NUS
National University
of Singapore

CG1111A mBot Final Project Report

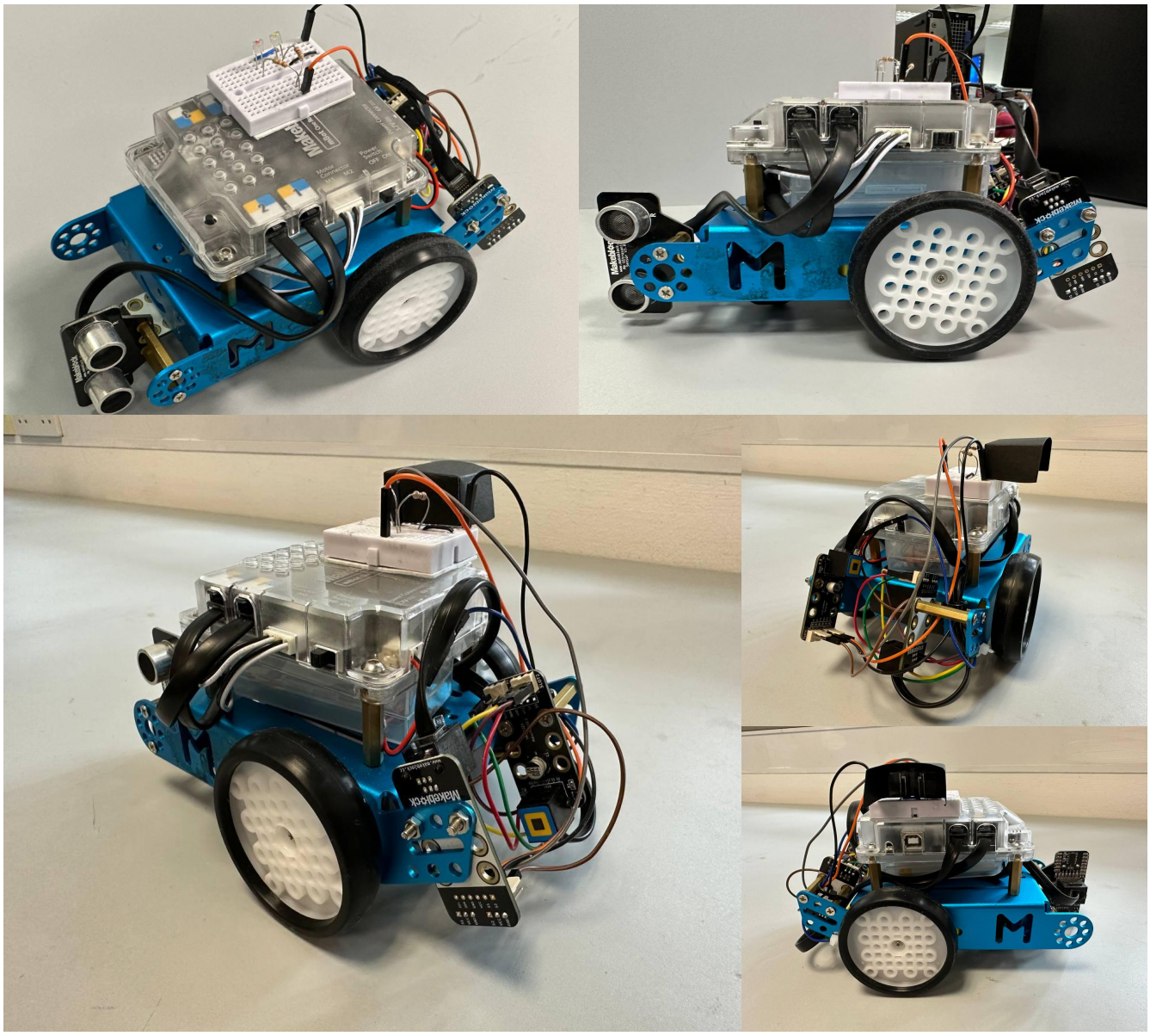
– Engineering Principles and Practice I –

Studio Group B03, Section 1, Team 1
Members:

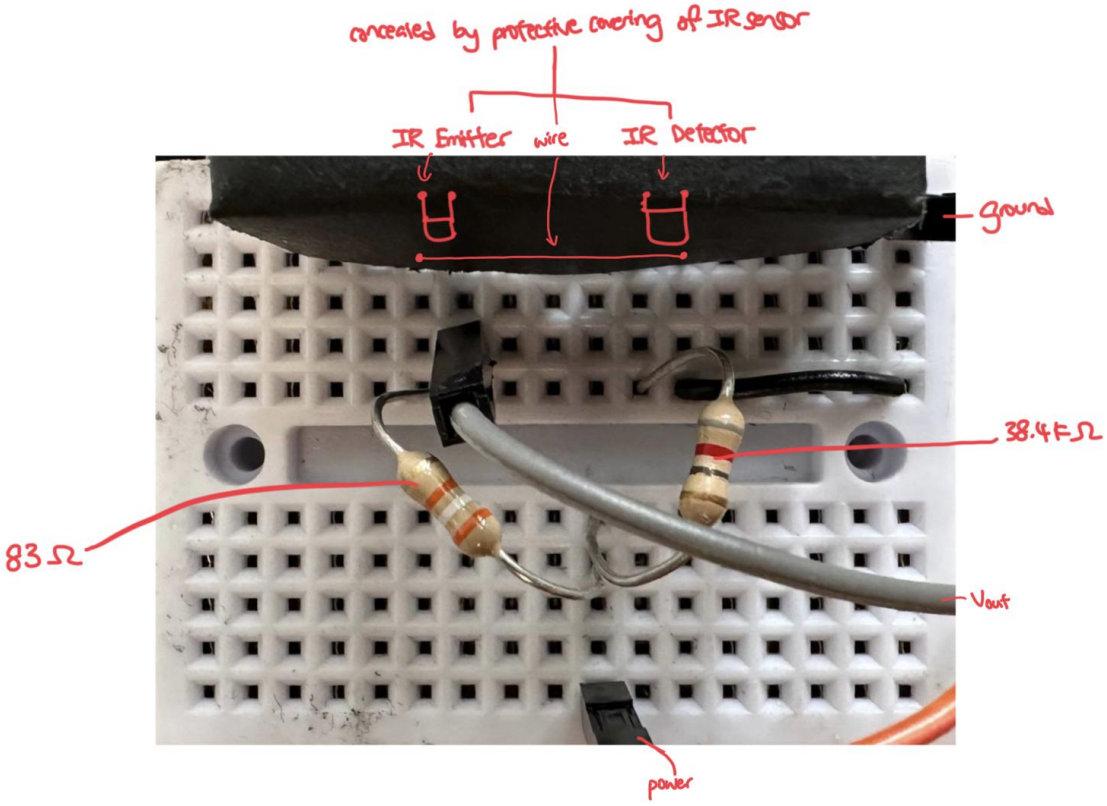
- Cervon Wong Teng Hao (A0261162N)
- Eu Zheng Xi (A0233767W)
- Jayson Ng (A0266095W)
- Gandhi Parth Sanjay (A0252403U)

Table of contents

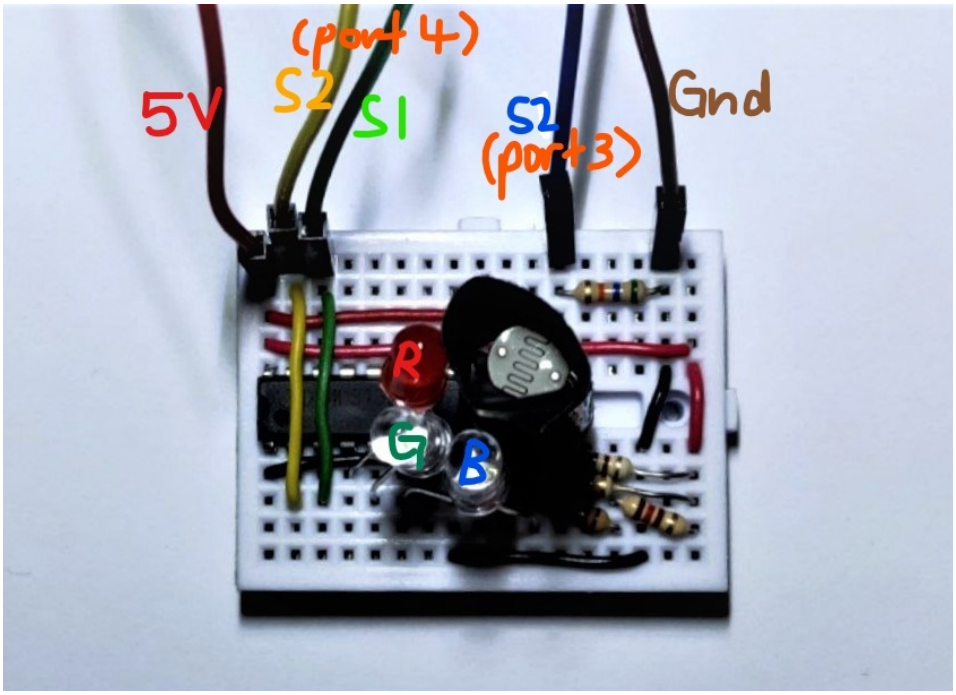
PICTURES OF OUR MBOT AND CIRCUITS



Pictures of the mBot

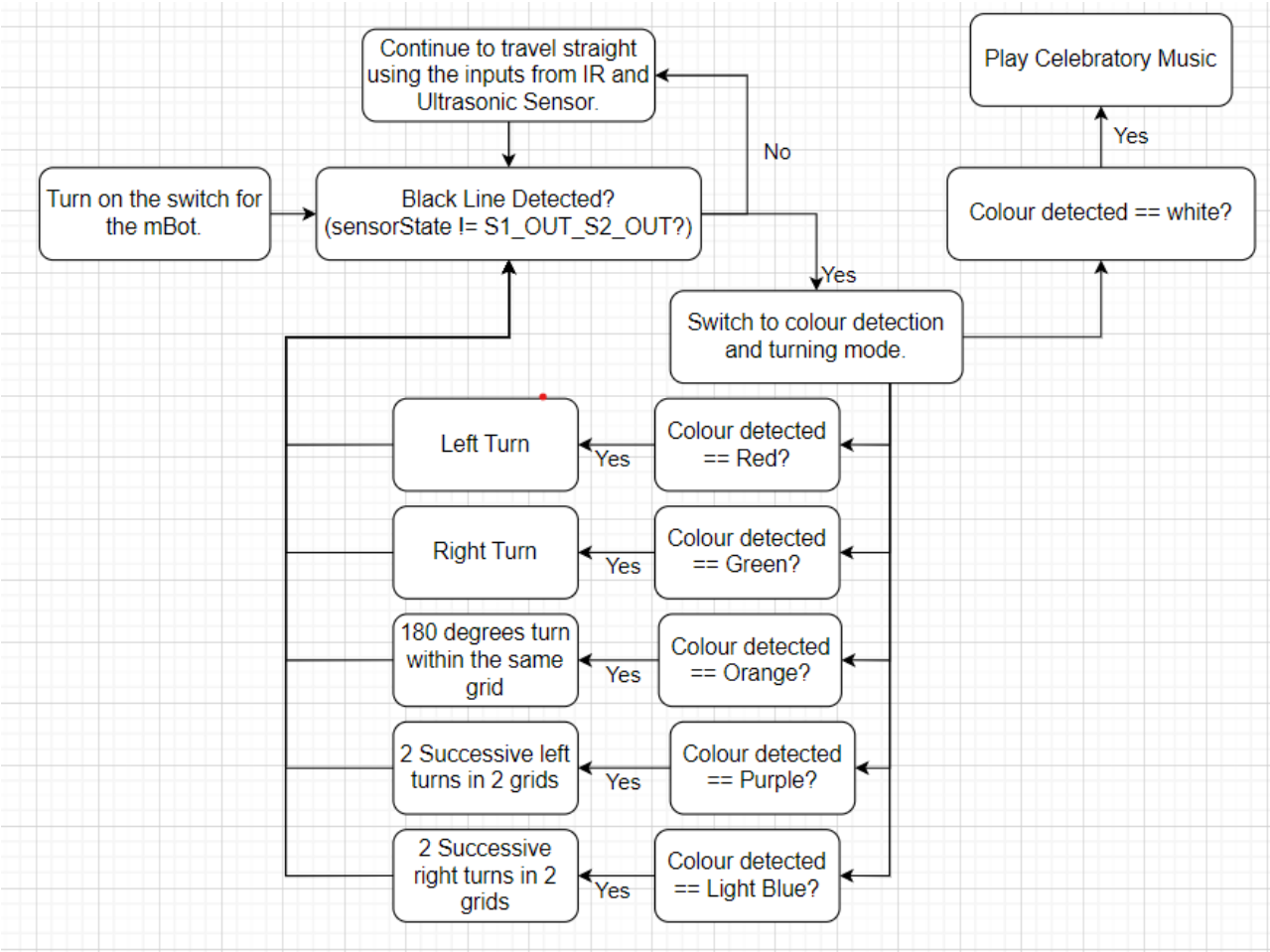


Annotated circuitry for the IR sensor



Annotated circuitry for LED sensor

ALGORITHM



Overall Algorithm of the mBot

To initiate the mBot run for the maze, we place it on track and switch it on. The mBot will then run and go through a series of challenges. When it detects a black strip, it stops and detects the colour and initiates either one of the turns. If it's white, the mBot plays the celebratory music to indicate the end of the maze. Otherwise, it continues to travel ahead and checks for black lines until it reaches a white waypoint.

A. Straight Line Movement

```
if (ultrasonicDistance < 7) {  //USS: 10.7, IR:370,
  spinRightWheelForward(MOTOR_SPEED_R - 15);
  spinLeftWheelForward(MOTOR_SPEED_L + 30);
  delay(25);
} else if (ultrasonicDistance <= 10.0) {
  spinRightWheelForward(MOTOR_SPEED_R - 10);
  spinLeftWheelForward(MOTOR_SPEED_L + 15);
  delay(25);
} else if (infraredReading < 280) {
  spinLeftWheelForward(MOTOR_SPEED_L - 25);
  spinRightWheelForward(MOTOR_SPEED_R + 45);
  delay(25);
} else if (infraredReading < 370) {
  spinLeftWheelForward(MOTOR_SPEED_L - 15);
  spinRightWheelForward(MOTOR_SPEED_R + 25);
  delay(25);
}
```

Figure A.1: Screenshot of the logic for the strafing of mBot to achieve a straight-line movement

The mBot relies on readings obtained from both the ultrasonic and the IR sensor to estimate its position within the track. These readings have been amended and modified before they are used in the algorithm (see Figure A.1).

The ultrasonic sensor and IR sensor are mounted on the left and right of the mBot respectively. When the mBot is placed in the centre of the track, processed readings from the ultrasonic and IR sensor are 10.7 and 370 respectively. A processed value less than 10.7 from the ultrasonic sensor indicates that the mBot is approaching the left of the track and should steer right to get back on track. A processed value of less than 280 from the IR sensor indicates that the mBot is heading toward the right. Hence, it should manoeuvre left.

Through the simulations conducted, our group observed that 7cm (from the ultrasonic sensor) and 280 (from the IR sensor) were the minimum values required for the mBot to stay within the track. Should readings approach these limits, the mBot would need to perform hard strafing movements to introduce significant changes in the direction of movement. Otherwise, light strafing enables minor changes in the direction of the movement.

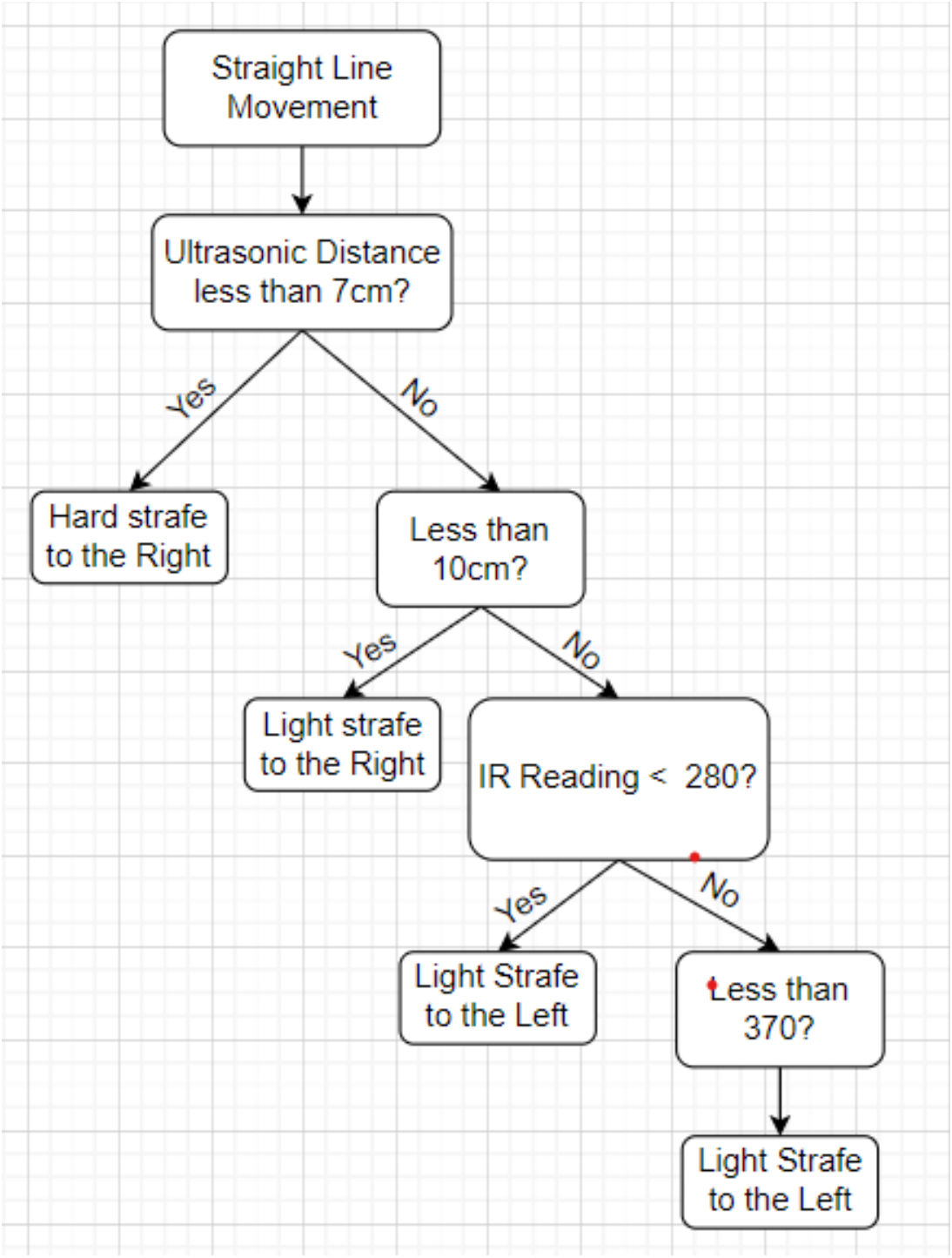


Figure A.2: Flowchart for the strafing algorithm of mBot

B. Sensors

Ultrasonic Sensor

```
const int TIMEOUT = 2000;
const float SPEED_OF_SOUND = 0.034;
const int ULTRASONIC = 12;
```

Figure B.1: Constants for ultrasonic sensor

```
pinMode(ULTRASONIC, OUTPUT);
pinMode(ULTRASONIC, INPUT);
```

Figure B.2: Setup for ultrasonic sensor

```
// Returns distance from left wall from ultrasonic sensor.
float getUltrasonicDistance() {
  // 1. Sends ultrasonic pulse.
  digitalWrite(ULTRASONIC, LOW);
  delayMicroseconds(2);
  digitalWrite(ULTRASONIC, HIGH);
  delayMicroseconds(10);
  digitalWrite(ULTRASONIC, LOW);

  // 2. Get duration of ultrasonic pulse.
  long duration = pulseIn(ULTRASONIC, HIGH, TIMEOUT);

  // 3. Calculate distance from left wall from duration.
  return duration * SPEED_OF_SOUND / 2.0;
}
```

Figure B.3: Algorithm to process readings obtained from the ultrasonic sensor.

The ultrasonic sensor functions by sending an ultrasonic pulse and measuring the time (in microseconds) required for it to receive an echo back. By assuming a constant speed of sound (340 m/s), the distance (in cm) between the ultrasonic sensor and the nearest surface can be calculated using the formula: speed of sound (in cm/microseconds) * time (in microseconds) / 2. We need to divide the time by 2 because the calculation for speed * time gives us the total distance taken by the sound wave; distance from the ultrasonic emitter to the wall and then back to the receiver.

Infrared Sensor

```
// Infrared subsystem
#define INFRARED A2
```

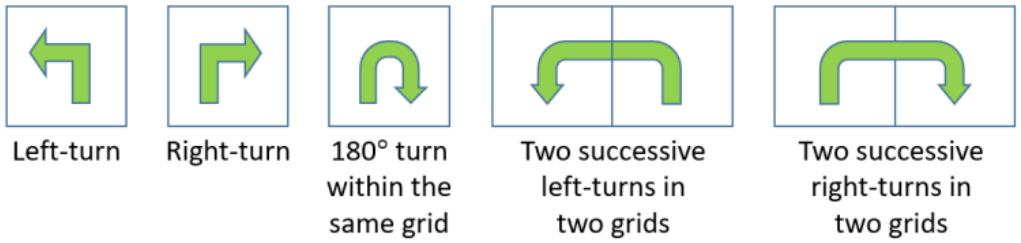
Figure B.4: INFRARED constant declaration

```
// Get distance from right wall from infrared subsystem.
int infraredReading = analogRead(INFRARED);
```

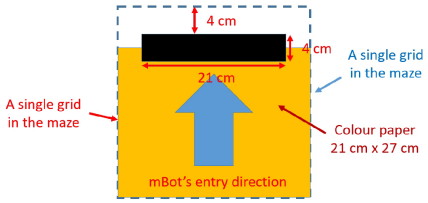
Figure B.5: Reading values from the INFRARED sensor

A reading of 370 is obtained from the IR sensor when the mBot is placed in the centre of the track. As time is required for the mBot to strafe away from the right wall (in the event it is too close to the right wall), it is necessary to maintain a minimum distance between the right wall and the mBot. Given this minimum distance, a value of 280 is registered. We were taught during studio sessions that IR readings and the distance measured are linearly related. This relationship is taken into account when deciding on the strafing logic explained in Figure A.1.

C. Waypoint Challenge



Colour	Interpretation
Red	Left-turn
Green	Right turn
Orange	180° turn within the same grid
Purple	Two successive left-turns in two grids
Light Blue	Two successive right-turns in two grids



At each waypoint challenge grid, besides the black strip, there will be a colour paper directly underneath the mBot. Depending on the colour of the paper, the mBot will execute one of the turns above. Further details on how the code and LDR sensor work will be discussed in the colour sensing subsystem below.

DETAILS ON SUBSYSTEMS

A. Black Strip Detection

In order for the mBot to successfully complete the maze, we needed a system that detects the black strip at every waypoint challenge, and this is where the 2 infrared (IR) proximity sensors of the mBot's line follower came in handy.

Both sensors are attached to the mCore port 2 with the help of RJ25. The sensors are placed facing down at the front of our mBot. Using the mBot library as well as the lineFinder we programmed out the sensor with a switch case statement. There are four cases namely S1_IN_S2_IN, S1_IN_S2_OUT, S1_OUT_S2_IN and S1_OUT_S2_OUT introduced to us. "IN" means that a reflected pulse cannot be detected, hence the ground is black at that specific sensor, S1 or S2.

Due to the reading speed, we used case S1_IN_S2_IN in our mBot to detect back strips to stop the mBot at every waypoint for the sensors to consistently stop only when it detects black for both IRs.

B. Calibration for the speed and turns of the mBot

```
// Motor and movement
const MeDCMotor leftMotor(M1);              // Assign leftMotor to port M1.
const MeDCMotor rightMotor(M2);             // Assign rightMotor to port M2.
const uint16_t MOTOR_SPEED_R = 176;         // Default speed of left motor when moving forward. Ranges from 0 to 255. 176
const uint16_t MOTOR_SPEED_L = 190;         // Default speed of right motor when moving forward. Ranges from 0 to 255. 190
const uint16_t TURN_SPEED = 170;            // Default speed when making turns. Ranges from 0 to 255.
const uint16_t TURN_TIME_90_DEG = 475;      // Duration (ms) for turning.
const uint16_t TURN_TIME_180_DEG = 850;     // Duration (ms) for 180 turn.
const uint16_t TURN_TIME_SUCC = 485;        // Duration (ms) for successive turns.
const uint16_t WAIT_TIME_AFTER_TURN = 1000; // Wait duration (ms) after a turn to stabilise mBot.
const uint16_t WAIT_TIME_AFTER_MOVE = 500;  // Wait duration (ms) after moving forward to stabilise mBot.
const uint16_t MOVE_TIME_ONE_TILE = 1000;   // Move duration (ms) for bot to move distance of one tile.
```

Figure B.1:Defining the different constants and assigning the motors to the ports in mBot's main control board

```
// Wrapper function to spin left wheel forward (anti-clockwise).
void spinLeftWheelForward(int16_t speed) {
  leftMotor.run(-speed);
}

// Wrapper function to spin left wheel backward (clockwise).
void spinLeftWheelBackward(int16_t speed) {
  leftMotor.run(speed);
}

// Wrapper function to spin right wheel forward (clockwise).
void spinRightWheelForward(int16_t speed) {
  rightMotor.run(speed);
}

// Wrapper function to spin right wheel backward (anti-clockwise).
void spinRightWheelBackward(int16_t speed) {
  rightMotor.run(-speed);
}
```

Figure B.2:Making wrapper functions to simplify the movement of the wheels

```
// Turns mBot to the right by 90 degrees (on the spot).
// □ Called when GREEN detected at waypoint.
// Parameters are optional and used for testing.
void turnRight(uint16_t turnSpeed = TURN_SPEED, uint16_t turnTime = TURN_TIME_90_DEG) {
  // Spin left motor forward and right motor backward to turn mBot to the right.
  spinLeftWheelForward(turnSpeed);
  spinRightWheelBackward(turnSpeed);
  delay(turnTime);  // Keep turning until turn is 90 degrees.

  // After turn, stop motors and wait for a short duration for mBot to stabilise.
  leftMotor.stop();
  rightMotor.stop();
  delay(WAIT_TIME_AFTER_TURN);
}

// Turns mBot to the left by 90 degrees (on the spot).
// ● Called when RED detected at waypoint.
// Parameters are optional and used for testing.
void turnLeft(uint16_t turnSpeed = TURN_SPEED, uint16_t turnTime = TURN_TIME_90_DEG) {
  // Spin left motor backward and right motor forward to turn mBot to the left.
  spinLeftWheelBackward(turnSpeed);
  spinRightWheelForward(turnSpeed);
  delay(turnTime - 10);  // Keep turning until turn is 90 degrees.

  // After turn, stop motors and wait for a short duration for mBot to stabilise.
  leftMotor.stop();
  rightMotor.stop();
  delay(WAIT_TIME_AFTER_TURN);
}
```

Figure B.3: Codes for the Left and Right turn of the mBot for colours Red and Green respectively turnTime for Left turn is reduced to ensure a consistent 90° turn. Speed of the left and right wheel was different despite inputting the same input value.

```
// Turns mBot by 180 degrees to the left (on the spot).
// □ Called when ORANGE detected at waypoint when closer to right wall.
// Parameters are optional and used for testing.
void turn180Left(uint16_t turnSpeed = TURN_SPEED, uint16_t turnTime = TURN_TIME_180_DEG) {
  // Turn mBot 180 degrees by spinning to the left.
  // Spin left motor backward and right motor forward to turn mBot to the left.
  spinLeftWheelBackward(turnSpeed);
  spinRightWheelForward(turnSpeed);
  delay(turnTime);  // Keep turning until turn is 180 degrees.

  // After turn, stop motors and wait for a short duration for mBot to stabilise.
  leftMotor.stop();
  rightMotor.stop();
  delay(WAIT_TIME_AFTER_TURN);
}
```

Figure B.4: Code for the 180° left turn for the mBot when it senses Orange colour at the waypoint challenge.

```
// Turns mBot to the left two times successively.
// □ Called when PURPLE detected at waypoint.
// Parameters are optional and used for testing.
void turnLeftThenLeft(uint16_t turnSpeed = TURN_SPEED, uint16_t turnTime = TURN_TIME_SUCC) {
  // 1. Turn mBot to the left by 90 degrees.
  turnLeft(turnSpeed, turnTime - 15);

  // 2. Then, move mBot forward by one tile.
  spinLeftWheelForward(MOTOR_SPEED_L);
  spinRightWheelForward(MOTOR_SPEED_R);
  delay(MOVE_TIME_ONE_TILE - 0);  // TODO: EDIT VALUE BASED ON LAB VALUES.

  // 3. After moving forward, stop motors and wait for a short duration for mBot to stabilise.
  leftMotor.stop();
  rightMotor.stop();
  delay(WAIT_TIME_AFTER_MOVE);

  // 4. Finally, turn mBot to the left by 90 degrees again.
  turnLeft(turnSpeed, turnTime - 35);
}

// Turns mBot to the right two times successively.
// ● Called when BLUE detected at waypoint.
// Parameters are optional and used for testing.
void turnRightThenRight(uint16_t turnSpeed = TURN_SPEED, uint16_t turnTime = TURN_TIME_SUCC) {
  // 1. Turn mBot to the right by 90 degrees.
  turnRight(turnSpeed, turnTime - 20);

  // 2. Then, move mBot forward by one tile.
  spinLeftWheelForward(MOTOR_SPEED_L);
  spinRightWheelForward(MOTOR_SPEED_R);
  delay(MOVE_TIME_ONE_TILE);  // TODO: EDIT VALUE BASED ON LAB VALUES.

  // 3. After moving forward, stop motors and wait for a short duration for mBot to stabilise.
  leftMotor.stop();
  rightMotor.stop();
  delay(WAIT_TIME_AFTER_MOVE);

  // 4. Finally, turn mBot to the right by 90 degrees again.
  turnRight(turnSpeed, turnTime - 45);
}
```

Figure B.5: Code for the consecutive left and right turns for the purple and light blue colour waypoint challenge. Similar to the left and right turns' code, we have made some slight changes within the code as the speed of the left and right wheel of our mBot was different. To ensure perfect turns, we calibrated the values according to the various circuits in our lab.

C. IR Sensor

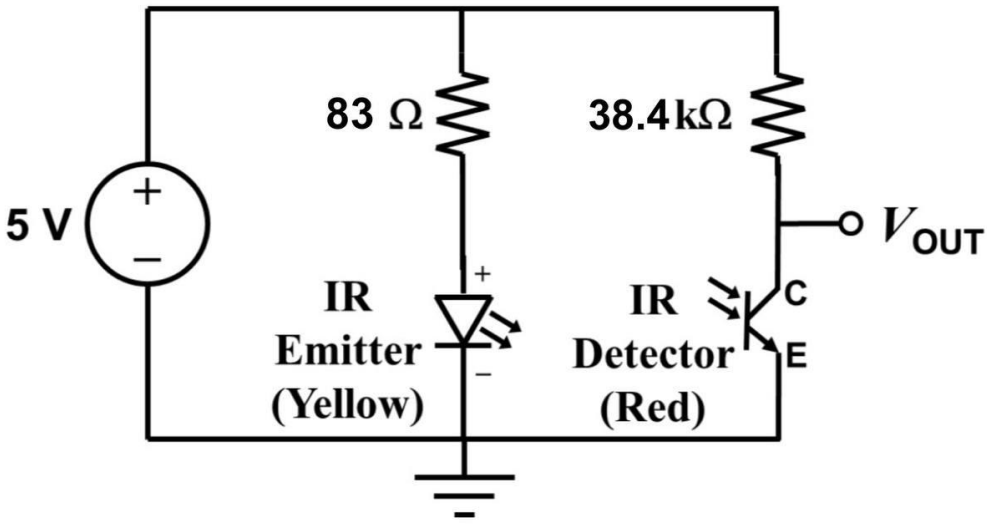
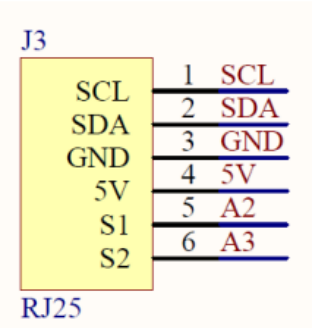


Figure C.1: Circuit diagram referenced for the construction of circuitry for the IR sensor

The IR sensor is placed on the right side of our mBot to detect the distance between the mBot and the right wall. The IR circuit is constructed with reference to the studio on the IR sensor. As the mBot was programmed initially to read IR values via the digitalRead function, adjustments have to be made to the values of the resistors. After conducting numerous experiments with different combinations of the resistor values, we finalised on the combination of 83Ω and 38.5kΩ.



Using the third port, we will have S1 reading from the IR detector and S2 reading from the LDR sensor previously decoded by the 2-to-4 chip from port 4's RJ25. At the same time, the IR emitter will get its output from the Y3.

D. Colour sensing subsystem

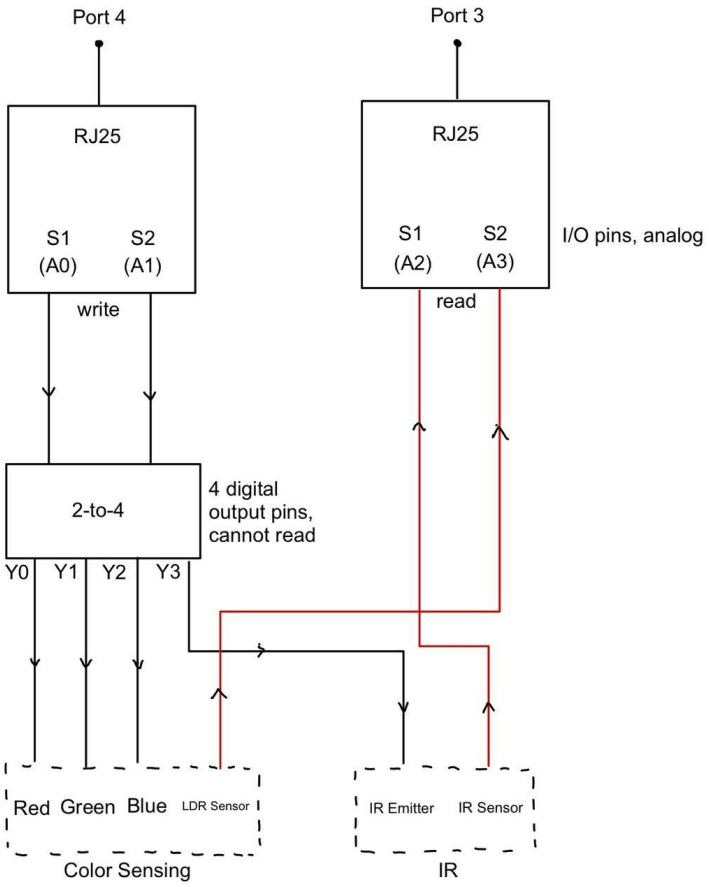


Figure D.1: Overall circuitry of sensors connected to mBot.

D.1 Overview of logic

The colour sensing subsystem is stuck to the bottom of our mBot and is used to detect the colour of the paper at each waypoint challenge. This colour sensing subsystem has three LEDs (Red, Green, and Blue) and one LDR which work in tandem to predict the colour at each waypoint. The flowchart below (Figure D.2) illustrates the steps taken to determine the colour at each waypoint.

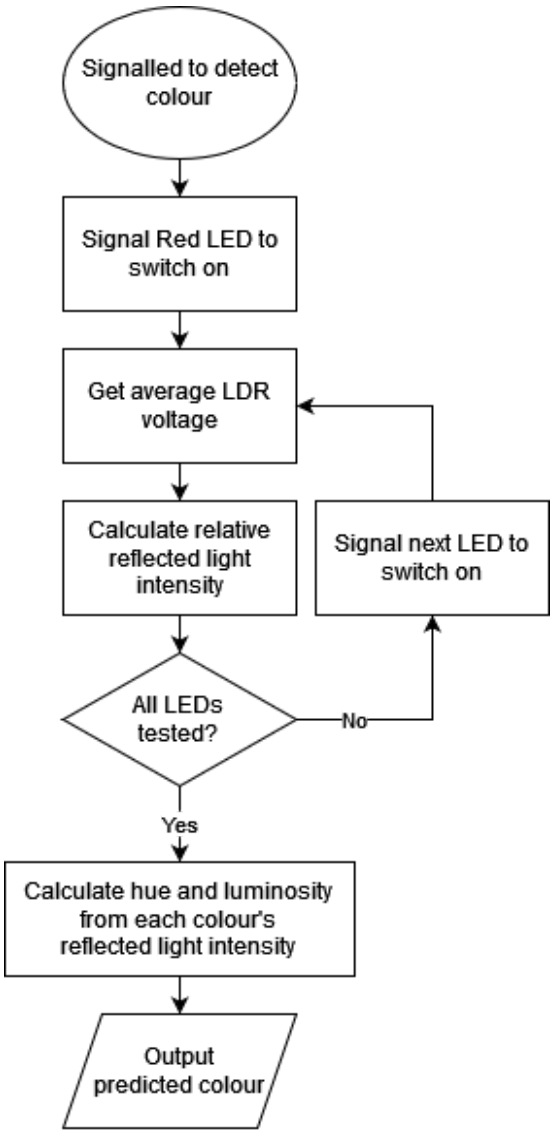


Figure D.2: Flow chart of steps needed to predict colour at waypoint challenge.

D.2 Circuit

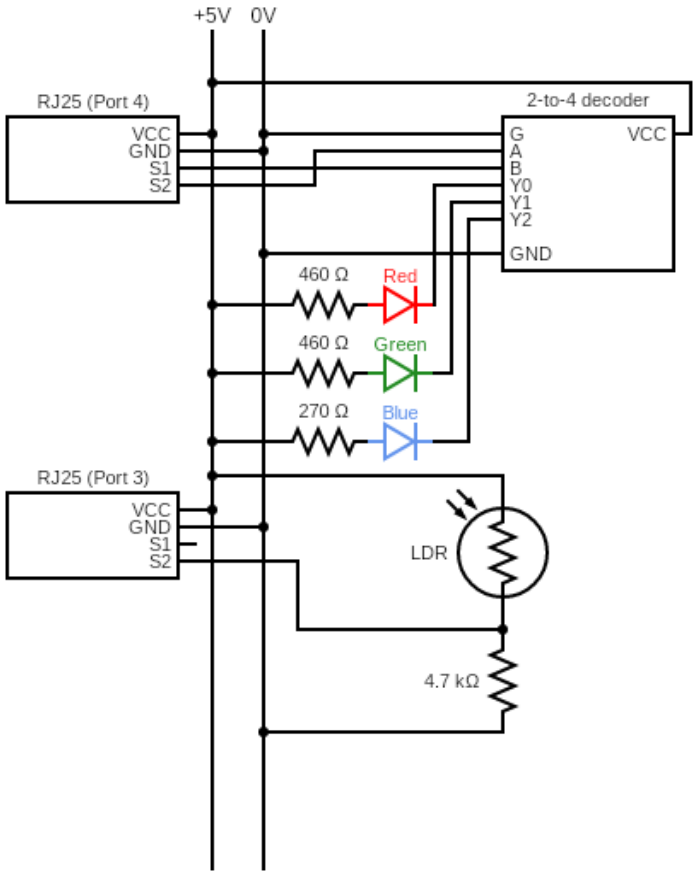


Figure D.3: Schematic of colour sensing subsystem.

We placed three LEDs parallel to each other so that they can be switched on and off independently. We placed a resistor with appropriate resistance values in series with each LED such that the current through each parallel branch is lower than 8 mA so as to not exceed the current limit of the 2-to-4 decoder IC. The resistance values are also chosen such that the current through each LED is high enough for light reflected on a surface to result in a meaningful resistance change in the LDR.

The Y0, Y1, and Y2 pins of the 2-to-4 decoder are at HIGH (5 V) by default and drop to 0 V when activated, turning on the Red, Green, or Blue LED respectively.

We use a 460 Ω resistor in series with the Red LED, a 460 Ω resistor in series with the Green LED, and a 270 Ω resistor in series with the Blue LED (see Figure D.3). This combination of resistors yields LED brightnesses effective for colour differentiation by our LDR sensor and algorithm. (The calibration and explanation will be explained in later sections.)

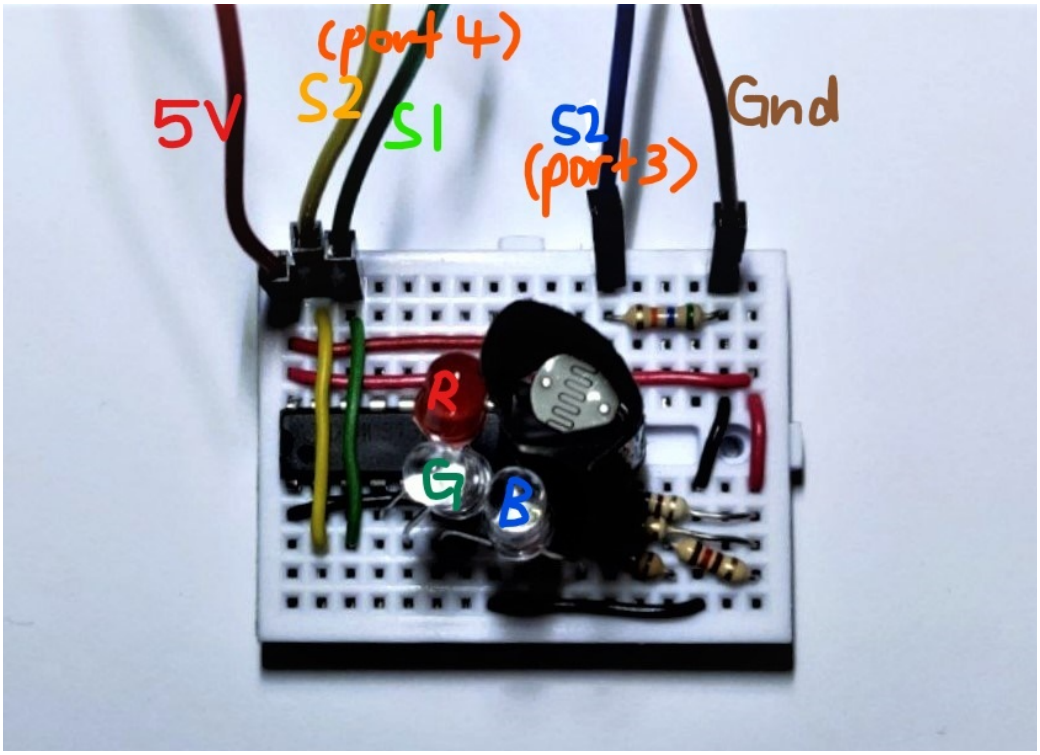


Figure D.4: Annotated breadboard of the light-sensing circuit subunit.

D.3 Signalling LEDs

To signal the Red, Blue, and Green LEDs we used the 2-to-4 decoder. It accepts two analogue inputs A and B and decodes the binary signal from these two inputs and outputs a digital signal into Y0, Y1, and Y2 (See Figure D.3). We feed signals from the two I/O ports S2 and S1 from an RJ25 adaptor into A and B respectively. When a certain combination of HIGH/LOW signals are received from the A and B pins, the Y0, Y1, and Y2 pins will be activated accordingly, activating the Red, Green, and Blue LEDs respectively. The table below shows this logic:

A	**B**	**Y0**	**Y1**	**Y2**	**LED turned on**
LOW	LOW	**LOW**	HIGH	HIGH	Red only
LOW	HIGH	HIGH	**LOW**	HIGH	Green only
HIGH	LOW	HIGH	HIGH	**LOW**	Blue only

Logic of decoding RJ25 signals into digital signals which turn on LEDs.

D.4 Determining reflectance of coloured paper

We read the voltage at the LDR circuit and use it as a proxy for the reflectance of the coloured paper at the waypoint for each LED colour. The LDR is placed in series with a high resistance (4.7 kΩ) resistor across a 5 V potential difference. One RJ25 I/O port is

used to detect the voltage across the high resistance resistor, which increases when higher intensity light is incident on the LDR and decreases when lower intensity light is incident on the LDR.

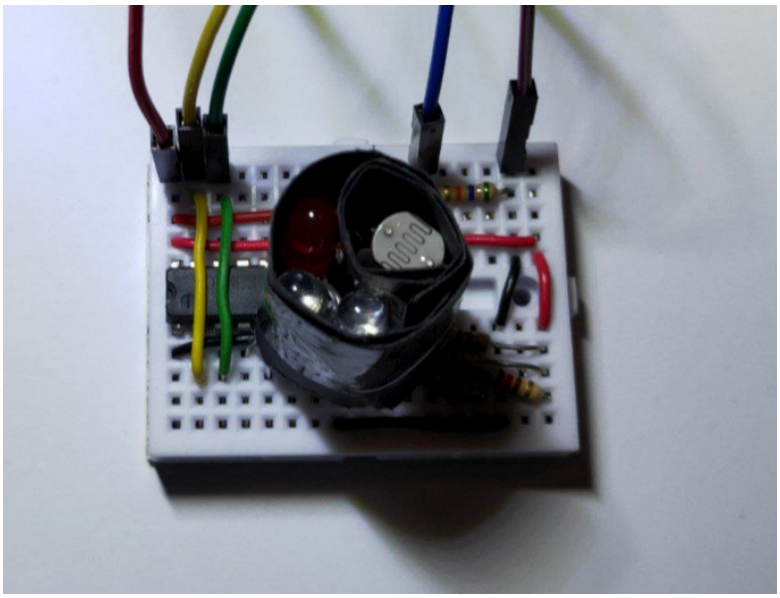


Figure D.5: The black paper shield is wrapped around the LDR and around the LDR+LEDs.

We take an average of five consecutive voltage readings to further reduce the chance of inconsistencies in readings. We also wait for a few hundred milliseconds after switching on the LED before first reading the voltage to ensure that the LDR resistance (and voltage) has stabilised.

D.5 Calculations

The steps to predict the colour at a waypoint from raw voltage readings are as such:

1. Get average voltage across the resistor when each LED is switched on.
2. Store them in variables `rawRed`, `rawGreen`, and `rawBlue`.
3. Using predetermined minimum and maximum voltage readings derived from the voltage readings when light reflects on pure white and pure black surfaces, normalise the voltage readings to a 0 to 1 scale, where 0 means the voltage reading is the minimum possible and 1 is the maximum possible.
4. Store them in variables `normRed`, `normGreen`, `normBlue`.
5. Calculate the hue using the normalised values.
6. Calculate the luminosity using the normalised values.
7. If the calculated luminosity is above a certain threshold, predict that the colour is WHITE.
8. If the calculated hue falls within a predetermined hue range corresponding to each colour, predict the colour accordingly.

This logic can also be visualised in pseudocode:

```
// Steps 1 and 2.
turnOnRedLed();
int rawRed = getAvgLdrReading();
turnOnBlueLed();
int rawBlue = getAvgLdrReading();
turnOnGreenLed();
int rawGreen = getAvgLdrReading();

// Steps 3 and 4.
float normRed = normalise(rawRed, MIN_RED, MAX_RED);
float normGreen = normalise(rawGreen, MIN_GREEN, MAX_GREEN);
float normBlue = normalise(rawBlue, MIN_BLUE, MAX_BLUE);

// Step 5.
float hue = calcHue(normRed, normGreen, normBlue);
// Step 6.
float lum = calcLum(normRed, normGreen, normBlue);

Colour colour;
// Step 7.
if (lum > MIN_LUM_WHITE)                    { colour = WHITE; }
// Step 8.
if (MIN_HUE_RED < lum < MAX_HUE_RED)        { colour = RED; }
if (MIN_HUE_ORANGE < lum < MAX_HUE_ORANGE) { colour = ORANGE; }
if (...) // If-statement logic is analogous for other colours.
```

Converting the red, green, and blue values into hue and luminosity using established algorithms allows us to easily distinguish between colours without us needing to understand the complex relationships between reflected red, green, and blue light for each colour.

CALIBRATION AND IMPROVEMENTS ON OUR CUSTOM SENSORS

Black Strip Detection

Initially we had conditions in the code, whereby if either of the IR detects a black, the mBot will automatically stop, but unfortunately we were facing issues where it did not stop when mBot reached the black strip. We believe that this is due to the reading speed of the code in cooperation with the sensors, therefore, we removed S1_IN_S2_OUT and S1_OUT_S2_IN, and implemented only one condition for it to stop, which is S1_IN_S2_IN.

IR Sensor

The IR circuit is constructed with reference to the studio on the IR sensor. As the mBot was programmed initially to read IR values via the digitalRead function, adjustments have to be made to the values of the resistors. This is because a change in value is only observed when the nearest surface is within 5 cm from the mBot. Given that we intend to mount the R sensor on top of the mBot, the distance for this change in value to be detected has to be at least 11cm.

Numerous experiments were conducted subsequently to determine the ideal combination of resistors for the IR circuit. In the end, the resistors' values of 83 ohms and 38.4k ohms shown before were chosen for our particular circuit. While there was a change to read values via the AnalogRead function, no changes were made to the physical circuit as the range of values obtained remained acceptable.

Combination of resistors (in Ω)	**Distance a change in value in detected using the digitalWrite function (in cm)**
150, 8.2k	~5
150, 15k	~ 7
150, 55.5k	~ 15
83, 38.4k	~ 11

Table illustrating the different combination of resistors and the distance. A change in value is detected via the digitalRead function

Colour Sensing Subsystem

To ensure a consistent and reliable reading, we put a shield around the LEDs and the LDR collectively to reduce excess ambient light incidents on the LDR. A shorter shield is also put around the LDR to prevent direct light from being incident on the LDR. Therefore, the majority of light incident on the LDR will be from the reflected light of LEDs on the colour paper surface. Figure D.5 shows this shielding structure fashioned from black paper. We use black paper as it absorbs the most light and reduces the effect of light scattering within the shield which may make readings inconsistent between runs.

Furthermore, we were also facing issues where the colour sensor was sensing the wrong colours, particularly orange and purple. After testing the sensor on different circuits and lighting, we came to a conclusion to adjust the values of the hue and luminosity for both colours. As a result, the sensor was able to consistently detect the correct colours.

180 Degrees Turn

During our mBot's runs on the different maze courses, we found that our mBot faced a problem of executing a left u-turn, even when the wall to the left of the mBot was too close.

Hence we decided to separate the u-turns into both a 180 degrees left and a 180 degrees right u-turn. We also changed our mechanism to take an ultrasonic distance reading from the left wall to determine if we needed to execute the right 180 degrees U-turn, instead of the default left 180 degrees U-turn.

```
// Turns mBot by 180 degrees to the left (on the spot).
// □ Called when ORANGE detected at waypoint when closer to right wall.
// Parameters are optional and used for testing.
void turn180Left(uint16_t turnSpeed = TURN_SPEED, uint16_t turnTime = TURN_TIME_180_DEG) {
  // Turn mBot 180 degrees by spinning to the left.
  // Spin left motor backward and right motor forward to turn mBot to the left.
  spinLeftWheelBackward(turnSpeed);
  spinRightWheelForward(turnSpeed);
  delay(turnTime);  // Keep turning until turn is 180 degrees.

  // After turn, stop motors and wait for a short duration for mBot to stabilise.
  leftMotor.stop();
  rightMotor.stop();
  delay(WAIT_TIME_AFTER_TURN);
}

// Turns mBot by 180 degrees to the right (on the spot).
// □ Called when ORANGE detected at waypoint when closer to left wall.
// Parameters are optional and used for testing.
void turn180Right(uint16_t turnSpeed = TURN_SPEED, uint16_t turnTime = TURN_TIME_180_DEG) {
  // Turn mBot 180 degrees by spinning to the left.
  // Spin left motor backward and right motor forward to turn mBot to the left.
  spinLeftWheelBackward(-turnSpeed);
  spinRightWheelForward(-turnSpeed);
  delay(turnTime);  // Keep turning until turn is 180 degrees.

  // After turn, stop motors and wait for a short duration for mBot to stabilise.
  leftMotor.stop();
  rightMotor.stop();
  delay(WAIT_TIME_AFTER_TURN);
}
```

Codes for the left and right u-turns

```
case C_ORANGE:
  if (getUltrasonicDistance() < 10.7) {
    turn180Right();
  } else {
    turn180Left();
  }
  break;
```

Code to check whether a left or right U-turn is needed (code is from the overall waypoint challenges code)

DIFFICULTIES FACED AND THE SOLUTION

Problem	Solution
When using the mBot at high speeds (>0.8 $speed_{max}$) we observed that the inertia of the robot caused it to oversteer and understeer. The precision of executing a turn was reduced.	Reduced the speed of the mBot to run at a slightly lower speed. On top of that, we also coded for a turning speed that was different from the straight-line running speed of the robot to ensure that our turns were precise.
Ambient light interfering with the IR sensor	Drawing inspiration from black line sensors of the mBot, we covered our IR sensor with a black box to reduce the effect of ambient light interfering with the IR sensors
Sensors not detecting the right colour due to the LDR system	By covering the LEDs and LDR with black papers, we were able to minimise the ambient light. In addition, we also adjusted the code so that it detects the correct hue and luminosity when arriving at a waypoint. And as a result, was able to detect the colour and do the correct operation
Any adjustments of protective covering around the LED resulted in a misalignment of the LED lights. This results in erroneous detection of colour	Secure the protective covering with tape and refrain from touching the protective covering surrounding the LED.
Straight line movement. Mainly, when it is too close to the right wall, the motor does not compensate enough tilt(speed) to adjust it to the left.	Experimented with different left and right motor speeds to ensure the perfect combination of both motor speeds when it detects too close to the wall
Understeered and oversteered on some occasions when it detects a colour	Together with the IR sensor, we calibrated the rotations and delays of each waypoint challenge to make sure that the robot does not overturn on every course of the maze
Having trouble with the orange waypoint, where the robot should be doing a 180 turn. Initially, we only have the robot do a	Modified behaviour such that the sensors detect which wall is further away from the sensor and turns in the direction of the

180-degree left turn, but soon realised that if the robot is too close to the left, then it will not turn properly	furthest wall.

WORK DISTRIBUTION

Cervon Wong Teng Hao	- Planned colour sensing subsystem - Constructed colour sensing subsystem - Coded logic for colour categorization algorithm - Reviewed, tidied, documented, and fixed bugs in code
Eu Zheng Xi	- Constructed the circuitry for the IR sensor - Coded logic for the strafing of the vehicle through readings obtained from the IR and Ultrasonic sensor - Calibrated movement speed and strafing magnitude based on the position of the mBot
Jayson Ng	- Documented overall report - Assembled and coded the black strip detection - Included the buzzer celebration into the code - Provided comments with calibrating and coding of the mBot subsystems
Gandhi Parth Sanjay	- Assembled the mBot - Tested various speeds for running the robot and came up with the final speeds for straight movement and turns. - Calibrated the different turns for the waypoint challenges - Put together the ultrasonic distance sensor and worked on the initial ultrasonic sensor code - Provided helpful comments for the final code and group report