CG1111 Engineering Principles and Practice 1

The A-maze-ing Race Project

Report Writing

13.11.2020

Group 4a-6

Name	Matriculation Number
NGUYEN TIEN KHOA	A0206416M
ROYCIUS LIM YUANWEI	A0218024R
ONG HAN QIN	A0216668U
SEE JIAN HUI	A0217701N

Table of Contents

Section 1: Source Code

Our main code for running the robot is as follow:

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

/*** Configurations ***/
#define IR_LEFT         A1
#define IR_RIGHT        A0
#define LDR_PIN         A6
#define LED_PIN         7
#define BUZZER_PIN      8
MeDCMotor               leftWheel(M1);
MeDCMotor               rightWheel(M2);
MeLineFollower          lineFinder(PORT_1);
MeUltrasonicSensor      ultraSensor(PORT_3);
MeRGBLed                led(LED_PIN);
MeBuzzer                buzzer;

// Movement
#define MOTOR_SPEED               220
#define TIME_TURN_90_DEGREE       260
#define TIME_GRID                 35                    // time to travel 1 grid
#define TIME_DELAY                20                    // delay b4 recheck position
#define TIME_MUL                  5                     // time multiplier for IR
adjustment
#define K_DIST                    (255/2)               // max correction to mvmt
#define D_FRONT                   5                     // cm
#define V_LEFTIR                  435                   // threshold for IR sensor
values
#define V_RIGHTIR                 570
#define RIGHT_TURN                556
```

```
// Color
#define COL_DIST        4000                    // threshold for comparing colours
#define BLA_VAL         {270, 255, 284}
#define GRE_VAL         {323, 282, 325}

#define RED_ARR         {204, 81, 77}           // normalised RGB values
#define GRE_ARR         {52, 124, 66}
#define YEL_ARR         {255, 206, 121}
#define PUR_ARR         {140, 145, 189}
#define BLU_ARR         {167, 233, 220}
#define BLA_ARR         {0,0,0}
#define NUM_OF_COLOURS  6                       // black, red, green, yellow, purple, blue

// Calibration
#define CALLIBRATE_SEC  3                       // delay before IR and colour calibration
#define NUM_OF_SAMPLES  50                      // number of samples for calibration
#define IR_WAIT         100                     // delay between IR measurements
#define RGB_WAIT        50                      // delay between each LED flash
#define LDR_WAIT        10                      // delay between each LDR measurement
#define LED_MAX         255                     // max value of each RGB component

/********** Global Variables **********/
bool busy = false;

int blackArray[] = BLA_VAL;
int greyDiff[] = GRE_VAL;
static int allColourArray[6][3] = {BLA_ARR, RED_ARR, GRE_ARR, YEL_ARR, PUR_ARR, BLU_ARR};

/********** Main Program **********/
void setup() {
  pinMode(IR_LEFT, INPUT);
  pinMode(IR_RIGHT, INPUT);
  pinMode(LDR_PIN, INPUT);
```

```
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  Serial.begin(9600);

  busy = false;
}

void stopRunning(const int i);

void loop() {
  if (busy) return;

  if (isAtBlackLine() == true) { // both sensors not in black line
    // waypoint detected
    busy = true;
    stopRunning(0);

    // color challenge
    int colourRes;
    do {
      colourRes = getColour();
    } while (colourRes == -1);

    if (colourRes > 0) { // is color challenge (not black)
      colorWaypoint(colourRes);
      busy = false;
      return;
    }

    // finished
    finishWaypoint();
  }
  else {
    moveForward();
```

```
  }
}

/*** Movement ***/
void stopRunning(const int i = 10) {
  rightWheel.stop();
  leftWheel.stop();
  if (i) delay(TIME_DELAY * i);
}

// right motor is
int rightSpeed = MOTOR_SPEED * 0.78;
int leftSpeed = -MOTOR_SPEED;

void moveForward() {
  if (ultraSensor.distanceCm() < D_FRONT) {
    stopRunning();
    return;
  }

  int new_delay = 50;
  int leftReading = analogRead(IR_LEFT);
  int rightReading = analogRead(IR_RIGHT);

  if (leftReading < V_LEFTIR) {
    rightWheel.stop();
    leftWheel.run(-MOTOR_SPEED);
    delay(new_delay);
  }
  else if (rightReading < V_RIGHTIR) {
    leftWheel.stop();
    rightWheel.run(MOTOR_SPEED);
    delay(new_delay);
  }
```

```
  else {
    leftWheel.run(leftSpeed);
    rightWheel.run(rightSpeed);
  }
}

void forwardGrid() {
  for (int i = 0; i < TIME_GRID; i++) {
    if (ultraSensor.distanceCm() < D_FRONT) break;
    leftWheel.run(leftSpeed);
    rightWheel.run(rightSpeed);
    delay(TIME_DELAY);
  }
  stopRunning();
}

void turnRight() {
  leftWheel.run(-MOTOR_SPEED);
  rightWheel.run(-MOTOR_SPEED);
  delay(TIME_TURN_90_DEGREE);
  stopRunning();
}

void turnLeft() {
  leftWheel.run(MOTOR_SPEED);
  rightWheel.run(MOTOR_SPEED);
  delay(TIME_TURN_90_DEGREE);
  stopRunning();
}

void doubleRight() {
  turnRight();
  stopRunning(15);
  forwardGrid();
```

```
  stopRunning(15);
  turnRight();
}

void doubleLeft() {
  turnLeft();
  stopRunning(15);
  forwardGrid();
  stopRunning(15);
  turnLeft();
}

void uTurn() {
  int rightReading = analogRead(IR_RIGHT);

  if (rightReading < RIGHT_TURN) {
    turnLeft();
    turnLeft();
  }
  else {
    turnRight();
    turnRight();
  }
}



/*** Sensors ***/
long long square(const long long x) { return x * x; }

int getColour() {
  // Read colours
  float colourArray[3] = {0};
  for (int i = 0; i < 3; i++) { // red, green, blue
    led.setColor( // one-hot encoding
```

```
      ((1<<i)   &1) * LED_MAX,
      ((1<<i>>1)&1) * LED_MAX,
      ((1<<i>>2)&1) * LED_MAX
    );
    led.show();
    delay(RGB_WAIT);

    for (int j = 0; j < NUM_OF_SAMPLES; j++) { // take avg reading
      colourArray[i] += analogRead(LDR_PIN);
      delay(LDR_WAIT);
    }

    colourArray[i] /= NUM_OF_SAMPLES;
    colourArray[i] = (colourArray[i] - blackArray[i]) * 255 / greyDiff[i];
    Serial.println(colourArray[i]);
  }
  led.setColor(0,0,0); led.show();

  // Find colour with min euclidean distance > COL_DIST
  int idx = -1;
  int min_dist = COL_DIST;
  for (int i = 0; i < 6; i++) {
    long long curr_dist = 0;
    for (int j = 0; j < 3; j++)
      curr_dist += square(allColourArray[i][j] - colourArray[j]);

    if (curr_dist <= COL_DIST && curr_dist < min_dist) {
      idx = i;
      min_dist = curr_dist;
    }
  }
  return idx;
}
```

```
bool isAtBlackLine() {
  return lineFinder.readSensors() == S1_IN_S2_IN;
}

/*** Waypoints ***/
void colorWaypoint(const int colourRes) {
  switch (colourRes) {
    case 1: turnLeft(); break;    // red
    case 2: turnRight(); break;   // green
    case 3: uTurn(); break;       // yellow
    case 4: doubleLeft(); break;  // purple
    case 5: doubleRight(); break; // light blue
  }
}

void finishWaypoint() {
  // keys and durations found in NOTES.h
  int wholenote = (60000 * 4) / tempo;
  for (int i = 0; i < notes * 2; i = i + 2) {
    // calculates the duration of each note
    int divider = melody[i + 1];
    int noteDuration = 0;
    if (divider > 0) {
      // regular note, just proceed
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      // dotted notes are represented with negative durations
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5; // increases the duration in half for dotted notes
    }

    // we only play the note for 90% of the duration, leaving 10% as a pause
    buzzer.tone(BUZZER_PIN, melody[i], noteDuration * 0.9);
```

```
    // Wait for the specified duration before playing the next note.
    delay(noteDuration);

    // stop the waveform generation before the next note.
    buzzer.noTone(BUZZER_PIN);
  }
  busy = 1;
}

/*** Calibration ***/
void calibrateWB() {
  int whiteArray[3] = {0};

  // Max values
  Serial.print("Put WHITE sample. Calibrating MAX in ");
  for (int i = CALLIBRATE_SEC; i > 0; i--) {
    Serial.print(i); Serial.print(".. "); delay(1000);
  }
  for (int i = 0; i < 3; i++) {
    led.setColor( // one-hot encoding
      ((1<<i)   &1) * LED_MAX,
      ((1<<i>>1)&1) * LED_MAX,
      ((1<<i>>2)&1) * LED_MAX
    ); led.show();
    delay(RGB_WAIT);

    for (int j = 0; j < NUM_OF_SAMPLES; j++) {
      whiteArray[i] += analogRead(LDR_PIN);
      delay(LDR_WAIT);
    }
    whiteArray[i] /= NUM_OF_SAMPLES;
  }
  led.setColor(0,0,0); led.show();
  Serial.println("done.");
```

```
// Min values
Serial.print("Put BLACK sample. Calibrating MIN in ");
for (int i = CALLIBRATE_SEC; i > 0; i--) {
  Serial.print(i); Serial.print(".. "); delay(1000);
}
for (int i = 0; i < 3; i++) {
  led.setColor( // one-hot encoding
    ((1<<i)   &1) * LED_MAX,
    ((1<<i>>1)&1) * LED_MAX,
    ((1<<i>>2)&1) * LED_MAX
  ); led.show();
  delay(RGB_WAIT);

  blackArray[i] = 0;
  for (int j = 0; j < NUM_OF_SAMPLES; j++) {
    blackArray[i] += analogRead(LDR_PIN);
    delay(LDR_WAIT);
  }
  blackArray[i] /= NUM_OF_SAMPLES;
  greyDiff[i] = whiteArray[i] - blackArray[i];
}
led.setColor(0,0,0); led.show();
Serial.println("done.");

// Output calibrated
Serial.print("#define BLA_VAL         {");
Serial.print(blackArray[0]); Serial.print(", ");
Serial.print(blackArray[1]); Serial.print(", ");
Serial.print(blackArray[2]); Serial.println("}");
Serial.print("#define GRE_VAL         {");
Serial.print(greyDiff[0]); Serial.print(", ");
Serial.print(greyDiff[1]); Serial.print(", ");
Serial.print(greyDiff[2]); Serial.println("}");
```

```
}

void getColourCode() {
  // Read colours
  Serial.print("Put COLOR sample. Calibrating MIN in ");
  for (int i = 0; i < CALLIBRATE_SEC; i++) {
    Serial.print(i); Serial.print(".. "); delay(1000);
  }
  Serial.println();
  float colourArray[3] = {0};
  for (int i = 0; i < 3; ++i) { // red, green, blue
    led.setColor( // one-hot encoding
      ((1<<i)   &1) * LED_MAX,
      ((1<<i>>1)&1) * LED_MAX,
      ((1<<i>>2)&1) * LED_MAX
    );
    led.show();
    delay(RGB_WAIT);

    for (int j = 0; j < NUM_OF_SAMPLES; j++) { // take avg reading
      colourArray[i] += analogRead(LDR_PIN);
      delay(LDR_WAIT);
    }
    colourArray[i] /= NUM_OF_SAMPLES;
    colourArray[i] = (colourArray[i] - blackArray[i]) * 255 / greyDiff[i];
    Serial.print(colourArray[i]); Serial.print(", ");
  }
  led.setColor(0,0,0); led.show();
}

/* IR Sensor Calibration */
void getAvgDist() {
  // Take raw value and threshold
  float totalLeft = 0;
```

```
  float totalRight = 0;
  for (int i = 0; i < NUM_OF_SAMPLES; i++) {
    totalLeft += analogRead(IR_LEFT);
    totalRight += analogRead(IR_RIGHT);
  }
  float avgLeft = totalLeft / NUM_OF_SAMPLES;
  float avgRight = totalRight / NUM_OF_SAMPLES;
  Serial.print("LEFT: "); Serial.print(avgLeft); Serial.println();
  Serial.print("RIGHT: "); Serial.print(avgRight); Serial.println();
}
```

All the notes for playing the celebratory tune and an auxiliary function for printing the color detected during testing are contained inside the header file “Notes.h” below:

```
// File for defining constant objects and notes & celebratory music

#ifndef Notes_h
#define Notes_h

/* NOTES FOR CELEBRATORY TUNE */
#define NOTE_C5  523
#define NOTE_D5  587
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_G5  784
#define NOTE_A5  880
#define NOTE_AS5 932
#define REST      0

// change this to make the song slower or faster
int tempo = 280;

// notes of the melody followed by the duration.
// a 4 means a quarter note, 8 an eighteenth , 16 sixteenth, so on
// !!negative numbers are used to represent dotted notes,
// so -4 means a dotted quarter note, that is, a quarter plus an eighteenth!!
int melody[] = {
  NOTE_C5,4, //1
  NOTE_F5,4, NOTE_F5,8, NOTE_G5,8, NOTE_F5,8, NOTE_E5,8,
  NOTE_D5,4, NOTE_D5,4, NOTE_D5,4,
  NOTE_G5,4, NOTE_G5,8, NOTE_A5,8, NOTE_G5,8, NOTE_F5,8,
  NOTE_E5,4, NOTE_C5,4, NOTE_C5,4,
  NOTE_A5,4, NOTE_A5,8, NOTE_AS5,8, NOTE_A5,8, NOTE_G5,8,
```

```
  NOTE_F5,4, NOTE_D5,4, NOTE_C5,8, NOTE_C5,8,
  NOTE_D5,4, NOTE_G5,4, NOTE_E5,4, NOTE_F5,2, REST,4
};

// sizeof gives the number of bytes, each int value is composed of two bytes (16 bits)
// there are two values per note (pitch and duration), so for each note there are four bytes
int notes = sizeof(melody) / sizeof(melody[0]) / 2;

void printColour(const int colourRes) {
  String s;
  switch (colourRes){
    case 0: s="black"; break;
    case 1: s="red"; break;
    case 2: s="green"; break;
    case 3: s="yellow"; break;
    case 4: s="purple"; break;
    case 5: s="light blue"; break;
    default: s="not found"; break;
  }
  Serial.println(s);
}

#endif
```

Section 2: Design of Our mBot

The motherboard of the mBot is the black coloured board in the centre of the mBot with many components connected to it via wiring.

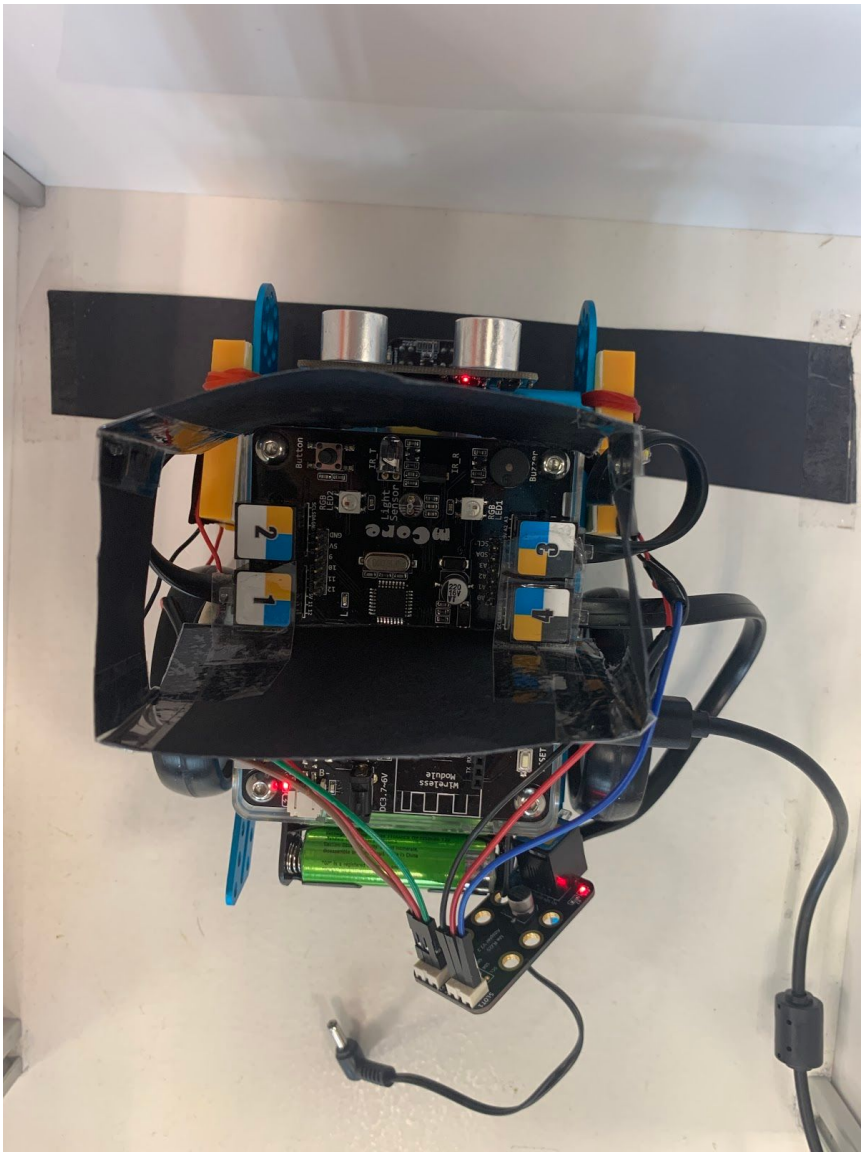

Figure 1. *The mBot from a bird's eye view*

The ‘eyes’ of the mBot is actually an ultrasonic sensor! The mBot is also wearing a ‘hat’ made out of black paper to improve its color sensing capabilities.

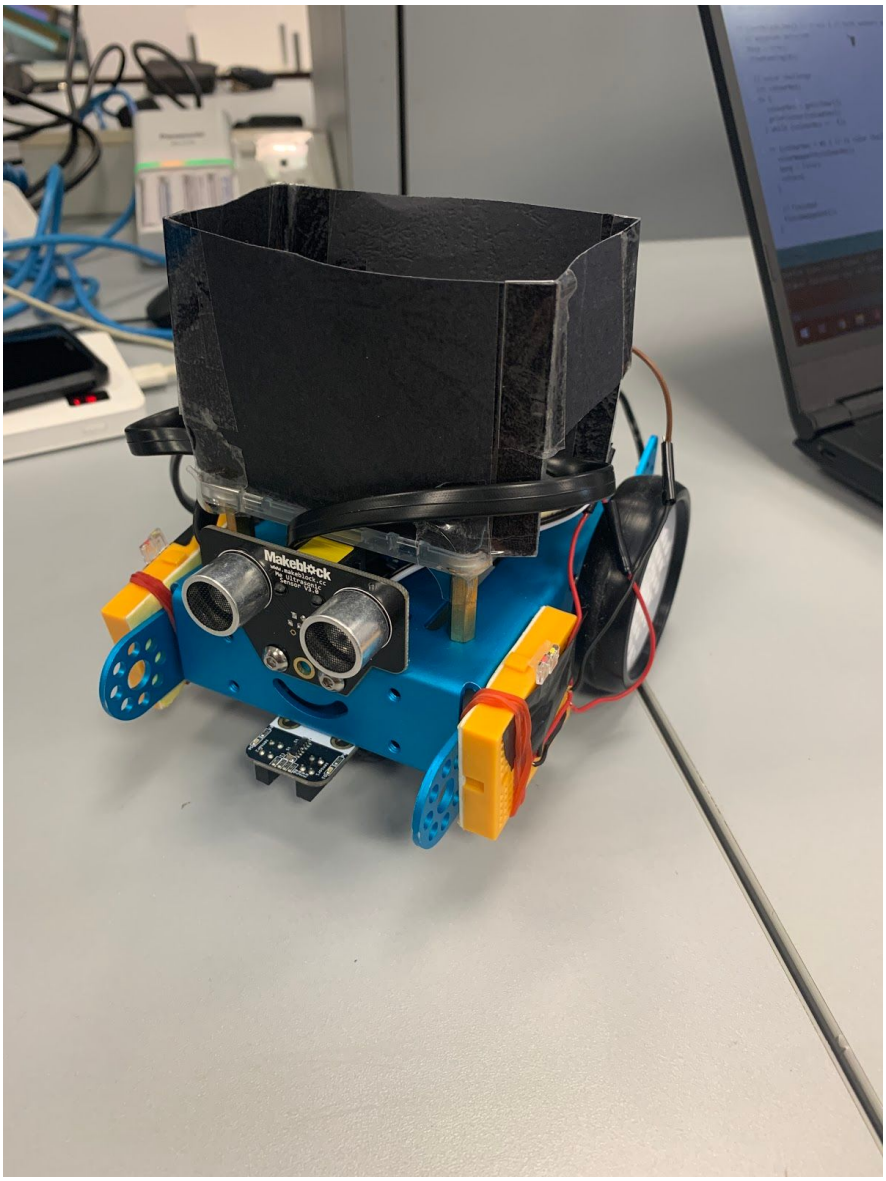

Figure 2. *The mBot when seen from the front*

The green colored batteries in the battery pack power the mBot with electrical energy.

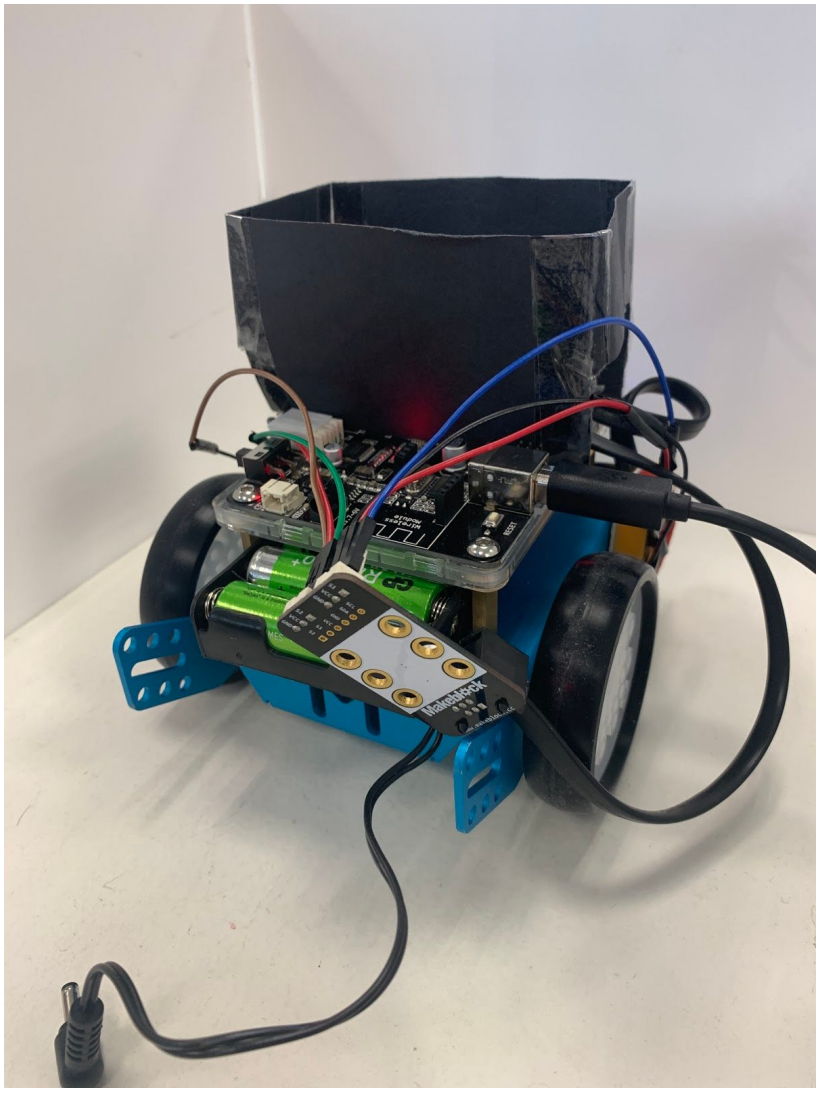

Figure 3. *The mBot when seen from the back view*

Note that there is one yellow bread board on each side of the mBot. These series of circuits detect the distance of the mBot from walls on the left and right. These two sensors are used to emit IR signals and receive reflected signals so that the mBot can gauge its distance from side walls.

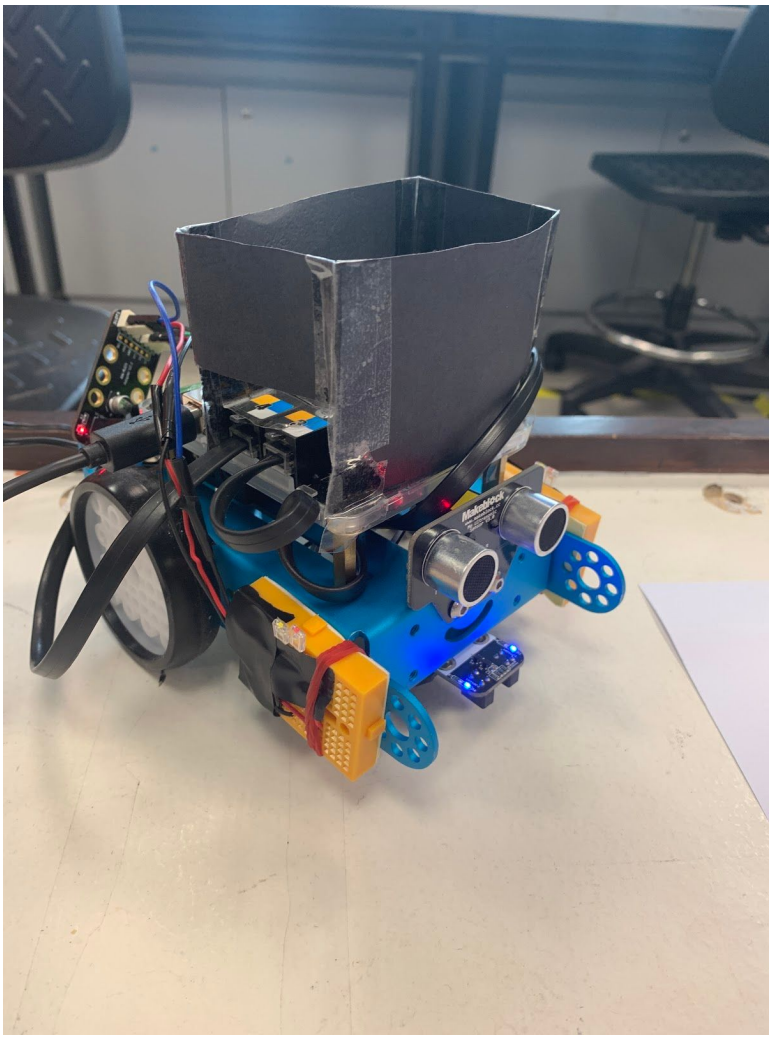

Figure 4. *The yellow breadboard one side of the mBot*

The light yellow components with wheels attached are the Permanent Magnet Direct Current (PMDC) motors which drive the mBot.

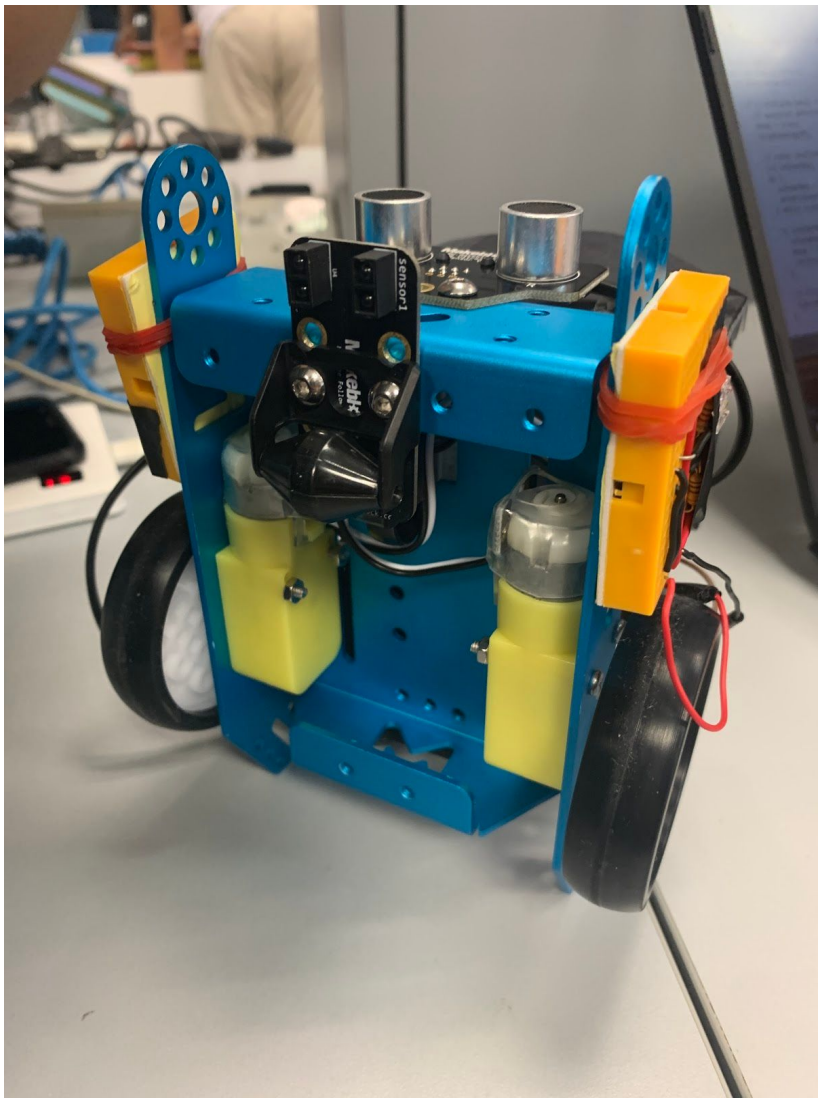

Figure 5. *The underside of the mBot*

I. The mCore (the motherboard)

Refer to Figure 1.

Just like any personal computers (PC) and laptops, the mBot comes with its very own motherboard which is the mCore. The mCore connects the various parts of the mBot together to form a functioning vehicle robot.

II. Permanent Magnet Direct Current (PMDC) Motors

(Yellow components underneath the mBot)

Refer to Figure 5.

The mBot is run on PMDC motors which adjust their torque based on the potential difference applied across the motors. The potential difference across the motors give rise to current flowing through the motors to turn the wheels connected to the PMDC motors

III. Ultrasonic sensor (Eyes)

Refer to Figure 2.

The ultrasonic sensor acts as a front facing proximity sensor and utilises the phenomenon of echolocation. By sending out ultrasonic soundwaves and detecting the reflected wave from an obstacle, the ultrasonic sensor gives the ability for the mBot to prevent crashing into any front facing obstruction.

IV. Battery Pack (note the green batteries)

Refer to Figure 3.

The battery pack supplies electrical power to the mBot and acts as a fuel source for the mBot to run. Batteries used in our mBot are rechargeable which is environmentally sustainable and leads to less electronic waste.

V. Left and Right IR sensors upgrade (Yellow breadboards on the side)

Refer to Figure 4.

A beautiful addition on top of the ultrasonic sensor. When paired with the source code, the mBot makes adjustments in its path of travel with respect to the voltage change across the IR sensors when near walls on the left and right.

VI. The black box upgrade (Paper cut out on the mBot that looks like a hat)

Refer to Figure 1 and Figure 2.

A box made out of black colored paper attached on top of the mBot and surrounding the light sensor for higher accuracy when calibrating color sensing. The black paper reduces external ambient light from interfering with the color sensor during calibration.

Section 3: Description of Our Overall Algorithm

Code in C Programming Language

1. Configurations

The code begins by defining the connections of the sensors and where they should be accessed on the mBot's mCore. Definitions made are the left and right IR sensors, the light dependent resistor, light emitting diode (LED) and the buzzer. Given the parts of the mBot, the source code assigns functions to activate the respective mBot parts which are the left DC motor, the right DC motor, the line finder, the ultrasonic sensor, the red, green, blue (RGB) LED and the buzzer.

Movement definitions include the motor speed, the amount of time needed to turn 90 degrees either left or right, the time to travel 1 grid box, the delay required to ensure the mBot senses its location in the maze, the time multiplier for the IR sensors, the correction of path of travel should the mBot move too close to the wall, the distance the mBot is away from a front facing wall and the threshold values of the IR sensors.

Definitions were then used to define the colors obtained during the calibration of the color sensors. The colors are Black, Green, Red, Yellow, Purple and Blue.

Calibration definitions included were the delay between IR sensor and color calibration, number of color samples to be taken from the maze, the delay between IR sensor measurements, the delay between each LED flash to take color samples, the delay between each LDR measurement and the maximum values of each RGB component.

2. Global Variables

Global variables allow the mBot to access these variables easily as they are defined globally. Meaning that these variables are accessible anywhere on the source code.

The boolean busy variable signals the mBot to keep moving straight as long as there are no obstacles or waypoint challenges.

The blackArray stores the value of black color.

The greyDiff stores differences between black and white samples to ensure better accuracy when detecting the color.

The allColourArray stores the values of corresponding tho waypoint challenge colors.

3. Main Program

The main program takes the definitions done previously in the configuration and calls the functions from the mBot library to put into action the movement of the mBot. In the main program, functions are taken from the libraries <MeMCore.h> and Notes.h. Note that in the main program, there are functions with "//" beside it. Functions have (); . This is to assist in re-calibration because when "//" is removed, we can activate the calibration function. Overall, the main function sets up the mBot, starts it moving, stops it when faced with an obstacle or waypoint challenge, detects the color at the way point challenge and when it detects black color at the way point challenge, it will stop and play the celebratory tune.

4. Movement

The movement section of the source code comprises functions to execute a stop, adjust left and right motor speed, move forward, turn right, turn left, turn a double right, turn a double left and make a UTurn.

5. Sensors

The sensors section utilises the color sensor on the mBot and the colourArray that was initialised in the global variable to detect color based on measurements taken at waypoint challenges.

6. Waypoints

In the source code, the waypoints are being split into cases.

Case 1: Red - Turn left

Case 2: Green - Turn right

Case 3: Yellow - UTurn

Case 4: Purple - Double left

Case 5: Light blue - Double right

Final case: Black - Stop and play celebratory tune

Implementation Details

Color Sensing

```
void colorWaypoint(const int colourRes) {
  switch (colourRes) {
    case 1: turnLeft(); break;    // red
    case 2: turnRight(); break;   // green
    case 3: uTurn(); break;       // yellow
    case 4: doubleLeft(); break;  // purple
    case 5: doubleRight(); break; // light blue
  }
}
```

Comments:

As seen from the colorWaypoint function, at waypoints, measurements of the color samples are taken first with the switch() function. The colourRes variable stores the value of the corresponding to the colour sample and executes movement based on value of colourRes. For the color red, it makes a left turn. For the color green, it makes a right turn. For the color yellow, it makes a 180 degree turn within the same grid. For the color purple, it makes two successive left turns in two grids. For the color light blue, it makes two successive right turns in two grids.

Proximity Sensing

Ultrasonic Sensor

```
void moveForward() {
  if (ultraSensor.distanceCm() < D_FRONT) {
    stopRunning();
    return;
  }
```

Comment:

From the code, if the distance of the mBot from the front facing wall is too close, ultraSensor.distanceCm() will detect and the stopRunning() function will be called which stops the mBot from moving forward any further.

IR Sensors

```
if (leftReading < V_LEFTIR) {
    rightWheel.stop();
    leftWheel.run(-MOTOR_SPEED);
    delay(new_delay);
  }
  else if (rightReading < V_RIGHTIR) {
    leftWheel.stop();
    rightWheel.run(MOTOR_SPEED);
    delay(new_delay);
  }
```

Comment:

V_LEFTIR and V_RIGHTIR are the values of the IR sensors when the mBot is travelling in the middle of the maze, equidistant from the left walls and right walls. The variables leftReading and rightReading are real time values when the mBot is travelling in the maze. Should the mBot travel too close to either the left or right walls, the code stops the corresponding wheel to make either a left or right turn to move away from the wall.

End of Maze

```
if (colourRes > 0) { // is color challenge (not black)
      colorWaypoint(colourRes);
      busy = false;
      return;
    }

    // finished
    finishWaypoint();
```

Comment:

In the above code, as long as colourRes is greater than 0, the colour detected is not black which signals to the mBot that it is not at the end of the maze yet. When the mBot detects black, colourRes will return a value of 0 and the mBot will call finishWaypoint(), indicating that it has detected that it is at the end of the maze.

```
void finishWaypoint() {
  // keys and durations found in NOTES.h
  int wholenote = (60000 * 4) / tempo;
  for (int i = 0; i < notes * 2; i = i + 2) {
    // calculates the duration of each note
    int divider = melody[i + 1];
    int noteDuration = 0;
    if (divider > 0) {
      // regular note, just proceed
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      // dotted notes are represented with negative durations
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5; // increases the duration in half for dotted notes
    }

    // we only play the note for 90% of the duration, leaving 10% as a pause
```

```
    buzzer.tone(BUZZER_PIN, melody[i], noteDuration * 0.9);

    // Wait for the specified duration before playing the next note.
    delay(noteDuration);

    // stop the waveform generation before the next note.
    buzzer.noTone(BUZZER_PIN);
  }
  busy = 1;
}
```

Comment:

At the end of the maze once the black colour sample is detected, finishWaypoint is called and the mBot plays a celebratory tune.

Steps Taken for Calibration

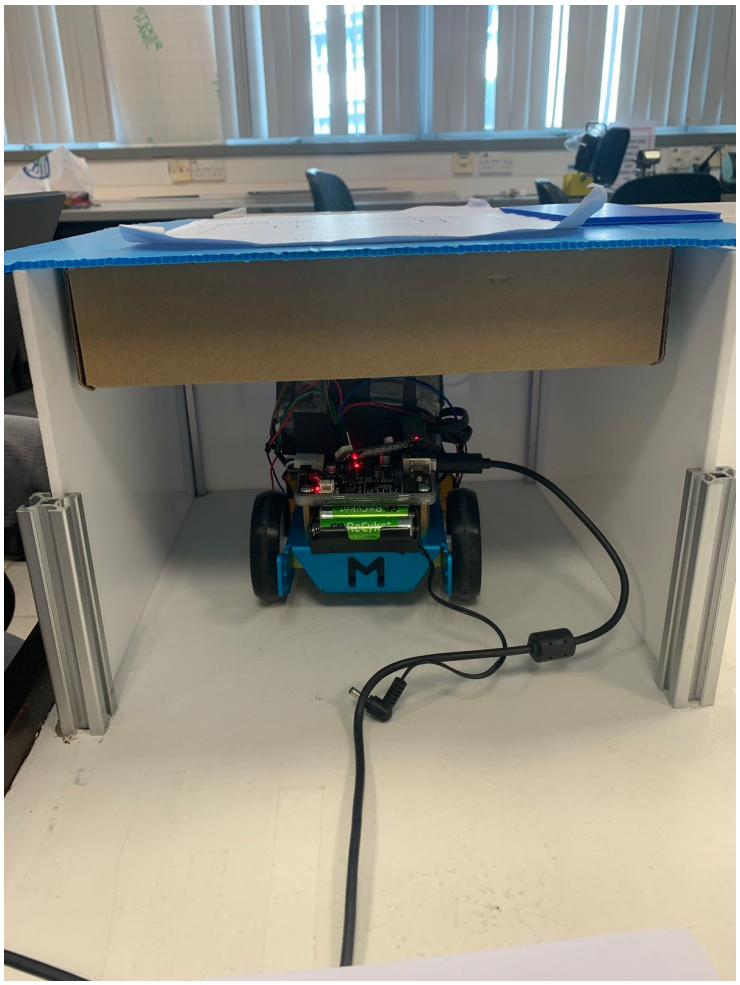

Figure 6. *The mBot during the color calibration using blue colored paper attached to the underside of the cardboard box*

Figure 7. *The mBot during calibration of the ultrasonic sensor using the sample white wall used in the final assessment*

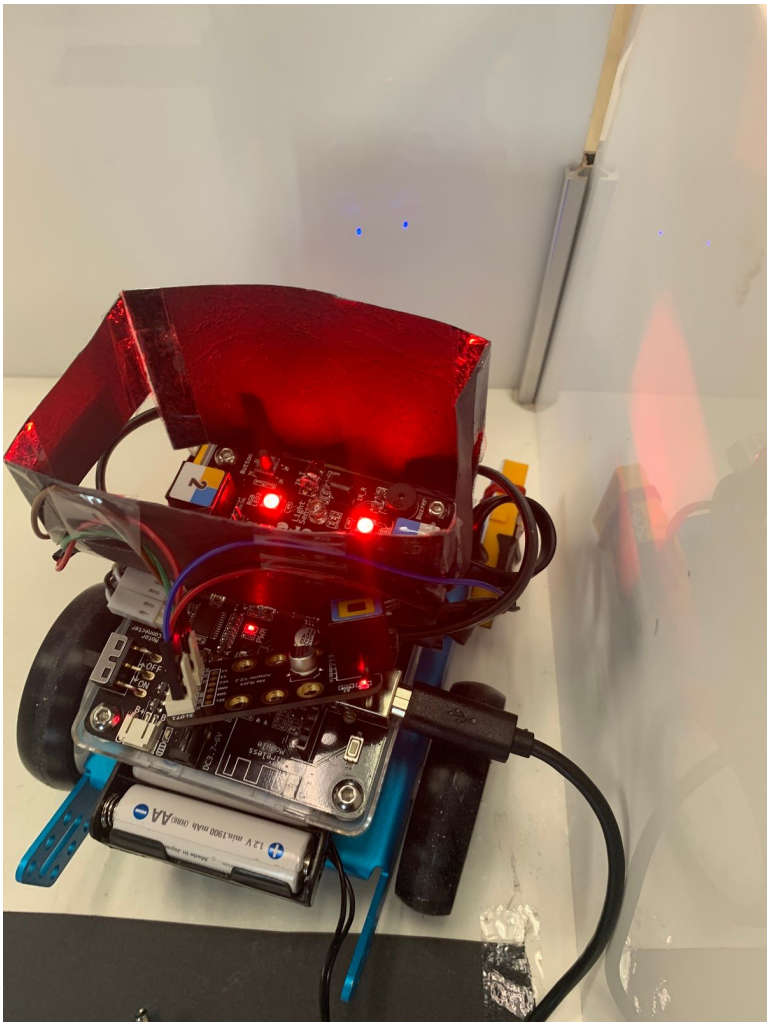

Figure 8. *The mBot during calibration of the IR sensor using the sample white wall used in the final assessment*

Color Sensing (refer to Figure 6.)

Paper of different colors being: Black, White, Red, Green, Yellow, Purple and Light Blue are being placed above the mBot for it to detect the color of the paper and convert it into numerical values via the source code.

Proximity Sensing (IR Sensors) (refer to Figure 8.)

1. Place the mBot in the maze.
2. Ensure that it is equidistant from the left and right walls.
3. Take readings of the IR sensors. These readings are the base case whereby the mBot travels safely since it is furthest from the left and right walls.
4. Place the mBot at a distance from the left wall whereby it should make a correction turn and take the value. This is the value when the mBot should make an adjustment as it is too near the left wall.
5. Repeat step 4 for the right wall.

Ultrasonic Sensor (refer to Figure 7.)

1. Place the mBot at a distance from the sample white wall where we deem it is too near and want it to stop.
2. Take the reading of the ultrasonic sensor at this distance from the wall.
3. Use this value in the source code for the stopping distance from wall of the mBot.

Section 4: Work division within team

NGUYEN TIEN KHOA

Being highly proficient in programming, he has directed the compilation and debugging of the source code for the mBot. The success of the mBot of the team has been highly reliant on his coding skills.

ONG HAN QIN

Having prior experience with Arduino, he collaborates with Nguyen Tien Khoa for the calibration of the mBot and adjustment of the source codes such as during the calibration of the IR sensor and adjusting the code accordingly. He played a big part in debugging the source code of our mBot. Also assisted in soldering wires and building of the IR sensors.

ROYCIUS LIM YUANWEI

Has systematic and clear thought processes which has made him proficient in using the breadboard and connecting wires. Being skilled in cutting wires as well, he built the IR sensors and also assisted in the soldering of wires for mBot connections.

SEE JIAN HUI

Came up with the report outline. Also assisted in the cumbersome tasks in the early stages of mBot testing such as building the blackbox upgrade, connecting and disconnecting the mBot to the laptop whenever a code is changed and assisting in color sensor calibration of the mBot.

Section 5: Difficulties Faced and the Steps Taken to Overcome Them

Cables from the original box being not suitable and cable management.

Due to the stiff ends of the cables out-of-the-box, the team utilised the soldering iron in the lab to create longer and bendable wires within reach enough to connect the components of the mBot. In addition, the cables were dangling and proved a hazard when the mBot is travelling. In order to solve the problem, the team used black tape to stick the loose wires onto the mBot.

Calibration of IR sensors

In the initial stages of the IR sensor calibration, the mBot would not move in the correct direction despite being near a wall. The team pondered over if the error was due to a hardware or software issue. It was due to the calibration values of the IR sensor that caused the undesired speed of the mBot wheels. The rotation of the wheels were also wrong which led the mBot to turn in the wrong direction. The team realised that the code which takes care of the IR sensors were inverted and made the necessary change. After adjusting the value of the speed and wheel directions in the source code, the IR sensors work.

Calibration of color sensors

After obtaining values for respective color papers, the mBot does not work consistently as the detection of colors was unstable. We decided to modify the hat of the mBot to a narrower box which better focuses the beam of light from the mCore onto the sample. This allowed us to obtain more accurate samples.