CG1111A Mbot Project Report

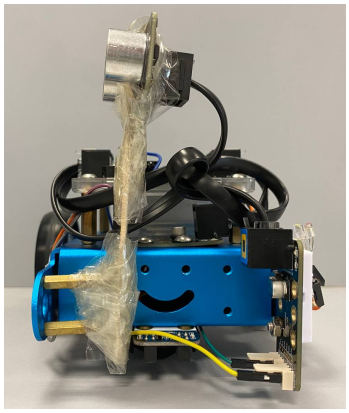
Studio Group 3, Group 5A

Name:	Student Number:
Sim Jun Hong	A0252616H
Seenivasaragavan Praneet	A0255314L
Sim Justin	A0257926N
Shao Sizhe	A0251963Y

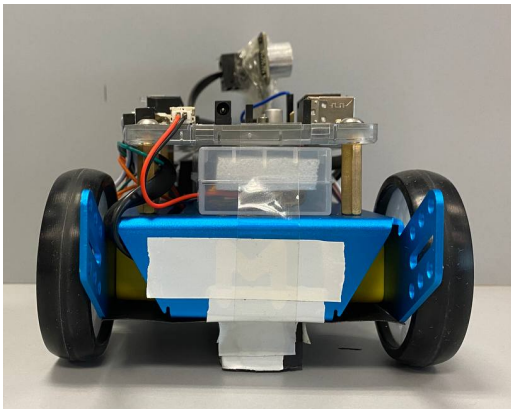
Table of Contents

1. MBot Pictures

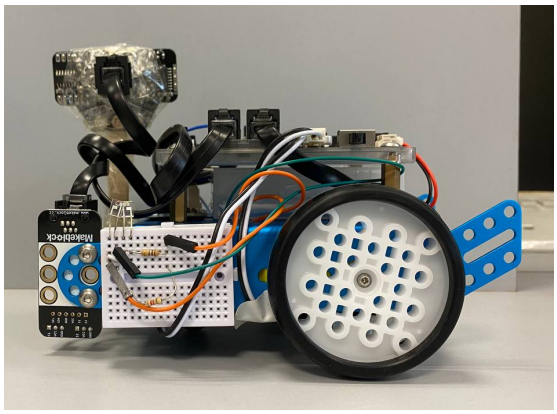
In this section we will be showcasing the fruits of our labour, our pride and joy, and the culmination of our efforts that is our MBot. Smile and wave, MBot, smile and wave.



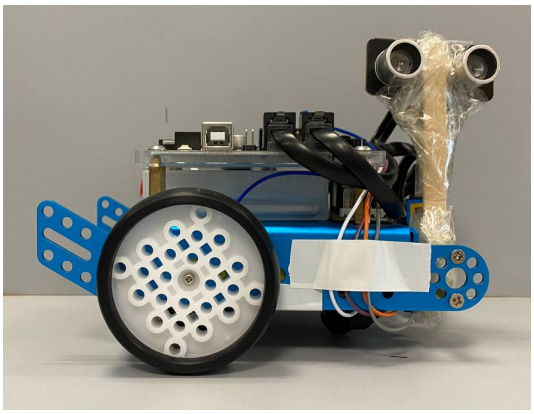
Front View



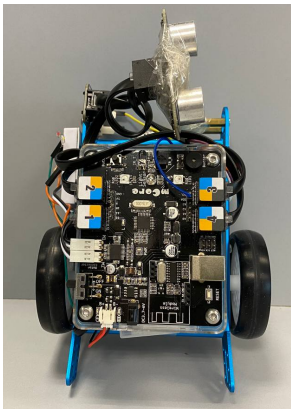
Back View



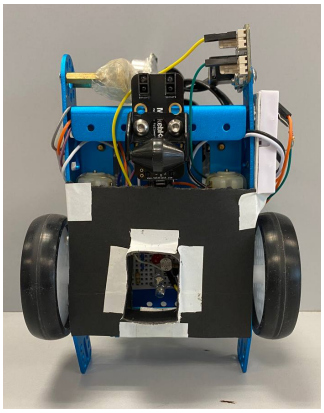
Left View



Right View



Top View



Bottom View

Figure I: Images of MBot

2. Sensor Breadboard Circuits

In this section, we detail the electrical circuits we have built to obtain our custom sensors, the RGB colour sensor and the IR proximity sensor, which help our mBot to accomplish various tasks and navigate the maze.

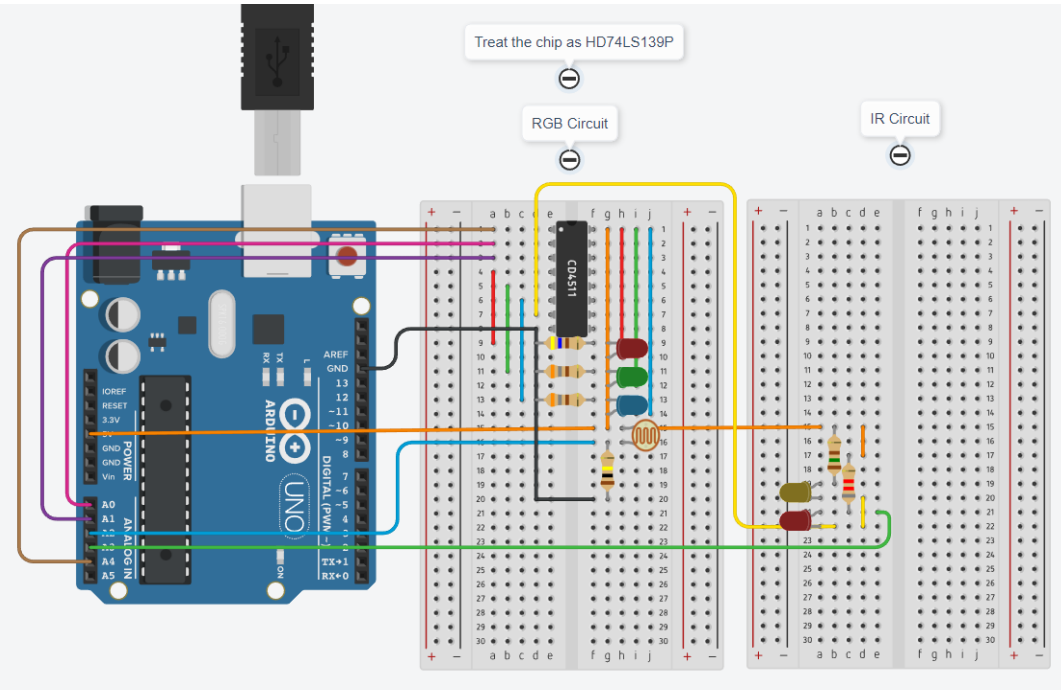


Figure II: TinkerCAD Diagram of Breadboard Circuit, Modified with Replacements

The biasing resistor values for the red, green and blue LED are 460Ω, 380Ω and 380Ω respectively. Our target current is 7mA, from the IV graph of the red LED, this corresponds to a voltage drop of 1.9V across the red LED. 5V - 1.9V = 3.1V. This results in a voltage drop of 3.1V across the biasing resistor. 3.1V / 7mA = 442Ω, hence 460Ω is an appropriate resistor value for the biasing resistor of the red LED. The same logic can be applied to obtain the resistor values for the green and blue LED.

Using the HD74LS139P chip, we are able to control 4 components using 2 inputs. Input A of the chip is connected to port A0 and input B of the chip is connected to port A1. When A and B are set to logic low, the red LED turns on. When A is logic high and B is logic low, the green LED turns on. When B is logic high and A is logic low, the blue LED turns on. When both A and B are set to logic high, the IR circuit is activated. We are also able to have a master switch, where we can turn off all the components by connecting the enable pin on the chip to port A4 and setting the logic level to high.

The voltage of the RGB circuit is read from port A2 and the voltage of the IR circuit is read from port A3.

Images for the various breadboard circuits are included below and the resistor value chosen for the IR detector will be elaborated on later in the report in the section “Subsystem: IR Proximity Detection”.

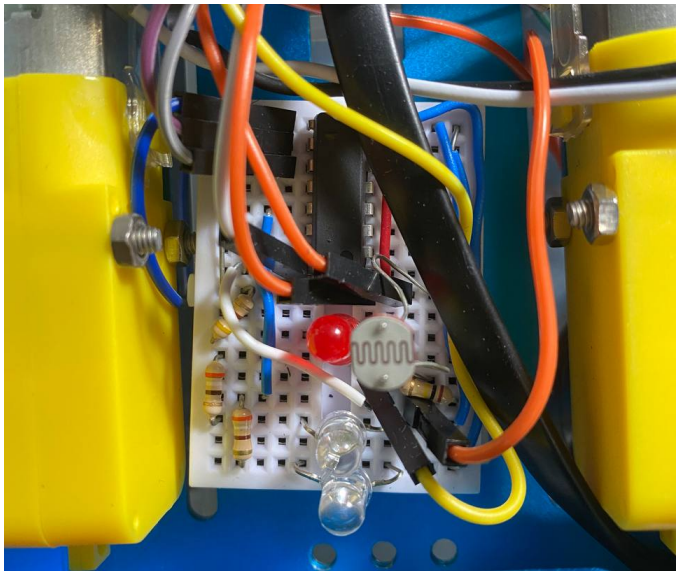


Figure III: RGB Detector Breadboard Circuit

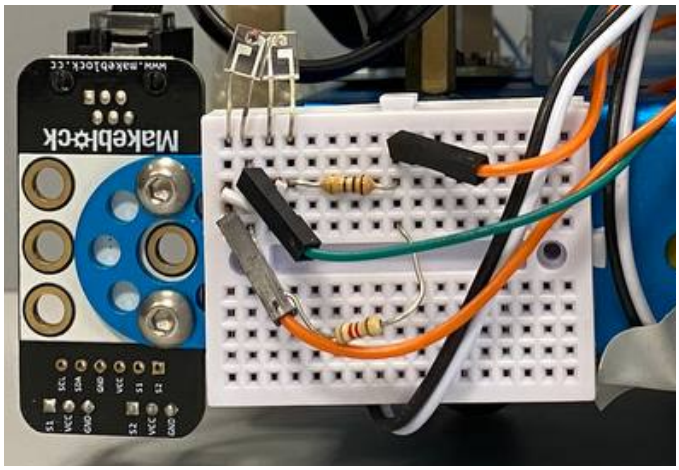


Figure IV: IR Proximity Detector Breadboard Circuit

3. Maze Algorithm Overview

In this section, we will describe the algorithm to navigate through the maze, expounding on some of the key design choices.

Here is the flowchart of the mBot's algorithm:

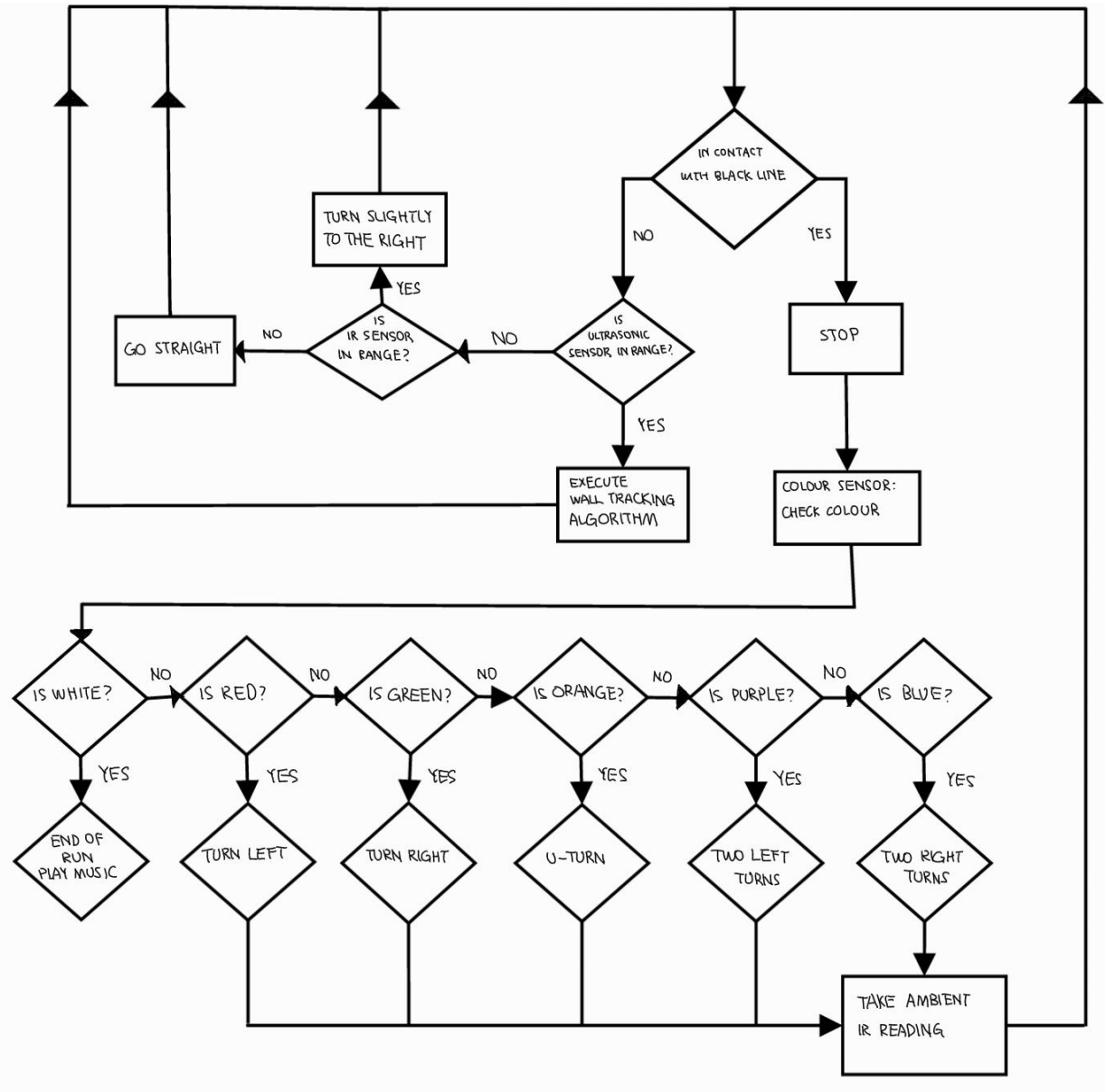


Figure V: Program Flow Diagram

During the run, our robot prioritises sensors in the following order: Line Sensor, Ultrasonic Sensor followed by IR Proximity Sensor. In the absence of a black line, the robot will perform

wall tracking based on the ultrasonic sensor readings when applicable. In the absence of both a line and ultrasonic sensor readings, our robot will move straight while reading values from the IR proximity detector, ensuring that our robot does not veer too far to the left. Once our robot has sensed a black line, it performs a colour detection algorithm to determine the colour it is on, and executes the tasks as required.

Colour	Action to Take
Red	Left Turn
Green	Right Turn
Orange	U Turn
Purple	2 Successive Left Turns
Blue	2 Successive Right Turns
White	End Run

Table I: Decoding of Challenges

Our robot also makes use of the onboard button on the MBot to simulate a software switch. Our robot initialises in the “OFF” state, during which our robot’s movements are halted. It switches to “ON” when the button is pressed and released. Pressing (and releasing) the button again while the robot is moving switches it back to “OFF”. For purpose of illustrating of our robot’s navigation algorithm, this step has been left out of the flowchart.

```
/**
 * This function returns a boolean value of whether the button has been pressed
 */
bool button_press() {
  int press = analogRead(BUTTON);
  if (press < 512) {
    return true;
  }
  return false;
}

/**
 * This function is used to control the ON-OFF state of the robot based on the button switch
 */
void switch_state() {
  int i = 0;
  // use of dummy counter to iterate until button is let go
  if (button_press()) {
    while (button_press()) {
      i++;
    }

    // Switches STATE from 0 to 1 or 1 to 0 (OFF->ON OR ON->OFF)
    if (STATE == OFF) {
      STATE = ON;
      // Reacquire IR baseline when switching to ON state
      find_ir_baseline();
    } else {
      STATE = OFF;
      off_lights();
    }
  }
}
```

Code Segment I: Implementation of Software Switch

For the end-of-run music to be played, we opted to play the chorus of “Nahida: Boundless Bliss” by Mihoyo (Mihoyo, 2022). Following the end-of-run music, our robot is set to “OFF” to prevent further actions from being taken.

```
if (STATE == ON) {
  // Set STATE to OFF to prevent further execution of movement
  STATE = OFF;
  off_lights();

  for (int i = 0; i < 7; i++) {
    buzzer.tone(song_1[i][0], (int)song_1[i][1]);
  }
  //buzzer.noTone();
  delay(NOTE_DURATION * 3.5);
  for (int i = 0; i < 14; i++) {
    buzzer.tone(song_2[i][0], (int)song_2[i][1]);
  }
  buzzer.noTone();
}
```

Code Segment II: End-of-Run Routine

Due to the length and complexity of the source code, full source code will not be included within this report but instead included within the enclosed zip folder under the folder “0_MBot”. Instead, snippets of code will be included in relevant sections.

4. Subsystem: Wall Tracking Algorithm

In this section, we discuss the implementation of our solution to the challenge through the use of software to complete various missions, namely wall detection and tracking, which forms the basis of localisation within the maze and serves as our means of traversal.

4.1 PID Control Theory

PID is a commonly used control loop mechanism employing feedback that is widely used in industrial control systems (ElProCus, n.d.). A proportional-integral-derivative (PID) controller uses a closed-loop feedback mechanism to control process variables to drive a system towards a setpoint. This control algorithm, as its name implies, involves 3 key factors: proportional, integral and derivative.

The proportional factor involves correcting a system proportional to its difference, the integral factor is used to eliminate the issue of steady-state error and consists of correcting a system according to the cumulative error resulting from the P factor. In contrast, the derivative factor is used to eliminate the issue of oscillations by providing a damping effect on a system.

4.2 Our Implementation

We opted to use a PD controller instead of a classical PID controller. Our use case employs an ultrasonic sensor to detect the distance between our robot and a wall to perform correction such that we can maintain a set distance between the two objects, thereby achieving wall tracking. We find that with the wall being a static target, making use of an integral gain would only serve to add an additional layer of complexity considering the lack of a steady-state error to correct. Our trials have also shown a classical PID controller to be less stable than our implemented PD controller, to be elaborated on below.

In our system, one wheel will always be travelling at maximum speed, while the other will decrease its speed according to a formula to achieve error correction. This enables us to travel at the maximum possible speed at any given point while still being able to achieve correction towards the desired distance between the robot and the wall, hence allowing our robot to achieve both speed and accuracy while traversing the maze. The correction that has to be applied to our robot through steering can be given by the following formula where $v(t)$ is the amount of correction that has to be applied in the opposing direction in PWM values.

$$v(t) = K_P * e(t) + K_D * \frac{de(t)}{dt}$$

Code implementation is as follows:

```
void pd_control() {
  // Calculate error as difference between ultrasonic and expected threshold
  // Calculate derivative as difference between current error and previous error
  // Calculate correction as a summation of P component and D component and typecast to int
  if (dist != OUT_OF_RANGE) {
    error = dist - threshold;
    error_delta = error - prev_error;
    correction_dble = kp * error + kd * error_delta;
    correction = (int)correction_dble;

    // Determine direction of correction and execute movement
    if (correction < 0) {
      L_motorSpeed = 255 + correction;
      R_motorSpeed = 255;
    } else {
      L_motorSpeed = 255;
      R_motorSpeed = 255 - correction;
    }
    move(L_motorSpeed, R_motorSpeed);

    // Initialise current error as new previous error
    prev_error = error;
  }
```

Code Segment III: PD Controller Implementation

In our algorithm, our robot calculates the correction required and does a subtraction from the required side to achieve turning. This turning is expected to cause the error to decrease and as such, result in our robot tracking the wall at a fixed distance.

4.3 Tuning Methods

We implemented the widely used Ziegler-Nichols Method for tuning the parameters of our PD controller input. This method is a form of Closed-loop P-Control Tuning Method involving empirically finding the value of Ultimate Gain, K_U and Ultimate Period, T_U. Ultimate Gain refers to the P Gain at which steady oscillations of the system first start to occur in the absence of the I and D factors, while the Ultimate Period refers to the period of these steady oscillations. The values of the two parameters are then calculated as a function of these values and implemented within the controller. The tuning method we used can be found below.

$K_U = 175, T_U = 0.6176$	K_P	K_D
PD	$K_U * 0.8$	$K_U * T_U * 0.1$

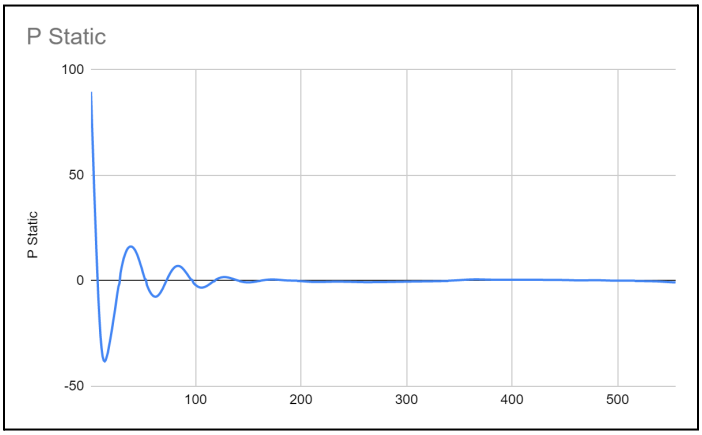
Table II: Tuning Parameters based on Ziegler-Nichols Method (Wikipedia, n.d.)

4.4 Further Trials

As mentioned earlier, we have also conducted additional trials prior to our decision in adopting the PD controller. Although there may have been slight inconsistencies in tuning between each trial along with other experimental errors, the following charts show a trend which indicates the superiority of a PD controller for our use case. In the following charts, the y-axes can be taken

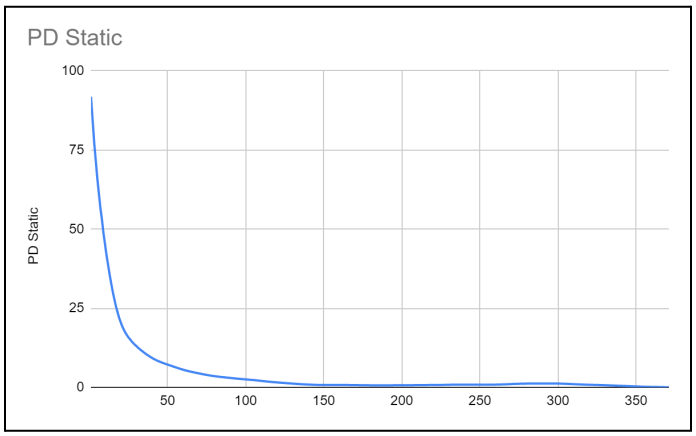
as a percentage error from the set point while the x-axes can be taken as the time taken in milliseconds.

In our first trial, following the attainment of the value for K_U and P_U, we tested a simple P controller to check the accuracy of our tuning. Results were as expected with multiple oscillations being made before our robot was able to stabilise at the desired set point. This poses significant problems as it introduces an element of lacking safety with our robot seemingly manoeuvring like a “drunk driver”.



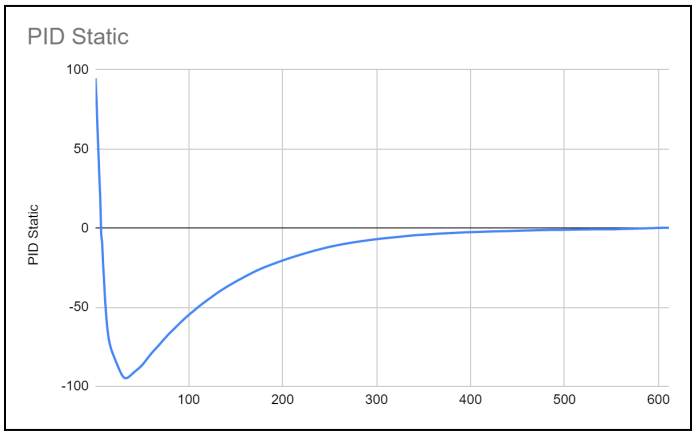
Graph I: P Controller

In our second trial, we aimed to fix this issue with the use of a PD controller, using the D component of the controller to provide a damping effect to prevent further oscillations after our robot has reached the desired set point. This implementation was largely successful, and further trials only served to cement this controller as our controller of choice.



Graph II: PD Controller

In our final trial, we tried to implement a PID controller as a test to see if it would prove to be an improvement over the PD controller we implemented earlier. While it cannot be further emphasised that little effort was made to perform proper calibration of these parameters, the PID implementation resulted in an overshoot prior to reaching the desired set point. This can be said to be a result of lag controller design principles of the integral component (Ozbay, 2019, #).



Graph III: PID Controller

Viewing the results of these trials, we decided that using a PD controller would be more desirable in our implementation of a wall-following robot. Although greater success with trials could potentially have been obtained with careful tuning of a PID controller, considering we already had a more-than-satisfactory working model, why fix something that isn't broken?

4.5 Improving Robustness

We initially mounted the ultrasonic sensor at a 45-degree angle on the right side frame of the robot. However, this resulted in a few issues which we resolved through the following methods.

Firstly, due to the dead zone of the ultrasonic sensor (3cm) (MakeBlock, 2016), imprecise turns would result in the ultrasonic sensor ending up within this dead zone and our robot would no longer be able to perform localisation due to inaccurate distance readings. Considering the hard-coded nature of our turns when performing challenges, we found this could become a significant problem as our robot may end up manoeuvring into the dead zone while performing tasks. Everyone loves having a handrail to hold onto but no one likes running headfirst into one, that would be an ouch! This was solved by introducing standoffs to the ultrasonic sensor such that it is inset into the chassis, allowing for the dead zone to lie within the chassis. This allows our robot to accurately measure its distance from the wall even when it is right beside the wall, thereby eliminating the problem of our robot potentially entering the dead zone of the ultrasonic sensor following turns.

The second was a far more specific issue, as we realised that mounting the ultrasonic sensor at a 45-degree angle would result in the sensor being able to detect the edge of the tables in the

absence of a wall on its right, resulting in our robot attempting to track said edge rather than perceiving the lack of a wall. The construction company may have forgotten to put up the “Keep Out” sign but our MBot has learnt its lessons and does not spend all its time looking downward staring at a smartphone. This was resolved by shifting the mounting point of the ultrasonic sensor to a higher point and reducing the angle at which it is overall positioned further from the ground plane, raising the sensor above the height of the edge of the table.

Images highlighting these changes are provided below.

Before

After

Figure VI: Comparison of Ultrasonic Placement Before & After Modifications

It should be noted that in the current iteration there is only one solid mounting point to which the ultrasonic sensor is secured. While there were initial concerns that this may allow for a pivoting action about the mounting axis, we are glad to say that this has not been a concern with tight screw connections when mounting. Additionally, we have placed another screw and nut such that it is in position to support the central standoff, preventing the ultrasonic sensor from performing (oriented to the above image) clockwise rotations.

5. Subsystem: IR Proximity Detection

In this section, we discuss the implementation of our solution to the challenge through the use of a custom IR proximity sensor as a means of sudden evasive action. With a one-dimension form of localisation characteristic of the ultrasonic sensor, there is a need for a layer of redundancy in cases where the ultrasonic sensor may fail.

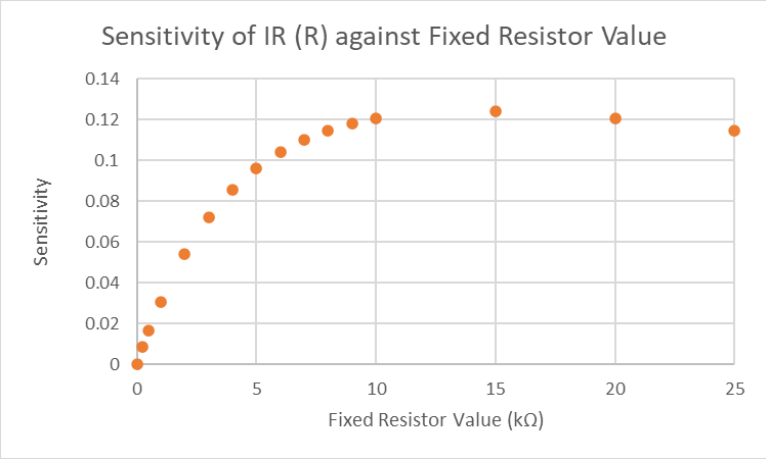
5.1 Purpose within our System

The IR transmitter and receiver system is used as an emergency measure to prevent unwanted turns in the direction opposing that of the ultrasonic sensor. Being a single-dimension sensor, the latter alone may not be entirely accurate in performing localisation. As such, the implementation of an IR proximity sensor, unreliable as it may be, helps provide us with an alternative means of traversing the maze, helping to prevent us from crashing into maze elements in our blindspot.

5.2 IR Receiver Characterisation

In designing the circuit for the IR proximity detector, we found difficulty determining the rating of the resistor we should use for its complementary fixed value resistor due to a lack of documentation of the resistance characteristics of the IR receiver. Compounded with the issue of the necessity of designing the circuit with ambient infrared rays in mind, we decided to perform our own characterisation of the IR receiver.

To perform this, we used the DSA lab as a reference for the ambient IR considering our runs would be performed in the said lab. We tested different fixed resistor values and tried to compare the difference in voltages read at 5cm and at 7cm, which would be our ideal range use case. This difference is taken as the sensitivity of our circuit, of which we tabulated the average across multiple scenarios (different times, locations within the lab, and days) and plotted the graph below.



Graph IV: Fixed Resistor Characterisation of IR Receiver

We saw a trend where the sensitivity would hit a peak between 10k and 15k ohms for the fixed resistor. With this in mind, we decided to use a 10k ohm resistor to complement the IR receiver when reading the voltage across the latter in order to achieve a greater variance of values when our robot detects left-side proximity.

5.3 Our Implementation

It cannot be mentioned enough that the IR proximity detector should not be used as a replacement for a distance sensor due to inaccuracies with this sensor, notably due to its lack of sensitivity to surrounding conditions, with the baseline at different points being a figure that may fluctuate at different points in the maze. Additionally, the range of the sensor can be considered to be rather abysmal and may result in unwanted movements, hitting the wall if we relied on it solely for navigation. As such, the sole purpose of the IR sensor would be as a proximity sensor and would move towards the right (since our IR sensor is on the left side of the robot) in the cases where our robot has been unable to effectively localise with the ultrasonic sensor and as such has veered too far towards a wall on the left side of the robot.

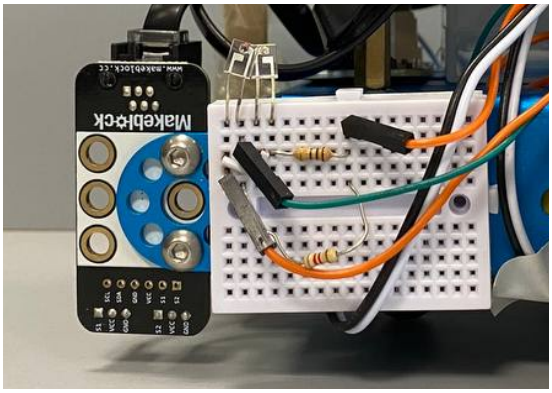


Figure VII: Implementation of IR Proximity Sensor

At the start of the run as well as after the completion of each challenge, the IR emitter is turned off to find the ambient IR value as a reference to compare with the IR readings when the emitter is on. Subsequently, the emitter is turned on once again in preparation for navigation around the maze.

Within our code, we decided on a threshold multiplier of 0.85 (i.e. 0.85x baseline IR value) as we found this roughly corresponded with 2cm from our robot and provided enough time for our robot to make corrections towards the right without crashing into the wall.

6. Subsystem: Colour Detection Algorithm

In this section, we discuss the implementation of our solution to the challenge through the use of software to complete various missions, namely colour detection for decoding tasks at each checkpoint.

6.1 K-Nearest Neighbour (KNN) Algorithm

KNN algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point (IBM, n.d.). This form of supervised machine learning enables us to classify low-dimensional (dimension referring to the number of independent variables) objects with high ease of implementation and without prior training (Simplilearn, 2022).

6.2 Our Implementation

We collected data pertaining to the computation of the grey difference by conducting 4 trials under very different circumstances (normal, glass surface, complete darkness) and taking the average values from these trials. With improvements to robustness (explained below), we found that the values obtained from our dataset have a very small standard deviation, indicating the effectiveness of these measures. The resulting values are used as the value for the black array and grey difference for the rest of our implementation.

	Average		
White	971.5	986.5	984.75
Black	939	907.5	913.5
Grey	32.5	79	71.25

Table III: Average Readings for Grey Diff Computation

We hypothesised that within a 3D plot, taking each of the individual RGB values as independent variables, each colour would occupy a distinct domain. To do this, we collected data on the various RGB values of the sample coloured papers (red, green, orange, purple, blue, white and black) under varying circumstances.

To test this hypothesis, we used CalcPlot3D (LibreTexts, n.d.) and obtained the following plot, which aligns with our hypothesis. As such, the implementation of the KNN algorithm would largely be effective in determining the colour our LDR is sensing when decoding tasks. For purpose of clarity, the colour white has been replaced with grey in the plot below. 6 datasets are included.

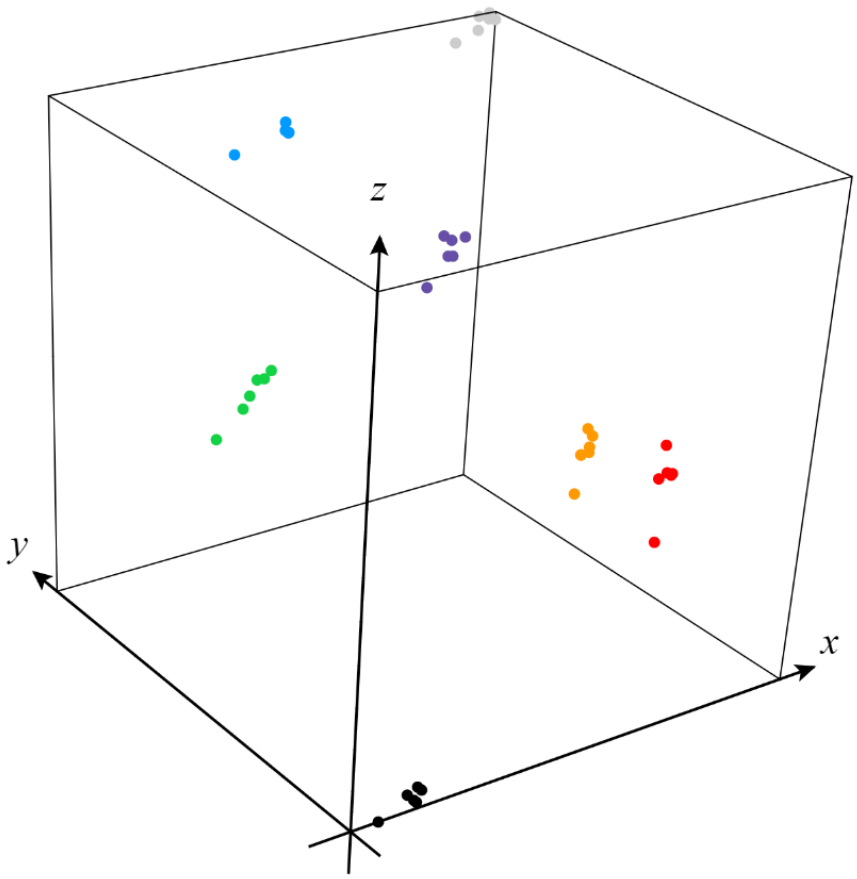


Figure VIII: 3D Point Plot of Colours by RGB Values (6 Datasets)

With a dataset of 70 values, our implementation of the KNN algorithm used a value of $k = 8$, taking the rounded-down value of the square root of the size of our dataset (Brand, 2020), striking a compromise between having insufficient points of comparison and having too large a time complexity characteristic of having too large a value of *k*.

When comparing a new data point to the dataset, the euclidean distance (Cuemath, n.d.) from each point in the dataset is taken and the mode colour of the closest *k* values is taken as the colour that the robot is sensing.

The dataset used for supervised training can be found in the appendix.

Code implementation is as follows:

```
int knn() {
  // Ordered from furthest k-th nearest neighbour to closest
  float closest_neighbour_dist[knn_k];
  int neighbour_index[knn_k];
  // Initialise Closest Neighbour Array
  for (int i = 0; i < knn_k; i++) {
    closest_neighbour_dist[i] = 450;  // 450 > sqrt(255 * 255 * 3) which is max possible value
  }

  // Find k closest neighbours
  float euclidean;
  for (int i = 0; i < dataset_count; i++) {
    for (int j = 0; j < 3; j++) {
      euclidean += sq(colourArray[j] - dataset[i][j]);
    }
    euclidean = sqrt(euclidean);
    // Compare to elements in closest_neighbour_dist
    for (int j = knn_k - 1; j >= 0; j--) {
      // Found desired position in closest_neighbour_dist
      if (euclidean < closest_neighbour_dist[j]) {
        // Shift elements in closest_neighbour_dist and neighbour_index
        for (int k = 0; k < j; k++) {
          closest_neighbour_dist[k] = closest_neighbour_dist[k + 1];
          neighbour_index[k] = neighbour_index[k + 1];
        }
        closest_neighbour_dist[j] = euclidean;
        neighbour_index[j] = i;
        break;
      }
    }
  }

  // Find mode color of k closest neighbours
  int neighbour_color[num_color] = { 0 };
  for (int i = 0; i < knn_k; i++) {
    neighbour_color[dataset[neighbour_index[i]][3]]++;
  }
  int closest_color = 0;
  for (int i = 1; i < num_color; i++) {
    if (neighbour_color[i] > neighbour_color[closest_color]) {
      closest_color = i;
    }
  }
  return closest_color;
}
```

Code Segment IV: KNN Algorithm Implementation

6.3 Effectiveness of Colour Sensor

In order to test the effectiveness of our algorithm, we passed random hues of each colour into the algorithm and tracked the precision of the outputs. Randomised colour codes were sourced online (CrispEdge, n.d.), and converted into decimal format before inputting them into our

algorithm. Results of these tests have yielded a 78% precision across 50 samples, with the breakdown of the scores by colour shown below. The colour white sees a lower sampling rate due to the randomiser using hues of grey as opposed to colours close to white, resulting in sample data not being as meaningful due to the use of the same values of red, green and blue.

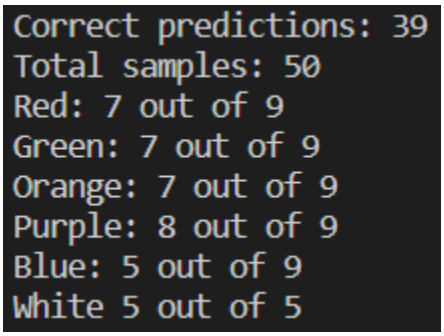


Figure IX: Results of KNN Algorithm Testing

The above test was conducted with 6 datasets. From the results, the algorithm is able to predict the correct colour most of the time, with blue being an outlier. However, we attribute this to the generated hues of blue being of a much wider spectrum than expected, with many of them visibly encroaching into the territory of purple instead.

Nonetheless, these results are exceedingly satisfactory considering the overfitted nature of our data since our algorithm has been able to, most of the time, correctly identify the colour provided. With the colours of the tiles used in the playfield being predetermined, and our data being trained precisely to handle these predetermined colours, we expect a precision of no less than 90% when the robot is navigating the maze.

A copy of the code (done in Python) and the sampling data used for this testing can be found in the folder “Colour Accuracy Trial”. Do note that the code only means to serve as a form of reference, and sadly does not follow good coding practices.

6.4 Improving Robustness

During initial testing of the colour sensor circuit, we found that despite being on the underside of the robot, it was still highly prone to external interference, namely light from surrounding sources. In order to minimise these sources of interference, we have given our robot a skirt to prevent light from reaching places that should not be seen.

Images highlighting these changes are provided below, much to the embarrassment of our MBot, so we do ask that you refrain from staring too much.

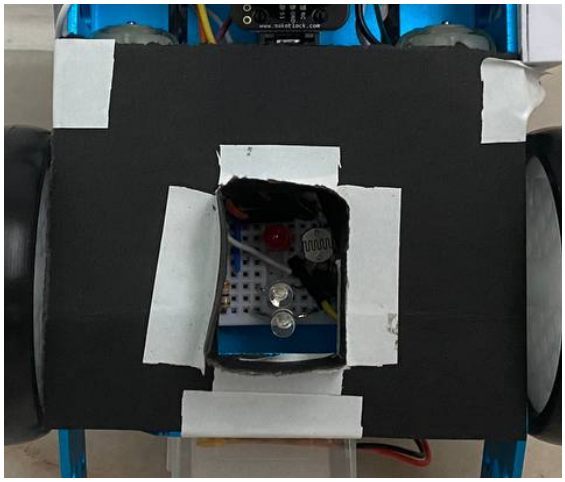


Figure X: Implementation of Skirt for LDR Isolation

We would really appreciate it if you stopped looking up others' skirts at this point. The isolation of the LDR with the implementation of the skirt has been largely successful. This is highlighted by our computation of the grey difference. As mentioned earlier, trials to collect data for this computation were done in largely extreme conditions. Despite this, we found the standard deviation of the results obtained to be very small, strongly indicating the independence of our colour sensor from surrounding conditions.

	Standard Deviation		
White	0.5774	0.5774	0.5000
Black	0.8165	3.1091	2.3805
Grey	0.5774	2.5820	2.0616

Table IV: Standard Deviation of Results from Grey Diff Computation

7. Challenges Faced

In this section, we will discuss some of the challenges we faced during the implementation of the robot and how we overcame these difficulties to accomplish the task ultimately.

7.1 Inconsistency of Custom Sensors

7.1.1 IR Proximity Detector

We found that the IR proximity detector was only effective at low ranges. This was extremely unreliable as the robot could get dangerously close to the wall before it detected anything, especially if the ambient IR conditions were to change. The range of output voltage was also very small, hence, there would be a large percentage error caused by environment variables.

In order to mitigate these problems, our team ensured that the ultrasonic sensor would be favoured over the IR proximity detector should our robot be able to detect readings from the former. Under this implementation, we managed to reduce the possible range of errors caused by the use of the IR proximity detector, achieving consistency when traversing the maze.

7.1.2 Colour Detection Sensor

Despite rigorous testing of the skirt for the colour sensor, we found that different lighting conditions still had the possibility of affecting our readings. Considering our colour detection algorithm worked on the basis of identifying the right RGB values, poorly tuned white and black readings would have a drastic effect on our robot's ability to detect the right colour, a problem we noticed under very specific conditions. Nonetheless, we banked on these conditions not being met for our graded runs and as such did not make any changes.

In hindsight, one way we could have solved this problem was through the performance of KNN on the specific analog values of each colour rather than doing so on the post-processing RGB values. This would help to eliminate our greatest source of error which was the calibration of white and black, an aspect we found to be exceptionally sensitive and had a great impact on our robot's ability to differentiate between the colours.

Additionally, it was observed that a depleted battery had critical impacts on the output voltage of the colour sensor, which caused the wrong colours to be detected. After finding out about this, we ensured the robot was fully charged before every lab session.

7.2 Low Precision of Hard Coding

Under initial testing at higher speeds, we found that our robot tended to be unable to accurately turn at a fixed angle despite not changing our program. We attributed this to the effect of sudden acceleration resulting in skidding wheels (Biswas & Kar, 2022, #) which would result in ineffective movement, causing our robot to be unable to complete the expected turn.

The resolution of this problem was a simple reduction in turning speed which helped to reduce the skid and produce more consistent turns.

Besides this, we had another issue with our robot performing a task, executing it and sensing the black line of the very task it just completed before it managed to exit the tile. To combat this, we included a slight move forward at the end of each challenge to guarantee that black line detection only recommences after our robot has left the challenge tile.

7.3 U-Turn

Another feature we implemented was to reverse slightly before executing a u-turn in order to prevent it from colliding with the wall in front of it while performing this turn.

7.4 Wire Management

The original wires were too long and would touch the wall when the robot was moving. We decided to trim the wires to the appropriate length so it does not protrude out of the breadboard to fix this issue.

This has created a compact robot design with almost no components protruding outside of the width of the wheels, preventing stray components from coming into contact with maze elements during the run.

7.5 Robot not moving straight

A peculiar problem we encountered during testing was our robot's inability to move straight even when coded to do so, with our robot veering slightly to the left when the wheels were coded to run at the same speed. This means that the right wheel was spinning faster than the left wheel. Adjustments were made to the code to ensure that both wheels would turn at the same speed physically, ensuring that the robot will go straight as intended.

```
// The things we need to do to go straight :(
move(200, 160);
```

Code Segment V: Motor Correction for Moving Straight

8. Division of Labour

In this section, we discuss the work division among the team, introducing each member's contributions to making this project a reality.

Name	Role
Sim Jun Hong	● Circuit Design ● General Software Development
S. Praneet	● Design of Maze Traversal Algorithm ● General Construction of mBot
Sim Justin	● Implementation of Wall-Tracking and Colour Detection Algorithm
Shao Sizhe	● General Construction of mBot ● Wire Management

Appendix

A1 Data Set for KNN Algorithm

Colour	R Value	G Value	B Value
Dataset 1			
Red	247	83	77
Green	55	174	120
Orange	247	148	74
Purple	151	145	187
Blue	103	213	233
White	255	255	251
Black	39	6	0
Dataset 2			
Red	227	69	51
Green	39	172	109
Orange	235	140	44
Purple	133	140	177
Blue	70	208	230
White	227	250	245
Black	15	0	0
Dataset 3			
Red	255	98	84
Green	78	185	130
Orange	255	153	66
Purple	156	153	191
Blue	102	214	238
White	251	255	255
Black	47	11	0
Dataset 4			
Red	251	88	73
Green	70	185	127
Orange	251	156	55
Purple	156	159	191
Blue	102	214	238
White	251	255	252
Black	47	14	0
Dataset 5			
Red	243	88	73
Green	62	179	123
Orange	251	150	62
Purple	164	153	191
Blue	102	214	234

White	243	253	248
Black	39	11	0
Dataset 6			
Red	251	93	73
Green	70	179	130
Orange	251	150	59
Purple	149	146	187
Blue	102	211	234
White	243	253	255
Black	39	4	0
Dataset 7			
Red	251	121	109
Green	98	188	144
Orange	251	172	98
Purple	172	175	202
Blue	125	221	241
White	255	255	255
Black	54	13	19
Dataset 8			
Red	243	101	91
Green	78	179	134
Orange	243	166	91
Purple	156	166	195
Blue	117	214	238
White	251	255	252
Black	47	20	12
Dataset 9			
Red	243	101	91
Green	78	179	134
Orange	219	156	80
Purple	133	159	180
Blue	94	208	227
White	227	246	245
Black	31	20	12
Dataset 10			
Red	211	91	73
Green	54	172	127
Orange	227	156	84
Purple	141	163	187
Blue	54	195	212
White	204	240	234
Black	11	16	39

References

Biswas, K., & Kar, I. (2022, September). Validating observer based on-line slip estimation for improved navigation by a mobile robot. *International Journal of Intelligent Robotics and Applications*, *6*(4), 1-12. ResearchGate. 10.1007/s41315-021-00216-w

Brand, A. (2020, May 23). *How to find the optimal value of K in KNN?* Towards Data Science. Retrieved October 27, 2022, from https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb

CrispEdge. (n.d.). *Random Color Generator (Hex color codes with color names)*. CrispEdge.com. Retrieved October 27, 2022, from https://www.crispedge.com/random-colors/

Cuemath. (n.d.). *Euclidean Distance Formula - Derivation, Examples*. Cuemath. Retrieved October 27, 2022, from https://www.cuemath.com/euclidean-distance-formula/

ElProCus. (n.d.). *PID Controller : Working, Types, Advantages & Its Applications*. ElProCus. Retrieved October 24, 2022, from https://www.elprocus.com/the-working-of-a-pid-controller/

IBM. (n.d.). *What is the k-nearest neighbors algorithm?* IBM. Retrieved October 27, 2022, from https://www.ibm.com/sg-en/topics/knn

LibreTexts. (n.d.). *CalcPlot3D*. LibreTexts. Retrieved October 27, 2022, from https://c3d.libretexts.org/CalcPlot3D/index.html

MakeBlock. (2016, December 21). *Me Ultrasonic Sensor - STEAM Projects*. Makeblock. Retrieved October 27, 2022, from https://www.makeblock.com/project/me-ultrasonic-sensor

Mihoyo. (2022, November 1). *Character Demo - "Nahida: Boundless Bliss"* | *Genshin Impact*. YouTube. Retrieved November 9, 2022, from https://www.youtube.com/watch?v=dpZhGzZ3hNc&ab_channel=GenshinImpact

Ozbay, H. (2019). *Introduction to Feedback Control Theory*. Taylor & Francis Group.

Simplilearn. (2022, September 9). *How to Leverage KNN Algorithm in Machine Learning?* Simplilearn. Retrieved October 27, 2022, from https://www.simplilearn.com/tutorials/machine-learning-tutorial/knn-in-python

Wikipedia. (n.d.). *PID controller*. Wikipedia. Retrieved October 24, 2022, from https://en.wikipedia.org/wiki/PID_controller