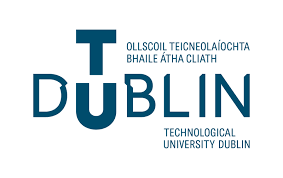
******

**SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING**

**BEng Tech (Ord) Electronics and Communications Engineering Level 7**

**Programme Code: (DT008)**

**<YEAR 3>**

**Name of Supervisor (Derek Gillmor)**

**April 20, 2020**

**Project Report**

**Student Name** Sajjad Ullah **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Project Title** Arduino Robot**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**DECLARATION**

This report is submitted in partial fulfilment of the requirements of the Degree in Electronic and Communications Engineering (DT008) of the Technological University Dublin.

I, the undersigned, declare that this report is entirely my own written work, except where otherwise accredited, and that it has not been submitted for a degree or other award to any other university or institution.

A close up of a logo

Description automatically generated

Signature of student: ………………………………….

Date: ………20/04/20 …………………………………

**Contents**

[Introduction 5](#_Toc38215844)

[Project Background 7](#_Toc38215845)

[H-Bridge L293D 7](#_Toc38215846)

[Pulse Width Modulation (PWM) 9](#_Toc38215847)

[Ultrasonic sensor HC-SR04 10](#_Toc38215848)

[Arduino uno 12](#_Toc38215849)

[ESP8266 Node MCU 13](#_Toc38215850)

[Technical description and construction 14](#_Toc38215851)

[Hardware 14](#_Toc38215852)

[Initial Arduino uno proto shield 14](#_Toc38215853)

[Final ESP8266 stripboard design 15](#_Toc38215854)

[Software 17](#_Toc38215855)

[MotorDirection class 17](#_Toc38215856)

[DistanceSensor class 20](#_Toc38215857)

[Test procedures and results 22](#_Toc38215858)

[Test 1 Single wheel with L293D simulation 22](#_Toc38215859)

[Test 2 Class behaviour test with L293D 23](#_Toc38215860)

[Test 3 Hardware validation of L293D 25](#_Toc38215861)

[Test 4 HC-SR04 sensors simulation 26](#_Toc38215862)

[Test 5 Hardware verification of sensors with L293D 27](#_Toc38215863)

[Test 6 ESP8266 webserver 28](#_Toc38215864)

[Test 7 ESP Final testing of all components 29](#_Toc38215865)

[Test 8 Modify code to work with serial monitor input 30](#_Toc38215866)

[Conclusion 31](#_Toc38215867)

[References 32](#_Toc38215868)

[Appendices 33](#_Toc38215869)

[Appendix A – Behaviours for MotorDirection Class 33](#_Toc38215870)

[Appendix B – Behaviours for DistanceSesnor Class 35](#_Toc38215871)

[Appendix C – ESP8266 Webserver code 36](#_Toc38215872)

[Appendix D – Program code Arduino ESP8266 version 38](#_Toc38215873)

[Appendix E – Program code Arduino uno version 41](#_Toc38215874)

**Table of Figures**

[Figure 1 Proposed working environment 5](#_Toc38215815)

[Figure 2 H-Bridge circuit [1] 7](#_Toc38215816)

[Figure 3 L293d pinout [2] 7](#_Toc38215817)

[Figure 4 Diagram for wheel testing 8](#_Toc38215818)

[Figure 5 Pulse Width Modulation graph [4] 9](#_Toc38215819)

[Figure 6 HC-SR04 pinout [6] 10](#_Toc38215820)

[Figure 7 Working principle of HC-SR04 [6] 10](#_Toc38215821)

[Figure 8 Diagram for sensor testing 11](#_Toc38215822)

[Figure 9 Arduino uno pinout [7] 12](#_Toc38215823)

[Figure 10 ESP8266 Mode MCU pinout [8] 13](#_Toc38215824)

[Figure 11 Proto shield Rev3 [9] 14](#_Toc38215825)

[Figure 12 Rev3 placed onto Uno(Left) and soldered Rev3 (Right) 14](#_Toc38215826)

[Figure 13 Hardware layout diagram 15](#_Toc38215827)

[Figure 14 Top view of final copper stripboard 16](#_Toc38215828)

[Figure 15 Bottom view of final copper stripboard 16](#_Toc38215829)

[Figure 16 MotorDirection class 19](#_Toc38215830)

[Figure 17 DistanceSensor class 21](#_Toc38215831)

[Figure 18 Testing l293d with one wheel 22](#_Toc38215832)

[Figure 19 Testing l293d with two wheels 23](#_Toc38215833)

[Figure 20 class test with l293d 24](#_Toc38215834)

[Figure 21 Hardware verification of l293d 25](#_Toc38215835)

[Figure 22 Simlation of HC-SR04 sesnors 26](#_Toc38215836)

[Figure 23 HC-SR04 Sensor test code 26](#_Toc38215837)

[Figure 24 HC-SR04 Sensor result 27](#_Toc38215838)

[Figure 25 Top and side view (Left & Middle) of setup and Testing (Right) 27](#_Toc38215839)

[Figure 26 ESP webserver(Left) and Serial monitor result (Right) 28](#_Toc38215840)

[Figure 27 Top view of finished robot 29](#_Toc38215841)

[Figure 28 Side view of finished robot 29](#_Toc38215842)

[Figure 29 Function for input from serial monitor 30](#_Toc38215843)

# Introduction

Technology permeates our daily lives from smartphones to computers, everything is connected via the internet but recently a new wave of devices are being connected. Devices such as smart watches and smart televisions are now becoming connected to the internet in what is commonly referred to as the Internet of Things (IoT). Inspiration for this final year project came from an IoT enabled device for use in a warehouse environment.

The decision was made to use Arduino hardware and software tools to create a robot with the overall aim of navigating an enclosed area such as a warehouse, for the purpose of reaching a desired destination and travelling back with the desired contents. Online services that will be used to achieve this are the Arduino Integrated Development Environment (IDE) and the Tinkercad design software.

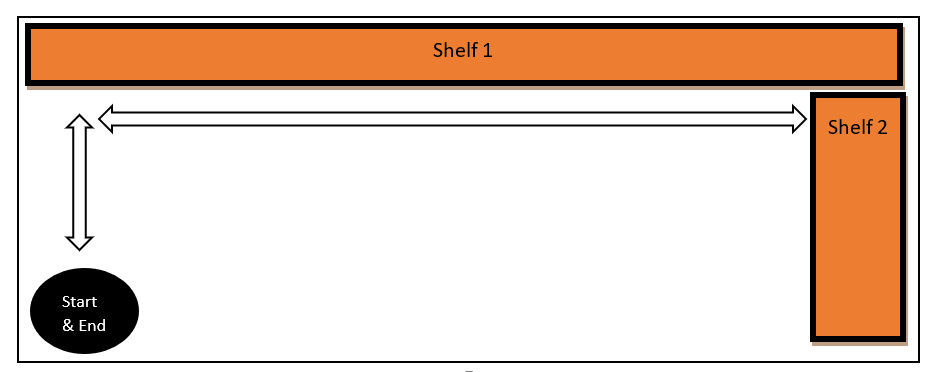


Figure 1 Proposed working environment

Dimensions of robot and environment are presented in Table 1.

Table 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Width** | **Height** | **Length** |
| Robot Dimensions | 13 cm | < 14cm | 17 cm |
| Environment Dimensions | 3 m | 10 cm | 2 m |

A grid system is to be used for identifying unique shelf sections. Using the Figure 1 layout as a basic implementation the robot will start and end at the same designated spot. Once a user has entered in the desired shelf coordinate it will begin using its ultrasonic sensors at regular intervals for the purpose of identifying its relative distance from walls, ensuring it maintains a parallel path along the shelfs until a time when it identifies a turn it must make. After it makes this turn it will continue along this pathway until the correct shelf location has been reached.

The overall goal of the project was then split into more manageable objectives:

* Produce a user defined data type for sensor behaviours e.g. *find distance* and *calculate cm*.
* Produce a user defined data type for wheel behaviours e.g. *forward*, *left*, *right* etc.
* Set up wireless communication for remote transmission of grid coordinates.
* Get wheels to rotate forwards, backwards and at different speeds.
* Use sensors to get correct distance measurements.
* Attain movement in a straight path.
* Attain 90 degree turns.

The solution proposed makes use of Arduino hardware e.g. Boards & shields and the Arduino IDE to create a two wheeled robot capable of navigating an enclosed area with the utilization of ultrasonic sensors to detect walls, shelfs and obstacle detection. Initial work on creating code for the wheels and verifying its intended behaviour will be done in Tinkercad where hardware simulation will be the quickest route to achieving the aims for that aspect of the project.

# Project Background

## H-Bridge L293D

The L293D chip was chosen because it contains H-Bridge circuits that can be used to provide the control mechanisms for the two wheels, this type of circuit allows for the ability to change the direction of wheel rotation without requiring any physical reconnections to be made and the chip allows the motors to be powered from an external source there-by removing the limitation of using the power supply provided by the Arduino board which at max can supply five volts.

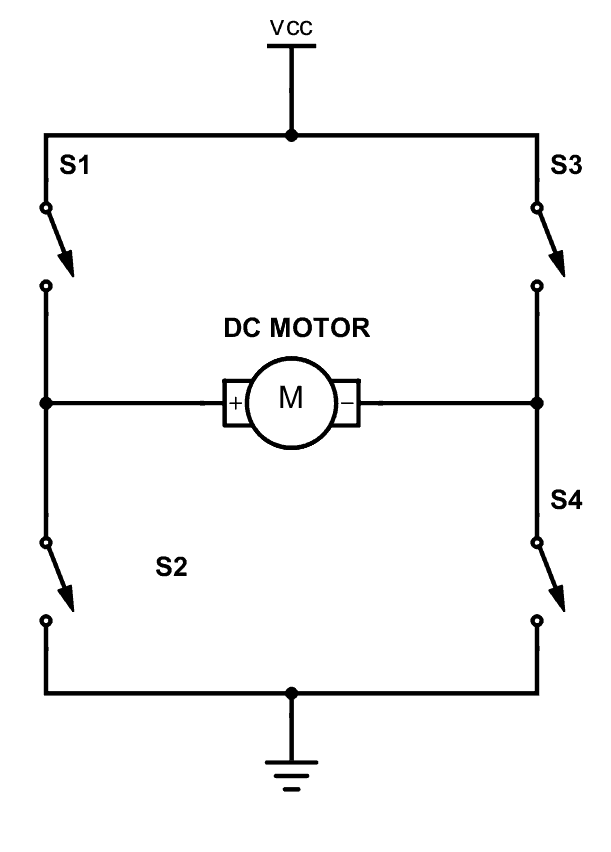


Figure 2 H-Bridge circuit [1]

In figure 2 the spin of the DC motor is dependent on which set of switches are closed so either switches S1 and S4 can be closed for spin in one direction or S3 and S2 can be closed for spin in the reverse direction, while powered by VCC.

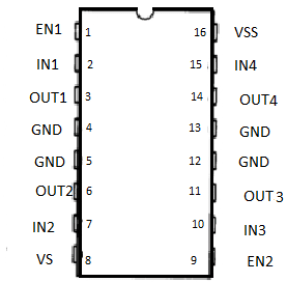


Figure 3 L293d pinout [2]

Information on the pinout of the l233d was used to create Table 2 with descriptions on the function of each pin from information obtained on *microcontrollerslab.com*.

Table 2 L293d pinout descriptions [3]

|  |  |
| --- | --- |
| **Pin-[Name]** | **Description of pin** |
| 1 -Enable Left | When High/Low turns on/off pins 3 & 6. |
| 2 -Input 1 | When High, current goes through output 1. |
| 3 -output 1 | Should be connected to a motor terminal |
| 6 -output 2 | Should be connected to a motor terminal |
| 7 -Input 2 | When High, current goes through output 2. |
| 8 -Vcc2 | This is the power supply to the motors |
| 16-Vcc1 | This is the power supply to the chip.[should be 5v] |
| 15-Input 4 | When High, current goes through output 4. |
| 14-Output 4 | Should be connected to a motor terminal |
| 11-output 3 | Should be connected to a motor terminal |
| 10-Input 3 | When High, current goes through output 3. |
| 9-Enable Right | When High/Low turns on/off pins 11 & 14. |
| 4,5,12,13-GND | Ground connector pins |

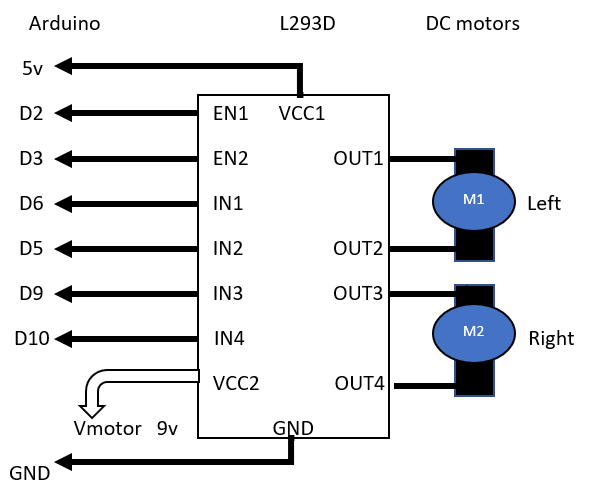


Figure 4 Diagram for wheel testing

Circuit testing of the l293d was done using the diagram shown in Figure 4, where the initial wheel behaviours were put to the test. By using Pulse Width Modulation pins (D5,D6,D9,D10) on the Arduino we could obtain the ability to adjust the power between a range of 0 – 9 volts and attain a variety of speeds at the wheels.

## Pulse Width Modulation (PWM)

By using PWM pins we can adjust the perceived voltage source to our wheels thereby removing the need to have multiple voltage sources or a voltage regulator for the wheels if we want to change the speed at which they rotate at. Pulse width modulation is a method for producing analogue results with digital techniques, the PWM method works by producing a square wave where the ratio of time on and time off can produce any voltage value between 5 volts and 0 volts on the Arduino PWM compatible output [4].

PWM will be used to enhance the operation of the l293d by allowing the speed of rotation to be controlled.

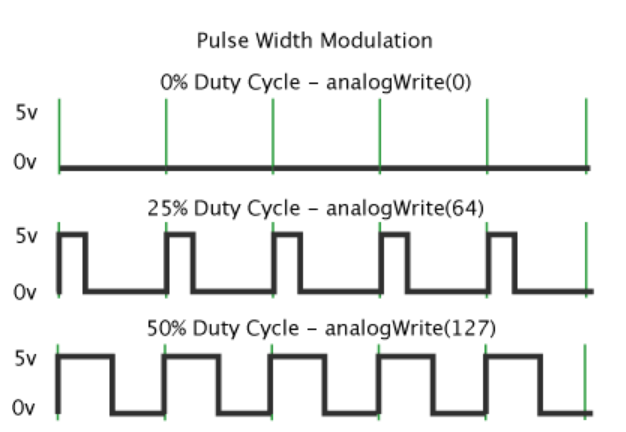


Figure 5 Pulse Width Modulation graph [4]

Figure 5 shows how PWM is implemented in Arduino, if we want only 25% of the power to be at the wheels then the desired duty cycle would be for the signal to only be HIGH for 25% of the period which will require an *analogueWrite(64)* command. This is calculated by:

, *where*

## Ultrasonic sensor HC-SR04

The sensor type HC-SR04 uses sound to produce distance measurements, its measuring angle is 15 degrees and was chosen for its ability to detect distances of up to 400cm [5] for which our requirement to measure the robot’s position within the small-scale environment is ideal.

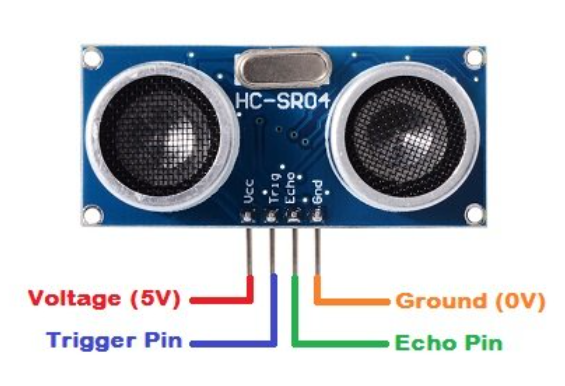


Figure 6 HC-SR04 pinout [6]

Pinout descriptions were obtained to better understand the functionality of each pin and how they would need to be used.

Table 3 HC-SR04 pinout descriptions [6]

|  |  |
| --- | --- |
| **Pin** | **Description** |
| VCC | 5v power for the device |
| Trig | Sends an echo signal |
| Echo | Detects the reflected echo signal |
| GND | Ground pin |

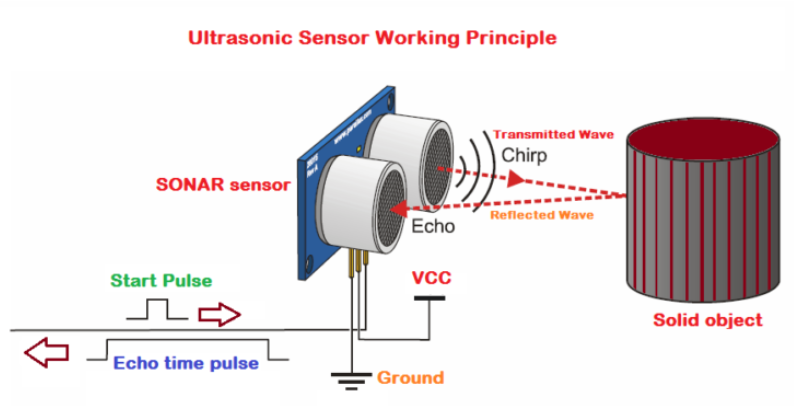


Figure 7 Working principle of HC-SR04 [6]

The sensor operates at 40 MHz with two main parts transmitter(Trig) and receiver(Echo), where the first is responsible for transmitting the wave out to an object for which the pin must be kept HIGH for 10 µs to initiate [6]. The next steps are listed below:

The waves that travel out at 40 MHz will be in 8 cycle sonic bursts.

These bursts are then detected by the receiver and can be converted into a distance measurement.

Conversion is possible because we know the speed at which the waves travel is the speed of sound and the time it takes to be received is the echo time so multiplying these values (due to the relationship between speed, distance and time) give the round trip distance but we divide by two to get the object distance.

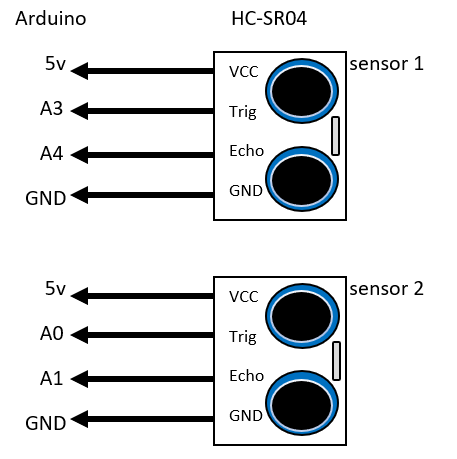


Figure 8 Diagram for sensor testing

The diagram shown in Figure 8 where analogue pins were used for testing the sensors and verifying that correct distance measurements were being reported before moving these pins in the final design to suitable locations.

## Arduino uno

The Arduino uno was the initial development board for the project due to having past familiarity with the board during past academic years and already having a good understanding of the pin layout.

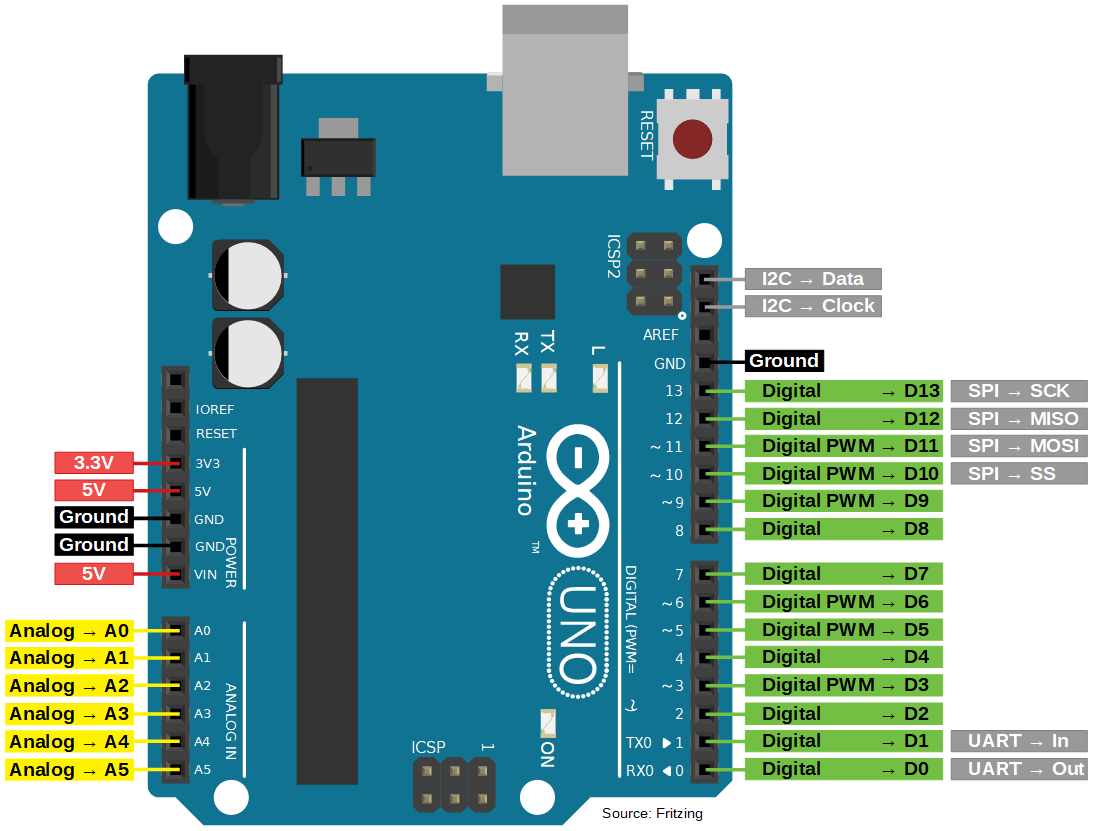


Figure 9 Arduino uno pinout [7]

The uno board contains 14 digital pins and 6 analogue pins with 6 capable of PWM output, this was more than enough for the hardware testing and development of the class data types that would be used for the wheels and sensors.

## ESP8266 Node MCU

Towards the end of the project a decision was made to switch from a standard Arduino uno to the ESP8266 (ESP) which contains onboard Wi-Fi capabilities but while it has 16 General Purpose Input/output (GPIO) pins only eleven can be used for input/output (I/O) functions [8] compared to the fourteen that was available on the uno.

For this reason, there were only enough pins for two sensors after pins were reserved for the l293d. This was seen as an acceptable trade-off for wireless communication functionality.

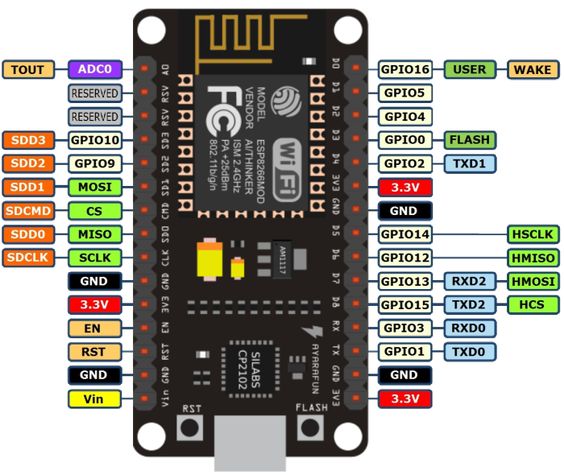


Figure 10 ESP8266 Mode MCU pinout [8]

When setting pin modes on the ESP its pin names have alias names in the Arduino IDE which are the ones that must be used, this is particularly true for the TX and RX pins if they are to be used as I/O pins, shown in Table 4.

Table 4 ESP8266 pin mapping for IDE [8]

|  |  |  |
| --- | --- | --- |
| **Pin name** | **label** | **Alias Name in Arduino IDE** |
| D0 | GPIO 16 | D0 |
| D1 | GPIO 5 | D1 |
| D2 | GPIO 4 | D2 |
| D3 | GPIO 0 | D3 |
| D4 | GPIO 2 | D4 |
| D5 | GPIO 14 | D5 |
| D6 | GPIO 12 | D6 |
| D7 | GPIO 13 | D7 |
| D8 | GPIO 15 | D8 |
| RX | GPIO 3 | D9 |
| TX | GPIO 1 | D10 |

# Technical description and construction

## Hardware

Due to initial testing being done with an Arduino uno it was decided to use a proto shield of type Rev3, shown in figure 11 to place the components onto for robot behaviour refinement. Then a copper stripboard was designed for use with the ESP8266 to add Wi-Fi functionality to the final robot design.

### Initial Arduino uno proto shield

The proto shield was designed to fit onto the uno like a shield. The only drawback to using for the testing was that it contained no natural horizontal or vertical connections like the ones which would be found in a stripboard for example, so any cross connections that would be required had to be done manually.

A circuit board

Description automatically generated

Figure 11 Proto shield Rev3 [9]

A circuit board

Description automatically generatedA circuit board

Description automatically generated

Figure 12 Rev3 placed onto Uno(Left) and soldered Rev3 (Right)

In Figure 12 on the right are all the soldered wire connections which were made long to reach over across the board for direct connections. The compact nature of the board made it a difficult job to solder together the links but the ability to place this onto the uno like a shield (left) made it a worthwhile effort.

### Final ESP8266 stripboard design

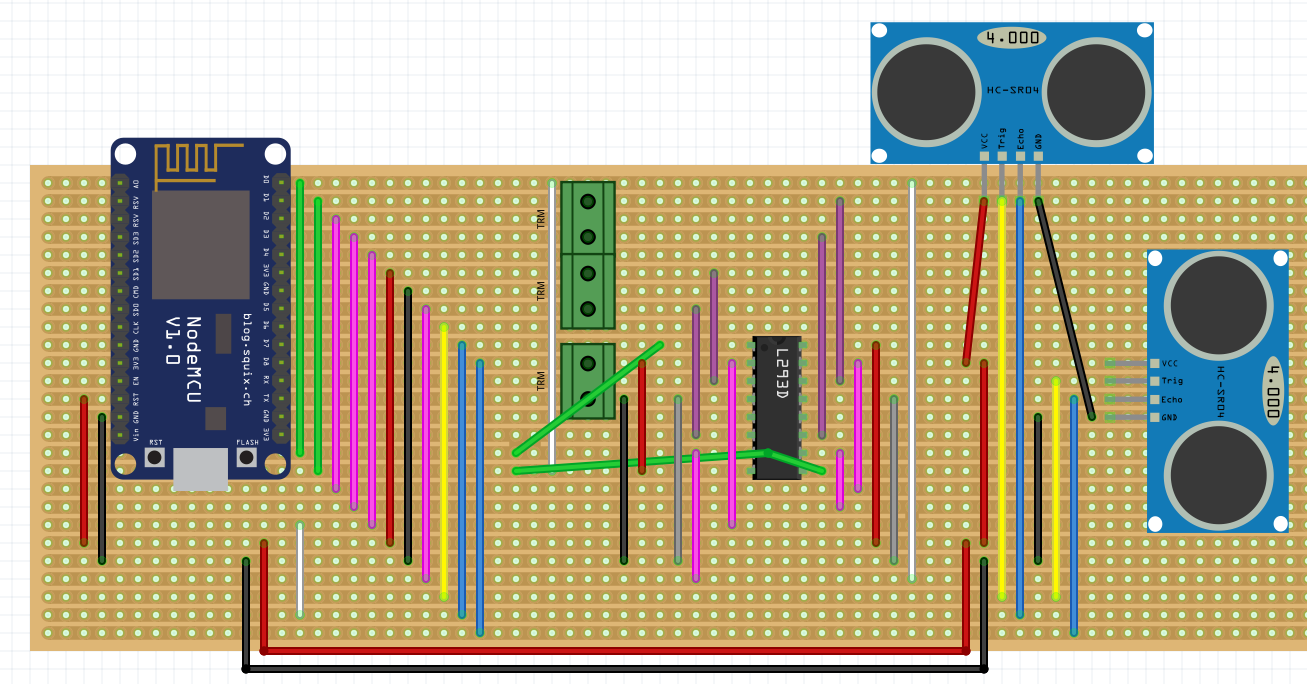


Figure 13 Hardware layout diagram

Figure 13 shows the stripboard layout for the final design of the components onto a breadboard where sub-sections were created by cutting tracks in order to isolate some connections and prevent unwanted cross connections. The areas where tracks were cut are represented by the white lines that split the breadboard into three main sections.

On the far left is the ESP8266 where the natural horizontal paths have been used to interconnect this section with the middle segment. To the left of the l293d are a set of three screw terminals shown by the green boxes that are used to connect the two motors and attach the external battery supply for the wheels. While the top sensor at the edge of the breadboard looks to be connected onto a single line, each pin has been separated from the other by cutting the track between each pin.

Table 5 Colour codes for Stripboard layout

|  |  |
| --- | --- |
| **Colour** | **Purpose** |
| Green | Enable Pins for L293d |
| Pink | Input Pins for L293d |
| Red | Power supply |
| Black, Grey | Ground |
| Yellow | Trig pin for HC-SR04 |
| Blue | Echo pin for HC-SR04 |
| Purple | DC Motor terminals |

The design in figure 13 was used to assign all the connections before implementing it onto the copper stripboard allowing for quick implementation and soldering of the final layout.

A circuit board

Description automatically generated

Figure 14 Top view of final copper stripboard

A picture containing electronics, circuit, clock

Description automatically generated

Figure 15 Bottom view of final copper stripboard

We have placed white boxes around in Figure 15 to highlight where breaks have taken place. Apart from the three sections used to split the copper stripboard other ones are placed between pins on the ESP board and the l293d, this is done because we do not want parallel pins to be connected onto the same pathways.

## Software

For the software a decision was made to create a C++ user defined data type referred to as a class for the HC-SR04 sensor and wheels. Doing so would avoid a long overall program and provide a high-level understanding on what the program is doing just by reading it. Creating the classes would allow multiple objects of the same type to be used for example the code for the HC-SR04 is going to be same for both sensors so we will create an object for the two allowing us to use the same class behaviours again.

### MotorDirection class

The wheels class will be called *MotorDirection*, where the required attributes needed for this class have been identified in Table 6.

Table 6 Attributes for MotorDirection class

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Type and Identifier** |
| DutyCycle | This attribute represents the PWM duty cycle. | int DutyCycle; |
| MotorA | This attribute represents the pin for motor A of the l293d. | int MotorA; |
| MTR\_A\_pin1 | This attribute represents motor A input 1 pin of the l293d. | int MTR\_A\_pin1; |
| MTR\_A\_pin2 | This attribute represents motor A input 2 pin of the l293d. | int MTR\_A\_pin2; |
| MotorB | This attribute represents the pin for motor B of the l293d. | int MotorB; |
| MTR\_B\_pin1 | This attribute represents motor B input 1 pin of the l293d. | int MTR\_B\_pin1; |
| MTR\_B\_pin2 | This attribute represents motor B input 2 pin of the l293d. | int MTR\_B\_pin2; |

The two wheels were now modelled with function behaviours designed around these attributes. From initial wheel testing, information was gathered on what state each l293d pin would be required to be in for generating the desired wheel actions.

Table 7 is a behaviour table showing what attributes we will use, what their identifiers will be and their parameters along with a short description of what the function does.

Table 7 Behaviour table for MotorDirection class

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes**  **Used** | **Function**  **Identifier** | **Description of Function** | **Type, Identifier, Parameters** |
| All attributes used | MotorDirection() | This function will be the default constructor. | MotorDirection(void) |
| All attributes used | SetMotorValues() | This function will be used to set the attribute values. | void SetMotorValues(  int motor1,int pin1,int pin2,  int motor2,int pin3,int pin4 ); |
| All attributes used | Forward() | This function will make both wheels spin forwards. | void Forward(void); |
| All attributes used | Backward() | This function will make both wheels spin reverse. | void Backward(void); |
| MotorA  MTR\_A\_pin2  MTR\_A\_pin1  MotorB | Right() | This function will make the right wheel spin forwards. | void Right(void); |
| MotorA  MotorB  MTR\_B\_pin2  MTR\_B\_pin1 | Left() | This function will make the left wheel spin forwards. | void Left(void); |
| All attributes used | RightR() | This function will make the right wheel spin faster than left one. | void RightR(int val); |
| All attributes used | LeftL() | This function will make the left wheel spin faster than right one. | void LeftL(int val); |
| MotorA  MotorB | StopWheels() | This function will turn the Enable pins LOW, thus turning off the wheels. | void StopWheels(); |
| DutyCycle | SetDutyCycle() | This function will set a value for the duty cycle | void SetDutyCycle(  int AcycleNum); |
| DutyCycle | GetDutyCycle() | This function will return the value in the attribute to main. | int GetDutyCycle(); |

An object oriented approach was taken when creating the class in figure 16 where all relevant wheel behaviours have now been created in the public section of the class, the attributes have been put in the private section because we only want the functions inside the class to have access to the attributes and have the ability to change them.

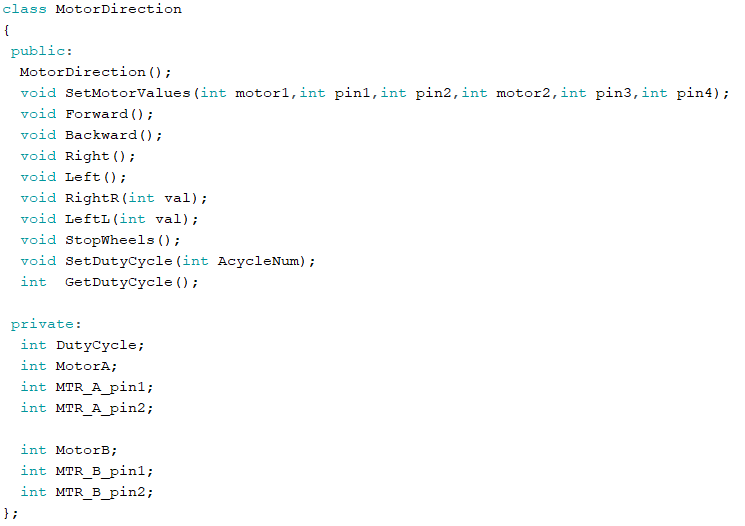


Figure 16 MotorDirection class

The full behaviours for each of the functions shown in Figure 16 is displayed in Appendix A. The default constructor is called when an object of type MotorDirection is created, this initialises the attributes for the pins to zero.

The setMotorValues is called in the setup to run once and assign the pin values to the attributes and set required pins as input or outputs accordingly.

The setDutyCycle is used to pass in the ratio as a percentage value (0-100) of the power that the user wants to be sent to the wheels from the external battery for the wheels.

The rest of the functions are called during program execution to conduct appropriate wheel movements.

### DistanceSensor class

As for the HC-SR04 sensor, the same actions were taken in creating the class *DistanceSensor.* Apart from attributes for the pin values two more called *duration signal* that is to be used for storing the data on how long it took the transmitted waves to be received and one called *distance\_cm*, for converting the time measurement into a distance value using the relationship between speed, distance and time.

The attributes needed for the DistanceSensor class have been identified in Table 8.

Table 8 Attributes for DistanceSensor class

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Type and Identifier** |
| Trig\_pin | This attribute represents the trigger pin. | int Trig\_pin; |
| Echo\_pin | This attribute represents the Echo pin. | int Echo\_pin; |
| Duration\_Signal | This attribute represents the duration of the received Echo signal. | float Duration\_Signal; |
| Distance\_cm | This attribute represents the calculated distance measurement in cm. | float Distance\_cm; |

Table 9 Behaviour table for DistanceSensor class

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes**  **Used** | **Function**  **Identifier** | **Description of Function** | **Type, Identifier,**  **Parameters** |
| Trig\_pin  Echo\_pin | DistanceSensor() | This function will be the default constructor. | DistanceSensor(); |
| Trig\_pin  Echo\_pin | SetUltrasonicPins() | This function will be the regular constructor. | void SetUltrasonicPins(  int aTrig, int aEcho); |
| All attributes used | FindDistance() | This function will set the trigger and receive the Echo. | float FindDistance(); |
| Duration\_Signal | Calculate\_cm() | This function converts the Echo into a distance value in cm units. | float Calculate\_cm(); |

The Distance sensor class is relatively straightforward to implement with few behaviours required because the sensor does only two jobs, find and calculate distance values.

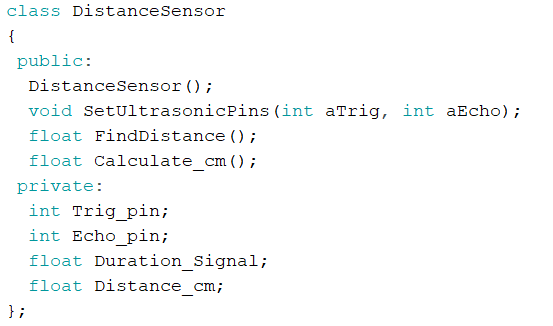


Figure 17 DistanceSensor class

The behaviour of each function is shown in full in Appendix B. The default constructor is called when an object of type DistanceSensor is created, this initialises the attributes for the pins to zero.

The *setUltrasonicPins* is called in the setup to run once and assign the pin values to the attributes and set the required pins as input or output accordingly.

When the *FindDisatnce* is called the Trigger pin sends out the signal which is reflected back from an object at the Echo pin, saved and its return call will execute the *calculate\_cm* and return the centimetres value from that calculation.

# Test procedures and results

Initial testing was done in Tinkercad where a project was setup for testing the code and getting a better understanding of how the l293d should be connected to other components. Then hardware testing was done by building on a breadboard then moved onto a copper stripboard for the finished design.

## Test 1 Single wheel with L293D simulation

Objective: The become familiar with l293d pinout and test this component with a wheel.

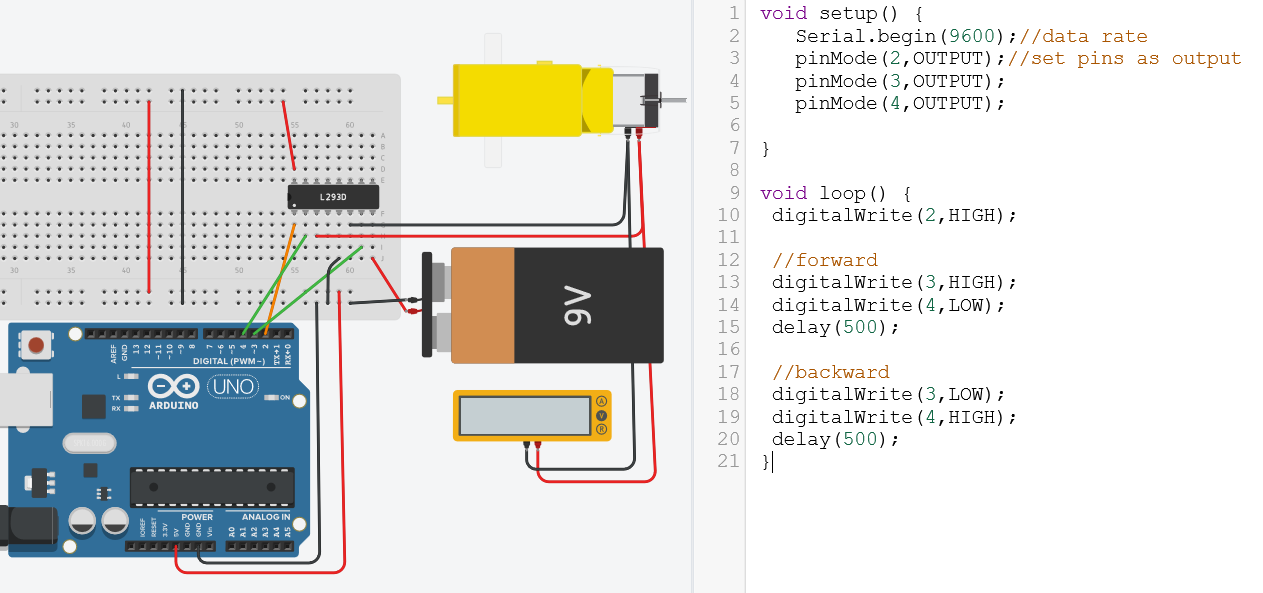


Figure 18 Testing l293d with one wheel

Set-up: To start off a single wheel is connected to the l293d for testing with the Arduino in Tinkercad software to simulate the circuit in figure 18.

Method: To directly turn pins HIGH/LOW in the loop to see forwards and backwards wheel behaviour.

Result: A better understand of how the pins of the l293d need to be turned HIGH or LOW to produce the desired behaviour was achieved.

## Test 2 Class behaviour test with L293D

Objective: To connect two wheels to the l293d and use the newly created MotorDirection class.

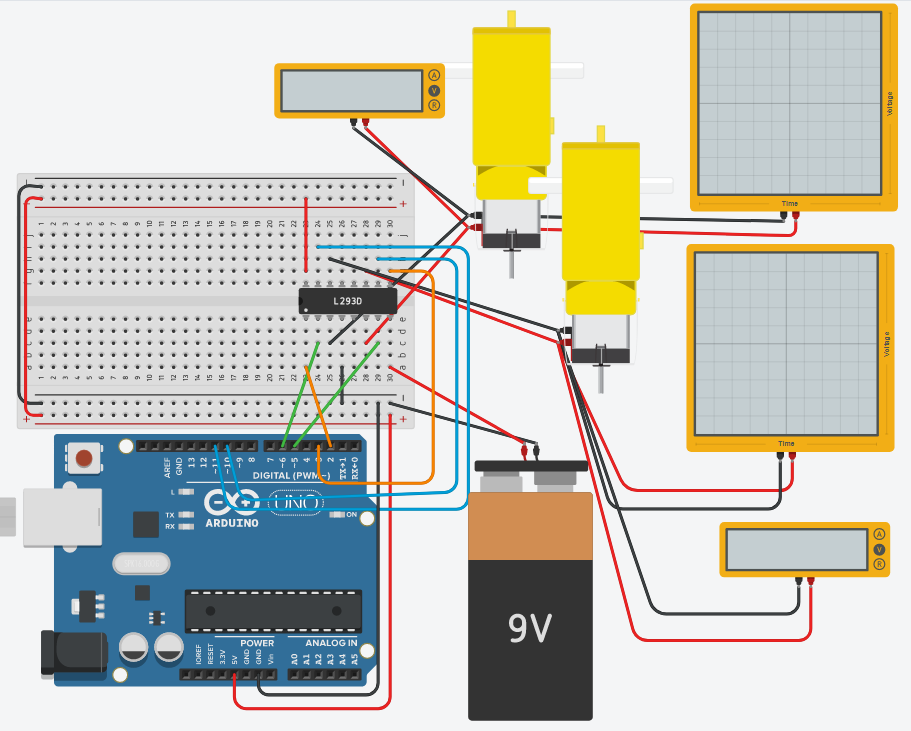


Figure 19 Testing l293d with two wheels

Set-up: Test 2 setup was expanded in Figure 19 with another wheel and virtual oscilloscopes were connected to view the PWM signals, this setup was used to validate the class behaviours.

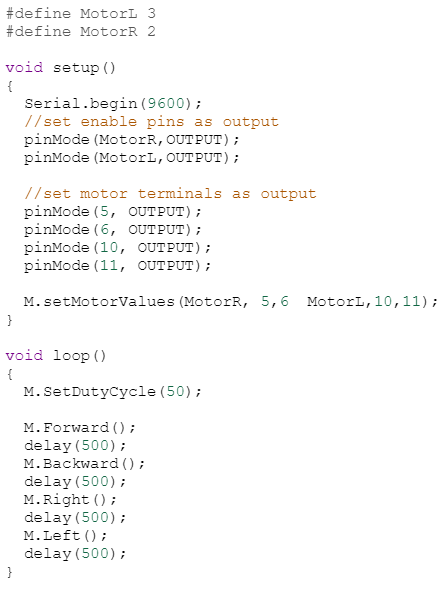


Figure 20 class test with l293d

In Figure 20 the function behaviours were tested by calling the *SetMotorValues* to save the assigned pin values to the attributes created in the class. Then wheel behaviours are called in the order shown to verify each function accomplishes its intended purpose.

Result: The class functions have made it much easier to understand the commands that are executed than before because they provide a high level view of the executed behaviours, they can now be called to execute the corresponding pins commands for each of the wheel behaviours shown during the loop in Figure 20.

## Test 3 Hardware validation of L293D

Objective: Verify the results seen in test 2 with hardware implementation.

A close up of a device

Description automatically generated

Figure 21 Hardware verification of l293d

Set-up: The circuit from test 2 is built in hardware to validate the results seen in the simulation, this is why Figures 21 and 19 looks very similar.

Result: Figure 20 code was used again for this test and all the functions preformed as expected the wheels were spinning forwards, backwards, right and left.

## Test 4 HC-SR04 sensors simulation

Objective: To connect HC-SR04 sensors to Arduino.

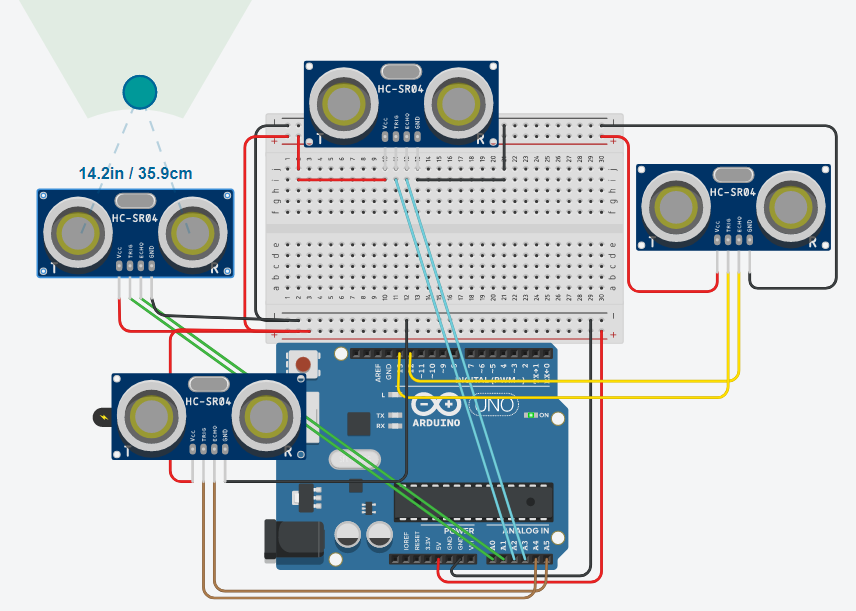


Figure 22 Simlation of HC-SR04 sesnors

Set-up: 4x HC-SR04 sensors are connected to the Arduino, mostly in analogue pins because the digital pins are assigned to the l293d. Their placement in Figure 22 corresponds to their real positions of left, right, back and front.

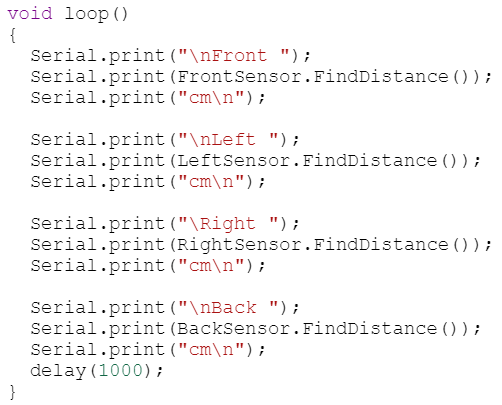


Figure 23 HC-SR04 Sensor test code

For this test we simply want to display each distance value of the virtual objects to the serial monitor for which the code is shown in Figure 23.

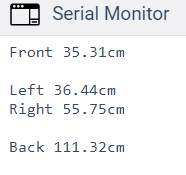


Figure 24 HC-SR04 Sensor result

Result: The Distance sensor class was performing as expected and produced the correct distance results of the virtual objects that were placed in front of each sensor. By comparing the front sensor distance value in Figure 24 (35.31cm) and the virtual object distance for the front sensor in Figure 22 (35.9cm) we see the result shown is very close to the distance we placed the virtual object.

## Test 5 Hardware verification of sensors with L293D

Objective: To test the movement with sensor feedback.

A picture containing indoor, wall, floor

Description automatically generatedA picture containing indoor, wall, floor, sitting

Description automatically generated

Figure 25 Top and side view (Left & Middle) of setup and Testing (Right)

Set-up: The three sensors have been soldered onto a proto Rev3 board with three screw terminals for the motor connections and battery. The breadboard pins has been aligned with the Arduino uno to fit on like a shield.

Result: The robot was able to move in a parallel path along the wall with code used to make adjustments due to its tendency to divert from a straight path.

## Test 6 ESP8266 webserver

Objective: To setup a webserver on the ESP for entering in X & Y coordinates.

Set-up: Use the ESP 8266 board with its Wi-Fi libraries (ESP8266WiFi.h, WiFiClient.h, ESP8266WebServer.h).

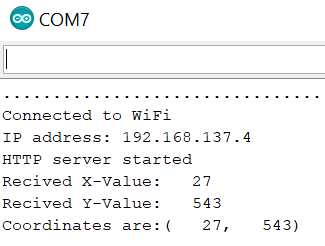
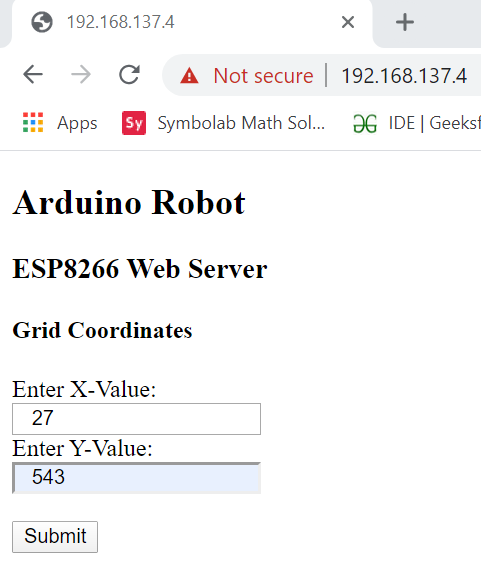


Figure 26 ESP webserver(Left) and Serial monitor result (Right)

Code for the webserver was modified from *circuits4you.com [10],* it is shown in full in Appendix C.

In Figure 26 on the right when a Wi-Fi connection is established there is an output showing the Internet Protocol (IP) address to use for connecting to the webserver, this is placed into the browser shown on the left which displays the HTML webpage that was created. Here X-values and Y-values of 27 and 543 have been entered and are shown correctly back to the serial monitor.

Result: The webserver was set up and worked as expected by receiving the X and Y values and saving them to local attributes on the ESP, ready for the program to being making decisions.

## Test 7 ESP Final testing of all components

Objective: To test Wi-Fi and robot navigability using coordinates.

**A circuit board

Description automatically generated**

Figure 27 Top view of finished robot

**A picture containing yellow, truck, parked, street

Description automatically generated**

Figure 28 Side view of finished robot

Set-up: The final stripboard design has all the components such as battery for both ESP and for the motors with the sensors connected (Figure 27) for testing if all aspects of the robot work as intended.

Result: After program upload (Appendix D), the wheels were not responding to the transmitted coordinates after which debugging time was spent trying to find what the hardware/software issue could be. The ESP was found to be unresponsive and broken. This resulted in having to use the setup in test 5 for the final adjustments to the program shown in Appendix E, using the serial monitor to receive the coordinates instead.

## Test 8 Modify code to work with serial monitor input

Due to using the setup from test 5 for the rest of the project time during the Covid-19 lockdown since the ESP had broken, input from serial monitor will be a substitute for the wireless input of coordinates.

Objective: convert the program that was created for the ESP to run on the Arduino uno with serial input.

Information on how to read character strings from the serial monitor was found at [*www.c-sharpcorner.com*](http://www.c-sharpcorner.com) *[11].* This was used to create a function called *GetValFromMonitor()* which will read the numbers as characters then convert the characters into integer values using the *toInt()* command.

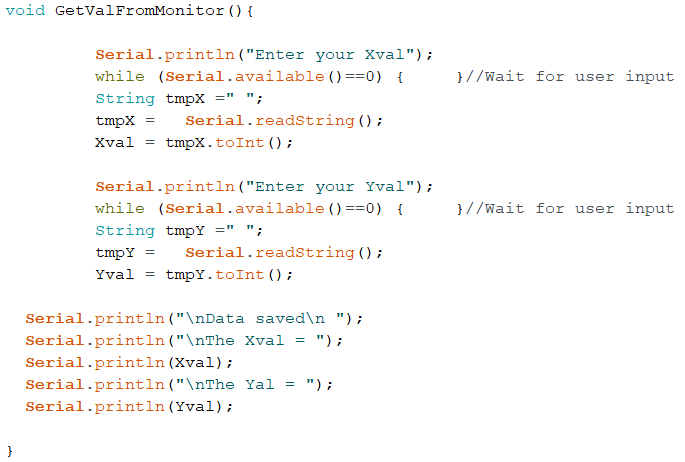


Figure 29 Function for input from serial monitor

In Figure 29 local variables of type string are created called *tmpX* and *tmpY* to store the character input then when the *toInt* command is called the converted integer values are saved to variables (Xval & Yval) that were created in the main program.

Result: The function was able to read the character input and convert into integer values for use in the rest of the program.

# Conclusion

If given the opportunity to improve the project there would be an increase of the ultrasonic sensors to six, one front and back one top left & right and one bottom left & right. The two sensors at each side would allow for the ability to accurately position the robots body parallel to shelfs by ensuring both sensors are equal distance from the wall. An Arduino mega would be suitable for the increased number of sensors and as for the code both classes that were created can be reused by creating multiple objects of the same type. The wheel array could be improved by replacing them with stepper motors which can rotate the wheels at accurate angles, allowing for precise positioning of the robot along a shelf.

As for the Wi-Fi communication the webserver that has been set up is working as intended but some feedback about the package could be sent back to the website to better inform the user of package details and possibly the robots status on fetching the package.

Alternative solutions could entail local GPS navigation within a confined area and use four stepper motor wheels for accurate positioning and a robotic arm for picking up the package with cameras for object recognition/detection. The circuit for this setup could be put onto a custom printed circuit board.

As the project progressed, we had to remove two sensors due to hardware constraints but managed to complete all the important aspects of the project and got a demonstration of the proposed solution working as intended. Skills learned from the three-year course was used to assist in project areas such as coding to create user defined data types, hardware and software debugging to find and fix any technical issues, project planning for breaking down the project into sub tasks which are more manageable. In addition, the skill required for understanding and designing a circuit was used in the design, building and integration of multiple components.

# References

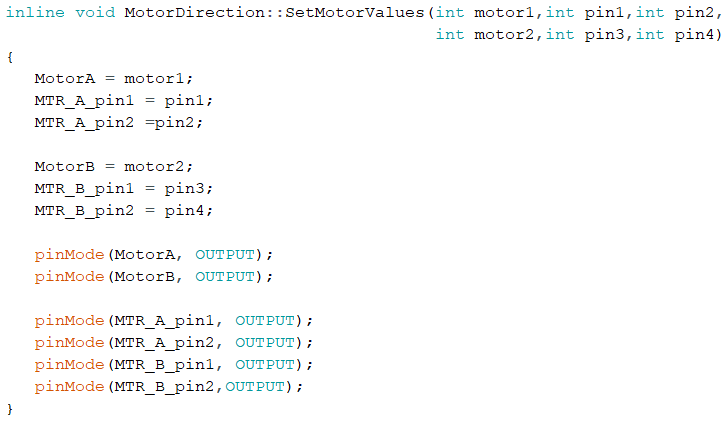
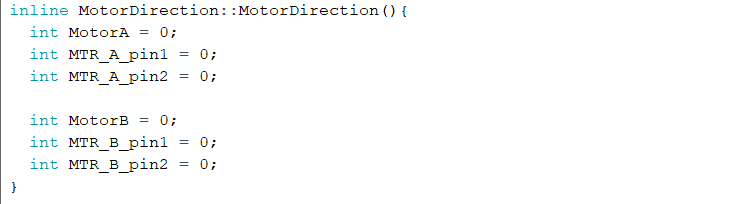
1. Y. Dahl, "What is an H-Bridge? - Build Electronic Circuits", *Build Electronic Circuits*, 2018. [Online]. Available: https://www.build-electronic-circuits.com/h-bridge/. [Accessed: 03- Feb- 2020].
2. "L293D Motor Driver IC with Arduino | H-bridge IC", Electronics Hobbyists. [Online]. Available: https://electronicshobbyists.com/l293d-motor-driver-ic-arduino-h-bridge-ic/. [Accessed: 01- Feb- 2020].
3. "L293D Motor Driver IC introduction, pinouts and how to use", Microcontrollers Lab. [Online]. Available: https://microcontrollerslab.com/l293d-motor-driver-ic-introduction-pinouts-example/. [Accessed: 02- Feb- 2020].
4. T. Hirzel, "Arduino - PWM", Arduino.cc. [Online]. Available: https://www.arduino.cc/en/tutorial/PWM. [Accessed: 07- Feb- 2020].
5. "Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino | Random Nerd Tutorials", Random Nerd Tutorials. [Online]. Available: https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/. [Accessed: 14- Feb- 2020].
6. S. Nasir, "Ultrasonic Sensor Arduino Interfacing - The Engineering Projects", *The Engineering Projects*, 2017. [Online]. Available: https://www.theengineeringprojects.com/2017/08/ultrasonic-sensor-arduino-interfacing.html/. [Accessed: 21- Feb- 2020].
7. "Arduino Uno Tutorial [Pinout] - DIYI0T", *DIYI0T*. [Online]. Available: https://diyi0t.com/arduino-uno-tutorial/. [Accessed: 10- Feb- 2020].
8. "ESP8266 NodeMcu Pinout - ESP8266 Shop", *ESP8266 Shop*. [Online]. Available: https://esp8266-shop.com/esp8266-guide/esp8266-nodemcu-pinout/. [Accessed: 03- Mar- 2020].
9. J. Nussey, "An Overview of Arduino Shields - dummies", *dummies*. [Online]. Available: https://www.dummies.com/computers/arduino/an-overview-of-arduino-shields/. [Accessed: 18- Feb- 2020].

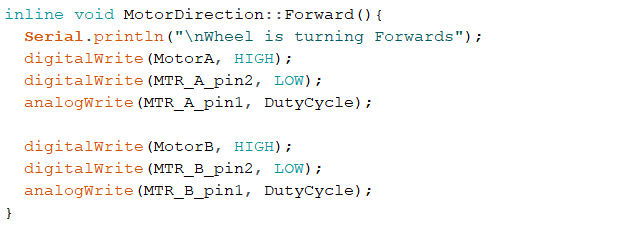
[10].M. Thakur, "ESP8266 Node MCU Handling HTML web forms data | Circuits4you.com", ……*Circuits4you.com*, 2019. [Online]. Available: https://circuits4you.com/2019/03/20/esp8266-.......receive-post-get-request-data-from-website/. [Accessed: 07- Mar- 2020].

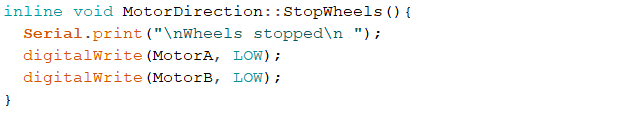
[11].D. Dash, "Reading Input From Serial Monitor In Arduino", *C-sharpcorner.com*, 2016. [Online]. ///////Available: https://www.c-sharpcorner.com/article/reading-input-from-serial-monitor-in-|||||||arduino/. [Accessed: 25- Mar- 2020].

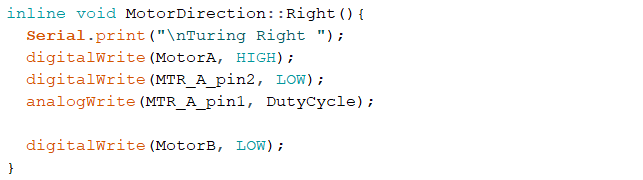
# Appendices

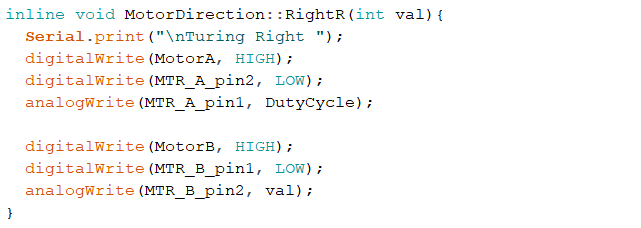
## Appendix A – Behaviours for MotorDirection Class

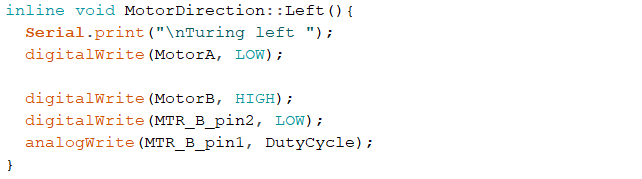


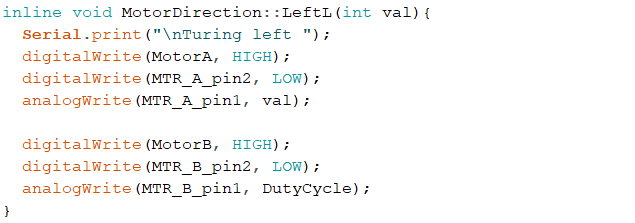


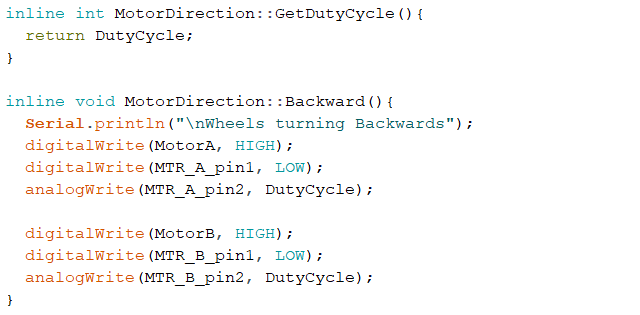
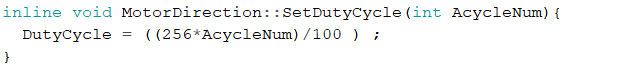




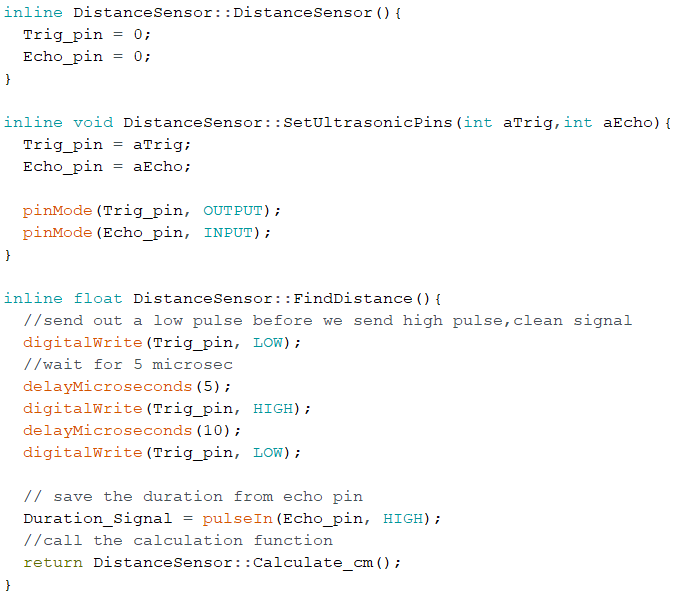


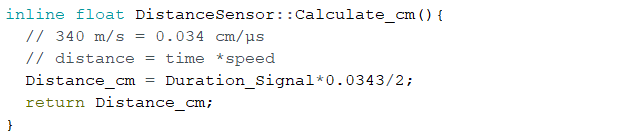






## Appendix B – Behaviours for DistanceSesnor Class

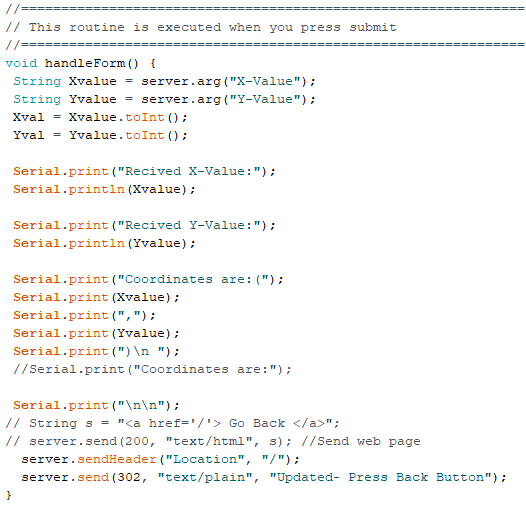


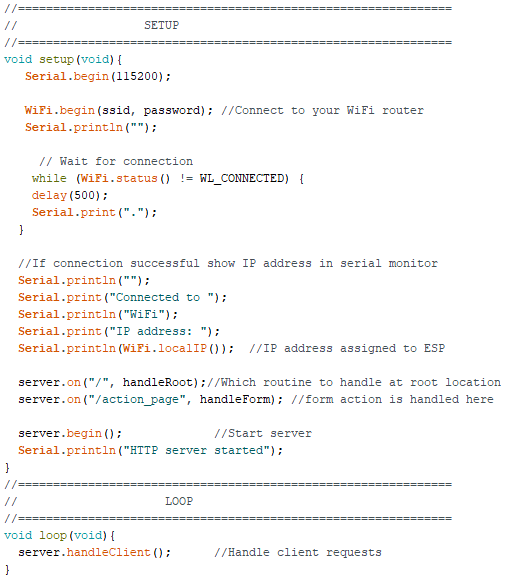


## Appendix C – ESP8266 Webserver code

Code Modified from *circuits4you.com* [10].

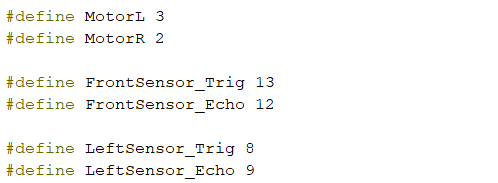


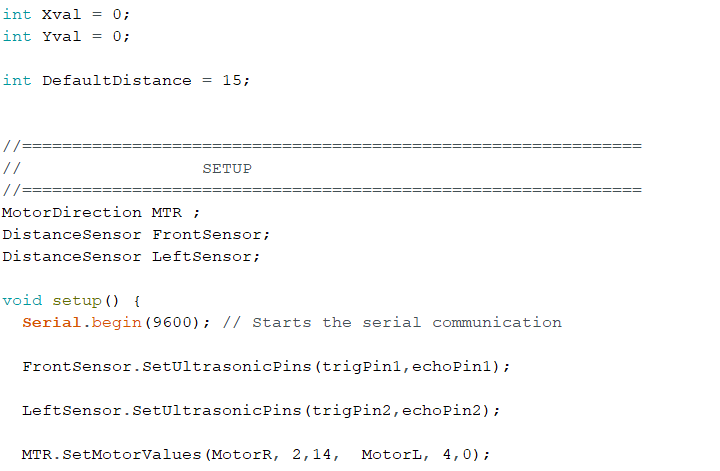




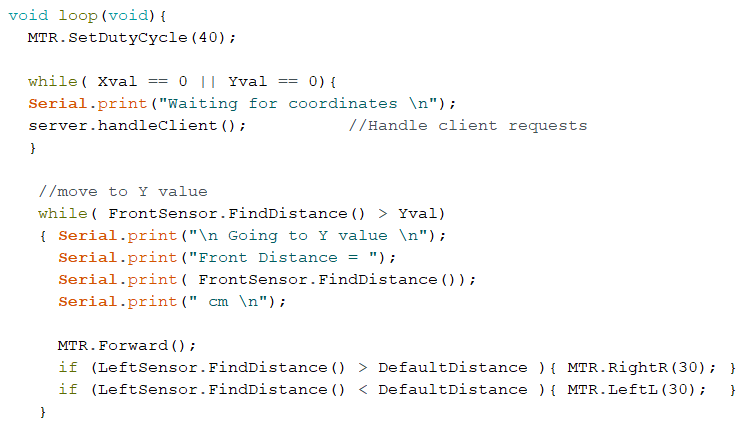
## Appendix D – Program code Arduino ESP8266 version

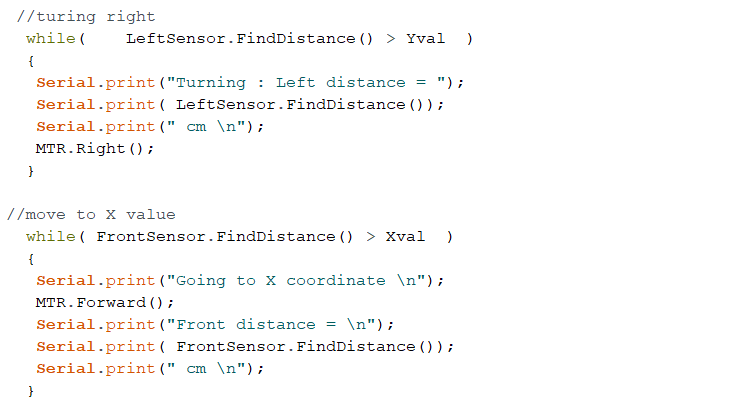
Code from Appendix A,B and C is included here but not shown to avoid repetition.

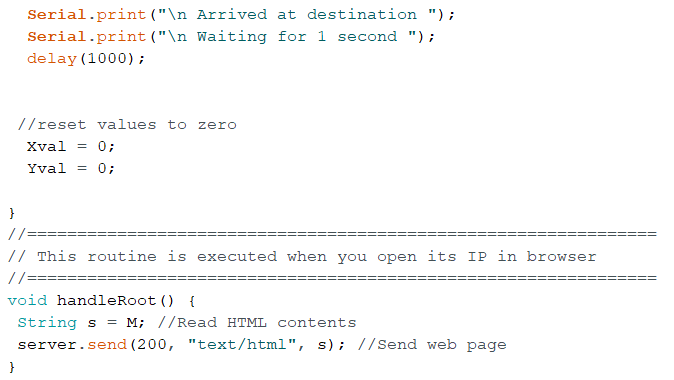


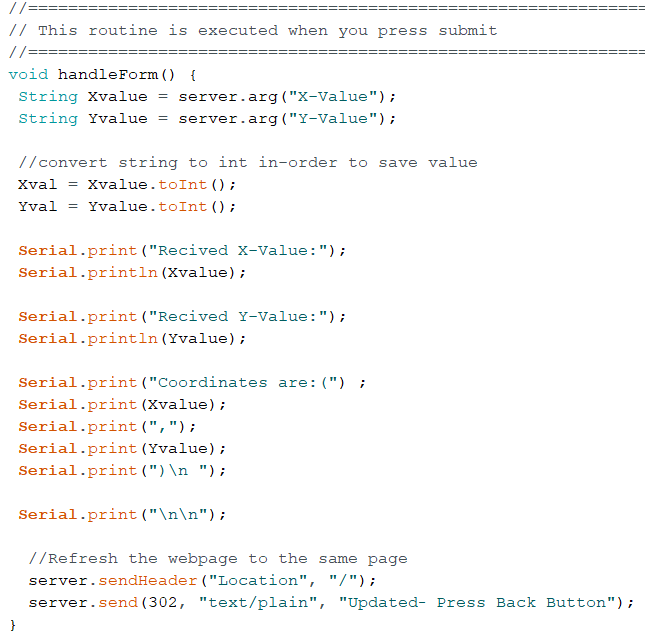












## Appendix E – Program code Arduino uno version

Code from Appendix A and B is included here but not shown to avoid repetition.

