CG1111A Engineering Principles and Practice I

09/11/22

Studio B01

Section 4, Team 1

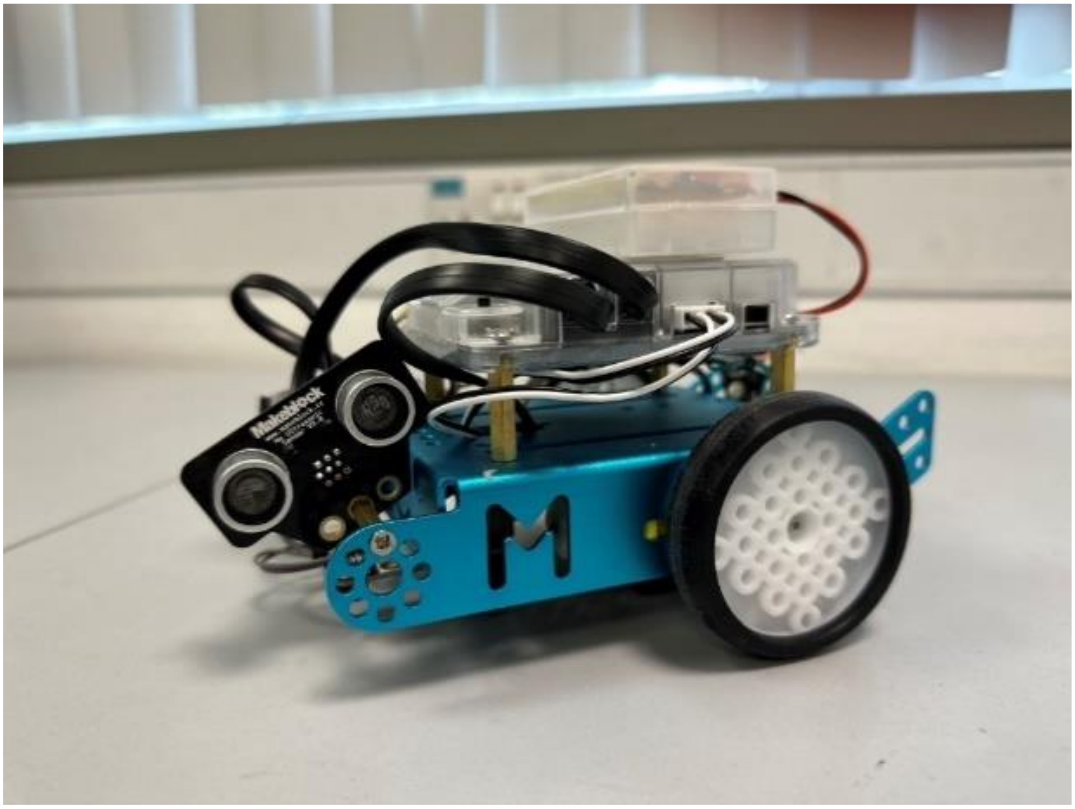
Damien Wee Joon Long (A0252597R)

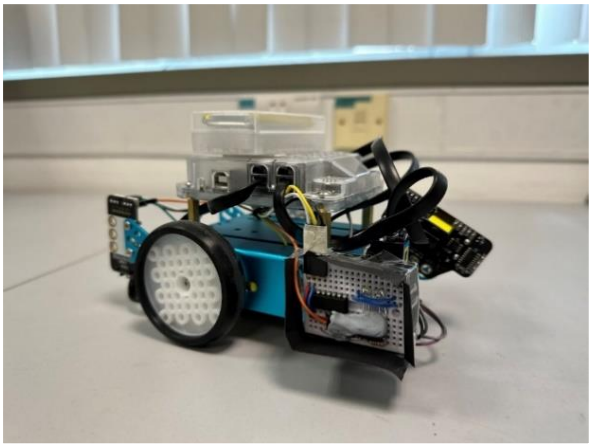
Davian Kho Yong Quan (A0252623L)

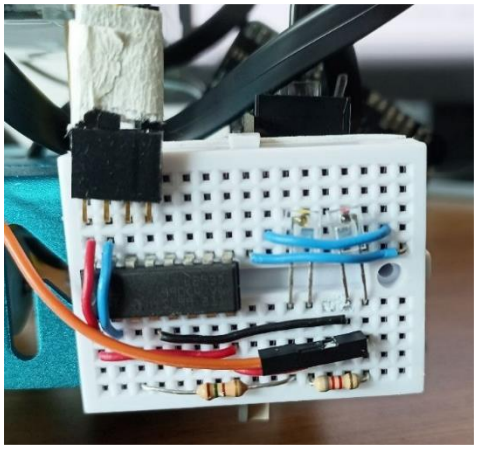
Dileep Pillai Vaibhav (A0266793L)

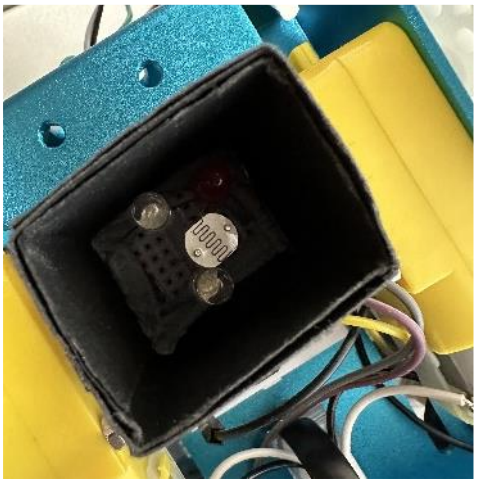
Murali Krishnan Dheekshitha (A0256963N)

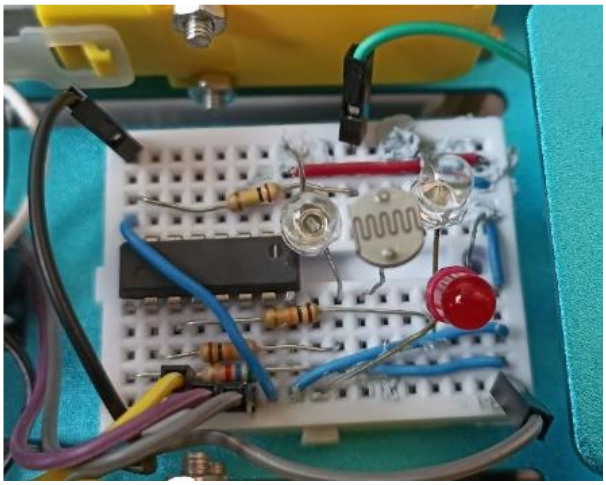
1. **Pictures of the robot and breadboard circuit**











2. Overall algorithm

When our robot starts, it will constantly be looking for a black line. If it does not detect a black line, it will continue to move forward while checking the readings from the infrared and ultrasonic sensors. If it is too near to the left wall, it will veer right and if it is too near to the right wall, it will veer left. If the robot detects a black line, it will know that it has reached a waypoint and will stop to read the colour using the colour detector made of LEDs and an LDR. Depending on the colour read, it will perform different manoeuvres as shown in the flowchart below. However, if white is the colour detected, it will stop and play some music to signal the end. Otherwise, this entire process is looped until it finally detects the colour white.

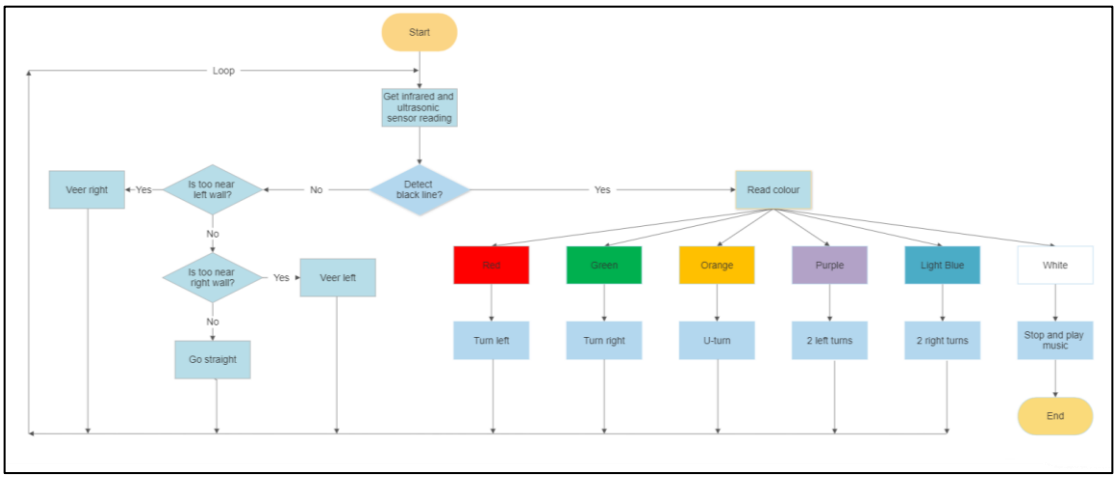


Fig. 1 Flowchart for overall algorithm

```
void loop() {
  // Get the left-side distance between the ultrasonic sensor and the
wall
  float left_dist = read_ultrasonic();

  // Get the right-side distance between the IR sensor and the wall
  long right_dist = read_IR();

  // Checks if black line is detected
  if (!is_black_line()) {
    // If black line is not detected
    delay(10); // Delay induced to prevent overcorrection by the bot
    if (left_dist > 0  && left_dist < 9.5) {
      veer_right();
    } else if (right_dist == 1) {
      veer_left();
    } else {
      goStraight();
    }
  } else {
    // If black line is detected
    Stop();
    delay(250);
    colourReading();
    makeDecision();
  }
}
```

Fig. 2 Void Loop code (overall idea of the algorithm)

3. Implementation of subsystems

a. Infrared sensor

Our infrared sensor was placed on the right side of the robot. The working principle behind our IR sensor is as such. The nearer the wall is from the IR sensor, the lower the voltage will be at the IR detector. We used analogRead() to convert this voltage into an analogue reading of range 0 – 1023. We created a function “long read_IR” that reads an analogue reading based on the voltage and either returns 1 if the analogue reading is between the range of 30 (IR MIN) to 400 (IR MAX) or returns 0 if the reading is outside of this range. If it returns 1, it means the robot is too near the wall. It returns 0 if the robot is at an acceptable distance away from the wall.

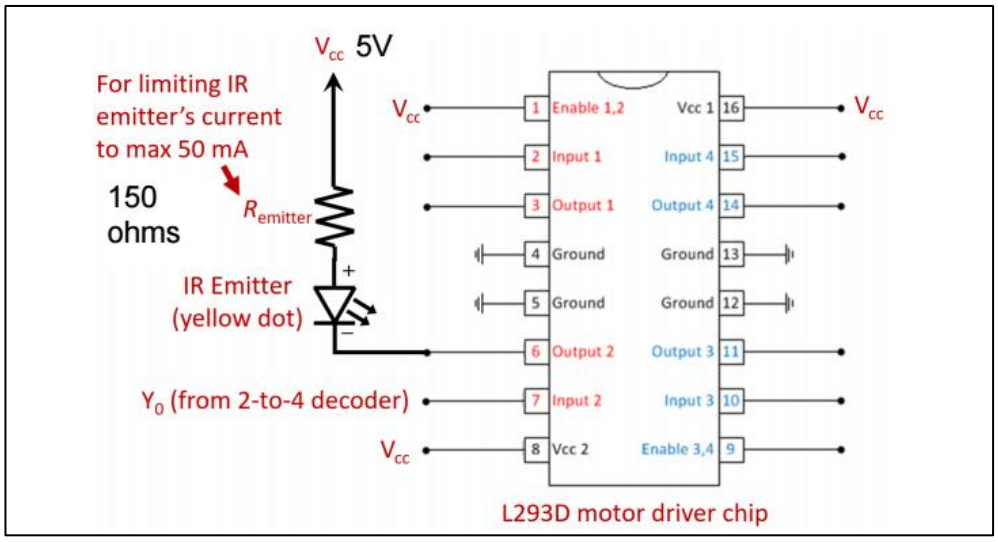


Fig. 3.1 IR Emitter circuit diagram

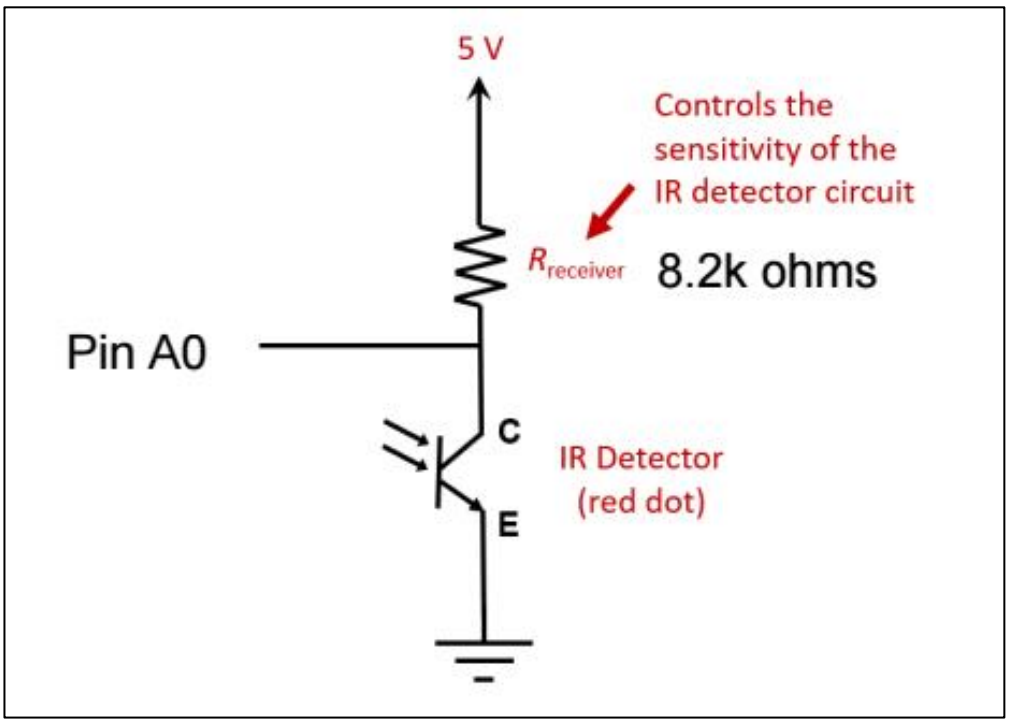


Fig. 3.2 IR detector circuit diagram

```
/**
 * Below are variables and a function required for the IR sensor to work
 * Function read_IR reads in values from the IR receiver and checks if
the the reading is within a specified range
 *
 * @return 1 if IR is within range, return 0 if IR is out of range
 */

// Port number and constants for IR
#define IR_MIN 30
#define IR_MAX 400
int IRPin = A0;

long read_IR() {
  float IR_reading = analogRead(IRPin);
  if (IR_reading > IR_MIN && IR_reading < IR_MAX) {
    return 1;
  } else {
    return 0;
  }
}
```

Fig. 4 IR detector function

b. Ultrasonic sensor

Our ultrasonic sensor was placed on the left side of the robot. The ultrasonic sensor works by transmitting a short burst of 40 kHz ultrasonic pulses towards a target and measuring how long it takes for the pulses to be echoed back to the receiver. The measured duration can then be used to calculate the actual distance by halving it. We created a function “float read_ultrasonic” that returns the actual distance the robot is away from the wall. One of the manoeuvres we needed to accomplish is U-turning. As we set our robot to turn leftwards during u-turning, we wanted our robot to hug the right wall to give it ample space to U-turn and not hit the wall. Thus, we set the robot to veer right if it is less than 9.5cm away from the left wall.

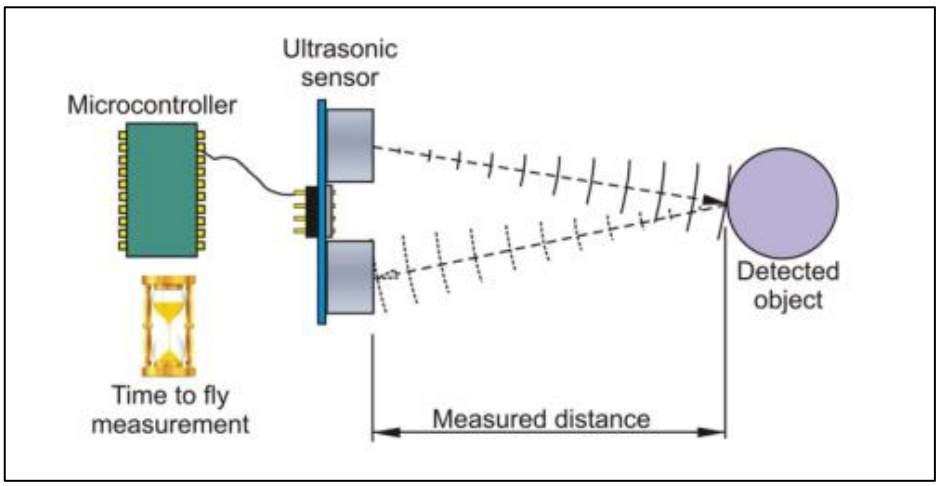


Fig. 5 How an Ultrasonic Sensor works

```
/**
 * Below are variables and function required for the Ultrasonic sensor to
work
 * The function read_ultrasonic reads in the duration of each pulse
created by the sensor, and uses the duration to calculate the distance
between
 * the ultrasonic sensor and the wall
 *
 * @returns Distance between wall and the ultrasonic sensor
 */

// Port number for Ultrasonic sensor and Constants for Ultrasonic Sensor
#define ULTRASONIC 12
#define TIMEOUT 2000 // Max microseconds to wait; choosen according to
max distance of wall
#define SPEED_OF_SOUND 340
MeUltrasonicSensor ultraSensor(PORT_1);

float read_ultrasonic() {
  pinMode(ULTRASONIC, OUTPUT);
  digitalWrite(ULTRASONIC, LOW);
  delayMicroseconds(2);
  digitalWrite(ULTRASONIC, HIGH);
  delayMicroseconds(10);
  digitalWrite(ULTRASONIC, LOW);
  pinMode(ULTRASONIC, INPUT);
  float duration = pulseIn(ULTRASONIC, HIGH, TIMEOUT);
  float distance = duration / 2.0 / 1000000 * SPEED_OF_SOUND * 100;
  return distance;
}
```

Fig. 6 Ultrasonic detector function

c. Line detector

This Makeblock line sensor uses infrared (IR) rays to detect whether there is any black line. It is mounted underneath the mBot in front of its mini caster wheel. There are two pairs of IR transmitter + receiver (Sensor 1 & Sensor 2) on this line sensor. We created a function “bool is_black_line” that returns true if a black line is detected and return false otherwise. Once our robot detects a black line, it will stop and execute the waypoint challenge.

```
//Line Sensor Ports
MeLineFollower lineFinder(PORT_2);

bool is_black_line() {
  return lineFinder.readSensors() != S1_OUT_S2_OUT;
}
```

Fig 7 Line detector function

d. Colour sensor

The colour sensor in the mBot is housed within the frame of the bot, with the Light-Emitting Diode (LED)s and Light Dependent Resistors (LDR) pointing downwards. The turning on and off the LEDs are controlled by the function “turnOnOff”, that writes in a specific order of high and low values for the A2 and A3 port, which will be translated into singular output activation and deactivation via the HD74LS139P 2-to-4 decoder IC chip. The LDR, on the other hand, was connected to the A1 port directly, requiring an analogRead() compatible port to read in LDR values. We also created a calibration function (which was removed from the final code) that takes in the white and black values from the LDR and creates a grey colour with the data collected, which is black

colour corrected for imperfect absorption of the entire colour spectrum. Thereafter, the grey colour was used to measure the remaining colours required for the challenge. Another function called colourReading() was made to sift through the possible colour values for the bot, and when the correct colour has been read, it will call its respective movement functions for subsequent action.

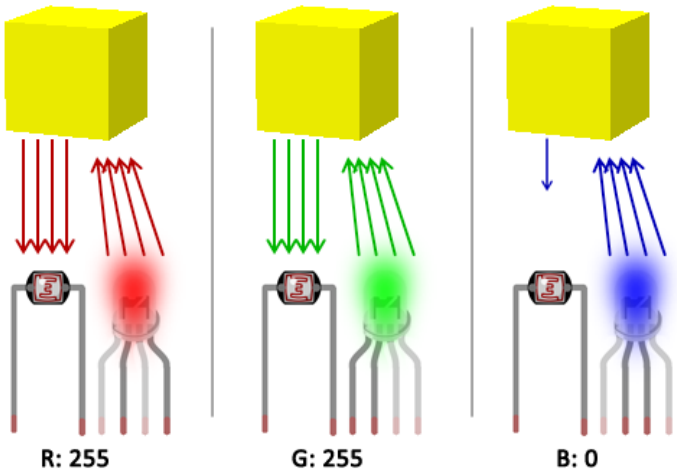


Fig. 8 How RGB is acquired through LED and LDR

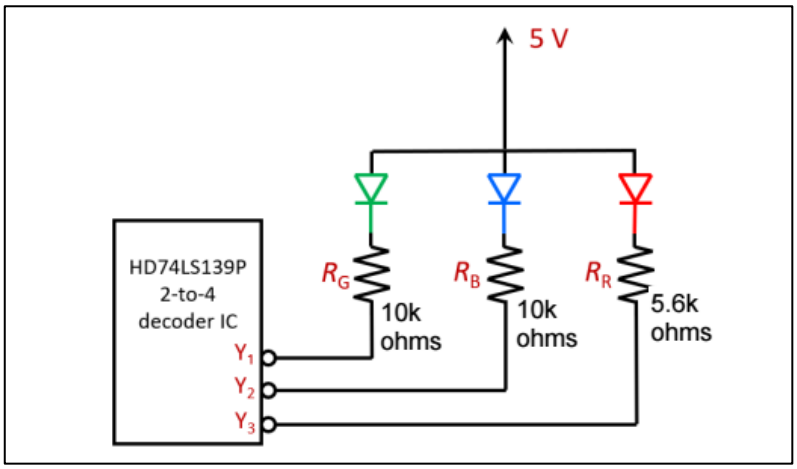


Fig. 9 Circuit diagram for LEDs

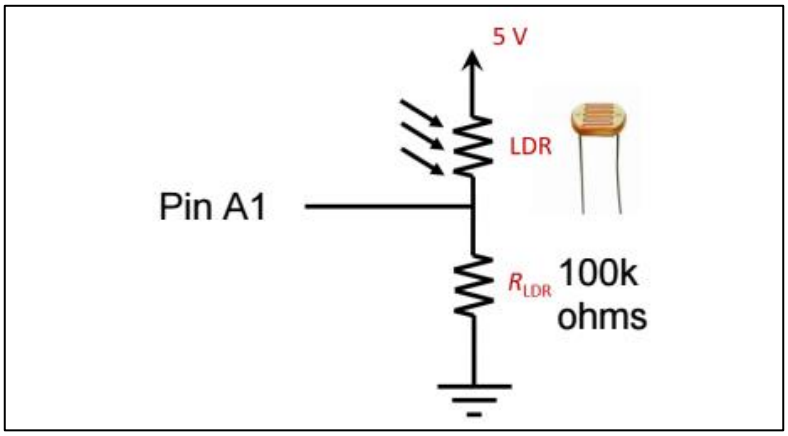


Fig. 10 circuit diagram for LDR

```
// Define time delay before the next RGB colour turns ON to allow LDR to
stabilize
#define RGBWait 200 // in milliseconds

// Define time delay before taking another LDR reading
long LDRWait = 1000; // in milliseconds

// White and black array colour data previously calibrated, and greyDiff
calculated based on the white and black values
float whiteArray[3] = {801, 877, 886};
float blackArray[3] = {314, 552, 595};
float greyDiff[3] = {487, 325, 291};

// mysteryColour array to store the RGB readings of the incoming colour
float mysteryColour[3] = {0, 0, 0};

/**
 * Function reads the colour and stores it as RGB data within the
mysteryColour array
 */

void colourReading()
{
  // Cycles through all 3 LEDs
  for (int ledColour = 0 ; ledColour < 3 ; ledColour += 1) {
    turnOnOff(ledColour, 1);
    delay(RGBWait);
    mysteryColour[ledColour] = LDR_Reading(ledColour);
    mysteryColour[ledColour] = (mysteryColour[ledColour] -
blackArray[ledColour]) / (greyDiff[ledColour]) * 255;
    turnOnOff(ledColour, 0);
    delay(RGBWait);
  }
}

/**
 * Function reads the LDR value once. It also accounts for increased time
required for LDR to stabilise when reading red
 *
 * @param[in] j The colour of the LED. 0 is red, 1 is green, 2 is blue
 *
 * @return The LDR value recorded
 */

int LDR_Reading(int j)
{
  if (j == 0) {
    LDRWait = 2000;
  }
  delay(LDRWait);
  int reading = analogRead(A1);
  LDRWait = 1000;
  return reading;
}
```

Fig. 11 Colour Reading Function

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

Fig. 12 Deciding what movement to execute based on colour

```
/**
 * Function reads the colour data and sifts through the conditionals to
check for colour, before proceeding with the subsequent actions.
 * Values within each conditional are calibrated based on the black and
grey colour previously measuered
 */

void makeDecision() {
  //White
  if ((mysteryColour[0] >= 222) && (mysteryColour[1] >= 249) &&
(mysteryColour[2] >= 248)) {
    Stop();
    play_song();
  }
  //Orange
  else if ((mysteryColour[0] >= 220) && (mysteryColour[1] >= 130) &&
(mysteryColour[2] >= 55)) {
    u_turn();
  }
  //Red
  else if ((mysteryColour[0] >= 215) && (mysteryColour[1] >= 50) &&
(mysteryColour[2] >= 45)) {
    turnLeft();
  }
  //Purple
  else if ((mysteryColour[0] >= 168) && (mysteryColour[1] >= 130) &&
(mysteryColour[2] >= 175)) {
    doubleLeft();
  }
  //Light Blue
  else if ((mysteryColour[0] >= 130) && (mysteryColour[1] >= 190) &&
(mysteryColour[2] >= 230)) {
    doubleRight();
  }
  //Green
  else if ((mysteryColour[0] >= 100) && (mysteryColour[1] >= 150) &&
(mysteryColour[2] >= 110)) {
    turnRight();
  }
}
```

Figure 13. Colour Interpretation Function

e. Movements

The following are the movements our robot can perform:

Function	Action
void Stop()	Stop moving
void goStraight()	Move straight
void veer_left()	Adjust leftwards slightly
void veer_right()	Adjust rightwards slightly
void u_turn()	Make 180 degrees turn within the same grid
void doubleLeft()	Turn left once, move straight for 1 grid, then turn left again.
void doubleRight()	Turn right once, move straight for 1 grid, then turn right again.

```
/**
 * Below are 9 seperate movement functions that is unique to each
different movement, and are called upon when required after colour
reading
 * or when preventive action is needed to prevent the bot from hitting
the wall
 */

MeDCMotor leftMotor(M1); // assigning leftMotor to port M1
MeDCMotor rightMotor(M2); // assigning RightMotor to port M2
uint8_t motorSpeed = 180; // Standardised motor speed

void Stop() {
  leftMotor.stop(); // Stop left motor
  rightMotor.stop(); // Stop right motor
}

//Going forward
void goStraight() {
  leftMotor.run(-motorSpeed-4); // Negative: wheel turns anti-clockwise
  rightMotor.run(motorSpeed); // Positive: wheel turns clockwise
}

void veer_left() {
  leftMotor.run(motorSpeed); // Positive: wheel turns clockwise
  rightMotor.run(motorSpeed); // Positive: wheel turns clockwise
  delay(5);
}

void veer_right() {
  leftMotor.run(-motorSpeed); // Positive: wheel turns anti-clockwise
  rightMotor.run(-motorSpeed); // Positive: wheel turns anti-clockwise
  delay(5);
}

// Turning left (on the spot):
void turnLeft() {
  leftMotor.run(motorSpeed); // Positive: wheel turns clockwise
  rightMotor.run(motorSpeed); // Positive: wheel turns clockwise
  delay(420); // Keep turning left for this time duration
}

// Turning right (on the spot):
void turnRight() {
  leftMotor.run(-motorSpeed); // Positive: wheel turns anti-clockwise
  rightMotor.run(-motorSpeed); // Positive: wheel turns anti-clockwise
  delay(430); // Keep turning right for this time duration
}

// Turning left (on the spot):
void u_turn() {
  leftMotor.run(motorSpeed); // Positive: wheel turns clockwise
  rightMotor.run(motorSpeed); // Positive: wheel turns clockwise
  delay(825); // Keep turning left for this time duration
}
```

```
void doubleLeft() {
  turnLeft();
  goStraight();
  delay(905);
  turnLeft();
}

void doubleRight() {
  turnRight();
  goStraight();
  delay(885);
  turnRight();
}
```

Fig. 14 Respective Movement Functions

4. Steps taken for calibrating and improving the robustness of robot

a. Colour reading

i. We sacrificed a bit of speed for higher colour reading accuracy. When calibrating the various RGB values for the different colours, we noticed that our LDR took approximately 1 to 2 seconds to stabilise and give a consistent reading. Thus, to improve its accuracy, we added a delay to let it stabilise fully before reading its value. It is also worth noting that the readings for red took an extra second to stabilise compared to blue and green, taking 2 seconds instead of 1 second.

ii. Instead of taping black paper around the robot like most groups did, we only used a chimney to block out ambient light. The chimney was made by pasting a black paper border around the colour sensor and holding it down to the breadboard with blu-tack to minimise the amount of light that can enter through the edges of between the breadboard and the LDR. We improved it further by precisely measuring the distance between the bot and the ground, and we made our chimney touch the ground as the bot moved. This prevents any light leakage from the bottom.

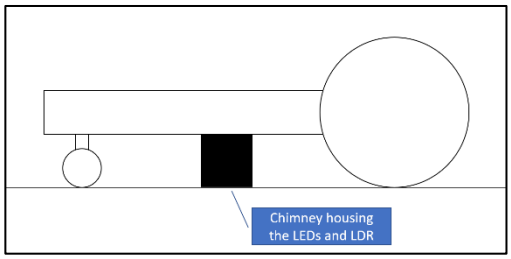


Fig. 15 chimney touching the floor

b. IR sensor

i. We added shielding around the breadboard. We found that our IR sensor behaves erratically in different parts of the lab. This can be attributed to different amounts of ambient light present at different parts of the lab. One reason could be that the window blinds were not covered fully. As such, we added shielding made of black paper around our IR sensor and breadboard to minimise any disturbances caused by ambient light.

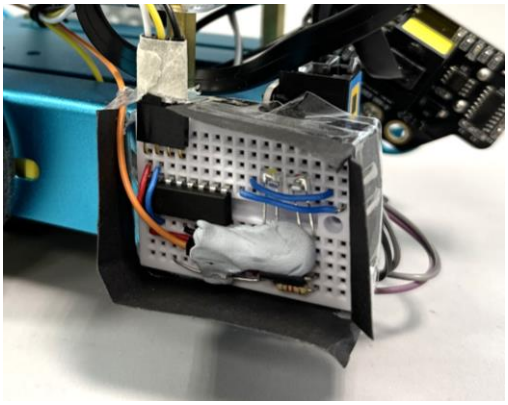


Fig. 16 IR sensor shielding

c. Movements and turns

i. Our 2 motors behaved slightly differently even when given the same code, i.e. although we set both motors to the same speed, one side will still move slightly slower than the other. As such, we fine-tuned the speeds and delay timings for the various manoeuvring actions. For example, we had to find an optimum delay timing when the mBot made two consecutive turns; the delay should not be too much such that it crashes into the wall of the maze, nor should it be too little such that the mBot does not turn completely and align parallel to the wall of the maze. We had to accurately calibrate this delay because we realised that even if the robot is not properly aligned with the wall, the small errors can compound as the mBot travels further into the maze, and eventually crash into the wall or affect the alignment of the next turn.

5. Details about work division within your team – each member's role in the project.

a. Damien

- Co-wrote the code with Davian.
- wrote the code for ultrasonic, line detector and IR sensor
- Edited and fixed a few bugs in the code for movements
- Add a decision-making function that reads the RGB values from Davian's colour-sensing code.

b. Davian

- Co-wrote the code with Damien.
- Wrote the base code for movements and manoeuvring
- Wrote the main code for the colour sensor
- Assisted with calibration of movements and colour sensor

c. Vaibhav

- The main person who fixed and took care of all the circuits, wiring and anything hardware related.
- Our colour sensor's reliability and accuracy can be attributed to his ingenious idea to block out all ambient light from reaching our LDR using blue-tack.
- Assisted with calibration of movements and colour sensor

d. Dheekshitha

- Assisted with calibration of movements and colour sensor

6. **Any significant difficulties and the steps taken to overcome them.**
   a. **Faulty components**

      When we tried to run the mBot without the IR receiver, it seemed to work fine, but it started to glitch when the IR was connected. We tested it with another spare IR detector and eventually deduced that our original IR detector was faulty. We also faced the same issue with our LDR. Initially, our colour readings kept varying under the exact same conditions and did not stabilise even when given sufficient time to do so. As we had made sure there was no light leakage, we were confident that the LDR was faulty and had to be replaced as well. We found the root cause of the problem by testing each component individually until we could narrow down and isolate a single component.

   b. **Resistor Values colour sensors**

      Initially, our colour sensor was inconsistent. After consulting with the Professor, we realised that our LED resistance values were too low, allowing the LEDs to produce colours that were too bright for the sensor to read accurately and reliably. As such, we spent a lot of time trying different resistor values for the LEDs. Eventually, we found the optimal resistance value, such that the brightness of the LED is sufficient to reflect the light off the coloured paper and read the RGB values accurately. It must also not be too low, as the output pin's logic LOW voltage will rise to keep the current under 8mA as a self-protection mechanism, which can cause the circuit behaviour to become unpredictable.