The A-maze-ing Race

CG1111A Final Project

Dexter Tan, Thaw Tun Zan, Tran Duc Khang and Prannaya Gupta

B02 Section 6 Team 2

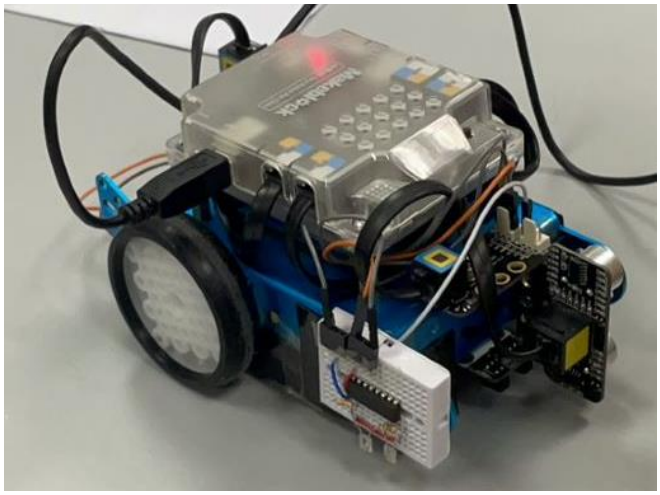

Table of Contents

Approach 2
Our Design Process 2
Overall Design 2
Colour sensor design 3
IR Detector-Emitter Design 4
Programming Methodology 5
Forward-Backward Movement 5
Rotation 6
Course Correction 8
Solving the Challenges 10
Challenges Faced 11
Challenge #1 – IR Sensor Randomly Switching Off 11
Challenge #2 – The Battery's Voltage 11
Work Distribution Matrix 12

Approach

For this project, our team took a two-pronged approach. Through splitting up the workload into hardware and software, we were able to maximise efficiency. With greater proficiency in Arduino, Khang and Prannaya oversaw the coding while Khang, Dexter and Thaw were responsible for assembling the components[1].

Our Design Process

Overall Design

Based on the "The A-maze-ing Race Project 2022" documentation, four components determined our mBot's manoeuvrability, namely the line sensor, ultrasonic sensor, colour sensor and IR sensor.

For the colour sensor, an LED array (red, blue, and green LEDs) with a separate light-dependent resistor (LDR) was used to isolate the colour. Skirting is relevantly placed around the chassis to ensure minimal interference. Once the line sensor detects the black strip, the colour sensor kicks in, allowing the robot to execute the appropriate turn.

The infrared emitter-sensor pair and the ultrasonic sensor was used to isolate distance measurements, allowing us to maintain the distance between the bot and the wall while implementing relevant course correction.

However, while the line sensor and ultrasonic sensor were provided, the colour sensor and IR emitter-sensor pair had to be built from scratch.

[1] A more detailed **Work Distribution Matrix** can be found at the end of this report.

Colour sensor design

We design the colour sensor using a 2-to-4 decoder which ensures signals are decoded to light up individual LEDs. There are four individual settings for the LDR Colour Sensor, notably: `LDR_OFF`, `LDR_RED`, `LDR_GREEN` and `LDR_BLUE`. During the colour sensing, each LED is individually lit up for a second, and the LDR response from the light is explored.

The colour sensor is designed as follows:

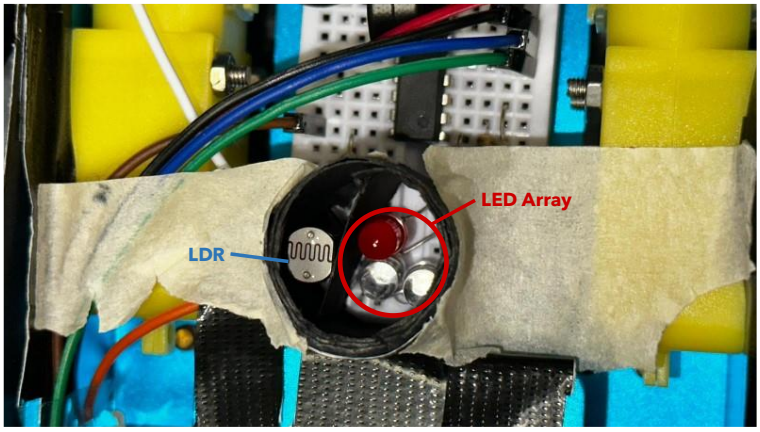

Figure 1: Colour Sensor

Some notable considerations made:

- **Shorter height of LEDs compared to the LDR.** To detect the colour of the paper, the LDR senses the intensity of light reflected off it. However, since light emitted from the LEDs travels in every direction, the height was made shorter to prevent light from LED from reaching the LDR directly. This eliminates false readings.
- **Two sets of black papers were used.**
    - One set is used to separate the LEDs from the LDR (Figure 1), which further prevents light from the LEDs from reaching the LDR directly.
    - Another set is used as skirting which encapsulates the LEDs and LDR bundle (Figure 1), as well as below the mBot chassis (Figure 2). This reduces the amount of ambient light reaching the LDR, minimizing systematic errors.

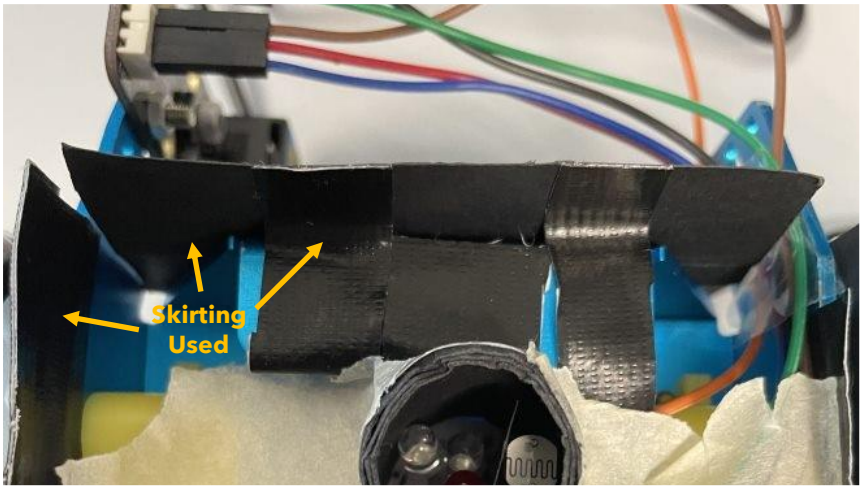

Figure 2: Additional Skirting Employed

IR Detector-Emitter Design

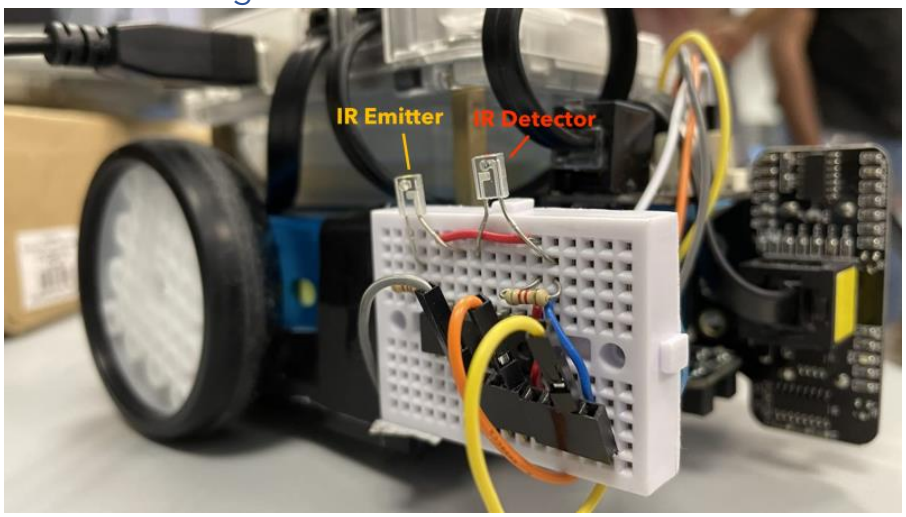

Figure 3: IR Detector-Emitter Duo on the right cheek of the robot.

A proper pass by value is used to determine when the wall is close enough to the IR board for the Course Correction protocol to kick in. This is because the current amplifier that is used to power the IR emitter and detector amplifies minor fluctuations in background Infrared Radiation as well, making it susceptible to systemic errors.

Programming Methodology

Forward-Backward Movement

Based on empirically derived forward speed (roughly 13cm/s as stored in the speed macro), our program uses a simple move function to move the mBot. For a given distance s, the time interval t can be derived using the formula $t = s/v$. The actual polarity of the distance, $sign(s)$ is then used to determine the velocity of motors and the elapsed time. These values are then used to run the function as shown in the code below.

```
#define V 200
#define SPEED 13

// ... later on in the code
void move(float distance) {
    float t = distance / SPEED;
    float v = distance > 0 ? V : -V;
    t = t > 0 ? t : -t;
    motor1.run(-v);
    motor2.run(v);
    delay(1000 * t);
    motor1.stop();
    motor2.stop();
}
```

Figure 4: Move function

V will be elaborated on later in **Rotation**, but for the large part we have designed a simple movement function that can be mathematically represented as $m(s)$, where, by default, $s = 20\ \text{cm}$.

To increase the speed of our robot, we employed a simplistic motor movement protocol. This kicks in when the robot is travelling in a straight line i.e., neither solving colour challenges nor correcting course. A local value is used to store the state of the robot at any time, and the loop function allows us to evaluate if the robot should be stopped, or not.

Rotation

We picked a high wheel motor "speed" to reduce the friction between the floor and the wheels and measured the time taken for the robot to spin 20 revolutions, after which, we can obtain the period of rotation T of the bot. From this, we were able to make the robot turn relatively precise angles by varying the time t we leave the motors on. As a result of this, human error was minimized.

$$t = \frac{T}{360°} \times \theta$$

As seen in the code, this is implemented as follows:

```
#define V 200
#define period 1800

// ... later on in the code
void rotate(float angle) {
    float t = (period / 360) * angle;
    float v = t > 0 ? V : -V;
    t = t > 0 ? t : -t;
    motor1.run(v);
    motor2.run(v);
    delay(t);
    motor1.stop();
    motor2.stop();
}
```

Figure 5: Rotation function

The code extrapolates the time to elapse, t and speed at which the motors should be turned, v. This v is a function of the polarity of the angle, $sign(\theta)$and a centralised general motor speed, V, which is used to isolate the central direction of motion.

The general motor speed is represented by a relatively arbitrary value between 0 and 255. However, due to the friction and the weight of the mBot, we noticed that this speed is not what is followed by the mBot. We realized that the robot does not perform well at low and high speeds, and we chose to go with $V = 200$ as it is a higher speed.

Note that when $\theta > 0$, the robot rotates left (as mentioned in the code documentation). This means that for the "Red" set-up, $\theta = 90°$, while for the "Green" set-up, $\theta = -90°$. This ensures that proper results are received.

The motors are then run at this speed for the given time t. Since the motors are laterally inverted, having the same speed ensures that we can rotate it about a stationary central axis (which is located right in between the two wheels). Due to the sheer weight of the mBot, the bot does not slide off when rotating about the central axis, although certain inconsistencies are still observed, which can cause the robot to slightly bump against the walls. This problem is later explored in **Course Correction**.

This allows them to be run for a specific duration at a relatively consistent angular velocity ω. Thus, we can denote this function mathematically as $r(\theta)$, where θ is the angle to be rotated. We will use this notation in the rest of the report.

An example of this is shown below (this is under normal working capacity):

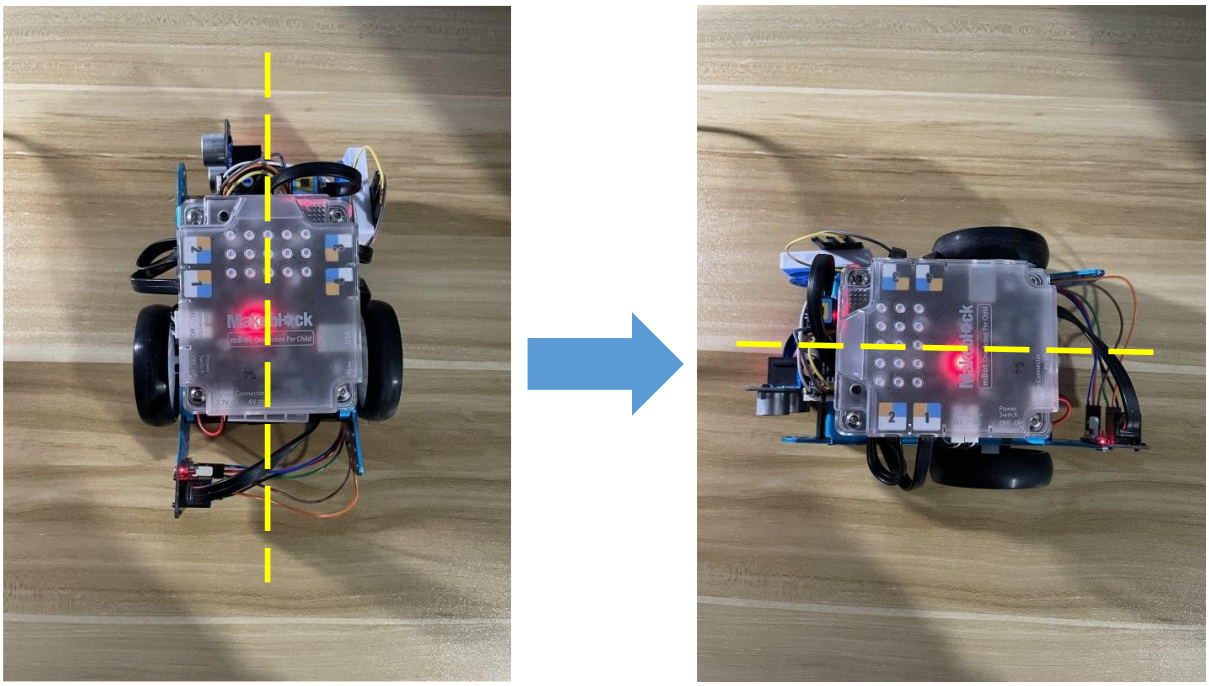

Figure 4: An illustration of $r(90°)$, which is slightly inaccurate under normal battery conditions

Notably, one issue with the rotation of the mBot is that the precision of the rotation depends on the battery level. However, based on several time intervals of usage, we notice that the depletion of the battery power does not evidently affects the performance of the mBot until the very end, and only a miniscule snowball effect is observed. Thus, we keep our faith in the efficacy of the rotation function.

Course Correction

Due to frictional effects, many times, the wheel is unable to turn properly or fully. This incomplete rotation can lead to problems with maintaining the straightness of the robot's course. Hence, we need to develop a course correct algorithm which can allow the mBot to correct itself. This is done via a slight perturbation to the heading angle of the robot, based on signals from the ultrasonic sensor and IR sensor. We can mathematically denote this perturbation as the angle, α.

Our course correction algorithm builds upon the turning function, and is based on three underlying details as follows:

- The width of each box is ~ 20 cm.
- The Infrared Sensor is used as a secondary measure in ideal circumstances due to fluctuating background radiation under differing conditions leading to its inaccuracy.
- The ultrasonic sensor gives accurate results and should be placed roughly 3-3.5 cm away from the central axis.

Hence, we use the ultrasonic sensor as a primary source to determine the distance from the wall, followed by the IR sensor, which is used minimally. The chain of command follows as below:

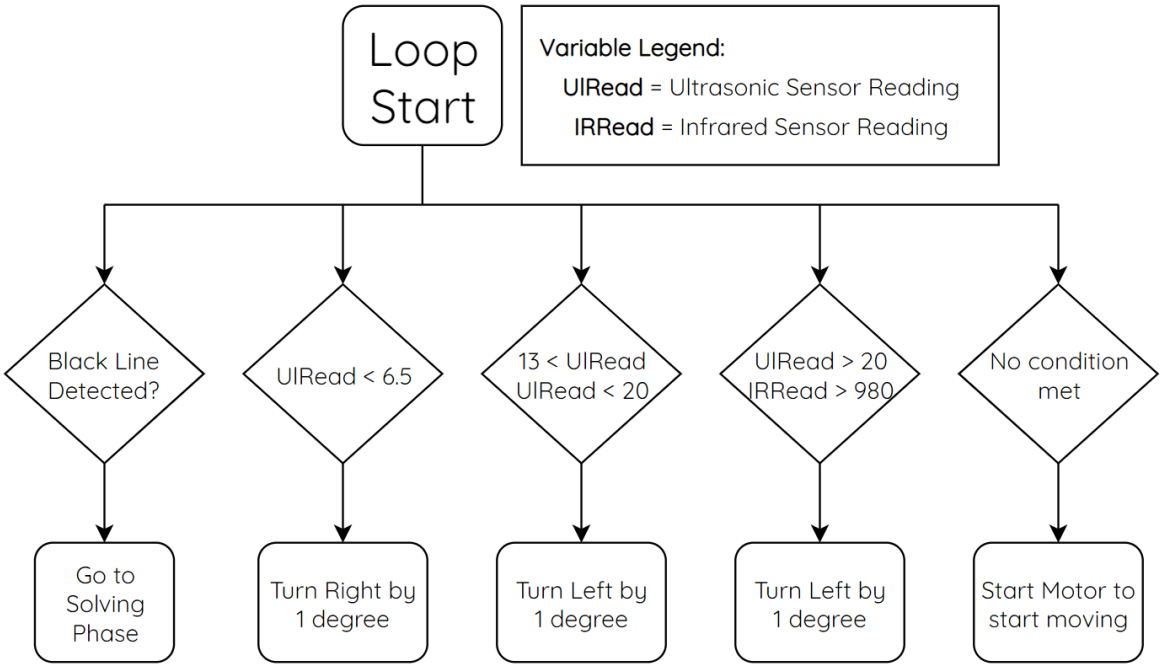

Figure 5: Control Flow of the mBot for every iteration run

A small angle of 1° was chosen for movement due to the general frequency of the loop function. This is due to the general frequency of the loop function, which ostensibly causes amplification of minor perturbations due to positive feedback, resulting in large errors.

This can hence be visualised by the following figures, which highlight two different cases of note:

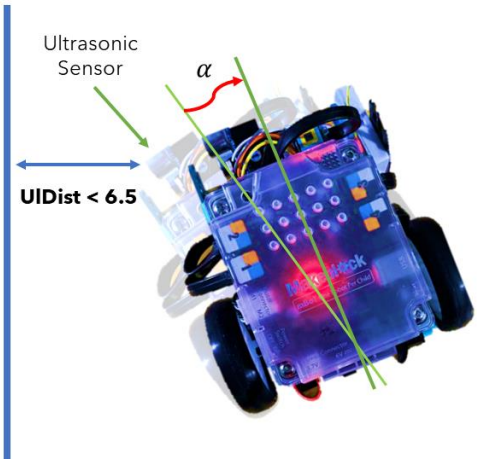

Figure 6.1: Ultrasonic Sensor is very close to a wall

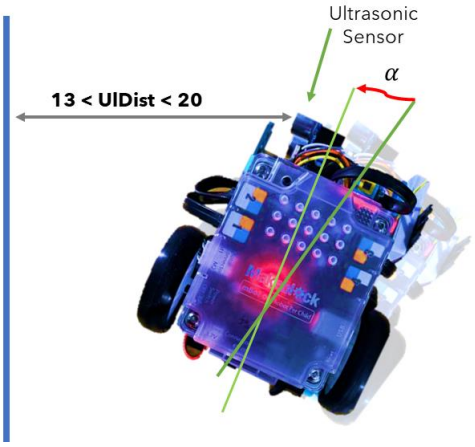

Figure 6.2: Ultrasonic Sensor is very far from the wall

Figure 6: Course Correction Corner Cases

In the case of this project, $\alpha = 1°$. We use this process to ensure that the bot turns away in case it is approaching the robot. An example of this algorithm (specifically Case 1 above) in real life is show below:

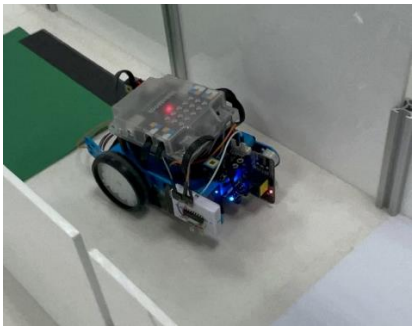

Figure 7.1: Initial State of the mBot

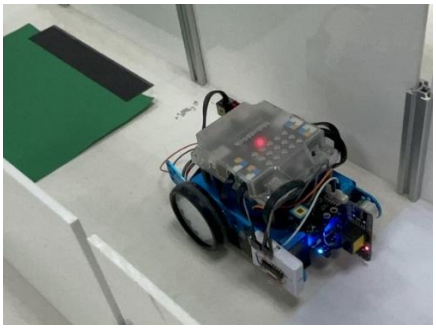

Figure 7.2: Corrected State of the mBot

Figure 7: Course Correction Process in Real Life

Solving the Challenges

To solve the challenges, we adapt a simple framework of utilising the colour sensor to detect the colour. This is done by leaving each LED switched on for a single second separately, and individually reading the analog reading from the LDR every second. This allows us to get individual readings for each colour's reflection.

The 1-second interval is largely chosen due to the nature of the LDR Response, which seemingly fluctuates based on an excitation curve. After a second, this reading largely stabilises, which allows us to get a proper reading, which can be scaled to get an RGB value.

Following colour detection, we use a minor pipeline, as illustrated below, to dictate the movement of the robot.

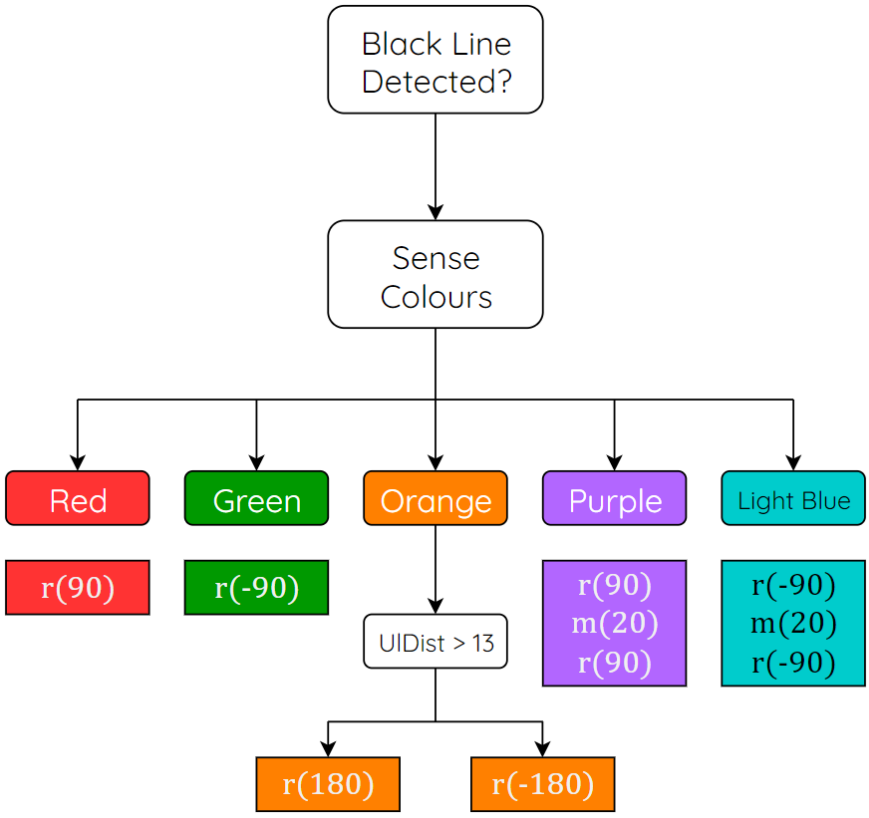

Figure 8: Colour Sensing Pipeline

Not Visualised: At white, the robot plays "Imperial March" from *Star Wars*.

One notable consideration made is when orange is detected. Since the robot needs to turn a total 180° loop, we can choose to either have the robot turn left or right. To decide, we detect the distance from the Ultrasonic Sensor, and if this is a large distance, we turn left since there is more space towards the left. Otherwise, it turns right. This largely summarises our programming methodology and pipeline.

Challenges Faced

Challenge #1 – IR Sensor Randomly Switching Off

The IR Sensor kept randomly switching off throughout the majority of the testing process, which led to our robot being unable to correct the course properly. For instance, this led to the robot completely turning at red challenges, since the IR would give a very random value which is sometimes greater than 980 (the background radiation determined). This means that it turns 90 degrees to the left without even reaching the black line, which is completely unexpected. We hence moved the IR reading to a secondary measure since this was problematic for us.

Challenge #2 – The Battery's Voltage

We noticed that when the robot is exposed to higher amounts of friction, the battery voltage determines a large amount of how much the robot turns. This means that if the course is not clean or the robot unfortunately experience a large amount of friction on the colour challenges, it will rotate slightly differently at different voltages. The battery itself drops in voltage very quickly, hence we made sure to discharge the battery to a degree where it obtains stability for a certain period, such that it can sustain itself, and measure the precise rotation and movement readings there.

Work Distribution Matrix

Task	Khang	Dexter	Prannaya	Thaw
Hardware (Building the Bot)				
mBot Base and Securing Ultrasonic Sensor + MeLine Follower	x			
IR Sensor	x	xx		x
Colour Sensor	xx	xx		x
Skirting	xxx	x		
Component Integration	x			
Software (Coding the Bot)				
Robot Movement (Rotation and Movement)	xx		x	
Robot States (MeLine Follower Integration + Colour Sensor Integration)	x			
Colour Sensor	x		x	
Course Correction	x			
Code Clean-Up			x	
Code Documentation			x	
Report				
Design Details		xx	xxx	x
Programming Methodology			x	
Diagrammatic Design + Data Collection and Analysis			x	

Legend:

- **Only one "x" in the row**: The member did all the work for that task
- **Multiple "x" in a row**: The number of "x"s under each member indicates their relative contribution