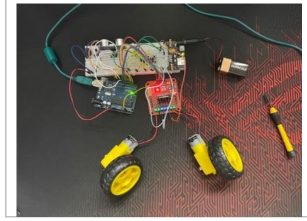
Development Process

Journal Entries (Critical Thinking)

Image	Journal Entry
Image of class result:	**Incorporation of the L298N Component in the System** *Monday 10th of July* Today was the initial prototyping of the L298N module and DC Motors as the parts had just arrived. This hour was spent by connecting all the L298N ports to the Arduino. Also, the wires were soldiered onto the DC Motors for a more reliable connection
Image of class result: 	**Incorporation of the L298N Component in the System** *Tuesday 11th of July* Test code was used to trigger the DC Motors. The wheels spun slowly and were not properly synchronised. However, upon numerous tests, I incorporated another power supply module to separately power the L298N which resulted in a higher performance. This was because the Power Supply Module allowed the L298N to receive 5V from another source, totalling to 10V into the L298N.

Commented [IW1]: Can you explain why this improvement happened?

Image of class result:	**Transferring the Breadboard to a Pinboard** *Thursday 13th of July* This period marked the initial construction of the pinboard. Terminals are being used as the permanent part of the pinboard as this allows for more flexibility with the wires in the circuit. For example, a longer wire may be needed for the button to reach its slot in the casing. Not much progress was made physically on the breadboard as planning was taken to figure out where each terminal should go to make the long run easier. A few terminals for the buttons and the power supply module for the L298N were permanently soldiered.
Image of class result:	**Transferring the Breadboard to a Pinboard** *Monday 17th of July* The terminals and wiring (such as resistors) responsible for connecting the buttons to the circuit are now permanently implemented in the pinboard. Just to get another task out of the way, the base was laser cut. Although no progress was made with the base cut out, it was found that the pinboard dimensions did not fit in the designated area. After some critical thinking, a plan for a block to be used as a spacer was recorded.

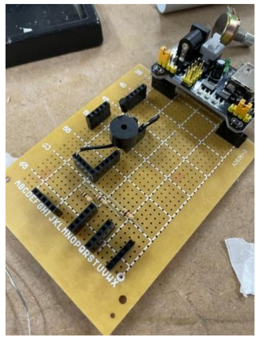
Image of class result: 	**Transferring the Breadboard to a Pinboard** *Tuesday 18th of July* All terminals and pins were soldered into the pinboard. It was decided that the buzzer will not be permanently soldiered onto the pinboard to preserve the wires for future classes. However, I was quite sure if the buzzer pins would reach the bottom of the terminal, so it was quickly tested by connecting jumper cables to the terminal and then running it through the circuit. The buzzer was the first component to be fully connected in the circuit. Also, testing was conducted to ensure that the wiring was working properly. When connecting the GND terminal, it seemed too messy to have a pin connect diagonally under the pinboard, so it was instead, connect between the upper side of 2 terminals while travelling under the buzzer for better compatibility.
Image of class result:	**Transferring the Breadboard to a Pinboard** *Wednesday 19th of July* All components were successfully plugged into the pinboard and between the relevant components. All the individual systems were tested against the Circuito testing code and most components were working fine. However, one issue remained; The L298N would not send enough voltage to its motors (more information and values can be found in the testing table). This created a huge problem because its mobility is one of the key systems in this solution.

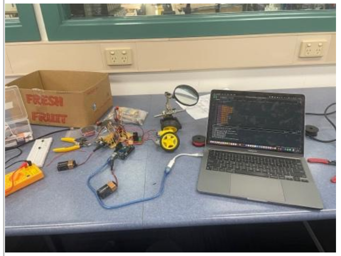
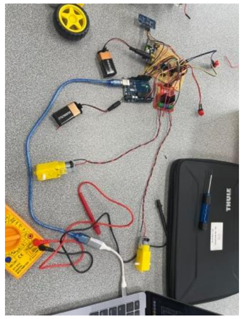
Image of class result: 	**Pinboard Testing** *Thursday 20th of July* This class purely consisted of testing the L298N component to figure out where the voltage stops being supplied. All the wires in the pinboard were found to be running the correct value of 5 volts including the wires entering the voltage ports of the L298N. Only the output ports show a lack of voltage which could mean there is a faulty component in the L298N itself.
Image of class result: 	**L298N Testing** *Monday 24th of July* Upon retrieving my project from storage, it was observed that the copper pins on both DC motors were severely damaged from intense bending. To fix this (and prevent more damage) I removed the wires from 'through the slot' and instead placed the wire over what was left of the copper with a decent amount of soldering metal. Also, during this fix I increased the length of the wires so that there is less movement pressure on the copper pins. Similarly, I intertwined the positive and negative wires to create a more solid wire. After replacement of wires, the L298N was still faulty, more testing should be conducted next lesson.

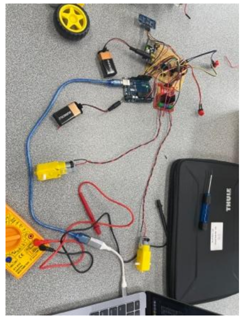
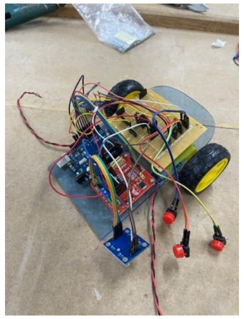
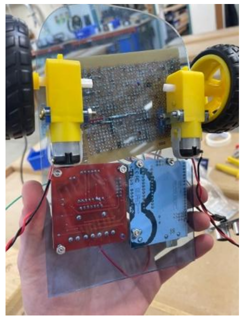
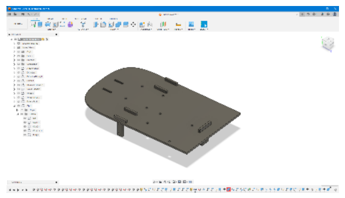
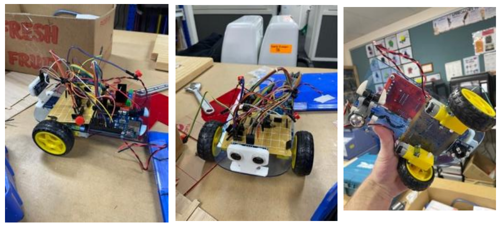
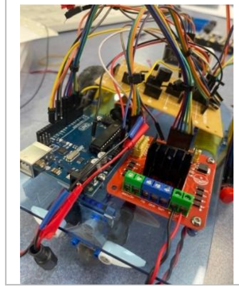
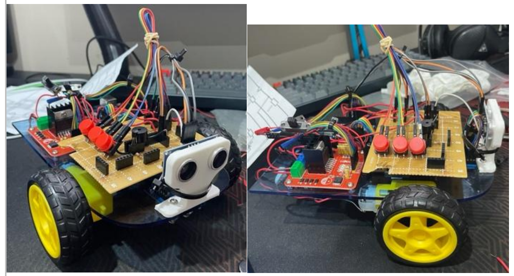
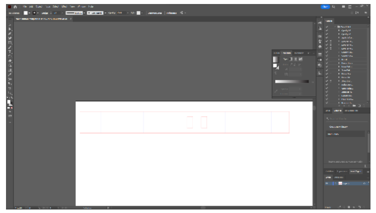
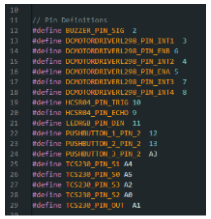
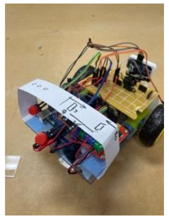
Image of class result: 	**L298N Testing** *Monday 24th of July* With assistance from Mr. Watson, it was concluded that there was not enough voltage being outputted to the DC motors (as previously found) however, Mr. Watson focused the possible problem towards the PSM. The 5V from the PSM was not enough voltage to power the DC motors yet, it was working on the breadboard which was weird. To solve this, I took a 9V adapter and stripped the positive and negative wires. From here I soldered the wires and directly connected the battery into the L29N component. This allowed the L298N to work perfectly and also left the PSM redundant.
Image of class result:  	**L298N Testing** *Thursday 27th of July* In today's double I worked on constructing the proper solution. This involved mapping all the components to the plastic base. The holes for the Arduino Uno were previously cut out, alongside the L298N however there was an issue. The pre-cut holes were mapped to a different L298N model which made the hole misalign. To solve this issue, I manually cut holes for the physical component and used tape to prevent cracking at the plastic is delicate towards power tools. Upon initial construction, Miss. Rhiny had found a plastic-bagged kit which contains a few components and screws. The screws perfectly fir my components and were secured on the plastic base using bolts. There are still a few cuts that must be made using the laser cut machine (as these are rectangles and not simply holes).

Image of class result: 	**Fusion Model Changes** *Sunday 30th of July* I changed the Fusion 360 model to incorporate the TCS230 colour sensor and the pivot wheel. The front of the base has two rectangle holes which allow for the pins (and wires) to shoot through the base and reach the Arduino Uno. Once the base is laser cut, I will manually drill the holes to properly secure the TCS230.
Image of class result: 	**Solution Construction** *Tuesday 1st of August* The base was re-laser cut to incorporate the slots for the TCS230 colour sensor and the back pivot wheel. This involved the disassembly and reassembly of my solution. The back pivot wheel was secured to the back on the base however, there was a problem with the component being too lose. To solve this, rubber Lego pieces were placed under the base and a wire was used to add pressure on top of the base which resulted in the pivot wheel becoming firmly locked in place. To further develop my solution in regard to the floating components (i.e. components that attach to the casing and not the base, like the buttons), I found a 3D model of an Ultrasonic Sensor holder from Thingverse. I used 2 screws to secure both the stand and the TCS230 colour sensor.
Image of class result: 	**Solution Construction** *Wednesday 2nd of August* As the solution has been constructed, with all components where they should be, I worked on incorporating the battery supply which will enable the solution to operate without direct connection to the computer. I positioned the batteries directly under the pinboard which allowed the board to be raised and secured while also providing power to the components. On the first battery I extended the Pos/Neg wires by soldering another wire to reach the L298N. On the second battery, I extended the wires also however, I incorporated a DC male port so that it could power the Arduino Uno.

Commented [IW2]: A larger photo with Annotations would be better.

Image of class result: 	**Solution Construction** *Thursday 3rd of August* After the batteries have been properly secured, I worked on minimise the number of wires and their length in the solution. All male-to-male wires were replaced with industry standard red or black wires that were perfectly cut to size. This eliminated the overall complexity in the solution which was unnecessary and allows for a case to be better fit to the base. However, all male-to-female wires remained in the solution despite their size. Die to these wires coming from the Arduino kits, I am not able to change their length also, I cannot replace the wires as this type of connection is only available from the kit. The buttons were propped on the side of the pinboard so they can be pressed (for future testing).
Image of class result:   	**Button Stand Prototyping** *Tuesday 8th of August* During this class I spent the first half (30 minutes) setting up the code for my solution. This consisted of defining all the ports in which the Arduino Uno communicates with, creating a template for all the variables. After this, I start to prototype different designs for the button holder. A rectangular piece of paper was used to model a potential design before it was then measured (buttons were measured with a calliper, lengths were measured with a ruler) and created in Adobe Illustrator.

```
// Pin Definitions
#define BUZZER_PIN_SIG  2
#define DCMOTORDRIVERL298_PIN_INT1  3
#define DCMOTORDRIVERL298_PIN_ENB 6
#define DCMOTORDRIVERL298_PIN_INT2  4
#define DCMOTORDRIVERL298_PIN_ENA 5
#define DCMOTORDRIVERL298_PIN_INT3  7
#define DCMOTORDRIVERL298_PIN_INT4  8
#define HCSR04_PIN_TRIG 10
#define HCSR04_PIN_ECHO 9
#define LEDRGB_PIN_DIN  11
#define PUSHBUTTON_1_PIN_2  12
#define PUSHBUTTON_2_PIN_2  13
#define PUSHBUTTON_3_PIN_2  A3
#define TCS230_PIN_S1 A4
#define TCS230_PIN_S0 A5
#define TCS230_PIN_S3 A2
#define TCS230_PIN_S2 A0
#define TCS230_PIN_OUT  A1
```

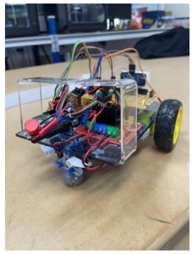
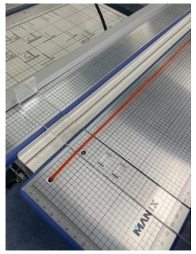
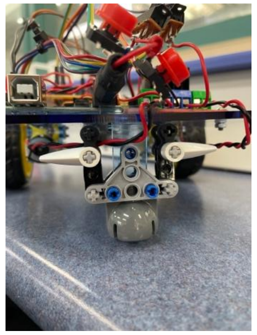
Image of class result:  	**Button Stand Prototyping** *Thursday 10th of August* During this class, I prototyped the physical button stand. 3 versions were laser cut with mini alterations, such as the button slot sizing and the overall height sizing. The plastic rectangles were heated and bent using the school's acrylic heater and bent alongside the square tool.
Image of class result: 	**Button Stand Prototyping** *Tuesday 15th of August* The pivot wheel has 4 rubber pieces which act as a 'suppression system' to support the load of medicine and objects that are in transport. Also, it was found that the rubber pieces compressed too much which results in a loose pivot wheel. To solve this, I inserted 2 axles into the rubber pieces which significantly limited the amount of compression, resulting in a more secured pivot wheel. Subsequently, I continued to prototype the physical button stand. All versions were numbered. It was found that the most recent design had a width which was 6 mm shorter than the width of the base, so the width was altered in Adobe Illustrator. The plastic rectangles were heated and bent using the school's acrylic heater and bent alongside the square tool. There was not enough time to finish bending the final design.

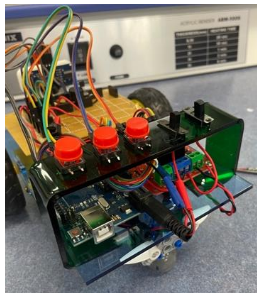
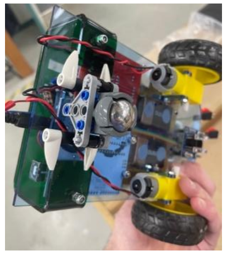
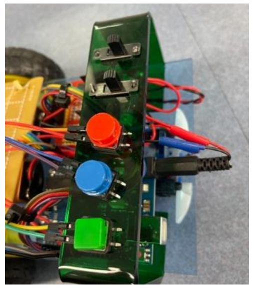
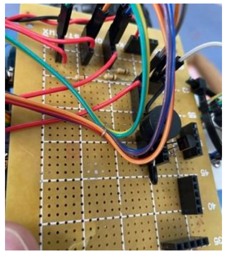
Image of class result:  	**Button Stand Finalisation Version 1.0** *Thursday 17th of August* This class involved the drilling and attaching of the button stand. The stand was placed on top of the base but under the Arduino Uno and L298N, this allowed for the holes to be marked on the stand. The holes were drilled then the stand was placed in its proper position. The screws were secured with nuts however, to properly hold the stand in place and give way for the curve, a nut was placed in between the bottom wing and the base. This helped to keep the stand level. Once the stand was properly fixed, the toggle switches had to be unsoldered and resoldered to fit into the laser cut slots. The 3 buttons were secured onto the plastic stand with double sided tape. This was chosen due to both surfaces being flat and plastic. Double sided tape was ineffective when trying to secure the toggle switches as the side 'wings' are not flat therefore, the wings could not stick to the tape. Tape may be placed on the sides of the switches in future versions.
Image of class result:  	**Button Stand Finalisation Version 1.1** *Monday 28th of August* When I returned to my project, I found that the three buttons had fallen off the double-sided tape and a wire had disconnected from one of the toggle switches as a result of the switches not being properly secured into place so, during this class I wanted to properly secure the buttons and toggle switches to the stand. For the buttons, I secured the wires to the pinboard with a wire (as seen in image 2) to restrict movement and allow for the wires to stay bent in a position that influenced the buttons to remain on the stand, hoping it would increase contact time with the tape. In order to secure the buttons, double sided table was not sufficient as the flaps are not flat, meaning that the surface that is in contact with the tape is very minimal. Instead, I used a 1.5 mm drill bit to drill a hole through the flaps and the plastic. From here, as there were no screws with a cap and the correct width, I inserted a nail which was then bent with force from a pair of pliers. This resulted in a secure placement of the switch. Upon securing the buttons to the stand, I replaced the button heads from all red, to red; green and blue.

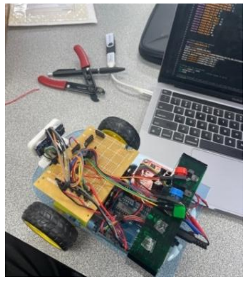
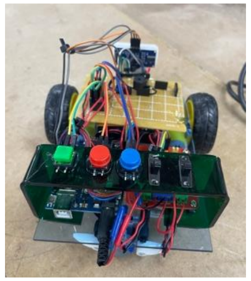
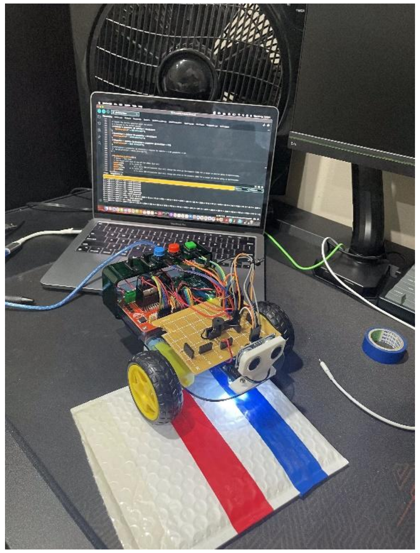
Image of class result:  	**Button Stand Finalisation Version 1.2** *Tuesday 29th of August* When I returned to my project, once again I was presented with the buttons that had fallen off the stand. I simply removed the wire and decided to try another way. To solve this issue, I went for a more permanent solution, hot glue. As the cables were already in the right angle, I applied hot glue to the stand and pressed the buttons to the material, creating a mould. This can be seen in image 2. When updating my code (assigning port values), I noticed that the wire for the buzzer was missing. To fix this, I ran a wire under the pinboard, connecting both ports.
Image of class result: 	**Colour Detection System** *Wednesday 30th of August* During this class, I took the time to configure the TCS230 component. I followed a setup guide from Random Nerd Tutorials which hosted explanations and code which was used. The first stage involved running a frequency script that outputted the raw frequencies of the TCS230. The exact frequencies that are picked up by the TCS230 are unique and vary across each individual TCS230 manufactured and environment/raw colour. I held each colour (red, green, and blue) directly under the TCS230, from up close and 5cm away (from the sensor itself to the surface). From here I recorded the lowest and highest values to create a bracket from every colour. Once these values were recorded, I incorporated the colour sensing code into my script. The script utilises the map() function to convert the frequency brackets to RGB values. This allows the correct colour to be used within code (i.e., I can use the colour as a variable). Furthermore, (after testing (v2)) the colour sensing code was altered to incorporate the white colour detection. This was achieved by creating a relationship that checked to see if the blue colour value is greater than the green and red colour value and, makes sure that both the green and red colour values are greater than 1. This specific relationship was achieved through trial and error. The detection of white now allows the TCS230 to know when it has veered off the coloured line.

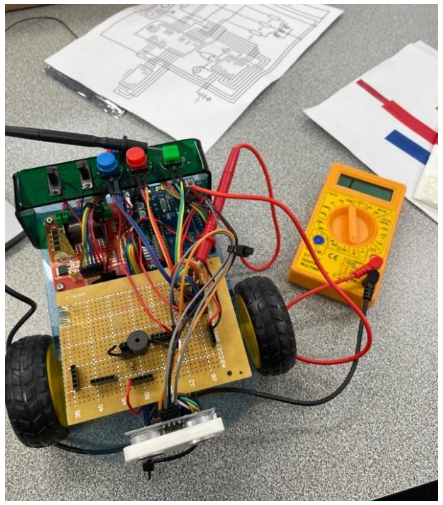
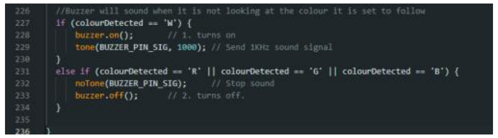
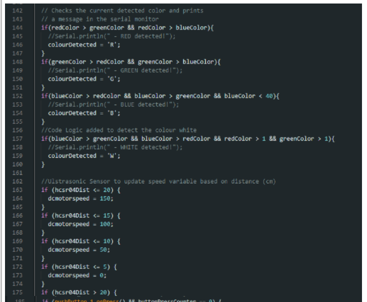
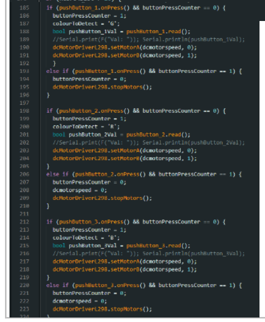
Image of class result: 	**Button Stand Finalisation Version 1.2** *Thursday 31st of August* Upon completion of the TCS230 configuration, I started to work on the functions of the buttons. The blue button printed the present distance from the ultrasonic sensor to the nearest surface in the serial monitor, the red button rotated the DC motors, and the green button was to sound the buzzer. Both the blue and red buttons worked; however, the green button did not work. To solve this error, I used the concept of trial and error to identify the fault. A physical observation of the circuit was issued which found no problems. From here, a multi-meter was used to check for a complete circuit. When the prongs are connected properly and the button was pressed, the multi-metre emitted a beep. I then switched the mode to DCV 20, and the prongs did not read 5V when the button was pressed. This meant the circuit was complete however, no power was running through the button. To efficiently use my resources, Mr. Watson took over the testing stage and observed the circuit with a multi-metre. The teacher removed the pinboard (resulting in the wires coming off). After the visual inspection, the wires were reconnected and when the test code was run, the button was letting out a signal. As this test was found to only be a mistake due to dodgy wire placement, it will not be recorded in the testing table.

Image of class result:
Image 1: Buzzer script
Image 2: Assigning the 'colourDetected' variable and changing the DC motor speed based on the HC-SR04 distance.
Image 3: Button scripts
 	

`//Buzzer will sound when it is not looking at the colour it is set to follow`
`if (colourDetected == 'W') {`
`    buzzer.on();        // 1. turns on`
`    tone(BUZZER_PIN_SIG, 1000); // Send 1KHz sound signal`
`}`
`else if (colourDetected == 'R' || colourDetected == 'G' || colourDetected == 'B') {`
`    noTone(BUZZER_PIN_SIG);     // Stop sound`
`    buzzer.off();       // 2. turns off.`
`}`
 	
`}`
	Line Following Script Part 1
	Sunday 3rd of September
	
	During today I wanted to start working on the foundations of the line following script, i.e. what must be written and working before the actual line following functions can be implemented. The foundation consisted of 3 subsystems: changing speed of the motors based off the distance to the nearest object; having the solution set to follow a specific colour based on the button pressed and comparing that to the colour currently being detected; and having a buzzer sound when the colour detected is not the line to be followed. To actively change the speed, the DC motor speed must be set to a variable rather than a single integer. This allows the speed to be changed no matter where it is referenced in the script. The DC motor speed variable was initially set to 200 (max) and then several if



```cpp
// Checks the current detected color and prints
// a message in the serial monitor
if(redColor > greenColor && redColor > blueColor){
  //Serial.println(" - RED detected!");
  colourDetected = 'R';
}
if(greenColor > redColor && greenColor > blueColor){
  //Serial.println(" - GREEN detected!");
  colourDetected = 'G';
}
if(blueColor > redColor && blueColor > greenColor && blueColor < 40){
  //Serial.println(" - BLUE detected!");
  colourDetected = 'B';
}
//Code Logic added to detect the colour white
if(blueColor > greenColor && blueColor > redColor && redColor > 1 && greenColor > 1){
  //Serial.println(" - WHITE detected!");
  colourDetected = 'W';
}

//Ulstrasonic Sensor to update speed variable based on distance (cm)
if (hcsr04Dist <= 20) {
  dcmotorspeed = 150;
}
if (hcsr04Dist <= 15) {
  dcmotorspeed = 100;
}
if (hcsr04Dist <= 10) {
  dcmotorspeed = 50;
}
if (hcsr04Dist <= 5) {
  dcmotorspeed = 0;
}
if (hcsr04Dist > 20) {
```

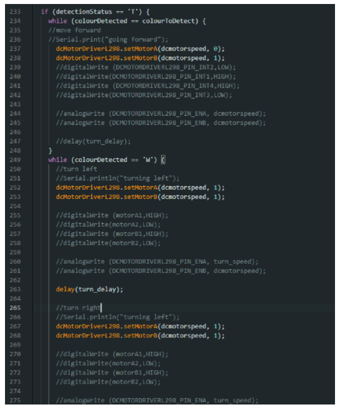


```cpp
if (pushButton_1.onPress() && buttonPressCounter == 0) {
  buttonPressCounter = 1;
  colourToDetect = 'G';
  bool pushButton_1Val = pushButton_1.read();
  //Serial.print(F("Val: ")); Serial.println(pushButton_1Val);
  dcMotorDriverL298.setMotorA(dcmotorspeed, 0);
  dcMotorDriverL298.setMotorB(dcmotorspeed, 1);
  }
else if (pushButton_1.onPress() && buttonPressCounter == 1) {
  buttonPressCounter = 0;
  dcMotorDriverL298.stopMotors();
}

if (pushButton_2.onPress() && buttonPressCounter == 0) {
  buttonPressCounter = 1;
  colourToDetect = 'R';
  bool pushButton_2Val = pushButton_2.read();
  //Serial.print(F("Val: ")); Serial.println(pushButton_2Val);
  dcMotorDriverL298.setMotorA(dcmotorspeed, 0);
  dcMotorDriverL298.setMotorB(dcmotorspeed, 1);
}
else if (pushButton_2.onPress() && buttonPressCounter == 1) {
  buttonPressCounter = 0;
  dcmotorspeed = 0;
  dcMotorDriverL298.stopMotors();
}

if (pushButton_3.onPress() && buttonPressCounter == 0) {
  buttonPressCounter = 1;
  colourToDetect = 'B';
  bool pushButton_3Val = pushButton_3.read();
  //Serial.print(F("Val: ")); Serial.println(pushButton_3Val);
  dcMotorDriverL298.setMotorA(dcmotorspeed, 0);
  dcMotorDriverL298.setMotorB(dcmotorspeed, 1);
}
else if (pushButton_3.onPress() && buttonPressCounter == 1) {
  buttonPressCounter = 0;
  dcmotorspeed = 0;
  dcMotorDriverL298.stopMotors();
```

statements followed which changed the variable's integer value based off the distance that was being read by the HC-SR04 at the time. If the path was clear, the solution would travel at the top speed however, if there is an obstacle within 5cm of the sensor, the DC motors would stop. This worked when the motors were coded to spin directly from the loop however, there were issues when the buttons triggered the DC motors to spin. All three buttons were programmed to assign a colour to be followed. The script had 2 variables called 'colourToDetect' and 'colourDetected'. The 'colourToDetect' variable was assigned by the button and the 'colourToDetect' variable was assigned by the TCS230 script. Another variable named 'buttonCounter' was included to approach the on/off problem. In theory, if the counter is at 0, the pressing of the button should follow one if statement which then makes the counter 1. As a result, then next button press will result in the 2nd if statement being called. However, as seen in the testing log, his system is yet to work. Lastly, two if statements were implemented in the code under 2 different circumstances; if the TCS230 is actively looking at the RGB colour white; if the TCS230 is actively looking at the RGB colours red, green, or blue. In the instances were the TCS230 was looking at the RGB colour white, the buzzer would sound. Additionally, if the TCS230 was then actively looking at the coloured line, it would be silent.

Image of class result: 	**Line Following Script Part 2** *Monday 4th of September* To solve the limitations of using a singular colour sensor, potential solutions were brainstormed (with teacher assistance) and visualised through pseudocode. If the TCS230 is detecting the right colour, it will simply move forward. Furthermore, if the TCS230 detects white, it will proceed to turn left and right until the line is found once again. Originally, it was decoded to have the robot spin in a single direction. However, this would result in the robot spinning to far to the left which resulted in an accidental U turn. The series of motor spins happen inside a while loop which will run the code until the condition is false i.e., the code will keep moving forward while reading the right colour and will move to the other while loop. Mr. Watson also put forward the question regarding what happens when the robot reaches the line. This solution will be thought of once the robot is able to follow a line.
Image of class result: **Brainstorming was simply written and presented on a whiteboard**	**Line Following Script Part 3** *Monday 4th of September* To solve the limitations of using a singular colour sensor, potential solutions were brainstormed (with teacher assistance) and visualised through pseudocode. If the TCS230 is detecting the line, it should move forward; if the TCS230 is detecting another colour, it should look for that colour. This is done through turning a certain degree to the left then turning right a certain degree. It was noted that a full 360 is impossible as the robot would result in doing a U turn. This concept was implemented into the code through two while loops that were activated under the conditions; if the TCS230 is detecting the right colour; if the TCS230 is detecting another colour.

```
if (detectionStatus == 'T') {
  while (colourDetected == colourToDetect) {
  //move forward
  //Serial.print("going forward");
    dcMotorDriverL298.setMotorA(dcmotorspeed, 0);
    dcMotorDriverL298.setMotorB(dcmotorspeed, 1);
    //digitalWrite (DCMOTORDRIVERL298_PIN_INT2,LOW);
    //digitalWrite(DCMOTORDRIVERL298_PIN_INT1,HIGH);
    //digitalWrite (DCMOTORDRIVERL298_PIN_INT4,HIGH);
    //digitalWrite(DCMOTORDRIVERL298_PIN_INT3,LOW);

    //analogWrite (DCMOTORDRIVERL298_PIN_ENA, dcmotorspeed);
    //analogWrite (DCMOTORDRIVERL298_PIN_ENB, dcmotorspeed);

    //delay(turn_delay);
  }
  while (colourDetected == 'W') {
    //turn left
    //Serial.println("turning left");
    dcMotorDriverL298.setMotorA(dcmotorspeed, 1);
    dcMotorDriverL298.setMotorB(dcmotorspeed, 1);

    //digitalWrite (motorA1,HIGH);
    //digitalWrite(motorA2,LOW);
    //digitalWrite (motorB1,HIGH);
    //digitalWrite(motorB2,LOW);

    //analogWrite (DCMOTORDRIVERL298_PIN_ENA, turn_speed);
    //analogWrite (DCMOTORDRIVERL298_PIN_ENB, dcmotorspeed);

    delay(turn_delay);

    //turn right
    //Serial.println("turning left");
    dcMotorDriverL298.setMotorA(dcmotorspeed, 1);
    dcMotorDriverL298.setMotorB(dcmotorspeed, 1);

    //digitalWrite (motorA1,HIGH);
    //digitalWrite(motorA2,LOW);
    //digitalWrite (motorB1,HIGH);
    //digitalWrite(motorB2,LOW);

    //analogWrite (DCMOTORDRIVERL298_PIN_ENA, turn_speed);
```