1. Introduction

The objective of the "A-maze-ing Race" project is to program our m-Bot to complete a maze, by solving "way-point challenges". As seen in Figure 1, a maze contains a starting point at a corner, straights, turns – with coloured papers underneath, and an ending point with a white paper. Each coloured paper has a black piece of tape at the end of the paper.

When solving the maze, the robot must abide by the following rules:

1. The robot must move in a near-straight line.
2. The robot must make near 90 degree, or 180 degree turns.
3. The robot needs to stop at the coloured paper, and then classify the colour and make the respective turn.
4. If the colour detected is white, the robot must stop moving and play a celebratory tune.

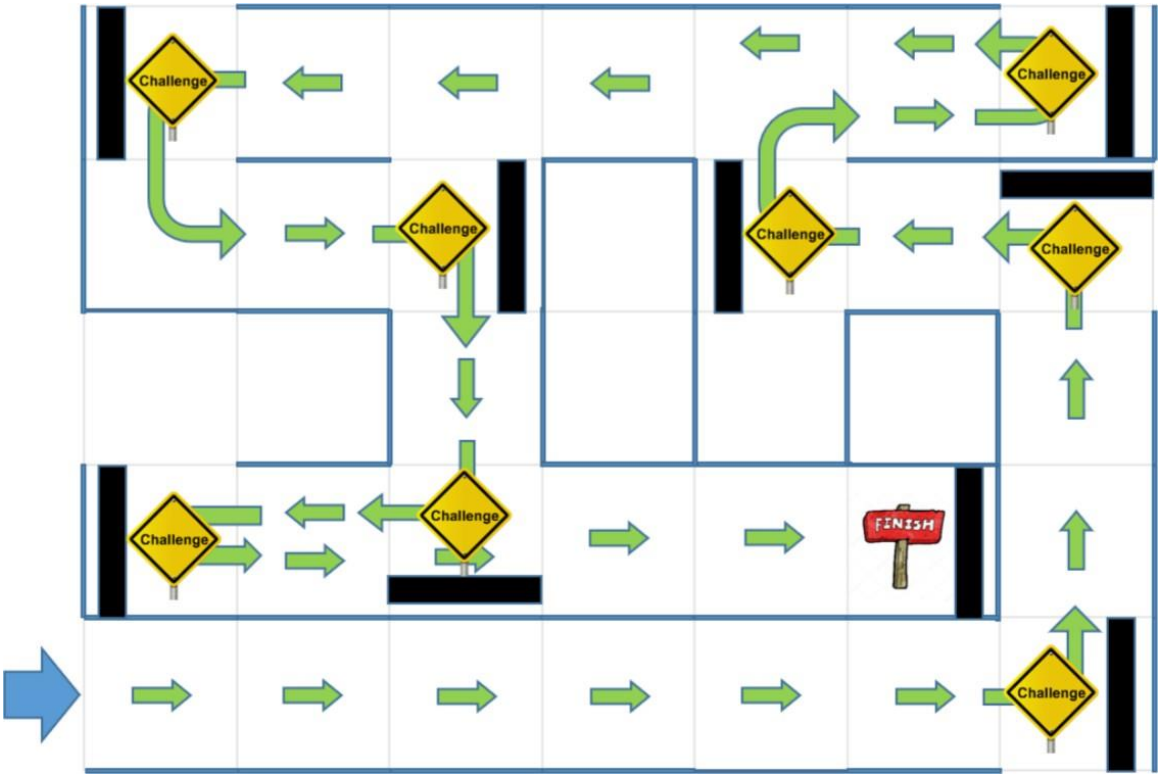

Figure 1. Sample Maze Layout.

In order to stop at the black strips, a line-sensor is given. To ensure that our robot travels in a straight line, we are provided with an ultrasonic distance sensor and a self-built infrared (IR) distance sensor on either side. Thirdly, in order to identify the coloured papers on the maze floor, we are given a Light Depended Resistor (LDR) and 3 RGB Light Emitting Diodes (LED) to build our custom colour sensor. Finally, we are given the mCore, which uses the ATmega328P Microcontroller to program our robot.

Waypoint Challenges

As mentioned above, the robot needs to decode the colour and make the necessary turn. The required turn for each colour is described in Table 1 and Figure 2.

Colour	**Movement/Action**
Red	Left turn
Green	Right turn
Orange	180-degree turn within the same grid
Purple	Two successive left-turns in two grids
Light Blue	Two successive right-turns in two grids
White	Stop moving and play tone – end of maze.

Table 1. Colour and Movement Mapping

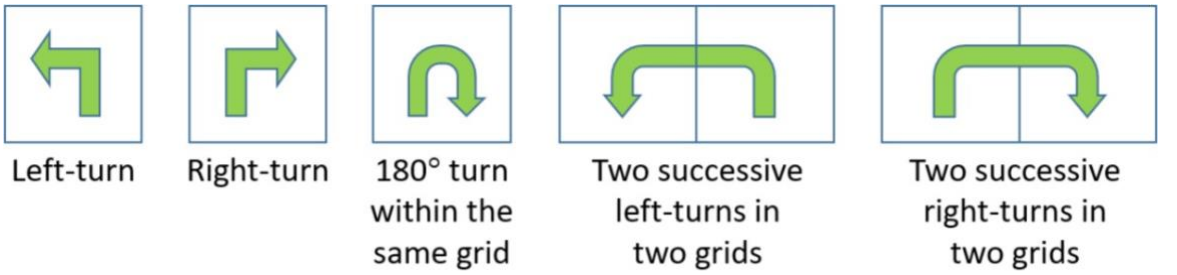

Figure 2. Movement Diagrams

2. Circuit Design & Schematics

In order to detect colour, a custom colour sensor must be built on a mini-breadboard, and to detect distance a custom IR distance sensor must be constructed on another mini-breadboard. 4 digital pins are needed to interface with 3 RGB LEDs and 1 IR emitter, however, only 2 digital pins are available. Therefore a 2-to-4 Decoder was used to trigger the LEDs/emitter individually.

2.1 2-to-4 Decoder

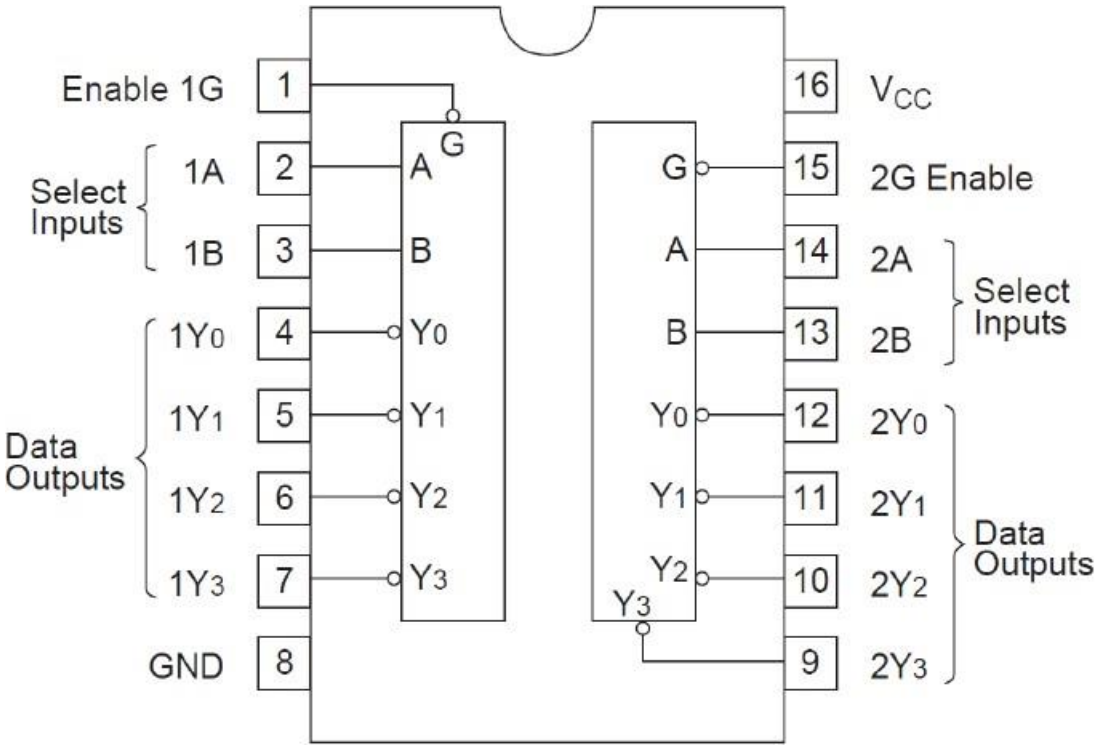

Figure 3. Circuit diagram of the 2-4 decoder chip

The 2-to-4 Decoder chip (Fig. 3) provided takes in inputs (2A & 2B) from the signal pins (attached to the S1 and S2) on the RJ25 and returns 4 outputs (Y_0-Y_3). Manipulating the two inputs can provide 4 different sets of outputs to trigger the 3 LEDs (Y_1-Y_3) and IR emitter (Y_0) separately according to Fig 4.

Inputs			Outputs			
Enable	Select					
G	B	A	Y_0	Y_1	Y_2	Y_3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

Figure 4. Inputs and their respective outputs from the 2-4 decoder

The chip is mounted on the same breadboard as the RGB LEDs. A wire connects the output (Y_0) of the decoder to the input of the L239D of the IR distance sensor. Furthermore, it can be noted that the 2-4 decoder functions by acting as the negative terminal of the circuit, meaning if output "HIGH" means that the potential of the decoder output pin and positive terminal of the LED/IR emitter would be equal, ensuring no current flows and the LED/IR emitter is turned off as the potential difference is 0 (Kirchhoff's Voltage Law). On the other hand, to turn on the LED/IR emitter, the logic of the decoder pin will be "LOW", creating a potential difference with the positive terminal of the LED/IR emitter.

2.2 IR Distance Sensor

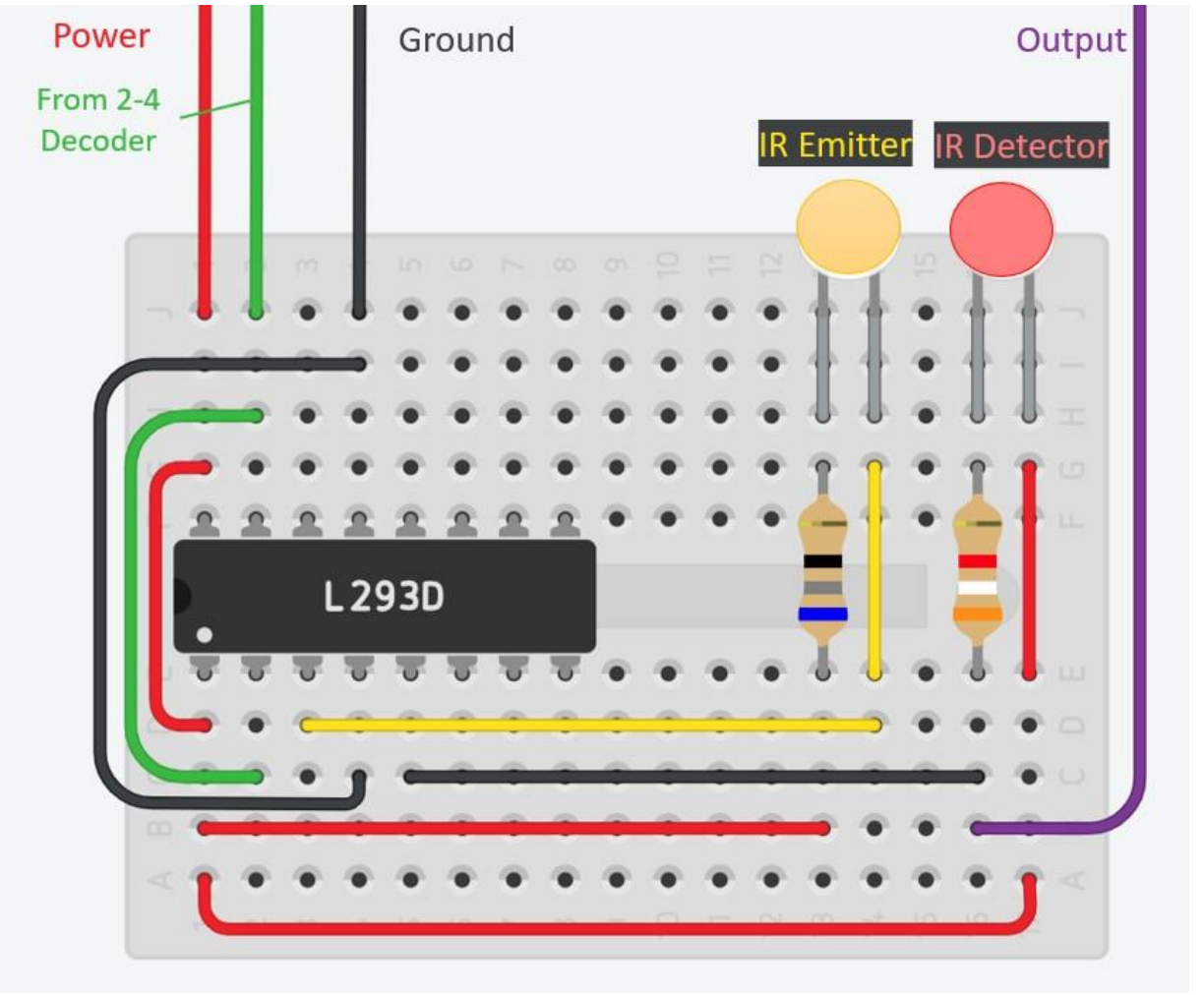

Figure 5. Circuit Diagram of the IR Distance Sensor

The IR distance sensor (Fig. 5) consists of and IR emitter and detector, which receive current from a L239D H-bridge motor driver instead of the 2-4 decoder itself. This is because the maximum current output of the decoder is insufficient to power the IR emitter and detector. The output of the IR detector is connected to an RJ25 adapter which reads the variance in voltage when the IR distance sensor detects a wall.

As can be observed in Fig 5, the IR detector is continuously receiving current and is always receiving IR. On the other hand, the emitter is continuously turned on and off by the output

(Y_0) from the 2-to-4 decoder. This is to ensure that the detector can detect the variance between IR from the emitter and ambient light. This allows the IR to make more accurate detections of walls.

Configuring the resistors for the IR distance sensor

To determine suitable resistor values for the IR emitter and detector, we initially started off with the values of used in the previous graded labs on photoelectric sensors, 150Ω and 8.2kΩ for the emitter and detector respectively. However, this did not provide a large enough variance in voltage.

First, we decided to increase the amount of current flowing in the emitter, as we were informed that the emitter having low current would reduce the distance sensed by the detector. As we were told that maximum threshold current in the emitter should be 50mA and the voltage is 1.5V. Hence, using Ohm's Law we can calculate the lowest possible resistance for the emitter to be 1.5/(50 x 10^-3) = 30Ω. However, at this resistance the variance in output voltage and the maximum voltage were low.

First, we must ensure that the maximum voltage output was at a suitable level. After systematic trial and error of the resistor for the emitter we found that voltage to be around 60Ω. The table below illustrates our process.

Resistance/ Ω	Remarks
150	Maximum voltage and variance were low
30	Maximum voltage was high, but variance was low
90	Maximum voltage was high, but variance was still low
60	Both were at suitable levels.

Table 2. Changes in resistance values and their impacts

We decided to halve resistance of the resistor for the detector to increase the sensitivity and improve the variance. However, further decreases in resistance made the variance in output voltage too low. We concluded that the detector was overly sensitive and was being affected too much by ambient light. After minor changes to both resistance values, we arrived at suitable values that provided sufficient maximum voltage and variance in voltage output:

$R_{emitter} = 68Ω$
$R_{detector} = 3900Ω$

2.3 Colour Sensor

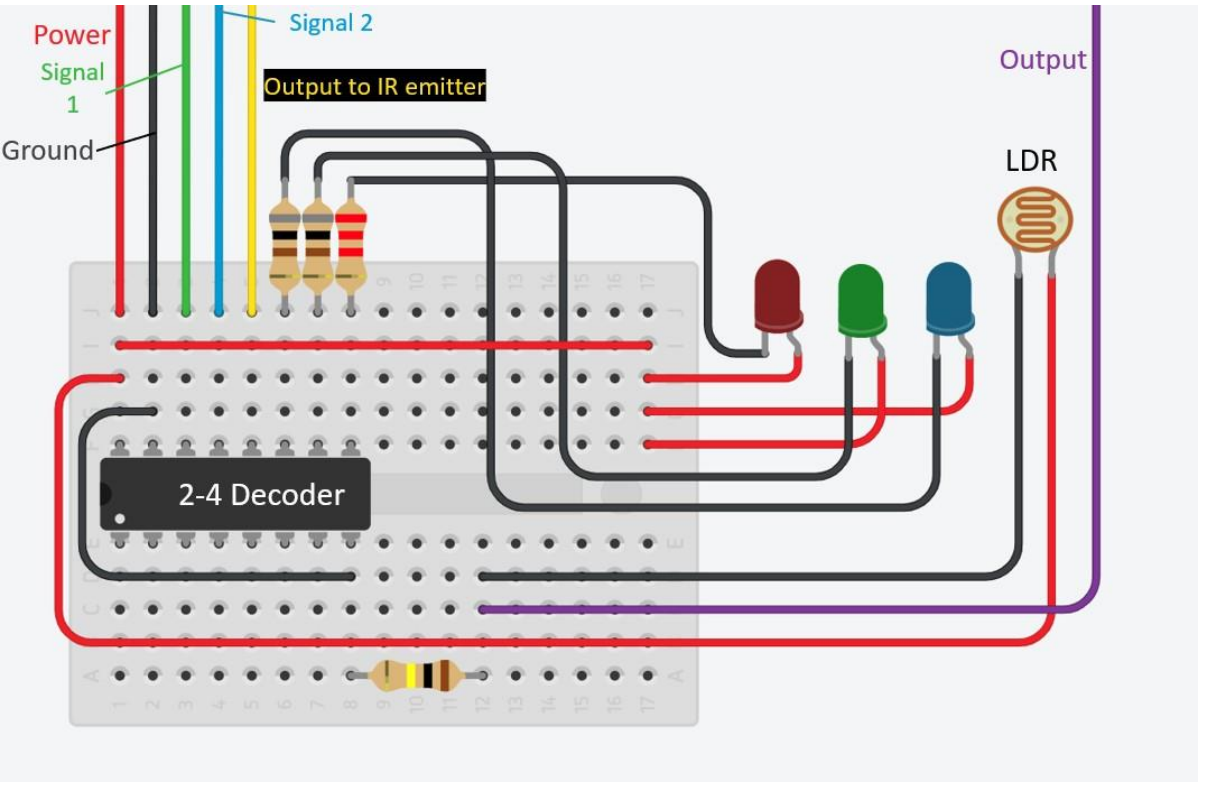

Figure 6. Circuit Diagram of the Colour Sensor

The colour sensor (Fig. 6) consists of a set of RGB LEDs and an LDR. The LEDS are triggered individually as explained above. As the LEDs turn on one by one, they shine each colour individually onto a surface resulting in the LDR's resistance varying according to the light reflected by the surface. The output of the LDR is also connected to an RJ25 adapter which reads variance in voltage for each colour, read, green and blue. This can be used to determine the RGB values of the colour of the surface and hence, the colour of the surface can be determined by matching it to the predetermined RGB values of each colour.

Configuring the resistors for the colour sensor

The resistor used for the LDR was fixed at 100kΩ as we had previously used it for the graded labs, and it had provided suitable resulting data.

For the red LED we decided to initially use a 2.7kΩ resistor (similar to the one in the lab). This means that the current flowing in the decoder through pin Y_1 = 5/2700 = 0.185 mA (Ohm's Law), which is under the 8 mA threshold for the decoder.

For the green and blue LEDs, we were not able to use the labs recommended resistance of 220Ω as current flowing through pins Y_2 and Y_3 = 5/220 = 22.7 mA, well above the 8mA threshold for the decoder. As a result, we decided to increment the resistance values to 800Ω. Hence, the current in pins Y_2 and Y_3 = 5/800 = 6.25 mA, below the threshold for the decoder.

The next issue was that the green and blue LEDs were brighter than the red one. As a result, we reduced the resistance of the red LED periodically by 100 Ω (subject to the availability of resistors in the lab) until we it was approximately as bright as the others. The value we reached was 2.2kΩ.

Hence, we arrived at the final resistance values:

$R_R = 2200\Omega$
$R_G = 800\Omega$
$R_B = 800\Omega$
$R_{LDR} = 100\ 000\Omega$

3. Design & Analysis of Algorithms

The sensors must now be able to interface with ATmega328P microcontroller's algorithms for the robot to function as intended. In the following Figure 7, the flow chart describes the overall algorithm used to navigate the maze.

3.1 Overview of Algorithm

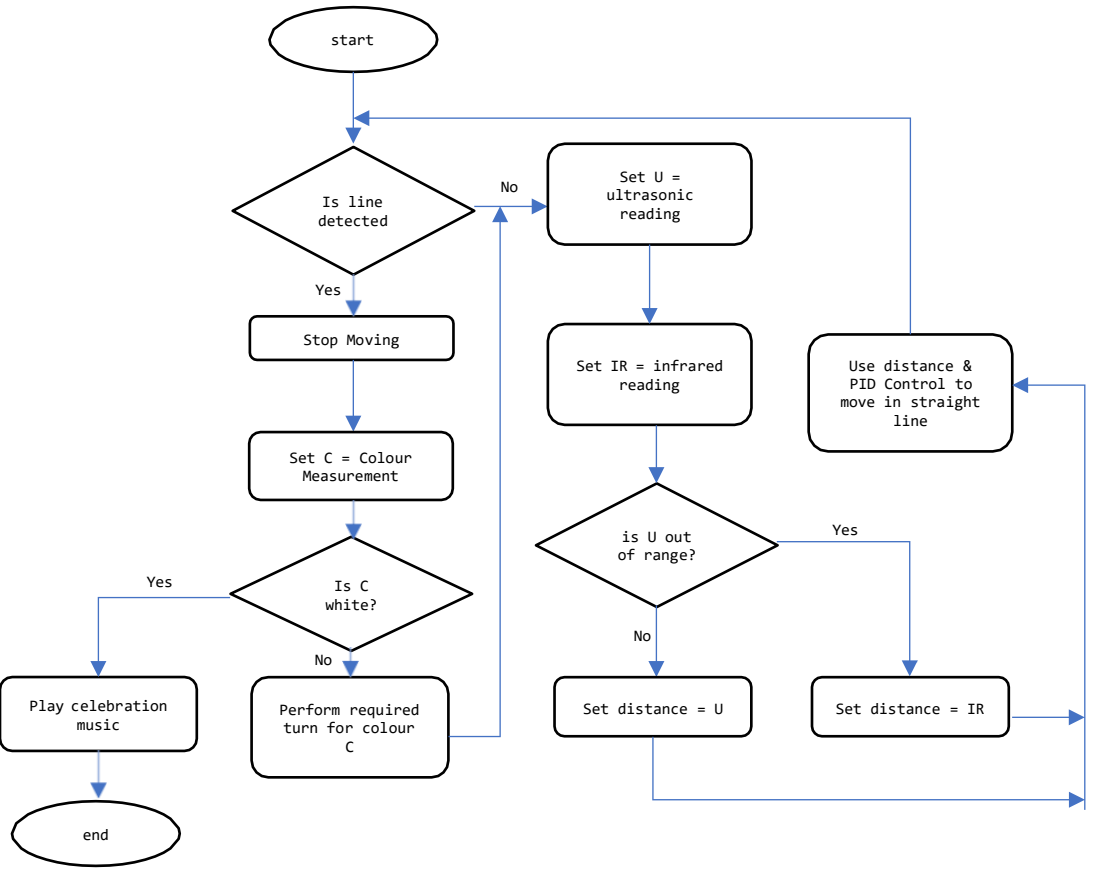

Figure 7. Algorithm Flowchart

As seen above, the line sensing dictates the primary control flow of the program. When a line is detected, the robot is at the black strip, which implies a way point challenges needs solving. Therefore, the robot halts, solves the challenges and continues with the PID movement.

If a line is not detected, then a robot is simply moving straight, therefore it uses either the ultrasonic distance measurement or the infrared (IR) distance measurement for its PID controller to continue moving in a linear manner till the next way point.

If white is detected as a waypoint, then the program plays a tune, and the robot stops moving. There are 3 physical subsystems (ultrasonic, infrared, and colour sensor) that must work in tandem with the algorithm in order to function correctly.

3.2 Line Sensor

In order to signify that there is a coloured paper underneath a robot, a a black strip has been placed at the end of the coloured paper, as shown in the figure below.

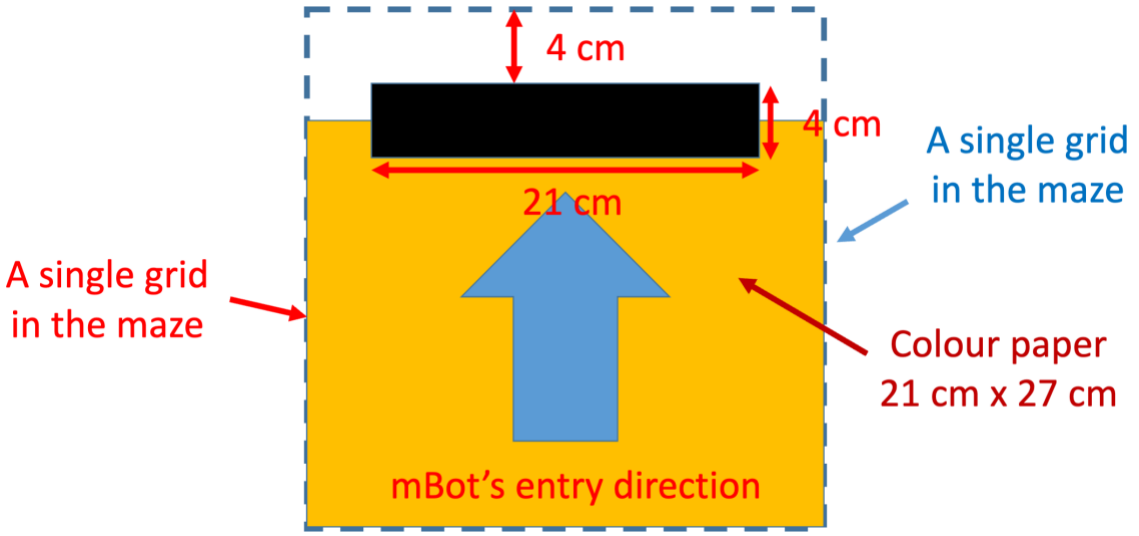

Figure 8. Positioning of black strip (taken from project documentation)

The line sensor contains 2 sets of infrared receiver and emit ers positioned closed to the ground. Black surfaces generally absorb most visible light and infrared light, therefore the line sensor will detect negligible reflection of IR when traversing over the black strip, and the line sensor will detect the IR reflection when travelling on white/coloured surfaces. However, there is small possibility that the entire line sensor is not over black strip or if one set of IR sensor does not work properly.

Line Detection

To accommodate for the above possibility, the robot comes to a stop if any one of the line sensor detects a black strip. The following code snippet demonstrates the simple control flow.

```
bool isLineDetected() {
  int sensorState = lineFinder.readSensors();
  return sensorState == S1_OUT_S2_IN || sensorState == S1_IN_S2_OUT ||\
         sensorState == S1_IN_S2_IN;
}

void loop() {
    if (isLineDetected()) {
      stopMoving();
      //Colour Detection Below
    else {
      //Continue Moving Straight
    }
}
```

3.3 Ultrasonic Distance Measurement

For the ultrasonic sensor, we are provided with a MakeBlock Ultrasonic sensor, that is able to send out ultrasonic pulses and measure the time taken to receive the echo of its pulse accurately, within a range of 3 to 400cm. The sound of speed in air can then be used to calculate the distance between the robot and the adjacent wall.

The minimum distance for the ultrasonic sensor to be able to detect its echo is about 3 cm, therefore the sensor was mounted 3 cm inwards such that the ultrasonic is always able to convey a meaningful reading *(see Appendix)*. However, sometimes the wall adjacent to the ultrasonic sensor maybe removed, resulting in a very large reading, in that case the IR sensor will be used instead. (*see Section 3.4)*.

Calculating Distance

The ultrasonic sensor outputs in *microseconds* the time needed to receive the pulse. Let speed of sound be the constant *s* in m/s. Let r be the distance in *cm*.

$$2 * r = (s * 100)(t * 10^{-6}) \text{ (Eq. 1A)}$$

$$r = \frac{s*t*\&0^{-4}}{(} \text{ (Eq. 1B)}$$

Equation 1A and 1B. Calculating distance (r) from duration (t). r is in centimetres; t is in microseconds and s is defined in metres/second.

To accurately measure the distance *r*, the speed of sound must be measured experimentally.

Experimentally Determining Speed of Sound

From the equation above, the distance and time are linearly related. By plotting time against distance, we can use the gradient to calculate the speed of sound in air.

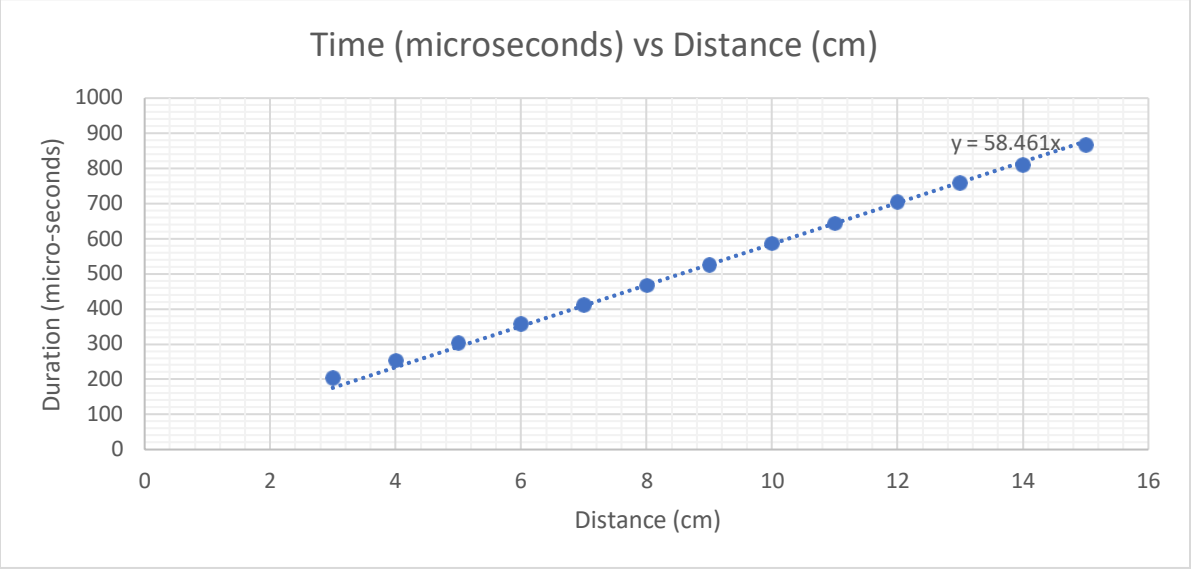

Figure 9. Time to receive echo (microseconds) vs Distance between Ultrasonic Sensor and Maze Wall.

From the above Figure 5, and by rearranging Equation 1B, we obtain the following expressions for the gradient.

$$t = \frac{(}{s* \&0^{!4}} \cdot r \text{ (Equation 2A)}$$

$$s = \frac{(}{m \cdot \&0^{!4}} \text{ (Equation 2B)}$$

Equations 2A and 2B. 2A describes the equation of the plot in Figure 5. And 2B describes how to use the gradient m found in the chart to obtain the speed of sound.

By using the gradient *58.461* as *m*, we can substitute into *Equation 2B* to obtain the constant speed of sound as **342 m/s.**

Minimising Latency of Ultrasonic Reading

The ultrasonic sensor will emit a pulse and measure the time for the pulse's echo to return to the sensor. The maximum time that the sensor waits is known as the **`time_out`** constant.

If there is no adjacent wall, then the ultrasonic sensor will wait a long time before returning *0*, as an indicator for out of range. However, in order for the PID controller to work smoothly (*Section 3.6*), fast feedback from the ultrasonic sensor is necessary. Therefore, the latency of the ultrasonic sensor must be minimised as much as possible to ensure that the robot is highly responsive to changes in distance – this will ensure that the corrections to the trajectory will be made quickly before the robot goes off-course too much.

From the grid diagram in Figure 4, each grid is slightly wider than 27 cm. Although only less than half of 27 cm is ideal distance for the ultra-sensor from the wall for the robot to be aligned to the centre, we can maximise our error margin to 27 cm, though not all of that is needed. Using speed of sound as **342** m/s, we can use *Equation 2A* to calculate the **`time_out`** constant to be 1578, rounded to **1600** (microseconds).

```
//Ultrasonic
#define TIMEOUT 1600
#define SPEED_OF_SOUND 342
#define ULTRASONIC 12

double getDistance() {
  //Emitting Sound Waves
  pinMode(ULTRASONIC, OUTPUT);
  digitalWrite(ULTRASONIC, LOW);
  delayMicroseconds(2);
  digitalWrite(ULTRASONIC, HIGH);
  delayMicroseconds(10);
  digitalWrite(ULTRASONIC, LOW);
  pinMode(ULTRASONIC, INPUT);

  //Duration to Receive Echo
  long duration = pulseIn(ULTRASONIC, HIGH, TIMEOUT);
  //Distance Calculation
  return duration * SPEED_OF_SOUND * 0.0001 / 2;
}
```

3.4 Infrared (IR) Distance Measurement

On the other side of the robot, a custom-built IR sensor is to be mounted. The circuit analysis of the IR distance sensor can be found in *Section 2.2*.

The IR sensor works by turning on the emitter for small periods of time, the IR pulse reflects off the white maze walls (absorbs minimal IR radiation and reflects most of it) and is measured on the receiver. This is repeated several thousand times in a second when the IR sensor is actively used to correct the course of the robot.

Relationship Between Distance and Voltage Measured

As seen in the circuit analysis in *Section 2.3*, when the IR receiver receives a signal, there is a drop in voltage across it. Therefore, the stronger the signal, the greater the dip in voltage. Hence, as seen in Figure 10, for closer distances to the wall, the analog voltage reading is low and it increases with distance and the voltage reading asymptotes towards the ambient/baseline IR reading.

Furthermore, the derivative of the voltage reading, that is the responsiveness to changes in distance is greater when the distance is smaller – which is expected since the reflected signal is stronger when the distance is smaller – and lower when the distance is further.

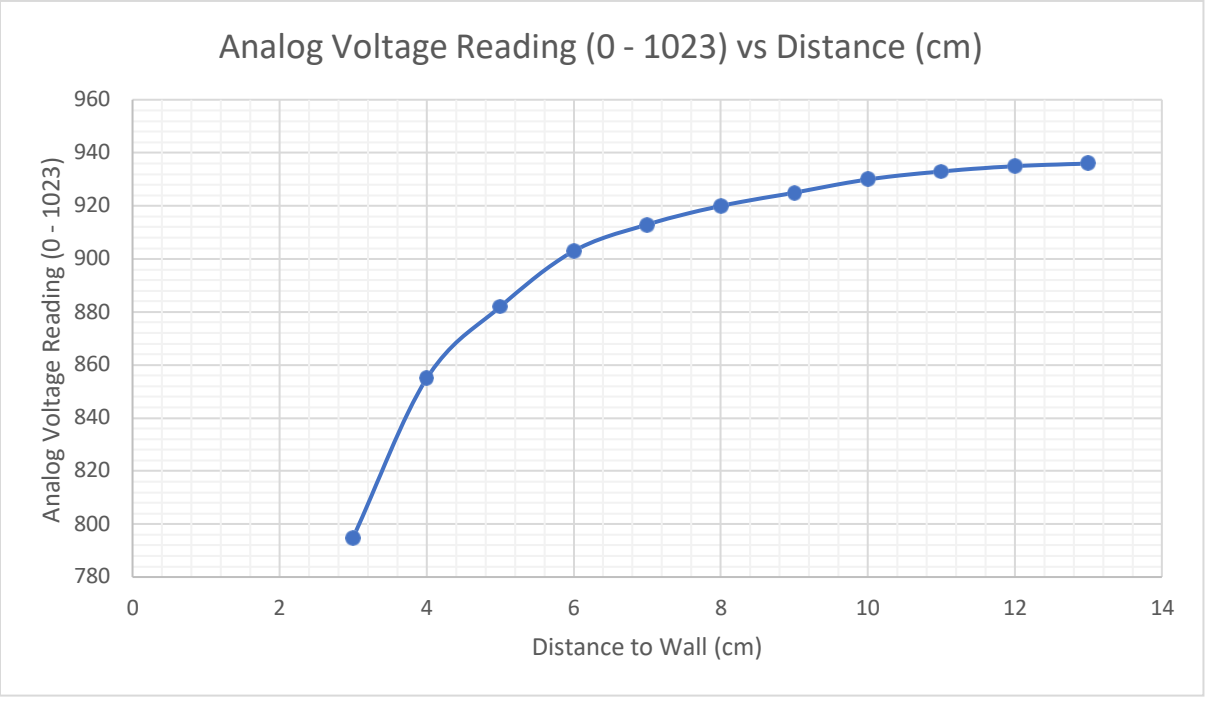

Figure 10. Analog Voltage Reading vs Distance for IR Receiver

IR Measurement Adjusted for Ambient Lighting

To account for changes in ambient lighting, the IR receiver's voltage can be subtracted from the baseline voltage, which allows the robot to accurately measure the *dip* in the voltage solely due to the IR reflection from the wall.

When the sensor is closer to the wall, the reflected signal is stronger, resulting in a larger *dip* in voltage when closer to the wall. As the sensor moves further, the dip becomes smaller. This relationship is demonstrated when plotting *dip in voltage vs distance* in the following figure.

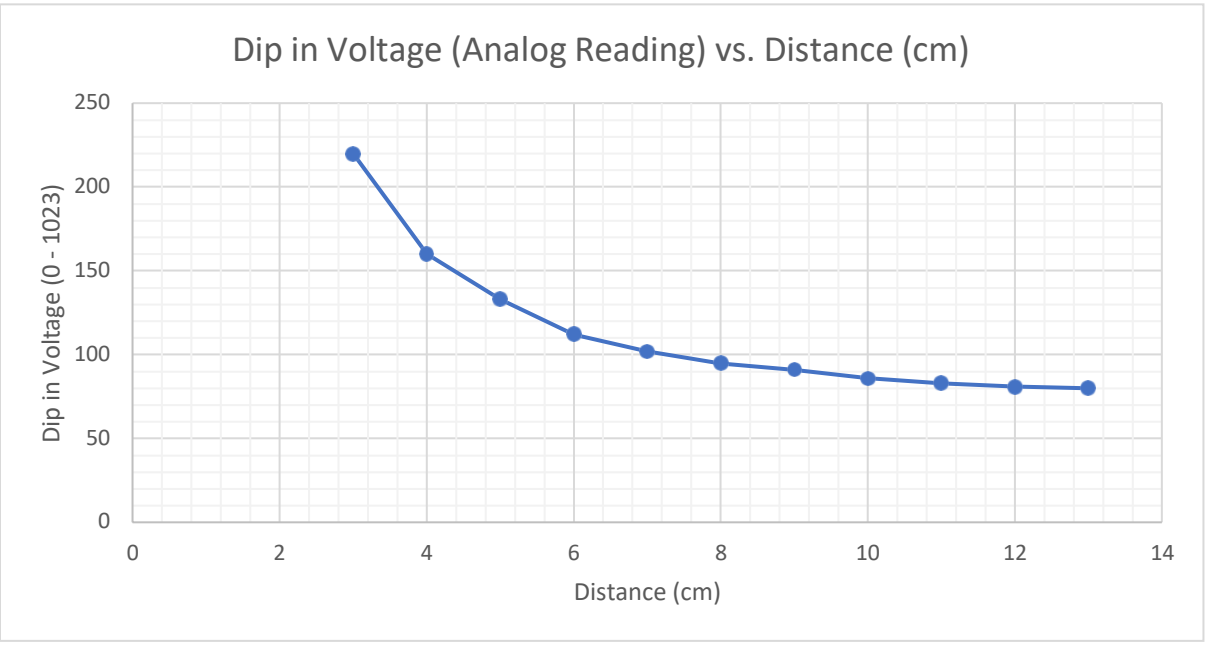

Figure 11. Dip in Voltage (Analog Reading 0~1023) vs Distance (cm) from sensor.

By using the relationship between the *dip in voltage* and distance, the effects of ambient IR radiation on the measurement can be eliminated. The dip in voltage is now solely dependent on the reflectivity of the maze walls, which can be assumed to be approximately constant.

Model for IR Sensor

The IR sensor has several useful properties that can be exploited to create a reliable distance – voltage model.

1. The infrared from the emitter radiates in all directions, therefore the intensity of light incident on the maze wall and reflected back is inversely proportional to the square of the distance between the emitter and the wall.
2. The intensity of light entering the receiver is directly proportional to the voltage measured across the receiver.

Using the above properties, the distance measured by the IR can be approximated by the following model (Equation 3C).

$$V \text{ a } I \text{ (Equation 3A)}$$

$$I = \frac{1}{r^2} + c \text{ (Equation 3B)}$$

$$\therefore V = \frac{k}{r^2} + c \text{ (Equation 3C)}$$

Equation 3A, 3B, 3C. (I) represents intensity of IR incident on receiver. The voltage (V) measured by the receiver can be said to be inversely proportional to the square of the distance (r). ***k and c are constants to be determined.***

In equation 3C, *k* is a constant of proportionality that relates the voltage reading to the distance from the wall. The constant *c* is the minimal *dip* when the IR emitter is turned on. Ideally, the minimal dip must be 0, so that when the wall is too far away to be detected, there is no dip in the voltage reading in the receiver. However, due to the radial spread of the IR waves from the emitter, the the receiver is not properly shielded from the emitter and receives some baseline amount of IR radiation from the emitter. Since this effect is constant, it can be represented with the constant *c*.

Inverse-Squared Law

Equation 3C uses the inverse-squared law to model the relationship between voltage reading and distance. In order to verify the model and ascertain the unknown constants *k* and *c*, the voltage reading (V) can be plot against $\frac{\&}{r^2}$ which will linearize the relationship, allowing us to obtain *k* and *c* using the gradient and y-intercept.

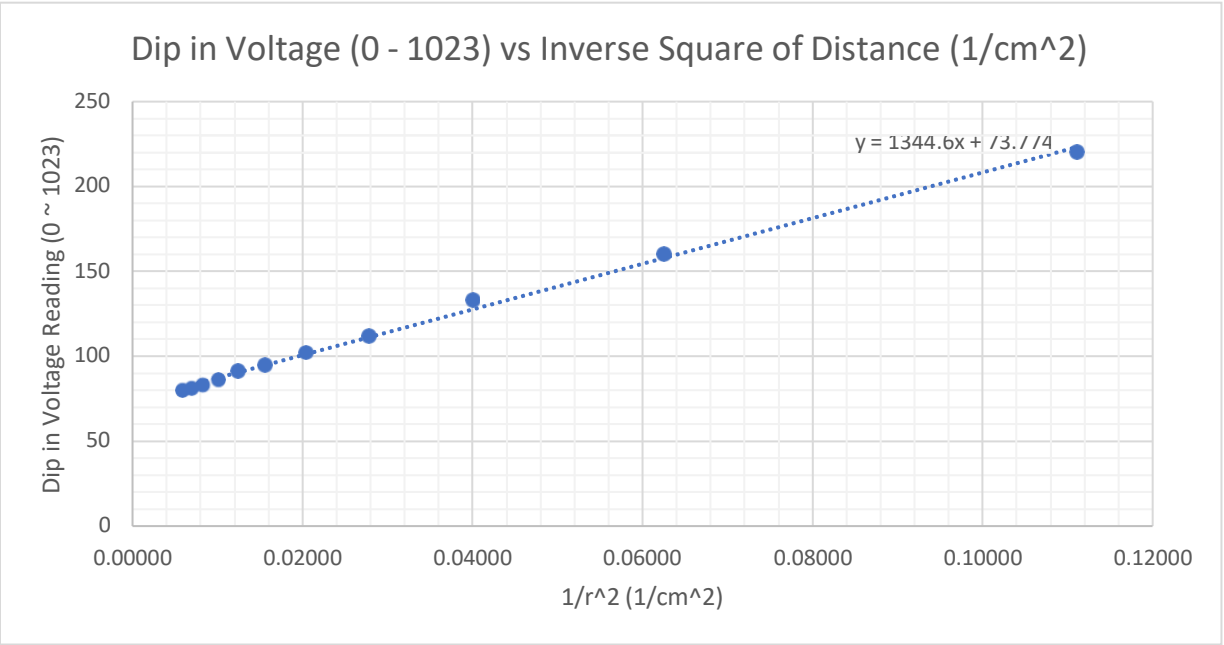

Figure 12. Dip in Voltage (0 – 1023) vs Inverse Squared Distance (1/cm^2)

As seen from the above figure, there is a strong linear relationship between the two variables, suggesting that inverse squared law can model the IR sensor well. From the graph above, *k* and *c* were empirically determined to be **1344.6** and **73.774** respectively. These constants are solely dependent on the reflectivity of the maze walls, and are not dependent on the ambient IR radiation, therefore, these constants need to be determined only once in lab setting.

By rearranging Equation 3C with the new *k* and *c* constants, we can obtain the following equation to determine distance from the analog voltage dip (*baseline reading minus sensor reading*).

$$r = *\frac{\overline{k}}{V\%c} \quad \text{(Equation 4A)}$$

$$r = *\frac{\overline{13((.*}}{V\%+3.++(} \quad \text{(Equation 4B)}$$

Equation 4A and 4B. Equation 4B is the empirically determined formula to calculate distance in cm (r) from analog voltage dip (V).

The above equation and constants were implemented as follows:

```
#define IR_STABILIZE 100
#define IR_RECEIVER A2

// Returns IR Reading in centimetres
float getIRDistance() {
  // Sensor Read
  turnOnIR();
  delayMicroseconds(IR_STABILIZE);
  int sensorReading = analogRead(IR_RECEIVER);

  // Base Line Read
  turnOffIR();
  delayMicroseconds(IR_STABILIZE);
  int baseline = analogRead(IR_RECEIVER);

  return convertToDistance(baseline - sensorReading);
}

// Applies an equation to convert dip in voltage into distance
float convertToDistance(int val) {
  const float k = 1344.6;
  const float c = 73.774;

  return sqrt((k)/(val - c));
}
```

3.5 Colour Classification Algorithm

The colour sensor consists of 1 Red, 1 Green and 1 Blue LED, triggered with the 2-to-4 decoder one at a time. There is a light dependent resistor (LDR) whose physical resistance decreases when light is incident on its surface. This property is exploited in the LDR to measure its response to RGB light, which are the 3 primary wavelengths needed to identify to any colour. The circuit schematics can be found in *Section 2.3*.

LDR Response Time

Before an effective algorithm can be developed, the LDR's physical properties must be analysed. One crucial property is the amount of time it takes for the changes in the LDR's resistance to stabilise in response to a new wavelength, this time is referred to as the LDR *response time*.

The LDR response time was measured by placing the colour sensor over *white* paper, and turning on the RGB LEDs one at a time, and recording the voltage across the LDR over a period of time for each colour.

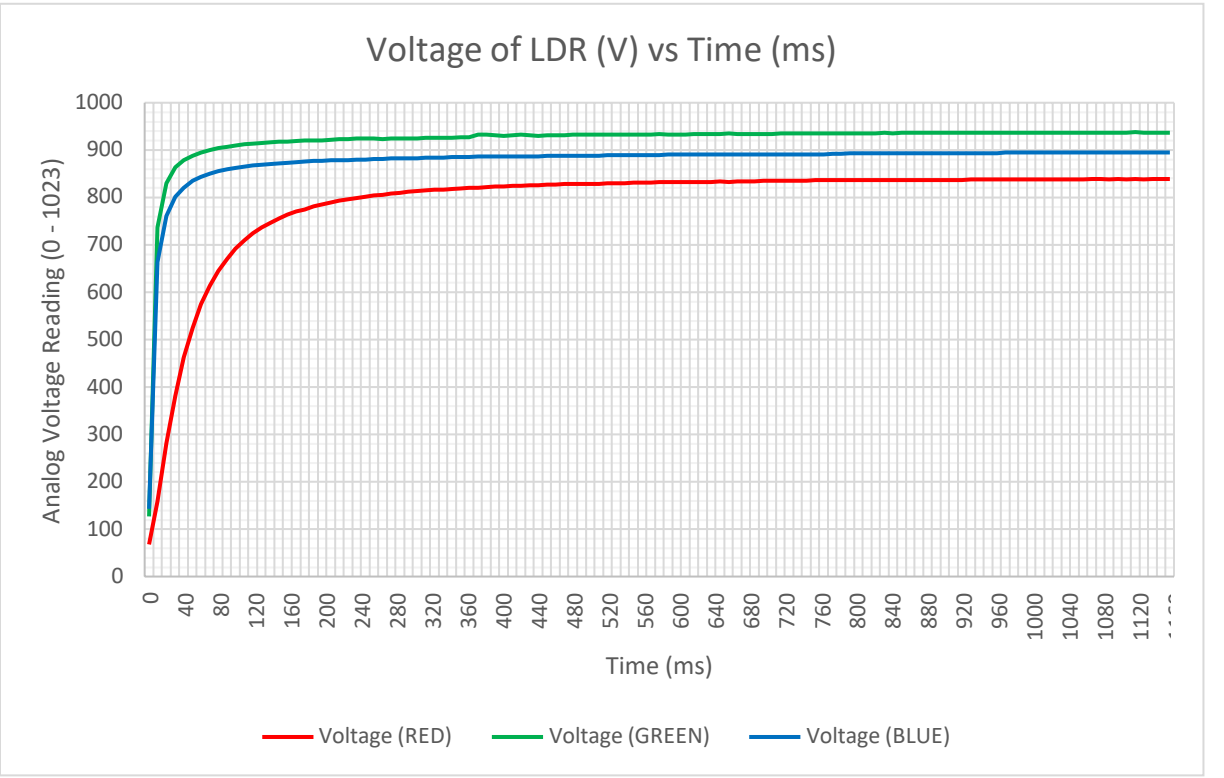

Figure 13. LDR Response Time

As seen in the above graph, the LDR stabilises much faster for the blue and green LEDs compared to the red LED. The LDR must be allowed to stabilised before taking a reading in order to obtain consistent and accurate readings from the LDR. In the program, a macro constant **`LDR_WAIT`** was defined at **320ms**, which is a very long time to wait, but ensures that the readings are stable and accurate. As seen from the graph above, at 320ms, the LDR has stabilised for all colours and all colours can be measured reliably. Albeit 320ms being a significant amount of time, reliability was prioritised over speed for our group.

Scaling RGB Analog Values

When reading from analog voltage readings (0 to 1023), the theoretical value for black is (0, 0, 0) and the theoretical value for white is (1023, 1023, 1023). However, due to ambient light, reflectivity of the paper underneath, etc – the real values for black and white are far from those.

For instance, an example RGB reading for white and black may look something like:

```
long white[3] = {805, 950, 919};
long black[3] = {355, 813, 719};
long range[3] = {450, 137, 200};
```

As seen above the white and black RGB arrays are quite distant from the ideal values, and the range between white and black can be said to be the operating range of the LDR. The bigger the range, the more distinct each colour looks from other colours.

Any new RGB measurement is scaled within the possible range by the following formula:

```
rgb[i] = (long) (((double) (average - black[i]) / (double) (range[i])) * 255.0);
```

`average` refers to the average reading for a particular coloured RGB, and the minimum possible value for that colour is subtracted from the reading, and the result is scaled on the possible range for that value such that the resultant value for RGB is between [0, 255].

Sometimes, certain coloured papers reflect more of the same coloured wavelength than the white, for instance the red paper may reflect more red light than the white maximum. Such values are artificially capped to be between 255 and 0 (if the coloured paper reflects too little of that colour).

Scaling the colour values in the 0 to 255 RGB space provides a more accurate classification and distinction between colours.

Colour Classification – Nearest Pythagorean Distance

Let RGB be a 3D space, measuring 0 to 255 on each axis. (0, 0, 0) would represent the theoretical black value and (255, 255, 255) would represent the theoretical white value. If we can plot 6 points for each colour (red, orange, blue, purple, green, white), when a new point or an RGB value is introduced, then the nearest existing point is the classification of the new colour.

Before determining the nearest existing colour, the 6 *scaled* colours must be calibrated and stored before running the program. Prior to which, the white and black arrays must be already measured. Then, the robot is placed on each colour on the maze, takes 90 samples (10 for each colour, repeated thrice), and the average is used as the calibrated `colour[6][3]` array.

```
long colour[6][3] = {
  {240, 110, 89}, //red - 0-255 Values
  {167, 173, 201}, //purple
  {255, 182, 110}, //orange
  {79, 182, 126}, //green
  {124, 219, 240}, //blue
  {255, 255, 255}, //white
};
```

The nearest existing point can be measured by using the Pythagorean distance between two points as such:

```
long squared_error = (rgb[0] - colour[i][0]) * (rgb[0] - colour[i][0]) +
                     (rgb[1] - colour[i][1]) * (rgb[1] - colour[i][1]) +
                     (rgb[2] - colour[i][2]) * (rgb[2] - colour[i][2]);
```

where the `rgb[3]` array represents the new colour to be identified and the colour array represents the calibrated colours.

3.6 Proportion-Integration-Derivative (PID) Controller

To enable the robot to move in a straight line and avoid bumps with the adjacent walls, a PID controller was used. A PID controller is a closed-loop feedback system that allows a system to delicately maintain balance over a set point.

PID Variables
A PID control system to travel in a straight requires the following variables:

1. **Current Distance/Process Variable (distance)** – This variable is measured in the PID loop and it must be as close as possible to the target value or setpoint.
2. **Target Distance/Set point (targetDistance)** – This is the value that the system must ideally maintain. However, there is bound to be some deviation from the target distance.
3. **Error (e)** – This is the variable that measures the deviation from the target distance.
4. **Proportion (K_p)** – This variable will perform adjustments to the motor speed *proportional* to the current error.
5. **Integral (K_i) –** This variable stores the cumulative sum of errors so far in the loop, and it eliminates accumulated error from the proportion variable.
6. **Previous Error (previousError)** – This variable memoizes the last error.
7. **Derivative (e') –** This variable meausres the rate of change error by using (error – previousError)/(Time Interval).
8. **Derivative Constant (K_d) –** This variable proportionally corrects the motor speed with respect to the derivate.

We decided that using the *integration* variable $\mathbf{K_i}$ is an overkill for our scope and the PID system will work adequately well without the integration factor.

The PID system's correction function u(e) can be represented as:

$$e = targetDistance - distance$$

$$e^{\cdot} = \frac{previousError - e}{\Delta t}$$

$$u(e) = K_p * e + K_d * e'$$

Equation 5. PID Control Function.

The correction function $u(e)$ is applied to the motor speed to correct the trajectory of the robot as it is moving. The following code snippet shows how it is implemented:

```
// PID Terms
double Kp = 12.0;
double Kd = 4;
double previousError = 0;
double targetDistance = 12.0;


// Calculate PID variables
double error = targetDistance - distance;
double derivative = error - previousError;

// Correction
int leftSpeed = motorSpeed + (Kp * error + Kd * derivative);
int rightSpeed = motorSpeed - (Kp * error + Kd * derivative);

// Keeping Maximum Speed within [-255, 255]
if (leftSpeed > 255) leftSpeed = 255;
if (rightSpeed > 255) rightSpeed = 255;
if (leftSpeed < -255) leftSpeed = -255;
if (rightSpeed < -255) rightSpeed = -255;

leftMotor.run(-leftSpeed);
rightMotor.run(rightSpeed);

// Update previous error
previousError = error;
delay(50);
```

Fine-Tuning PID Constants

The values the two constants **Kp** and **Kd** are critical for the PID to function. These two variables must be pruned such that the PID system is **critically damped**, which would remove oscillatory behaviour that will cause the robot to travel in a zig zag manner.

The algorithm we used to manually fine-tune our values is simple and is based on divide-and-conquer and systematic trial and error.

Fine Tuning **Kp**

1. Fix Kd to 0
2. Set Kp to a big number, for example 50.
3. The robot should either oscillate or behave erratically.
4. If the robot behaves erratically or oscillating too much (excessive correction)
    a. Half the current Kp.
5. Else if the robot is oscillating but not colliding with the walls (strong correction)
    a. Reduce Kp by small decrements till the oscillation is eliminated.
6. Else if the robot is oscillating but is colliding with the walls (insufficient correction)
    a. Increase Kp by small increments till there is sufficient correction.
7. Repeat Step 3 till oscillation is removed

Fine Tuning **Kd**

1. Fix **Kp** to the value found in the above steps.

2. Set Kd to a big number, such as 10;
3. The robot should behave erratically, therefore half Kd.
4. If the robot is jerky (too strong Kd)
    a. Reduce Kd by small decrements.
5. If the robot is not turning fast enough when placed close to the wall
    a. Increase Kd by small increments
6. Repeat till Steps 4 and 5 are not true.

Using the above algorithm, we arrived at our constants **Kp** = 12.0 and **Kd** = 4.0.

Missing Walls & Out of Range Values
In the maze, the walls adjacent to either the ultrasonic sensor or the IR sensor or both might be missing, causing the distance measured to be a very high value. In order to tackle these situations, we have the following decision table.

Conditions	Ultrasonic is out of range	T	T	F
	IR is out of range	T	F	-
Actions	distance = lastDistance	✓		
	distance = getUltraDistance()			✓
	distance = getIRDistance()		✓	

Table 3. Decision Table on Distance Value for Handling Missing Walls.

As seen above, if the ultrasonic gives a reading that is within range, then the PID control system will use that value to centralise the robot, as only one the distance to the wall on one side is needed to keep the robot travelling in a straight line.

When the ultrasonic sensor's reading is out of range, the IR's reading is checked. If the IR's reading is out of range as well, then the robot will travel "blind" by using the previous distance as the current distance. Since there is a maximum of 1 adjacent wall removed continusly, the robot's linear trajectory will not be disturbed in one grid. If the ultrasonic is out of range, but the IR's distance reading is in range, then the IR's value is used.

The ultrasonic's reading is preferred over the IR reading because its readings are more sensitive than the IR readings at longer distances (> 10cm), which is necessary for the PID control to make smooth adjustments.

3.7 Turns and Celebration Tune

Turns
After solving each way point challenge, the robot has to make a turn. The turns can be classified into two types:

1. Simple 90-degree turns
2. Compound turns

Simple 90 degree turns are the building block turns, to turn left and turn right, and the turning duration is hard coded as such:

```
#define TURNING_TIME 389
#define TURNING_TIME 389
#define TURN_WAIT 200
MeDCMotor leftMotor(M1);
MeDCMotor rightMotor(M2);
uint8_t motorSpeed = 200;

void turnRight() {
  leftMotor.run(-motorSpeed);
  rightMotor.run(-motorSpeed);
  delay(TURNING_TIME);
  stopMoving();
  delay(TURN_WAIT); // prevent slipping
}

void  stopMoving()  {
  leftMotor.stop();
  rightMotor.stop();
}
```

Compound movements simply consists of a simple turn, move forward by 1 grid, and the same turn again. And they are implemented using the simple functions. For the special case of UTurns, the turns are the simple turns but with twice the duration.

Celebration Tune

The celebration tune uses the inbuilt buzzer, and plays a few simple keys using the buzzer programming object.

```
bool isLineDetected() {
  int sensorState = lineFinder.readSensors();
  return sensorState == S1_OUT_S2_IN || sensorState == S1_IN_S2_OUT ||\
         sensorState == S1_IN_S2_IN;
}

void loop() {
    if (isLineDetected()) {
      stopMoving();
      //Colour Detection Below
    else {
      //Continue Moving Straight
    }
}
```

4. Fine-Tuning

Lastly, now that we have the essential circuits and systems in place, when calibrating and testing our robot, reliability of performance must be ensured.

4.1 Calibration of Colour Sensor

Skirting
To ensure that the LDR was not affected by ambient light, a layer of black skirting was taped to the chassis (*see Appendix*). Additionally, a "chimney" of black skirting was placed our the colour sensor to further isolate it and ensure that it is as dark as possible under the LDR. This ensured that the LDR readings were consistent and reliable regardless of ambient lighting.

Furthermore, the resistor values were chosen such that the LEDs were not "too bright" from the perspective of the LDR. If the LEDs were too bright, then all colours look similar under the robot, resulting in a smaller white – black range. Therefore, the LED's brightness were reduced using higher resistance values such that the a wider white – black range could be obtained.

Calibration Sequence
When calibrating the colour sensor, first the robot takes the average of 90 (10 for each colour, repeated thrice) samples from **white** and **black** papers to calculate the maximum range of RGB values. Then this range is incorporated into the code before calibrating for each colour.

Secondly, the robot is placed on each of the 5 colours on the maze (red, orange, purple, blue, green) and then colour arrays are populated with the average of 90 samples before the run.

These measured ensured that the colour sensor could identify every colour correctly in both the mystery maze and the usual maze used to calibrate the robot.

4.2 Calibration of IR Sensor

Calibration of Constants
As seen in Section 3.4, the IR sensor is able to function regardless of ambient IR radiation, therefore, the IR sensor needs to be calibrated only once using the lab's maze walls. After this calibration is done, the constants **k** and **c** do not have to be modified again, unless the reflectivity of the maze wall changes.

Build
The IR sensors are separated from each other by 2 columns on the breadboard (see Appendix) to minimise leakage from the emitter onto the receiver as much as possible. And, they are placed on the outer part of the chassis such that the distance between the IR and the wall is within the range (< 11cm) where the IR's readings are very accurate and predictable.

4.3 Increasing Ruggedness

During the testing phases, bumps and accidents are to be expected. However, the circuits are constructed to be rugged and to be able to withstand most accidents and bumps. The resistors are trimmed and fit snugly into the breadboard holes so that any collisions will not knock the resistors off.

The wires are trimmed so that only the required length is used, and jumper wires are avoided as much as possible to avoid loose cable issues and cables coming out when the robot bumps into walls.

The RJ25 cables were taped onto the mbot so that they will not come loose when we carry the robot or when the robot bumps into walls. And we colour coded the signal wires, the input and output jumper cables, and the power and ground cables for ease of debugging should any of these issues occur.

Last but not least, the robot was repetitively tested in multiple mazes to ensure that the robot is functional regardless of changes in ambient lighting, friction on the maze floor, positioning of walls, etc – to ensure that we did not fine tune our robot excessively to fit one maze.

5. Project Work & Challenges

5.1 Challenges Faced

Throughout the project we faced numerous difficulties with the colour sensor.

Colour Sensor
The first difficulty we faced was trying to find a set of resistor values that would provide a large enough voltage range to detect the various colours easily with minimal room for error. We had to use a trial and error method with numerous sets of resistors with different values to eventually find the right set of resistor values.

The second difficulty we faced was calibrating the colour sensor. The set of RGB values determined from the calibration process were always changing due to numerous reasons such as change in ambient light and people accidentally shifting the electrical components of the colour sensor. Thus, we had to recalibrate the sensor numerous times. To prevent the ambient light from tampering with our values, we had to reinforce the skirting around the colour sensor. Under the algorithm, we increased the input data of the colour sensor so that the mean value we would use in the algorithm would have a lesser magnitude of error.

Skirting
The next issue came when our reinforced skirts became a hindrance to the movement of our mBot, specifically when the mBot was required to turn at the junctions. We were reluctant to tamper with the skirting as this could risk too much ambient light being detected by the colour sensor, affecting its values. Hence instead, we made the mBot turn faster by changing its code, allowing it to overcome the hindrance caused by the skirting.

Ultrasonic Sensor

Initially, our Ultrasonic sensor kept on detecting the wooden panel of the maze in areas where the walls were removed. This would cause the Mbot to move closer to the wooden panel as it tries to adjust itself. Thus, if the mBot were to perform a 180 degree turn when it detects the color orange, the mBot would hit the panel. To prevent this we measured the distance from the mBot to the wooden panel at areas with the orange paper. We then change the code of the ultrasonic sensor to ignore values greater or equal to the distance we measured. Thus, the mBot stopped detecting the wooden panel in areas without a wall.

IR Sensor

Additionally, the IR sensor we had assembled was not sensitive enough to detect the walls of the maze at further distances. By the time the IR sensor detected the walls and the mBot tried to adjust itself, it would be too late and the mBot would collide with the walls. In order to make the IR sensor more sensitive to changes in distance between the mBot and the walls of the maze, we tried different sets of resistor values and finally was able to make the IR more sensitive.

Short Circuiting of RJ25

When constructing the mBot, we were informed to put sheets of paper imbetween our electrical components and the metallic frame of the mBot to prevent a short-circuiting as the metal points of the electrical components are in contact with the frame of the mBot. However, the paper we used was to thin and the electrical components had sharp solder points. Due to constant movement, a hole was formed in the paper between on of our RJ25 and the metallic frame. This caused our RJ25 to short-circuit and it no longer worked after that. After deducing this, we decided to use thicker sheets of paper to prevent this from happening again.

6. Appendix (Pictures of Robot)