

LINE FOLLOWING ROBOT

SAT Completed by Noah Bakr

ABSTRACT

Upon recent analysis of the transportation of medicine in a hospital, a robot is offered as a solution. The robot can assist the staff through transportation of medicine and equipment from one place to another. By incorporating this system into Australian hospitals, nurses can focus on more important situations that require a physical presence, instead of being held by running small errands or having to put a hold on those tasks.

Noah Bakr

Systems Engineering SAT

Investigation (Design Brief/Evaluation Criteria)

Since the introduction of the coronavirus in 2019, hospitals have had more patients than ever before, with every room being occupied at all times. Due to the continual overflow of patients, staff are forced to work longer hours. Also, there is not enough staff to facilitate every person, leaving patients unattended for hours. Now the simple answer would be to hire more staff however, hiring more staff is simply not an option. There is a shortage of nurses in Australia and it is estimated that nursing workforce of Australia will face a shortage of up to 123,000 nurses by 2030. (Mannix, 2021)

Over recent years we have seen manual factories be transformed into automatic, machine run factories that are far more efficient than humans. Automation can act as a temporary orderly which can assist hospital staff or patients and can help during an emergency (usually triggered by a call alarm). Automation can be introduced using a line following robot. The robot can assist the staff through transportation of medicine and equipment from one place to another. By incorporating this system into Australian hospitals, nurses can focus on more important situations that require a physical presence, instead of being held by running small errands or having to put a hold on those tasks.

Input	Process	Output	Loop
<ul style="list-style-type: none">• Detect weight of object/s needed to transport• Button that triggers the system to follow an order (digital input)• Collision detection to keep the robot from hitting an object in its path (ultrasonic sensor)• An RGB sensor that relays colour data from the line on the floor• The room number/s that the object/s must be delivered to	<ul style="list-style-type: none">• Calculate the shortest path to reach every room efficiently by ordering the numbers by floor level then room number in ascending order• Check if there is an object in the storage compartment• Digital input triggers code• Follow a line• React to an obstruction	<ul style="list-style-type: none">• Drive/stop motors• Open storage compartment	<ul style="list-style-type: none">• During the route of the robot, the ultrasonic sensor is continuously reading and relaying measurement values to the Arduino Uno. When an object is blocking its path, the threshold is met• When robot is not in transit (awaiting the collection of objects), the force sensor will be continuously reading and relaying weight values to the Arduino Uno

Constraints

- Have a digital input which triggers the system
- Have at least one output mechanism
- The program/system must repeat. This means once the threshold is met the program continues to measure and provide an output
- Have a colour detection solution that can tell the lines from one another

Considerations

- Be automated
- Have different storage slots for different rooms

Factors that influence the creation and use of the system

- The weight of the object/s
- The speed of which the system will travel at
- The foot traffic of the journey
- The time to complete this project (deadline)p
- The technology that is available currently
- The materials that is available currently

Evaluation Criteria

No.	Criterion	Justification	How will this be achieved?	Testing Method
1	Does the robot follow a line?	This factor is important since the robot must be able to adopt the already-placed lines in a hospital to ensure the correct and most efficient path is followed to deliver the medication.	This will be achieved by programming a colour sensing system which utilises the RGB sensor component to record both the width and colour of a line.	The system will be tested by checking to see if the RGB colour sensor is able to distinguish between different coloured lines and returns a value
2	Does the robot safely operate within the vicinity of people and obstructions?	This factor is important since the robot will be deployed in a high foot traffic area and completing the journey is crucial to the purpose of the robot. Failure to do so could result in an unfortunate outcome.	This will be achieved by programming a detection system which utilises the ultrasonic sensor component to measure distances from an object in the path.	The system will be tested by placing objects in front of the active robot and recording the decisions made.
3	Is the robot able to transport objects?	This factor is important since the robot's main purpose is to replace the nurse's need to transport medication.	This will be achieved by creating a compartment to house the objects that require delivery.	The system will be tested by placing a range of objects inside the storage compartment and recording the robot's movement.
4	Is the robot able to repeat itself whilst staying switched on?	This factor is important as the robot must be able to transport goods to a specific room (point B) but, then also be able to return to base upon the request from the nurse.	This will be achieved by programming a condition in the Arduino which awaits a digital input to initiate movement.	The system will be tested by initiating the robot's transport feature but, then also initiating the robot's return feature, checking to see if the robot is able to stay on and await a digital input

Research

Drive System (Choice of Motor)

A Servo Motor is an electric rotary motor that allows for precise control of angular velocity, position, and acceleration. The servo motor consists of a sensor for position feedback which enables the rotor to always control its position and speed. Also, unlike the DC motor, servo motors have adopted a closed loop system. (Saini, 2022)

The closed loop system refers to the cycle of feedback, error detection, and correction. The real-world position, speed and/or torque of the servo motor is fed through to the sensor to compare the predetermined value against its position and then calculates the errors between them. The servo motor then corrects the current motion in real time using the error information to ensure that the system can achieve the intended precision. (OMRON, 2007)

This component is not practical in a drive system due to the closed loop system and the inability to continuously spin forward or backwards to achieve motion.

A DC motor is an electric motor that converts DC electrical energy into mechanical energy (rotation of shaft) for driving a mechanical load. There are two types of a DC Motor, brushed and brushless. Both types can be used in the same situation, supplying the same purpose or rotation however there are slight differences. A brushed motor has a shorter lifespan due to the brushes wearing out over time. This factor is not good to be used in a hospital scene as higher maintenance is required. Speed and acceleration on the brushless motor is higher which allows the system to transport medication fast and the increased acceleration allows for the robot to re-enter its route as quickly as possible when it is disturbed by an obstruction. Also, to respect the environment of the patients, brushless motors are significantly quieter. (Millett, 2022)

In terms of performance, Servo motors are focused on precision and position while DC Motors focus on high torque and, being able to operate under a load. DC motors can continuously spin forward or backwards to achieve motion. For this drive system, a DC motor is most appropriate.

In order to gain full control over the DC motors, an Interface L298N DC Motor Driver Module can be implemented to control the speed and spinning direction. (Last Minute Engineers, 2023)

Line Detection Subsystem

An infrared (IR) line sensor detects the presence of a black line by emitting infrared (IR) light and detecting the light levels that return to the sensor. They do this using two components: an emitter and a light sensor (receiver).

The greatest limitation to this component is its lack of colour detection. The IR sensor can only detect that there is a black line and return a 0 or 1 value (detecting a line or not). This function renders the component unable to fulfill the design brief. (Future Learn, 2022) However, this problem can be solved with an RGB sensor.

The TCS230 Colour Sensor is equipped with a lens filter and 4 white LEDs. The lens allows for a more focused and precise reading of colour to ensure the returned value is not affected by any surrounding colours. The 4 LEDs make sure that the lens pick up the true colour of the surface, preventing a potential problem with room lighting. (Santos & Santos, n.d.)

Obstacle Detection Subsystem

Black is the absence of colour. This is because a black surface absorbs all colour of white light and reflecting no light, as a result being black. Due to the nature of the IR sensor infrared component, the infrared light beam will be absorbed by the black object, making the obstruction seem invisible to the component. This can lead to the robot crashing into obstructions in the hospital which is dangerous. (Olesen, 2013)

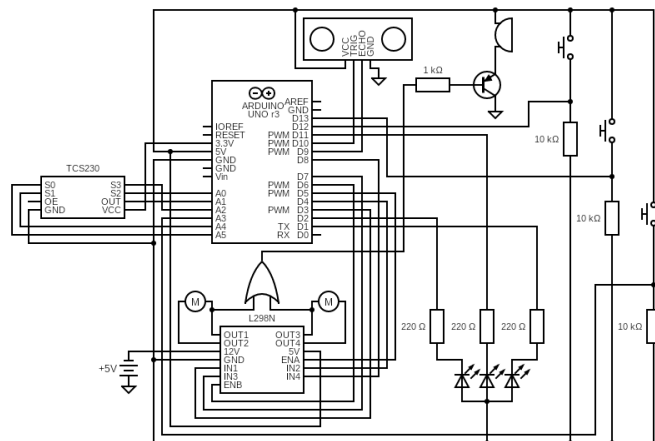
An ultrasonic sensor does not take the colour of an object into account instead, the sensor uses sound waves. Sound waves are emitted and will bounce off any and all surfaces, ensuring accuracy when detecting obstacles. Due to the high number of rooms in a hospital environment, along the robot's path, variations in light can be a regular occurrence for the IR sensor to pick up. This factor causes the IR sensor's values to fluctuate and become inconsistent/inaccurate. The ultrasonic sensor is completely insensitive towards light, dust, smoke, mist, vapour, lint etc. (MaxBotix, 2017)

Preferred Option

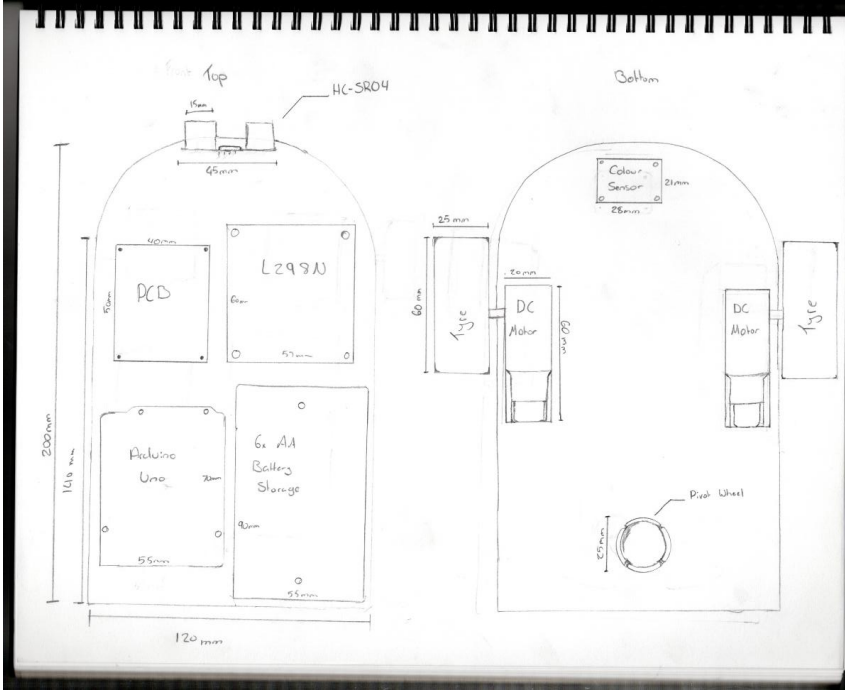
Link to the Circuit.io can be found [here](#).

How the system works

- The ultrasonic sensor continuously reads the distance from itself to the closest object at all times to ensure that the robot does not collide with an obstruction.
- The system is equipped with a TCS230 colour sensor. The sensor will check to see if the determined colour is being detected. More information on the sensor can be found [here](#).
- Wheels are attached to the DC Motors, which are connected/controlled to the L298N DC Motor Driver module.
- The RGB LED provides a visual queue to the nurse, so they are able to know which coloured line is to be followed and to check if they have selected the right line.
- The buzzer will sound a short beep to indicate the departure or arrival of the trip.



Physical System Sketch



Fusion 360 Model
Link: <https://a360.co/3MMgo4u>

Planning

Timeline: Production Plan

Step	Process	Tools/equip/materials	Safety	Time
	Circuit Creation			

1	Gather electrical components and laser cut base	Plastic base, Arduino Uno, Ultrasonic Sensor, RGB Sensor, L298N, DC Motor x2, Printed Circuit Board (PCB), Secondary Power Supply, Button x3, RGB LED, Buzzer, Resistors		
2	Place main electrical components in their corresponding slot	Arduino Uno, Ultrasonic Sensor, RGB Sensor, L298N, DC Motor x2, Printed Circuit Board (PCB), Secondary Power Supply, AA Battery Pack	Do not force components	20 mins
3	Lock in electrical components with screws	3D Printed (non-threaded) Screws or metal (threaded) screws		
4	Connect main components together with jumper wires and printed circuit board (PCB) as the central hub	Jumper wires, Main Electrical Components, Printed Circuit Board (PCB)	Make sure wires are not tangled	20 mins
5	Connect secondary components to main components with jumper wires	Jumper wires, Button x3, RGB LED, Buzzer, Resistors		
6	Plug Arduino Uno into computer and install test script	USB 2.0 Printer Cable, Arduino System, Computer, Arduino IDE Script		10 mins
7	Run Ciculo.io test script	Arduino system		
8	Solder all wires to their respective slot in the Printed Circuit Board (PCB)	Printed Circuit Board (PCB), Electrical Components, Jumper Wires, Soldering Iron, Solder Wire, Sponge	Safety glasses	30 mins
9	Run Ciculo.io test script	Arduino system		10 – 20 mins
10	Locate and fix any errors if script does not run	Multimeter, Soldering Iron, Solder Wire, Sponge	Safety glasses	
11	Plug Arduino Uno into computer and install runtime script	USB 2.0 Printer Cable, Arduino System, Computer, Arduino IDE Script		
	Exterior Installation			
1	Gather mechanical (exterior) components and casing	Pivot Wheel, 3D Printed Casing, 3D Printed Door, Large Wheel x2		

2	Attach wheels to DC motors	Large Wheel x2	Do not force components.	15 mins
3	Attach pivot wheel to front of base (underside)	Pivot Wheel, 3D Printed (non-threaded) Screws or metal (threaded) screws		
4	Combine 3D printed parts. Slide door inside case railing	3D Printed Casing, 3D Printed Door		
5	Slide 3D printed case over the Arduino system base	3D Printed Casing, Laser cut base with Arduino System		
6	Attach electrical components to casing	3D Printed Casing, Ultrasonic Sensor, Button x3, RGB LED		10 mins
Total Time				2 Hours

Third Party Materials, Components and Processes

Category	Third party Material	Description	Owner	Link (if applicable)
Software				
	Arduino IDE	The compiling software which is needed to program the Arduino Uno	Arduino	https://www.arduino.cc/en/software
	Generated Arduino IDE Script	An Arduino Uno script generated by Circuito.io (a circuit creation software) which assigns all correct ports and has test code to examine the status of each component	Circuito.io	https://www.circuito.io/ https://www.circuito.io/
	Fusion 360	The software used to create the non-electronic components of the project (laser cut base and 3D printed shell)	Autodesk	https://www.autodesk.com.au/products/fusion-360 https://www.autodesk.com.au/products/fusion-360
3D Models				
	Arduino Uno	A 3D model of the Arduino Uno with real world dimensions	Andrew Whitham	https://grabcad.com/library/arduino-uno-r3-1 https://grabcad.com/library/arduino-uno-r3-1
	Ultrasonic sensor (HC-SR04)	A 3D model of the HC-SR04 with real world dimensions	Dejan	https://thangs.com/designer/HowToMechatronics/3d-model/HC-SR04%20Ultrasonic%20Sensor%203D%20Model-48028

	DC Board (L298N)	A 3D model of the L298N with real world dimensions	Tijani Jouini	https://grabcad.com/library/l298n-17
	DC Motor	A 3D model of the DC Motor with real world dimensions	Moustafa Nabil	https://grabcad.com/library/mini-gear-dc-motor-6-v-yellow-1
	Colour Sensor	A 3D model of the TCS34725 with real world dimensions	Davor Granić	https://grabcad.com/library/tcs34725-rgb-sensor-2
	Rubber Wheel	A 3D model of the Rubber Wheels with real world dimensions	Amine Bouabid	https://grabcad.com/library/dc-motor-with-wheel-1
Machinery				
	3D Printer	The machine responsible for creating the shell.	Aitken College	Not Applicable
	Laser cutter	The machine which is responsible for precisely cutting the plastic base	Aitken College	Not Applicable
Components				
	Arduino Starter Kit	The kit which contains all electrical components to be used in the project	Arduino	Not Applicable

Risk Assessment

Risk Assessment

Equipment or Process	Hazard Identification	Consequence	Likelihood	Risk Rating	Engineering Measures	Procedures	Protective equipment
Battery Pack	Spark	Med	High	High	Male output used to combine wires	Make sure batteries are always out of the casing unless in use.	Not applicable

	Electrical Fire	High	Low	Med	Male output used to combine wires.	Keep the battery pack on the table in case of spark, it will not light the carpet.	Fire extinguisher, Fire Blanket
	Circuit Shortage/Burnt Fingers	High	High	High	Male output used to combine wires.	Keep fingers away from wires. Only load batteries into pack when circuit is connected.	Rubber gloves
Resistors							
	Circuit Shortage	Med	High	Med	Regulate current with correct resistors	Make sure to use appropriate resistors where needed	Not applicable
	Power Surge	Med	Low	Med	Use an Arduino Uno that has a maximum voltage of 5V and low current	Control and monitor current and voltage that is being put into the circuit.	Safety Glasses, Rubber Gloves (if working with high power)
3D Printer							
	Skin burns	High	Low	Med	Glass shield to cover 3D printer, stopping human access. 3D printer will not operate if door is open.	Students must be supervised when near 3D printer. Only operator must interact with 3D printer	Tap with cold water
	Physical Injury (Removing Excess Plastic)	Med	High	Med	Use necessary force with tools once 3D printed object is clamped to table. Use blunt chisel	Student must scrape away from themselves to avoid injury.	Safety Glasses, Pliers, Table Clamp, Chisel, Rubber Gloves
Laser Cutter							
	Fire	High	High	High	Glass shield to cover laser cutter when in use. Keep area clean and free of debris and flammable material	Operator must always be supervising the laser cutter when in use. The shield should be down	Properly maintained fire extinguisher should be present

	Eye Damage	High	High	High	Glass shield diffuses the exposure which could cause severe eye damage	Operator must not look directly at the laser	Safety Glasses can diffuse the exposure
--	------------	------	------	------	--	--	---

Diagnostic Testing

Testing Table

Test/calculation	Purpose of the test/calculation and procedural steps.
<p>Buzzer Functionality</p> <p><u>Video of result:</u> Buzzer v1.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the buzzer's sound output is working properly</p> <p><u>Equipment required:</u> The Arduino Uno with its components and breadboard, and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect all components to the breadboard. 2. Connect Arduino Uno to the computer. 3. Run test code. 4. Type "1" in the console <p><u>Results:</u> The buzzer was silent when the code was run.</p> <p><u>Review:</u> From this point, as there are no code errors, I will firstly replace the buzzer. If that still does not work, I will examine the connections of the circuit.</p>

<p>Buzzer Functionality v2</p> <p><u>Video of result:</u> Buzzer v2.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the buzzer's sound output is working properly</p> <p><u>Equipment required:</u> The Arduino Uno with its components and breadboard, and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect all components to the breadboard. 2. Connect Arduino Uno to the computer. 3. Change the Arduino port from 3 to 2. 4. Run test code. 5. Type "1" in the console <p><u>Results:</u> The buzzer sounded clicks at 0.5 second intervals.</p> <p><u>Review:</u> From this point, as there are no component errors regarding the buzzer, I will continue to test the other components.</p>
<p>HC-SR04 Functionality (Ultrasonic Sensor)</p> <p><u>Video of result:</u> HC-SR04 v1.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the distance reading output from the HC-SR04 is correct</p> <p><u>Equipment required:</u> The Arduino Uno with its components and breadboard, and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect all components to the breadboard. 2. Connect Arduino Uno to the computer. 3. Place a ruler in front of the sensor as a measurement indicator. 4. Run test code. 5. Type "3" in the console <p><u>Results:</u> The HC-SR04 outputted the correct values</p> <p><u>Review:</u></p>

	<p>From this point, as there are no component errors regarding the HC-SR04, I will continue to test the other components.</p>
<p>RGB LED Functionality</p> <p><u>Video of result:</u> RGB LED v1.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the LED can turn on as well as change its colour output.</p> <p><u>Equipment required:</u> The Arduino Uno with its components and breadboard, and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect all components to the breadboard. 2. Connect Arduino Uno to the computer. 3. Run test code. 4. Type "4" in the console <p><u>Results:</u> The RGB LED did not output a light</p> <p><u>Review:</u> From this point, as there are no code errors, I will firstly replace the LED. If that still does not work, I will examine the connections of the circuit.</p>
<p>RGB LED Functionality v2</p> <p><u>Video of result:</u> RGB LED v2.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the LED can turn on as well as change its colour output.</p> <p><u>Equipment required:</u> The Arduino Uno with its components and breadboard, and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect all components to the breadboard. 2. Connect Arduino Uno to the computer. 3. Flip the RGB LED pins and connect the port 11 wire to the parallel breadboard row. 4. Run test code. 5. Type "4" in the console <p><u>Results:</u> The RGB LED turned on (without console command) and cycles through green and blue</p> <p><u>Review:</u> From this point, as there are no component errors regarding the RGB LED, I will continue to test the other components.</p>

<p>Button Functionality</p> <p><u>Video of result:</u> Buttons v1.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the button will output a signal when it is pressed.</p> <p><u>Equipment required:</u> The Arduino Uno with its components and breadboard, and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 6. Connect all components to the breadboard. 7. Connect Arduino Uno to the computer. 8. Run test code. 9. Type "5" in the console (Button 1) 10. Type "6" in the console (Button 2) 11. Type "7" in the console (Button 3) <p><u>Results:</u> All buttons returned a '1' in the serial monitor. Note: a '0' indicates the button is not being pressed</p> <p><u>Review:</u> From this point, as there are no component errors regarding the buttons, I will continue to test the other components.</p>
<p>L298N Functionality</p> <p><u>Video of result:</u> L298N v1.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the L298N will spin the DC Motors at a given speed and direction. The use of an L298N is required for the direction to be controlled. By using PWM ports on the Arduino Uno, the speed of the DC Motors are able to be controlled in one direction however, the L298N chip has a H gate which allows for a 2 direction spinning.</p> <p><u>Equipment required:</u> The Arduino Uno with its components and breadboard, L298N with DC motors and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect all components to the breadboard. 2. Connect L298N to Arduino Uno PWM ports. 3. Connect Arduino Uno to the computer. 4. Run test code. 5. Type "2" in the console <p><u>Results:</u></p>

Commented [IW1]: explain the need for using the L298 Board

	<p>Right DC motor (connected to Motor B ports) started turning first at which Motor B followed shortly after. Also, it was noted that the rotation speed of the DC motors was inconsistent.</p> <p><u>Review:</u> From this point, as the motors are spinning at a low performance, I will try to incorporate another power supply as it seems that the Arduino Uno cannot provide enough power to both the 5V and VMS ports.</p>
<p>L298N Functionality v2</p> <p><u>Video of result:</u> L298N v2.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the L298N will spin the DC Motors at a given speed.</p> <p><u>Equipment required:</u> The Arduino Uno with its components and breadboard, L298N with DC motors, Power Supply Module (PSM) and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 6. Connect all components to the breadboard. 7. Connect L298N to Arduino Uno PWM ports. 8. Connect PSM to L298N VMS port and plug 9V Battery into component. 9. Connect Arduino Uno to the computer. 10. Run test code. 11. Type "2" in the console <p><u>Results:</u> Both DC motors started spinning at the same time with the same speed.</p> <p><u>Review:</u> From this point, as there are no component errors regarding the L298N and its DC motors, I will continue to test the other components.</p>
<p>Pinboard Functionality</p> <p><u>Video of result:</u> Pinboard Testing.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the pinboard has successfully replaced the breadboard</p> <p><u>Equipment required:</u> The Arduino Uno with its components and pinboard, L298N with DC motors, Power Supply Module (PSM) and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect L298N to Arduino Uno PWM ports. 2. Connect PSM to L298N VMS port and plug 9V Battery into PSM. 3. Plug another 9V Battery into Arduino

	<ol style="list-style-type: none"> 4. Connect Arduino Uno to the computer. 5. Run test code. 6. Type “1” in the console (Buzzer) 7. Type “2” in the console (L298N Motor Driver) 8. Type “3” in the console (Ultrasonic Sensor – HC-SR04) 9. Type “5” in the console (Mini Pushbutton Switch #1) 10. Type “6” in the console (Mini Pushbutton Switch #2) 11. Type “7” in the console (Mini Pushbutton Switch #3) <p><u>Results:</u> All components were functioning as expected however, the DC motors did not spin. The L298N lit up (indicating the ports were being used to receive instructions from the Arduino) however, the DC motors did not spin, instead a beep-like sound was emitted for the duration that the DC motors were expected to spin.</p> <p><u>Review:</u> From this point, I will continue to test the L298N and its DC motors and figure out what the error is.</p>
<p>L298N Functionality</p> <p><u>Video of result:</u> L298N v3.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to figure out why the DC motors are not spinning</p> <p><u>Equipment required:</u> The Arduino Uno, pinboard, L298N with DC motors, Power Supply Module (PSM) and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect L298N to Arduino Uno PWM ports. 2. Connect PSM to L298N VMS port and plug 9V Battery into PSM. 3. Plug another 9V Battery into Arduino 4. Connect Arduino Uno to the computer. 5. Run test code. 6. Type “2” in the console (L298N Motor Driver) <p><u>Results:</u> When active, both the Motor A and Motor B ports on the L298N released a voltage value of $0.25 \pm 0.05V$.</p> <p><u>Review:</u> The L298N was working fine on the breadboard. From this point, I will visually inspect the connections on the pinboard to locate a potential ‘short-circuit’ then, I will continue to test the L298N and its DC motors and figure out what the error is.</p>

<p>L298N Functionality</p> <p><u>Video of result:</u> L298N v4.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to observe the L298N performance with a 9V power supply</p> <p><u>Equipment required:</u> The Arduino Uno, pinboard, L298N with DC motors, 9V Battery and the Circuito.io test code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect 9V Battery to L298N VMS port. 2. Connect Arduino Uno to the computer. 3. Run test code. 4. Type "2" in the console (L298N Motor Driver) <p><u>Results:</u> When active, both the Motor A and Motor B ports spin.</p> <p><u>Review:</u> From this point, as there are no component errors regarding the L298N and its DC motors, I will continue to test the other components.</p>
<p>TCS230 Functionality</p> <p><u>Video of result:</u> TCS230 v1.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to observe the colour reading nature of the TCS230</p> <p><u>Equipment required:</u> The entire construction (including TCS230 and Arduino Uno) and the TCS230 colour frequency code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect Arduino Uno to the computer. 2. Run test code. 3. Open serial monitor 4. Observe value outputs. <p><u>Results:</u> The red, green, and blue frequency values fluctuate, however there is a clear pattern, the lowest value is the color that is being picked up. This means the TCS230 and the code are working properly as colors can be distinguished by using the lowest value.</p> <p><u>Review:</u> From this point, as there are no component errors regarding the TCS230, I will continue to test the other components.</p>

<p>TCS230 Colour Detection Functionality</p> <p><u>Video of result:</u> TCS230 v2.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to ensure that the TCS230 would be able to accurately identify the colour that is directly under the sensor (referring to pre coded colours such as red, green and blue)</p> <p><u>Equipment required:</u> The entire construction (including TCS230 and Arduino Uno), white sheet with both red and blue tape and, the TCS230 colour sensing code.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect Arduino Uno to the computer. 2. Run TCS230 colour sensing code. 3. Open serial monitor 4. Place down white sheet 5. Place TCS230 over red tape 6. Observe value outputs. 7. Place TCS230 over blue tape 8. Observe value outputs. <p><u>Results:</u> When the TCS230 is placed above the red tape, the serial monitor returns “Red Detected”, showing that the correct color is detected. Once the TCS230 is no longer presented with red tape, both its value and output changes. This is also the same for the blue tape. However, there is a problem with the code knowing when it is not looking at the line anymore (specifically the blue line). Since the RGB value of white meets the criteria for blue, both colors (blue and white) are seen and read as blue. This creates problems for line differentiation as if set to blue, the TCS230 will still believe it is always following the line.</p> <p><u>Review:</u> A physical solution to this problem would be to cover the sides of the blue tape with red tape, however this is not cost efficient. From this point, as the TCS230 is unable to identify the colour/shade white, I will incorporate an if statement in the code that should be able to pick up on white values.</p>
<p>TCS230 Colour Detection Functionality</p> <p><u>Video of result:</u> TCS230 v3.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to ensure that the TCS230 would be able to accurately identify the colour that is directly under the sensor (referring to pre coded colours such as red, green, blue and white)</p> <p><u>Equipment required:</u> The entire construction (including TCS230 and Arduino Uno), white sheet with both red and blue tape and, the TCS230 colour sensing code.</p>

	<p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect Arduino Uno to the computer. 2. Run TCS230 colour sensing code. 3. Open serial monitor 4. Place down white sheet 5. Place TCS230 over red tape 6. Observe value outputs. 7. Place TCS230 over white space 8. Observe value outputs. <p><u>Results:</u> When the TCS230 is placed above the red tape, the serial monitor returns “Red Detected”, showing that the correct color is detected. Once the TCS230 is no longer presented with red tape, the serial monitor now returns “White Detected”.</p> <p><u>Review:</u> From this point, as there are no component errors regarding the TCS230, I will continue to develop my code to account for line following.</p>
<p>HC-SR04 Functionality</p> <p><u>Video of result:</u> HC-SR04 v2.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the distance recorded by the HC-SR04 is able to accurately alter the speed of the DC motors in real time.</p> <p><u>Equipment required:</u> The entire construction (including HC-SR04 and Arduino Uno).</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect Arduino Uno to the computer. 2. Run main code. 3. Open serial monitor 4. Move object towards HC-SR04 5. Observe value outputs. <p><u>Results:</u> By default, the value of speed is 200 as the distance is greater than 20. When the object (hand) moves closer to the sensor, despite there being code to change the speed based off increments of five, the speed is only changed to account for a distance level of 4. There are 5 levels of speed, 1 being close and speed = 0 and 5 where distance is far and speed =</p>


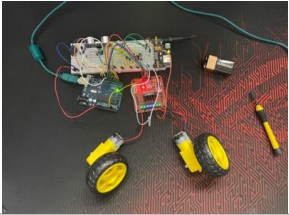
	<p>200.</p> <p><u>Review:</u> From this point, as the code is only reactive at 15 cm or higher, I will alter the code to incorporate all levels of distance.</p>
<p>HC-SR04 Functionality</p> <p><u>Video of result:</u> HC-SR04 v3.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the distance recorded by the HC-SR04 can accurately alter the speed of the DC motors in real time.</p> <p><u>Equipment required:</u> The entire construction (including HC-SR04 and Arduino Uno).</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect Arduino Uno to the computer. 2. Run main code. 3. Open serial monitor 4. Move object towards HC-SR04 5. Observe value outputs. <p><u>Results:</u> By default, the value of speed is 200 as the distance is greater than 20. When the object (hand) moves closer to the sensor, the DC motor speed changes in increments of 50, from 200 to 0. At less than or equal to 20cm, the speed is set to 150. At less than or equal to 15cm, the speed is set to 100. At less than or equal to 10cm, the speed is set to 50. Finally, at less than or equal to 5cm, the speed is set to 0.</p> <p><u>Review:</u> From this point, as the code is changing the speed variables based off the distance readings, I will move onto testing a different subsystem. Also noted, when the L298N was powered on, the speed of the DC motors did change in real time. This was not shown in the video as it was not needed; only the value of the speed variable is needed.</p>
<p>Buzzer Functionality with Colour Detection</p> <p><u>Video of result:</u> Buzzer v3.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the buzzer will sound when the colour sensor is sensing white and if the buzzer will stop buzzing once the TCS230 is sensing a line.</p> <p><u>Equipment required:</u> The entire construction (including Buzzer, TCS230 and Arduino Uno) and white sheet with both red and blue tape.</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect Arduino Uno to the computer. 2. Run main code.

	<ol style="list-style-type: none"> Place TCS230 over red tape Observe noise outputs. Place TCS230 over white paper Observe noise outputs. Place TCS230 over blue tape Observe noise outputs. <p><u>Results:</u> The buzzer was silent when the TCS230 was reading both the red and blue colored tape however, the buzzer let out a constant noise when it was presented with the RGB values of white.</p> <p><u>Review:</u> From this point, as the buzzer is sounding only when it sees white RGB values, I will move onto testing a different subsystem.</p>
<p>L298N Functionality with HC-SR04</p> <p><u>Video of result:</u> L298N v5.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the distance recorded by the HC-SR04 can accurately alter the speed of the DC motors in real time</p> <p><u>Equipment required:</u> The entire construction (including HC-SR04 and Arduino Uno).</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> Connect Arduino Uno to the computer. Run main code. Open serial monitor Move object towards HC-SR04 Observe value outputs. <p><u>Results:</u> When the red button is pressed the L298N spins the wheels at a speed of 200. However, the previously 'distance alters speed' system no longer works for the speed. The serial monitors output the speed variable and shows that it is actively changing based off the distance however, this is not mirrored in the actual spinning speed. It was also sound the repressing of the button did not turn off the motors as coded, it simply reruns the initial push button if statement. On top of that, the speed at which the motor spins are decided by what the current reading is as the button is pressed. This can be seen at the end where the hand is smothering the sensor, causing a speed of 0 which results in the DC motors to stop spinning as the button is pressed.</p> <p><u>Review:</u></p>

	<p>From this point, as the DC motor speed does not active change when it is triggered by the pressing of a button, I will continue to experiment with the arrangement of code to find why the executed operations are not updating the variables.</p>
<p>L298N Functionality with TCS230</p> <p><u>Video of result:</u> Evaluation Video 1.MOV</p>	<p><u>Purpose of test:</u> This test was conducted to see if the colour detected by the TCS230 can accurately alter the rotation of the DC motors in real time to guide the robot along the coloured path</p> <p><u>Equipment required:</u> The entire construction (including TCS230 and Arduino Uno).</p> <p><u>Procedural steps for the test:</u></p> <ol style="list-style-type: none"> 1. Connect Arduino Uno to the computer. 2. Run main code. 3. Place TCS230 sensor over red line 4. Press button 2 (red) 5. Observe physical outputs (wheel direction) 6. Reset the Arduino Uno 7. Place TCS230 sensor over blue colour 8. Press button 2 (red) 9. Observe physical outputs (wheel direction) <p><u>Results:</u> When the TCS230 is detecting the red line and button 2 is pressed (red), The L298N spins both DC motors forward at a speed of 200. However, once the robot ventures off the line, it still moves forward and does not execute the secondary while loop which allows it to spin left to find the line once again. Furthermore, the opposite was tested. When the TCS230 is detecting a color other than red and button 2 is pressed (red), The L298N spins both DC motors in opposite directions at a speed of 200 to spin left. However, once the robot eventually detects the red line again, it still spins left in a never-ending cycle and does not execute the primary while loop which allows it to move forward when the line is being detected.</p> <p><u>Review:</u> From this point, as the DC motor rotation does not actively change according to the colour detected by the TCS230, I will continue to experiment with the arrangement of code to find why the executed operations are not updating the variables that influence the direction of the DC motors</p>

Development Process

Journal Entries (Critical Thinking)

Image	Journal Entry
<p><u>Image of class result:</u></p> 	<p><u>Incorporation of the L298N Component in the System</u></p> <p><i>Monday 10th of July</i></p> <p>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 soldered onto the DC Motors for a more reliable connection</p>
<p><u>Image of class result:</u></p> 	<p><u>Incorporation of the L298N Component in the System</u></p> <p><i>Tuesday 11th of July</i></p> <p>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.</p>

Commented [IW2]: Can you explain why this improvement happened?

Image of class result:

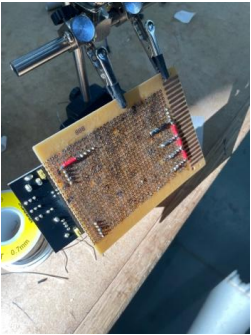


Transferring the Breadboard to a Pinboard

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 soldered.

Image of class result:

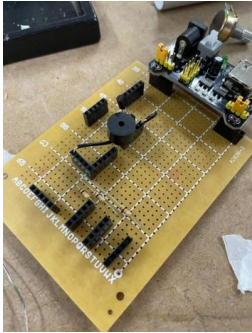


Transferring the Breadboard to a Pinboard

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 soldered 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:

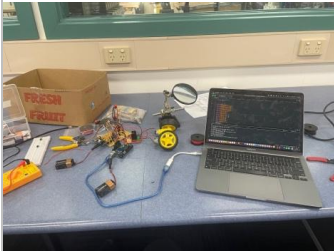


Transferring the Breadboard to a Pinboard

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:

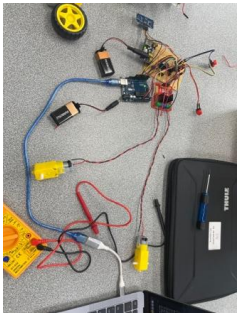


Pinboard Testing

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:

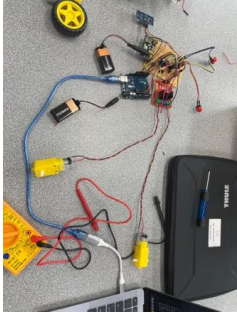


L298N Testing

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:

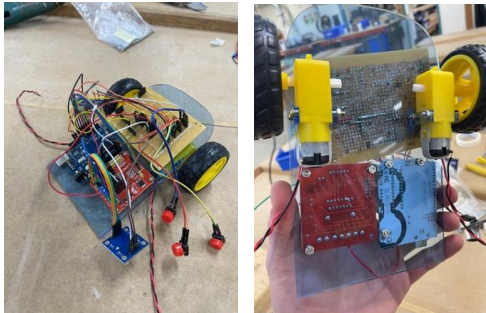


L298N Testing

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 L298N component. This allowed the L298N to work perfectly and also left the PSM redundant.

Image of class result:

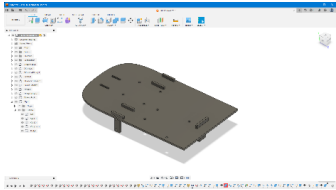


L298N Testing

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 fit 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:

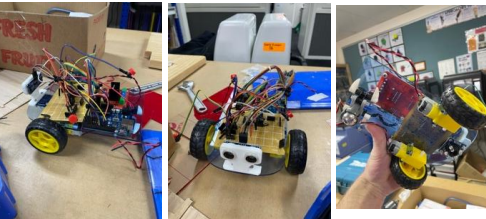


Fusion Model Changes

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:

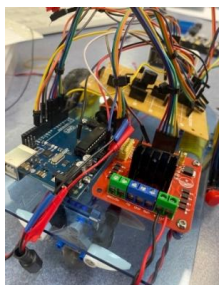


Solution Construction

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 loose. 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 [Thingiverse](#). I used 2 screws to secure both the stand and the TCS230 colour sensor.

Image of class result:



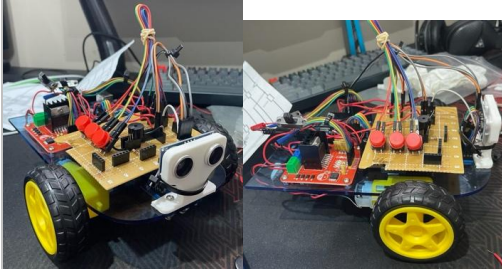
Solution Construction

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 [IW3]: A larger photo with Annotations would be better.

Image of class result:

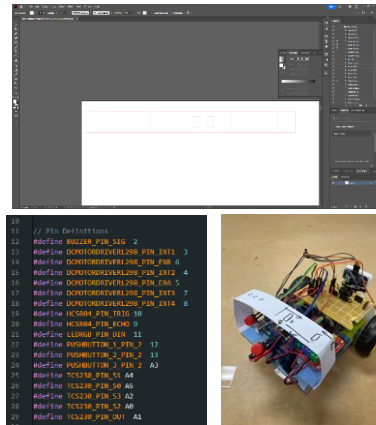


Solution Construction

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. Due 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:

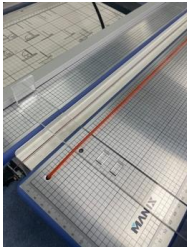


Button Stand Prototyping

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.

Image of class result:

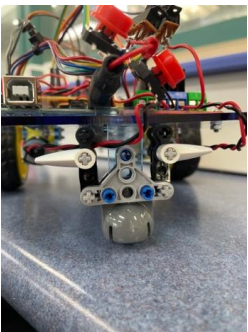


Button Stand Prototyping

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:

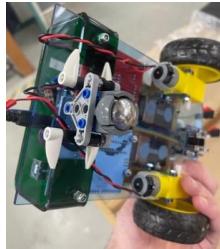
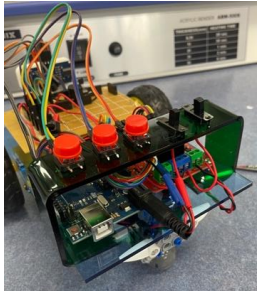


Button Stand Prototyping

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:

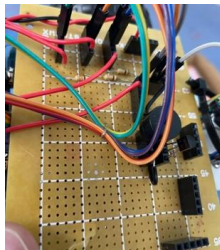
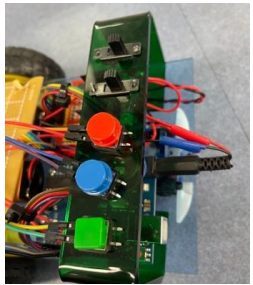


Button Stand Finalisation Version 1.0

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:

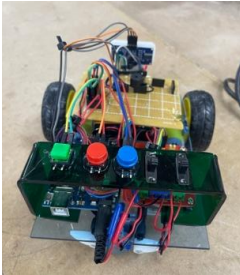
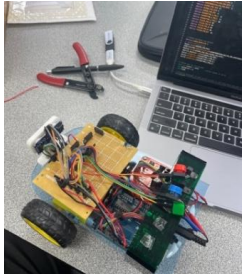


Button Stand Finalisation Version 1.1

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 tape 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:

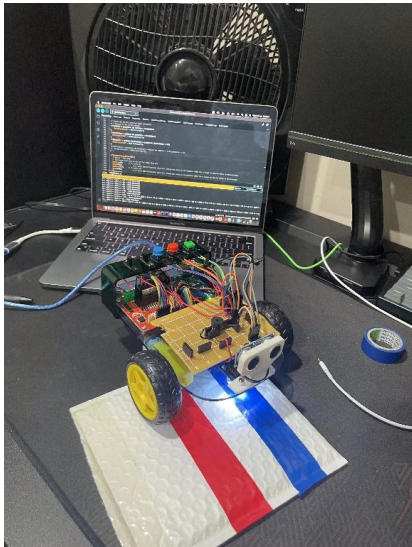


Button Stand Finalisation Version 1.2

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:

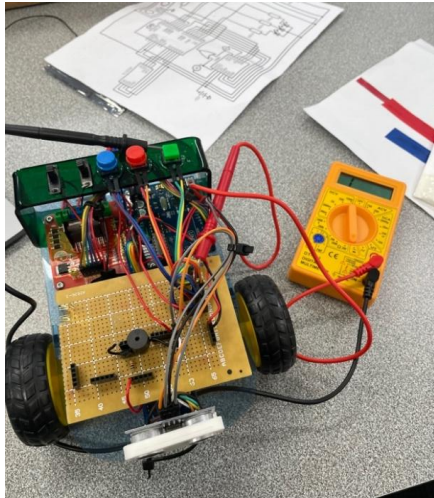


Colour Detection System

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-metre 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

```
226 //Buzzer will sound when it is not looking at the colour it is set to follow
227 if (colourDetected == 'W') {
228   buzzer.on(); // 2, turns on
229   tone(BUZZER_PIN_510, 1000); // Send 10Hz sound signal
230 }
231 else if (colourDetected == 'R' || colourDetected == 'G' || colourDetected == 'B') {
232   noTone(BUZZER_PIN_510); // Stop sound
233   buzzer.off(); // 2, turns off
234 }
235 }
236 }
```

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

```

147 // checks the current detected color and prints
148 // a message to the serial monitor
149 if (redColor > greenColor && redColor > blueColor) {
150     colourDetected = 'R';
151 }
152 if (greenColor > redColor && greenColor > blueColor) {
153     colourDetected = 'G';
154 }
155 if (blueColor > redColor && blueColor > greenColor && blueColor < 40) {
156     colourDetected = 'B';
157 }
158 //Code logic added to detect the colour white
159 if (blueColor > greenColor && blueColor > redColor && redColor > 1 && greenColor > 1) {
160     colourDetected = 'W';
161 }
162 //Ultrasonic sensor to update speed variable based on distance (cm)
163 if (HC_SR04dist < 20) {
164     motorSpeed = 0;
165 }
166 if (HC_SR04dist < 10) {
167     motorSpeed = 10;
168 }
169 if (HC_SR04dist < 5) {
170     motorSpeed = 0;
171 }
172 if (HC_SR04dist < 20) {
173     if (pushButton_3.isPressed() && buttonPressCounter == 0) {
174         buttonCounter = 1;
175         pushButton_3.release();
176         //Serial.println("Val: "); Serial.println(buttonCounter_Val);
177         digitalWrite(LED_BUILTIN, LOW);
178         digitalWrite(LED_BUILTIN, HIGH);
179     }
180     else if (pushButton_3.isPressed() && buttonPressCounter == 1) {
181         buttonPressCounter = 0;
182         digitalWrite(LED_BUILTIN, LOW);
183     }
184 }
185 if (pushButton_3.isPressed() && buttonPressCounter == 0) {
186     buttonPressCounter = 1;
187     relayRelay = 'R';
188     Serial.println("Val: "); Serial.println(buttonCounter_Val);
189     digitalWrite(LED_BUILTIN, LOW);
190     digitalWrite(LED_BUILTIN, HIGH);
191 }
192 else if (pushButton_3.isPressed() && buttonPressCounter == 1) {
193     buttonPressCounter = 0;
194     relayRelay = 'B';
195     digitalWrite(LED_BUILTIN, LOW);
196 }
197 if (pushButton_3.isPressed() && buttonPressCounter == 0) {
198     buttonCounter = 1;
199     pushButton_3.release();
200     //Serial.println("Val: "); Serial.println(buttonCounter_Val);
201     digitalWrite(LED_BUILTIN, LOW);
202     digitalWrite(LED_BUILTIN, HIGH);
203 }
204 else if (pushButton_3.isPressed() && buttonPressCounter == 1) {
205     buttonPressCounter = 0;
206     digitalWrite(LED_BUILTIN, LOW);
207 }
208 }

```

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 where 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:

[illegible]

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.

Evaluation

Does the robot follow a line?

For the robot to be able to follow a line, it first must be able to detect the coloured line and distinguish it from the floor. This involved calibrating the TCS230 to incorporate the line colour values. This system worked by having the Arduino Uno code compare two variables, the

colour that is currently being detected by the sensor and the colour that it is set to follow. The script compares these values to know when it is following the line and when the robot veers off the line. It was found that the highest accuracy came from a white floor as some darker surfaces could be grouped under the blue RGB values. For the robot to start following a line, a button press (1 of 3) will assign the variable and tell the Arduino Uno what colour should be followed. The line following component of the code is setup up under an if statement that contains two while loops. The if statements checks that a button is pressed and then either while loop (drive forward or turn) will be active depending on the coloured line status i.e., reading the correct colour or outside the line. The code is written with correct syntax and should work however, this is not the case. Once either while loop is initially triggered, the while loop stays locked and does not transition to the other loop as it simply ignores all code outside the active loop. A suggestion to overcome this error is to create a nested loop that has the variable updating code inside of the while loop. In the [Evaluation Video 1.MOV](#) video, based off the rotation of the DC motors, the robot is able to initially register whether the correct line is being detected or not. However, the reset button on the Arduino Uno must be pressed before switching to the other while loop as these loops are locked.

Does the robot safely operate within the vicinity of people and obstructions?

The code was successful in its ability to determine the DC motor speed based off the distance from the closest object. Relevant testing occurred in the test log record "L298N Functionality with HC-SR04". There were no errors in the creation of this system if the robot was told to drive from the void loop() though, further code development led to issues. It was found that if the code was running a while() loop, the Arduino Uno did not update the speed variable (which is read from the HC-SR04), resulting in the speed being static. A nested loop which contains the speed changing if statements is suggested as a solution to this problem, a piece of code cannot be ignored if it is present in the loop. However, this system does not prevent all collision accidents from occurring. Due to hardware limitations (such as a limited number of ports on the Arduino Uno) only one HC-SR04 was implemented. This means that the robot can only detect what is directly in front of the robot. In a high foot traffic area, the robot could potentially be trampled over. Further improvements such as visibility could be improved. A flag that is attached to the robot and is present at average eye level is recommended as an improvement. In the [Evaluation Video 2.MOV](#) video, the speed of the DC motor wheels are dependant on the distance of the hand towards the HC-SR04. The wheels spin at a speed of 200 normally however, as the hand gets closer, the speed slows down until it is not spinning anymore. Note, the code was altered to have the DC motors spin triggered in the void loop environment, rather than in a nested loop as this allowed for the variables to be actively updated. When the hand is directly over the HC-SR04, the component will read a distance of 0 and start spinning the motors at a speed of 200 (as seen in the video). This is due to distance related hardware limitations of the HC-SR04. The component is not able to send a signal past 300 centimetres, this means that if there is simply nothing in its path, it will return a distance of 0 and not move. To fix this, code was added that assigned a distance of 0 to a speed of 200. The robot will slow down to a stop before reaching an obstruction which means a reading of 0 should not apply to a situation where there is an obstruction as the robot will stop with at least 5 centimetres and the sensors must be purposefully covered. A

recommendation to this system is to have the code check if it was previously moving as this would allow the robot to know if it is driving or should still be stopped.

Is the robot able to transport objects?

The robot can transport objects utilising its drive system however, the created robot is a prototype that focuses on the drive system and its subsystems. This means while some objects could be placed on the robot and transported, there is not a designated object holder that has been implemented in the physical construction. However, this is easily implemented in the future as this feature can be considered when designing the body (case) for the robot as a second level of prototyping (a level that would be started once the internal systems were working). As this subsystem was not focused on, there are no recorded tests regarding this feature. In the [Evaluation Video 3.MOV](#) video, a small object (Listerine packet) is placed on a small empty area on the pinboard. As there is no security, the speed of the robot led to the object falling off. This means that the current prototype version of the robot is unable to transport objects.

Is the robot able to repeat itself whilst staying switched on?

As the code is not fully completed (errors still active), the robot must continuously be reset in order to test/activate another button. The code states that once a button is pressed, it triggers a while loop which is the overall driving and line detecting system. However, once the Arduino enters the while loop, it ignores the 'stop' code (outside the while loop) which stops both motors when the button is pressed a second time, resulting in an endless loop that must be switched off. Furthermore, there is currently no way for the robot to know that it has reached its destination (hospital room) as it is programmed to follow a line and once there is no line to follow (the end), the robot will be searching for that line. The simple user interface does not allow for a destination configuration and must purely rely on the line. An additional colour sensor could be placed in front of the first to check for 'no line' however, this will lead to problems when the line is turning. Another suggestion to improve this subsystem would be to implement a Near Field Communication (NFC) reader in the robot and NFC chips at the doors of the rooms which would feed into the reader and allow the robot to know which room it is at. However, there is not enough ports on the Arduino Uno (hardware limitation) and this system may be costly for a hospital. A simpler solution is to change the line colour and have the robot stop under the colour change conditions (rather than completely ignoring all colours) but then again, how would it be able to pass that room to move on to the next. In the [Evaluation Video 1.MOV](#) video, based off the rotation of the DC motors, the robot is able to initially register whether the correct line is being detected or not. However, the reset button on the Arduino Uno must be pressed before switching to the other while loop as these loops are locked. This means that the robot is not fully automatic and cannot operate on its own.

Arduino Code

TCS230 Calibration Code – Rui Santos

```
/*  
*****  
Rui Santos  
Complete project details at http://randomnerdtutorials.com  
*****/  
  
// TCS230 or TCS3200 pins wiring to Arduino  
#define S0 A2  
#define S1 A1  
#define S2 A4  
#define S3 A3  
#define sensorOut A5  
  
// Stores frequency read by the photodiodes  
int redFrequency = 0;  
int greenFrequency = 0;  
int blueFrequency = 0;  
  
void setup() {  
  // Setting the outputs  
  pinMode(S0, OUTPUT);  
  pinMode(S1, OUTPUT);  
  pinMode(S2, OUTPUT);  
  pinMode(S3, OUTPUT);  
  
  // Setting the sensorOut as an input  
  pinMode(sensorOut, INPUT);  
  
  // Setting frequency scaling to 20%  
  digitalWrite(S0,HIGH);  
  digitalWrite(S1,LOW);  
}
```



```
// Begins serial communication
Serial.begin(9600);
}
void loop() {
  // Setting RED (R) filtered photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,LOW);

  // Reading the output frequency
  redFrequency = pulseIn(sensorOut, LOW);

  // Printing the RED (R) value
  Serial.print("R = ");
  Serial.print(redFrequency);
  delay(100);

  // Setting GREEN (G) filtered photodiodes to be read
  digitalWrite(S2,HIGH);
  digitalWrite(S3,HIGH);

  // Reading the output frequency
  greenFrequency = pulseIn(sensorOut, LOW);

  // Printing the GREEN (G) value
  Serial.print(" G = ");
  Serial.print(greenFrequency);
  delay(100);

  // Setting BLUE (B) filtered photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,HIGH);
```

```
// Reading the output frequency
blueFrequency = pulseIn(sensorOut, LOW);

// Printing the BLUE (B) value
Serial.print(" B = ");
Serial.println(blueFrequency);
delay(100);
}
```

Line Following Code

Firmware (Main Code)

```
// Include Libraries
#include "Arduino.h"
#include "Buzzer.h"
#include "DCMDriverL298.h"
#include "NewPing.h"
#include "Button.h"

// Pin Definitions & Wire Colours
#define BUZZER_PIN_SIG 2 //Red
#define DCMOTORDRIVERL298_PIN_INT1 3 //Yellow
#define DCMOTORDRIVERL298_PIN_ENB 6 //Red
#define DCMOTORDRIVERL298_PIN_INT2 4 //Green
#define DCMOTORDRIVERL298_PIN_ENA 5 //Brown
#define DCMOTORDRIVERL298_PIN_INT3 7 //Blue
#define DCMOTORDRIVERL298_PIN_INT4 8 //Purple
#define HCSR04_PIN_TRIG 10 //Black
#define HCSR04_PIN_ECHO 9 //Grey
#define PUSHBUTTON_1_PIN_2 12 // Red
#define PUSHBUTTON_2_PIN_2 13 //Red
```

```
#define PUSHBUTTON_3_PIN_2 A0 //Red
#define TCS230_PIN_S1 A1 //Brown
#define TCS230_PIN_S0 A2 //Blue
#define TCS230_PIN_S3 A3 //Green
#define TCS230_PIN_S2 A4 //Yellow
#define TCS230_PIN_OUT A5 //Orange

// object initialization
Buzzer buzzer(BUZZER_PIN_SIG);
DCMDriverL298
dcMotorDriverL298(DCMOTORDRIVERL298_PIN_ENA,DCMOTORDRIVERL298_PIN_INT1,DCMOTORDRIVERL298_PIN_INT2,DCMOTORDRIVERL298_PIN_ENB,DCMOTORDRIVERL298_PIN_INT3,DCMOTORDRIVERL298_PIN_INT4);
NewPing hcsr04(HCSR04_PIN_TRIG,HCSR04_PIN_ECHO);
Button pushButton_1(PUSHBUTTON_1_PIN_2);
Button pushButton_2(PUSHBUTTON_2_PIN_2);
Button pushButton_3(PUSHBUTTON_3_PIN_2);

// Stores frequency read by the photodiodes
int redFrequency = 0;
int greenFrequency = 0;
int blueFrequency = 0;

// Stores the red. green and blue colors
int redColor = 0;
int greenColor = 0;
int blueColor = 0;

//Outlines motor speed as a variable since it will change
int dcmotorspeed = 200; // MAX 255
int turn_speed = 230; // MAX 255
int turn_delay = 10;
```

```
//Declares the colour detecting variables
int colourDetected = 0;
int colourToDetect = 0;
int detectionStatus = 0;

//Declares the number of buttons pressed
int buttonPressCounter = 0;

// Setup the essentials for your circuit to work. It runs first every time your circuit is powered with electricity.
void setup()
{

    pinMode(BUZZER_PIN_SIG, OUTPUT);

    pushButton_1.init();
    pushButton_2.init();
    pushButton_3.init();

    pinMode(TCS230_PIN_S0, OUTPUT);
    pinMode(TCS230_PIN_S1, OUTPUT);
    pinMode(TCS230_PIN_S2, OUTPUT);
    pinMode(TCS230_PIN_S3, OUTPUT);

    // Setting the sensorOut as an input
    pinMode(TCS230_PIN_OUT, INPUT);

    // Setting frequency scaling to 20%
    digitalWrite(TCS230_PIN_S0,HIGH);
    digitalWrite(TCS230_PIN_S1,LOW);

    // Begins serial communication
    Serial.begin(9600);
```

```
}
```

```
// Main logic of your circuit. It defines the interaction between the components you selected. After setup, it runs over and over again, in an eternal loop.
```

```
void loop()
```

```
{
```

```
//Declare Ultrasonic distance variable constantly
```

```
int hcsr04Dist = hcsr04.ping_cm();
```

```
//delay(10);
```

```
//Serial.print(F("Distance: ")); Serial.print(hcsr04Dist); Serial.println(F("[cm]"));
```

```
// Setting RED (R) filtered photodiodes to be read
```

```
digitalWrite(TCS230_PIN_S2,LOW);
```

```
digitalWrite(TCS230_PIN_S3,LOW);
```

```
// Reading the output frequency
```

```
redFrequency = pulseIn(TCS230_PIN_OUT, LOW);
```

```
// Remaping the value of the RED (R) frequency from 0 to 255
```

```
// You must replace with your own values. Here's an example:
```

```
// redColor = map(redFrequency, 70, 120, 255,0);
```

```
redColor = map(redFrequency, 26, 136, 255,0);
```

```
// Printing the RED (R) value
```

```
//Serial.print("R = ");
```

```
//Serial.print(redColor);
```

```
delay(100);
```

```
// Setting GREEN (G) filtered photodiodes to be read
```

```
digitalWrite(TCS230_PIN_S2,HIGH);
```

```
digitalWrite(TCS230_PIN_S3,HIGH);
```

```
// Reading the output frequency
```

```
greenFrequency = pulseIn(TCS230_PIN_OUT, LOW);
```

```
// Remapping the value of the GREEN (G) frequency from 0 to 255
// You must replace with your own values. Here's an example:
// greenColor = map(greenFrequency, 100, 199, 255, 0);
greenColor = map(greenFrequency, 46, 108, 255, 0);

// Printing the GREEN (G) value
//Serial.print(" G = ");
//Serial.print(greenColor);
delay(100);

// Setting BLUE (B) filtered photodiodes to be read
digitalWrite(TCS230_PIN_S2,LOW);
digitalWrite(TCS230_PIN_S3,HIGH);

// Reading the output frequency
blueFrequency = pulseIn(TCS230_PIN_OUT, LOW);
// Remapping the value of the BLUE (B) frequency from 0 to 255
// You must replace with your own values. Here's an example:
// blueColor = map(blueFrequency, 38, 84, 255, 0);
blueColor = map(blueFrequency, 62, 156, 255, 0);

// Printing the BLUE (B) value
//Serial.print(" B = ");
//Serial.print(blueColor);
delay(100);

// Checks the current detected color and prints
// a message in the serial monitor
if(redColor > greenColor && redColor > blueColor){
  Serial.println(" - RED detected!");
  colourDetected = 2;
}
if(greenColor > redColor && greenColor > blueColor){
```

```
Serial.println(" - GREEN detected!");
colourDetected = 1;
}
if(blueColor > redColor && blueColor > greenColor && blueColor < 40){
Serial.println(" - BLUE detected!");
colourDetected = 3;
}
//Code Logic added to detect the colour white
if(blueColor > greenColor && blueColor > redColor && redColor > 1 && greenColor > 1){
Serial.println(" - WHITE detected!");
colourDetected = 4;
}

//Ultrasonic 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 > 1 && hcsr04Dist <= 5) {
dcmotorspeed = 0;
}
if (hcsr04Dist > 20 || hcsr04Dist < 1) {
dcmotorspeed = 200;
}

//Serial.print("Speed: "); Serial.print(dcmotorspeed); Serial.print(" "); Serial.print(buttonPressCounter);
//Serial.print(colourDetected);
//delay(10);
```

```
//Button press determines colour to follow
//Repressing the sma ebutton will stop the system
if (pushButton_1.onPress() && buttonPressCounter == 0) {
  detectionStatus = 1;
  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) {
  detectionStatus = 'F';
  buttonPressCounter = 0;
  dcMotorDriverL298.stopMotors();
}

if (pushButton_2.onPress() && buttonPressCounter == 0) {
  detectionStatus = 1;
  buttonPressCounter = 1;
  colourToDetect = 2;
  bool pushButton_2Val = pushButton_2.read();
  //Serial.print(F("Val: ")); Serial.println(pushButton_2Val);
}
else if (pushButton_2.onPress() && buttonPressCounter == 1) {
  //detectionStatus = 'F';
  buttonPressCounter = 0;
  //dcmotorspeed = 0;
  //dcMotorDriverL298.stopMotors();
}

if (pushButton_3.onPress() && buttonPressCounter == 0) {
  detectionStatus = 'T';
```



```
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) {
  detectionStatus = 'F';
  buttonPressCounter = 0;
  dcmotorspeed = 0;
  dcMotorDriverL298.stopMotors();
}

if (detectionStatus == 1) {
  while (colourDetected == colourToDetect) {
    //move forward
    //Serial.print("going forward");
    //delay(100);
    //Serial.println(colourDetected); Serial.print(" "); Serial.println(colourToDetect);
    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);
    if (colourDetected != colourToDetect) {
      break;
    }
  }
}
```

```
}

}

while (colourDetected != colourToDetect) {
  //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);

  if (colourDetected == colourToDetect) {
    break;
  }

  //turn right
  //Serial.println("turning left");
  //dcMotorDriverL298.setMotorA(dcmotorspeed, 0);
  //dcMotorDriverL298.setMotorB(dcmotorspeed, 0);

  //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);
  }
}

//Buzzer will sound when it is not looking at the colour it is set to follow
if (colourDetected == 4) {
  buzzer.on();    // 1. turns on
  tone(BUZZER_PIN_SIG, 1000); // Send 1KHz sound signal
}
else if (colourDetected == 1 || colourDetected == 2 || colourDetected == 3) {
  noTone(BUZZER_PIN_SIG); // Stop sound
  buzzer.off();    // 2. turns off.
}
}
}

```

Button.cpp (All Circuit.io code moving forward)

```

#include "Button.h"

#include <Arduino.h>

Button::Button(const int pin) : m_pin(pin)
{
}

void Button::init()
{
}

```

```
pinMode(m_pin, INPUT);
// set begin state
m_lastButtonState = read();
}

//read button state.
bool Button::read()
{
    return digitalRead(m_pin);
}

//In General:
//if button is not pushed function will return LOW (0).
//if it is pushed function will return HIGH (1).

bool Button::onChange()
{
    //read button state. '1' is pushed, '0' is not pushed.
    bool reading = read();
    // If the switch changed, due to noise or pressing:
    if (reading != m_lastButtonState) {
        // reset the debouncing timer
        m_lastDebounceTime = millis();
        m_pressFlag = 1;
    }

    if ((millis() - m_lastDebounceTime) > m_debounceDelay) {
        // whatever the reading is at, it's been there for longer
        // than the debounce delay, so take it as the actual current state:
```

```
// if the button state has changed:
if (m_pressFlag) { //reading != m_lastButtonState) {
    //update the buton state
    m_pressFlag = 0;

    // save the reading. Next time through the loop,
    m_lastButtonState = reading;
    return 1;

}
}
m_lastButtonState = reading;

return 0;
}

bool Button::onPress()
{
    //read button state. '1' is pushed, '0' is not pushed.
    bool reading = read();
    // If the switch changed, due to noise or pressing:
    if (reading == HIGH && m_lastButtonState == LOW) {
        // reset the debouncing timer
        m_lastDebounceTime = millis();
        m_pressFlag = 1;
    }

    if ((millis() - m_lastDebounceTime) > m_debounceDelay) {
        // whatever the reading is at, it's been there for longer
        // than the debounce delay, so take it as the actual current state:

        // if the button state has changed:
```

```
if (m_pressFlag) {
    // save the reading. Next time through the loop,
    m_pressFlag = 0;
    m_lastButtonState = reading;
    return 1;
}
}
m_lastButtonState = reading;

return 0;
}

bool Button::onRelease()
{
    //read button state. '1' is pushed, '0' is not pushed.
    bool reading = read();
    // If the switch changed, due to noise or pressing:
    if (reading == LOW && m_lastButtonState == HIGH) {
        // reset the debouncing timer
        m_lastDebounceTime = millis();
        m_pressFlag = 1;
    }

    if ((millis() - m_lastDebounceTime) > m_debounceDelay) {
        // whatever the reading is at, it's been there for longer
        // than the debounce delay, so take it as the actual current state:

        // if the button state has changed:
        if ( m_pressFlag) { //reading == LOW && m_lastButtonState == HIGH) {
            // save the reading. Next time through the loop,
            m_lastButtonState = reading;
```

```
    m_pressFlag = 0;
    return 1;

}
}
m_lastButtonState = reading;

return 0;

}
```

Button.h

```
#ifndef _BUTTON_H_
#define _BUTTON_H_

//button class:
class Button {
public:
    Button(const int pin);
    //initialize button instance
    void init();
    //Read button state - without debounce
    bool read();
    //return True on both button events, Press or Release
    bool onChange();
    //return True only on Press
    bool onPress();
    //return True only on Release
    bool onRelease();
};
```

```
private:
    const int m_pin;
    bool m_lastButtonState; //state variables
    long m_lastDebounceTime = 0; // the last time the output pin was toggled
    const int m_debounceDelay = 50; // the debounce time; increase if the output flickers
    bool m_pressFlag = 0;

};

#endif // _BUTTON_H_
```

Buzzer.cpp

```
#include "Buzzer.h"

Buzzer::Buzzer(const int pin) : Switchable(pin)
{
}

}
```

Buzzer.h

```
/** \addtogroup Buzzer
 * @{
 */

#ifndef BUZZER_H
#define BUZZER_H

#include "Switchable.h"
```



```
//solenoid driver class:
class Buzzer : public Switchable {
public:
    Buzzer(const int pin);
};

#endif //BUZZER_H
/** @*/
```

DCMDriverL298.cpp

```
#include <Arduino.h>
#include "DCMDriverL298.h"

/**
 * Construct a DC Motor Driver instance.<BR>
 * It constructs two DC Motor instances, motorL and motorR.<BR>
 * enA, enB - enable pins for motors. connected to PWM pin on Arduino board.<BR>
 * pinA1,pinA2,pinB1,pinB2 - direction pin of the motors. connected to digital pins on Arduino board.
 */
DCMDriverL298::DCMDriverL298(const int enA, const int pinA1, const int pinA2, const int enB, const int pinB1, const int pinB2) : m_enA(enA), m_enB(enB), m_pinA1(pinA1),
m_pinA2(pinA2), m_pinB1(pinB1), m_pinB2(pinB2)
{
    pinMode(m_pinA1,OUTPUT);
    pinMode(m_pinA2,OUTPUT);
    pinMode(m_pinB1,OUTPUT);
    pinMode(m_pinB2,OUTPUT);
    pinMode(m_enA,OUTPUT);
    pinMode(m_enB,OUTPUT);

    stopMotors();
}

/**Set DC motor A speed and direction.
```

```
*/  
void DCMDriverL298::setMotorA(int speed, bool dir)  
{  
    setMotor(m_enA, m_pinA1, m_pinA2, speed, dir);  
}  
  
/**Set DC motor B speed and direction.  
*/  
void DCMDriverL298::setMotorB(int speed, bool dir)  
{  
    setMotor(m_enB, m_pinB1, m_pinB2, speed, dir);  
}  
  
/**Stop DC motor A.  
*/  
void DCMDriverL298::stopMotorA()  
{  
    off(m_enA, m_pinA1, m_pinA2);  
}  
  
/**Stop DC motor B.  
*/  
void DCMDriverL298::stopMotorB()  
{  
    off(m_enB, m_pinB1, m_pinB2);  
}  
  
/**Stop both DC motors.  
*/  
void DCMDriverL298::stopMotors()  
{  
    stopMotorA();  
    stopMotorB();  
}  
  
/**Set DC motor speed and direction.  
*/
```

```
void DCMDriverL298::setMotor(int pinPWM, int pinDir1, int pinDir2, int speed, bool dir)
{
    analogWrite(pinPWM, speed);
    digitalWrite(pinDir1, !dir);
    digitalWrite(pinDir2, dir);
}

/**Turn off DC motor
 */
void DCMDriverL298::off(int pinPWM, int pinDir1, int pinDir2)
{
    analogWrite(pinPWM, 0);
    digitalWrite(pinDir1, LOW);
    digitalWrite(pinDir2, LOW);
}
```

DCMDriverL298.h

```
/** \addtogroup DCMotorWDriver
 * @{
 */

#ifndef _DCMDRIVERL298_H_
#define _DCMDRIVERL298_H_

//DcMotor driver class:
class DCMDriverL298
{
public:
    DCMDriverL298(const int enA, const int pinA1, const int pinA2, const int enB, const int pinB1, const int pinB2);
    void setMotorA(int speed, bool dir);
```

```

void setMotorB(int speed, bool dir);
void stopMotorA();
void stopMotorB();
void stopMotors();
void setMotor(int pinPWM, int pinDir1, int pinDir2, int speed, bool dir);
void off(int pinPWM, int pinDir1, int pinDir2);
private:
    const int m_enA, m_enB, m_pinA1, m_pinA2, m_pinB1, m_pinB2;
};

#endif // _DCMDRIVERL298_H_
/** @*/

```

NewPing.cpp

```

// -----
// Created by Tim Eckel - teckel@leethost.com
// Copyright 2016 License: GNU GPL v3 http://www.gnu.org/licenses/gpl.html
//
// See "NewPing.h" for purpose, syntax, version history, links, and more.
// -----

#include "NewPing.h"

/**
 * NewPing constructor.<BR>
 * trigger_pin - arduino pin connected to TRIG port on sensor.<BR>
 * echo_pin - arduino pin connected to ECHO port on sensor.<BR>
 * max_cm_distance - (optional) by default = 500. define max value to be returned
 */

NewPing::NewPing(uint8_t trigger_pin, uint8_t echo_pin, unsigned int max_cm_distance) {

```

```

#if DO_BITWISE == true
    _triggerBit = digitalPinToBitMask(trigger_pin); // Get the port register bitmask for the trigger pin.
    _echoBit = digitalPinToBitMask(echo_pin);      // Get the port register bitmask for the echo pin.

    _triggerOutput = portOutputRegister(digitalPinToPort(trigger_pin)); // Get the output port register for the trigger pin.
    _echoInput = portInputRegister(digitalPinToPort(echo_pin));      // Get the input port register for the echo pin.

    _triggerMode = (uint8_t *) portModeRegister(digitalPinToPort(trigger_pin)); // Get the port mode register for the trigger pin.
#else
    _triggerPin = trigger_pin;
    _echoPin = echo_pin;
#endif

    set_max_distance(max_cm_distance); // Call function to set the max sensor distance.

#if (defined (__arm__) && defined (TEENSYDUINO)) || DO_BITWISE != true
    pinMode(echo_pin, INPUT);    // Set echo pin to input (on Teensy 3.x (ARM), pins default to disabled, at least one pinMode() is needed for GPIO mode).
    pinMode(trigger_pin, OUTPUT); // Set trigger pin to output (on Teensy 3.x (ARM), pins default to disabled, at least one pinMode() is needed for GPIO mode).
#endif

#if defined (ARDUINO_AVR_YUN)
    pinMode(echo_pin, INPUT);    // Set echo pin to input for the Arduino Yun, not sure why it doesn't default this way.
#endif

#if ONE_PIN_ENABLED != true && DO_BITWISE == true
    *_triggerMode |= _triggerBit; // Set trigger pin to output.
#endif
}

/**
 * Standard ping methods.<BR>
 * Return time of flight

```

```

* max_cm_distance - (optional) by default = 500. define max value to be returned
*/

unsigned int NewPing::ping(unsigned int max_cm_distance) {
    if (max_cm_distance > 0) set_max_distance(max_cm_distance); // Call function to set a new max sensor distance.

    if (!ping_trigger()) return NO_ECHO; // Trigger a ping, if it returns false, return NO_ECHO to the calling function.

    #if URM37_ENABLED == true
        #if DO_BITWISE == true
            while (!(*_echoInput & _echoBit))          // Wait for the ping echo.
        #else
            while (!digitalRead(_echoPin))             // Wait for the ping echo.
        #endif
            if (micros() > _max_time) return NO_ECHO; // Stop the loop and return NO_ECHO (false) if we're beyond the set maximum distance.
    #else
        #if DO_BITWISE == true
            while (*_echoInput & _echoBit)             // Wait for the ping echo.
        #else
            while (digitalRead(_echoPin))             // Wait for the ping echo.
        #endif
            if (micros() > _max_time) return NO_ECHO; // Stop the loop and return NO_ECHO (false) if we're beyond the set maximum distance.
    #endif

    return (micros() - (_max_time - _maxEchoTime) - PING_OVERHEAD); // Calculate ping time, include overhead.
}

/*
* Read distance from sensor.<BR>
* Return distance in cm
*/

unsigned long NewPing::ping_cm(unsigned int max_cm_distance) {
    unsigned long echoTime = NewPing::ping(max_cm_distance); // Calls the ping method and returns with the ping echo distance in uS.

```

```

#if ROUNDING_ENABLED == false
    return (echoTime / US_ROUNDTRIP_CM); // Call the ping method and returns the distance in centimeters (no rounding).
#else
    return NewPingConvert(echoTime, US_ROUNDTRIP_CM); // Convert uS to centimeters.
#endif
}

/*
 * Read distance from sensor.<BR>
 * Return distance in inch
 */
unsigned long NewPing::ping_in(unsigned int max_cm_distance) {
    unsigned long echoTime = NewPing::ping(max_cm_distance); // Calls the ping method and returns with the ping echo distance in uS.
    #if ROUNDING_ENABLED == false
        return (echoTime / US_ROUNDTRIP_IN); // Call the ping method and returns the distance in inches (no rounding).
    #else
        return NewPingConvert(echoTime, US_ROUNDTRIP_IN); // Convert uS to inches.
    #endif
}

unsigned long NewPing::ping_median(uint8_t it, unsigned int max_cm_distance) {
    unsigned int uS[it], last;
    uint8_t j, i = 0;
    unsigned long t;
    uS[0] = NO_ECHO;

    while (i < it) {
        t = micros(); // Start ping timestamp.
        last = ping(max_cm_distance); // Send ping.

        if (last != NO_ECHO) { // Ping in range, include as part of median.

```

```

    if (i > 0) {          // Don't start sort till second ping.
        for (j = i; j > 0 && uS[j] - 1] < last; j--) // Insertion sort loop.
            uS[j] = uS[j - 1];          // Shift ping array to correct position for sort insertion.
        } else j = 0;          // First ping is sort starting point.
        uS[j] = last;          // Add last ping to array in sorted position.
        i++;                  // Move to next ping.
    } else it--;              // Ping out of range, skip and don't include as part of median.

    if (i < it && micros() - t < PING_MEDIAN_DELAY)
        delay((PING_MEDIAN_DELAY + t - micros()) / 1000); // Millisecond delay between pings.

}
return (uS[it >> 1]); // Return the ping distance median.
}

/**
 * Standard and timer interrupt ping method support functions (not called directly)
 */

boolean NewPing::ping_trigger() {
#ifdef DO_BITWISE == true
#ifdef ONE_PIN_ENABLED == true
    *_triggerMode |= _triggerBit; // Set trigger pin to output.
#endif
#endif

    *_triggerOutput &= ~_triggerBit; // Set the trigger pin low, should already be low, but this will make sure it is.
    delayMicroseconds(4);           // Wait for pin to go low.
    *_triggerOutput |= _triggerBit; // Set trigger pin high, this tells the sensor to send out a ping.
    delayMicroseconds(10);          // Wait long enough for the sensor to realize the trigger pin is high. Sensor specs say to wait 10uS.
    *_triggerOutput &= ~_triggerBit; // Set trigger pin back to low.

#ifdef ONE_PIN_ENABLED == true
    *_triggerMode &= ~_triggerBit; // Set trigger pin to input (when using one Arduino pin, this is technically setting the echo pin to input as both are tied to the same Arduino pin).

```



```

#endif

#if URM37_ENABLED == true
  if (!(*_echoInput & _echoBit)) return false;    // Previous ping hasn't finished, abort.
  _max_time = micros() + _maxEchoTime + MAX_SENSOR_DELAY; // Maximum time we'll wait for ping to start (most sensors are <450uS, the SRF06 can take up to
34,300uS!)
  while (*_echoInput & _echoBit)                  // Wait for ping to start.
    if (micros() > _max_time) return false;        // Took too long to start, abort.
#else
  if (*_echoInput & _echoBit) return false;        // Previous ping hasn't finished, abort.
  _max_time = micros() + _maxEchoTime + MAX_SENSOR_DELAY; // Maximum time we'll wait for ping to start (most sensors are <450uS, the SRF06 can take up to
34,300uS!)
  while (!(*_echoInput & _echoBit))                // Wait for ping to start.
    if (micros() > _max_time) return false;        // Took too long to start, abort.
#endif
#else
  #if ONE_PIN_ENABLED == true
    pinMode(_triggerPin, OUTPUT); // Set trigger pin to output.
  #endif

  digitalWrite(_triggerPin, LOW); // Set the trigger pin low, should already be low, but this will make sure it is.
  delayMicroseconds(4);           // Wait for pin to go low.
  digitalWrite(_triggerPin, HIGH); // Set trigger pin high, this tells the sensor to send out a ping.
  delayMicroseconds(10);          // Wait long enough for the sensor to realize the trigger pin is high. Sensor specs say to wait 10uS.
  digitalWrite(_triggerPin, LOW); // Set trigger pin back to low.

  #if ONE_PIN_ENABLED == true
    pinMode(_triggerPin, INPUT); // Set trigger pin to input (when using one Arduino pin, this is technically setting the echo pin to input as both are tied to the same Arduino pin).
  #endif

  #if URM37_ENABLED == true
    if (!digitalRead(_echoPin)) return false;    // Previous ping hasn't finished, abort.

```

```

    _max_time = micros() + _maxEchoTime + MAX_SENSOR_DELAY; // Maximum time we'll wait for ping to start (most sensors are <450uS, the SRF06 can take up to
34,300uS!)
    while (digitalRead(_echoPin))          // Wait for ping to start.
        if (micros() > _max_time) return false;    // Took too long to start, abort.
    #else
    if (digitalRead(_echoPin)) return false;        // Previous ping hasn't finished, abort.
    _max_time = micros() + _maxEchoTime + MAX_SENSOR_DELAY; // Maximum time we'll wait for ping to start (most sensors are <450uS, the SRF06 can take up to
34,300uS!)
    while (!digitalRead(_echoPin))          // Wait for ping to start.
        if (micros() > _max_time) return false;    // Took too long to start, abort.
    #endif
#endif

    _max_time = micros() + _maxEchoTime; // Ping started, set the time-out.
    return true;                          // Ping started successfully.
}

void NewPing::set_max_distance(unsigned int max_cm_distance) {
    #if ROUNDING_ENABLED == false
    _maxEchoTime = min(max_cm_distance + 1, (unsigned int) MAX_SENSOR_DISTANCE + 1) * US_ROUNDTRIP_CM; // Calculate the maximum distance in uS (no rounding).
    #else
    _maxEchoTime = min(max_cm_distance, (unsigned int) MAX_SENSOR_DISTANCE) * US_ROUNDTRIP_CM + (US_ROUNDTRIP_CM / 2); // Calculate the maximum
distance in uS.
    #endif
}

#if TIMER_ENABLED == true && DO_BITWISE == true

/*
 * Timer interrupt ping methods (won't work with non-AVR, ATmega128 and all ATtiny microcontrollers)
 */

```

```

void NewPing::ping_timer(void (*userFunc)(void), unsigned int max_cm_distance) {
    if (max_cm_distance > 0) set_max_distance(max_cm_distance); // Call function to set a new max sensor distance.

    if (!ping_trigger()) return; // Trigger a ping, if it returns false, return without starting the echo timer.
    timer_us(ECHO_TIMER_FREQ, userFunc); // Set ping echo timer check every ECHO_TIMER_FREQ uS.
}

boolean NewPing::check_timer() {
    if (micros() > _max_time) { // Outside the time-out limit.
        timer_stop(); // Disable timer interrupt
        return false; // Cancel ping timer.
    }

    #if URM37_ENABLED == false
        if (!(*_echoInput & _echoBit)) { // Ping echo received.
    #else
        if (*_echoInput & _echoBit) { // Ping echo received.
    #endif
        timer_stop(); // Disable timer interrupt
        ping_result = (micros() - (_max_time - _maxEchoTime) - PING_TIMER_OVERHEAD); // Calculate ping time including overhead.
        return true; // Return ping echo true.
    }

    return false; // Return false because there's no ping echo yet.
}

/*
 * Timer2/Timer4 interrupt methods (can be used for non-ultrasonic needs)
 */

```

```

// Variables used for timer functions
void (*intFunc)();
void (*intFunc2)();
unsigned long _ms_cnt_reset;
volatile unsigned long _ms_cnt;
#if defined(__arm__) && defined(TEENSYDUINO)
    IntervalTimer itimer;
#endif

void NewPing::timer_us(unsigned int frequency, void (*userFunc)(void)) {
    intFunc = userFunc; // User's function to call when there's a timer event.
    timer_setup();      // Configure the timer interrupt.

#if defined (__AVR_ATmega32U4__) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).
    OCR4C = min((frequency>>2) - 1, 255); // Every count is 4uS, so divide by 4 (bitwise shift right 2) subtract one, then make sure we don't go over 255 limit.
    TIMSK4 = (1<<TOIE4);                // Enable Timer4 interrupt.
#elif defined (__arm__) && defined (TEENSYDUINO) // Timer for Teensy 3.x
    itimer.begin(userFunc, frequency);      // Really simple on the Teensy 3.x, calls userFunc every 'frequency' uS.
#else
    OCR2A = min((frequency>>2) - 1, 255); // Every count is 4uS, so divide by 4 (bitwise shift right 2) subtract one, then make sure we don't go over 255 limit.
    TIMSK2 |= (1<<OCIE2A);                // Enable Timer2 interrupt.
#endif
}

void NewPing::timer_ms(unsigned long frequency, void (*userFunc)(void)) {
    intFunc = NewPing::timer_ms_cntdown; // Timer events are sent here once every ms till user's frequency is reached.
    intFunc2 = userFunc;                  // User's function to call when user's frequency is reached.
    _ms_cnt = _ms_cnt_reset = frequency; // Current ms counter and reset value.
    timer_setup();                        // Configure the timer interrupt.

#if defined (__AVR_ATmega32U4__) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).

```

```

OCR4C = 249;      // Every count is 4uS, so 1ms = 250 counts - 1.
TIMSK4 = (1<<TOIE4); // Enable Timer4 interrupt.
#elif defined (__arm__) && defined (TEENSYDUINO) // Timer for Teensy 3.x
  itimer.begin(NewPing::timer_ms_cntdown, 1000); // Set timer to 1ms (1000 uS).
#else
  OCR2A = 249;      // Every count is 4uS, so 1ms = 250 counts - 1.
  TIMSK2 |= (1<<OCIE2A); // Enable Timer2 interrupt.
#endif
}

void NewPing::timer_stop() { // Disable timer interrupt.
#if defined (__AVR_ATmega32U4__) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).
  TIMSK4 = 0;
#elif defined (__arm__) && defined (TEENSYDUINO) // Timer for Teensy 3.x
  itimer.end();
#else
  TIMSK2 &= ~(1<<OCIE2A);
#endif
}

/*
 * Timer2/Timer4 interrupt method support functions (not called directly)
 */

void NewPing::timer_setup() {
#if defined (__AVR_ATmega32U4__) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).
  timer_stop(); // Disable Timer4 interrupt.
  TCCR4A = TCCR4C = TCCR4D = TCCR4E = 0;
  TCCR4B = (1<<CS42) | (1<<CS41) | (1<<CS40) | (1<<PSR4); // Set Timer4 prescaler to 64 (4uS/count, 4uS-1020uS range).
  TIFR4 = (1<<TOV4);
  TCNT4 = 0; // Reset Timer4 counter.

```

```

#elif defined (__AVR_ATmega8__) || defined (__AVR_ATmega16__) || defined (__AVR_ATmega32__) || defined (__AVR_ATmega8535__) // Alternate timer commands for
certain microcontrollers.
    timer_stop(); // Disable Timer2 interrupt.
    ASSR &= ~(1<<AS2); // Set clock, not pin.
    TCCR2 = (1<<WGM21 | 1<<CS22); // Set Timer2 to CTC mode, prescaler to 64 (4uS/count, 4uS-1020uS range).
    TCNT2 = 0; // Reset Timer2 counter.
#elif defined (__arm__) && defined (TEENSYDUINO)
    timer_stop(); // Stop the timer.
#else
    timer_stop(); // Disable Timer2 interrupt.
    ASSR &= ~(1<<AS2); // Set clock, not pin.
    TCCR2A = (1<<WGM21); // Set Timer2 to CTC mode.
    TCCR2B = (1<<CS22); // Set Timer2 prescaler to 64 (4uS/count, 4uS-1020uS range).
    TCNT2 = 0; // Reset Timer2 counter.
#endif
}

void NewPing::timer_ms_cntdown() {
    if (!_ms_cnt--) { // Count down till we reach zero.
        intFunc2(); // Scheduled time reached, run the main timer event function.
        _ms_cnt = _ms_cnt_reset; // Reset the ms timer.
    }
}
/*
#if defined (__AVR_ATmega32U4__) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).
ISR(TIMER4_OVF_vect) {
    intFunc(); // Call wrapped function.
}
#elif defined (__AVR_ATmega8__) || defined (__AVR_ATmega16__) || defined (__AVR_ATmega32__) || defined (__AVR_ATmega8535__) // Alternate timer commands for
certain microcontrollers.
ISR(TIMER2_COMP_vect) {
    intFunc(); // Call wrapped function.
}
*/

```

```

}
#elif defined (__arm__)
// Do nothing...
#else
ISR(TIMER2_COMPA_vect) {
    intFunc(); // Call wrapped function.
}
#endif
*/

#endif

// -----
// Conversion methods (rounds result to nearest cm or inch).
// -----
/**
 * Return Distance in cm
 */
unsigned int NewPing::convert_cm(unsigned int echoTime) {
    #if ROUNDING_ENABLED == false
        return (echoTime / US_ROUNDTRIP_CM); // Convert uS to centimeters (no rounding).
    #else
        return NewPingConvert(echoTime, US_ROUNDTRIP_CM); // Convert uS to centimeters.
    #endif
}

/**
 * Return Distance in Inched
 */
unsigned int NewPing::convert_in(unsigned int echoTime) {
    #if ROUNDING_ENABLED == false

```

```
    return (echoTime / US_ROUNDTRIP_IN);          // Convert uS to inches (no rounding).
#else
    return NewPingConvert(echoTime, US_ROUNDTRIP_IN); // Convert uS to inches.
#endif
}
```

NewPing.h

```
// -----
// NewPing Library - v1.8 - 07/30/2016
//
// AUTHOR/LICENSE:
// Created by Tim Eckel - teckel@leethost.com
// Copyright 2016 License: GNU GPL v3 http://www.gnu.org/licenses/gpl.html
//
// LINKS:
// Project home: https://bitbucket.org/teckel12/arduino-new-ping/wiki/Home
// Blog: http://arduino.cc/forum/index.php/topic,106043.0.html
//
// DISCLAIMER:
// This software is furnished "as is", without technical support, and with no
// warranty, express or implied, as to its usefulness for any purpose.
//
// BACKGROUND:
// When I first received an ultrasonic sensor I was not happy with how poorly
// it worked. Quickly I realized the problem wasn't the sensor, it was the
// available ping and ultrasonic libraries causing the problem. The NewPing
// library totally fixes these problems, adds many new features, and breaths
// new life into these very affordable distance sensors.
//
// FEATURES:
// * Works with many different ultrasonic sensors: SR04, SRF05, SRF06, DYP-ME007, URM37 & Parallax PING))).
// * Compatible with the entire Arduino line-up (and clones), Teensy family (including $19 96Mhz 32 bit Teensy 3.2) and non-AVR microcontrollers.
```



```
// * Interface with all but the SRF06 sensor using only one Arduino pin.
// * Doesn't lag for a full second if no ping/echo is received.
// * Ping sensors consistently and reliably at up to 30 times per second.
// * Timer interrupt method for event-driven sketches.
// * Built-in digital filter method ping_median() for easy error correction.
// * Uses port registers for a faster pin interface and smaller code size.
// * Allows you to set a maximum distance where pings beyond that distance are read as no ping "clear".
// * Ease of using multiple sensors (example sketch with 15 sensors).
// * More accurate distance calculation (cm, inches & uS).
// * Doesn't use pulseIn, which is slow and gives incorrect results with some ultrasonic sensor models.
// * Actively developed with features being added and bugs/issues addressed.
//
// CONSTRUCTOR:
//   NewPing(sonar(trigger_pin, echo_pin [, max_cm_distance])
//   trigger_pin & echo_pin - Arduino pins connected to sensor trigger and echo.
//   NOTE: To use the same Arduino pin for trigger and echo, specify the same pin for both values.
//   max_cm_distance - [Optional] Maximum distance you wish to sense. Default=500cm.
//
// METHODS:
//   sonar.ping([max_cm_distance]) - Send a ping and get the echo time (in microseconds) as a result. [max_cm_distance] allows you to optionally set a new max distance.
//   sonar.ping_in([max_cm_distance]) - Send a ping and get the distance in whole inches. [max_cm_distance] allows you to optionally set a new max distance.
//   sonar.ping_cm([max_cm_distance]) - Send a ping and get the distance in whole centimeters. [max_cm_distance] allows you to optionally set a new max distance.
//   sonar.ping_median(iterations [, max_cm_distance]) - Do multiple pings (default=5), discard out of range pings and return median in microseconds. [max_cm_distance]
allows you to optionally set a new max distance.
//   NewPing::convert_in(echoTime) - Convert echoTime from microseconds to inches (rounds to nearest inch).
//   NewPing::convert_cm(echoTime) - Convert echoTime from microseconds to centimeters (rounds to nearest cm).
//   sonar.ping_timer(function [, max_cm_distance]) - Send a ping and call function to test if ping is complete. [max_cm_distance] allows you to optionally set a new max
distance.
//   sonar.check_timer() - Check if ping has returned within the set distance limit.
//   NewPing::timer_us(frequency, function) - Call function every frequency microseconds.
//   NewPing::timer_ms(frequency, function) - Call function every frequency milliseconds.
//   NewPing::timer_stop() - Stop the timer.
//
```

```
// HISTORY:
// 07/30/2016 v1.8 - Added support for non-AVR microcontrollers. For non-AVR
// microcontrollers, advanced ping_timer() timer methods are disabled due to
// inconsistencies or no support at all between platforms. However, standard
// ping methods are all supported. Added new optional variable to ping(),
// ping_in(), ping_cm(), ping_median(), and ping_timer() methods which allows
// you to set a new maximum distance for each ping. Added support for the
// ATmega16, ATmega32 and ATmega8535 microcontrollers. Changed convert_cm()
// and convert_in() methods to static members. You can now call them without
// an object. For example: cm = NewPing::convert_cm(distance);
//
// 09/29/2015 v1.7 - Removed support for the Arduino Due and Zero because
// they're both 3.3 volt boards and are not 5 volt tolerant while the HC-SR04
// is a 5 volt sensor. Also, the Due and Zero don't support pin manipulation
// compatibility via port registers which can be done (see the Teensy 3.2).
//
// 06/17/2014 v1.6 - Corrected delay between pings when using ping_median()
// method. Added support for the URM37 sensor (must change URM37_ENABLED from
// false to true). Added support for Arduino microcontrollers like the $20
// 32 bit ARM Cortex-M4 based Teensy 3.2. Added automatic support for the
// Atmel ATtiny family of microcontrollers. Added timer support for the
// ATmega8 microcontroller. Rounding disabled by default, reduces compiled
// code size (can be turned on with ROUNDING_ENABLED switch). Added
// TIMER_ENABLED switch to get around compile-time "__vector_7" errors when
// using the Tone library, or you can use the toneAC, NewTone or
// TimerFreeTone libraries: https://bitbucket.org/teckel12/arduino-toneac/
// Other speed and compiled size optimizations.
//
// 08/15/2012 v1.5 - Added ping_median() method which does a user specified
// number of pings (default=5) and returns the median ping in microseconds
// (out of range pings ignored). This is a very effective digital filter.
// Optimized for smaller compiled size (even smaller than sketches that
// don't use a library).
```

```
//
// 07/14/2012 v1.4 - Added support for the Parallax PING))) sensor. Interface
// with all but the SRF06 sensor using only one Arduino pin. You can also
// interface with the SRF06 using one pin if you install a 0.1uf capacitor
// on the trigger and echo pins of the sensor then tie the trigger pin to
// the Arduino pin (doesn't work with Teensy). To use the same Arduino pin
// for trigger and echo, specify the same pin for both values. Various bug
// fixes.
//
// 06/08/2012 v1.3 - Big feature addition, event-driven ping! Uses Timer2
// interrupt, so be mindful of PWM or timing conflicts messing with Timer2
// may cause (namely PWM on pins 3 & 11 on Arduino, PWM on pins 9 and 10 on
// Mega, and Tone library). Simple to use timer interrupt functions you can
// use in your sketches totally unrelated to ultrasonic sensors (don't use if
// you're also using NewPing's ping_timer because both use Timer2 interrupts).
// Loop counting ping method deleted in favor of timing ping method after
// inconsistent results kept surfacing with the loop timing ping method.
// Conversion to cm and inches now rounds to the nearest cm or inch. Code
// optimized to save program space and fixed a couple minor bugs here and
// there. Many new comments added as well as line spacing to group code
// sections for better source readability.
//
// 05/25/2012 v1.2 - Lots of code clean-up thanks to Arduino Forum members.
// Rebuilt the ping timing code from scratch, ditched the pulseIn code as it
// doesn't give correct results (at least with ping sensors). The NewPing
// library is now VERY accurate and the code was simplified as a bonus.
// Smaller and faster code as well. Fixed some issues with very close ping
// results when converting to inches. All functions now return 0 only when
// there's no ping echo (out of range) and a positive value for a successful
// ping. This can effectively be used to detect if something is out of range
// or in-range and at what distance. Now compatible with Arduino 0023.
//
// 05/16/2012 v1.1 - Changed all I/O functions to use low-level port registers
```

```
// for ultra-fast and lean code (saves from 174 to 394 bytes). Tested on both
// the Arduino Uno and Teensy 2.0 but should work on all Arduino-based
// platforms because it calls standard functions to retrieve port registers
// and bit masks. Also made a couple minor fixes to defines.
//
// 05/15/2012 v1.0 - Initial release.
// -----

/** \addtogroup HCSR-04
 * @{
 */

#ifndef NewPing_h
#define NewPing_h

#if defined (ARDUINO) && ARDUINO >= 100
#include <Arduino.h>
#else
#include <WProgram.h>
#include <pins_arduino.h>
#endif

#if defined (__AVR__)
#include <avr/io.h>
#include <avr/interrupt.h>
#endif

// Shouldn't need to change these values unless you have a specific need to do so.
#define MAX_SENSOR_DISTANCE 500 // Maximum sensor distance can be as high as 500cm, no reason to wait for ping longer than sound takes to travel this distance and
back. Default=500
#define US_ROUNDTRIP_CM 57 // Microseconds (uS) it takes sound to travel round-trip 1cm (2cm total), uses integer to save compiled code space. Default=57
#define US_ROUNDTRIP_IN 146 // Microseconds (uS) it takes sound to travel round-trip 1 inch (2 inches total), uses integer to save compiled code space. Default=146
#define ONE_PIN_ENABLED true // Set to "false" to disable one pin mode which saves around 14-26 bytes of binary size. Default=true
```

```

#define ROUNDING_ENABLED false // Set to "true" to enable distance rounding which also adds 64 bytes to binary size. Default=false
#define URM37_ENABLED false // Set to "true" to enable support for the URM37 sensor in PWM mode. Default=false
#define TIMER_ENABLED true // Set to "false" to disable the timer ISR (if getting "__vector_7" compile errors set this to false). Default=true

// Probably shouldn't change these values unless you really know what you're doing.
#define NO_ECHO 0 // Value returned if there's no ping echo within the specified MAX_SENSOR_DISTANCE or max_cm_distance. Default=0
#define MAX_SENSOR_DELAY 5800 // Maximum uS it takes for sensor to start the ping. Default=5800
#define ECHO_TIMER_FREQ 24 // Frequency to check for a ping echo (every 24uS is about 0.4cm accuracy). Default=24
#define PING_MEDIAN_DELAY 29000 // Microsecond delay between pings in the ping_median method. Default=29000
#define PING_OVERHEAD 5 // Ping overhead in microseconds (uS). Default=5
#define PING_TIMER_OVERHEAD 13 // Ping timer overhead in microseconds (uS). Default=13
#if URM37_ENABLED == true
  #undef US_ROUNDTRIP_CM
  #undef US_ROUNDTRIP_IN
  #define US_ROUNDTRIP_CM 50 // Every 50uS PWM signal is low indicates 1cm distance. Default=50
  #define US_ROUNDTRIP_IN 127 // If 50uS is 1cm, 1 inch would be 127uS (50 x 2.54 = 127). Default=127
#endif

// Conversion from uS to distance (round result to nearest cm or inch).
#define NewPingConvert(echoTime, conversionFactor) (max((((unsigned int)echoTime + conversionFactor / 2) / conversionFactor, (echoTime ? 1 : 0)))

// Detect non-AVR microcontrollers (Teensy 3.x, Arduino DUE, etc.) and don't use port registers or timer interrupts as required.
#if (defined (__arm__) && defined (TEENSYDUINO))
  #undef PING_OVERHEAD
  #define PING_OVERHEAD 1
  #undef PING_TIMER_OVERHEAD
  #define PING_TIMER_OVERHEAD 1
  #define DO_BITWISE true
#elif !defined (__AVR__)
  #undef PING_OVERHEAD
  #define PING_OVERHEAD 1
  #undef PING_TIMER_OVERHEAD
  #define PING_TIMER_OVERHEAD 1

```

```
#undef TIMER_ENABLED
#define TIMER_ENABLED false
#define DO_BITWISE false
#else
#define DO_BITWISE true
#endif

// Disable the timer interrupts when using ATmega128 and all ATtiny microcontrollers.
#if defined (__AVR_ATmega128__) || defined (__AVR_ATtiny24__) || defined (__AVR_ATtiny44__) || defined (__AVR_ATtiny84__) || defined (__AVR_ATtiny25__) || defined (__AVR_ATtiny45__) || defined (__AVR_ATtiny85__) || defined (__AVR_ATtiny261__) || defined (__AVR_ATtiny461__) || defined (__AVR_ATtiny861__) || defined (__AVR_ATtiny43U__)
#undef TIMER_ENABLED
#define TIMER_ENABLED false
#endif

// Define timers when using ATmega8, ATmega16, ATmega32 and ATmega8535 microcontrollers.
#if defined (__AVR_ATmega8__) || defined (__AVR_ATmega16__) || defined (__AVR_ATmega32__) || defined (__AVR_ATmega8535__)
#define OCR2A OCR2
#define TIMSK2 TIMSK
#define OCIE2A OCIE2
#endif

class NewPing {
public:
    NewPing(uint8_t trigger_pin, uint8_t echo_pin, unsigned int max_cm_distance = MAX_SENSOR_DISTANCE);
    unsigned int ping(unsigned int max_cm_distance = 0);
    unsigned long ping_cm(unsigned int max_cm_distance = 0);
    unsigned long ping_in(unsigned int max_cm_distance = 0);
    unsigned long ping_median(uint8_t it = 5, unsigned int max_cm_distance = 0);
    static unsigned int convert_cm(unsigned int echoTime);
    static unsigned int convert_in(unsigned int echoTime);
#if TIMER_ENABLED == true
    void ping_timer(void (*userFunc)(void), unsigned int max_cm_distance = 0);
#endif
};
```

```
boolean check_timer();
unsigned long ping_result;
static void timer_us(unsigned int frequency, void (*userFunc)(void));
static void timer_ms(unsigned long frequency, void (*userFunc)(void));
static void timer_stop();
#endif

private:
boolean ping_trigger();
void set_max_distance(unsigned int max_cm_distance);
#if TIMER_ENABLED == true
boolean ping_trigger_timer(unsigned int trigger_delay);
boolean ping_wait_timer();
static void timer_setup();
static void timer_ms_cntdown();
#endif
#if DO_BITWISE == true
uint8_t _triggerBit;
uint8_t _echoBit;
volatile uint8_t *_triggerOutput;
volatile uint8_t *_echoInput;
volatile uint8_t *_triggerMode;
#else
uint8_t _triggerPin;
uint8_t _echoPin;
#endif
unsigned int _maxEchoTime;
unsigned long _max_time;
};

#endif
/** @*/
```

Switchable.cpp

```
#include "Switchable.h"
#include <Arduino.h>

Switchable::Switchable(const int pin) : m_pin(pin)
{
    // Set pin as output
    pinMode(m_pin, OUTPUT);
    // Start state if off
    off();
}

//turn on:
void Switchable::on()
{
    digitalWrite(m_pin, 1); //high
    m_state = true;
}

//turn off:
void Switchable::off()
{
    digitalWrite(m_pin, 0); //low
    m_state = false;
}

void Switchable::toggle()
{
    digitalWrite(m_pin, !m_state); //low
    m_state = !m_state;
}

// dim pin
```



```
void Switchable::dim(int dimVal)
{
    analogWrite(m_pin, dimVal);
}

bool Switchable::getState()
{
    return m_state;
}

void Switchable::setState(bool state)
{
    digitalWrite(m_pin, state);
    m_state = state;
}
```

Switchable.h

```
#ifndef _SWITCHABLE_H_
#define _SWITCHABLE_H_

//Base class for output that can be switched on/off via single digital pin:
class Switchable
{
public:

    // Constructor accepts pin number for output
    Switchable(const int pin);

    // Turn pin on
    void on();
```

```
// Turn pin off
void off();

// Toggle pin
void toggle();

// dim pin
void dim(int dimVal);

// Get current state
bool getState();

// Set state with bool variable
void setState(bool state);

private:

const int m_pin; //output pin
bool m_state; //current state
};

#endif // _SWITCHABLE_H_
```