THE A-MAZE-ING RACE PROJECT REPORT

CG1111A - Engineering Principles and Practice I

TEAM B01-S6-T1

Name	Matriculation Number
Bui Phuong Nam	A0244121X
Bian Rui	A0239073A
Gunit Mittal	A0206438E
Shin Jin	A0240216W
Brandon Owen Sjarif	A0245892R

1. Overall Algorithms and Implementation

1.1 Overall Algorithm

Figure 1 below shows the overall algorithms implemented on the mBot. When the mBot is turned on, it will start to move forward. As it moves forward, it will continuously check whether a black line is detected in front of it. If one is detected, the mBot would go through the colour detection process to check the colour of the colour paper placed below the mBot. It would then move according to the colour detected - except when the colour is white, as this indicates that the mBot has exited the maze. Hence, the mBot will run the command to stop the mBot.

If the black line is not detected, the mBot would check the distance between itself and any obstacles on its right using the InfraRed (IR) proximity sensor. If it is close enough to a right obstacle, the mBot would move slightly to the left. Conversely, if it does not detect anything on its right, it would then check the left distance using the ultrasonic sensor algorithm. Naturally, if there are any objects on its left, the mBot would move slightly to the right. Otherwise, the mBot will continue to move forward.

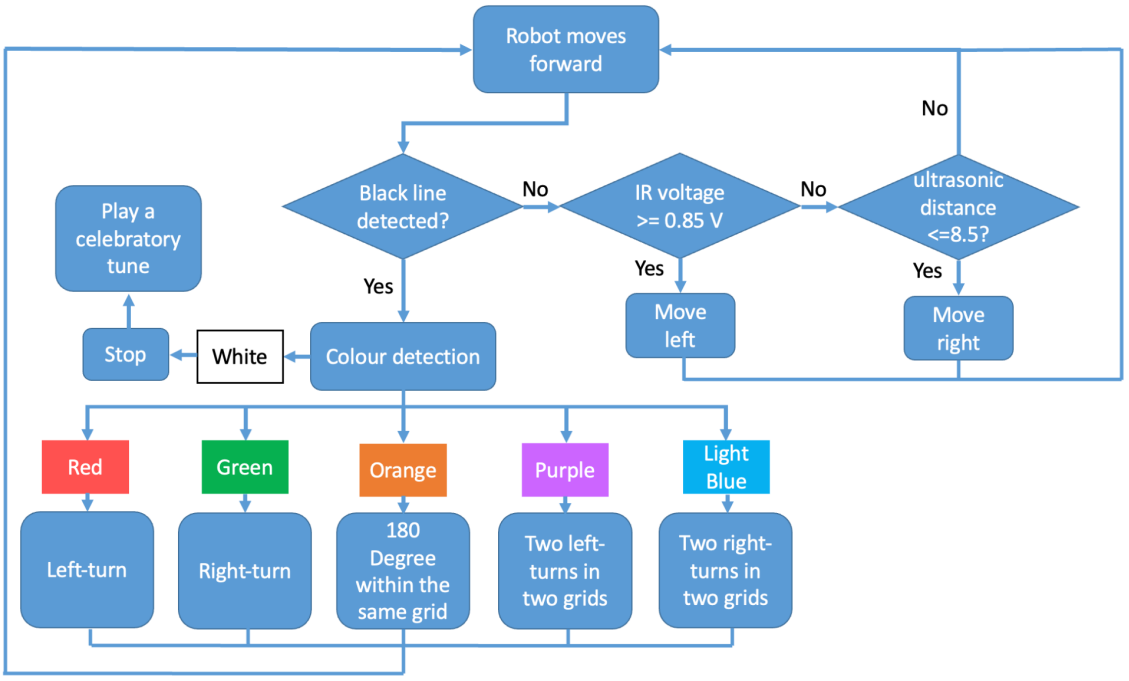

Figure 1: Algorithm Pseudocode

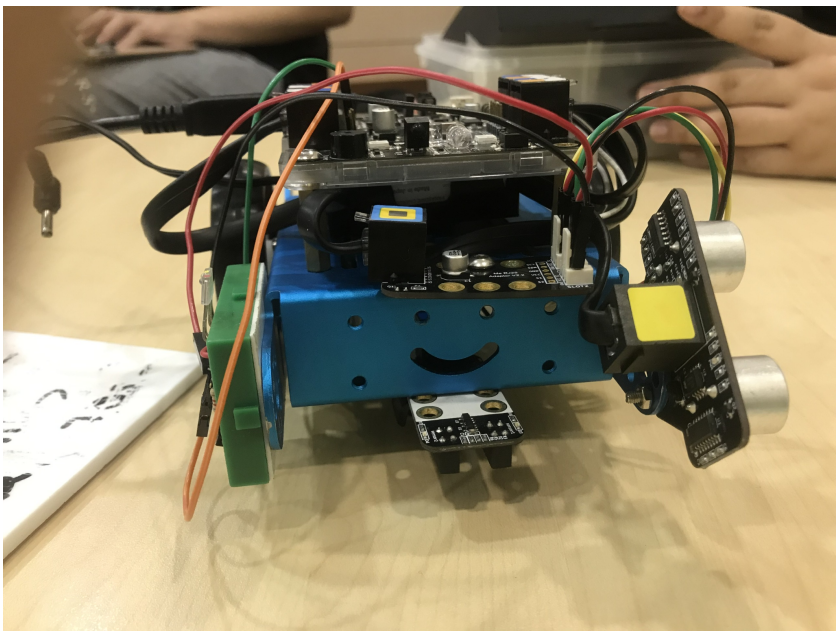

Figure 2: MBot's sensors' location picture

1.2 Keeping the Path of the mBot Straight

With the help of both the ultrasonic sensor and the InfraRed (IR) proximity sensor we were able to ensure that the mBot would travel in a straight path such that it would not run into any walls within the maze.

We used the main electronic circuit pictured below in order to manage and achieve the following algorithms.

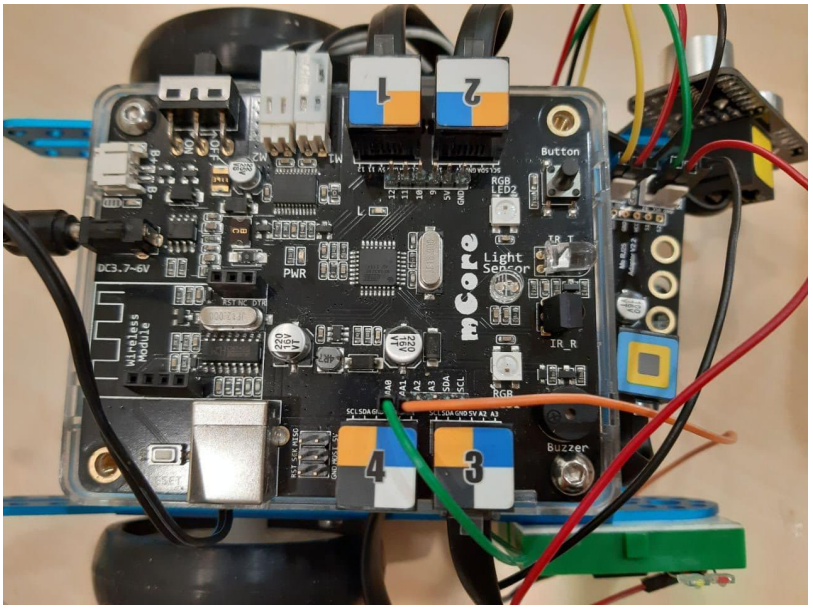

Figure 3: Main electronic circuit

1.2.1 Ultrasonic Sensor

The ultrasonic sensor was attached on the left-hand side of our mBot and detects the distance of the nearest object from the side of the mBot. The algorithm for this sensor was comparatively simpler than the other algorithms as it computed distance by emitting an ultrasonic wave and measuring the time taken for the wave to be bounced back from the object and detected by the receiver. The distance is returned from the function `ultraSensor.distanceCm()` in centimetres. A while loop that was used to repeatedly check for close surfaces and compare it to our set value - 8.5 cm - allowed the mBot to reorient itself and prevent itself from colliding with the walls on its left.

1.2.2 InfraRed (IR) Proximity Sensor

For this section of the mBot's detection, IR emitters and sensors were utilized to measure the distance between the right side of the mBot and the closest wall. This combination allowed us to effectively create an IR proximity sensor circuit to ensure that the mBot is able to detect and avoid walls such that it will be able to travel in a straight path.

As the right side of the mBot draws nearer to the walls, the voltage output from the IR detector increases. As seen by the breadboard circuit for the IR sensor above, when the IR sensor reads 0.85 cm and below, this indicates that the mBot is too close to a wall, and hence it will run a command to steer the robot to the left. This turning takes place till the voltage is IR detection voltage is lowered, and finally becomes lesser than the IR_DETECTION_THRESHOLD.

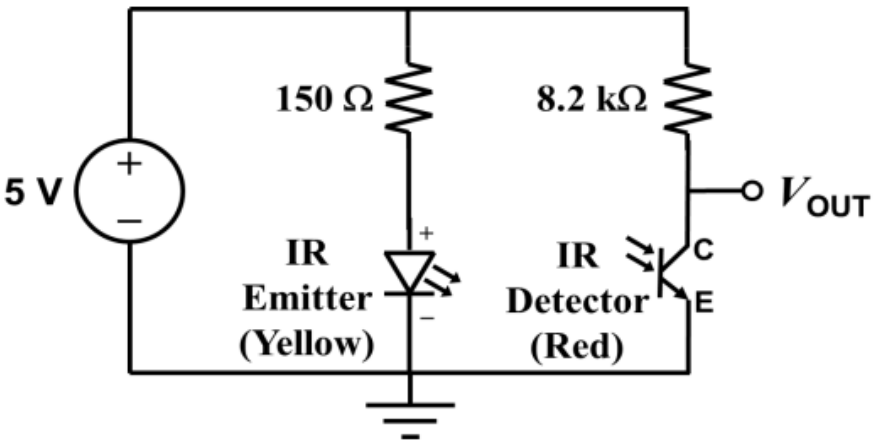

Figure 4: IR Sensor Circuit Diagram

This sensor circuit is based on the studio circuit we have previously been shown during labs. The physical circuit attached to our mBot is depicted below:

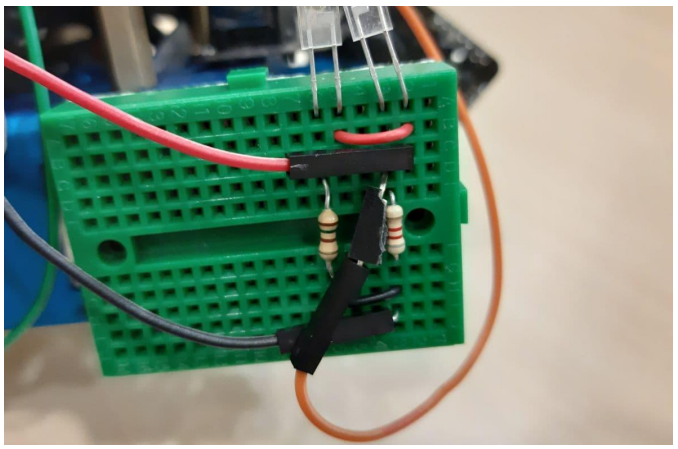

Figure 5: Physical IR Sensor Circuit

As for the code running the algorithm, we used simple comparisons to pre-calibrated thresholds. The algorithm is as follows:

STEP 1: Use the analogRead() function to read in raw values of the voltage across the detector (receiver). One thing to note is, unlike the studio circuit the voltage value is inversely proportional to the object's distance from the wall.

STEP 2: Compare the read values to the threshold values that have been pre-calibrated at around 8.5 cm from the sensors. In our case, the left was 0.85V.

STEP 3: In the case that the value read is less than the threshold, we execute a slight turn in the appropriate direction.

This along with the Ultrasonic Sensor prevents the mBot from crashing into the walls and getting stuck.

1.3 Colour Sensing Algorithm

Colour sensing in our mBot was done by measuring the amount of red, green, and blue light that the colour paper reflects. This reflected light will then be detected using the light-dependent resistor (LDR) in the built circuit. The resistance of the LDR changes with different amounts of light, which changes the amount of voltage across it - a property of which will be useful in colour detection. This technique was further improved through the effective implementation of an in-house circuit built by the team instead of the mBot's pre-built RGB LED and light sensor.

Our color sensor circuit consists of a common anode RGB LED lamp and a LDR. The power of each RGB lamp is controlled via a HD74LS139P 2-to-4 Decoder IC Chip, emitting red, green, and blue light each. The assembly of the circuit is shown in figure 6.

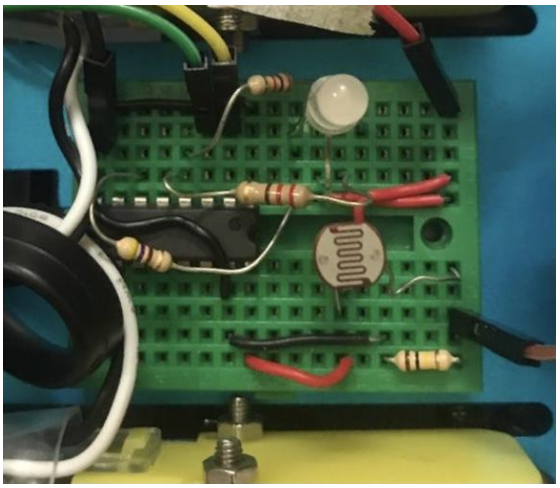

Figure 6: Colour Sensor Circuit Diagram

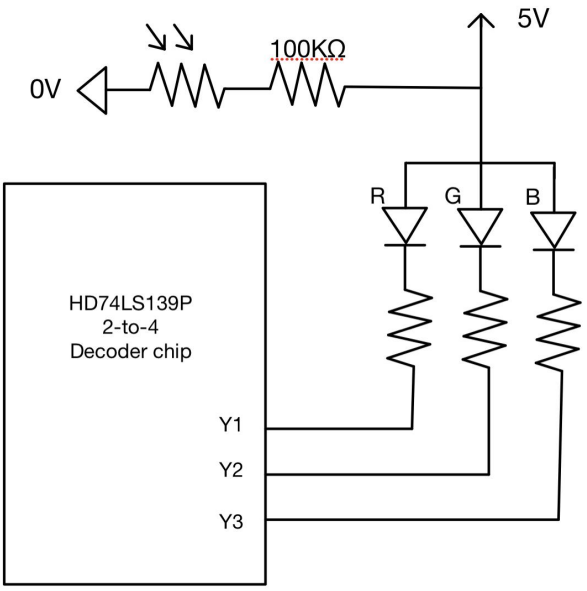

Figure 7: Colour Sensor Circuit Diagram

The different amounts of light reflected back into the LDR was used to describe the characteristics of specific colors and are read as 10-bit values. By shining the RGB lamp to a white and a black sheet, two sets of 10-bit values are measured from the LDR voltage in response to the reflection of the RGB light on the black and white papers. The two sets of values are stored in `whiteArr[]` and `blackArr[]` respectively for later usage.

The reflection of RGB light on other colors will produce a set of readings between black and white since white reflects the most RGB light and black absorbs the most. This set of values is stored in `colourArr[]`. A set of three RGB color codes of a specific color can be calculated by determining the position of `colourArr[]` in the LDR response gradient from white to black.

To achieve the above goal, an array called `greyDiff[]` was created and stored the differences between the entries of the same measured colored light in `whiteArr[]` and `blackArr[]`, denoting the range of LDR responses from black to white. Each value in `colourArr[]` is then deducted by those in `blackArr[]`, which is then divided by corresponding entries in `greyDiff[]`. The code that represents the above equation is:

```
for i = 0, 1, 2 do
     greyDiff[i] = whiteArr[i] - blackArr[i];
     colourArr[i] =(colourArr[i] - blackArr[i]) / greyDiff[i];
```

This results in a set of three values between 0 and 1, indicating the position of the LDR response on the gradient from black to white when the RGB LEDs are shone on the colored paper beneath the mBot. Scaling this set of values by 255 will give the RGB colour code of the detected colour. These RGB color codes can be further used to decide which colour is detected.

1.4 Movement of the MBot

For the movement of the mBot outside of the color challenge, we used a `forward` function in order to command the movement of the mBot within the straight portions of the maze. As seen in Figure 1, the mBot will first check if the line-follower sensors read a return value of 3. This means that both sensors are detecting a black strip, and thus the mBot will use the colour detection algorithm to try and detect colour paper beneath it to decide to turn or end the maze. If a black strip is not detected, it will then check for any objects on its right using the IR proximity sensor and objects on its left using the ultrasonic sensor. If there are none, it will continue moving forward.

As for the movement within the color challenge portion of the maze, we created a `movements` function in order to simplify any form of movement that we command the mBot to do. Using `switch` and the color detection algorithm, we were able to call a `turn` function multiple times in order to turn the mBot according to the color it has detected.

1.5 End-of-Maze Detection

Throughout the maze, the mBot will keep checking to see if there is a black strip and the white-colored paper under it to indicate the end of the maze. When it does detect this point, it will run the function in order to stop the movement of the mBot and play the ending music we have chosen. To keep in the theme with the futuristic nature of this project, we have decided on the Star Wars theme to signal the ending of the maze.

2. Steps Taken for Calibration

2.1 Distance Sensing and Movement

To ensure that the mBot would maintain a straight path we had to measure the distance between the mBot against the left and right walls. Through these distance values, we were able to implement and calibrate the required distance that the mBot would have to maintain against the surrounding walls. For the ultrasonic sensor, we would only have to key in the required distance in the program code, however, it was not so simple for the IR proximity sensor. Using the distance values we measured, we needed to place a temporary board that was the same distance from the right side of the robot and measure the amount of voltage outputted by the IR sensor. From this, we were able to identify the appropriate voltage to be compared to by the mBot. Ultimately, we decided to keep the robot at least 8.5 cm from the left wall and 8.5 cm from the right wall, which requires voltage output from IR to be below 0.85 V.

As for the calibration of turns taken by the mBot, it was unfortunately mostly trial and error. Given that we knew that the turning radius would change depending on the amount of voltage outputted by the battery, we had to calibrate the amount of motorspeed and duration whilst the batteries on the mBot were at full capacity. From this, through multiple tedious estimations of motorspeed and duration, we were able to correctly identify the appropriate degree of turning for the mBot to turn right-angles.

2.2 Colour Sensing

To determine the range of each colour's RGB colour code, the 10-bit LDR voltage readings from reflection on white and black must be obtained first. The RGB lamp emits red, green, and blue light on a piece of white paper placed under it. The reflection of each colour light causes different voltage responses in LDR. The voltage response is extremely sensitive to the power source voltage so we ensure the batteries are fully charged before calibration. Three 10-bit values are read and stored in `whiteArr[]` from the LDR to note its voltage responses to RGB lights reflected from white paper. The same procedure is implemented on black paper and the three readings are stored in `blackArr[]`.

With `whiteArr[]` and `blackArr[]`, `greyDiff[]` can be calculated. The same procedure is implemented on red, orange, green, light blue, and purple color paper, and the results are stored in `colourArr[]`. These results are then converted to the RGB color codes detected for each color through the method mentioned in 1.3 and stored in `colour[]`. RGB colour code for white is simply stored as `{255, 255, 255}` in `colour[]` as well. If a detected colour code is closest to a color code in `colour[]`, it will be recognized as that color.

3. Work Division

At the beginning of the creation of the mBot, our group members were given tasks pertaining to the main algorithms and construction of the mBot.

The division of the group work amongst our team members are as follows:

- Bian Rui: Colour detection algorithm, circuit assembly and report writing.
- Gunit: Ultrasonic sensor algorithm, assembly of the mBot and compling code.
- Nam Bui: IR proximity sensor algorithm and circuit assembly.
- Brandon: Report writing.
- Jin: Report writing.

Whilst this was working great at first, there were specific algorithms that were severely more challenging and took more time than others. An example of which is the colour-detection algorithm as we realized that there were severe problems in the accuracy of the colours being detected. In the event that any group members had any difficulties with their algorithm, the other group members would try to help out.

4. Significant Difficulties and Steps to Overcome

4.1 Covid-19

Due to the overlooming presence of the Covid-19 pandemic, most of our group members were unable to attain approval in order to enter Singapore since the beginning of the semester. Hence, we lacked the practical experience needed to efficiently handle as well as construct the circuitry on the breadboards. To solve this, more trial and error was needed in order to efficiently and elegantly build the circuits.

Additionally, due to the lab restriction of only having 3 people per group be able to enter during lab hours, reduced our effective communication as members were excluded from joining the lab session. This made it more difficult for our group to effectively use lab resources. To counteract this issue, we held more physical meetings outside of lab hours to work on the mBot together as a group.

4.2 Lighting Changes

One of the major problems that hindered the successful execution of our robot was the differences in lighting and the presence of shadows that came with it. Whilst the mBot would be able to successfully run in our meeting rooms, we knew that in a brighter location such as the lab, it would be a significant hindrance in the detection of the color paper underneath it.

Originally, we intended to use the intervals between the RGB values of the different colors of the papers. By measuring the intervals of RGB values of the paper, the color-sensing algorithm would be able to differentiate the colors and run the function needed to turn it in the right direction. However, this solution was extremely prone to ambient light and we were not able to effectively implement it into the mBot.

To overcome this, we used a different approach. Through calibrating the mBot using black and white paper whilst in the lab, we were able to obtain the calibrated readings necessary in our code. The mBot would then be able to detect the colour paper using the gradations between the ranges from the values reflected on the white and black paper. This method proved to be effective in allowing the mBot to determine the colors with reliable accuracy in our testing.

Unfortunately, in the final run, the colour sensor was still not accurate enough in differentiating between red and orange as well as purple and blue even though we have performed thorough calibration in the lab before the final run. This is largely due to the changing ambient light influencing the accuracy of colour detection. Although we did not succeed in the final run, a potential solution would have been to minimise the impact of ambient light by installing a black chimney around the colour sensor.

4.3 Turning Radius of the mBot and the Depletion of Battery

Another of the most significant difficulties we encountered was the turning radius of the mBot. Whilst it was initially difficult to estimate the amount of motor speed and the duration of the activation of the mBot, through trial and error, we were able to identify the correct values. However, this caused us to run into another issue, depending on the amount of voltage still present in the battery, the amount of voltage delivered to the motor will vary. Thus, the turning radius, as well as the speed of the mBot, will once again change.

The tiniest difference in the voltage present in the battery would cause the movement of our mBot to differ drastically. While it was possible to initially calibrate and set the battery to its maximum depletion as it would be more reliable and precise, we worried that it would take a much longer time to complete the maze.

4.4 Missing Walls

It is possible for the IR proximity sensor and ultrasonic sensor to fail to keep the car moving straight when either the right or left wall is missing. If the mBot is too close to the right and the IR proximity sensor cannot detect the missing right wall, it will not be able to steer left to keep the path straight. We overcame this issue by checking whether the mBot is too far from a wall. In addition to checking if `ultraSensor.distanceCm()` is smaller than 8.5 cm, we also check if it is larger than 13 cm in the case where the mBot is too close to the right. However, if the distance detected exceeds 30 cm, it indicates that the left wall is missing and thus the mBot should keep moving forward. The same approach is implemented on the IR proximity sensor. If IR voltage drops below 0.6 V, the mBot is too close to the left and when IR voltage drops below 0.4 V, this means that the right wall is missing, hence causing the extremely low voltage output from the IR proximity sensor. This algorithm helps the car to move straight when one of the walls is missing.