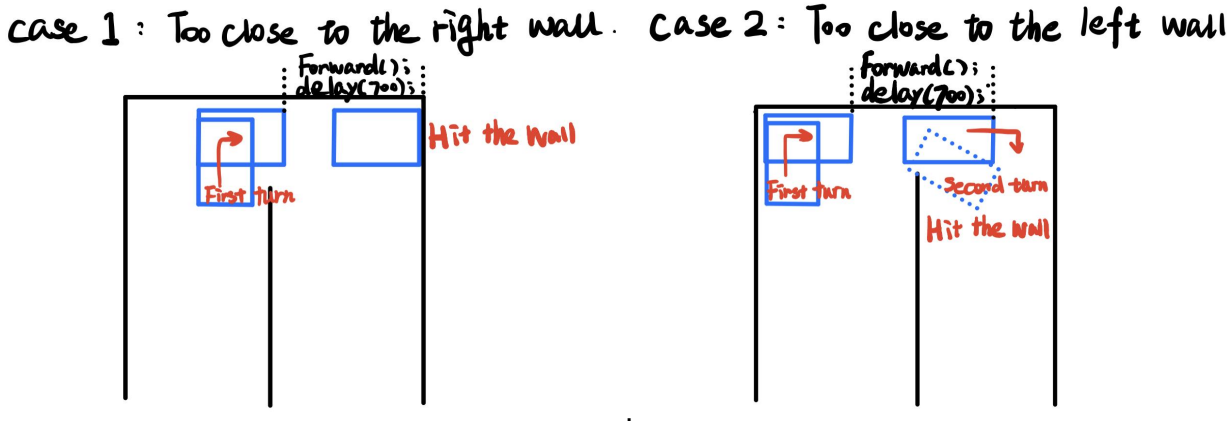
Zhou Chengxu:
Wrote the code for the color sensor's calibration, line follower sensor and the ultrasonic sensor.
Worked on the improvement of algorithms for the 2 successive turns movement.

Wrote the Section 3 color sensor's part and section 2 color sensor's part.

Section 5: Challenges and Actions Taken

One of the first challenges we faced was ensuring the mBot moved forward the appropriate distance in between the two successive left/right turns in two grids (Purple / Light Blue).

Our original approach was using Forward() and delay() to control how much the mBot moves forward. However, 2 adverse outcomes presented themselves. First, if the mBot enters the turn too close to the direction it is turning, it may bump into the wall before executing the second turn. Second, if the mBot enters the turn too far away from the direction it is turning, it may not move forward sufficiently to clear the turn/pillar



To address this issue, we added another conditional statement for the forward movement of this type of successive turn. We check if the distance to the wall is less than 35cm, then we move forward until this distance is less than 9cm and break out of the loop to execute the second turn. If the distance is greater than 35cm either because there is no wall in front, or the wall is too far away, we can safely use Forward(), delay(700) to move clear the required distance before making the second turn.

```
else if (cmd == 2){ // light blue
  TurnRight();
  delay(300);
  Stop();
  delay(500);
  if(UltrasonicSensor.distanceCm() <= 35) {
    while(1) {
      if (UltrasonicSensor.distanceCm() < 9) {
        Stop();
        break;
      } else {
        Forward();
      }
     }
    } else {
    Forward();
    delay(700);
  }
  Stop();
  delay(500);
  TurnRight();
  delay(300);
  Stop();
  delay(500);
}
```

Adding this additional check made our successive turns much more consistent. Our mBot would never hit the wall during our testing after we implemented this check.

The second challenge we faced was similar to the first in terms of potential collision with walls. This occurs when our mBot tried to execute the 180 degree turn within the same grid (Yellow).

Our original approach was to just execute two 90 degree right turns when the colour yellow is detected. But we ran into the adverse scenario where the mBot enters the yellow turn grid too close to the right wall. When this happens, the mBot hits or scrapes the wall during the turning movement. If we program the robot to turn left 180 degrees, it will run into a similar scenario if it enters the grid too close to the left wall.

Hence, we came up with the solution to check which direction is safe to turn based on the readings the IR proximity sensors return us before the turn. We first check if there is space on the right to turn with the conditional (analogRead(IR2) > 750) Based on our calibrations, the right IR sensor returns a value of around 750 when it is around 6cm away from the wall, which is safe to turn. If there is no space on the right, we check the left IR sensor (analogRead(IR1) > 610) Similarly, the left IR sensor returns a value of around 610 when it is 6 cm away from the wall. In the worst case scenario where we do not end up in either conditions, we just hope for the best and turn right. However, during our testing, this seldom happened.

```
else if (cmd == 3){ // yellow
  Stop();
  delay(300);
  Serial.println(analogRead(IR2));
  if (analogRead(IR2) > 750){
    TurnRight();
    delay(300);
    Stop();
    delay(500);
    TurnRight();
    delay(300);
    Stop();
    delay(500);
  }
  else if (analogRead(IR1) > 610){
    TurnLeft();
    delay(300);
    Stop();
    delay(500);
    TurnLeft();
    delay(300);
    Stop();
    delay(500);
  }
  else{
    TurnRight();
    delay(300);
    Stop();
    delay(500);
    TurnRight();
    delay(300);
    Stop();
    delay(500);
  }
}
```

Lastly, the most incalculable challenge we faced was the varying values returned by our IR proximity sensors. We determined a few factors that caused this variability in values.

First, the voltage values we get from analogRead() changes depending on the state of our batteries. If the batteries are not charged to near full periodically, the values returned from analogRead() are actually significantly different which affects the threshold values we set for when to adjust left and right when moving forward. We managed to determine a suitable threshold of around 560 and it worked well throughout our testing from 3/11/2020 to the day of the race.

Second, collisions with the wall during the testing may have caused some hardware damage to our IR sensors. The first time we noticed some anomaly, it was because the IR sensors which protruded from our breadboard were knocked out of place such that the detector obstructed the emitter. This was easily resolved by pushing the IR sensors back in place. However, past a certain point, these anomalies became even more common and would cause our mBot to spin on the spot even without walls. Our IR sensors might have become too damaged at that point.

Third, some of our wiring came loose around 15 minutes before we had to surrender our mBot. As such, we only had time to use some tape to secure it in place. This also affected the voltage readings from AnalogRead() but we did not have enough time to recalibrate the proximity sensors. This resulted in haphazard movements which caused our mBot to crash into walls or misalign turns even though our other colour sensors and turns were accurate.

Overall, it was quite a pity that our mBot failed to perform to its full potential due to some last minute hardware issues. Our mBot actually managed to clear many mazes built by other groups and by ourselves as early as the Tuesday one week before the evaluation. We have multiple success videos taken on Tuesday and Thursday where both TAs were present to witness our mBot's many successes. We feel that we have met the requirements of the project and even gone beyond in certain aspects such as the extra logic to ensure our double turns will not have collisions (see Section 5.1) We will be happy to share with you the videos and hope you can take this into consideration when grading our final evaluation run. Thank you.